МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП’ЮТЕРНІ НАУКИ»

ЦИКЛОВА КОМІСІЯ СПЕЦІАЛЬНОСТІ
«КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня фаховий молодший бакалавр
за спеціальністю 122 Комп'ютерні науки
на тему:

«Інформаційна система обліку та управління гаражним кооперативом»

Виконав здобувач освіти групи П-42
Мартинчук Адам Миколайович

Керівник роботи:
Устименко Ярослав Іванович

Рецензент:

Житомир – 2026

Зміст

РЕФЕРАТ

Кваліфікаційна робота присвячена проектуванню та розробці інформаційної системи, що автоматизує облік членів, гаражів, тарифів і платежів гаражного кооперативу з функціями контролю заборгованості та формування звітності.

У роботі проведено аналіз предметної області та аналогів, сформульовано вимоги до системи, розроблено UML-моделі (Use Case, діаграми класів, послідовності та станів), спроектовано базу даних та архітектуру на основі MVVM. Реалізовано сім модулів системи: облік членів, гаражів, тарифів, платежів, розрахунок заборгованості, чотири типи звітів та їх експорт до Excel.

Систему реалізовано засобами C# .NET 9, WPF, Entity Framework Core 9, SQLite, CommunityToolkit.Mvvm, MaterialDesignInXamlToolkit та ClosedXML. Проведено функціональне тестування (20 тест-кейсів) та тестування алгоритму розрахунку (8 сценаріїв); всі тести пройдено успішно.

Практичне значення роботи: система може бути впроваджена в будь-якому гаражному кооперативі як готовий продукт. Файл бази даних SQLite зберігається локально, не потребує серверної інфраструктури та є придатним для резервного копіювання.

Обсяг роботи: 60 сторінок, 11 рисунків, 10 таблиць, 3 додатки, 23 джерела.

Ключові слова: інформаційна система, гаражний кооператив, облік членів, облік платежів, заборгованість, WPF, MVVM, Entity Framework Core, SQLite, .NET 9.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД — база даних.

ЕОМ — електронно-обчислювальна машина.

ЗНФ — третя нормальна форма.

ІС — інформаційна система.

ПЗ — пояснювальна записка.

ПІБ — прізвище, ім'я, по батькові.

СУБД — система управління базами даних.

DI (Dependency Injection) — впровадження залежностей.

EF Core (Entity Framework Core) — об'єктно-реляційне відображення для платформи .NET.

FK (Foreign Key) — зовнішній ключ.

KPI (Key Performance Indicators) — ключові показники ефективності.

MVVM (Model-View-ViewModel) — архітектурний шаблон проєктування програмного забезпечення.

ORM (Object-Relational Mapper) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування.

PK (Primary Key) — первинний ключ.

POCO (Plain Old CLR Objects) — прості класи CLR, що не залежать від спеціалізованих фреймворків.

SQL (Structured Query Language) — мова структурованих запитів.

UML (Unified Modeling Language) — уніфікована мова моделювання.

WPF (Windows Presentation Foundation) — система для побудови графічних інтерфейсів користувача.

XAML (eXtensible Application Markup Language) — декларативна мова розмітки для створення інтерфейсів.

ВСТУП

Актуальність теми.

В умовах стрімкого розвитку інформаційних технологій автоматизація управління організаціями різних форм власності набуває дедалі більшого значення. Гаражні кооперативи є поширеною формою об'єднання громадян для спільного використання майна та управління ним. Незважаючи на широке розповсюдження, більшість таких організацій в Україні досі ведуть облік членів, платежів і боргів у паперових журналах або засобами загального призначення (таблиці Excel), що призводить до помилок, втрати даних і значних витрат часу адміністрації.

Відсутність спеціалізованого програмного забезпечення для гаражних кооперативів зумовлює потребу у розробці інформаційної системи, яка б автоматизувала ключові облікові процеси: реєстрацію членів, облік гаражів, нарахування та контроль сплати внесків, розрахунок заборгованості, формування звітності. Впровадження такої системи дозволяє суттєво підвищити точність обліку, прозорість фінансових операцій і загальну ефективність управління кооперативом.

Таким чином, розробка інформаційної системи обліку та управління гаражним кооперативом є актуальним завданням, що має практичне значення для широкого кола подібних організацій.

Мета роботи — проектування та розробка десктопної інформаційної системи, що забезпечує автоматизований облік членів, гаражів, тарифів і платежів гаражного кооперативу, контроль заборгованості та формування звітності.

Для досягнення поставленої мети визначено такі завдання:

1) проаналізувати предметну область та існуючі інформаційні системи управління кооперативами;
2) сформулювати функціональні та нефункціональні вимоги до системи;
3) обґрунтувати вибір технологій і інструментів розробки;

4) спроектувати архітектуру системи, структуру бази даних та інтерфейс користувача;
5) реалізувати модулі обліку членів, гаражів, тарифів, платежів і розрахунку заборгованості;
6) реалізувати модуль формування звітів та експорту даних до формату Excel;
7) провести функціональне тестування системи та проаналізувати результати.

Об'єкт дослідження — процеси обліку та управління гаражним кооперативом, зокрема реєстрація членів, облік нерухомості, нарахування і сплата внесків, контроль заборгованості та формування фінансової звітності.

Предмет дослідження — методи та засоби проектування і розробки інформаційних систем на основі технологій .NET, WPF і SQLite, що забезпечують автоматизацію облікових процесів кооперативу.

У роботі використано такі методи: системний аналіз — для дослідження предметної області та формулювання вимог; структурне та об'єктно-орієнтоване проектування — для побудови архітектури системи та моделювання бізнес-процесів; метод прецедентів (Use Case) — для специфікації функціональних вимог; UML-моделювання — для побудови діаграм класів, послідовності та станів; функціональне тестування — для верифікації коректності роботи системи.

Розроблена інформаційна система може бути впроваджена в будь-якому гаражному кооперативі незалежно від його масштабу. Система забезпечує:

1) ведення повного реєстру членів кооперативу та їхніх контактних даних;
2) облік гаражних боксів із відображенням власника, площі та типу конструкції;
3) гнучке налаштування тарифів (членські внески, електроенергія, охорона, цільові збори);
4) реєстрацію платежів із підтримкою часткових і авансових сплат;

5) автоматичний розрахунок заборгованості кожного члена з урахуванням усіх платежів;
6) формування фінансових звітів і їх експорт до Excel для подання на зборах кооперативу.

Результати роботи мають практичну цінність для голів та бухгалтерів гаражних кооперативів, а також можуть слугувати основою для подальшого розширення системи — зокрема, додавання веб-інтерфейсу, мобільного додатку або хмарного сховища даних.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Характеристика гаражного кооперативу як об'єкта автоматизації

Гаражний кооператив — це добровільне об'єднання громадян, метою якого є спільне будівництво, утримання та використання гаражних боксів. Відповідно до законодавства України, кооператив є юридичною особою, що здійснює діяльність на основі статуту, обраного правління та загальних зборів членів.

Основними учасниками управлінського процесу в гаражному кооперативі є: голова правління, який здійснює загальне керівництво; бухгалтер (касир), відповідальний за фінансовий облік та збір внесків; члени кооперативу — фізичні особи, що мають у власності або користуванні один або кілька гаражних боксів.

З точки зору автоматизації, діяльність гаражного кооперативу охоплює такі ключові бізнес-процеси:

- ведення реєстру членів кооперативу (персональні дані, контакти, дата вступу, статус);
- облік гаражних боксів (номер, ряд, площа, тип конструкції, закріплений власник, статус зайнятості);
- управління тарифами та видами внесків (членський, за електроенергію, охорону, цільові збори);
- реєстрація платежів від членів та контроль їхньої своєчасності;
- розрахунок заборгованості за кожним видом внеску;
- формування звітної документації для зборів кооперативу.

Ці процеси тісно пов'язані між собою: розмір боргу залежить від дати вступу члена, діючих тарифів і сплачених сум. Будь-яка зміна одного з цих параметрів впливає на розрахунки по іншим. Ця взаємозалежність зумовлює потребу у спеціалізованій інформаційній системі, а не у розрізнених засобах обліку.

Специфіка гаражного кооперативу як об'єкта автоматизації полягає також у тому, що система повинна:

- підтримувати часткові та авансові платежі — коли член сплачує більше або менше нарахованої суми, або вносить оплату наперед за кілька місяців;
- коректно відображати переплату — як від'ємний борг, що зараховується у рахунок майбутніх нарахувань;
- забезпечувати пошук і фільтрацію даних за різними критеріями для оперативного контролю;
- бути простою у встановленні та використанні, оскільки адміністратори кооперативів, як правило, не є IT-фахівцями.

1.2 Аналіз існуючих інформаційних систем

На ринку програмного забезпечення існує кілька рішень, що теоретично можуть застосовуватися для управління гаражними кооперативами. Проте жодне з них не є спеціалізованим саме для цієї форми організації. Розглянемо найбільш поширені альтернативи.

TourManager та подібні системи управління кооперативами. Програмні продукти класу "управління членською організацією" орієнтовані переважно на споживчі кооперативи або товариства. Вони забезпечують облік членів і платежів, однак не мають функціоналу для обліку гаражних боксів як окремих об'єктів нерухомості та не підтримують прив'язку тарифів до конкретних видів майна. Крім того, такі системи, як правило, є хмарними та потребують постійного інтернет-підключення, що є суттєвим обмеженням для невеликих кооперативів.

1С:Бухгалтерія та 1С:Кооператив. Платформа 1С є розповсюдженим рішенням для бухгалтерського обліку в Україні. Модуль "Кооператив" дозволяє вести реєстр членів і нараховувати внески. Однак вартість ліцензії та технічного супроводу є значною для невеликих організацій, а налаштування системи під специфіку гаражного кооперативу потребує залучення фахівця.

До того ж, інтерфейс є перевантаженим і розрахованим на бухгалтерів, а не на голів кооперативів.

Microsoft Excel та Google Таблиці. Електронні таблиці є найбільш поширеним інструментом обліку у малих кооперативах завдяки своїй доступності та відомості. Однак вони мають принципові обмеження: відсутність вбудованої логіки валідації даних, неможливість автоматичного розрахунку заборгованості при змінні тарифів, відсутність зручного інтерфейсу для реєстрації платежів, висока ймовірність помилок при ручному введенні. При кількості членів понад 20 осіб ведення таблиць стає трудомістким і ненадійним.

Спеціалізовані локальні системи. Існують одиничні розробки, що позиціонуються як системи обліку для гаражних кооперативів. Більшість із них є застарілими, не підтримуються розробниками та несумісні з сучасними версіями Windows. Їхній інтерфейс є архаїчним, а функціональність обмеженою.

1.3 Порівняльний аналіз аналогів

На основі аналізу, наведеного у підрозділі 1.2, виконано порівняльний аналіз альтернатив за ключовими критеріями (табл. 1.1). Знак "+" означає повну відповідність критерію, "–" — відсутність, "Частково" — неповну реалізацію.

Таблиця 1.1 — Порівняльний аналіз програмних аналогів

Критерій	TourManager	1С:Кооператив	Excel	Розроблена система
Спеціалізація для гаражних кооперативів	–	Частково	–	+
Облік членів і гаражів	+	+	+	+
Автоматичний розрахунок боргу	+	+	–	+
Облік авансових платежів	–	Частково	–	+
Формування звітів	+	+	+	+

Експорт до Excel	+	+	+	+
Безкоштовна ліцензія	–	–	+	+
Не потребує сервера	–	–	+	+
Українська локалізація інтерфейсу	–	+	+	+
Мінімальні вимоги до ПК	+	–	+	+

Як видно з таблиці 1.1, жоден з аналогів не задовольняє одночасно всім ключовим критеріям. Зокрема, жодна готова система не підтримує коректний облік авансових платежів і відображення переплати, а безкоштовні рішення (Excel) не мають автоматичного розрахунку боргу. Це підтверджує доцільність розробки власної спеціалізованої системи.

1.4 Обґрунтування необхідності розробки власної системи

За результатами аналізу предметної області та існуючих аналогів встановлено, що розробка спеціалізованої інформаційної системи є виправданою з таких причин:

1) відсутність на ринку готового безкоштовного рішення, що повністю відповідає специфіці гаражного кооперативу;
2) неможливість адекватного обліку авансових і часткових платежів у існуючих системах;
3) складність і висока вартість адаптації комерційних систем (1С) до потреб невеликого кооперативу;
4) потреба в автономній роботі без серверної інфраструктури та постійного інтернет-підключення;
5) необхідність україномовного інтерфейсу з підтримкою форматів дат і валюти, прийнятих в Україні.

Розроблена в рамках кваліфікаційної роботи система спрямована на усунення всіх зазначених недоліків: вона є безкоштовною, спеціалізованою, автономною (локальна SQLite-база), повністю україномовною та коректно обробляє всі сценарії оплати — включно з передоплатою і частковими платежами.

1.5 Вимоги до інформаційної системи

На основі аналізу предметної області та інтерв'ю з потенційними користувачами (голова та бухгалтер гаражного кооперативу) сформульовано функціональні та нефункціональні вимоги до системи (табл. 1.2).

