Полтавський університет економіки і торгівлі

Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___»_____202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБЛІКУ ФІНАНСОВИХ ВИТРАТ ТА ДОХОДІВ»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Шабельник Антон Юрійович
_____«____»_____ 202_ р.
(підпис)

Науковий керівник к. ф.-м. н., доцент, Ольховська Олена Володимирівна
_____«____»_____ 202_ р.
(підпис)

Рецензент _____

ПОЛТАВА 2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
Проектування програмного забезпечення	Це етап розробки, на якому визначається архітектура системи, логіка взаємодії між модулями, структура бази даних і користувацький інтерфейс. Метою проектування є створення надійної, масштабованої та зручної у використанні системи для обліку фінансових операцій.
Фінансова операція	Це окремий запис у системі, що відображає факт отримання доходу або здійснення витрати. Кожна операція містить суму, дату, тип дохід або витрата, категорію та додатковий опис.
Доходи	Це грошові кошти, які надходять користувачеві в результаті його діяльності або інших джерел, заробітна плата, стипендія, подарунки тощо.
Витрати	Це грошові кошти, які користувач витрачає на різні потреби, такі як харчування, транспорт, комунальні послуги тощо.

ВСТУП

У сучасних умовах розвитку цифрових технологій інформаційні системи відіграють важливу роль у повсякденному житті людини. Особливої актуальності набувають програмні засоби, призначені для автоматизації фінансових процесів, оскільки ефективне управління особистими фінансами є одним із ключових чинників фінансової стабільності та раціонального використання грошових ресурсів. Зростання кількості фінансових операцій, різноманітність джерел доходів і напрямів витрат зумовлюють необхідність використання спеціалізованого програмного забезпечення для їх обліку та аналізу.

Традиційні методи ведення фінансового обліку, такі як паперові записи або використання загальних електронних таблиць, мають низку недоліків: високу ймовірність помилок, відсутність автоматизованого аналізу, складність структурування даних та обмежені можливості візуалізації результатів. У зв’язку з цим виникає потреба в розробці програмного забезпечення, яке забезпечує зручний, надійний та ефективний облік фінансових витрат і доходів користувача.

Проектування програмного забезпечення для обліку фінансових витрат та доходів передбачає комплексний підхід, що охоплює аналіз вимог користувача, розробку структури даних, побудову логіки обробки інформації та створення інтуїтивно зрозумілого користувацького інтерфейсу. Важливим аспектом є забезпечення точності фінансових розрахунків, цілісності даних та можливості подальшого розширення функціональності системи. [1]

Мета кваліфікаційної роботи – розробка програмного забезпечення для обліку фінансових витрат та доходів, яке забезпечує зручне введення, збереження, обробку й аналіз фінансових даних користувача.

Об’єкт розробки – програмне забезпечення для обліку фінансових витрат та доходів користувача, призначене для автоматизації процесів введення, збереження, обробки та аналізу фінансових даних.

Предмет розробки – методи та алгоритми розробки програмного забезпечення для обліку фінансових витрат і доходів, включно з логікою обробки даних, структурою бази даних та організацією користувацького інтерфейсу.

Методи дослідження – застосовувався аналіз предметної області для вивчення процесу обліку фінансових витрат і доходів та визначення вимог до програмного забезпечення. Метод системного аналізу дозволив розглянути програмний продукт як цілісну систему з взаємопов’язаних компонентів. Моделювання використовувалося для побудови логічної структури системи та моделей даних. Проектування програмного забезпечення забезпечило визначення архітектури, основних модулів і їх взаємодії. Під час реалізації застосовувався метод об’єктно-орієнтованого програмування мовою C#, що сприяло структурованості та зручності підтримки коду. Аналіз і синтез використовувалися для поєднання окремих компонентів у єдину систему. Коректність роботи програмного забезпечення перевірялася за допомогою тестування та експериментального використання з тестовими даними, що дозволило оцінити його функціональні можливості. [2]

Актуальність розробки – у сучасних умовах динамічного розвитку бізнесу, високої конкуренції та постійних змін на ринках, ефективне управління фінансовими потоками є важливою складовою успішної діяльності як підприємств, так і приватних осіб. Щоденний облік доходів та витрат дозволяє своєчасно відстежувати фінансовий стан, планувати бюджет, виявляти неефективні витрати та приймати обґрунтовані економічні рішення. Розробка спеціального програмного забезпечення для обліку фінансових витрат і доходів дає змогу автоматизувати ці процеси,

підвищити якість обліку, спростити ведення фінансової документації, а також забезпечити доступ до аналітичних даних у реальному часі.

