

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Система автоматизації процесу обробки даних для
ведення оперативного обліку та бухгалтерської
звітності»**

Виконав студент групи ЗПЗк-21-1
ФІЛІМОНЧУК Максим Іванович

Керівник роботи:
ТОЛСТОЙ Ігор Анатолійович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ФІЛІМОНЧУК Максим Іванович**

Керівник роботи: **ТОЛСТОЙ Ігор Анатолійович**

Тема роботи: **«Система автоматизації процесу обробки даних для ведення оперативного обліку та бухгалтерської звітності»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р. №75с**

Вихідні дані для роботи: процеси для забезпечення ведення обліку з використанням комп'ютерних технологій, системи автоматизації оперативного та бухгалтерського обліку.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Толстой І.А.	20.03.2024р.	20.03.2024р.
2	Толстой І.А.	18.05.2024р.	18.05.2024р.
3	Толстой І.А.	16.06.2024р.	16.06.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота складається з прикладного програмного додатку для ведення оперативного та бухгалтерського обліку розробленого для операційної системи Windows та пояснювальної записки.

Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 41 найменувань (90 стор.), 8 додатків (94 стор.), 37 таблиць та 59 ілюстрацій по тексту. Повний обсяг роботи становить 110 сторінок, з них 92 сторінки основного тексту.

Мета роботи є створення системи для автоматизації процесу обробки даних для ведення оперативного обліку та бухгалтерської звітності засобами .NET Core за допомогою мови C# із використанням бази даних SQLite. Додаток надасть можливість користувачам мати актуальні дані по їхніх залишках товарів, боргів та надасть інформацію про продажі.

Додаток має простий інтерфейс що облегшує взаємодію користувачам та дозволяє швидко організувати ведення обліку самостійно.

КЛЮЧОВІ СЛОВА: БАЗА ДАНИХ, БУХГАЛТЕРІЯ, ОБЛІК, ПРОДАЖІ, ЗВІТНІСТЬ, ФРЕЙМВОРК, ORM, .NET, ENTITY FRAMEWORK CORE, ВІДОБРАЖЕННЯ.

					ІПЗ.КР.Б – 121 – 24 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.	Філімончук М.І.				Система автоматизації процесу обробки даних для ведення оперативного обліку та бухгалтерської звітності Пояснювальна записка	Літ.	Арк.	Аркуші
Керівник	Толстой І.А.						3	
Н. контр.	Тичина О.П.					Житомирська політехніка, група ЗІПЗк-21-1		
Зав. каф.	Вакалюк Т.А.							

ABSTRACT

The final qualification work consists of an application software application for operational and accounting developed for the Windows operating system and an explanatory note.

The graduation thesis consists of an introduction, 3 chapters, conclusions, a list of references of 41 titles (90 pages), 8 appendices (94 pages), 37 tables and 59 illustrations in the text. The total volume of the work is 110 pages, including 92 pages of the main text.

The purpose of the work is to create a system for automating the data processing process for operational accounting and financial reporting by means of .NET Core using the C# language with the use of the SQLite database. The application will allow users to have up-to-date data on their balances of goods, debts and provide information on sales.

The application has a simple interface that facilitates user interaction and allows you to quickly organize accounting on your own.

KEY WORDS: DATABASE, ACCOUNTING, SALES, REPORTING, FRAMEWORK, ORM, .NET, ENTITY FRAMEWORK CORE, REFLECTION.

					ИПЗ.КР.Б – 121 – 24 – ИПЗ	Арк.
						4

3MICT

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. Аналіз напрямків використання інформаційних технологій для системи автоматизації обробки даних для оперативного обліку та бухгалтерської звітності.....	9
1.1 Постановка задачі	9
1.2 Аналіз аналогічних програмних продуктів	12
1.3 Вибір архітектури системи автоматизації процесу обробки даних для ведення оперативного обліку та бухгалтерської звітності	15
1.4 Інструментальні засоби реалізації програмного продукту та вимоги до апаратних засобів.	20
Висновки до першого розділу	21
РОЗДІЛ 2. Проектування системи автоматизації обробки даних для оперативного обліку та бухгалтерської звітності.....	22
2.1 Визначення варіантів використання системи	22
2.2 Розробка бази даних системи	39
2.3 Проектування та реалізація алгоритмів роботи системи	48
2.4 Реалізація функціоналу системи автоматизації операційного обліку та бухгалтерської звітності	58
Висновки до другого розділу	67
РОЗДІЛ 3. Інтерфейс та порядок роботи з системою автоматизації обробки даних для ведення оперативного та бухгалтерського обліку	68
3.1 Порядок встановлення та налаштування параметрів системи.....	68
3.2 Структура інтерфейсу системи автоматизації обробки даних для ведення оперативного та бухгалтерського обліку	68
3.3 Тестування роботи додатку	84
Висновки до третього розділу	88
ВИСНОВКИ	89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	90
ДОДАТКИ	94

					ИЗ.КР.Б – 121 – 24 – ИЗ	Арк.
						5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

.NET – платформа від Microsoft для створення програмного забезпечення.

API – Application Programming Interface (Прикладний програмний інтерфейс).

BL – шар бізнес логіки що реалізується додатком.

CRM – це система управління відносинами з клієнтами. Фактично це програмне забезпечення для відділу продажу.

CRUD – 4 основні функції управління даними «створення, читання, оновлення і видалення».

DDD – підхід до моделювання програмного забезпечення.

DI – впровадження залежності.

DIP – принцип інверсії залежностей.

ER-діаграма – це різновид блок-схеми, де показано, як різні «сутності» пов'язані між собою усередині системи.

EF – фреймворк що використовує об'єктно-реляційне відображення (ORM) для роботи з базами даних, використовуючи об'єктно-орієнтоване програмування.

Linq – компонент Microsoft .NET Framework, який додає нативні можливості виконання запитів даних до мов, що входять у .NET.

ORM – технологія яка дозволяє пов'язування об'єктів класів в кодї із записами в БД.

SDK – набір із засобів розробки, утиліт і документації, який дає програмістам змогу створювати прикладні програми за визначеною технологією або для певної платформи.

SQL – це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також керування базами даних.

WPF – система для побудови клієнтських інтерфейсів.

БД – база даних.

ПК – персональний комп'ютер.

ВСТУП

Актуальність теми: облік оточує нас що дня, будь то підрахунок сімейного бюджету чи кількість проданого товару. Ведення облік надає можливість відслідковувати та контролювати окремі господарські операції та процеси з метою керівництва ними та відображати особисту або підприємницьку діяльність. Для цього потрібен оперативний та бухгалтерський облік. Системи що допомагають реалізувати ведення обліку надають велику кількість функцій що може стати складним для розуміння звичайного користувача, особливо якщо система шукається для особистого обліку або ведення невеликого бізнесу. Через цю складність користувач не почне користуватися системою. Для подібної потрібний простий інструмент що надасть їм спрощену модель ведення оперативного та бухгалтерського обліку.

Оперативний облік дає можливість відслідковувати так контролювати окремі операції з процесами на найважливіших ділянках господарської діяльності з метою керівництва ними. Так, за допомогою оперативного обліку здійснюється контроль за виконанням договорів поставки товарно-матеріальних цінностей, використанням робочого часу, обладнання і техніки, випуском та реалізацією продукції та іншого [1].

Бухгалтерський облік відображає діяльність підприємств, організацій, установ. Він охоплює всі засоби господарства і джерела їх формування, всі господарські процеси і результати діяльності [1].

Практична цінність: використання розробленого додатку надасть користувачам можливість почати вести товарний та грошовий облік безкоштовно та потреби у довготривалому навчанні. Для цього користувачеві буде надано простий інтерфейс, функціонал створення замовлень покупців та постачальників, робити прибуткові або видаткові товарні накладні, фіксувати надходження та витрати грошових коштів з рахунків або кас. Для аналізу надаються звіти через які можна буде дізнатися про залишки товарів, залишки грошових коштів, продажі, закупівлі, борги клієнтів та постачальників.

Мета: даної роботи полягає в розробці прикладного додатку для ведення оперативного та бухгалтерського обліку засобами .NET Core мови програмування C# та з використанням бази даних SQLite. Додаток повинен забезпечити легке ведення оперативного обліку та бухгалтерської звітності для малого бізнесу.

Встановлена мета обумовлює наступні завдання:

- визначення архітектури та структури системи;
- проектування бази даних;
- визначення технологій для реалізації задачі;
- визначення функціональних вимог предметної області дослідження;
- створення графічного інтерфейсу;
- аналіз існуючих аналогів;
- розробка інформаційного та математичного забезпечення системи.

Об'єктом дослідження: є процеси ведення обліку з використанням комп'ютерних технологій.

Предметом дослідження: є реалізація процесів автоматизації обробки даних для операційного обліку та бухгалтерської звітності за допомогою .NET Core.

Розроблений прикладне програмне забезпечення може бути використаний малим бізнесом таким як невеликі магазини продуктів або інших товарів, магазини в соціальних мережах та для ведення домашнього обліку без потреби розгортання додаткового програмного забезпечення окрім встановлення .NET.

РОЗДІЛ 1. Аналіз напрямків використання інформаційних технологій для системи автоматизації обробки даних для оперативного обліку та бухгалтерської звітності

1.1 Постановка задачі

Кваліфікаційна робота полягає у створенні системи автоматизації обробки даних для оперативного обліку та бухгалтерської звітності. Головною метою завдання є створення зручної та простої системи, яка допоможе клієнтам здійснювати ведення обліку матеріальних та фінансових ресурсів, відображати їхню взаємодію зі своїми клієнтами в документованому вигляді та отримання облікової та бухгалтерської звітності.

Автоматизація бізнес-процесів – це впровадження програмної системи, яка виконує типові процедури по сучасним алгоритмам, таким як складання і виписка документів, контроль виконання облікових, складських операцій. Завдяки подібним нововведенням рівень роботи підприємства якісно підвищується [2].

Автоматизація процесів в переважній більшості випадків сприяє підвищенню якості продукції та послуг, слід зазначити наступні позитивні аспекти її введення [2]:

- спрощується введення та пошук інформації;
- інформація обробляється і передається набагато швидше;
- в єдиному інформаційному просторі обслуговуються окремі підрозділи або все підприємство;
- запрограмовані звіти і документи складаються автоматично;
- зростає зручність оперування базами даних, а робота стає більш точною.

З огляду на перераховані переваги та загальне спрощення ведення обліку і взаємодії із клієнтами було визначено перелік задач які потрібно вирішити задля досягнення даного результату:

- авторизація (логін та пароль);
- створення довідникової інформації;
- редагування довідникової інформації;
- видалення довідникової інформації;
- створення документів;
- редагування документів;
- видалення документів;
- проведення документів;
- відміна проведення документів;
- заповнення на основі документів;
- формування звітів;
- створення записів у реєстрах відомостей (ціни, штрих коди, зв'язки контрагентів і контактів);
- видалення записів у реєстрах відомостей;
- створення записів у реєстрах накопичення (при проведенні документів);
- видалення записів у реєстрах накопичення (при відміні проведення документів).

Для реалізації описаних задач в системі необхідні наступні об'єкти:

1. Довідники:

- 1.1. Номенклатура.
- 1.2. Одиниці виміру.
- 1.3. Склади.
- 1.4. Організації.
- 1.5. Види цін.
- 1.6. Контрагенти.
- 1.7. Валюти.
- 1.8. Банки.
- 1.9. Банківські рахунки.

1.10. Контакти контрагентів.

1.11. Користувачі.

1.12. Каса.

2. Документи:

2.1. Замовлення покупців.

2.2. Замовлення постачальників.

2.3. Прибуткова накладна.

2.4. Видаткова накладка.

2.5. Прибутковий касовий ордер.

2.6. Видатковий касовий ордер.

2.7. Списання з рахунку.

2.8. Надходження на рахунок.

3. Звіти:

3.1. Залишки товарів на складах.

3.2. Взаєморозрахунки з контрагентами.

3.3. Продажі.

4. Регістри відомостей:

4.1. Ціни номенклатури.

4.2. Курси валют.

4.3. Штрих коди номенклатури.

4.4. Зв'язок контрагент/контакт.

5. Регістри накопичення:

5.1. Залишки товарів на складах.

5.2. Взаєморозрахунки з контрагентами.

5.3. Продажі.

Довідники – об'єкти які слугують для зберігання статичної інформації довідкового характеру. Довідники забезпечують зручне зберігання і доступ до даних, які є необхідними для автоматизації облікових процесів.

Документи – об'єкт що зберігає в собі інформацію про виконані операції що були здійснені підприємством або організацією; Проведений документ є

підставою для здійснення облікових проводок і служить доказом факту здійснення певної операції та її характеристик.

Звіти – об’єкт що відображає інформацію про діяльність підприємства за певний період часу. Звіти використовують для аналізу, контролю та прийняття рішень по управлінню ресурсами та фінансами.

Регістр відомостей – об’єкт що призначений для зберігання інформації в розрізі визначених вимірів що разом складають унікальний ключ даних по якому можна буде відразу отримати пов’язані дані. Регістри відомостей бувають двох типів періодичні, до визначених вимірів додається ще дата що дозволяє отримувати історію зміни даних по ключ за період, та не періодичні в ньому дані зберігаються лише в розрізі визначених вимірів.

Регістр накопичення – об’єкт призначений для зберігання та узагальнення даних про виконані операції та їх накопичення протягом певного періоду часу. Вони використовуються для ведення обліку та аналізу діяльності підприємства що забезпечує контроль та управління різними аспектами діяльності підприємства.

1.2 Аналіз аналогічних програмних продуктів

Для здійснення порівняльного аналізу було обрано наступні системи: **ODOO** та **LP-CRM**. Аналіз даних додатків дозволить отримати розуміння практик та підходів які були використані в них що в подальшому може позитивно вплинути на розроблюване програмне забезпечення [3, 4].

Odoo – це програмне забезпечення з відкритим кодом для інтеграції таких процесів, як бухгалтерський облік, фінанси, людські ресурси, управління інформацією, інвентаризація, закупівлі та інші програми. Це найпопулярніше програмне забезпечення з відкритим кодом і йому довіряють у різних галузях [5].

Однією з ключових переваг Odoo є її модульна архітектура та велика наявність уже готових модулів, яка дозволяє користувачам обирати лише ту

мінусом відсутності повноцінного додатку який можна було б встановити на власному комп'ютері в вести облік без доступу до мережі.

Таким чином, LP-CRM є простим і доступним рішенням для малих та середніх підприємств, які шукають ефективний інструмент для управління взаємовідносинами з клієнтами, який не складно налаштувати і використовувати.

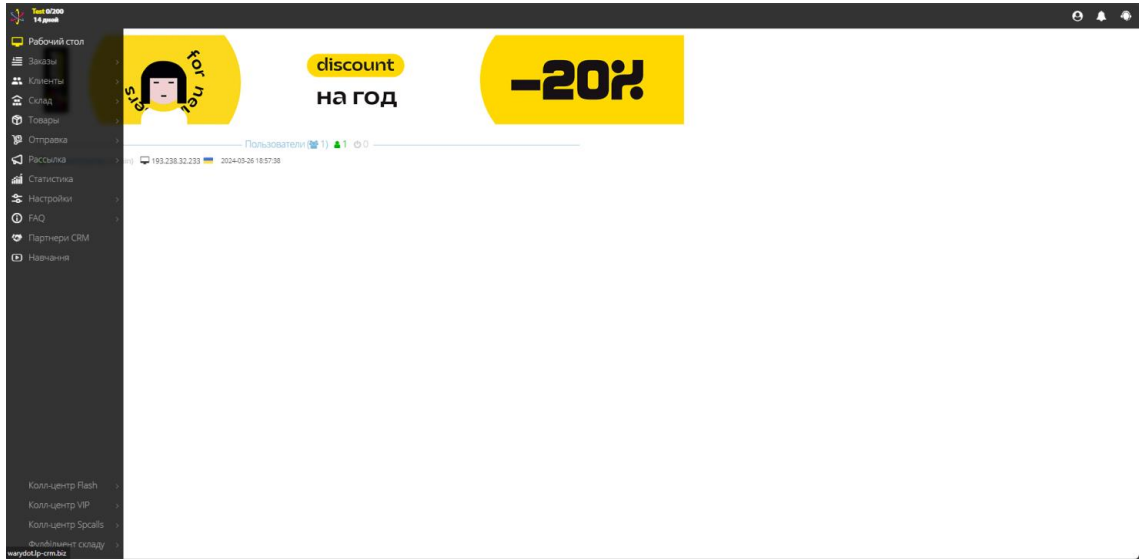


Рис. 1.2 – головний екран в LP-CRM

Наведені вище аналоги являються найбільш розповсюдженими рішеннями для ведення обліку на даний момент. Тому давайте проведемо порівняння характеристик Odoo, LP-CRM та розроблюваного програмного забезпечення в межах кваліфікаційної роботи. Для кращого візуального порівняння сильних та слабких сторін кожного продукту було створено таблицю порівняння (табл. 1.1).

Таблица 1.1

Таблица порівняння облікових систем

Характеристика	Odoo	LP-CRM	Розроблюваний додаток
Ціна	-	-	+
Кросплатформеність	+	+	-
Веб-інтерфейс	+	+	-

1.	2.	3.	4.
Можливість встановлення на ПК	+	-	+
Внесення змін в функціонал	+	-	-
Модульність	+	-	-
Підтримка	+	+	-
Можливість роботи офлайн	-+	-	+
Інтеграції	+	+	-

1.3 Вибір архітектури системи автоматизації процесу обробки даних для ведення оперативного обліку та бухгалтерської звітності

Перед розробкою програмного забезпечення важливо проектувати його архітектуру, щоб забезпечити ефективність, масштабованість, довготривалу життєздатність продукту та можливість швидкого внесення змін. Найважливішим пунктом є можливість швидкого та легкого внесення змін, особливо для подібних систем ведення обліку. Адже будь-яка помилка при роботі з даними може призвести до зупинки штатної роботи що в свою чергу може призвести фінансових втрат бізнесу.

У світі розробки програмного забезпечення з'явилися два широко розповсюджені підходи, які є потужними інструментами для створення надійних та зручних для обслуговування додатків: Чиста архітектура (Clean Architecture) та доменно-орієнтоване проектування (DDD) [10].

З точки зору відмінностей, Clean Architecture в першу чергу фокусується на загальній структурі програмної системи, тоді як DDD зосереджується на моделюванні основної бізнес-області. Clean Architecture розглядає архітектурні рівні та управління залежностями, тоді як DDD надає принципи та шаблони для моделювання домену та співпраці з експертами в цій галузі [10].

Вибір між чистою архітектурою та DDD залежить від конкретних вимог та контексту проекту. Clean Architecture підходить для проектів, де архітектурні проблеми, такі як розділення проблем і можливість тестування, є

першочерговими. З іншого боку, DDD більш доречний, коли складність предметної області та співпраця з експертами в цій галузі відіграють значну роль у формуванні дизайну програмного забезпечення. Враховуючи що при розробці проекту не знадобиться співпраця з експертами в галузі та потрібно сфокусуватися на загальній структурі тому для розробки проекту було обрано Clean Architecture. Розглянемо ближче що з себе представляє Clean Architecture.

Clean Architecture – це архітектурний патерн, який приділяє велику увагу організації та структурі програмної системи. Його основні цілі включають зручність супроводу, гнучкість і можливість тестування. Шаблон сприяє розділенню проблем і виступає за незалежність від зовнішніх фреймворків і технологій [10].

Clean Architecture складається з окремих прошарків, включаючи сутності, варіанти використання, інтерфейси та фреймворки/драйвери, кожен з яких має чітко визначені обов'язки та залежності. Архітектура забезпечує дотримання правила залежності, згідно з яким залежності завжди повинні бути спрямовані всередину, що забезпечує високу модульність і легку заміну або модифікацію зовнішніх компонентів [10].

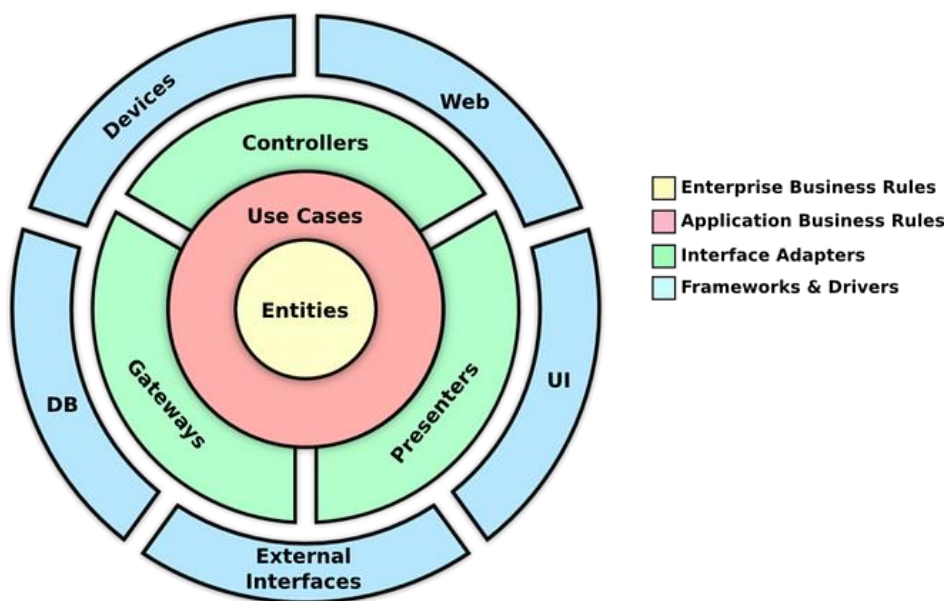


Рис. 1.3 – схема структури Clean Architecture

Розроблюване програмне забезпечення буде розділено на наступні прошарки, кожен з яких буде представлено у вигляді окремого проекту в середині одного рішення:

- **Domain;**
- **BL;**
- **Infrastructure;**
- **Presentation.**

Давайте подивимось, як буде організований код у вигляді прошарків у .NET Core додатку.

Прошарок Core або Domain є центом проекту розроблюваного з використанням Clean Architecture, і всі залежності в проекті повинні вказувати на даний прошарок, в собі він може містити [11]:

- доменні сутності;
- інтерфейси;
- системі перерахунку.

Нижче наведено зразок структури проекту.

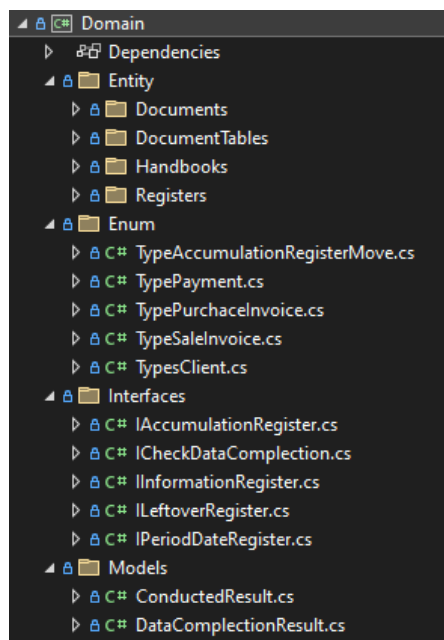


Рис 1.4 – схема структури прошарку Domain

Прошарок бізнес логіки. На цьому рівні ми розробляємо логіку домену з відповідною реалізацією. Інтерфейси цього рівня керують як бізнес-вимогами,

так і реалізацією. Для реалізації бізнес-логіки та виконання операцій, пов'язаних з варіантами використання, ми використовуємо прикладний рівень. Але як же бізнес рівень зможе реалізувати логіку, не знаючи, як реалізовані репозиторії або інші зовнішні сервіси. Для роз'єднання прошарків використовується механізм DIP [11, 12].