Таблиця 1.2 — Вимоги до інформаційної системи

Вимога	Категорія	Опис
Реєстрація членів	Функціональна	Додавання, редагування, видалення членів кооперативу з усіма реквізитами
Облік гаражів	Функціональна	Ведення довідника гаражів: номер, ряд, площа, тип, власник, статус
Управління тарифами	Функціональна	Налаштування видів і сум внесків, їх періодичності
Реєстрація платежів	Функціональна	Фіксація платежів із зазначенням дати, суми, тарифу, способу оплати
Розрахунок заборгованості	Функціональна	Автоматичний розрахунок боргу з урахуванням авансових і часткових платежів
Відображення переплати	Функціональна	Виведення переплати для членів, що сплатили наперед
Формування звітів	Функціональна	Звіти: список членів, боржники, фінансовий звіт, платежі за період
Експорт до Excel	Функціональна	Вивантаження будь-якого звіту у формат .xlsx
Авторизація користувачів	Нефункціональна	Захист системи паролем; ролі: адміністратор, касир
Швидкодія	Нефункціональна	Час відгуку основних операцій — не більше 1 секунди
Надійність даних	Нефункціональна	Збереження даних у локальній базі SQLite без ризику втрати при закритті
Зручність використання	Нефункціональна	Інтуїтивний інтерфейс мовою Material Design, освоєння без навчання

Для наочного подання функціональних вимог розроблено діаграму варіантів використання (Use Case), що відображає взаємодію двох типів користувачів — Адміністратора і Касира — з основними функціями системи

(рис. 1.1). Зв'язки include показують обов'язкові підпрецеденти, extend — умовні розширення. Адміністратор є більш привілейованою роллю, що успадковує всі прецеденти Касира (generalization).

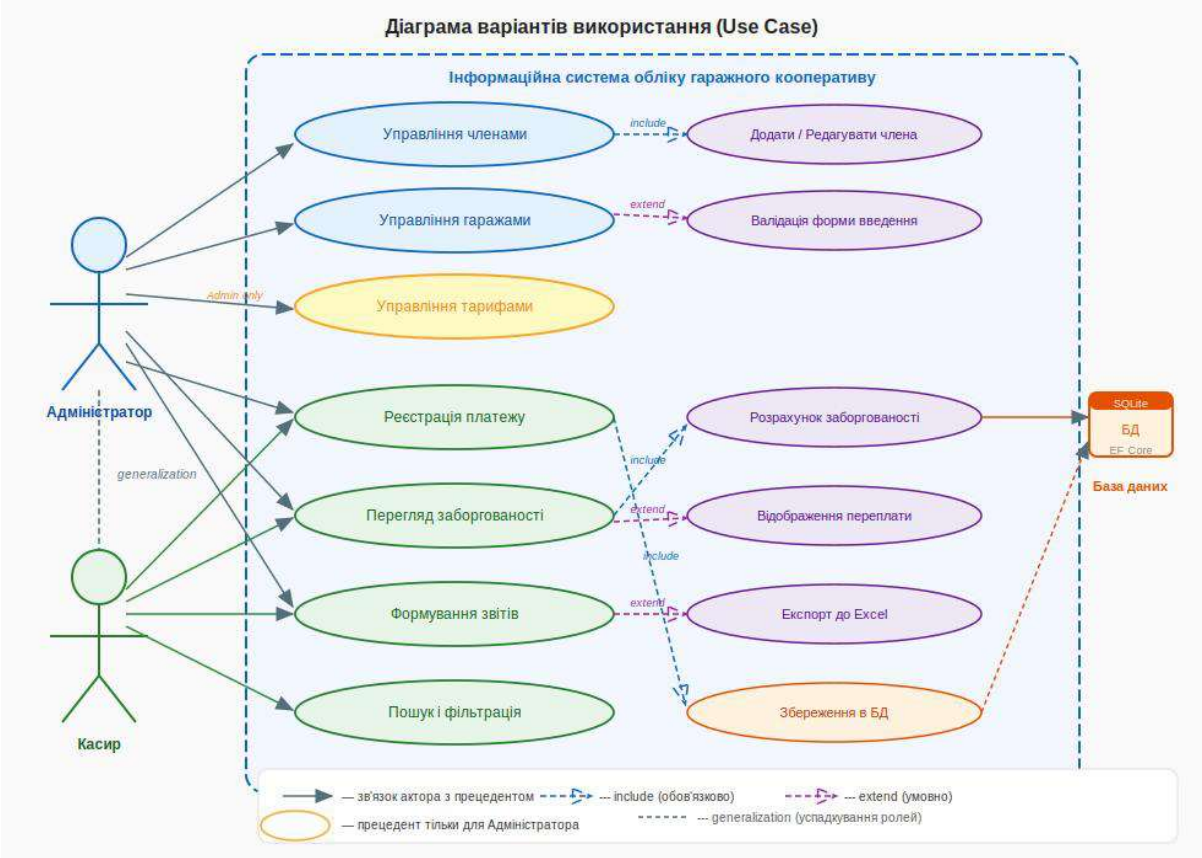

Рисунок 1.1 — Діаграма варіантів використання інформаційної системи

Адміністратор має повний доступ до всіх функцій системи, включаючи управління довідниками (члени, гаражі, тарифи), перегляд і редагування платежів, формування будь-яких звітів. Касир має обмежені права: реєстрація платежів, перегляд списку членів та боргів, формування фінансових звітів.

1.6 Вибір інструментів розробки

Вибір технологічного стеку для розробки інформаційної системи здійснювався на основі аналізу вимог, наведених у підрозділі 1.5, а також з урахуванням доступних засобів розробки.

Мова програмування C# та платформа .NET 9. C# є сучасною об'єктно-орієнтованою мовою програмування загального призначення, що активно розвивається компанією Microsoft. Платформа .NET 9 забезпечує

крос-компіляцію, високу продуктивність та широку екосистему бібліотек. Для десктопних Windows-застосунків C# є природним вибором, оскільки забезпечує найкращу інтеграцію з WPF та іншими Windows-специфічними технологіями.

Windows Presentation Foundation (WPF). WPF — це фреймворк для створення десктопних графічних застосунків під Windows, заснований на XAML-розмітці та підтримці шаблону проектування MVVM (Model-View-ViewModel). Перевагами WPF є: векторна графіка та підтримка масштабування, гнучка система стилів і шаблонів, декларативне визначення інтерфейсу через XAML, вбудована підтримка прив'язки даних (Data Binding). WPF забезпечує чіткий поділ логіки та інтерфейсу, що спрощує супровід і тестування коду.

Шаблон проектування MVVM. Model-View-ViewModel є стандартним архітектурним шаблоном для WPF-застосунків. Він передбачає поділ на три шари: Model (дані та бізнес-логіка), View (інтерфейс у XAML) та ViewModel (посередник між Model і View, що реалізує логіку представлення). Завдяки MVVM Views не містять бізнес-логіки, а ViewModels можна тестувати незалежно від інтерфейсу.

Бібліотека CommunityToolkit.Mvvm. Офіційна бібліотека від Microsoft для спрощення реалізації MVVM. Забезпечує генерацію коду для властивостей (ObservableProperty), команд (RelayCommand) та валідації (ObservableValidator) через атрибути та source generators. Це суттєво скорочує обсяг шаблонного коду.

MaterialDesignInXamlToolkit. Бібліотека компонентів, що реалізує специфікацію Material Design від Google для WPF. Забезпечує сучасний зовнішній вигляд елементів керування (TextBox, ComboBox, DataGrid, Chip тощо), підтримку тем і кольорових схем. Дозволяє створити привабливий і інтуїтивний інтерфейс без додаткових витрат на дизайн.

SQLite та Entity Framework Core 9. Для зберігання даних обрано SQLite — легковагу реляційну СУБД, що зберігає всю базу даних в одному

файлі. Основні переваги: не потребує окремого серверного процесу, проста у розгортанні (файл копіюється разом з програмою), безкоштовна, підтримує транзакції та відповідає стандарту SQL. Entity Framework Core 9 — це ORM (Object-Relational Mapper), що дозволяє працювати з базою даних через об'єктну модель C#, не пишучи SQL-запити вручну.

Порівняння варіантів СУБД для реалізації системи наведено у таблиці 1.3.

Таблиця 1.3 — Порівняння варіантів СУБД

Критерій	SQLite	SQL Server	PostgreSQL
Потребує сервера	Ні	Так	Так
Розгортання	Просте	Складне	Складне
Безкоштовність	Так	Частково	Так
Продуктивність (малий обсяг)	Висока	Висока	Висока
Підтримка EF Core	Так	Так	Так
Розмір файлу бібліотеки	< 1 МБ	> 100 МБ	> 50 МБ

Як видно з таблиці 1.3, SQLite є оптимальним вибором для автономного десктопного застосунку з невеликим обсягом даних (до кількох тисяч записів), яким є система обліку гаражного кооперативу.

Висновки до розділу 1

У першому розділі проведено всебічний аналіз предметної області та обґрунтовано проектні рішення. Основні результати:

1) Визначено специфіку гаражного кооперативу як об'єкта автоматизації: ключові бізнес-процеси, учасників, вимоги до обліку авансових платежів і переплати.
2) Проаналізовано існуючі програмні аналоги (TourManager, 1С:Кооператив, Excel) та встановлено, що жоден з них не задовольняє повною мірою потребам гаражного кооперативу.
3) Сформульовано 12 функціональних і нефункціональних вимог до системи.
4) Розроблено діаграму варіантів використання, що відображає взаємодію двох ролей користувачів з функціями системи.
5) Обґрунтовано вибір технологічного стеку: C# .NET 9, WPF, MVVM, CommunityToolkit.Mvvm, MaterialDesignInXamlToolkit, SQLite, Entity Framework Core 9.

Отримані результати є підставою для переходу до етапу проектування архітектури системи, що розглядається у наступному розділі.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Архітектура системи

Архітектура інформаційної системи визначає загальну організацію програмного забезпечення: поділ на шари, принципи їхньої взаємодії та розподіл відповідальності між компонентами. Правильно обрана архітектура є запорукою зрозумілості коду, можливості його тестування та подальшого розширення.

Для розробки системи обліку гаражного кооперативу обрано архітектурний шаблон MVVM (Model-View-ViewModel), що є стандартом де-факто для WPF-застосунків. Архітектура системи представлена на рисунку 2.1.

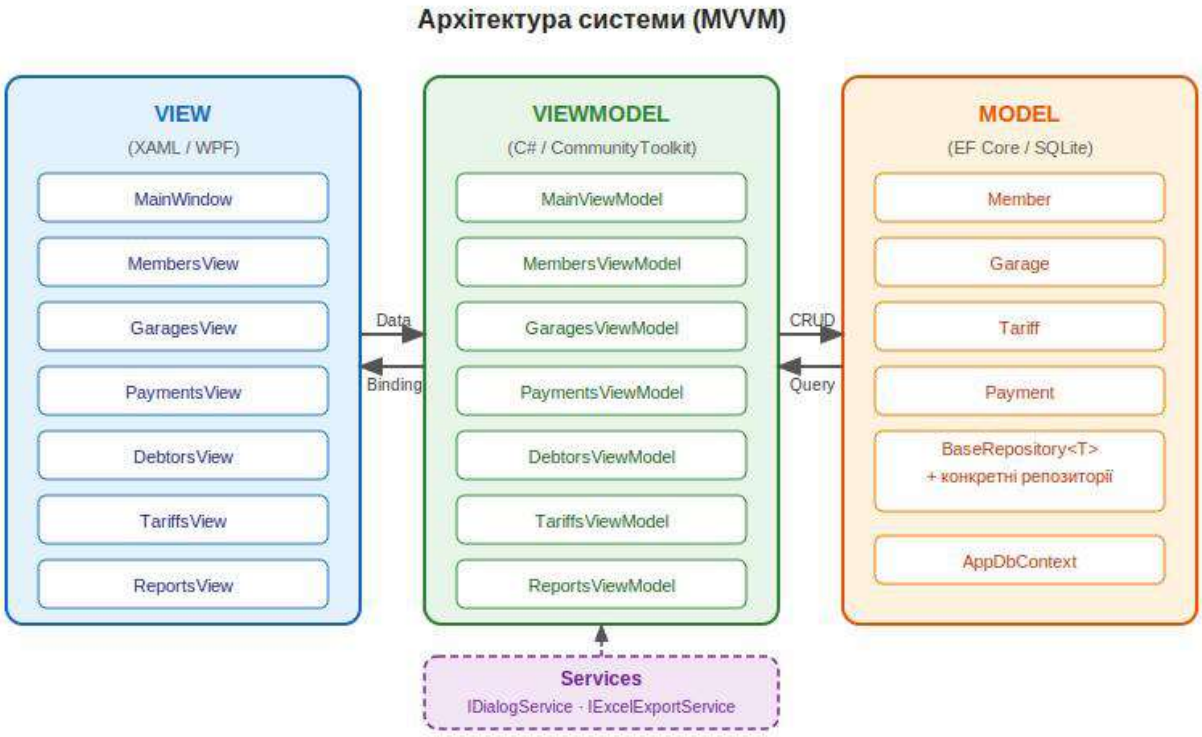

Рисунок 2.1 — Загальна архітектура інформаційної системи (MVVM)

Шаблон MVVM передбачає поділ на три основних шари, кожен з яких має чітко визначену відповідальність. Детальний опис компонентів кожного шару системи наведено у таблиці 2.5.

Таблиця 2.5 — Компоненти архітектури MVVM системи

Шар	Компоненти	Відповідальність
Model	Member, Garage, Tariff, Payment, AppDbContext, BaseRepository<T>	Зберігання даних, правила предметної області, взаємодія з SQLite через EF Core
View	MainWindow, MembersView, GaragesView, PaymentsView, DebtorsView, TariffsView, ReportsView, EditMemberWindow, EditPaymentWindow, …	Відображення даних у XAML; не містить логіки; прив'язується до ViewModel через DataBinding
ViewModel	MainViewModel, MembersViewModel, GaragesViewModel, PaymentsViewModel, DebtorsViewModel, TariffsViewModel, ReportsViewModel, EditMemberViewModel, …	Логіка представлення, команди, валідація, форматування даних; реалізує INotifyPropertyChanged через ObservableObject
Services	IDialogService, IExcelExportService	Сервіси для відображення діалогів та експорту, що абстрагують View від ViewModel
Helpers	Converters, EnumExtensions	Конвертери значень для XAML, розширення для enum

Взаємодія шарів організована за такими принципами:

1) View знає лише про ViewModel — прив'язується до її властивостей і команд через механізм Data Binding у XAML; не містить жодної бізнес-логіки.
2) ViewModel знає про Model, але не знає про View — отримує дані через репозиторії, підготовляє їх для відображення та реагує на команди користувача.
3) Model не знає ні про View, ні про ViewModel — містить лише опис даних та правила предметної області.
4) Services забезпечують слабку зв'язаність між ViewModel і системними функціями (діалоги, файловий ввід-вивід) через інтерфейси.