Кваліфікаційна робота складається з чотирьох розділів. У першому розділі сформульовано постановку задачі, що полягає у створенні застосунку для управління особистими фінансами. У другому розділі здійснено огляд застосунку Money Pro для обліку доходів і витрат, а також проаналізовано аналогічний застосунок Spending Tracker. У третьому розділі детально описано покрокову логіку роботи коду та алгоритмів фінансового обліку, наведено UML-діаграми функціонування застосунку і його програмної частини. У четвертому розділі висвітлено процес розробки застосунку для управління фінансами та подано опис його роботи. Обсяг пояснювальної записки: 47 стор., в т.ч. основна частина – 45 стор., джерела – 20 назв.

1. ПОСТАНОВКА ЗАДАЧ

Метою даної роботи є розробка програмного забезпечення для обліку фінансових витрат та доходів. Для досягнення цієї мети необхідно вирішити такі завдання:

1. Провести аналіз предметної області та визначити основні вимоги до системи.
2. Розробити структуру бази даних для збереження фінансових операцій, категорій витрат та доходів, а також даних користувачів.
3. Спроектувати архітектуру програмного забезпечення, що забезпечує ефективну взаємодію між модулями, логікою обробки даних та користувацьким інтерфейсом.
4. Реалізувати програму мовою C# з використанням об’єктно-орієнтованого підходу та платформи .NET.
5. Забезпечити можливість ведення обліку доходів і витрат, створення записів, їх редагування та видалення.
6. Гарантування коректності й достовірності обробки інформації, запобігання повторному внесенню даних та мінімізація помилок під час введення.
7. Провести тестування програмного забезпечення для перевірки коректності роботи, надійності та відповідності поставленим вимогам.
8. Розробити зручний та інтуїтивно зрозумілий користувацький інтерфейс для ефективної роботи користувача з системою.

Виконання зазначених завдань дозволить створити надійну та функціональну систему для обліку фінансів, що сприятиме підвищенню фінансової грамотності та ефективного управління особистими коштами.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд застосунку Money Pro для обліку фінансових витрат та доходів

Розглянемо програмне забезпечення для керування фінансами “Money Pro”. [3]

Насамперед слід інсталювати застосунок. Для цього потрібно знайти програму під назвою Money Pro або перейти за відповідним посиланням у браузері, після чого натиснути кнопку «Отримати» чи «Встановити» і дочекатися завершення процесу встановлення. Після цього застосунок з’явиться в меню «Пуск» і буде готовий до використання.

Крім того, застосунок можна завантажити безпосередньо з Microsoft Store. (рис. 2.1).