Даний прошарок містить :

- моделі бізнес логіки;
- інтерфейси для реалізації сервісів;
- валідатори;
- мапери.

Нижче наведено зразок структури проекту.

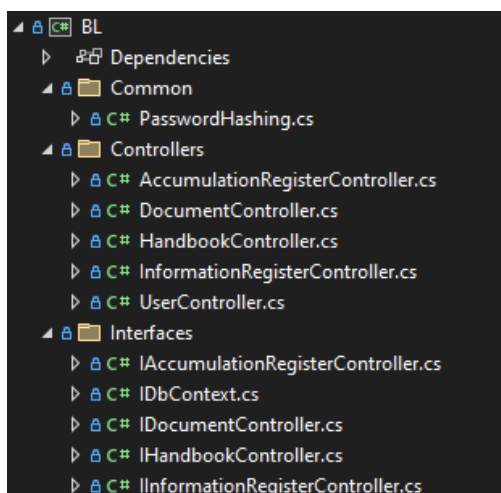


Рис 1.5 – схема структури прошарку бізнес логіки

Прошарок інфраструктури. Більшість залежностей додатку від зовнішніх ресурсів повинні бути реалізовані через класи, визначені в інфраструктурному проекті. Ці класи повинні відповідати інтерфейсам, описаним в прошарку Domain або бізнес логіки. Ці залежності можуть включати, наприклад, провайдерів електронної пошти, доступ до файлів, клієнти веб-API тощо [11]. У цьому проекті в даному прошарку буде реалізовано основний інтерфейс підключення до бази даних з використанням EF Core. Завдяки цьому база даних залежить виключно від описаних базових дій в бізнес логіці, це гарантує відсутність прямих зв'язків з прошарком бізнес логіки та Domain. Інші ж залежності враховуючи те що вони не являються

зовнішніми елементами, будуть реалізовані в прошарку бізнес логіки. Вони будуть створені для можливості абстрагування при роботі зі базою даних.

Нижче наведено приклад того, як виглядатиме проект, якщо звернути увагу на те, що всі залежності абстраговано до цього проекту, таким чином ми можемо замінити реалізацію бази даних із використанням Entity Framework Core на будь-яку іншу, тай взагалі якби було більше залежностей ми могли б замінити будь-яку з них, не зачіпаючи інші прошарки.

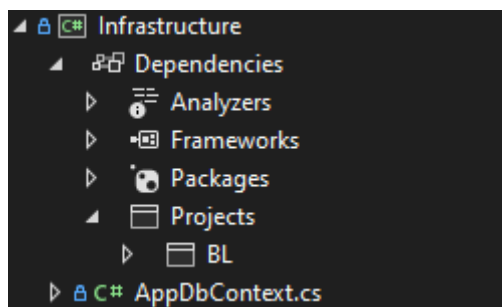


Рис 1.6 – схема структури прошарку інфраструктури

Прошарок відображення. На цьому етапі здійснюється реалізація логіки інтерфейсу користувача, керуючись інтерфейсами, які диктуються як бізнес-вимогами, так і реалізацією. Останній рівень, рівень інтерфейсу користувача, слугує споживачем усіх інших рівнів у цій архітектурній структурі. На цьому рівні ми визначимо всі конфігурації сервісів для ін'єкції залежностей. Завдяки цьому ми можемо створювати декілька варіантів інтерфейсів та налаштовувати їх по різному не змінюючи нічого в основній логіці проекту [11].

Нижче наведено зразок структури проекту.

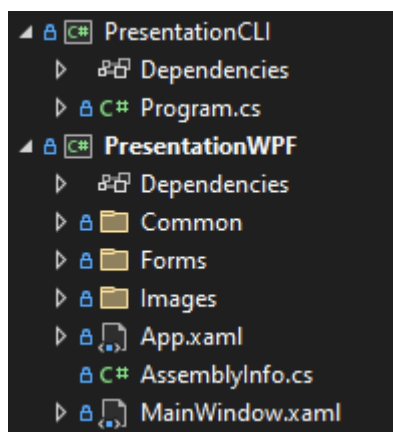


Рис 1.7 – схема структури прошарку відображення

1.4 Інструментальні засоби реалізації програмного продукту та вимоги до апаратних засобів.

Для реалізації облікової системи було обрано наступний стек технологій, оскільки він відповідає нашим вимогам щодо ефективності, швидкості розробки:

1. .NET: це безкоштовна крос-платформена платформа з відкритим вихідним кодом для створення сучасних додатків та потужних хмарних сервісів [13].

2. C#: мову програмування C# було обрано, оскільки вона є потужною, ефективно та добре підтримується в екосистемі .NET. Вона дозволяє швидко розробляти та забезпечує високу продуктивність [14].

3. WPF: це фреймворк для створення інтерфейсів додатків під операційну систему «Windows», попри те що це не дуже він не дуже ефективний при великій кількості елементів збільшується використання оперативної або відео пам'яті що може призвести до уповільнення роботи [15, 16].

4. Entity Framework Core 8: є потужним ORM для роботи з базами даних в .NET. Entity Framework Core пропонує багато можливостей і покращень, що дозволить нам ефективно взаємодіяти з базою даних [17].

5. SQLite: було обрано як база даних для нашої системи через його легкість використання, порівняно з іншими реляційними базами даних. Він працює на багатьох платформах і не вимагає окремого сервера, що робить його ідеальним вибором для невеликих до середніх проектів, як наш [18].

6. Visual Studio 2022: середовище розробки Visual Studio є творчим стартовим майданчиком, який можна використовувати для редагування, налагодження та складання коду, а також для публікації програми. На додаток до стандартного редактора та відладчика, що надаються більшістю інтегрованих середовищ розробки, Visual Studio включає компілятори, засоби завершення коду, графічні конструктори та багато інших функцій для покращення процесу розробки програмного забезпечення [19].

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						20

Облікова система, як будь-яке програмне забезпечення, має свої вимоги до комп'ютерної системи, на якій вона буде працювати. Ось деякі загальні вимоги до комп'ютера для нашої облікової системи:

1. Операційна система Windows 10.
2. Процесор мінімально 2-х ядерний з тактовою частотою 2.3 ГГц.
3. Оперативна пам'ять 8 ГБ.
4. Місце на диску 128 МБ.

Висновки до першого розділу

В першому розділі було проведено аналіз предметної області та визначено мету та завдання на бакалаврську роботу. Розглянуто існуючих конкурентів по системах оперативного обліку та бухгалтерської звітності. Було зроблено вибір архітектури на користь Clean Architecture. Дана архітектура допоможе розділити проект для подальшого його організації та доопрацювання. Також було визначено структуру та інструментальних засобів для реалізації майбутнього проекту.

РОЗДІЛ 2. Проектування системи автоматизації обробки даних для оперативного обліку та бухгалтерської звітності

2.1 Визначення варіантів використання системи

Розроблювана облікова система призначена для легкого ведення базового обліку без непотрібних елементів що можуть ускладнити процес ведення обліку на початку. Тому було виділено основні дії при роботі з обліковими системами які найчастіше використовуються користувачами інших облікових систем та тільки їх було додано в проект. Нижче наведена діаграма варіантів використання (рис. 2.1).

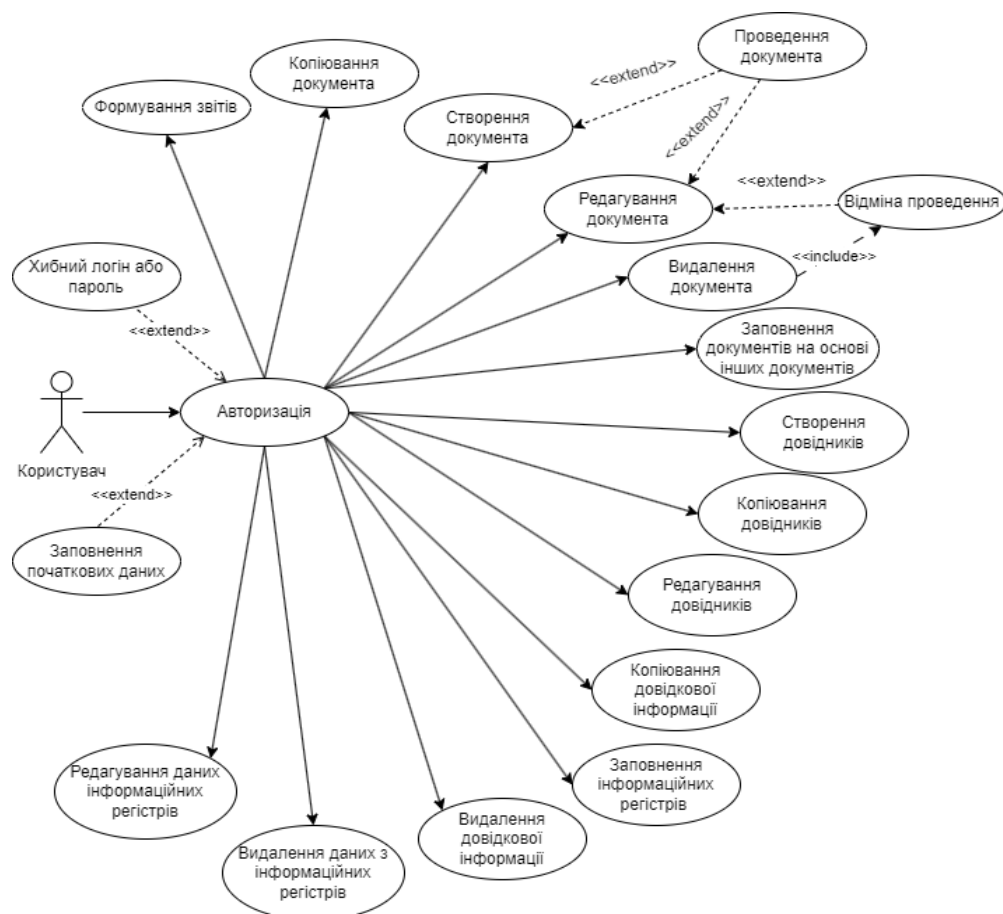


Рис. 2.1 – діаграма варіантів використання

Опис діаграми варіантів використання наведено в (табл. 2.1 – 2.11)

Таблиця 2.1

Опис прецеденту «Авторизація»

Назва варіанту використання	Авторизація
Основні актори	Користувач
Короткий опис	Для забезпечення безпеки даних обліку та одноосібного доступу до них користувачу потрібно ввести свій логін та пароль. Якщо ж це перший запуск користувачу буде запропоновано створити нового користувача.

Таблиця 2.2

Опис прецеденту «Створення довідників/документів»

Назва варіанту використання	Створення довідників/документів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість створювати власні елементи наявних в системі довідників/документів для подальшого їхнього використання.

Таблиця 2.3

Опис прецеденту «Редагування довідників/документів»

Назва варіанту використання	Редагування довідників/документів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість редагувати наявні елементи довідників/документів.

Таблиця 2.4

Опис прецеденту «Видалення довідників/документів»

Назва варіанту використання	Видалення довідників/документів
Основні актори	Користувач
Короткий опис	Для забезпечення цілісності даних користувачу надається можливість установити помітку видалення, а не повністю видаляти наявні елементи довідників/документів із бази даних.

Таблиця 2.5

Опис прецеденту «Проведення документів»

Назва варіанту використання	Проведення документів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість проводити документи. При проведенні документів формуються записи в регістрах відомостей які по іншому називають рухи документа вони і формують облік.

Таблиця 2.6

Опис прецеденту «Відміна проведення документів»

Назва варіанту використання	Відміна проведення документів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість відмінити проведений документ, що видалить його рухи із регістрів накопичення.

Таблиця 2.7

Опис прецеденту «Формування звітів»

Назва варіанту використання	Формування звітів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість формувати визначені в системі звіти для аналізу оперативного обліку та взаєморозрахунків з клієнтами.

Таблиця 2.8

Опис прецеденту «Заповнення інформаційних регістрів»

Назва варіанту використання	Заповнення інформаційних регістрів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість створювати записи в розрізі визначених вимірів що гарантує їхню унікальність в наданих розрізах та дозволяє однозначно вибрати дані на основі заданого відбору.

Таблиця 2.9

Опис прецеденту «Редагування даних інформаційних реєстрів»

Назва варіанту використання	Редагування даних інформаційних реєстрів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість редагувати дані записів реєстрів в розрізі визначених вимірі враховуючи і самі виміри.

Таблиця 2.10

Опис прецеденту «Видалення даних інформаційних реєстрів»

Назва варіанту використання	Видалення даних інформаційних реєстрів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість видаляти записи інформаційних реєстрів.

Таблиця 2.11

Опис прецеденту «Копіювання документів/довідників»

Назва варіанту використання	Створення довідників/документів
Основні актори	Користувач
Короткий опис	Користувачу надається можливість копіювати наявні в системі документи та довідники.

Функціональні вимоги:

- створення довідникової інформації та документів;
- копіювання довідникової інформації та документів;
- редагування довідникової інформації та документів;
- встановлення помітки видалення для елементів довідників та документів;
- проведення документів;
- відміна проведення документів;
- формування звітності по залишкам товарів, боргам клієнтів та продажам;
- створення записів в реєстрах відомостей;
- редагування записів в реєстрах відомостей;

- видалення записів в реєстрах відомостей;
- авторизація для входу в систему;
- заповнення документів на основі інших.

Не функціональні вимоги:

- система повинна забезпечити безперебійну роботу та збереження даних;
- система повинна перевіряти коректність введених даних;
- система повинна бути стійкою до помилок пов'язаних із базою даних;
- система повинна швидко та коректно працювати на вказаних апаратних вимогах.

Аналіз вимог та визначення переліку об'єктів в пункті 1.1 які будуть забезпечувати роботу програмного додатку дозволило сформулювати діаграму класів додатку. Для кожного із шарів додатку було створено окрему діаграму класів для кращого розуміння що в кожному із них буде знаходитися. Загальна діаграма класів із всіма зв'язками знаходиться в додатках. При створенні діаграми об'єкти було розділено по фреймам і позначено кольорами:

- жовтий – інтерфейси (interface);
- зелений – класи (class);
- синій – перелік (enum).

Було створено додаткове правило по яким будувалася діаграма, завдяки якому вдалося зменшити кількість відображуваних одночасно зв'язків та збільшити структурованість. Правило полягає в групуванні класів в фрейми що об'єднує їх, від групування можна будувати зв'язок із іншими класами. Зв'язок між групуванням і іншим об'єктом означає що всі елементи що знаходяться в групуванні мають даний зв'язок із цим елементом. Дане правило не виключає можливості класів що знаходяться в групі відображати окремі зв'язки з елементами які не пов'язані із іншими класами що знаходяться в групі. В місцях де зв'язки часто перетиналися для зв'язків було змінено колір щоб

відокремити їх один від одного наглядно, колір не несе ніякої додаткової інформації він слугує тільки для кращої демонстрації зв'язку.

Додаток було розділено на 4 прошарки «Domain», «BL», «Infrastructure» та «PresentationWPF» в межах архітектурного підходу Clean architecture.

На (рис 2.2) наведено діаграму класів прошарку «PresentationWPF». Дана діаграма відображає лише класи прошарку без зовнішніх зв'язків, повна діаграма зі зв'язками знаходиться в додатках (Додаток А).

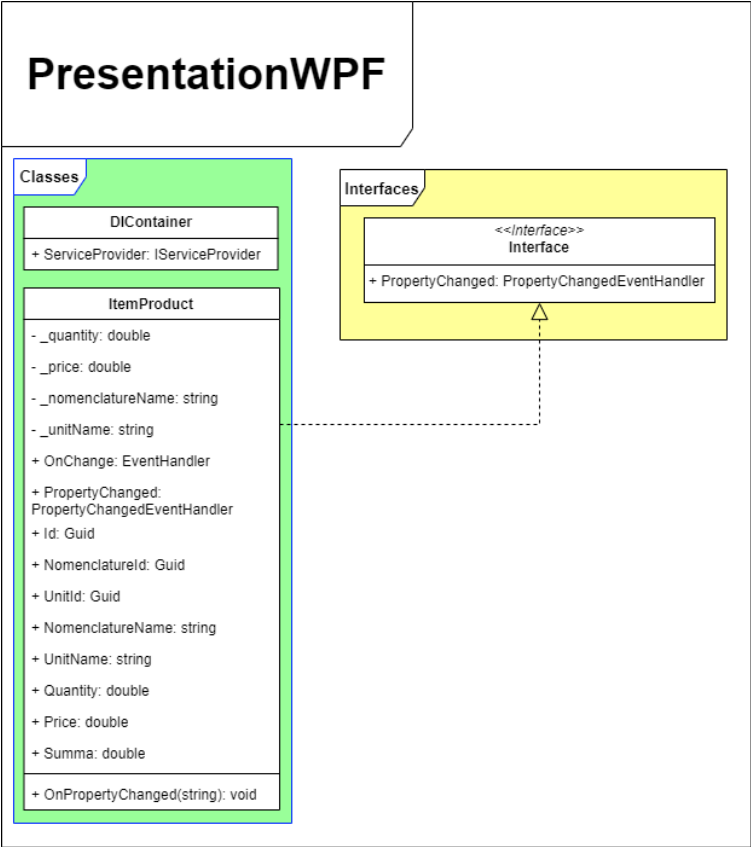


Рис. 2.2 – Діаграма класів прошарку «PresentationWPF»

Опис діаграми класів (див. рис. 2.2). Діаграма складається з 2-х класів та 1-го інтерфейсу.

DIContainer – статичний клас що містить в собі контейнер із залежностями проекту. Даний клас використовується для реалізації шаблону проектування «Service locator» суть даного шаблону в тому щоб використовувати контейнер із залежностями на пряму не вказуючи залежності класів через параметри конструктора [20]. Даний шаблон використовується лише на рівні «PresentationWPF».

ItemProduct – клас який рядок товару в документах замовлення покупця, замовлення постачальника, видаткова накладка, прибуткова накладна:

- **OnPropertyChanged(string)** – метод що оброблює зміну даних у властивостях класу та викликає оновлення відображення на формі.

В якості параметра приймає назву змінюваної властивості.

На (рис. 2.3) наведено діаграму класів для прошарку «Infrastructure». Дана діаграма відображає лише класи прошарку без зовнішніх зв'язків, повна діаграма зі зв'язками знаходиться в додатках (Додаток А).

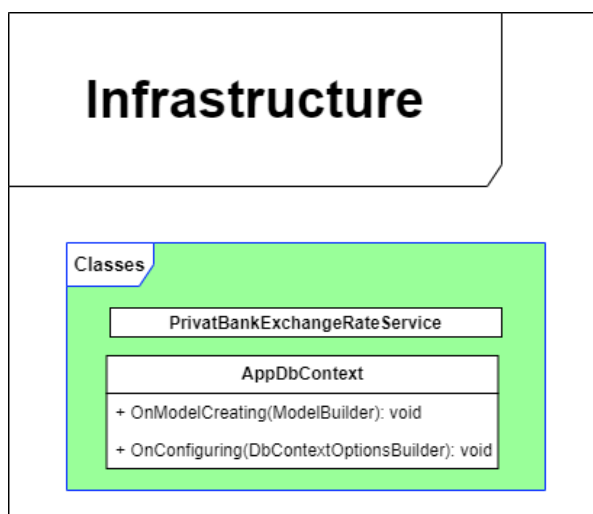


Рис. 2.3 – Діаграма класів прошарку «Infrastructure»

Опис діаграми класів (див. рис. 2.3). Діаграма складається з 2-х класів.

PrivateBankExchangeRateService – клас що відповідає за оновлення інформації по курсах валют.

AppDbContext – клас що реалізує взаємодію із базою даних:

- **OnConfiguring(DbContextOptionsBuilder)** – перевизначений метод, відповідає за початкове налаштування класу для взаємодії із базою даних. Параметром передається об'єкт через який відбувається налаштування класу [21];
- **OnModelCreating(ModelBuider)** – перевизначений метод, відповідає за створення та налаштування моделей які будуть використовуватися. Параметром передається об'єкт через який відбувається налаштування моделей класу [22].

Рис. 2.4 – діаграма класів прошарку «VL»

Опис діаграми класів (див. рис. 2.4). Діаграма складається із 7-ми класів та 6-ти інтерфейсів.

PasswordHashing – клас що відповідає за хешування пароллю:

- Hash(string) – виконує хешування переданого рядка;
- HashAsync(string) – виконує хешування переданого рядка асинхронно.

ExgangeRate – клас модель що описує дані курсу валюти.

UserController – клас що відповідає за авторизацію користувачів:

- `Authenticate(string, string)` – виконує пошук користувача за переданими логіком та паролем;
- `LoginIsUniq(string)` – перевіряє чи існує переданий логін в базі даних.

HandbookController – клас для взаємодії із об'єктами довідників:

- `GetNextCode<T>(DbSet<T>)` – пропорохує новий код для елемента довідника.

DocumentController – клас для взаємодії із об'єктами документів:

- **GetNextNumber<T>(DbSet<T>)** – прораховує новий номер для документу.

InformationRegisterController – клас для взаємодії із об'єктами реєстрів відомостей.

AccumulationRegisterController – клас для взаємодії із об'єктами реєстрів накопичення.

IHandbookController – інтерфейс що описує функції які має реалізувати клас взаємодії з довідниками:

- **GetHandbooks<T>(Expression<Func<T, bool>>, int, int)** – функція повертає список елементів вказаних шаблоном, приймає делегат для відбору елементів результату, кількість елементів яку пропустити та кількість елементів яку вибрати;
- **GetHandbook<T>(Guid)** – функція виконує пошук довідника по унікальному ідентифікатору, якщо елемента з ідентифікатором не існує повертає null;
- **GetHandbook<T>(Func<T, bool>)** – функція виконує пошук довідника по заданому відбору переданому делегатом, якщо елемента не знайдено за відбором, повертає null;
- **AddOrUpdateAsync<T>(T, bool)** – виконує додавання або зміну наявного елемента в базі даних, повертає унікальний ідентифікатор доданого елемента;
- **AddOrUpdateRangeAsync<T>(IEnumerable<T>)** – виконує додавання або зміну списку переданих елементів в базу даних;
- **RemoveAsync<T>(Guid, bool)** – виконує видалення елемента із бази даних за переданим унікальний ідентифікатором.

IDocumentController – інтерфейс що описує функції які має реалізувати клас взаємодії з документами:

- `GetDocuments<T>(int, int)` – функція повертає список документів заданого шаблону, приймає кількість елементів яку пропустити та кількість елементів яку вибрати;
- `GetDocument<T>(Guid)` – функція виконує пошук документу по унікальному ідентифікатору, якщо елемента з ідентифікатором не існує повертає `null`;
- `AddOrUpdateAsync<T>(T, bool)` – виконує додавання або зміну наявного елемента в базі даних, повертає унікальний ідентифікатор доданого елемента;
- `RemoveAsync<T>(Guid, bool)` – виконує видалення елемента із бази даних за переданим унікальний ідентифікатором;
- `RemoveRangeAsync<T>(Func<T, bool>)` – виконує видалення масиву елементів що відповідають відбору переданому делегату;
- `ConductedDocumentAsync<T>(T)` – виконує проведення документу, при проведенні, при наявності, перевіряються та записуються рухи документа в реєстри накопичення, повертає результат проведення із списком помилок якщо такі є;
- `UnConductedDocumentAsync<T>(T)` – виконує відміну проведення документу, видаляє наявні рухи документа із реєстрів накопичення.