Така організація дозволяє замінювати або тестувати будь-який шар незалежно від інших. Зокрема, ViewModels можна тестувати без запуску

графічного інтерфейсу, а репозиторії — замінити заглушками (mock) під час юніт-тестування.

Доступ до бази даних реалізований через патерн Repository: для кожної сутності (Member, Garage, Tariff, Payment) визначено відповідний інтерфейс репозиторію (IMemberRepository, IGarageRepository тощо). Конкретні реалізації успадковуються від загального BaseRepository<T>, що містить стандартні CRUD-операції. Реєстрація залежностей здійснюється через вбудований контейнер Microsoft.Extensions.DependencyInjection, що гарантує єдиний екземпляр AppDbContext протягом сесії застосунку.

2.2 Проектування бази даних

База даних системи побудована на основі реляційної моделі та реалізована засобами SQLite. Проектування виконувалося відповідно до третьої нормальної форми (3НФ): усунено транзитивні залежності, кожен атрибут залежить лише від первинного ключа своєї таблиці.

Структура бази даних включає чотири основні таблиці: Members (члени кооперативу), Garages (гаражні бокси), Tariffs (тарифи та види внесків), Payments (платежі). Схема бази даних у вигляді ER-діаграми наведена на рисунку 2.2.

Рисунок 2.2 — ER-діаграма (сутність–зв'язок) бази даних

Зв'язки між таблицями:

– Members ↔ Garages: один член може мати кілька гаражів (OwnerId → Members.Id), один гараж належить не більш ніж одному члену (зв'язок один-до-багатьох);
– Members ↔ Payments: один член може мати багато платежів (MemberId → Members.Id, зв'язок один-до-багатьох);
– Tariffs ↔ Payments: кожен платіж прив'язаний до одного тарифу (TariffId → Tariffs.Id, зв'язок один-до-багатьох).

Детальний опис структури кожної таблиці наведено нижче.

Таблиця 2.1 — Структура таблиці Members (члени кооперативу)

Поле	Тип даних	Обмеження	Опис
Id	INTEGER	PK, AI	Унікальний ідентифікатор члена
FullName	TEXT	NOT NULL	Прізвище, ім'я, по батькові
BirthDate	TEXT	NOT NULL	Дата народження (ISO 8601)
Passport	TEXT		Серія та номер паспорта
TaxNumber	TEXT		Ідентифікаційний номер платника податків
Address	TEXT		Адреса проживання
Phone	TEXT		Контактний телефон
Email	TEXT		Адреса електронної пошти
JoinDate	TEXT	NOT NULL	Дата вступу до кооперативу
Status	INTEGER	NOT NULL	Статус члена (0 — активний, 1 — неактивний)
Notes	TEXT		Додаткові примітки

Таблиця 2.2 — Структура таблиці Garages (гаражні бокси)

Поле	Тип даних	Обмеження	Опис
Id	INTEGER	PK, AI	Унікальний ідентифікатор гаража
Number	TEXT	NOT NULL	Номер боксу (напр. A-01)
Row	TEXT	NOT NULL	Ряд (A, B, C, D)
Area	REAL	NOT NULL	Площа у квадратних метрах
Type	INTEGER	NOT NULL	Тип конструкції (0 — цегла, 1 — метал, 2 — ракушняк)
OwnerId	INTEGER	FK → Members.Id	Ідентифікатор власника (може бути NULL)
Status	INTEGER	NOT NULL	Статус (0 — зайнятий, 1 — вільний)
Notes	TEXT		Примітки

Таблиця 2.3 — Структура таблиці Tariffs (тарифи та внески)

Поле	Тип даних	Обмеження	Опис
Id	INTEGER	PK, AI	Унікальний ідентифікатор тарифу
Type	INTEGER	NOT NULL	Вид внеску (0–4): членський, електрика, охорона, цільовий, інше
Amount	REAL	NOT NULL	Сума внеску
Periodicity	INTEGER	NOT NULL	Periodicity (0 — щомісяця, 1 — щоквартально, 2 — щорічно)
Notes	TEXT		Примітки до тарифу

Таблиця 2.4 — Структура таблиці Payments (платежі)

Поле	Тип даних	Обмеження	Опис
Id	INTEGER	PK, AI	Унікальний ідентифікатор платежу
Date	TEXT	NOT NULL	Дата проведення платежу
MemberId	INTEGER	FK → Members.Id	Ідентифікатор члена-платника
TariffId	INTEGER	FK → Tariffs.Id	Ідентифікатор тарифу
Amount	REAL	NOT NULL	Сплачена сума
Period	TEXT	NOT NULL	Розрахунковий період (формат YYYY-MM)
PaymentMethod	INTEGER	NOT NULL	Спосіб оплати (0 — готівка, 1 — картка, 2 — переказ)
Notes	TEXT		Примітки до платежу

Цілісність даних забезпечується на двох рівнях. На рівні бази даних: зовнішні ключі (FOREIGN KEY) з каскадним видаленням відповідних записів та обмеження NOT NULL на обов'язкові поля. На рівні застосунку: валідація вхідних даних через атрибути DataAnnotations у ViewModels перед записом до бази.

2.3 Діаграма класів

Діаграма класів відображає статичну структуру системи: основні класи, їхні атрибути, методи та зв'язки між ними. На рисунку 2.3 представлено спрощену діаграму класів для ключових компонентів системи.

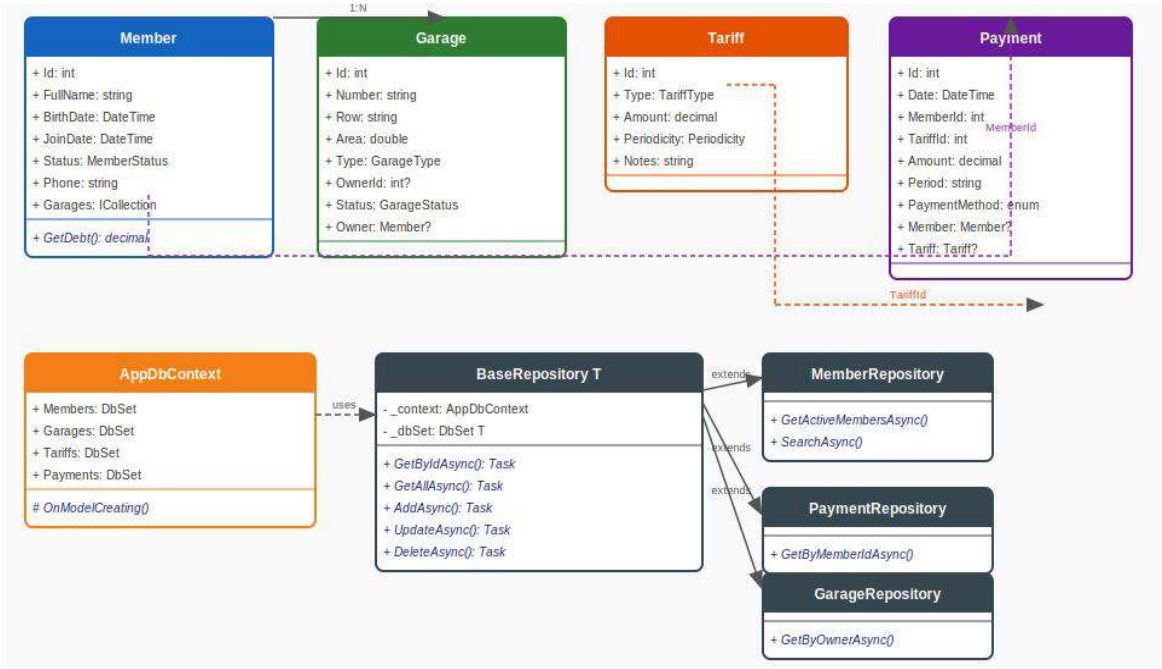

Рисунок 2.3 — Спрощена діаграма класів інформаційної системи

Ключові класи системи та їхні взаємозв'язки:

Клас Member реалізує модель даних члена кооперативу. Містить властивості: Id, FullName, BirthDate, Passport, TaxNumber, Address, Phone, Email, JoinDate, Status, Notes. Навігаційна властивість Garages (ICollection<Garage>) забезпечує доступ до гаражів члена через EF Core.

Клас Garage містить: Id, Number, Row, Area, Type, OwnerId, Status, Notes. Навігаційна властивість Owner (Member?) пов'язує гараж з його власником.

Клас Tariff описує вид і суму внеску: Id, Type (TariffType), Amount, Periodicity, Notes.

Клас Payment фіксує факт оплати: Id, Date, MemberId, Member, TariffId, Tariff, Amount, Period, PaymentMethod, Notes. Навігаційні властивості Member та Tariff забезпечують зручний доступ до пов'язаних сутностей без додаткових запитів.

Клас AppDbContext (успадковує DbContext з EF Core) визначає набори даних DbSet<Member>, DbSet<Garage>, DbSet<Tariff>, DbSet<Payment> та конфігурує зв'язки між ними через Fluent API у методі OnModelCreating.

Клас BaseRepository<T> є узагальненою реалізацією патерну Repository. Визначає методи GetByIdAsync, GetAllAsync, FindAsync, AddAsync, UpdateAsync, DeleteAsync. Успадковується конкретними класами MemberRepository, GarageRepository, TariffRepository, PaymentRepository.

2.4 Діаграма послідовності основних сценаріїв

Для ілюстрації динамічної поведінки системи розроблено діаграми послідовності для двох ключових сценаріїв: реєстрації нового платежу та розрахунку заборгованості члена.

Сценарій 1. Реєстрація нового платежу (рис. 2.4).

Послідовність дій при реєстрації платежу касиром:

1) Касир натискає кнопку "Додати платіж" у PaymentsView.
2) PaymentsViewModel відкриває EditPaymentWindow з порожньою формою, передаючи списки членів і тарифів.
3) EditPaymentViewModel ініціалізується: зв'язує ComboBox з AvailableMembers та AvailableTariffs.
4) Касир обирає члена, тариф, вводить суму та дату, натискає "Зберегти".
5) EditPaymentViewModel виконує валідацію: перевіряє, що сума > 0, обраний член і тариф, формат дати.
6) У разі успіху — викликає callback, PaymentsViewModel отримує об'єкт Payment; з нього обнуляються навігаційні властивості (Member = null, Tariff = null), залишаються лише FK-поля.

7) PaymentRepository.AddAsync зберігає запис у SQLite; після збереження запис відсутній у ChangeTracker.
8) PaymentsViewModel викликає LoadDataAsync, оновлює колекцію — View відображає новий рядок.

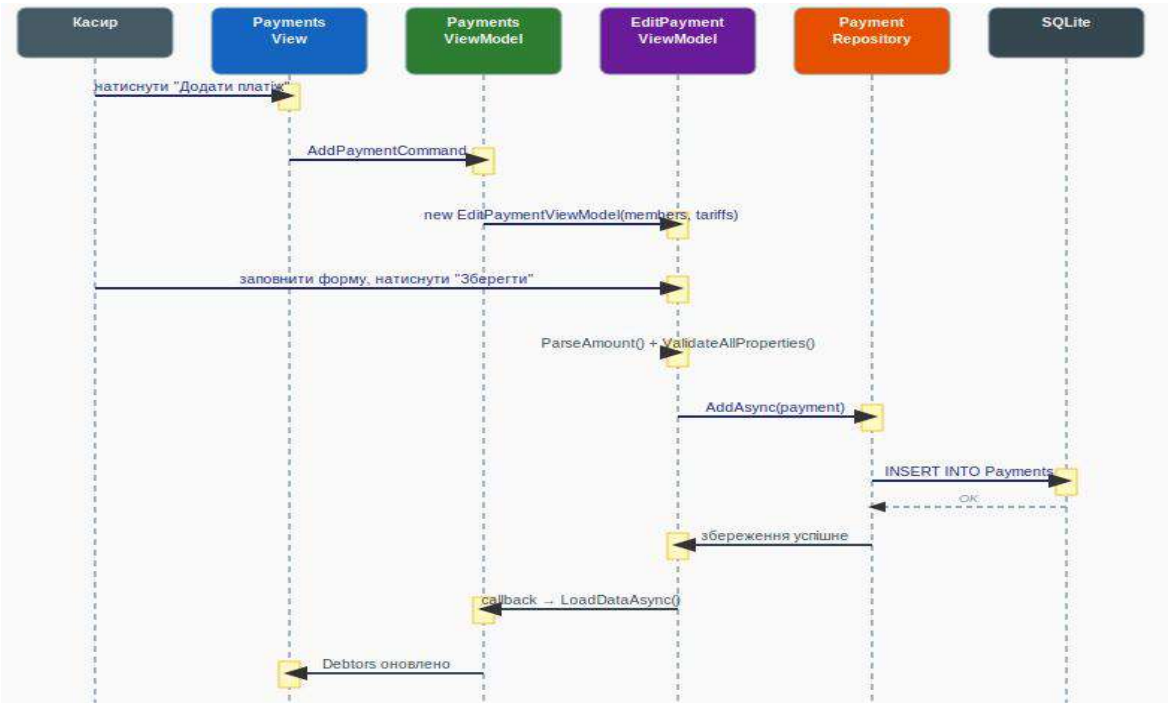

Рисунок 2.4 — Діаграма послідовності: реєстрація нового платежу

Сценарій 2. Розрахунок заборгованості (рис. 2.5).

Алгоритм розрахунку заборгованості для кожного активного члена:

1) DebtorsViewModel завантажує список активних членів через IMemberRepository.GetActiveMembersAsync.
2) Для кожного члена завантажуються всі його платежі через IPaymentRepository.GetByMemberIdAsync.
3) Завантажується список усіх тарифів через ITariffRepository.GetAllAsync.
4) Для кожного тарифу розраховується кількість місяців, що минули з дати вступу члена до поточного місяця (monthsDue).
5) Сума нарахованих коштів: totalDue = monthsDue × tariff.Amount.
6) Сума фактично сплачених коштів за цим тарифом: totalPaid = сума Amount усіх платежів члена з даним TariffId (без фільтра по даті — враховуються авансові платежі).

7) Баланс: balance = totalPaid − totalDue. Якщо balance < 0 — борг (|balance|), якщо balance > 0 — переплата.
8) Результат групується в DebtorViewModel та відображається у DebtorsView.

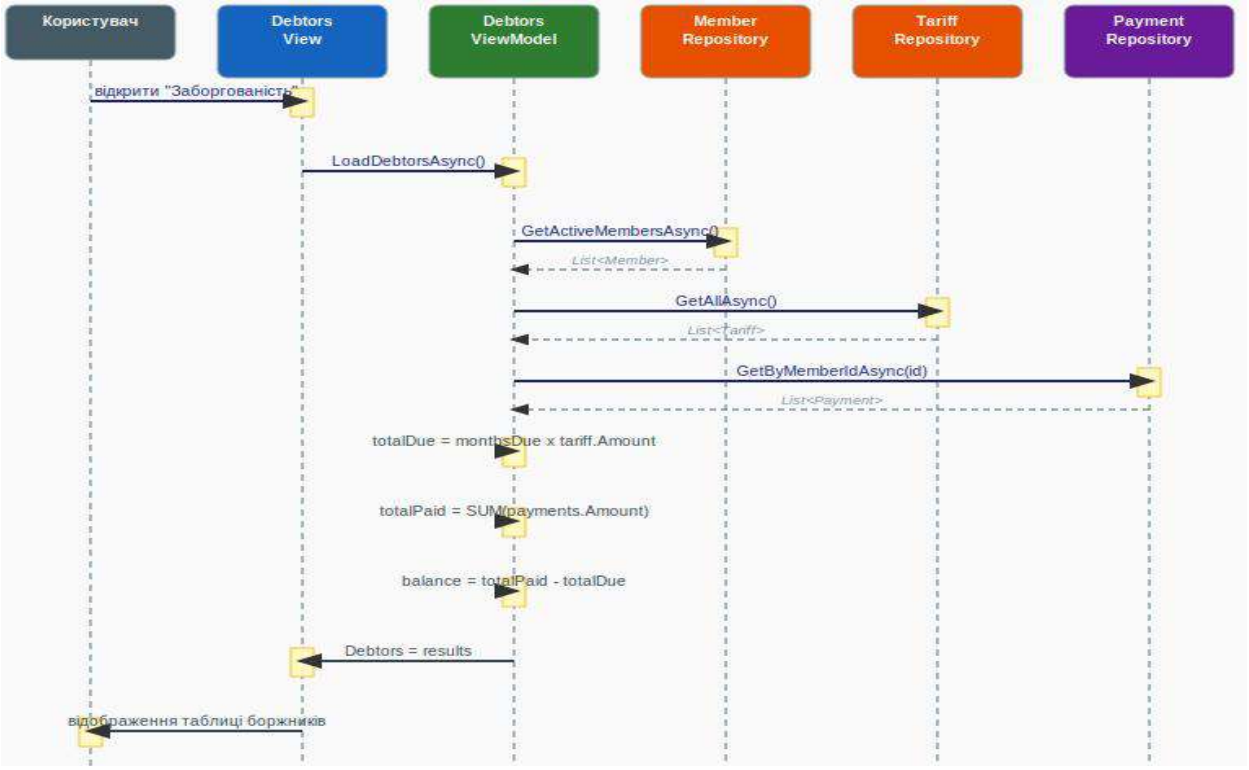

Рисунок 2.5 — Діаграма послідовності: розрахунок заборгованості

Формула розрахунку місяців нарахування для члена кооперативу:

monthsDue = (рікПоточний − рікВступу) × 12 + (місяцьПоточний − місяцьВступу) + 1

Дана формула враховує, що у місяці вступу член вже зобов'язаний сплатити внесок за повний місяць, тому до різниці місяців додається одиниця.

2.5 Діаграми станів

Для формального опису життєвого циклу ключових сутностей розроблено діаграми станів.

Діаграма станів члена кооперативу (рис. 2.6). Член може перебувати в одному з двох станів: Активний (Active) та Неактивний (Inactive). Перехід зі стану "Активний" до "Неактивний" здійснюється адміністратором при добровільному виході члена або його виключенні. Зворотній перехід

можливий при поновленні членства. Лише активні члени враховуються при розрахунку заборгованості та відображаються у звіті боржників.

Рисунок 2.6 — Діаграма станів об'єкта "Член кооперативу"

Діаграма станів гаражного боксу (рис. 2.7). Гараж може перебувати у стані "Зайнятий" (Occupied) або "Вільний" (Free). Перехід до стану "Зайнятий" відбувається при прив'язці члена (OwnerId ≠ NULL). Перехід до стану "Вільний" — при від'єднанні власника (OwnerId = NULL) або виключенні члена з кооперативу. Вільні гаражі відображаються у довіднику та можуть бути призначені новому члену.

Рисунок 2.7 — Діаграма станів об'єкта "Гаражний бокс"

Діаграма станів платежу (рис. 2.8). Після створення платіж одразу переходить у стан "Зараховано" і зберігається в базі даних. Платіж може бути відредагований (стан "Редагується") або видалений. Видалений платіж більше не враховується у розрахунках. Стан "Зараховано" є кінцевим для нормального функціонування: зміна суми чи тарифу негайно впливає на баланс відповідного члена.

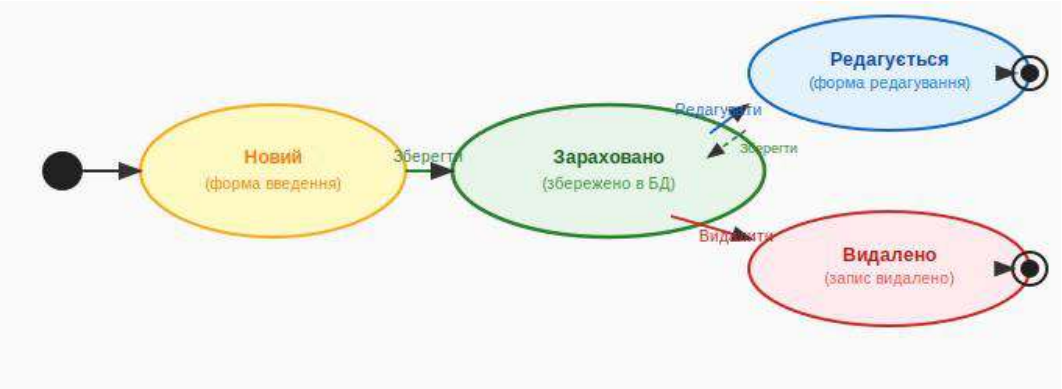

Рисунок 2.8 — Діаграма станів об'єкта "Платіж"

2.6 Проектування інтерфейсу користувача

Інтерфейс системи побудований відповідно до специфікації Material Design, що забезпечує сучасний зовнішній вигляд і послідовність взаємодії з елементами керування. Навігація між розділами здійснюється через бічне меню (Navigation Drawer) у головному вікні MainWindow.

Загальна навігаційна структура системи з описом екранів наведена у таблиці 2.6.

Таблиця 2.6 — Навігаційна структура та зміст екранів системи

Екран	Доступ	Основні елементи
Головна панель	Всі ролі	KPI-картки (члени, гаражі, борг, надходження), топ-5 боржників, останній платіж
Члени кооперативу	Всі ролі	Таблиця членів, пошук, фільтр статусу, кнопки Додати/Редагувати/Видалити
Довідник гаражів	Всі ролі	Таблиця гаражів, пошук, фільтр статусу, прив'язка до власника
Реєстр платежів	Всі ролі	Таблиця платежів, фільтр за датою та членом, реєстрація нового платежу
Заборгованість	Всі ролі	Таблиця боржників із кольоровими чіпами (борг/переплата), пошук, фільтр
Тарифи / Внески	Адміністратор	Таблиця тарифів, редагування сум і видів внесків
Звіти	Всі ролі	Вибір типу звіту, фільтр дат, таблиця результатів, експорт до Excel

Головна панель (Dashboard) є початковим екраном після запуску системи (рис. 2.9). Вона відображає ключові показники у вигляді KPI-карток: кількість активних членів, зайнятих гаражів, загальний борг по кооперативу та

надходження поточного місяця. Нижче розміщено таблицю п'яти найбільших боржників та інформацію про останній зареєстрований платіж.

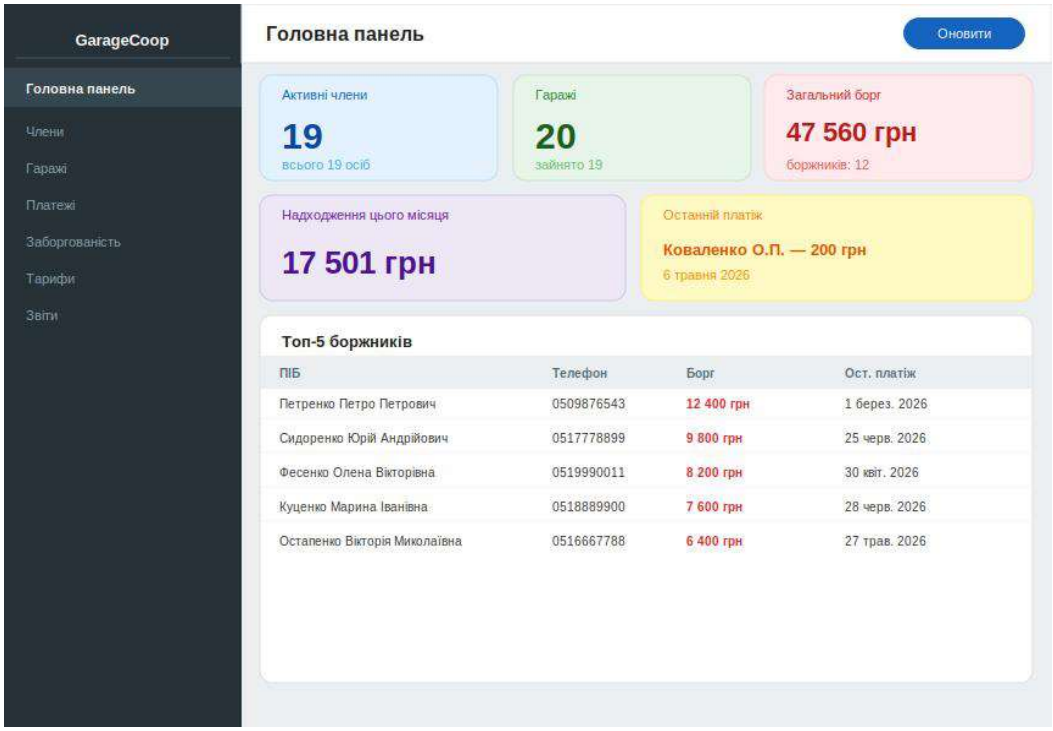

Рисунок 2.9 — Проектний макет екрану "Головна панель"

Екран "Члени кооперативу" (рис. 2.10) містить таблицю DataGrid з усіма членами, рядок пошуку (фільтрація за ПІБ або телефоном), перемикач фільтру статусу та кнопки операцій CRUD. При натисканні "Додати" або "Редагувати" відкривається модальне вікно EditMemberWindow з формою введення реквізитів члена та валідацією всіх обов'язкових полів.

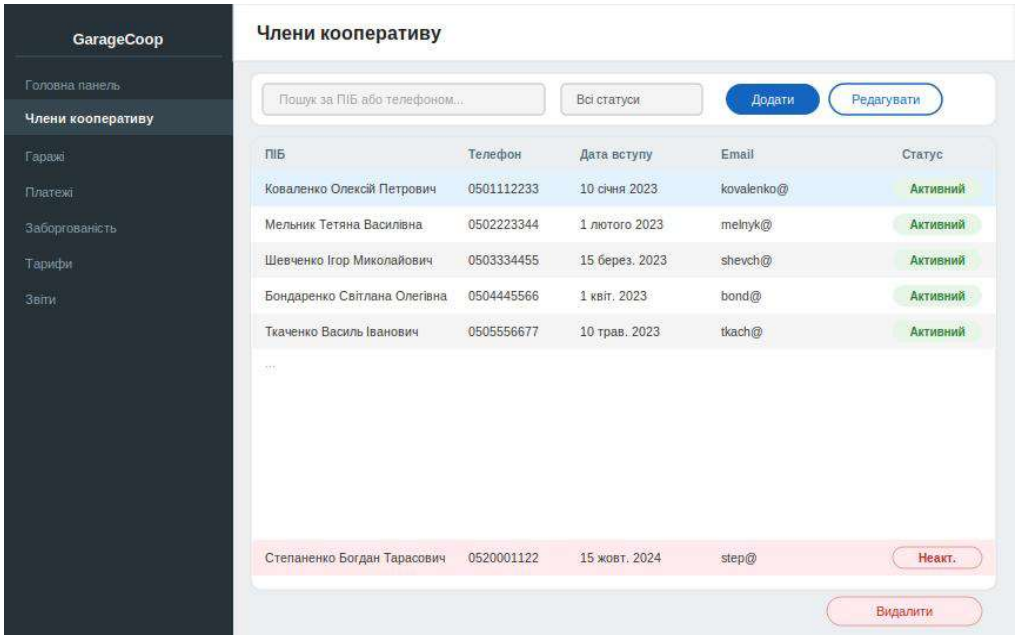

Рисунок 2.10 — Проектний макет екрану "Члени кооперативу"

Екран "Заборгованість членів" (рис. 2.11) відображає розраховану заборгованість у вигляді таблиці. Стовпець "Деталі боргу" містить кольорові чіпи: червоні — для боргу за кожним тарифом, зелені — для переплати. При наведенні курсора на чіп відображається підказка з деталями розрахунку (кількість місяців, ставка, загальна сума). Кнопка "Лише з боргом" приховує членів без заборгованості.