InformationRegisterController – інтерфейс описує функції які має реалізувати клас для взаємодії із реєстрами відомостей:

- `GetListData<T>(int, int, Func<T, bool>)` – функція повертає список елементів заданих шаблоном, в якості параметрів приймає кількість елементів що пропускаються, кількість елементів що вибираються та делегат відбору для обмеження вибірки за параметрами;
- `GetLastDateData<T>(DateTime, Func<T, bool>)` – повертає список актуальних елементів заданих шаблоном на дату, в якості параметрів приймає дату на яку потрібно отримати дані та делегат відбору;

- **AddOrUpdateAsync<T>(T, bool, bool)** – виконує додавання або зміну наявного елемента в базі даних, повертає унікальний ідентифікатор доданого елемента;
- **AddOrUpdateRangeAsync(IEnumerable<IInformationRegister>)** – виконує додавання або зміну списку елементів у базі даних;
- **DeleteAsync<T>(Func<T, bool>)** – виконує видалення елементів за переданим делегатом відбору.

IAccumulationRegisterController – інтерфейс описує функції які має реалізувати клас для взаємодії із реєстрами накопичення:

- **GetDataList<T>(Func<T, bool>)** – повертає список елементів встановленого шаблону, в якості параметрів приймає делегат відбіру;
- **GetLeftoverList<T, G, V>(List<T>, Func<T, G>, Func<IGrouping<G, T>, V>)** – повертає групований список елементів заданих шаблоном V, в якості параметрів приймає початковий не групований список, делегат групування та делегат що перетворює результат групування в шаблон типу V;
- **AddOrUpdateAsync(IAccumulationRegister, bool)** – виконує додавання або зміну наявного елемента в базі даних;
- **AddOrUpdateRangeAsync(IEnumerable<IAccumulationRegister>)** – виконує додавання або зміну списку елементів у базі даних;
- **RemoveAsync(IAccumulationRegister)** – виконує видалення елемента із бази даних.
- **RemoveRangeAsync(IEnumerable<IAccumulationRegister>)** – виконує видалення елементів із бази даних.

IExchangeRateService – інтерфейс що описує функції які має реалізувати клас для отримання даних курсу валют:

- **GetExchangeRatesAsync(string)** – виконує отримання курсів валют.

IDbContext – інтерфейс що описує контекст бази даних додатку та її функції:

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						32

- `GetPropertyData<T>()` – повертає властивість контексту бази даних яке відповідає за взаємодію із заданим шаблоном типу;
- `IncludeVirtualProperty<T>(IQueryable<T>)` – включає в запит поля об’єктів що позначені модифікатором `virtual` і являються класами або колекціями (працює як ліве з’єднання двох таблиць в мові запитів SQL);
- `SaveChangesAsync(CancellationToken)` – виконує збереження внесених змін до бази даних, в якості параметрів приймає токен відміни операції [23];
- `Add(object)` – виконує відстеження сутності, при збереженні даних сутності будуть додані до бази даних [24];
- `Update(object)` – виконує відстеження даних сутності, встановлює стан сутності «Модифіковано» в контексті бази даних [25];
- `Remove(object)` – виконує відстеження сутності із встановленням стану «Видалено» при збереженні контексту бази даних сутність буде видалено [26];
- `UpdateRange(IEnumerable<object>)` – виконує відстеження даних списку сутностей, встановлює для сутностей стан «Модифіковано» в контексті бази даних [27];
- `RemoveRange(IEnumerable<object>)` – виконує відстеження даних списку сутностей, встановлює для сутностей стан «Видалено» при збереженні контексту бази даних сутності будуть видалені [28];
- `Entry(object)` – повертає для переданої сутності об’єкт що дозволяє відстежити стан сутності в контексті бази даних [29].

В прошарку «BL» для класів які пов’язані з роботою над базою даних використовується шаблон проектування DI. При даному підході залежності передаються в якості посилань в конструктор класу.[30] Сам контейнер із залежностями конфігурується на рівні «PresentationWPF».

На (рис. 2.5) наведено діаграму класів прошарку «Domain». Дана діаграма відображає лише класи прошарку без зовнішніх зв'язків, повна діаграма зі зв'язками знаходиться в додатках (Додаток А).

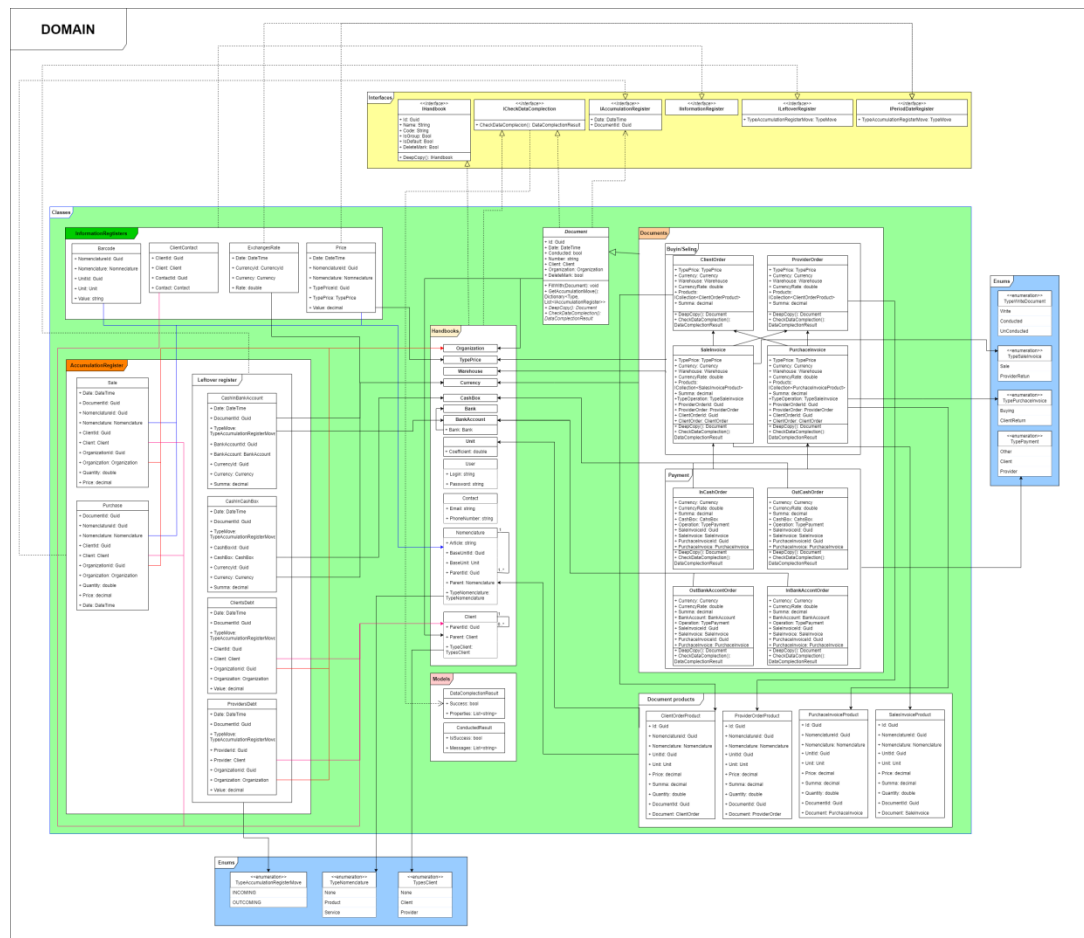


Рис. 2.5 – діаграма класів прошарку «Domain»

Опис діаграми класів (див. рис. 2.5). Діаграма складається із 37-ми класів, 6-ти інтерфейсів та 7-ми переліків.

Bank – клас що описує сутність довідника банку.

BankAccount – клас що описує сутність довідника банківського рахунку.

CashBox – клас що описує сутність довідника каси.

Client – клас що описує сутність довідника контрагента.

Contact – клас що описує сутність довідника контакту.

Currency – клас що описує сутність довідника валюти.

Nomenclature – клас що описує сутність довідника номенклатури.

Organization – клас що описує сутність довідника організації.

TypePrice – клас що описує сутність типу ціни.

Unit – клас що описує сутність довідника одиниці виміру.

User – клас що описує сутність довідника користувача.

Warehouse – клас що описує сутність довідника складу.

Document – абстрактний клас який повинні наслідувати всі класи документи, містить в собі базові властивості та функції для документу:

- FillWith(Document) – функція яка виконує заповнення документу на основі даних іншого документу, якщо клас наслідник її перевизначив;
- GetAccumulationMove() – повертає список рухів документу;
- CheckDataComplection() – абстрактна функція виконує перевірку заповнення даних документу, реалізувати її повинен кожен клас наслідувач;
- DeepCopy() – функція для реалізації патерну «Прототип», виконує копіювання документу, реалізувати її повинен кожен клас наслідувач [31].

ClientOrder – клас що описує сутність документу замовлення покупця:

- CheckDataComplection() – виконує перевірку заповнення даних документу замовлення покупця;
- DeepCopy() – функція для реалізації патерну «Прототип», виконує копіювання документу замовлення покупця [31].

InBankAccountOrder – клас що описує сутність документу надходження грошових коштів на банківський рахунок:

- CheckDataComplection() – виконує перевірку заповнення даних документу надходження грошових коштів на банківський рахунок;
- DeepCopy() – функція для реалізації патерну «Прототип», виконує копіювання документу надходження грошових коштів на банківський рахунок [31].

InCashOrder – клас що описує сутність документу надходження грошових коштів в касу:

- **CheckDataComplection()** – виконує перевірку заповнення даних документу надходження грошових коштів в касу;
- **DeerCopy()** – функція для реалізації патерну «Прототип», виконує копіювання документу надходження грошових коштів в касу [31].

OutBankAccontOrder – клас що описує сутність документу витрата грошових коштів із банківського рахунку:

- **CheckDataComplection()** – виконує перевірку заповнення даних документу витрата грошових коштів із банківського рахунку;
- **DeerCopy()** – функція для реалізації патерну «Прототип», виконує копіювання документу витрата грошових коштів із банківського рахунку [31].

OutCashOrder – клас що описує сутність документу витрата грошових коштів із каси:

- **CheckDataComplection()** – виконує перевірку заповнення даних документу витрата грошових коштів із каси;
- **DeerCopy()** – функція для реалізації патерну «Прототип», виконує копіювання документу витрата грошових коштів із каси [31].

ProviderOrder – клас що описує сутність документу замовлення постачальнику:

- **CheckDataComplection()** – виконує перевірку заповнення даних документу замовлення постачальнику;
- **DeerCopy()** – функція для реалізації патерну «Прототип», виконує копіювання документу замовлення постачальнику [31].

PurchaceInvoice – клас що описує сутність документу прибуткова накладна:

- **CheckDataComplection()** – виконує перевірку заповнення даних документу прибуткова накладна;
- **DeerCopy()** – функція для реалізації патерну «Прототип», виконує копіювання документу прибуткова накладна [31].

SaleInvoice – клас що описує сутність документу видаткова накладна:

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						36

- **CheckDataComplection()** – виконує перевірку заповнення даних документу видаткова накладна;
- **DeerpCopy()** – функція для реалізації патерну «Прототип», виконує копіювання документу видаткова накладна [31].

ClientOrderProduct – клас що описує сутність товару в замовлені покупця.

ProviderOrderProduct – клас що описує сутність товару в замовлені постачальнику.

PurchaceInvoiceProduct – клас що описує сутність товару в прибутковій накладній.

SalesInvoiceProduct – клас що описує сутність товару у видатковій накладній.

Barcode – клас що описує сутність реєстру відомостей штрих кодів номенклатури.

ClientContact – клас що описує сутність реєстру відомостей зв'язку контрагентів із контактами.

ExchangesRate – клас що описує сутність реєстру відомостей курсів валют.

Price – клас що описує сутність реєстру відомостей ціни номенклатури.

CashInBankAccount – клас що описує сутність реєстру накопичення грошові кошти на банківському рахунку.

CashInCashBox – клас що описує сутність реєстру накопичення грошові кошти в касі.

ClientsDebt – клас що описує сутність реєстру накопичення борги клієнтів.

Leftover – клас що описує сутність реєстру накопичення залишки номенклатури.

ProvidersDebt – клас що описує сутність реєстру накопичення борги постачальників.

Purchase – клас що описує сутність реєстру накопичення закупівлі.

Sale – клас що описує сутність реєстру накопичення продажі.

ConductedResult – клас що описує результат виконання проведення документу.

DataComplectionResult – клас що описує результат перевірки наповненості об'єкту.

IHandbook – інтерфейс який повинні реалізувати довідники, описує обов'язкові властивості що повинні бути у довідника:

- DeepCopy() – функція для реалізації патерну «Прототип», виконує копіювання об'єкту довідника [31].

IAccumulationRegister – інтерфейс який повинні реалізувати реєстри накопичення, описує обов'язкові властивості для реєстра накопичення.

ILeftoverRegister – інтерфейс який повинні реалізувати реєстри накопичення якщо в них зберігається залишкова інформація.

InformationRegister – інтерфейс який повинні реалізувати реєстри відомостей.

IPeriodDateRegister – інтерфейс який повинні реалізувати реєстри відомостей якщо по ним можна отримувати інформацію на дату (наприклад отримати курс валюти на задану дату, навіть якщо на вказану дату немає записів).

ICheckDataComplection – інтерфейс який зобов'язує реалізувати функцію перевірки заповнення даних об'єкту.

TypeAccumulationRegisterMove – перелік типів рухів залишкових реєстрів накопичення.

TypeNomenclature – перелік типів номенклатури.

TypePayment – перелік типів оплат.

TypePurchaseInvoice – перелік типів прибуткових накладних.

TypeSaleInvoice – перелік типів видаткових накладних.

TypesClient – переліки типів контрагентів.

ICheckDataComplection – перелік типів запису документу.

2.2 Розробка бази даних системи

В якості місця збереження даних було обрано СКБД SQLite. Для роботи з SQLite не потрібно встановлювати додаткового програмного забезпечення, що в свою чергу спрощує встановлення та початок роботи із додатком [18]. Для взаємодії з базою даних використовується ORM EF Core, даний фреймворк надає декілька підходів до розробки бази даних:

- code first – спочатку описуються класи сутності додатку після чого на їхній основі генеруються таблиці бази даних [32];
- data base first – спочатку створюється база даних із таблицями після чого створюють класи сутностей що описують таблиці бази даних [33].

При розробці додатку було використано підхід Code first. На основі описаних сутностей в додатку EF Core створив 35 таблиць в базі даних:

- **Banks** – таблиця для збереження даних банків;
- **BankAccounts** – таблиця для збереження даних банківських рахунків;
- **Barcodes** – таблиця для збереження даних штрих кодів номенклатури;
- **CashBoxes** – таблиця для збереження даних кас;
- **CashInCashBackAccounts** – таблиця для збереження даних про грошові кошти на рахунках;
- **CashInCashBoxes** – таблиця для збереження даних про грошові кошти в касах;
- **ClientOrderProducts** – таблиця для збереження даних про товари замовлення покупця;
- **ClientOrders** – таблиця для збереження даних документу замовлення покупця;
- **Clients** – таблиця для збереження даних контрагентів;

- **ClientsContacts** – таблиця для збереження даних зв'язку контактів та контрагентів;
- **ClientsDebts** – таблиця для збереження даних боргів клієнтів;
- **Contacts** – таблиця для збереження даних контактів;
- **Currencies** – таблиця для збереження даних валют;
- **ExchangesRates** – таблиця для збереження даних курсів валют;
- **InBankAccountOrders** – таблиця для збереження даних документів надходження грошових коштів на рахунок;
- **InCashOrders** – таблиця для збереження даних документів надходження грошових коштів в касу;
- **Leftovers** – таблиця для збереження даних залишків номенклатури;
- **Nomenclatures** – таблиця для збереження даних номенклатури;
- **Organizations** – таблиця для збереження даних організацій;
- **OutBankAccountOrders** – таблиця для збереження даних документів витрат грошових коштів із банківського рахунку;
- **OutCahsOrders** – таблиця для збереження даних документів витрат грошових коштів із кас;
- **Prices** – таблиця для збереження даних цін номенклатури;
- **ProviderOrderProducts** – таблиця для збереження даних товарів замовлення постачальника;
- **ProviderOrders** – таблиця для збереження даних документів замовлення постачальника;
- **ProvidersDebts** – таблиця для збереження даних боргів постачальників.
- **PurchaseInvoiceProducts** – таблиця для збереження даних товарів прибуткової накладної;
- **PurchaseInvoices** – таблиця для збереження даних документів прибуткових накладних;

- **Purchases** – таблиця для збереження даних про закупки номенклатури;
- **Sales** – таблиця для збереження даних про продажі номенклатури;
- **SalesInvoiceProducts** – таблиця для збереження даних про товари видаткових накладних;
- **SalesInvoices** – таблиця для збереження даних документів видаткових накладних;
- **TypePrices** – таблиця для збереження даних типів цін;
- **Units** – таблиця для збереження даних одиниць виміру;
- **Users** – таблиця для збереження даних користувачів;
- **Warehouses** – таблиця для збереження даних складів.

Опис структури кожної з таблиць наведено у (табл. 2.12 – 2.36).

Таблиця 2.12

Опис таблиць «Banks», «CashBoxes», «Currencies», «Organizations», «Warehouses», «TypePrices»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
Name	TEXT	Найменування банку
Code	TEXT(9)	Код банку
IsGroup	INTEGER	Прапорець групи
IsDefault	INTEGER	Прапорець елемента по-замовчуванню
DeleteMark	INTEGER	Прапорець помітки видалення

Таблиця 2.13

Опис таблиці «BankAccounts»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
Name	TEXT	Найменування банку
Code	TEXT(9)	Код банку
IsGroup	INTEGER	Прапорець групи
IsDefault	INTEGER	Прапорець елемента по-замовчуванню
DeleteMark	INTEGER	Прапорець помітки видалення
BankId	TEXT	Ідентифікатор банку

Таблиця 2.14

Опис таблиці «Barcodes»

Поле	Тип	Призначення
NomenclatureId	TEXT	Ідентифікатор номенклатури
UnitId	TEXT	Ідентифікатор одиниці виміру
Value	TEXT	Значення штрих коду

Таблиця 2.15

Опис таблиці «CashInBankAccounts»

Поле	Тип	Призначення
Date	TEXT	Дата запису
DocumentId	TEXT	Ідентифікатор документу
BankAccountId	TEXT	Ідентифікатор банківського рахунку
CurrencyId	TEXT	Ідентифікатор валюти
TypeMove	INTEGER	Тип запису
Summa	TEXT	Сума

Таблиця 2.16

Опис таблиці «CashInCashBoxes»

Поле	Тип	Призначення
Date	TEXT	Дата запису
DocumentId	TEXT	Ідентифікатор документу
CashBoxId	TEXT	Ідентифікатор каси
CurrencyId	TEXT	Ідентифікатор валюти
TypeMove	INTEGER	Тип запису
Summa	TEXT	Сума

Таблиця 2.17

Опис таблиць «ClientOrderProducts», «SalesInvoiceProducts»,
«ProviderOrderProducts», «PurchaseInvoiceProducts»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
NomenclatureId	TEXT	Ідентифікатор Номенклатури
UnitId	TEXT	Ідентифікатор одиниці виміру
Price	TEXT	Ціна
Summa	TEXT	Сума
Quantity	REAL	Кількість
DocumentId	TEXT	Ідентифікатор документу

Таблиця 2.18

Опис таблиць «ClientOrders», «ProviderOrders»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
TypePriceId	TEXT	Ідентифікатор типу ціни
CurrencyId	TEXT	Ідентифікатор валюти
WarehouseId	TEXT	Ідентифікатор складу
CurrencyRate	REAL	Курс валюти
Date	REAL	Дата документу
Conducted	INTEGER	Прапорець проведення документу
Number	TEXT(9)	Номер документу
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
DeleteMark	INTEGER	Прапорець помітки видалення документу

Таблиця 2.19

Опис таблиці «Clients»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
Name	TEXT	Найменування
Code	TEXT(9)	Код
IsGroup	INTEGER	Прапорець групи
IsDefault	INTEGER	Прапорець елемента по-замовчуванню
DeleteMark	INTEGER	Прапорець помітки видалення
ParentId	TEXT	Ідентифікатор батьківського елемента
TypeClient	INTEGER	Тип контрагента

Таблиця 2.20

Опис таблиці «ClientsContacts»

Поле	Тип	Призначення
ClientId	TEXT	Ідентифікатор контрагента
ContactId	TEXT	Ідентифікатор контакту

Таблиця 2.21

Опис таблиці «ClientsDebts»

Поле	Тип	Призначення
Date	TEXT	Дата
DocumetId	TEXT	Ідентифікатор документу
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації

Продовження таблиці 2.21

1.	2.	3.
TypeMove	INTEGER	Тип руху
Value	TEXT	Сума

Таблиця 2.22

Опис таблиці «Contacts»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
Email	TEXT	Електрона пошта
PhoneNumber	TEXT	Номер телефону
Name	TEXT	Найменування
Code	TEXT(9)	Код
IsGroup	INTEGER	Прапорець групи
DeleteMark	INTEGER	Прапорець помітки видалення
IsDefault	INTEGER	Прапорець елемента по-замовчуванню

Таблиця 2.23

Опис таблиці «ExchangesRates»

Поле	Тип	Призначення
Date	TEXT	Дата
CurrencyId	TEXT	Ідентифікатор валюти
Rate	REAL	Курс валюти

Таблиця 2.24

Опис таблиці «InBankAccontOrders»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
CurrencyId	TEXT	Ідентифікатор валюти
CurrencyRate	REAL	Курс валюти
AccountId	TEXT	Ідентифікатор банківського рахунку
Operation	INTEGER	Тип операції
Summa	TEXT	Сума
SaleInvoiceId	TEXT	Ідентифікатор видаткової накладної
PurchaseInvoiceId	TEXT	Ідентифікатор прибуткової накладної
Date	REAL	Дата документу
Conducted	INTEGER	Прапорець проведення документу
Number	TEXT(9)	Номер документу
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
DeleteMark	INTEGER	Прапорець помітки видалення документу

Таблиця 2.25

Опис таблиці «InCashOrders»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
CurrencyId	TEXT	Ідентифікатор валюти
CurrencyRate	REAL	Курс валюти
CashBoxId	TEXT	Ідентифікатор каси
Operation	INTEGER	Тип операції
Summa	TEXT	Сума
SaleInvoiceId	TEXT	Ідентифікатор видаткової накладної
PurchaseInvoiceId	TEXT	Ідентифікатор прибуткової накладної
Date	REAL	Дата документа
Conducted	INTEGER	Прапорець проведення документа
Number	TEXT(9)	Номер документа
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
DeleteMark	INTEGER	Прапорець помітки видалення документа

Таблиця 2.26

Опис таблиці «Leftovers»

Поле	Тип	Призначення
Date	TEXT	Дата
DocumentId	TEXT	Ідентифікатор документа
TypeMove	INTEGER	Тип руху
NomenclatureId	TEXT	Ідентифікатор номенклатури
WarehouseId	TEXT	Ідентифікатор складу
Value	REAL	Кількість

Таблиця 2.27

Опис таблиці «Nomenclatures»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
Article	TEXT	Артикул
BaseUnitId	TEXT	Ідентифікатор одиниці виміру
Name	TEXT	Найменування
Code	TEXT(9)	Код
IsGroup	INTEGER	Прапорець групи
IsDefault	INTEGER	Прапорець елемента по-замовчуванню
DeleteMark	INTEGER	Прапорець помітки видалення
ParentId	TEXT	Ідентифікатор батьківського елемента
TypeNomenclature	INTEGER	Тип номенклатури

Таблица 2.28

Опис таблиці «OutBankAccontOrders»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
CurrencyId	TEXT	Ідентифікатор валюти
CurrencyRate	REAL	Курс валюти
AccountId	TEXT	Ідентифікатор банківського рахунку
Operation	INTEGER	Тип операції
Summa	TEXT	Сума
SaleInvoiceId	TEXT	Ідентифікатор видаткової накладної
PurchaseInvoiceId	TEXT	Ідентифікатор прибуткової накладної
Date	REAL	Дата документу
Conducted	INTEGER	Прапорець проведення документу
Number	TEXT(9)	Номер документу
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
DeleteMark	INTEGER	Прапорець помітки видалення документу

Таблица 2.29

Опис таблиці «OutCashOrders»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
CurrencyId	TEXT	Ідентифікатор валюти
CurrencyRate	REAL	Курс валюти
CashBoxId	TEXT	Ідентифікатор каси
Operation	INTEGER	Тип операції
Summa	TEXT	Сума
SaleInvoiceId	TEXT	Ідентифікатор видаткової накладної
PurchaseInvoiceId	TEXT	Ідентифікатор прибуткової накладної
Date	REAL	Дата документу
Conducted	INTEGER	Прапорець проведення документу
Number	TEXT(9)	Номер документу
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
DeleteMark	INTEGER	Прапорець помітки видалення документу

Таблица 2.30

Опис таблиці «Prices»

Поле	Тип	Призначення
Date	TEXT	Дата
NomenclatureId	TEXT	Ідентифікатор номенклатури
TypePriceId	TEXT	Ідентифікатор типу ціни
Value	TEXT	Ціна

Таблиця 2.31

Опис таблиці «ProvidersDebts»

Поле	Тип	Призначення
Date	TEXT	Ідентифікатор
DocumentId	TEXT	Ідентифікатор документу
ProviderId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
TypeMove	INTEGER	Тип руху
Value	TEXT	Сума

Таблиця 2.32

Опис таблиць «PurchaseInvoices», «SalesInvoices»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
CurrencyId	TEXT	Ідентифікатор валюти
CurrencyRate	REAL	Курс валюти
TypePriceId	TEXT	Ідентифікатор типу ціни
WarehouseId		Ідентифікатор складу
TypeOperation	INTEGER	Тип операції
ProviderOrderId	TEXT	Ідентифікатор замовлення постачальнику
ClientOrderId	TEXT	Ідентифікатор замовлення покупця
Date	REAL	Дата документу
Conducted	INTEGER	Прапорець проведення документу
Number	TEXT(9)	Номер документу
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
DeleteMark	INTEGER	Прапорець помітки видалення документу

Таблиця 2.33

Опис таблиці «Purchases»

Поле	Тип	Призначення
Date	TEXT	Дата
DocumentId	TEXT	Ідентифікатор документу
NomenclatureId	TEXT	Ідентифікатор номенклатури
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
Quantity	REAL	Кількість
Price	TEXT	Ціна

Таблиця 2.34

Опис таблиці «Sales»

Поле	Тип	Призначення
Date	TEXT	Дата
DocumentId	TEXT	Ідентифікатор документу
NomenclatureId	TEXT	Ідентифікатор номенклатури
ClientId	TEXT	Ідентифікатор контрагента
OrganizationId	TEXT	Ідентифікатор організації
Quantity	REAL	Кількість
Price	TEXT	Ціна

Таблиця 2.35

Опис таблиці «Units»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
Name	TEXT	Найменування
Code	TEXT(9)	Код
IsGroup	INTEGER	Прапорець групи
Coefficient	REAL	Коефіцієнт одиниці виміру
DeleteMark	INTEGER	Прапорець помітки видалення
IsDefault	INTEGER	Прапорець елемента по-замовчуванню

Таблиця 2.36

Опис таблиці «Users»

Поле	Тип	Призначення
Id	TEXT	Ідентифікатор
Name	TEXT	Найменування
Code	TEXT(9)	Код
IsGroup	INTEGER	Прапорець групи
Coefficient	REAL	Коефіцієнт одиниці виміру
DeleteMark	INTEGER	Прапорець помітки видалення
IsDefault	INTEGER	Прапорець елемента по-замовчуванню
Login	TEXT	Логін
Password	TEXT	Хеш паролю

Фізична ER-діаграма бази даних знаходиться в додатках (Додаток Б).

2.3 Проектування та реалізація алгоритмів роботи системи

Додаток являтиме собою .NET WPF додаток що має реалізувати всі вимоги поставлені до нього при цьому бути простим у використанні, для

						Арк.
					ІПЗ.КР.Б – 121 – 24 – ПЗ	48

забезпечення цього було розроблено діаграми взаємодії що описують алгоритм роботи:

- перший запуск та авторизація;
- взаємодія з довідниками;
- взаємодія з документами;
- взаємодія з регістрами відомостей;
- взаємодія зі звітами.

Алгоритм першого запуску та авторизації наведено на (рис. 2.6).

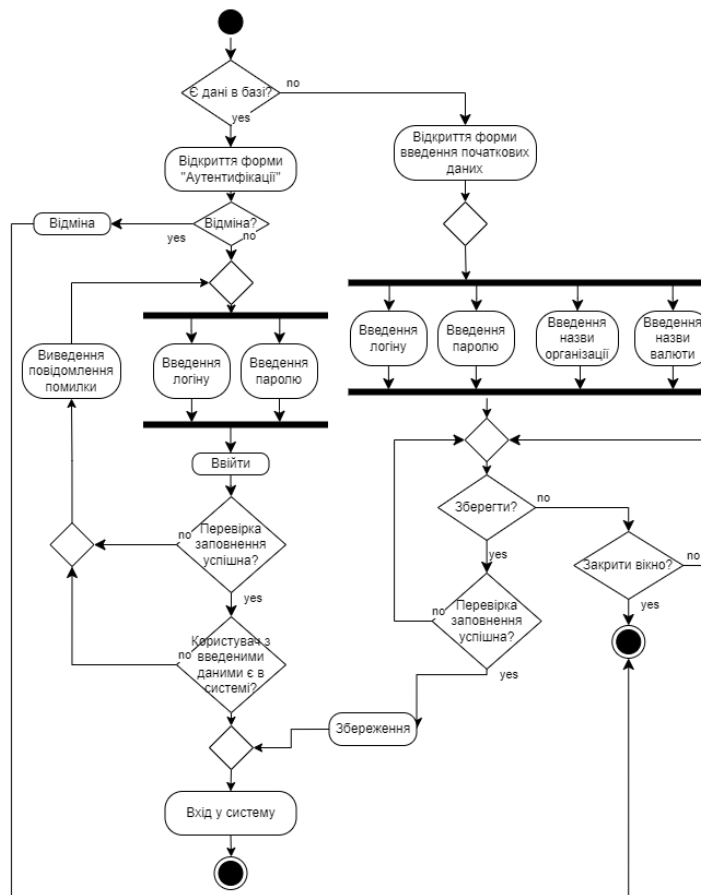


Рис. 2.6 – Діаграма взаємодії «Першого запуску та авторизації»

Детальний опис діаграми діяльності (див. рис. 2.6):

- **початкове заповнення даних:** при запуску додатку відбувається перевірка наявності даних в базі, якщо дані відсутні відкривається форма для початкового заповнення. На ній потрібно ввести наступні дані «Логін», «Пароль», «Назва організації» та «Назва валюти». Після чого натиснути «Зберегти», перед збереженням

відбудеться перевірка на наповненість і в разі успішності дані будуть збережені до бази даних і відкриється основна форма додатку. Якщо ж дані заповнені не всі то користувачу видається повідомлення про помилку заповнення даних;

- **авторизація:** при запуску відбувається перевірка наявності даних в базі, якщо дані присутні відкривається форма авторизації, на якій потрібно ввести «Логін» та «Пароль» та натиснути кнопку вхід. Після перевірки заповнення якщо вона не успішна користувачу видається повідомлення про помилку заповнення. В іншому випадку відбувається пошук за введеними даними користувача, якщо користувача не знайдено користувачу показується повідомлення про невірність введених даних. Якщо користувача знайдено відкривається основна форма додатку.

Алгоритм взаємодії із довідниками наведено в додатках (Додаток В), для опису алгоритму був використаний довідник «Номенклатура» так як він охоплює всі можливі варіанти дій над довідниками. Тому дана діаграма буде дійсна і для інших довідників, зміниться лише набір полів доступних для заповнення та деякі функції. Детальний опис діаграми:

- **взаємодія з елементами:** користувачу надається 4 дії над елементом довідника:
 - **додавання:** користувачу потрібно натиснути кнопку «Створити» після чого відкриється пуста форма елемента. Користувач може заповнити поля елемента. Після чого натиснути кнопку «Записати» відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому випадку користувачу виведеться повідомлення із переліком не заповнених полів. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді

відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;

- **редагування:** користувачу потрібно здійснити подвійний клік по потрібному елементу що відкриє форму елементу із заповненими даними, далі користувач може редагувати поля елементу. Після чого натиснути «Записати» відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому випадку користувачу виведеться повідомлення із переліком не заповнених полів. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;
- **копіювання:** користувачу потрібно здійснити одинарний клік по потрібному елементу що призведе до його виділення та натиснути кнопку «Копіювати». Відкриється форма елементу із заповненими даними, далі користувач може редагувати поля елементу. Після чого натиснути «Записати» відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому випадку користувачу виведеться повідомлення із переліком не заповнених полів. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;

- **встановлення помітки видалення:** користувачу потрібно здійснити одинарний клік по потрібному елементу що призведе до його виділення та натиснути кнопку «Видалити». Користувачу буде поставлено питання чи потрібно «Помітити елемент на видалення?» з варіантами «Так» та «Ні». У випадку позитивної відповіді для елементу встановлюється помітка на видалення, в іншому випадку ніякої дії не відбувається. Якщо елемент уже помічений на видалення буде поставлено питання чи потрібно зняти помітку, при позитивній відповіді помітка знімається.
- **взаємодія з групами:** користувачу надається 2 дії над групами довідників:
 - **додавання:** користувачу потрібно натиснути кнопку «Група» після чого відкриється порожня форма групи де користувач може заповнити поля групи після чого натиснути кнопку «Записати» відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому випадку користувачу виведеться повідомлення із переліком не заповнених полів. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;
 - **редагування:** користувачу потрібно здійснити подвійний клік по потрібній групі в списку груп, після чого відкриється форма групи де користувач може змінити поля групи після чого натиснути кнопку «Записати» відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому

випадку користувачу виведеться повідомлення із переліком не заповнених полів. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних.

Алгоритм взаємодії із документами наведено в додатках (Додаток Г), для опису алгоритму був використаний документ «Замовлення покупця» так як він охоплює всі можливі варіанти дій над документами. Тому дана діаграма буде дійсна і для інших документів, зміниться лише набір полів доступних для заповнення та деякі функції. Детальний опис діаграми:

- **додавання:** користувачу потрібно натиснути кнопку «Створити» після чого відкриється пуста форма документу. Користувач може заповнити поля документу та списки товарів при наявності. Після чого натиснути кнопку «Записати» або «Провести» відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому випадку користувачу виведеться повідомлення із переліком не заповнених полів. У випадку проведення для документу додатково відбувається перевірка проведення, при якій перевіряються його рухи в негативному результаті якої документ не буде записано і буде виведено повідомлення про помилку проведення, в позитивному для документу буде встановлено прапорець проведення. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;
- **редагування:** користувачу потрібно здійснити подвійний клік по потрібному документу що відкриє форму документу із заповненими

даними, далі користувач може редагувати поля елемента та списки товарів при наявності. Після чого натиснути «Записати», «Провести» або «Відмінити проведення» якщо документ проведений на момент відкриття, відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому випадку користувачу виведеться повідомлення із переліком не заповнених полів. У випадку проведення для документу додатково відбувається перевірка проведення, при якій перевіряються його рухи в негативному результаті якої документ не буде записано і буде виведено повідомлення про помилку проведення, в позитивному для документу буде встановлено прапорець проведення. Якщо відміна проведення рухи документа будуть видалені, а прапорець проведення знятий. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» або «Провести» якщо в документу проставлено прапорець проведення при негативному форма закривається без збереження даних;

- **копіювання:** користувачу потрібно здійснити одинарний клік по потрібному документу що призведе до його виділення та натиснути кнопку «Копіювати». Відкриється форма документу із заповненими даними. Користувач може заповнити поля документу та списки товарів при наявності. Після чого натиснути кнопку «Записати» або «Провести» відбудеться перевірка наповненості обов'язкових полів в результаті якої якщо усе заповнено елемент буде записано до бази даних в іншому випадку користувачу виведеться повідомлення із переліком не заповнених полів. У випадку проведення для документу додатково відбувається перевірка проведення, при якій перевіряються його рухи в негативному результаті якої документ не

буде записано і буде виведено повідомлення про помилку проведення, в позитивному для документу буде встановлено прапорець проведення. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;

- **встановлення помітки видалення:** користувачу потрібно здійснити одинарний клік по потрібному документу що призведе до його виділення та натиснути кнопку «Видалити». Користувачу буде поставлено питання чи потрібно «Помітити елемент на видалення?» з варіантами «Так» та «Ні». У випадку позитивної відповіді для документу встановлюється помітка на видалення, якщо документ має помітку проведення відбувається відміна проведення. При негативній відповіді ніякої дії не відбувається. Якщо елемент уже помічений на видалення буде поставлено питання чи потрібно зняти помітку, при позитивній відповіді помітка знімається;
- **заповнення на основі:** користувачу потрібно здійснити одинарний клік по потрібному документу що призведе до його виділення та натиснути на кнопку меню «Заповнити на основі» та обрати документи який потрібно заповнити на основі вибраного. Після чого відкриється форма нового документу що було вибрано із заповненими даними із документа основи.

Алгоритм взаємодії із регістрами відомостей наведено в додатках (Додаток Д), для опису алгоритму був використаний документ «Штрих коди». Дана діаграма буде дійсна і для інших регістрів відомостей, зміниться лише набір полів доступних для заповнення. Детальний опис діаграми:

- **додавання:** користувачу потрібно натиснути кнопку «Створити» після чого відкриється пуста форма регістра відомостей. Користувач може заповнити поля регістру. Після чого натиснути

кнопку «Записати» відбудеться перевірка наповненості обов'язкових полів при негативному результаті користувачу виведеться повідомлення із переліком не заповнених полів в позитивному результаті то відбувається перевірка унікальності запису якщо результат позитивний елемент буде записано до бази даних в іншому випадку користувачу буде видано повідомлення про те що запис із подібними полями існує. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;

- **редагування:** користувачу потрібно здійснити подвійний клік по потрібному запису що відкриє форму реєстра відомостей із заповненими даними, далі користувач може редагувати поля реєстру. Після чого натиснути кнопку «Записати» відбудеться перевірка наповненості обов'язкових полів при негативному результаті користувачу виведеться повідомлення із переліком не заповнених полів в позитивному результаті то відбувається перевірка унікальності запису якщо результат позитивний елемент буде записано до бази даних в іншому випадку користувачу буде видано повідомлення про те що запис із подібними полями існує. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;
- **копіювання:** користувачу потрібно здійснити одинарний клік по потрібному запису що призведе до його виділення та натиснути кнопку «Копіювати». Відкриється форма реєстру відомостей із

заповненими даними. Користувач може заповнити поля реєстру. Після чого натиснути кнопку «Записати» відбудеться перевірка наповненості обов’язкових полів при негативному результаті користувачу виведеться повідомлення із переліком не заповнених полів в позитивному результаті то відбувається перевірка унікальності запису якщо результат позитивний елемент буде записано до бази даних в іншому випадку користувачу буде видано повідомлення про те що запис із подібними полями існує. Або ж не зберігаючи зміни закрити форму, перед закриттям йому буде поставлено питання з варіантами «Так» та «Ні» чи зберегти зміни, при позитивній відповіді відбуваються дії аналогічні натисненню кнопки «Записати» при негативному форма закривається без збереження даних;

- **видалення:** користувачу потрібно здійснити одинарний клік по потрібному запису що призведе до його виділення та натиснути кнопку «Видалити», йому буде поставлено питання з варіантами «Так» та «Ні» чи видалити запис, при позитивній відповіді запис видаляється при негативному нічого не відбувається.

Алгоритм взаємодії зі звітами наведено на (рис. 2.7). Для побудови діаграми використовувався звіт «Продажі». Дана діаграма буде дійсна для інших звітів, зміниться лише набір полів відбору доступних для заповнення.

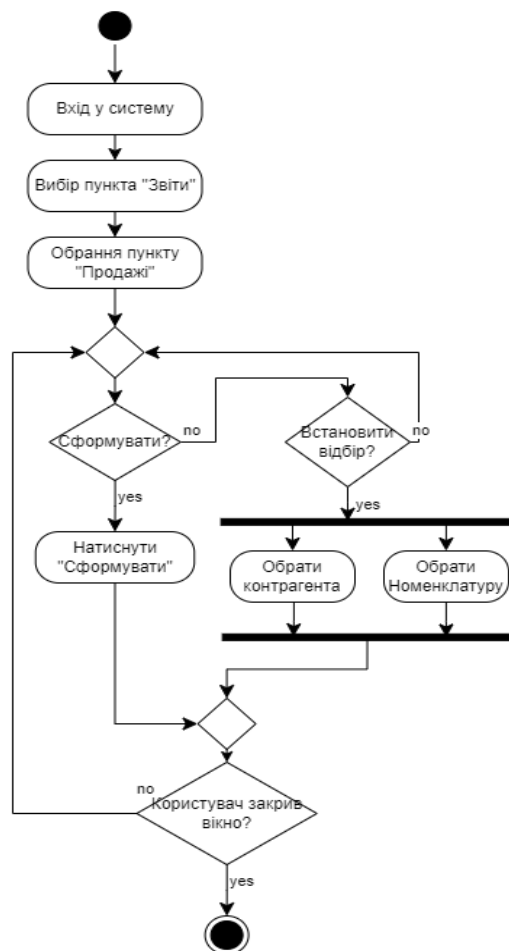


Рис. 2.7 – діаграма взаємодії «Робота зі звітами»

Детальний опис діаграми діяльності (див. рис. 2.7):

- **формування звіту:** користувачу потрібно натиснути кнопку «Сформувати» для отримання звіту. Також йому надається можливість встановити відбір заповнивши доступні поля.

2.4 Реалізація функціоналу системи автоматизації операційного обліку та бухгалтерської звітності

Для реалізації додатку було використано ORM EF Core що надає високий рівень абстрагування для роботи над базою даних, що є основними для облікових систем. Також надає можливість взаємодії із БД за допомогою Linq – запитів без необхідності писати SQL – запити [34].

В системі реалізовано контролери для роботи із довідниками, документами, регістрами відомостей, регістрами накопичення та користувачами, усі контролери окрім контролера для роботи з користувачами

реалізують CRUD методи. В даних контролерах описана основна логіка для операцій над об'єктами та базою даних. Щоб мати можливість реалізувати дані контролери без потреби явно вказувати до якої таблиці бази даних звертатися, таблиця бази даних описується властивістю контексту бази даних, було створено додаткову функцію контексту яка за допомогою механізму відображень в .NET буде по переданому типу повертати потрібну властивість контексту. Система відображень надає можливість визначати типи під час роботи додатку та створювати екземпляри даних типів та отримати доступ до його їх полів та властивостей. Саме доступ до полів та властивостей використано для реалізації функції для отримання властивості контексту по типу [35]:

```

1 public DbSet<T> GetPropertyData<T>() where T : class
2 {
3     PropertyInfo? property = GetType().GetProperties().FirstOrDefault(x =>
x.PropertyType == typeof(DbSet<T>));
4
5     if (property == null)
6         throw new Exception("Property not found");
7
8     object? propValue = property.GetValue(this);
9
10    if (propValue == null)
11        throw new Exception("Property value is null");
12
13    DbSet<T> data = (DbSet<T>)propValue;
14
15    return data;
16 }

```

Функція являється шаблонною для того щоб її можна було викликати для будь-якого типу, якщо він являється класом. Отримання властивості класу виконується в рядку 3, для поточного об'єкту викликається функція GetType(), так як виклик відбувається в середині класу вказівник this при виклику було пропущено, в результаті буде отримано об'єкт типу Type, даний об'єкт містить дані про тип об'єкта класу. Далі для Type викликається метод GetProperties() що повертає колекцію властивостей класу, для даної колекції викликається метод розширення FirstOrDefault(x => x.PropertyType == typeof(DbSet<T>)) завдяки якому виконує отримання властивості яка буде відповідати переданому

відбору, якщо ж ні один елемент не відповідає відбору буде повернуто null. Якщо поле було знайдено для нього в рядку 8 викликається функція для отримання значення що зберігається в даному полі, в якості параметру передається об'єкт з якого потрібно отримати значення отриманої властивості. Після чого отримане значення приводиться до типу колекції шаблону і повертається.

Наведена функція дає можливість не залежати від назв властивостей, що відображають назви таблиць в базі даних, і лише по переданому типу отримувати потрібні таблиці для подальшої роботи з ними.

Наступна функція дозволяє завантажувати по зв'язкам пов'язані об'єкти що знаходяться в сутності, майже в кожній сутності для зв'язків описано два поля, поле ідентифікатор та поле класу в яке при необхідності буде завантажено пов'язаний об'єкт. Дані поля помічені модифікатором virtual, EF Core може використовувати даний модифікатор для реалізації lazy loading для колекцій що позначені даним модифікатором [36]. В даному випадку модифікатор virtual використовується для полів класу та служить для визначення чи потрібно робити завантаження даних для даного поля. Для негайного завантаження даних використовується функція Include(), дана функція приймає назву поля дані якого потрібно завантажити негайно. Функція працює за принципом лівого з'єднання в SQL, при даному з'єднанні об'єкт який знаходиться зліва буде в будь-якому випадку вивантажений навіть якщо не по зв'язку не буде завантажено даних [37].

```

1 public IQueryable<T> IncludeVirtualProperty<T>(IQueryable<T> data) where T : class
2 {
3     var virtualProperty = typeof(T).GetProperties().Where(p => p.PropertyType.IsClass
4         && p.GetGetMethod().IsVirtual && p.PropertyType != typeof(string));
5
6     var copy = data;
7     foreach (var item in virtualProperty)
8         copy = copy.Include(item.Name);
9
10    virtualProperty = typeof(T).GetProperties().Where(
11        p => p.PropertyType.IsGenericType
12        && p.PropertyType.GetGenericTypeDefinition() == typeof(ICollection<>)
13        && p.GetGetMethod().IsVirtual);
14

```

						Арк.
					ІПЗ.КР.Б – 121 – 24 – ПЗ	60

```

15  foreach (var item in virtualProperty)
16  {
17      copy = copy.Include(item.Name);
18
19      var genericType = item.PropertyType.GetGenericArguments().First();
20
21      var propertyGeneric = genericType.GetProperties().Where(
22          p => p.PropertyType.IsClass
23          && p.GetGetMethod().IsVirtual
24          && p.PropertyType != typeof(string)
25          && p.PropertyType != item.DeclaringType);
26
27      foreach(var prop in propertyGeneric)
28      {
29          copy = copy.Include($"{item.Name}.{prop.Name}");
30      }
31  }
32
33  ChangeTracker.Clear();
34
35  return copy.AsNoTracking();
36 }

```

Функція являється шаблонною для того щоб її можна було викликати для будь-якого типу, якщо він являється класом в якості параметру передається колекція для формування запиту об'єктів в яких потрібно зробити завантаження пов'язаних даних. В рядках 3 – 4 виконується отримання типу шаблону функції і для нього виконується пошук віртуальних властивостей. В результаті буде отримано колекцію даних властивостей, по якій потрібно здійснити обхід та для кожної властивості викликати функцію Include(), передаючи ім'я поля. На рядках 11 – 13 виконується отримання пов'язаних колекцій, дана дія актуальна для документів які містять списки товарів, після чого обходимо отриману колекцію і виконуємо аналогічні дії що і для полів. Так як списки також містять зв'язки для них теж потрібно виконати завантаження даних. На рядку 19 відбувається отримання типу колекції списку товарі, адже колекція являється шаблонною. Після отримання типу по аналогії із полями виконується включення до запиту. В результаті виконання функції буде сформований запит що включатиме в собі всі пов'язані дані [38].