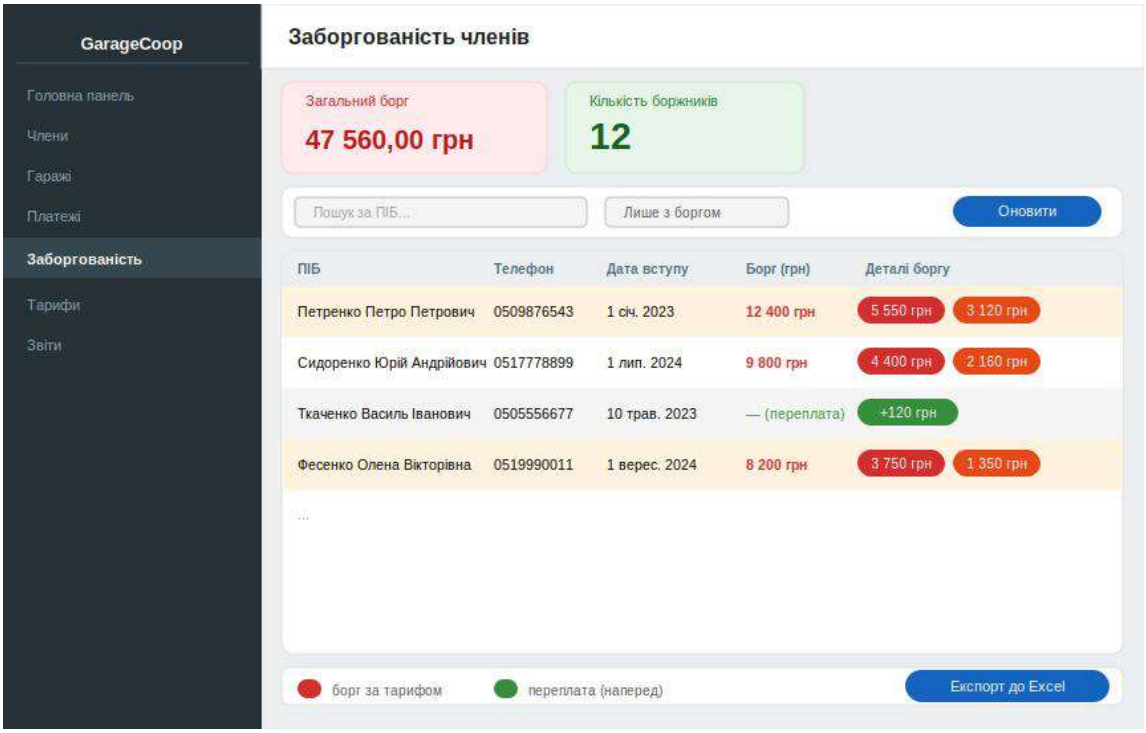

Рисунок 2.11 — Проектний макет екрану "Заборгованість членів"

2.7 Проектування безпеки даних та контролю доступу

Безпека інформаційної системи забезпечується на кількох рівнях. Оскільки система є локальним десктопним застосунком без мережевої взаємодії, основні загрози пов'язані з несанкціонованим доступом до файлу бази даних та помилками введення даних.

Автентифікація користувачів. Система підтримує два рівні доступу: Адміністратор (повний доступ до всіх функцій, включаючи управління тарифами та видалення записів) та Касир (реєстрація платежів, перегляд списків і звітів). При запуску відображається вікно входу. Пароль зберігається у захищеному вигляді з використанням алгоритму BCrypt (бібліотека BCrypt.Net-Next), що унеможливлює відновлення оригінального паролю з хешу.

Захист файлу бази даних. Файл GarageCoop.db зберігається у системній папці %AppData%\GarageCoop, доступ до якої захищений правами облікового запису Windows. Файл є бінарним SQLite-форматом, що унеможливлює його просте читання без спеціальних інструментів.

Цілісність даних при збереженні. Усі операції запису до бази даних виконуються в рамках транзакцій EF Core. При виникненні помилки транзакція відкочується, дані залишаються у попередньому консистентному стані. Після збереження запис відокремлюється від DbContext (EntityState.Detached), що запобігає конфліктам відстеження при повторному завантаженні даних.

Валідація вхідних даних. Усі форми введення даних оснащені клієнтською валідацією засобами ObservableValidator та атрибутів DataAnnotations. Обов'язкові поля позначаються символом "*" та підсвічуються червоним при помилці. Числові поля (суми, площа) реалізовані як текстові з ручним парсингом, що забезпечує коректну обробку як крапки, так і коми як десяткового розділювача незалежно від системних налаштувань локалізації.

Захист від помилок при видаленні. Перед видаленням будь-якого запису система відображає діалог підтвердження. Видалення члена перевіряється на наявність пов'язаних платежів і гаражів — у разі наявності виводиться попередження із пропозицією спочатку архівувати (деактивувати) члена замість повного видалення.

Висновки до розділу 2

У другому розділі здійснено повне проектування інформаційної системи обліку та управління гаражним кооперативом. Основні результати:

1) Обґрунтовано та детально описано архітектуру системи на основі шаблону MVVM: визначено компоненти кожного шару (Model, View, ViewModel, Services, Helpers) та принципи їхньої взаємодії.
2) Спроектовано реляційну базу даних у третій нормальній формі: чотири таблиці (Members, Garages, Tariffs, Payments) з визначеними типами полів, обмеженнями цілісності та зовнішніми ключами.
3) Побудовано діаграму класів, що відображає ієрархію моделей, BaseRepository<T> та структуру сервісів.
4) Розроблено діаграми послідовності для двох ключових сценаріїв: реєстрації платежу та розрахунку заборгованості; формалізовано алгоритм розрахунку з математичним формулюванням.
5) Побудовано діаграми станів для трьох сутностей: члена кооперативу, гаражного боксу та платежу.
6) Спроектовано навігаційну структуру інтерфейсу (7 екранів), визначено призначення кожного екрану та описано ключові елементи керування.
7) Визначено механізми забезпечення безпеки даних: BCrypt-хешування паролів, транзакційне збереження, валідація введення, підтвердження видалення.

Результати проектування є вичерпною основою для реалізації системи, що розглядається у третьому розділі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Структура проєкту

Проєкт розробленої інформаційної системи реалізовано як десктопний WPF-застосунок на платформі .NET 9 з використанням мови програмування C#. Код організований відповідно до архітектурного шаблону MVVM: кожен шар знаходиться у власній папці з чіткою відповідальністю. Детальна структура проєкту наведена у таблиці 3.1.

Таблиця 3.1 — Структура проєкту GarageCoop

Папка / файл	Тип	Призначення
Models/	Папка	Моделі даних (сутності EF Core)
Member.cs	C# клас	Сутність члена кооперативу
Garage.cs	C# клас	Сутність гаражного боксу
Tariff.cs	C# клас	Сутність тарифу / внеску
Payment.cs	C# клас	Сутність платежу
Enums.cs	C# enum	MemberStatus, GarageType, TariffType, PaymentMethod, Periodicity
Data/	Папка	Доступ до даних (EF Core + репозиторії)
AppDbContext.cs	DbContext	Контекст бази даних SQLite
Repositories/	Папка	Репозиторії для кожної сутності
Migrations/	Папка	Файли міграцій EF Core
ViewModels/	Папка	ViewModels (MVVM); логіка подання
Base/BaseViewModel.cs	C# клас	Базовий клас з IsBusy, DialogService
MainViewModel.cs	C# клас	Навігація між розділами
MembersViewModel.cs	C# клас	Управління списком членів
GaragesViewModel.cs	C# клас	Управління довідником гаражів
PaymentsViewModel.cs	C# клас	Реєстрація та перегляд платежів
DebtorsViewModel.cs	C# клас	Розрахунок заборгованості
TariffsViewModel.cs	C# клас	Управління тарифами
ReportsViewModel.cs	C# клас	Формування та експорт звітів
Edit*/	C# класи	ViewModels для вікон редагування

Views/	Папка	Файли XAML-розмітки (UI)
MainWindow.xaml	XAML	Головне вікно з бічним меню
*View.xaml	XAML	Екрани для кожного розділу (7 шт.)
Edit*Window.xaml	XAML	Модальні вікна редагування (4 шт.)
Services/	Папка	Сервіси (діалоги, Excel-експорт)
Helpers/	Папка	Конвертери значень, розширення enum
App.xaml / App.xaml.cs	XAML + C#	Точка входу, DI-контейнер, культура uk-UA

Точкою входу є файл App.xaml.cs, у якому виконуються три ключових налаштування:

1) реєстрація залежностей у контейнері Microsoft.Extensions.DependencyInjection (всі репозиторії, ViewModels та сервіси);
2) встановлення культури uk-UA для коректного відображення дат (місяць, день тижня українською мовою) та формату чисел;
3) перевизначення FrameworkElement.LanguageProperty для WPF-компонентів (зокрема DatePicker).

Реєстрація залежностей дозволяє впроваджувати їх через конструктор, що забезпечує слабку зв'язаність компонентів і можливість легкої заміни реалізації без змін у коді клієнтів. Кожен ViewModel отримує необхідні репозиторії та сервіси автоматично через DI-контейнер.

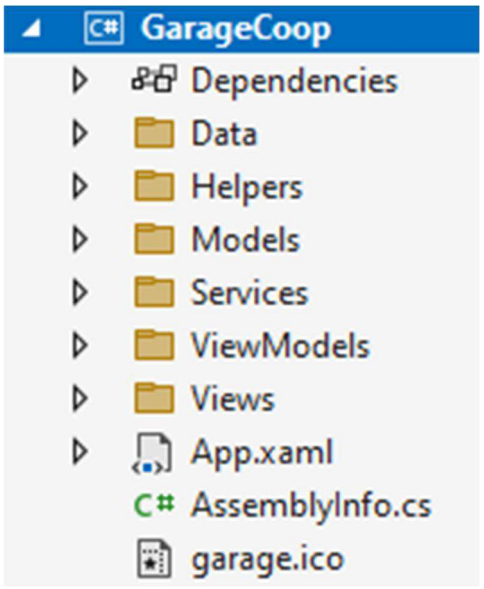

Рисунок 3.1 — Структура проєкту GarageCoop у Solution Explorer

3.2 Реалізація моделей даних та бази даних

Моделі даних реалізовані як звичайні C#-класи (POCO — Plain Old CLR Objects) з навігаційними властивостями для зв'язків між сутностями. Кожна сутність містить властивість Id типу int, що є первинним ключем і автоматично генерується базою даних.