HandbookController (контролер для роботи з довідниками). Контролер реалізує інтерфейс «IHandbookController» тому містить в собі функції описані

даним інтерфейсом та реалізує їхню логіку для роботи із даними довідників. Він використовується для отримання списків довідників для відображення їх в інтерфейсі, отримання даних існуючих елементів, створення нових та видалення елементів. Контролер також містить наступні функції:

- `GetNextCode<T>(DbSet<T> data)` – виконує генерацію коду для нового елементу. Відбувається отримання всіх наявних кодів в таблиці та їх сортування в порядку спадання, в такому випадку найбільший код буде першим в списку. Після чого найбільший код збільшується на 1 та повертається.

DocumentController (контролер для роботи з документами). Даний контролер реалізує інтерфейс «IDocumentController» що зобов'язує його реалізувати функції описані даним інтерфейсом. В реалізованих функція описується логіка поведінки для роботи із об'єктами документів. Контролер використовується для отримання даних документів для їхнього відображення в інтерфейсі. Також для створення нових документів та внесення змін до уже існуючих документів та видалення документів із бази даних. Документи мають додаткові варіанти запису це «Проведення» та «Відміна проведення». При проведенні формуються рухи документа і зберігаються, адже тільки проведений документ впливає на облік. При відміні проведення рухи документу видаляються [39].

При проведенні відбувається отримання рухів документа, якщо вони є. Кожен документ реалізує функцію що повертає його рухи у вигляді словника в якому ключ це тип реєстра накопичення в який потрібно виконати додавання даних, а значення це масив даних які потрібно зберегти. Після отримання рухів документа по ключам словника виконується пошук уже існуючих рухів документа щоб врахувати їх значення при перевірках, наприклад при повторному проведенні для залишків потрібно враховувати товари які уже списані документом і врахувати їх як доступними до списання. Наведено фрагмент функції проведення в якій відбувається отримання попередніх рухів документу.

						Арк.
					ІПЗ.КР.Б – 121 – 24 – ПЗ	62

```

1 Dictionary<Type, IEnumerable<IAccumulationRegister>> oldMoves = new();
2 if(document.Id != Guid.Empty)
3 {
4     Func<IAccumulationRegister, bool> func = k => k.DocumentId == document.Id;
5     foreach(var move in moves)
6     {
7         var oldMove = (IEnumerable<IAccumulationRegister>)_accumulationController
8             .GetType()
9             .GetMethod("GetListData")
10            .MakeGenericMethod(move.Key)
11            .Invoke(_accumulationController, [func]);
12
13         if(oldMove != null && oldMove.Count() != 0)
14             oldMoves.Add(move.Key, oldMove);
15     }
16 }

```

Для отримання попередніх рухів документу використовується механізм відображень, так як невідомо які типи знаходяться в словнику рухів. На рядку 4 визначається делегат який відповідає за передання умов відбору по яким з регістрів накопичення потрібно отримати дані. Далі відбувається обхід словника рухів для отримання значення ключа, який являється типом регістру накопичення. На рядках 7 – 9 відбувається отримання методу «GetListData» що знаходиться в контролері регістрів накопичення. Так як даний метод являється шаблонним йому потрібно задати тип для якого він буде викликатися, на рядку 10 для отриманого методу викликається функція «MakeGenericMethod(move.Key)», в якості параметру передається значення типу регістра накопичення із ключа словника. На рядку 11 для методу викликається функція «Invoke(_accumulationController, [func])» дана функція виконує виклик отриманого методу для контролера регістрів накопичення що заданий параметром «_accumulationController», в якості наступного параметру передається делегат відбору який буде підставлено у відповідний параметр функції «GetListData». Після виконання отриманий результат на рядку 12 приводиться до типу «IEnumerable<IAccumulationRegister>», по отриманій колекції робить обхід циклом і додавання в новий словник за типом регістра накопичення якому належать дані рухи.

Після того як було отримано нові та попередні рухи документа відбувається обхід нових рухів в циклі та перевірка чи після їх запису стан

даних в базі залишиться коректним, наприклад залишки товару після запису не стануть від’ємними. Якщо перевірки пройшли успішно відбувається видалення попередніх рухів, за їх наявності, встановлення прапорця проведення для документу і запис документа та нових рухів. Лістинг коду функції проведення наведено в додатках (Додаток Е).

При відміні проведення відбувається видалення поточних рухів документу, їхнє отримання аналогічне отриманню попередніх рухів при проведенні. Після отримання попередніх рухів відбувається їхнє видалення, зняття прапорця проведення документу та запис документа. Функція відміни проведення наведена в додатках (Додаток Є).

Контролер також містить наступні функції:

- `GetNextNumber<T>(DbSet<T> data)` – виконує генерацію номеру для нового документу. Відбувається отримання всіх наявних номерів в таблиці та їх сортування в порядку спадання, в такому випадку найбільший номер буде першим в списку. Після чого найбільший номер збільшується на 1 та повертається.

AccumulationRegisterController (контролер реєстрів накопичення) реалізовує інтерфейс «`IAccumulationRegisterController`» що зобов’язує його реалізувати функції описані даним інтерфейсом. В реалізованих функція описується логіка поведінки для роботи із об’єктами реєстрів накопичення, а саме додавання, оновлення, отримання та видалення даних із БД. Даний контролер використовується для отримання, фіксування або видалення рухів документів, також отримання даних для звітів. При формуванні звітів дані отримуються з відповідного реєстра накопичення або декількох та відображаються в певному вигляді для користувача. Для реєстрів накопичення що реалізують інтерфейс «`ILeftoverRegister`» присутня функція яка робить групування даних для отримання актуальних значень залишків за переданими параметрами і в результаті новий список об’єктів які були задані делегатом вибору.

```

public List<V> GetLeftoverList<T, G, V>(List<T> leftovers, Func<T, G> groupBy,
Func<IGrouping<G, T>, V> select) where T : class, ILeftoverRegister
{
    return leftovers.GroupBy(groupBy).Select(select).ToList();
}

```

InformationRegisterController (Контролер реєстрів відомостей) даний контролер реалізує інтерфейс «IInformationRegisterController» що зобов'язує його реалізувати функції для роботи із об'єктами реєстрів відомостей. Даний контролер використовується для виконання CRUD операцій над даними. Для реєстрів що реалізують інтерфейс «IPeriodDateRegister» що зобов'язує дані реєстри мати поле дати, доступна функція що виконує отримання актуальних даних на дату. При її виконанні отримується відсортований за спаданням список даних реєстру дата яких менша або рівна переданій, якщо дата не передана то встановлюється поточна, із отриманого списку формується новий список із значеннями, значення в списку не можуть повторюватися. Так як список відсортовано то в результаті буде отримано список даних реєстра які були актуальні на передану дату.

```

public List<T>? GetLastDateData<T>(DateTime date = default, Func<T, bool>?
selectionFunc = default) where T : class, IPeriodDateRegister
{
    IQueryable<T> data = _context.GetPropertyData<T>();

    if (date == default)
        date = DateTime.Now;

    data = _context.IncludeVirtualProperty(data);

    IEnumerable<T> result = data.AsNoTracking();

    if (selectionFunc != null)
        result = result.Where(selectionFunc);

    result = result.OrderByDescending(x => x.Date).Where(x => x.Date <= date);

    var listData = new List<T>();
    var dict = new Dictionary<T, bool>();

    foreach(var item in result)
    {
        if (!dict.TryGetValue(item, out bool value))
        {
            dict.Add(item, true);
            listData.Add(item);
        }
    }
}

```

```

    }
}

return listData;
}

```

UserController (контролер для роботи з користувачами) даний контролер використовується для авторизації користувача при запуску системи. Авторизація в якості параметрів приймає логін і пароль, пароль хешується, та виконує пошук користувача в базі даних і якщо знаходить повертає значення true, в іншому випадку false.

```

public async Task<bool> Authenticate(string login, string password)
{
    PasswordHashing hashing = new PasswordHashing();
    var passwordHash = await hashing.HashAsync(password);
    var user = await _context.Users.FirstOrDefaultAsync(u => u.Login.ToLower() ==
login.ToLower() && u.Password == passwordHash);
    return user != null;
}

```

Дані контролери реалізують основну логіку всього додатку, шар інтерфейсу взаємодії із базою даних саме через них.

PrivatBankExchangeRateService сторонній сервіс для отримання курсів валют по таймеру. Даний сервіс реалізує інтерфейс «IExchangeRateService» лістинг коду наведено в додатках (Додаток Ж). Даний сервіс використовує API Private bank щоб отримати дані про валюти. При запуску головної форми використовуючи об'єкт класу BackgroundWorker [40] запускається асинхронно функція у якій відбувається виклик сервісу отримання курсів валют та оновлення їхніх даних раз у заданий інтервал. Результатом виклику сервісу являється список даних курсів валют, при обробці вибираються лише ті що наявні в базі і найменування яких співпадає. Якщо найменування співпадає в реєстрі відомостей створюється новий запис на дату оновлення курсу із новим значенням курсу валюти.

```

private async void Bg_DoWork(object sender, DoWorkEventArgs e)
{
    while(true)
    {
        Thread.Sleep(TimeSpan.FromMinutes(TIMER));

        var result = await _exchangeRateService.GetExchangeRatesAsync();
    }
}

```

```

        var currencies = _handboolContorller.GetHandbooks<Currency>(w => !w.DeleteMark
        && !w.IsDefault);

        foreach (var item in currencies)
        {
            var data = result.FirstOrDefault(w => w.CurrencyName.ToUpper() ==
            item.Name.ToUpper());

            if (data != null)
            {
                var exchangeRate = new ExchangesRate();
                exchangeRate.Currency = item;
                exchangeRate.Date = DateTime.Now;
                exchangeRate.Rate = data.CurrengeSaleRate;
                await _informationRegisterController.AddOrUpdateAsync(exchangeRate);
            }
        }
    }
}

```

Висновки до другого розділу

В процесі проектування системи автоматизації оперативного обліку та бухгалтерської звітності було визначено та описано функціональні та не функціональні вимоги до додатку. Створено діаграму використання та розписано варіанти використання. Спроектовано структуру проекту та описано діаграму класів.

На основі сутностей додатку були створені таблиці бази даних та описано їхню структуру, побудовано фізичну ER-діаграму бази даних.

Побудовані та описані основні алгоритми взаємодії з «Довідниками», «Документами», «Регістрами відомостей» та «Звітами».

Реалізовано та наведено опис основного функціоналу додатку.

РОЗДІЛ 3. Інтерфейс та порядок роботи з системою автоматизації обробки даних для ведення оперативного та бухгалтерського обліку

3.1 Порядок встановлення та налаштування параметрів системи

Встановлення та налаштування програмного додатку вимагає наявності на комп'ютері набору бібліотек .NET SDK.

Для відображення компонентів що потрібні для розгортання системи було побудовано діаграму розгортання (рис. 3.1).

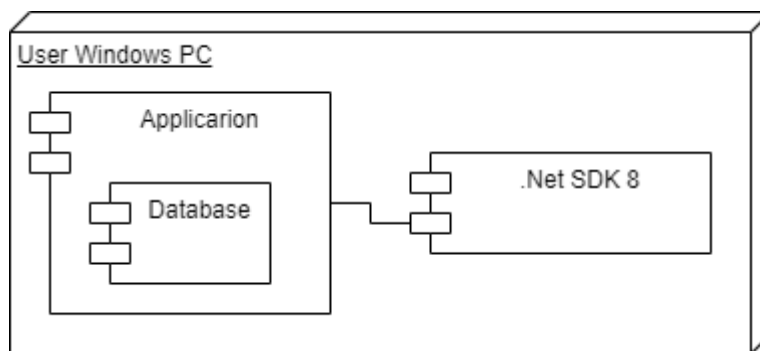


Рис. 3.1 – Діаграма розгортання

Опис процесу розгортання:

1. Встановити набір бібліотек та інструментів .NET SDK 8 що забезпечить роботу розроблюваного додатку [41];
2. Встановити розроблюваний додаток. Так як розроблюваний додаток в якості бази даних використовує SQLite це не зобов'язує встановлення додаткових провайдерів баз даних. База даних являється частиною додатку і компілюється разом із ним. При першому запуску за її відсутності вона буде створена.

3.2 Структура інтерфейсу системи автоматизації обробки даних для ведення оперативного та бухгалтерського обліку

Інтерфейс додатку було спроектовано та розроблено на платформі WPF з врахуванням що він буде використовуватися лише на комп'ютерах з операційною системою «Windows». Кожне вікно додатку має лише основні

елементи керування для взаємодії з даними. Дане рішення має допомогти користувачеві швидше зрозуміти можливості та розпочати працювати з системою.

При першому запуску додатку користувачу відкриється форма заповнення початкових даних (рис. 3.2). На даній формі користувач має ввести основні дані для подальшої авторизації в системі та дані про організацію та валюту які будуть вважатися по замовчуванню.

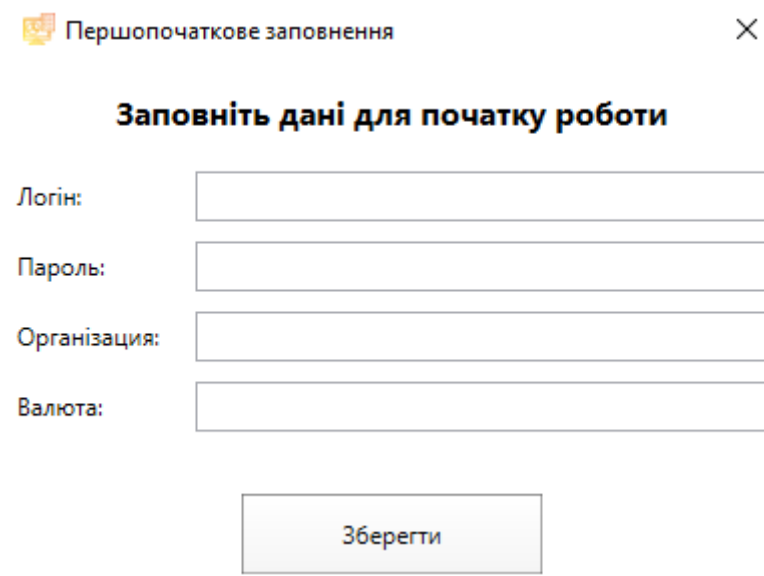


Рис. 3.2 – Форма заповнення початкових даних

При наступних запусках користувачу буде відкриватися форма авторизації (рис. 3.3) де потрібно ввести логін і пароль які він задав при першому запуску.

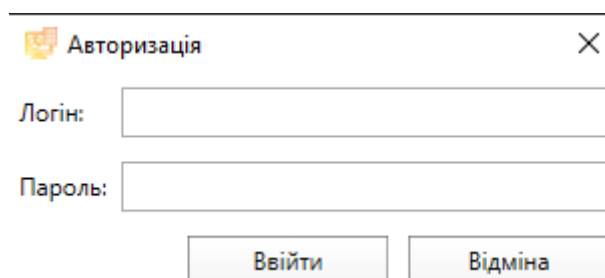


Рис. 3.4 – Форма авторизації

Якщо користувач введене невірні дані йому буде видано відповідне повідомлення (рис. 3.5) і в систему його не буде допущено.

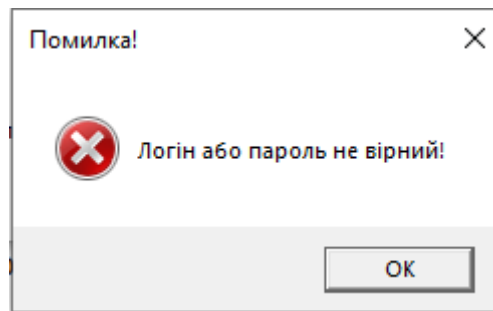


Рис. 3.5 – Повідомлення про невірність даних авторизації

Після введення коректних даних користувачу відкриється основне меню додатку (рис. 3.6). Меню розділено на дві частини, ліва містить кнопки із назвами типів об'єктів з якими можна взаємодіяти, права буде відображати елементи які відносяться до обраного меню.



Рис. 3.6 – Головне меню додатку

Обравши пункт меню «Довідники» користувачу буде доступний перелік об'єктів для взаємодії розділених на 3 колонки «Основні», «Облік» та «Розрахунки» (рис. 3.7). В колонці «Основні» знаходяться об'єкти які являються найрозповсюдженішими для взаємодії в системі. Колонка «Облік» містить об'єкти в розрізі яких ведеться основний оперативний облік, це не означає що по об'єктам із інших колонок не можна вести облік. Колонка «Розрахунки» містить в собі об'єкти в межах яких ведеться облік грошових коштів.

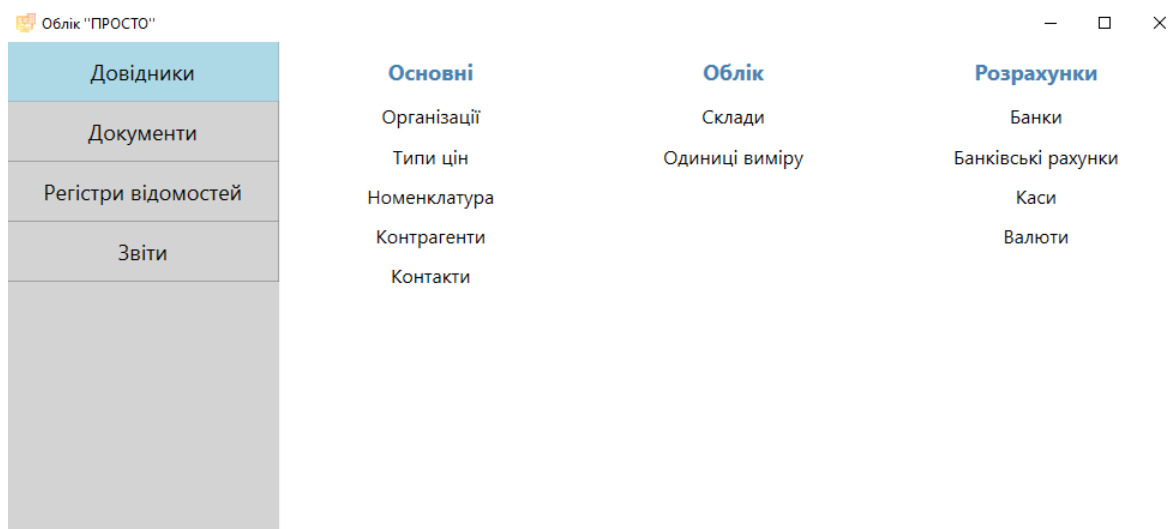


Рис. 3.7 – Пункт меню «Довідники»

Обравши будь-який пункт із колонок відкриється список обраного довідника. Списки можуть відрізнятися в залежності чи являється обраний довідник ієрархічним чи ні. Обравши довідник що не являється ієрархічним для користувача буде відображено наступний список (рис. 3.8).

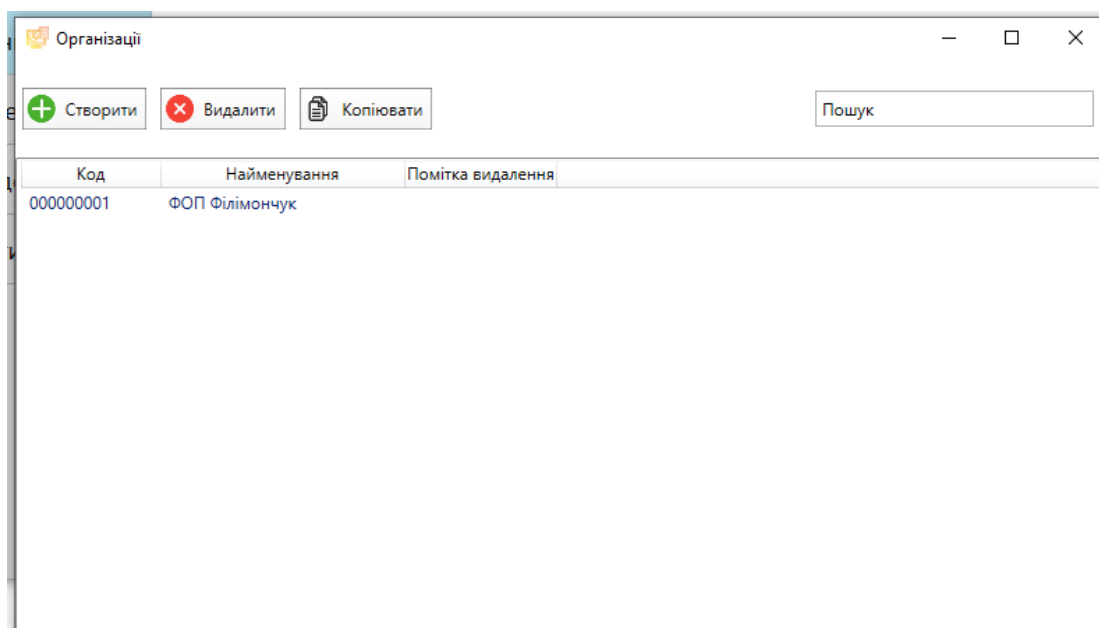


Рис. 3.8 – Список довідника «Організації»

На списком знаходиться панель з командами для взаємодії із елементами довідника. Під панеллю команд знаходиться сам список елементів довідника.

Натиснувши команду «Створити» відкриється форма нового елемента довідника (рис. 3.9).

Рис. 3.9 – Форма нового елемента «Організації»

На даній формі дається можливість заповнити найменування нової організації та зберегти її, код організації генерується системою тому він доступний тільки для читання (рис 3.10).

Рис. 3.10 – Форма записаного елемента «Організації»

Після запису новий елемент з'являється в списку довідників (рис 3.11).

Організації		
+ Створити × Видалити 📄 Копіювати Пошук		
Код	Найменування	Помітка видалення
000000001	ФОП Філімончук	
000000002	ФОП "Іванов Іван Іванович"	

Рис. 3.11 – Список довідника «Організації»

Використавши команду «Видалити» користувач може встановити для елемента помітку на видалення. При повторному використанні команди на елементі що позначений на видалення помітка видалення знімається і це відображається в колонці списку «Помітка видалення» (рис. 3.12). Перед встановленням чи зняттям помітки користувачу задається питання про виконання дії (рис. 3.13 – рис. 3.14).

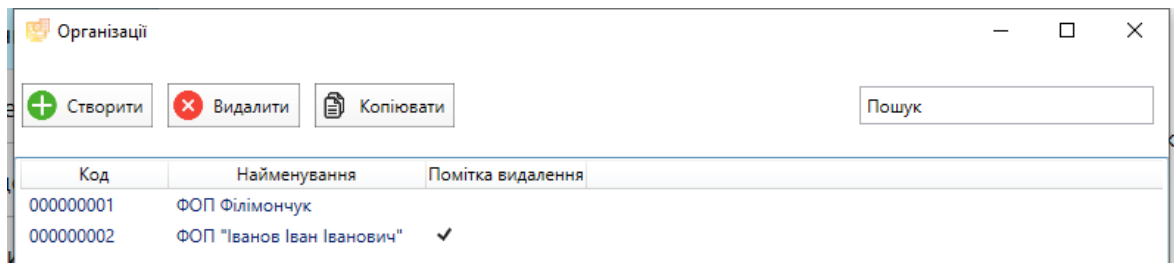


Рис. 3.12 – Список довідника «Організації» з поміченим на видалення елементом

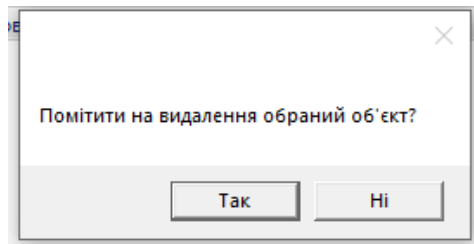


Рис. 3.13 – Діалог питання для встановлення помітки на видалення

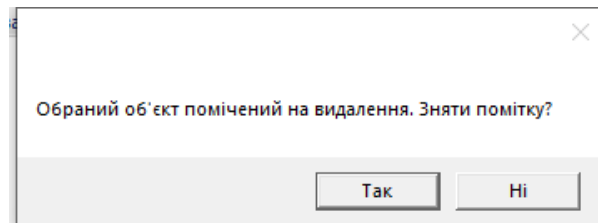


Рис. 3.14 – Діалог питання для зняття помітки на видалення

Обравши зі списку елемент і натиснувши команду «Копіювати» буде відкрита форма нового елемента але із заповненими даними (рис. 3.15) що відповідають даним елементу який копіювали.

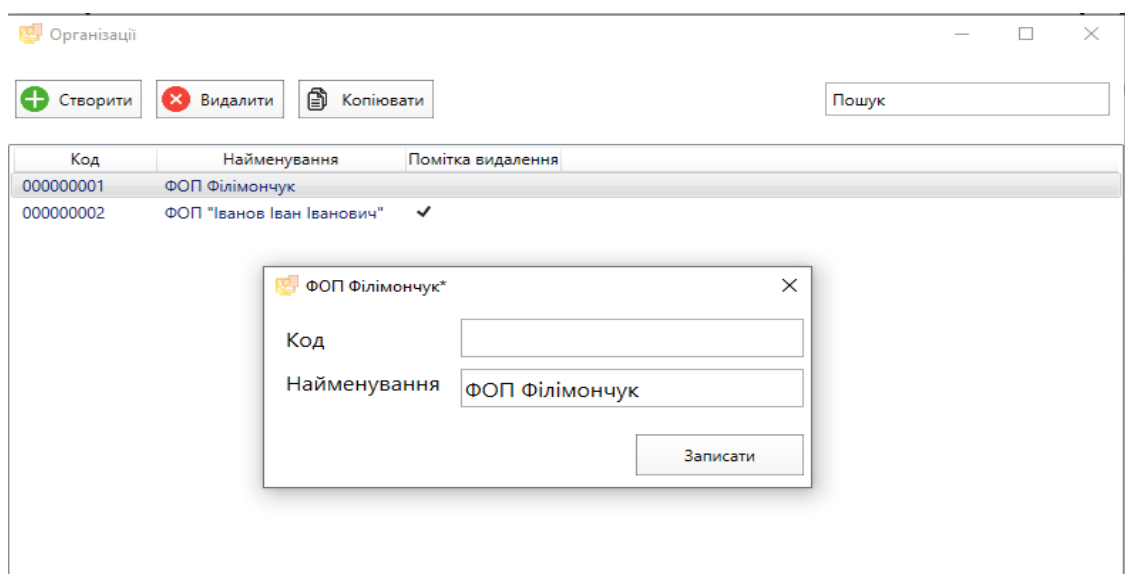


Рис. 3.15 – Форма скопійованого елемента

Користувач може здійснити пошук по назвам елементів ввівши відповідні символи в поле пошук (рис. 3.16).

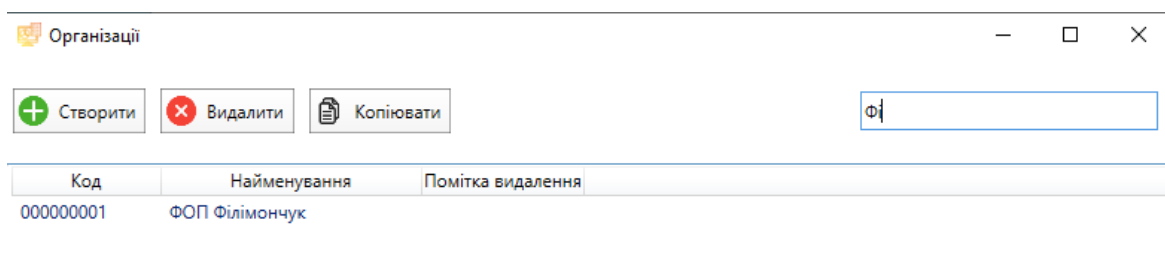


Рис. 3.16 – Список довідника «Організації» з відбором пошуку

Відкривши список довідника який являється ієрархічним (рис. 3.17) користувач побачить що окрім основного списку елементів з'явився додатковий список груп та в панелі команд з'явиться нова команда «Група» що дозволяє створювати групи. При активації елемента зі списку групи відбувається відбір елементів списку довідника, даним чином відображаються елементи які знаходяться в обрані групі.

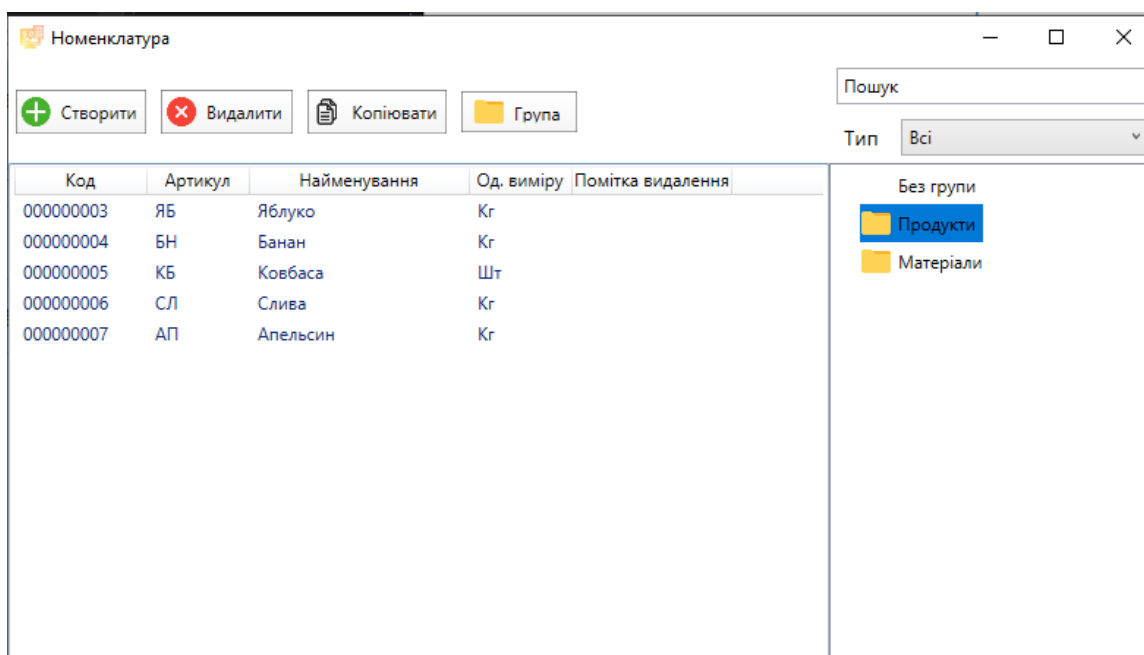


Рис. 3.17 – Список ієрархічного довідника «Номенклатура»

Натиснувши команду «Група» буде відкрито форму нової групи (рис. 3.18). На даній формі окрім найменування можна вибрати батьківську групу в якій буде знаходитися нова група або змінити в уже існуючій групі. При виборі групи відкривається список (рис. 3.19) існуючих груп з яких і виконується

вибір. Після запису новостворена група відображається в ієрархії груп (рис 3.20).

Рис. 3.18 – Форма групи довідника «Номенклатура»

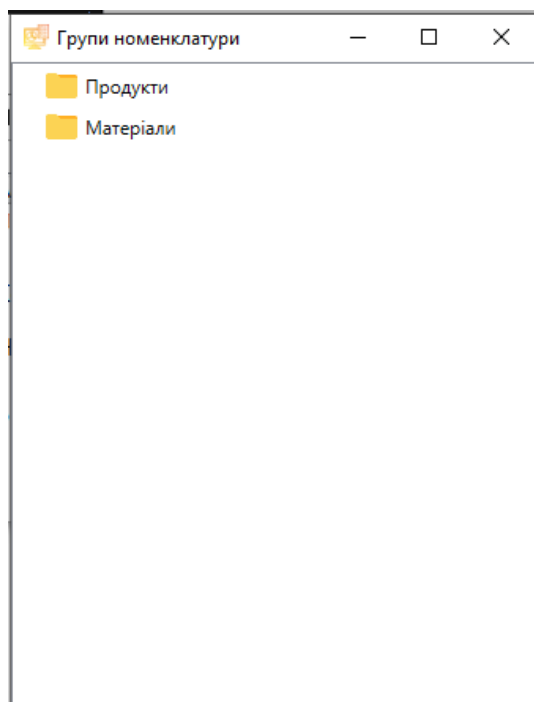


Рис. 3.19 – Список груп довідника «Номенклатура»

Рис. 3.20 – Список довідника «Номенклатура» відображення ієрархії груп

Для довідника номенклатури доступний відбір по типу номенклатури обираючи який в списку елементів буде відображено лише ті що відповідають даному типу, по замовчуванню відображаються всі елементи. Для вибору доступні 3 варіанти «Всі», «Товар» та «Послуга». При виборі пункту «Товари» в списку буде відображено лише елементи даним типом (рис. 3.21).

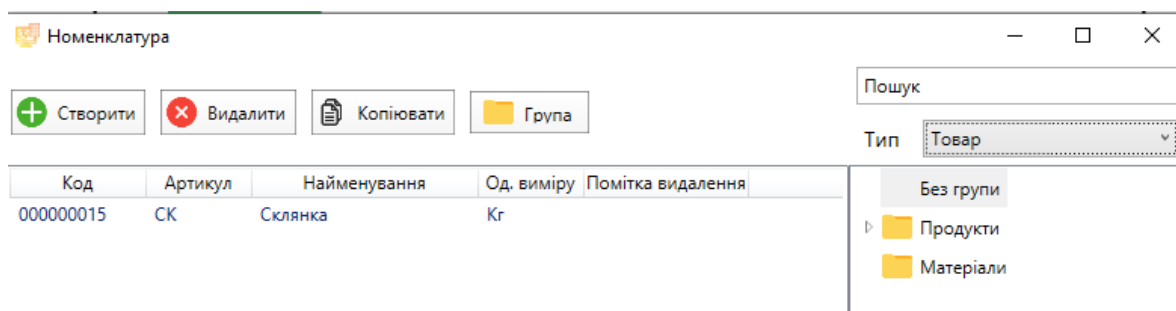


Рис. 3.21 – Список довідника фільтрування по типу елементу «Товар»
Обравши варіант «Послуги» фільтрація відбудеться відповідно (рис. 3.22).

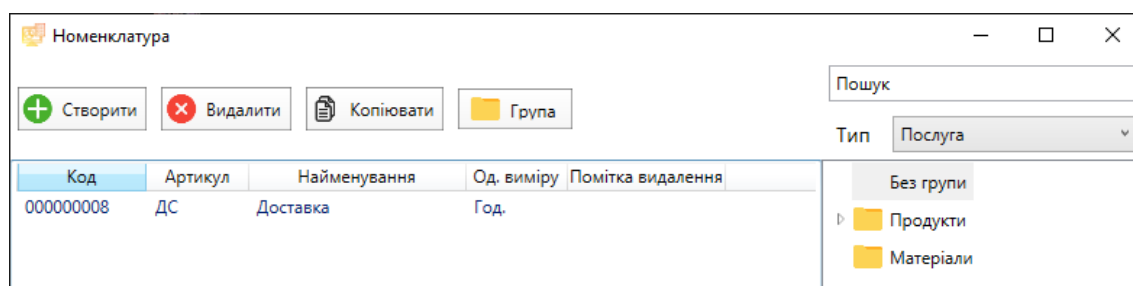


Рис. 3.22 – Список довідника фільтрування по типу елементу «Послуга»
Тип товару як і група задається на формі елементу номенклатури (рис. 3.23).

Код	000000008		
Найменування	Доставка		
Артикул	ДС		
Од. виміру	Год.	X	▼ ...
В групі		X	▼ ...
Тип	Послуга ▼		
Записати			

Рис. 3.23 – Форма елементу довідника «Номенклатура»

Інтерфейс взаємодії з іншими довідниками буде аналогічним враховуючи чи є довідник ієрархічним чи ні. Форми елементів відрізняються тільки кількістю полів доступних для заповнення.

Обравши пункт меню «Документи» для користувача буде відображено перелік доступних для взаємодії документів розділених на 3 колонки «Продажі», «Закупки» та «Розрахунки» (рис. 3.24). В колонці «Продажі» знаходяться документи через які здійснюються оформлення та фіксація продажів. В колонці «Закупки» знаходяться документи через які здійснюється оформлення та фіксація закупівель. В колонці «Розрахунки» знаходяться документи через які фіксуються оплати та витрати грошових коштів.

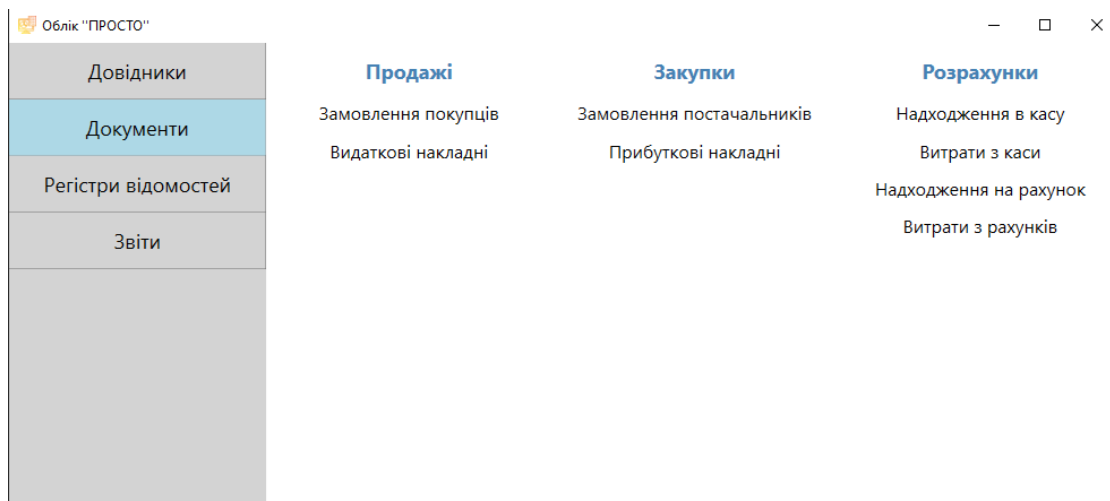


Рис. 3.24 – Пункт меню «Документи»

Обравши будь-який пункт із доступних документів буде відкрито список відповідного документу (рис. 3.25). Над списком знаходиться командна панель, в ній доступні команди «Створити», «Видалити», «Копіювати», меню «Заповнити на основі» та поле для пошуку. Меню «Заповнити на основі» відображається лише в списках документів «Продажі» та «Закупки» так як на основі даних документів можна створити документи розрахунків що являються кінцевими документами в процесі продажі або закупівлі. Для документів в колонці «Розрахунки» список ідентичний списку не ієрархічного довідника (див. рис. 3.11).

Видаткові накладні					
Створити Видалити Копіювати Заповнити на основі ▼ Пошук					
Дата	Номер	Операція	Контрагент	Сума документа	Проведений Помітка ви
15-06-2024	000000005	Продаж покупцю	Петро	100.0	✓
15-06-2024	000000004	Продаж покупцю	Петро	150.0	✓
15-06-2024	000000003	Продаж покупцю	Тарас	45.0	✓
15-06-2024	000000002	Продаж покупцю	Будівельники	3875.0	✓
15-06-2024	000000001	Продаж покупцю	Будівельники	3600.0	✓

Рис. 3.25 – Список документів «Видаткові накладні»

Створивши або відкривши документ відкривається форма документу. Форми документів можуть відрізнятися по наявності табличної частини із товарами (рис. 3.26) та її відсутністю (рис. 3.27).

Видаткова накладна

Записати

Провести

Відміна проведення

Дата

15 червня 2024 р. 18:27:41

Номер

000000005

Організація

ФОП Філімончук

X ▼ ...

Валюта

UAH

X ▼ ...

Курс

1

Контрагент

Петро

X ▼ ...

Тип ціни

Роздрібна

X ▼ ...

Склад

Основний

X ▼ ...

Тип операції

Продаж покупцю

Замовлення

Замовлення покупця 00

X ▼ ...

Додати

Видалити

Штрихкод

Номенклатура	Од. Виміру	Кількість	Ціна	Сумма
Цвяхи	Шт	100	1	100

Рис. 3.26 – Форма документу «Видаткові накладні»

Витрата з каси

Записати

Провести

Відміна проведення

Дата

15 червня 2024 р. 18:19:07

Номер

000000002

Організація

ФОП Філімончук

X ▼ ...

Валюта

UAH

X ▼ ...

Курс

1

Контрагент

Агрокомплекс

X ▼ ...

Каса

Основна

X ▼ ...

Тип операції

Від постачальника

Сума

450,0

Документ основа

Прибуткова накладна 000000001 ві

X ▼ ...

Рис. 3.27 – Форма документу «Витрата з каси»

При наявності табличної частини доступна команда для пошуку номенклатури по штрих коду (рис. 3.28).

Рис. 3.28 – Форма пошуку номенклатури за штрих кодом

Знайдений товар додається в список товарів із кількість 1 при наявності даного товару в списку його кількість збільшується на 1.

Натиснувши на команду «Провести» відбувається перевірка можливості проведення що описується рухами документа які він робить, якщо результат перевірки негативний користувачу виводиться повідомлення з описом причини неможливості провести документ (рис. 3.29).

Рис. 3.29 – Діалог повідомлення про помилку проведення

У разі успішного проведення для документу стає активною команда «Відміна проведення» (рис. 3.30).

Рис. 3.30 – Форма документу «Видатковій накладній» з доступною командою «Відміна проведення»

Натиснувши команду «Відміна проведення» рухи документа видаляються, а команда «Відміна проведення» знову стане недоступною (див. рис. 3.26).

Для вибору значення реквізиту документу якщо він являється довідником або іншим документом потрібно в полі вводу натиснути кнопку зі знаком «▼» після чого відкриється список (див. рис. 3.11) якщо це не ієрархічний довідник, якщо це ієрархічний (див. рис. 3.17).

Обравши пункт меню «Регістри відомостей» для користувача буде відображено перелік доступних для взаємодії реєстрів відомостей розділених на 2 колонки «Дані товарів» та «Дані клієнтів» (рис. 3.31). В колонці «Дані товарів» знаходяться реєстри в яких зберігається пов'язана інформація із товарами. В колонці «Дані клієнтів» знаходяться реєстри в яких знаходяться дані пов'язані із контрагентами.

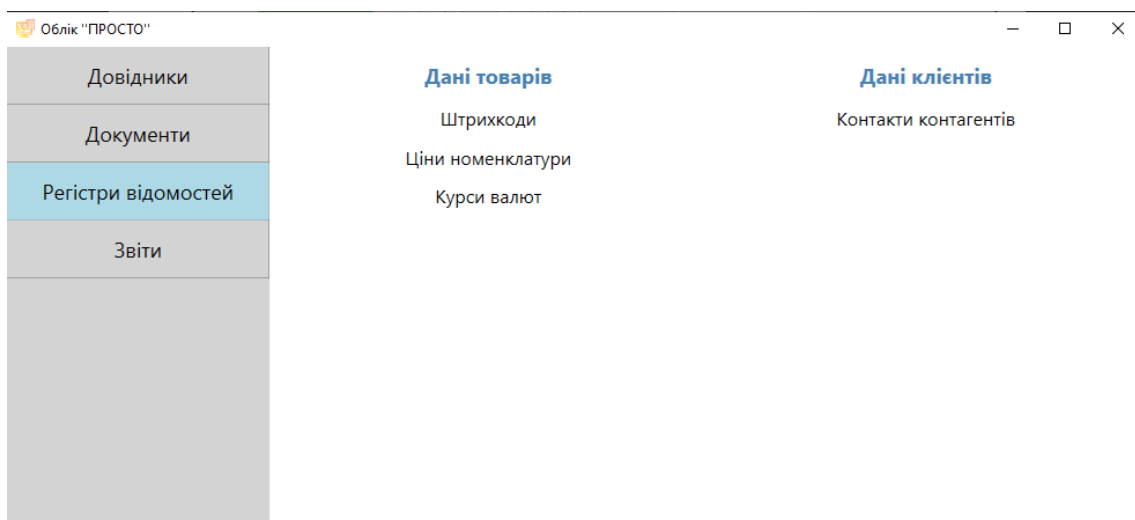


Рис. 3.31 – Пункт меню «Регістри відомостей»

Обравши будь-який пункт із доступних реєстрів відкриється список реєстра (рис. 3.32).

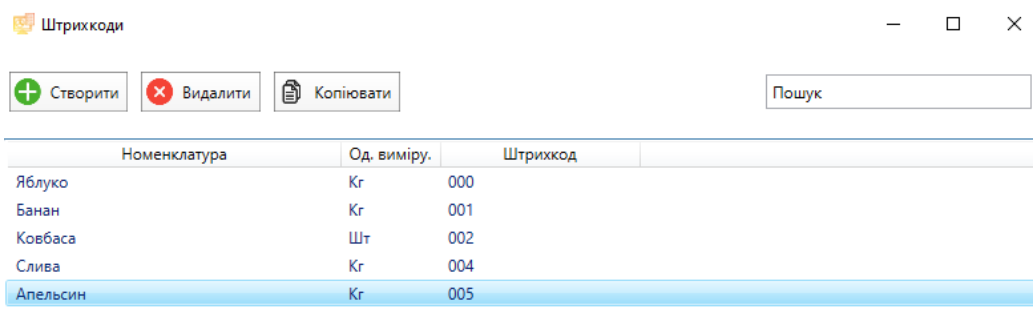


Рис. 3.32 – Список реєстра відомостей «Штрих коди»

Натиснувши команду «Створити» або натиснувши двічі по запису реєстра відкриється форма запису (рис. 3.33), в якій можна заповнити дані реєстра і записати їх.