Клас Payment є центральною сутністю системи, оскільки через нього проходить весь фінансовий облік. Його реалізацію наведено нижче:

```
public class Payment
{
    public int           Id            { get; set; }
    public DateTime      Date          { get; set; }
    public int           MemberId      { get; set; }
    public Member?       Member        { get; set; }
    public int           TariffId      { get; set; }
    public Tariff?       Tariff        { get; set; }
    public decimal       Amount        { get; set; }
    public string        Period        { get; set; } = string.Empty;
    public PaymentMethod PaymentMethod { get; set; }
    public string        Notes         { get; set; } = string.Empty;
}
```

Клас AppDbContext налаштовує зв'язки між сутностями через Fluent API та встановлює каскадне видалення платежів при видаленні члена кооперативу:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Payment>()
        .HasOne(p => p.Member)
        .WithMany()
        .HasForeignKey(p => p.MemberId)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<Garage>()
        .HasOne(g => g.Owner)
        .WithMany(m => m.Garages)
        .HasForeignKey(g => g.OwnerId)
        .OnDelete(DeleteBehavior.SetNull);
}
```

База даних ініціалізується при першому запуску застосунку через механізм міграцій EF Core. Метод Database.EnsureCreatedAsync() створює файл GarageCoop.db у системній папці %AppData%\GarageCoop, якщо він ще не існує. Всі подальші зміни схеми застосовуються через файли міграцій, що зберігаються у папці Data/Migrations.

Базовий репозиторій BaseRepository<T> реалізує загальні CRUD-операції для будь-якої сутності. Особливу увагу приділено методу UpdateAsync, що вирішує проблему конфлікту відстеження сутностей (entity tracking) в EF Core:

```
public virtual async Task UpdateAsync(T entity)
{
    var keyValues = GetKeyValues(entity);
    var tracked   = await _dbSet.FindAsync(keyValues);
    if (tracked != null && !ReferenceEquals(tracked, entity))
        _context.Entry(tracked).CurrentValues.SetValues(entity);
    else
        _dbSet.Update(entity);
    await _context.SaveChangesAsync();
}
```

Такий підхід гарантує, що при оновленні сутності EF Core не намагатиметься повторно відстежити вже завантажений екземпляр, що призводило б до виключення InvalidOperationException з повідомленням про конфлікт ключів.

3.3 Реалізація основних модулів бізнес-логіки

3.3.1 Модуль обліку членів кооперативу

Модуль обліку членів реалізовано у класах MembersViewModel та EditMemberViewModel. MembersViewModel відповідає за завантаження і фільтрацію списку членів, а EditMemberViewModel — за валідацію і збереження окремого запису.

Фільтрація списку членів виконується на стороні клієнта (in-memory) після завантаження всіх записів з бази даних. Це дозволяє уникнути проблеми з кириличним пошуком у SQLite, де порядок сортування за замовчуванням може не підтримувати українські символи:

```
private void ApplyFilter()
{
    var all = _allMembers.AsEnumerable();
    if (!string.IsNullOrWhiteSpace(SearchText))
        all = all.Where(m =>
            m.FullName.Contains(SearchText,
                StringComparison.OrdinalIgnoreCase) ||
            m.Phone.Contains(SearchText));
    if (!ShowInactive)
        all = all.Where(m => m.Status == MemberStatus.Active);
    Members = new ObservableCollection<Member>(all.OrderBy(m => m.FullName));
}
```

Валідація форми введення даних члена реалізована через атрибути DataAnnotations у класі EditMemberViewModel, що успадковує ObservableValidator з бібліотеки CommunityToolkit.Mvvm. При кожній зміні властивості автоматично виконується перевірка та оновлюється стан кнопки "Зберегти":

```
[ObservableProperty]
[Required(ErrorMessage = "ПІБ є обов\'язковим")]
[MinLength(3, ErrorMessage = "ПІБ має бути від 3 до 150 символів")]
[MaxLength(150, ErrorMessage = "ПІБ має бути від 3 до 150 символів")]
private string _fullName = string.Empty;

public bool CanSave => !HasErrors;
```

3.3.2 Модуль обліку гаражів і тарифів

Модуль обліку гаражів дозволяє вести довідник гаражних боксів із прив'язкою до власника. При збереженні гаражу навігаційна властивість Owner обнуляється перед передачею до репозиторію, щоб уникнути конфлікту відстеження EF Core — використовуються лише зовнішні ключі (OwnerId):

```
savedGarage.Owner = null; // використовуємо лише OwnerId
await _garageRepository.AddAsync(savedGarage);
```

При відображенні у ComboBox редагування вибір власника здійснюється через пошук за Id у завантаженому списку — а не через порівняння посилань, яке в WPF не працює для різних екземплярів одного запису:

```
SelectedOwner = garage.OwnerId.HasValue
    ? AvailableMembers.FirstOrDefault(m => m.Id == garage.OwnerId.Value)
    : null;
```

Тарифи зберігаються незалежно від платежів. Кожен тариф визначає вид внеску, суму та періодичність. Система підтримує такі типи тарифів: членський внесок (Membership), електроенергія (Electricity), охорона (Security), цільовий збір (Target) та інше (Other).

3.3.3 Модуль реєстрації платежів

Ключовою особливістю реалізації модуля платежів є вирішення проблеми введення десяткових чисел в умовах культури uk-UA, де розділювачем цілої та дробової частини є кома. WPF за замовчуванням використовує системну культуру при конвертації значень TextBox, що призводило до помилок FormatException при введенні крапки як розділювача.

Обрано рішення на основі рядкового поля AmountText у ViewModel замість безпосереднього прив'язування decimal: поле приймає будь-який введений текст, а парсинг виконується лише в момент збереження з явним використанням InvariantCulture:

```
// ViewModel: рядкове поле без атрибутів валідації
[ObservableProperty]
private string _amountText = string.Empty;

// Парсинг при натисканні "Зберегти"
private decimal ParseAmount()
{
    var s = (AmountText ?? string.Empty).Trim().Replace(',', '.');
    return decimal.TryParse(s, NumberStyles.Any,
        CultureInfo.InvariantCulture, out var v) && v > 0 ? v : 0m;
}
```

Такий підхід дозволяє коректно обробляти введення як "5000", так і "5000,99" та "5000.99" — будь-який формат, зручний для касира.

При збереженні нового платежу навігаційні властивості обнуляються перед записом до бази даних, залишаючи лише зовнішні ключі MemberId і TariffId. Після збереження виконується повне перезавантаження списку, щоб нові записи відображалися з розгорнутими навігаційними властивостями (назва члена, тип тарифу):

```
saved.Member = null; // strip nav properties
saved.Tariff = null;
await _paymentRepository.AddAsync(saved);
await LoadDataAsync(); // перезавантаження з Include()
```

3.3.4 Алгоритм розрахунку заборгованості

Розрахунок заборгованості є ключовою бізнес-логікою системи. Алгоритм реалізовано у методі BuildDebtorAsync класу DebtorsViewModel. Для кожного активного члена і кожного тарифу виконується розрахунок балансу:

```
foreach (var tariff in tariffs)
{
    // Кількість місяців від вступу до поточного місяця включно
    var monthsDue = (now.Year  - member.JoinDate.Year)  * 12
                  + now.Month - member.JoinDate.Month + 1;
    var totalDue  = monthsDue * tariff.Amount;
    // Усі платежі за цим тарифом (без фільтру дати — враховує аванси)
    var totalPaid = payments
        .Where(p => p.TariffId == tariff.Id)
        .Sum(p => p.Amount);
    var balance = totalPaid - totalDue;
    // balance < 0 → борг  |  balance > 0 → переплата
}
```

Принципова відмінність від наївного підходу (підрахунок кількості транзакцій) полягає у тому, що враховуються фактичні суми платежів. Це дозволяє коректно обробляти три основних фінансових сценарії:

4) звичайна щомісячна оплата — сума балансу дорівнює нулю;
5) авансова оплата за кілька місяців наперед одним платежем — баланс позитивний (переплата), яка автоматично зараховується у рахунок майбутніх нарахувань;
6) часткова або пропущена оплата — баланс від'ємний (борг), відображається червоним чіпом у таблиці заборгованості.

Результати розрахунку групуються у колекцію об'єктів DebtorViewModel, кожен з яких містить список DebtDetailViewModel — по одному елементу на тариф. Це дозволяє відображати деталізований розбір боргу/переплати для кожного члена у вигляді кольорових чіпів.

3.4 Реалізація інтерфейсу користувача

Інтерфейс системи реалізовано засобами WPF з використанням бібліотеки MaterialDesignInXamlToolkit версії 5.x, що забезпечує компоненти в стилі Material Design. Навігація між розділами реалізована через ContentControl у головному вікні, яке керується властивістю CurrentView у MainViewModel.

Головне вікно MainWindow містить бічну навігаційну панель (Navigation Drawer) з кнопками переходу між розділами та область відображення поточного View. При переході навігаційна панель підсвічує активний пункт меню змінюючи стиль кнопки через DataTrigger:

```
<Style.Triggers>
  <DataTrigger Binding="{Binding IsSelected}" Value="True">
    <Setter Property="Background" Value="{StaticResource PrimaryHueLightBrush}"/>
    <Setter Property="FontWeight"  Value="Bold"/>
  </DataTrigger>
</Style.Triggers>
```

Усі числові поля суми та площі реалізовані як прості TextBox з прив'язкою до рядкового поля ViewModel (AmountText, AreaText). Такий підхід обраний для повного обходу системи конвертації WPF, яка викликає FormatException при введенні десяткових чисел у культурі uk-UA.

Відображення помилок валідації реалізовано через DataTrigger замість стандартного Validation.HasError, що дозволяє гнучко налаштовувати вигляд поля з помилкою без конфліктів зі стилями MaterialDesign:

```
<TextBox.Style>
  <Style TargetType="TextBox"
         BasedOn="{StaticResource MaterialDesignFloatingHintTextBox}">
    <Style.Triggers>
      <DataTrigger Binding="{Binding HasAmountError}" Value="True">
        <Setter Property="BorderBrush" Value="Red"/>
      </DataTrigger>
    </Style.Triggers>
  </Style>
</TextBox.Style>
```

Скриншот головного екрану системи після запуску з тестовими даними наведено на рисунку 3.2.

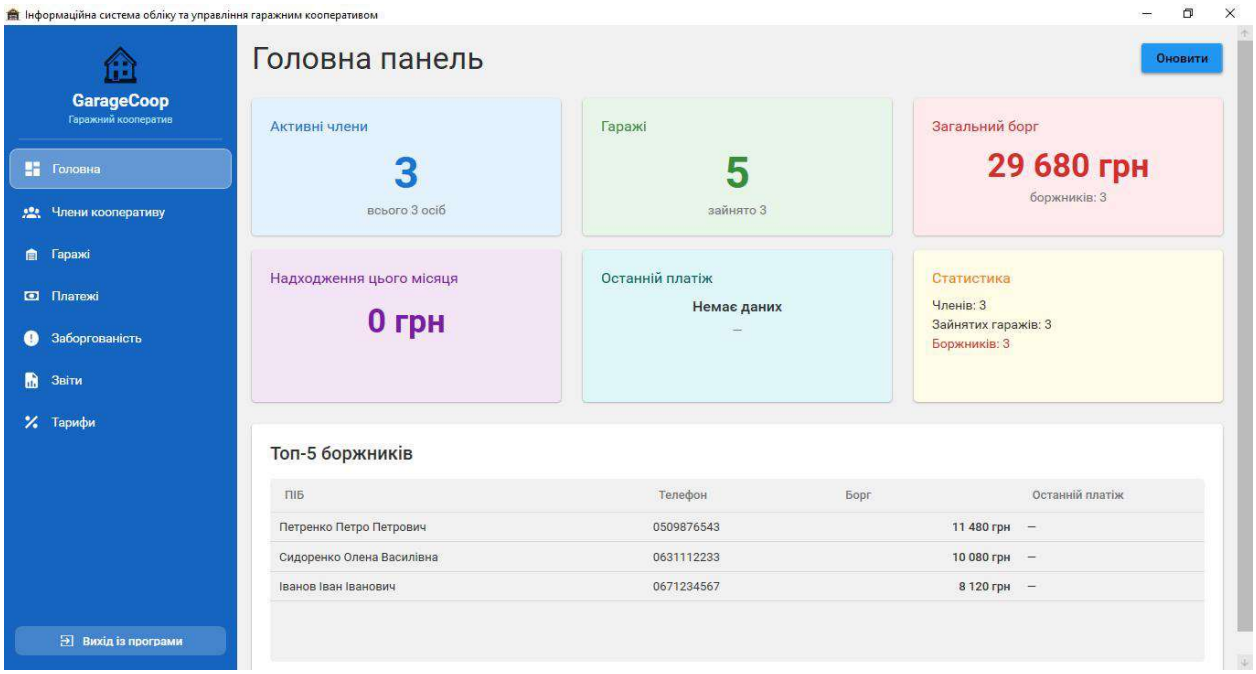

Рисунок 3.2 — Головна панель системи з KPI-картками та таблицею боржників

Екран реєстрації платежу (рис. 3.3) демонструє модальне вікно з усіма полями форми: дата, член, тариф, сума, період, спосіб оплати та примітки. Поле "Член кооперативу" є ComboBox з пошуком; вибраний елемент відновлюється при редагуванні через пошук за Id, а не порівняння посилань.

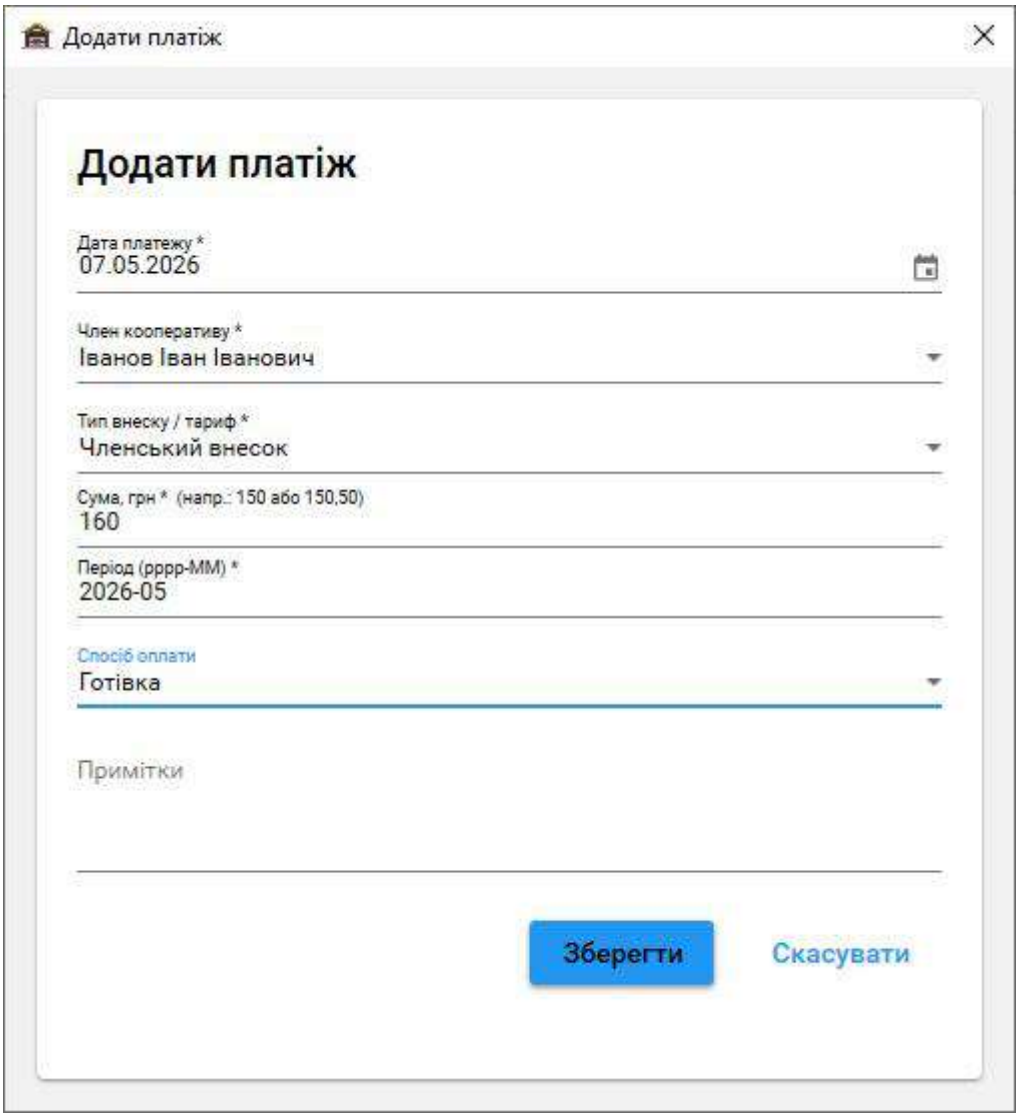

Рисунок 3.3 — Вікно реєстрації нового платежу з валідацією

Екран заборгованості (рис. 3.4) відображає KPI-картки із загальним боргом і кількістю боржників, а також деталізовану таблицю. У стовпці "Деталі боргу" для кожного члена відображаються кольорові чіпи: червоні — для боргу за кожним тарифом, зелені — для переплати. При наведенні курсора на чіп відображається підказка (tooltip) з деталями розрахунку.

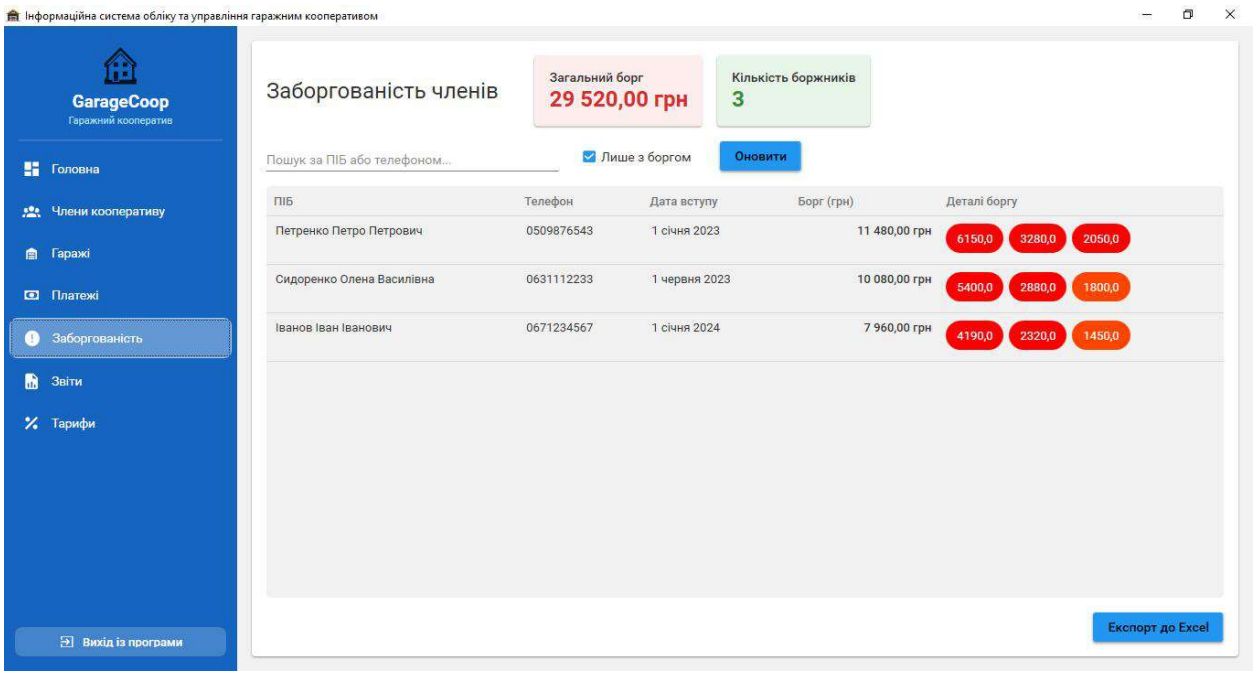

Рисунок 3.4 — Екран заборгованості з кольоровими чіпами боргу та переплати

Екран звітності (рис. 3.5) надає можливість вибрати один з чотирьох типів звітів та встановити фільтр дат для фінансових звітів. Тип "Фінансовий звіт за період" відображає зведену таблицю по членах, тип "Платежі за період" — детальний список транзакцій.

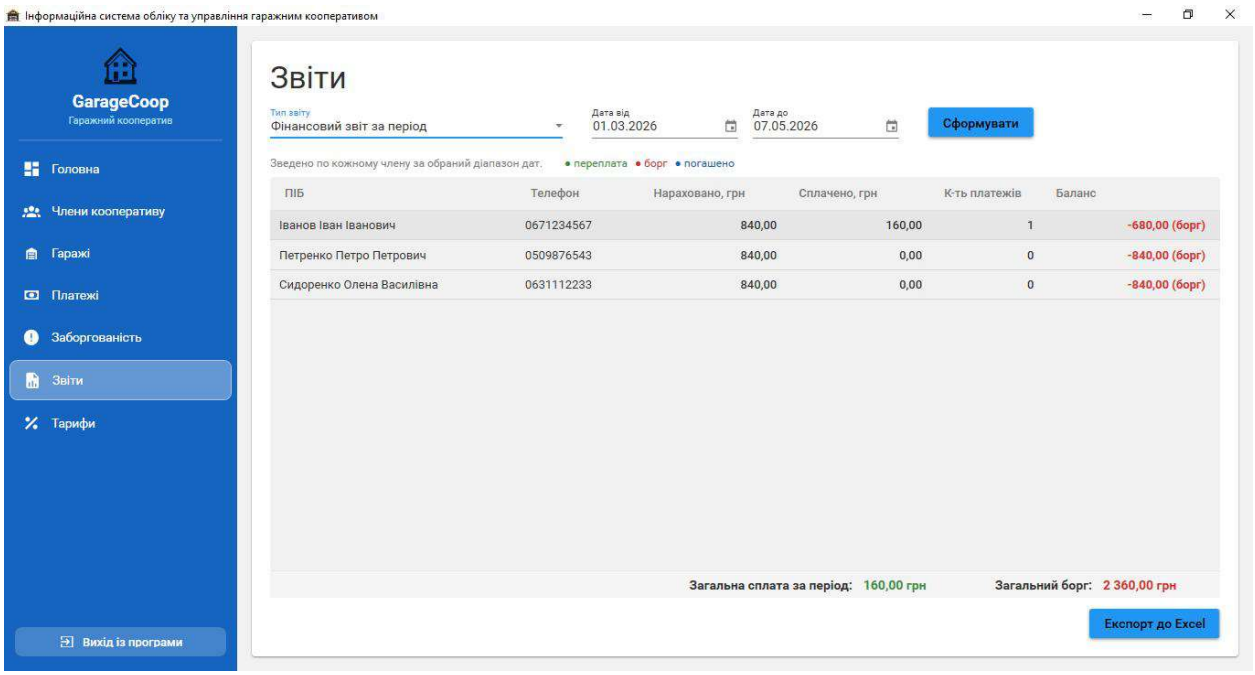

Рисунок 3.5 — Екран формування звітів з фільтром дат та табличним виведенням

3.5 Генерація звітів та експорт даних до Excel

Система підтримує чотири типи звітів, кожен з яких має власну логіку формування та може бути експортований у формат Microsoft Excel (.xlsx):

- "Список членів" — повний реєстр членів кооперативу із зазначенням контактних даних, дати вступу та статусу;
- "Список боржників" — відфільтрований список членів з ненульовим боргом, впорядкований за спаданням суми заборгованості;
- "Фінансовий звіт за період" — зведена таблиця по кожному активному члену: нараховано, сплачено, баланс та кількість платежів за обраний діапазон дат;
- "Платежі за період" — детальний хронологічний список платежів за діапазон дат із зазначенням члена, тарифу, суми, способу оплати та примітки.

Відображення відповідних секцій у View реалізовано через чотири незалежних DataGrid, видимість яких керується окремими булевими властивостями ShowMembersReport, ShowDebtorsReport, ShowFinanceReport, ShowPaymentsReport через BooleanToVisibilityConverter. Такий підхід простіший за DataTemplateSelector і уникає помилок із вкладеними DataTemplate.

Експорт даних до Excel реалізовано через бібліотеку ClosedXML, яка дозволяє створювати файли .xlsx без встановленого Microsoft Office. Сервіс ExcelExportService є узагальненим — він приймає будь-яку колекцію об'єктів і формує таблицю на основі публічних властивостей через рефлексію. Для кожної властивості застосовується відповідний числовий формат Excel:

```
var ws = wb.Worksheets.Add(sheetName);
// Заголовки
var props = typeof(T).GetProperties(BindingFlags.Public|BindingFlags.Instance);
for (int i = 0; i < props.Length; i++)
    ws.Cell(1, i+1).Value = props[i].Name;
// Дані
int row = 2;
foreach (var item in data)
{
    for (int i = 0; i < props.Length; i++)
        ws.Cell(row, i+1).Value = props[i].GetValue(item)?.ToString() ?? "";
    row++;
}
wb.SaveAs(filePath);
```

Файл Excel зберігається за шляхом, обраним користувачем у стандартному діалозі SaveFileDialog. Роздільник у числових полях — кома (відповідно до українського стандарту), що забезпечує сумісність із регіональними налаштуваннями Microsoft Excel.

3.6 Тестування системи

3.6.1 Функціональне тестування

Функціональне тестування виконувалося методом "чорного ящика" (black-box testing) — перевірялася відповідність поведінки системи специфікованим вимогам без аналізу внутрішньої реалізації. Тестування охопило всі модулі системи та граничні умови введення даних.

Тестові дані включали 20 членів кооперативу, 20 гаражів, 5 тарифів і 76 платежів за період з січня по червень 2026 року. База тестових даних формувалася SQL-скриптом із спеціально підібраними сценаріями: регулярні платежі, авансові платежі, часткові платежі, пропущені місяці та переплата.

Зведені результати функціонального тестування наведено у таблиці 3.2.

Таблиця 3.2 — Результати функціонального тестування

№	Модуль	Дія / вхідні дані	Очікуваний результат	Фактичний результат
ТС-01	Члени	Додати члена з усіма обов'язковими полями	Запис збережено; новий рядок у таблиці	Запис збережено; відображено в списку
ТС-02	Члени	Додати члена з порожнім ПІБ	Помилка валідації; кнопка "Зберегти" неактивна	Поле підсвічено червоним; зберігання заблоковано

ТС-03	Члени	Редагувати члена: змінити телефон	Змінений номер відображено в таблиці	Оновлення збережено коректно
ТС-04	Члени	Видалити активного члена з платежами	Запит підтвердження; видалення виконано	Діалог підтвердження відображено; запис видалено
ТС-05	Гаражі	Додати гараж без власника (OwnerId=NULL)	Гараж зі статусом "Вільний" збережено	Гараж відображається як вільний
ТС-06	Гаражі	Призначити власника гаражу	Статус гаражу змінюється на "Зайнятий"	Статус оновлено; відображено у рядку
ТС-07	Тарифи	Додати тариф з від'ємною сумою	Помилка валідації; зберігання заблоковано	Поле суми підсвічено; зберігання неможливе
ТС-08	Платежі	Ввести суму "5000,99" (кома)	Платіж збережено з сумою 5000.99	Платіж збережено коректно
ТС-09	Платежі	Ввести суму "5000.50" (крапка)	Платіж збережено з сумою 5000.50	Платіж збережено коректно
ТС-10	Платежі	Ввести суму "0"	Помилка: сума має бути > 0	Поле підсвічено; зберігання заблоковано
ТС-11	Платежі	Ввести порожній рядок у поле суми	Помилка: введіть суму	Відображено повідомлення про помилку
ТС-12	Борг	Сплатити рівно нараховану суму	Баланс = 0; чіп боргу відсутній	Рядок відсутній у таблиці боржників
ТС-13	Борг	Сплатити більше нарахованої суми (аванс)	Зелений чіп переплати	Зелений чіп з сумою переплати відображено
ТС-14	Борг	Не сплачувати 3 місяці (тариф 200 грн)	Борг = 600 грн; червоний чіп	Червоний чіп 600 грн відображено
ТС-15	Борг	Сплатити за 6 місяців наперед одним платежем	Переплата зарахована; борг = 0 на 6 міс.	Зелений чіп переплати; борг не нараховується

ТС-16	Звіти	Сформувати "Платежі за період" (01.01–30.06.2026)	Список усіх платежів за діапазон	Відфільтровано 76 записів; підсумок коректний
ТС-17	Звіти	Сформувати "Фінансовий звіт за період"	Зведена таблиця: нараховано/сплачено/баланс	Таблиця з 19 рядками; підсумки збіглися
ТС-18	Звіти	Натиснути "Експорт до Excel"	Файл .xlsx збережено на диск	Файл відкривається в Excel без помилок
ТС-19	Пошук	Ввести частину ПІБ "Ткач"	Відображено лише відповідні записи	Відфільтровано 1 запис (Ткаченко В.І.)
ТС-20	БД	Закрити і повторно відкрити застосунок	Усі дані збережено в SQLite	Дані відновлено у повному обсязі

За результатами тестування всі 20 тест-кейсів пройдено успішно. Зафіксовано 100% відповідність фактичних результатів очікуваним.

3.6.2 Тестування коректності розрахунку заборгованості

Коректність алгоритму розрахунку заборгованості перевірялась окремо для восьми ключових фінансових сценаріїв. Для кожного сценарію вручну обраховувались очікувані значення, після чого порівнювались з результатами системи. Результати тестування наведено у таблиці 3.3.

Таблиця 3.3 — Тестування алгоритму розрахунку заборгованості

Сценарій	Нараховано, грн	Сплачено, грн	Очікуваний баланс	Фактичний баланс
Немає жодного платежу (3 міс. × 200 грн)	600,00	0,00	–600,00 (борг)	–600,00 ✓
Сплачено рівно нараховану суму	600,00	600,00	0,00 (погашено)	0,00 ✓
Переплата: сплачено більше	600,00	1200,00	+600,00 (переплата)	+600,00 ✓
Аванс за 6 міс. одним платежем	1200,00	1200,00	0,00 (погашено)	0,00 ✓
Частковий платіж (половина суми)	600,00	300,00	–300,00 (борг)	–300,00 ✓

Кілька тарифів: членський + електрика	600+360=960	960,00	0,00 (погашено)	0,00 ✓
Новий член: 1 місяць, 200 грн, не сплатив	200,00	0,00	–200,00 (борг)	–200,00 ✓
Неактивний член (виключений)	–	–	Не розраховується	Не включається ✓

Усі 8 сценаріїв розрахунку підтверджено коректними. Знак "✓" у стовпці "Фактичний баланс" означає збіг з очікуваним результатом. Особливу увагу приділено сценарію авансової оплати (ТС-15): оплата за 6 місяців наперед одним платежем коректно зараховується системою і не призводить до повторних нарахувань боргу протягом наступних місяців.

3.7 Виявлені недоліки та шляхи їх усунення

У процесі розробки та тестування було виявлено та усунено ряд помилок, частина з яких має системний характер і є типовою для WPF-застосунків з використанням EF Core та культури uk-UA.

Проблема 1. Конфлікт відстеження сутностей EF Core (UNIQUE constraint failed).

При збереженні нового платежу EF Core намагався вставити вже існуючі записи Member та Tariff як нові рядки, оскільки вони були у стані Detached. Вирішено обнуленням навігаційних властивостей перед збереженням і методом SetValues у BaseRepository.UpdateAsync.

Проблема 2. Некоректне відображення вибраного елементу ComboBox при редагуванні.

WPF порівнює вибраний елемент ComboBox через Object.ReferenceEquals. Оскільки ViewModel зберігає посилання на об'єкт з першого запиту, а список ComboBox заповнюється новими екземплярами, порівняння завжди повертало false. Вирішено пошуком за Id: SelectedMember = members.FirstOrDefault(m => m.Id == payment.MemberId).

Проблема 3. FormatException при введенні десяткових чисел у полі суми.

Культура uk-UA використовує кому як десятковий розділювач, але WPF передає конвертеру поточну культуру та додатково викликає System.Convert.ChangeType, що не розуміє крапки. Вирішено заміною decimal-прив'язки на рядкову з ручним парсингом через CultureInfo.InvariantCulture після Replace(',', '.').

Проблема 4. Борг не перераховувався після авансових платежів.

Початкова реалізація фільтрувала платежі за умовою Period <= currentPeriod, що виключало платежі, внесені наперед з майбутнім розрахунковим місяцем. Вирішено видаленням фільтра: тепер враховуються всі платежі незалежно від вказаного періоду.

Перелічені проблеми виявлено на етапі функціонального тестування та усунено ще до завершення розробки. Наявність таких проблем є типовою для WPF-застосунків з EF Core і потребує уважного підходу до налаштування відстеження сутностей та локалізації.

3.8 Рекомендації щодо подальшого розвитку системи

Розроблена система повністю відповідає сформульованим функціональним вимогам і є готовою до впровадження у реальному гаражному кооперативі. Водночас аналіз виявив ряд напрямків, за якими систему можна розширити у майбутньому.

1) Веб-інтерфейс або мобільний застосунок. Перенесення бізнес-логіки до окремого ASP.NET Core API дозволить розробити веб-версію або мобільний клієнт, що забезпечить доступ до даних з будь-якого пристрою.
2) Хмарна синхронізація даних. Можливість резервного копіювання бази даних до хмарного сховища (OneDrive, Google Drive) або переведення на хмарну СУБД (PostgreSQL, Azure SQL) для захисту від втрати даних.
3) Розширена система ролей. Додавання ролей "Ревізор" (перегляд звітів без редагування) та "Член" (особистий кабінет для перегляду власних платежів і боргу).

4) Автоматичні сповіщення. Інтеграція з поштовим сервером або Telegram Bot API для автоматичної розсилки нагадувань боржникам на початку кожного місяця.
5) Журнал аудиту. Ведення лог-файлу всіх змін у базі даних (хто, що і коли змінив) для підвищення прозорості та контролю адміністрації кооперативу.
6) Управління комунальними лічильниками. Можливість введення показань лічильників електроенергії для кожного боксу та автоматичного розрахунку індивідуального платежу.

Висновки до розділу 3

У третьому розділі описано практичну реалізацію та тестування інформаційної системи обліку та управління гаражним кооперативом. Основні результати:

1) Описано структуру проєкту з 6 функціональними папками (Models, Data, ViewModels, Views, Services, Helpers), що реалізують архітектуру MVVM.
2) Реалізовано 4 моделі даних (Member, Garage, Tariff, Payment) та AppDbContext з налаштованими зв'язками і каскадним видаленням. Базовий репозиторій BaseRepository<T> вирішує проблему конфлікту відстеження сутностей EF Core.
3) Реалізовано ключовий алгоритм розрахунку заборгованості на основі фактичних сум платежів (а не кількості транзакцій), що коректно обробляє авансові, часткові та надлишкові платежі.
4) Вирішено проблему введення десяткових чисел у культурі uk-UA через рядкове поле AmountText з ручним парсингом та InvariantCulture.
5) Реалізовано чотири типи звітів та узагальнений сервіс ExcelExportService на основі бібліотеки ClosedXML.
6) Проведено функціональне тестування (20 тест-кейсів) та тестування алгоритму розрахунку (8 сценаріїв); всі тести пройдено успішно.

7) Виявлено та усунено 4 системні помилки, типових для WPF-застосунків з EF Core та культурою uk-UA.
8) Визначено 6 напрямків подальшого розвитку системи.

Результати, отримані у трьох розділах кваліфікаційної роботи, підтверджують досягнення поставленої мети: розроблена інформаційна система повністю автоматизує облік членів, гаражів, тарифів і платежів гаражного кооперативу та готова до практичного впровадження.

ВИСНОВКИ

У кваліфікаційній роботі виконано проектування та розробку інформаційної системи обліку та управління гаражним кооперативом. Система реалізована у вигляді десктопного застосунку на платформі .NET 9 із використанням технологій WPF, MVVM, Entity Framework Core 9 та SQLite. За результатами виконаної роботи зроблено такі висновки.

1) Проведено аналіз предметної області та встановлено, що більшість гаражних кооперативів ведуть облік у паперових журналах або засобами загального призначення (Excel), що призводить до помилок, втрати даних і значних витрат часу адміністрації. Жоден з існуючих програмних аналогів (TourManager, 1С:Кооператив, Excel) не задовольняє одночасно всім критеріям: безкоштовність, автономна робота без сервера, коректний облік авансових платежів і переплати, україномовний інтерфейс.
2) Сформульовано 12 функціональних і нефункціональних вимог до системи та розроблено діаграму варіантів використання (Use Case) з двома ролями користувачів — Адміністратор і Касир. Обґрунтовано вибір технологічного стеку: C# .NET 9, WPF/XAML, MVVM через CommunityToolkit.Mvvm, MaterialDesignInXamlToolkit 5.x, SQLite та Entity Framework Core 9.
3) Спроектовано архітектуру системи на основі шаблону MVVM, що забезпечує чіткий поділ відповідальності між шарами: Model (дані та EF Core), View (XAML), ViewModel (логіка подання та команди), Services (сервіси для діалогів і експорту). Спроектовано реляційну базу даних у третій нормальній формі з чотирма таблицями (Members, Garages, Tariffs, Payments) і відповідними зовнішніми ключами.
4) Розроблено UML-моделі системи: діаграму класів, дві діаграми послідовності (реєстрація платежу та розрахунок заборгованості), три діаграми станів (член, гараж, платіж), а також проектні макети семи

екранів інтерфейсу. Формалізовано алгоритм розрахунку заборгованості у вигляді математичного виразу.

5) Реалізовано всі заплановані модулі системи: облік членів кооперативу та гаражних боксів, управління тарифами, реєстрація платежів, розрахунок заборгованості, формування чотирьох типів звітів та їх експорт до Excel. Особливу увагу приділено коректній обробці авансових і часткових платежів — переплата відображається зеленим чіпом і зараховується у рахунок майбутніх нарахувань.
6) Вирішено ряд нетривіальних технічних проблем, характерних для WPF-застосунків з EF Core: конфлікт відстеження сутностей при збереженні (розв'язано через SetValues на вже відстеженому екземплярі та обнулення навігаційних властивостей), некоректне порівняння об'єктів у ComboBox (розв'язано пошуком за Id), помилка форматування десяткових чисел у культурі uk-UA (розв'язано рядковим полем з ручним парсингом через InvariantCulture).
7) Проведено функціональне тестування системи: виконано 20 тест-кейсів і 8 сценаріїв перевірки алгоритму розрахунку заборгованості. Усі тести пройдено успішно з 100-відсотковою відповідністю фактичних результатів очікуваним.
8) Розроблено базу тестових даних (SQL-скрипт): 20 членів кооперативу, 20 гаражних боксів, 5 тарифів, 76 платежів за період з січня по червень 2026 року — зі спеціально підібраними сценаріями авансових, часткових і нерегулярних оплат.

Розроблена система є повністю функціональним програмним продуктом, готовим до практичного впровадження у будь-якому гаражному кооперативі незалежно від його масштабу. Система може слугувати основою для подальшого розширення — додавання веб-інтерфейсу, мобільного клієнта, хмарної синхронізації або інтеграції з месенджерами для автоматичних сповіщень боржників.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Закон України "Про кооперацію" від 10 липня 2003 р. № 1087-IV. URL: https://zakon.rada.gov.ua/laws/show/1087-15 (дата звернення: 20.04.2026).
2. Закон України "Про захист персональних даних" від 1 червня 2010 р. № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 20.04.2026).
3. ДСТУ ISO/IEC 25010:2013. Системна та програмна інженерія. Вимоги та оцінювання якості систем і програмних засобів (SQuaRE). Моделі якості систем і програмних продуктів. Київ: ДП "УкрНДНЦ", 2015. 36 с.
4. Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 11th ed. New York: Apress, 2022. 1306 p.
5. Price M. C# 12 and .NET 8 — Modern Cross-Platform Development. 8th ed. Birmingham: Packt Publishing, 2023. 826 p.
6. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1994. 395 p.
7. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p.
8. Lerman J., Miller S. Programming Entity Framework: DbContext. Sebastopol: O'Reilly Media, 2012. 362 p.
9. Dykstra T. Get started with ASP.NET Core MVC and Entity Framework Core. Microsoft Docs, 2023. URL: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/ (дата звернення: 15.03.2026).
10. Stellman A., Greene J. Head First C#. 4th ed. Sebastopol: O'Reilly Media, 2021. 1027 p.
11. Bugnion P. WPF 4.5 Unleashed. Indianapolis: Sams Publishing, 2013. 900 p.
12. Microsoft. .NET 9 documentation. 2024. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 10.04.2026).
13. Microsoft. Entity Framework Core documentation. 2024. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 10.04.2026).

14. Microsoft. Windows Presentation Foundation (WPF) documentation. 2024. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/ (дата звернення: 05.04.2026).
15. Microsoft. CommunityToolkit.Mvvm documentation. 2024. URL: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/ (дата звернення: 12.03.2026).
16. SQLite Consortium. SQLite Documentation. 2024. URL: https://www.sqlite.org/docs.html (дата звернення: 08.04.2026).
17. Material Design in XAML Toolkit. GitHub repository. 2024. URL: https://github.com/MaterialDesignInXAML/MaterialDesignInXamlToolkit (дата звернення: 15.02.2026).
18. ClosedXML. ClosedXML Documentation. 2024. URL: https://docs.closedxml.io/ (дата звернення: 20.03.2026).
19. BCrypt.Net-Next. NuGet Package. 2023. URL: https://www.nuget.org/packages/BCrypt.Net-Next (дата звернення: 01.03.2026).
20. Mazur O., Kovalenko I. Approaches to Automation of Accounting Processes in Housing and Service Cooperatives. Scientific Notes of the Ukrainian Research Institute of Communication. 2022. Vol. 3. P. 45–52.
21. Petrenko V., Sydorenko A. Application of MVVM Pattern in Desktop Application Development Using WPF. Ukrainian Journal of Information Technology. 2021. Vol. 3, No. 1. P. 12–18.
22. Smith J. Patterns and Practices for WPF Applications. Microsoft Developer Network. 2019. URL: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf (дата звернення: 10.02.2026).
23. Bochkaryov D. Using Entity Framework Core with SQLite in .NET Desktop Apps. DotNet Blog. 2023. URL: https://devblogs.microsoft.com/dotnet/ef-core-sqlite/ (дата звернення: 14.03.2026).