Рис. 3.33 – Форма запису реєстра відомостей «Штрих коди»

Перед записом відбувається перевірка унікальності запису за вимірами по яким він ведеться, якщо перевірка успішна створюється новий запис і з'являється у списку записів (див. рис. 3.32), в іншому випадку відображається повідомлення про помилку запису (рис. 3.34).

Рис. 3.34 – Діалог повідомлення про помилку запису реєстра відомостей

Пункт «Звіти» відкриває для користувача перелік доступних для формування звітів розділених на 2 колонки «Облік» та «Взаєморозрахунки» (рис. 3.35). В колонці «Облік» знаходяться звіти які відображають інформацію по обліку товарів що зберігаються на складах, також звіти що відображають продажі та закупки товарів. В колонці «Взаєморозрахунки» знаходяться звіт для відображення боргів контрагентів та залишків грошових коштів по касам та банківським рахункам.

Рис. 3.35 – Головна форма пункт меню «Звіти»

Обравши звіт «Залишки на складах» відкриється форма звіту (рис. 3.36). Даний звіт являється залишковим і відображає дані по залишкам на складах. Натиснувши кнопку сформувати буде відображено дані по залишкам на складах у вигляді таблиці із групуванням по складу. В групуванні відображається назва складу і загальна кількість продукції на ньому.

Номенклатура	Кількість	Од.
Основний загалом: 1568		
Гіпсокартон	5	Шт
Цвяхи	650	Шт
Шуруп 2.5x13	800	Шт
Цемент	9	Шт
Штукатурка 25кг	20	Шт
Ковбаса	9	Шт
Апельсин	20	Кг
Слива	20	Кг
Банан	10	Кг
Яблуко	25	Кг

Рис. 3.36 – Звіт «Залишки на складах»

Для фільтрації даних можна встановити надані відбори що призведе до зменшення відображуваних даних в таблиці. Встановивши дату формування звіту, буде отримано дані на вказану дату по залишками товарів на складах, що надає можливість переглянути залишки товарів на будь-який період (рис. 3.37).

Номенклатура	Кількість	Од.
Основний загалом: 20		
Апельсин	20	Кг

Рис. 3.37 – Звіт «Залишки на складах» з відбором по номенклатурі «Апельсин»

Встановивши дату формування звіту, буде отримано дані на вказану дату по залишками товарів на складах, що надає можливість переглянути залишки товарів на будь-яку дату, якщо дата не задано то дані відображаються на дату формування звіту(рис. 3.38).

Залишки на складах

Сформувати Дата 16 червня 2024 р. 0:00:00 Склад X ▼... Номенклатура Апельсин X ▼...

Номенклатура	Кількість	Од.
Основний загалом: 10		
Апельсин	10	Кг

Рис. 3.38 – Звіт «Залишки на складах» з відбором по номенклатурі «Апельсин» та заданою датою залишків «16.06.2024 00:00:00»

Обравши звіт «Продажі» буде відкрита форма даного звіту (рис. 3.39). Даний звіт є оборотним і відображає дані по оборотам продажів товарів.

Продажі

Сформувати Період - Контрагент X ▼... Номенклатура X ▼...

Контрагента	Кількість	Сумма
Цвяхи кількість: 350 Шт сума: 350 UAH		
Будівельники	250	250
Петро	100	100
Цемент кількість: 16 Шт сума: 3375 UAH		
Будівельники	15	3225
Петро	1	150
Шуруп 2.5x13 кількість: 200 Шт сума: 250 UAH		
Будівельники	100	125
Будівельники	100	125
Гіпсокартон кількість: 4 Шт сума: 7500 UAH		
Будівельники	5	3750
Будівельники	-1	3750
Ковбаса кількість: 1 Шт сума: 45 UAH		
Тарас	1	45

Рис. 3.39 – Звіт «Продажі»

Для звіту можна встановити відбори для аналізу продажів по конкретним позиціям та контрагентам (рис. 3.40).

Продажі

Сформувати Період - Контрагент Будівельники X ▼... Номенклатура Гіпсокартон X ▼...

Контрагента	Кількість	Сумма
Гіпсокартон кількість: 4 Шт сума: 7500 UAH		
Будівельники	5	3750
Будівельники	-1	3750

Рис. 3.40 – Звіт «Продажі» з відбором по номенклатурі «Гіпсокартон» та контрагенту «Будівельники»

Задаючи період можна дізнатися скільки товарі було за нього продано (рис. 3.41).

Контрагента	Кількість	Сумма
Гіпсокартон	кількість: 5 Шт	сума: 3750 UAH
Будівельники	5	3750

Рис. 3.41 – Звіт «Продажі» з відбором по номенклатурі «Гіпсокартон» та контрагенту «Будівельники» та заданим періодом

3.3 Тестування роботи додатку

Проводилося тестування під час додавання функціоналу в додаток, також проведено повне тестування після завершення розробки, виявлені помилки виправлялися і повторно тестувалися.

Приклади із виявлених помилок:

- при генерації коді для довідників після коду з номером «000000010» генерація працювала не вірно, для всіх наступних елементів встановлювався код «000000010»;
- при зміні в групі батьківської групи, група не змінювала розташування в ієрархії;
- не завантажувалися дані документів основ в документів «витрата з каси»;
- при оновленні в фоновому завданні курсу валюти на списках об'єктів виникала помилка оновлення відображення через можливість оновлювати дані на формах лише в головному потоці.

При розробці функціоналу були створені тест-кейси для перевірки правильності роботи основного функціоналу додатку.

1. Авторизація:

- *передумова*: користувач знаходиться на формі авторизації;
- *кроки*: користувач коректно заповнює поля «логін», «пароль» та натискає кнопку «ввійти»;

- *очікуваний результат*: користувач потрапляє на головну форму;
- *фактичний результат*: користувач потрапляє на головну форму додатку.

2. Запис довідника не заповнивши обов'язкові поля:

- *передумова*: користувач знаходиться на формі списку довідника;
- *кроки*: користувач натискає команду «створити», відкривається форма нового елементу, користувач натискає кнопку «записати»;
- *очікуваний результат*: користувачу відображається повідомлення зі списком незаповнених полів, новий елемент не створюється;
- *фактичний результат*: користувачу відображається повідомлення зі списком незаповнених полів, новий елемент не створюється.

3. Запис довідника заповнивши обов'язкові поля:

- *передумова*: користувач знаходиться на формі списку довідника;
- *кроки*: користувач натискає команду «створити», відкривається форма нового елементу, користувач заповнює обов'язкові поля, користувач натискає кнопку «записати»;
- *очікуваний результат*: елемент записується, для нього генерується новий код, він з'являється в списку елементів;
- *фактичний результат*: елемент записується, для нього генерується новий код, він з'являється в списку елементів.

4. Копіювання елемента довідника:

- *передумова*: користувач знаходиться на формі списку довідника;
- *кроки*: користувач натискає команду «копіювати», відкривається форма нового елементу із заповненими даними, користувач натискає кнопку «записати»;
- *очікуваний результат*: елемент записується, для нього генерується новий код, він з'являється в списку елементів;
- *фактичний результат*: елемент записується, для нього генерується новий код, він з'являється в списку елементів.

5. Додавання елементу довідника в групу:

- *передумова*: користувач знаходиться на формі елементу ієрархічного довідника;
- *кроки*: користувач натискає кнопку вибору для поля «в групі» із списку груп обирається група, натискається кнопка «записати»;
- *очікуваний результат*: елемент записується, зникає із поточного списку елементів, при активізації вибраної групи елемент з'являється в списку;
- *фактичний результат*: елемент записується, зникає із поточного списку елементів, при активізації вибраної групи елемент з'являється в списку.

6. Видалення елемента довідника із групи:

- *передумова*: користувач знаходиться на формі елементу ієрархічного довідника;
- *кроки*: користувач натискає кнопку очистки для поля «в групі», натискається кнопка «записати»;
- *очікуваний результат*: елемент записується, зникає із поточного списку елементів, при активізації в списку груп «без груп» елемент з'являється в списку;
- *фактичний результат*: елемент записується, зникає із поточного списку елементів, при активізації в списку груп «без груп» елемент з'являється в списку.

7. Проведення документа з помилкою:

- *передумова*: користувач знаходиться на формі документу «видаткова накладна»;
- *кроки*: користувач заповнює реквізити документа, додає товар із кількістю 10000, при наявному залишку 10, та натискає кнопку «провести»;
- *очікуваний результат*: користувачу показується повідомлення через відсутність 9990 одиниць виміру товару на залишках;

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						86

- *фактичний результат*: користувачу показується повідомлення через відсутність 9990 одиниць виміру товару на залишках.

8. Успішне проведення документу:

- *передумова*: користувач знаходиться на формі документу «видаткова накладна»;
- *кроки*: користувач заповнює реквізити документа, додає товар із кількістю 5, при наявному залишку 10, та натискає кнопку «провести»;
- *очікуваний результат*: документ проводиться, товару на залишках стає 5;
- *фактичний результат*: документ проводиться, товару на залишках стає 5.

9. Відміна проведення документу:

- *передумова*: користувач знаходиться на формі документу «видаткова накладна»;
- *кроки*: користувач натискає кнопку «відміна проведення»;
- *очікуваний результат*: відбувається відміна проведення документу, списаний товар повертається на залишки;
- *фактичний результат*: відбувається відміна проведення документу, списаний товар повертається на залишки.

10. Створення запису в реєстрі відомостей:

- *передумова*: користувач знаходиться на формі списку реєстра відомостей «штрих коди»;
- *кроки*: користувач натискає кнопку «створити», обирає номенклатуру, заповнює штрих код;
- *очікуваний результат*: створюється новий запис реєстру відомостей, запис з'являється в списку;
- *фактичний результат*: створюється новий запис реєстру відомостей, запис з'являється в списку.

11. Створення запису в реєстрі відомостей з існуючими унікальними ключами:

- *передумова*: користувач знаходиться на формі списку реєстра відомостей «штрих коди»;
- *кроки*: користувач натискає кнопку «створити», обирає номенклатуру та одиницю, запис по яким уже є в списку, заповнює штрих код;
- *очікуваний результат*: повідомлення про помилку існування запису з подібними ключами;
- *фактичний результат*: повідомлення про помилку існування запису з подібними ключами.

12. Видалення запису реєстра відомостей:

- *передумова*: користувач знаходиться на формі списку реєстра відомостей «штрих коди»;
- *кроки*: користувач обирає запис та натискає кнопку «видалити», на питання чи видалити запис натискає «так»;
- *очікуваний результат*: обраний запис видаляється і зникає зі списку;
- *фактичний результат*: обраний запис видаляється і зникає зі списку.

Висновки до третього розділу

В даному розділі було сформовано діаграму розгортання, описано порядок встановлення та налаштування додатку. Розглянуто інтерфейс додатку та наведено порядок роботи із об'єктами «Довідники», «Документи», «Регістри відомостей» та «Звіти».

Було описано кейси для тестування та перевірено їх на розробленому додатку на правильність поведінки та наявність помилок. Виявлені помилки було виправлено.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано систему автоматизації обробки даних для ведення оперативного та бухгалтерського обліку. Проведено аналіз предметної області, визначено список задач для вирішення та об'єкти потрібні для реалізації додатку. Проведене порівняння із іншими системами для ведення обліку. Для розробки визначено архітектурний підхід у вигляді Clean architecture та описані системні вимоги для стабільної роботи.

При проектуванні додатку визначено та описано основні варіанти використання, сформовані функціональні та не функціональні вимоги. Розроблена структура додатку та описана діаграма класів для відображення зв'язків в системі. Наведено сутності системи на основі яких створена база даних. Розроблено та детально описані основні алгоритми роботи додатком, для кожного алгоритму побудована діаграма діяльності.

Надано схему та порядок розгортання системи автоматизації даних для ведення оперативного та бухгалтерського обліку. Наведено структурний опис інтерфейсу для «Довідники», «Документи», «Регістри відомостей» та «Звіти». Проведено тестування розробленого програмного додатку відповідно до створених тест-кейсів.

Додаток має простий інтерфейс що дозволяє користувачам швидко освоїти функціонал та почати вести оперативний облік товарів, встановлювати ціни, штрих коди, для товарів. Формувати замовлення покупців та постачальників. Фіксувати надходження чи витрату товарів, списання або надходження грошових коштів в каси або рахунки. Формувати звіти по залишками, продажам, закупкам, боргам та грошовим коштам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бухгалтерський облік: види господарського обліку [Електронний ресурс] – Режим доступу до ресурсу: <https://osvita.ua/vnz/reports/accountant/15729/>
2. Автоматизація бізнес-процесів [Електронний ресурс] – Режим доступу до ресурсу: <https://kaf-av.tntu.edu.ua/index.php/mn-abiturient/mn-articles/740-art-automation-business-processes>
3. Odoo [Електронний ресурс] – Режим доступу до ресурсу: https://www.odoo.com/uk_UA
4. LP-CRM [Електронний ресурс] – Режим доступу до ресурсу: https://lp-crm.biz/?reg=1&utm_source=blog-lpcrm
5. Odoo [Електронний ресурс] – Режим доступу до ресурсу: <https://odoo.org.ua/myodoo>
6. Apps and modules — Odoo 17.0 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.odoo.com/documentation/17.0/applications/general/apps_modules.html
7. Огляд архітектури — документація Odoo 17.0 [Електронний ресурс] – Режим доступу до ресурсу: https://www.odoo.com/documentation/17.0/developer/tutorials/server_framework_10_1/01_architecture.html
8. Change language — Odoo 17.0 documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.odoo.com/documentation/17.0/applications/general/users/language.html>
9. Про систему LP-CRM [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.lp-crm.biz/o-sisteme/>
10. Understanding Clean Architecture and Domain-Driven Design (DDD) [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/bimartechnoloji/understanding-clean-architecture-and-domain-driven-design-ddd-24e89caabc40>

11. Clean Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/codenx/clean-architecture-4c23dfa4c698>
12. Роберт М. Чиста архітектура: Мистецтво створення програмного забезпечення. / Роберт М. [Видання друге] – Харків: Фабула, 2023 – 368 с.
13. .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://dotnet.microsoft.com/en-us/>
14. Огляд C# [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview>
15. What is Windows Presentation Foundation [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0>
16. Sheridan Yuen Mastering Windows Presentation Foundation / Sheridan Yuen – [Second Edition] – UK: Packt Publishing Ltd, 2020 – 605 с.
17. Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/>
18. SQLite [Електронний ресурс] – Режим доступу до ресурсу: <https://sqlite.org/index.html>
19. Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: <https://visualstudio.microsoft.com/>
20. Service Locator [Електронний ресурс] – Режим доступу до ресурсу: <https://designpatternsphp.readthedocs.io/en/latest/More/ServiceLocator/README.html>
21. OnConfiguring [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.onconfiguring?view=efcore-8.0>
22. OnModelCreating [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.onmodelcreating?view=efcore-8.0>

23. SaveChangesAsync [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.savechangesasync?view=efcore-8.0>

24. Add [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.add?view=efcore-8.0>

25. Update [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.update?view=efcore-8.0>

26. Remove [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.remove?view=efcore-8.0>

27. UpdateRange [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.updaterange?view=efcore-8.0>

28. RemoveRange [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.remove?view=efcore-8.0>

29. Entry [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.entry?view=efcore-8.0>

30. Вprowadzenia zależności [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>

31. Швець О. Занурення в патерни проектування // Швець О. – [v2.40], 2021. – 395 с.

32. Code first [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database>

33. Data base first [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first>
34. Загальні відомості про LINQ [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/standard/linq/>
35. Reflection in .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/fundamentals/reflection/reflection>
36. Lazy Loading of Related Data [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/querying/related-data/lazy>
37. Невідкладне завантаження пов'язаних даних [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/querying/related-data/eager>
38. IQueryable<T> Interface [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/system.linq.iqueryable-1?view=net-8.0>
39. Проведення і відміна проведення документів .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.codejig.com/uk/entity2305843015656313960/view/4611686018427398203>
40. BackgroundWorker [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/api/system.componentmodel.backgroundworker?view=net-8.0>
41. .NET SDK overview .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/core/sdk>

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						94

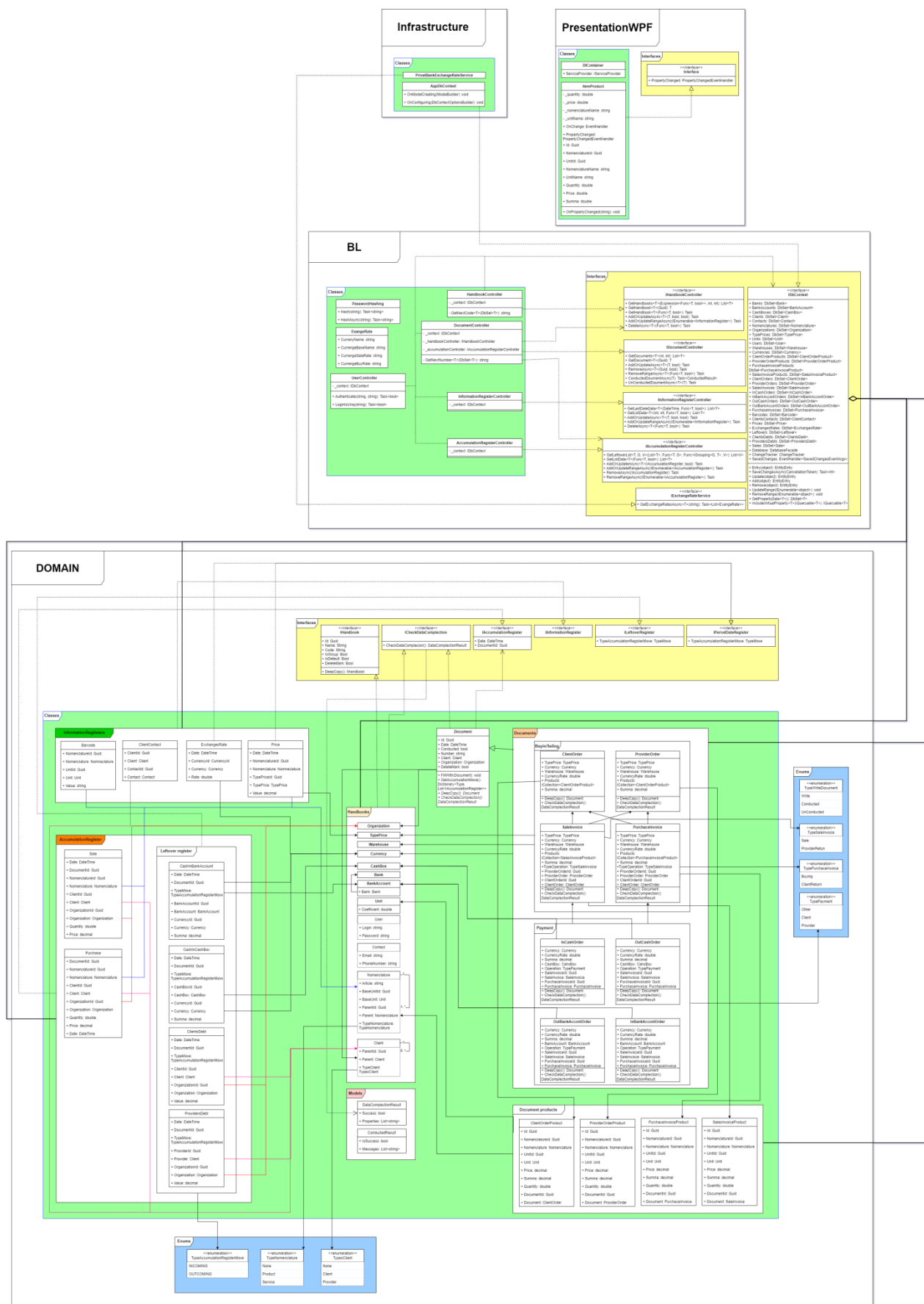


Рис. А.1 – Діаграма класів додатку

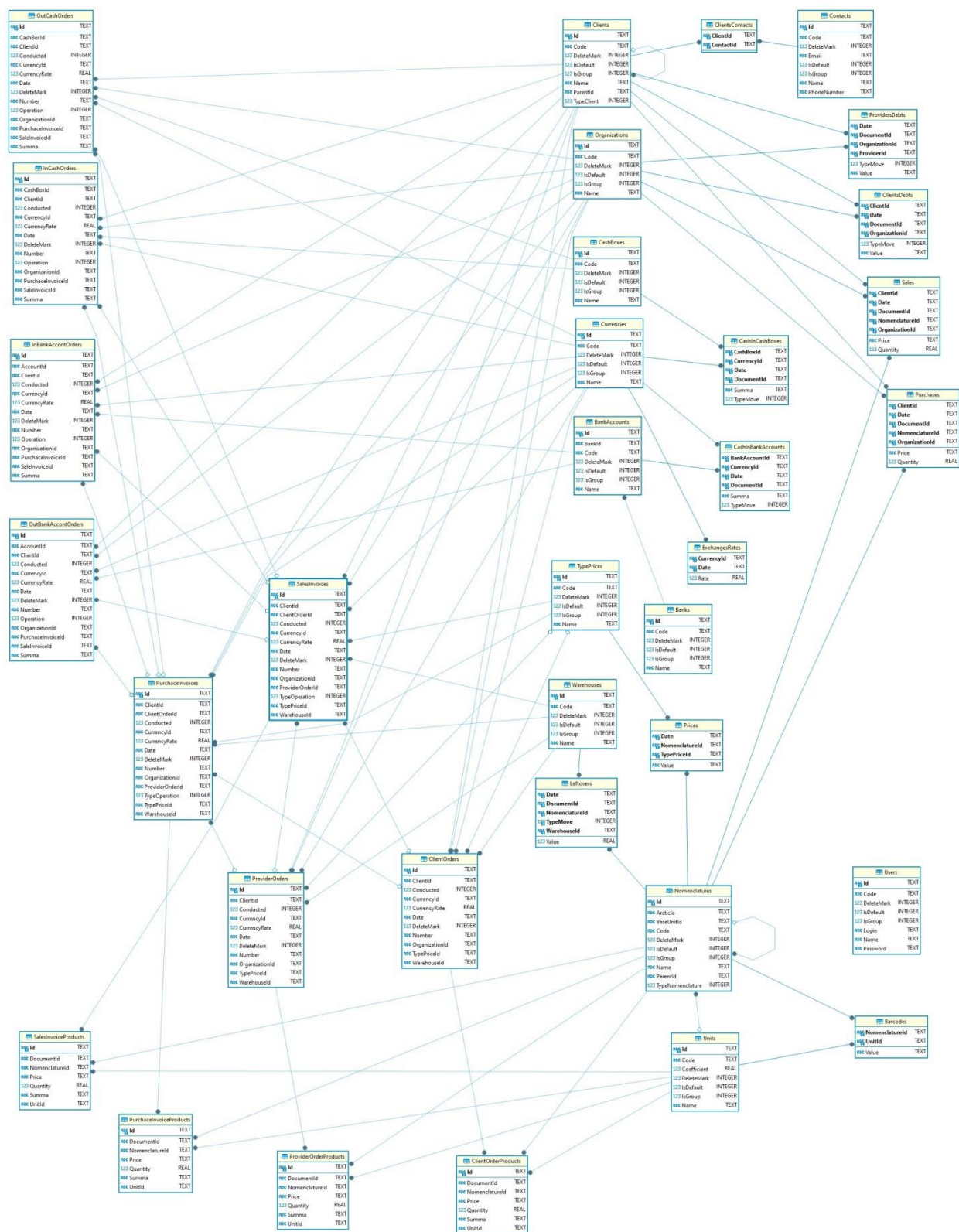


Рис. Б.1 – Фізична ER-діаграма

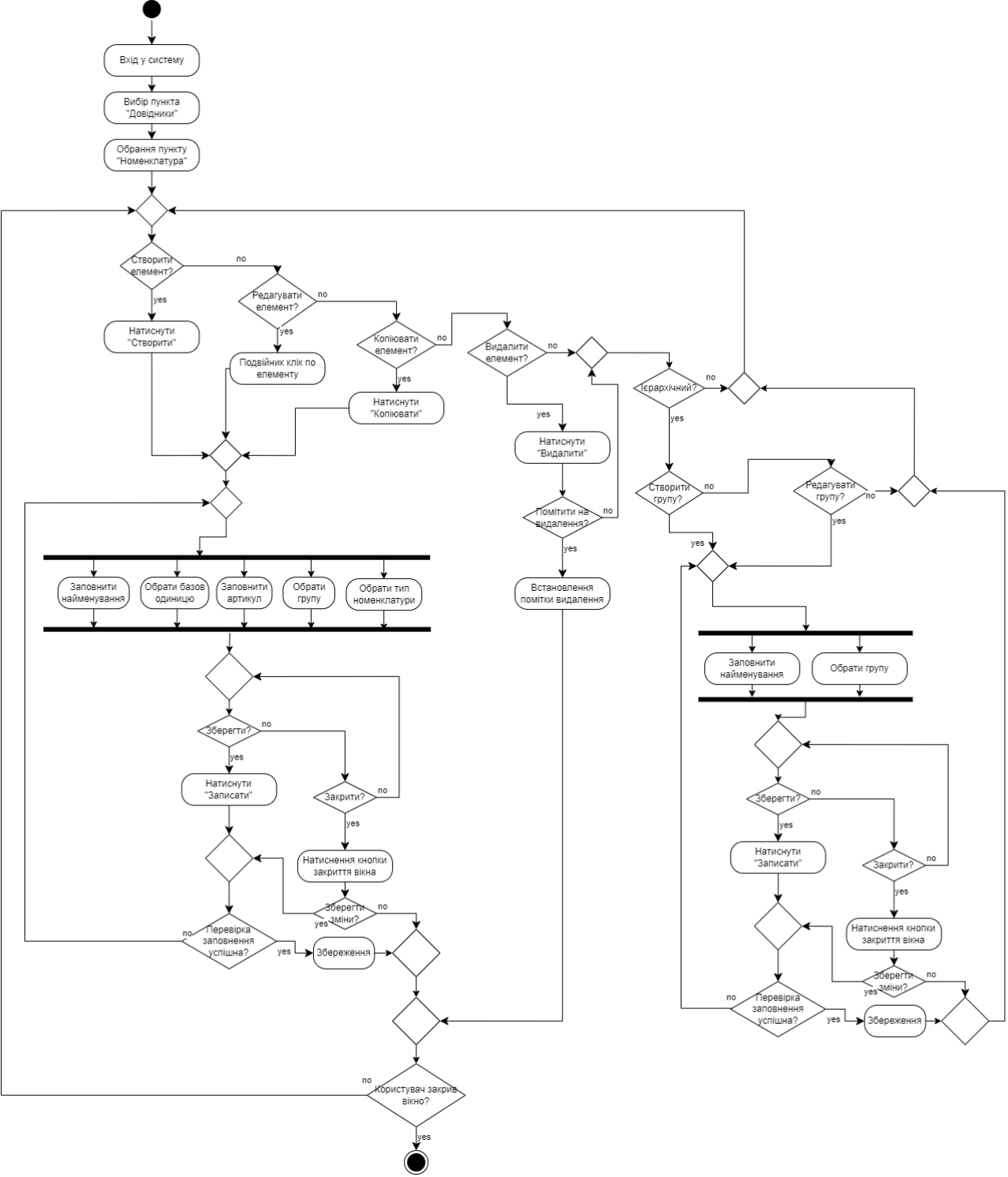


Рис. В.1 – діаграма діяльності «Робота з довідниками»

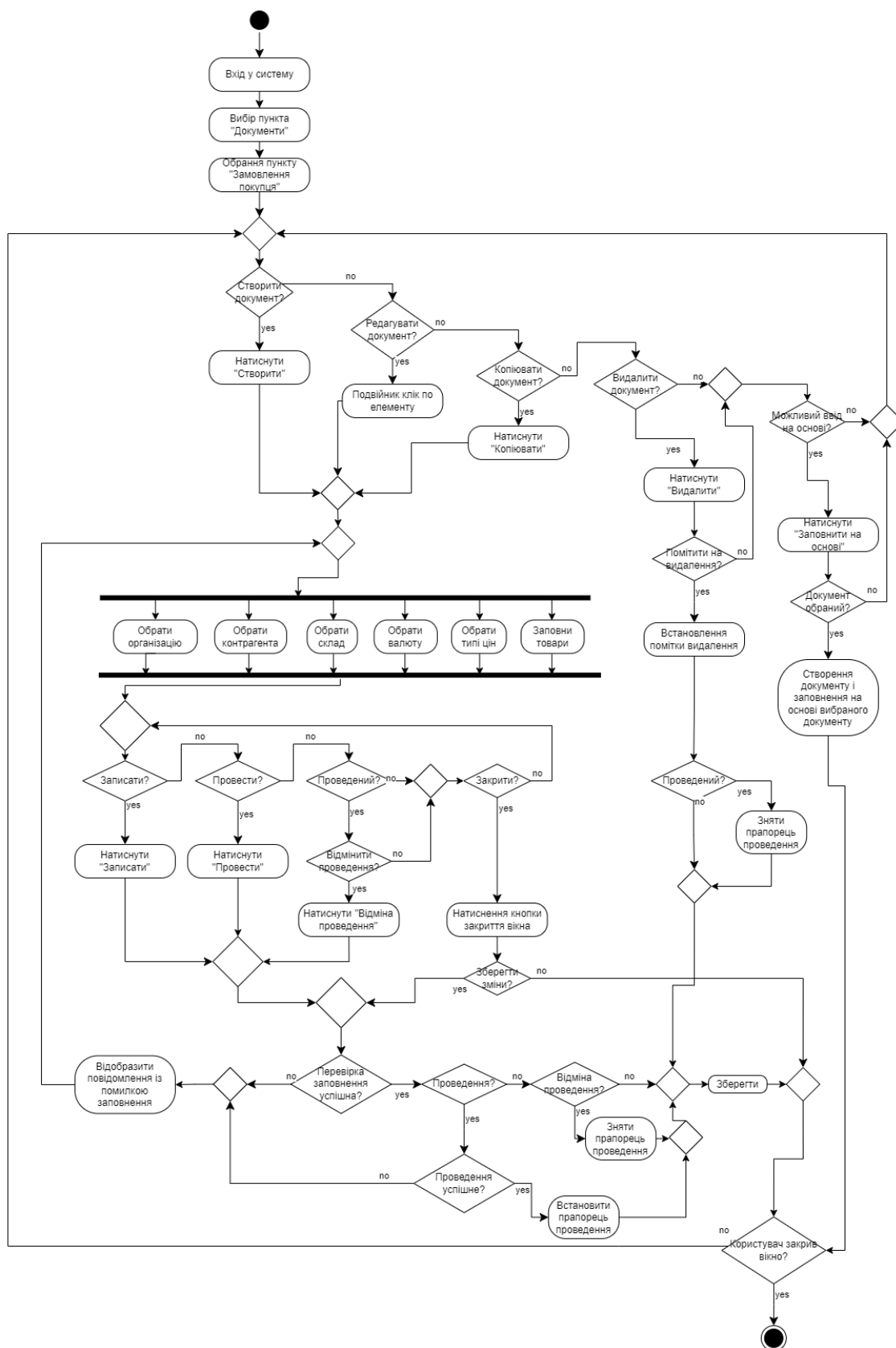


Рис. Г.1 – діаграма діяльності «Робота з документами»

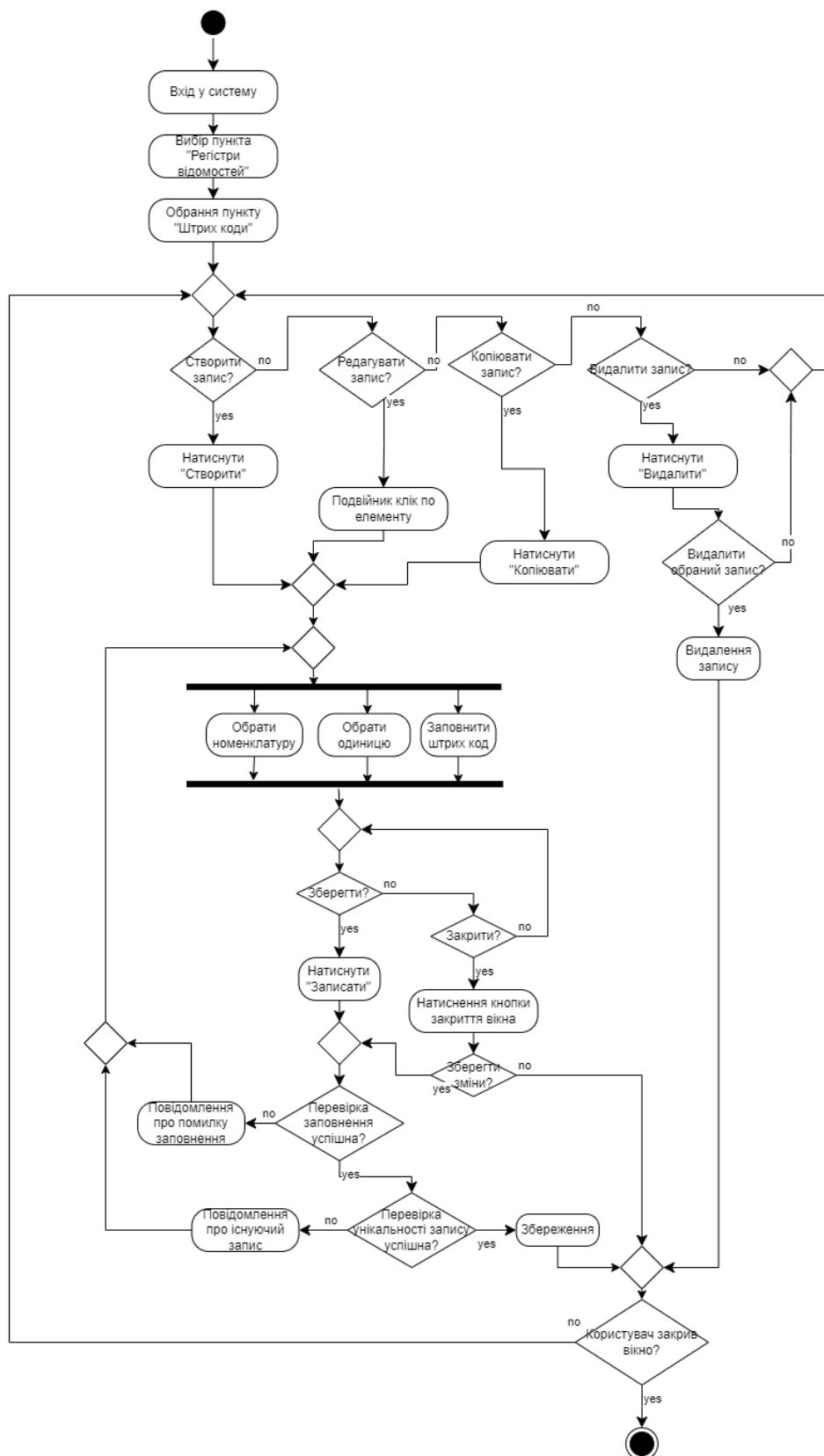


Рис. Д.1 – діаграма діяльності «Робота з регістрами відомостей»

Лістинг функції ConductedDocumentAsync класу DocumentController.

```
public async Task<ConductedResult> ConductedDocumentAsync<T>(T document) where T :
Document
```

```
{
    var result = new ConductedResult();
    var moves = document.GetAccumulationMove();

    Dictionary<Type, IEnumerable<IAccumulationRegister>> oldMoves = new();
    if (document.Id != Guid.Empty)
    {
        Func<IAccumulationRegister, bool> func = k => k.DocumentId == document.Id;
        foreach (var move in moves)
        {
            var oldMove = (IEnumerable<IAccumulationRegister>)_accumulationController
                .GetType()
                .GetMethod("GetListData")
                .MakeGenericMethod(move.Key)
                .Invoke(_accumulationController, [func]);

            if (oldMove != null && oldMove.Count() != 0)
                oldMoves.Add(move.Key, oldMove);
        }
    }

    foreach (var move in moves)
    {
        string type = move.Key.Name;
```

```

switch (type)
{
    case nameof(CashInCashBox):
    {

        CashInCashBox? item = move.Value.FirstOrDefault() as CashInCashBox;

        if (item != null && item.TypeMove ==
TypeAccumulationRegisterMove.OUTCOMING)
        {

            var data = _accumulationController.GetListData<CashInCashBox>(s =>
s.CashBoxId == item.CashBox.Id && s.CurrencyId == item.Currency.Id);

            var leftovers = _accumulationController.GetLeftoverList(data, g =>
g.CashBox.Id, s => new { CashBox = s.Key, Summa = s.Sum(sl => sl.TypeMove ==
TypeAccumulationRegisterMove.INCOMING ? sl.Summa : -sl.Summa) });

            decimal summa = 0;

            if (oldMoves.TryGetValue(move.Key, out var res))
            {
                CashInCashBox? prevData = res.FirstOrDefault() as CashInCashBox;

                if (prevData != null)
                {
                    summa = prevData.Summa;
                }

                var leftover = leftovers.FirstOrDefault();

```

```

        if (leftover == null || leftover.Summa + summa - item.Summa < 0)

            result.Messages.Add($"В касі {item.CashBox.Name} не достатньо коштів:
            {item.Summa - (leftover != null ? leftover.Summa + summa : 0)} {item.Currency?.Name}");

        }

        break;

    }

    case nameof(CashInBankAccount):

        {

            CashInBankAccount? item = move.Value.FirstOrDefault() as
            CashInBankAccount;

            if (item != null && item.TypeMove ==
            TypeAccumulationRegisterMove.OUTCOMING)

                {

                    var data = _accumulationController.GetListData<CashInBankAccount>(s =>
                    s.BankAccountId == item.BankAccount.Id && s.CurrencyId == item.Currency.Id);

                    var leftovers = _accumulationController.GetLeftoverList(data, g =>
                    g.BankAccount.Id, s => new { Summa = s.Sum(sl => sl.TypeMove ==
                    TypeAccumulationRegisterMove.INCOMING ? sl.Summa : -sl.Summa) });

                    var leftover = leftovers.FirstOrDefault();

                    decimal summa = 0;

                    if (oldMoves.TryGetValue(move.Key, out var res))

                        {

```

```
CashInBankAccount? prevData = res.FirstOrDefault() as  
CashInBankAccount;
```

```
    if (prevData != null)  
        summa = prevData.Summa;  
}
```

```
    if (leftover == null || leftover.Summa + summa - item.Summa < 0)  
        result.Messages.Add($"На рахунок {item.BankAccount.Name} не  
достатньо коштів: {item.Summa - (leftover != null ? leftover.Summa + summa - item.Summa : 0)}  
{item.Currency?.Name}");
```

```
    }
```

```
    break;
```

```
}
```

```
case nameof(Leftover):
```

```
{
```

```
    HashSet<Guid> nomenclatures = new();
```

```
    HashSet<Guid> warehouses = new();
```

```
    foreach (Leftover value in move.Value)
```

```
    {
```

```
        if (value.TypeMove == TypeAccumulationRegisterMove.OUTCOMING)
```

```
        {
```

```
            warehouses.Add(value.Warehouse.Id);
```

```
        }
```

```
        nomenclatures.Add(value.NomenclatureId);
```

```
    }
```

```
var nomenclatureList = _handbookController.GetHandbooks<Nomenclature>(w
=> nomenclatures.Contains(w.Id));
```

```
foreach (var item in nomenclatureList)
{
    if (item.TypeNomenclature == TypeNomenclature.Service)
    {
        move.Value.RemoveAll(x => ((Leftover)x).NomenclatureId == item.Id);
    }
}
```

```
if (warehouses.Count == 0)
    break;
```

```
var list = _accumulationController.GetListData<Leftover>(w =>
nomenclatures.Contains(w.NomenclatureId)
```

```

                                &&
warehouses.Contains(w.WarehouseId));
```

```
if (list.Count == 0)
{
```

```
    foreach (Leftover value in move.Value)
    {
```

```
        var nomenclature = nomenclatureList.FirstOrDefault(w => w.Id ==
value.NomenclatureId);
```

```
        if (nomenclature != null)
```

```

        result.Messages.Add($"Не          вистачає          {value.Value}
{nomenclature.BaseUnit.Name} залишків {nomenclature.Name} на {value.Warehouse.Name}");

    }

    break;
}

var leftovers = _accumulationController.GetLeftoverList(list,
    g => new { g.NomenclatureId, g.Warehouse.Id },
    s => new
    {
        id = string.Join("", s.Key.NomenclatureId, s.Key.Id),
        value = s.Sum(selector => selector.TypeMove ==
TypeAccumulationRegisterMove.INCOMING ? selector.Value : selector.Value * -1)
    });

if (leftovers.Count == 0)
{
    foreach (Leftover value in move.Value)
    {
        var nomenclature = nomenclatureList.FirstOrDefault(w => w.Id ==
value.NomenclatureId);

        if (nomenclature != null)
        {
            result.Messages.Add($"Не          вистачає          {value.Value}
{nomenclature.BaseUnit.Name} залишків {nomenclature.Name} на {value.Warehouse.Name}");

        }
    }
}

```

```

else
{
    Dictionary<string, double> oldMove = new();

    if (oldMoves.TryGetValue(move.Key,
        out IEnumerable<IAccumulationRegister> res))
    {
        if (res != null)
        {
            foreach (Leftover item in res)
            {
                oldMove.Add(string.Join("", item.NomenclatureId, item.WarehouseId),
                    item.Value);
            }
        }
    }

    foreach (Leftover item in move.Value)
    {
        var nomenclature = nomenclatureList.FirstOrDefault(w => w.Id ==
            item.NomenclatureId);

        var id = string.Join("", item.NomenclatureId, item.Warehouse.Id);
        var data = leftovers.FirstOrDefault(x => x.id == id);
        double leftover = 0;

        oldMove.TryGetValue(id, out leftover);

        if (data != null)
            leftover += data.value;
    }
}

```

```

var value = leftover - (item.Value);

if (value < 0)
{

    if (leftover < 0)
        value = item.Value;

    if (nomenclature != null)
        result.Messages.Add($"Не вистачає {Math.Abs(value)}
{nomenclature.BaseUnit.Name} залишків {nomenclature.Name} на {item.Warehouse.Name}");
    }
}

break;
}
default:
    break;
}
}

if (result.IsSuccess)
{
    using (var transaction = await _context.Database.BeginTransactionAsync())
    {
        try
        {

```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						107

```

        foreach (var item in oldMoves)
        {
            foreach (var value in item.Value)
            {
                await _accumulationController.RemoveAsync(value);
            }
        }

        document.Conducted = true;
        document.Date = DateTime.Now;

        await AddOrUpdateAsync(document);

        foreach (var move in moves)
        {
            {
                foreach (var item in move.Value)
                {
                    {
                        item.DocumentId = document.Id;
                    }
                }
            }

            await _accumulationController.AddOrUpdateRangeAsync(move.Value);
        }

        await transaction.CommitAsync();
    }
    catch (Exception e)
    {
        {
            await transaction.RollbackAsync();
            result.Messages.Add("Помилка під час проведення документу: " + e.Message);
        }
    }
}

```

```

        return result;
    }

```

Додаток Є

Лістинг функції UnConductedDoumentAsync класу DocumentController.

```

public async Task UnConductedDoumentAsync<T>(T document) where T : Document
{
    using (var transaction = await _context.Database.BeginTransactionAsync())
    {
        try
        {
            var moves = document.GetAccumulationMove();

            Dictionary<Type, IEnumerable<IAccumulationRegister>> oldMoves = new();
            if (document.Id != Guid.Empty)
            {
                Func<IAccumulationRegister, bool> kva = k => k.DocumentId == document.Id;
                foreach (var move in moves)
                {
                    var oldMove =
                        (IEnumerable<IAccumulationRegister>)_accumulationController.GetType().GetMethod("GetListD
ata").MakeGenericMethod(move.Key).Invoke(_accumulationController, [kva]);

                    if (oldMove != null && oldMove.Count() != 0)
                        oldMoves.Add(move.Key, oldMove);
                }
            }

            foreach (var item in oldMoves)
                foreach (var value in item.Value)
                    await _accumulationController.RemoveAsync(value);

            document.Conducted = false;

            await AddOrUpdateAsync(document);

            await transaction.CommitAsync();
        }
        catch (Exception)
        {
            await transaction.RollbackAsync();
        }
    }
}

```

Додаток Ж

Лістинг сервісу PrivatBankExchangeRateService.

```

public class PrivatBankExchangeRateService : IExchangeRateService

```

						Арк.
					ІПЗ.КР.Б – 121 – 24 – ПЗ	109

```

{
    private record Data(string? ccy, string? base_ccy, string? buy, string? sale);

    public async Task<List<ExchangeRate>> GetExchangeRatesAsync(string currencyName = "")
    {
        List<ExchangeRate> rates = new List<ExchangeRate>();
        var uri = new Uri(@"https://api.privatbank.ua/p24api/pubinfo?exchange&coursid=5");
        var request = new HttpRequestMessage(HttpMethod.Get, uri);

        var client = new HttpClient();
        client.Timeout = TimeSpan.FromSeconds(30);

        try
        {
            var result = await client.GetFromJsonAsync<List<Data>>(uri);

            if (result == null)
                return rates;

            foreach(var item in result)
            {
                if (!string.IsNullOrEmpty(currencyName) && item.ccy != currencyName)
                    continue;

                var rate = new ExchangeRate()
                {
                    CurrengeBaseName = item.base_ccy,
                    CurrenyName = item.ccy,
                };

                double sale = 0;
                double buy = 0;

                CultureInfo culture = CultureInfo.CreateSpecificCulture("en-US");

                double.TryParse(item.sale, NumberStyles.AllowDecimalPoint, culture, out sale);
                double.TryParse(item.buy, NumberStyles.AllowDecimalPoint, culture, out buy);

                rate.CurrengeBuyRate = buy;
                rate.CurrengeSaleRate = sale;

                rates.Add(rate);
            }
        }
        catch (Exception) {}

        return rates;
    }
}

```