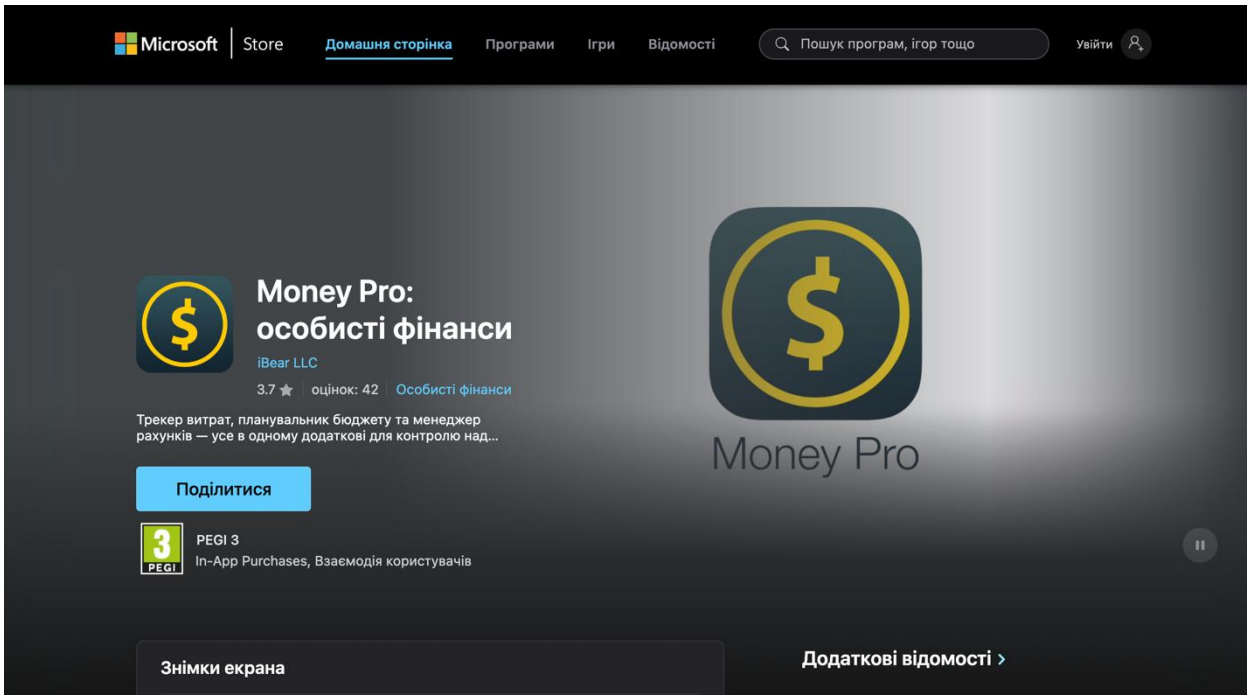

Рисунок 2.1 – застосунок Money Pro

Функція скорочення витрат реалізується комплексно та спрямована на формування усвідомленого фінансового планування. Користувач має можливість встановлювати бюджети для окремих категорій витрат і визначати граничні суми на певний період. (рис. 2.2). У застосунку є можливість контролювати кредитні картки користувача та банківські рахунки. (рис. 2.3).

Рисунок 2.2 – оптимізація витрат

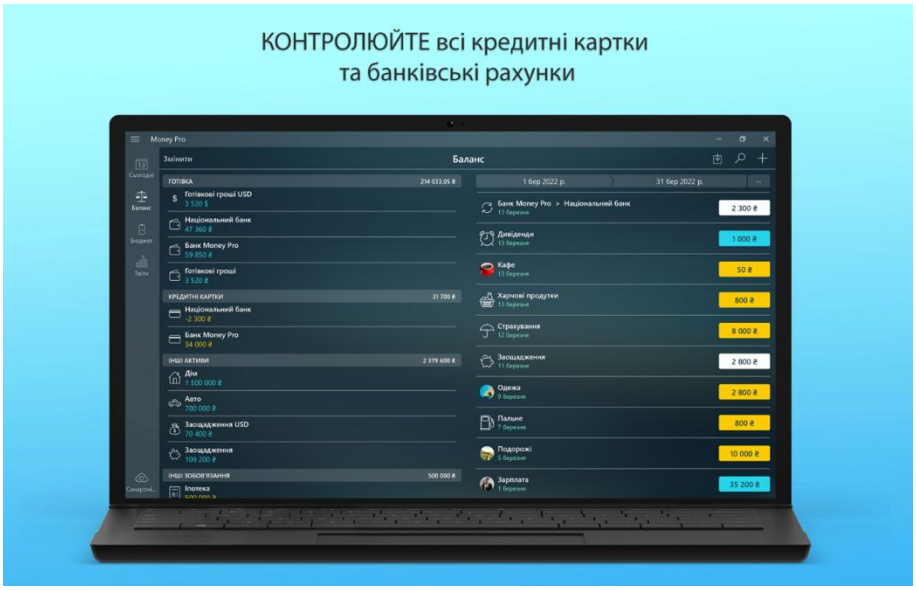

Рисунок 2.3 – контроль банківських рахунків

У застосунку Money Pro передбачено зручні інструменти для забезпечення вчасної сплати рахунків, що допомагає уникати прострочень і додаткових штрафів. Користувач може додавати регулярні платежі, зокрема комунальні послуги, оренду, підписки чи кредити, із зазначенням суми та дати оплати. (рис. 2.4).

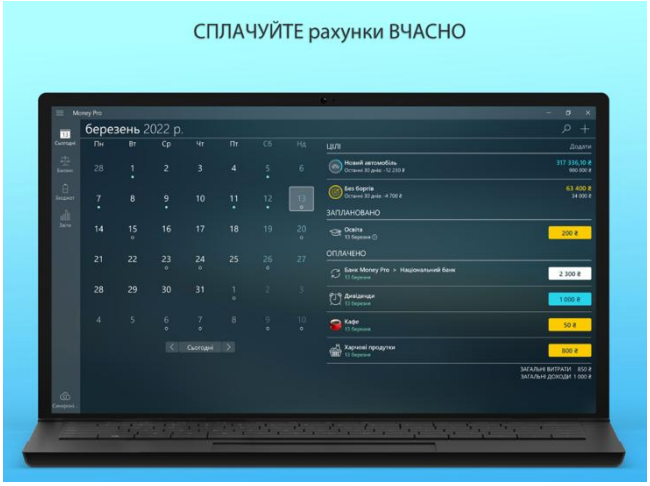

Рисунок 2.4 – можливість планування сплати рахунків

У застосунку Money Pro візуальний аналіз витрат та доходів відіграє важливу роль у розумінні фінансового стану користувача. (рис. 2.5).

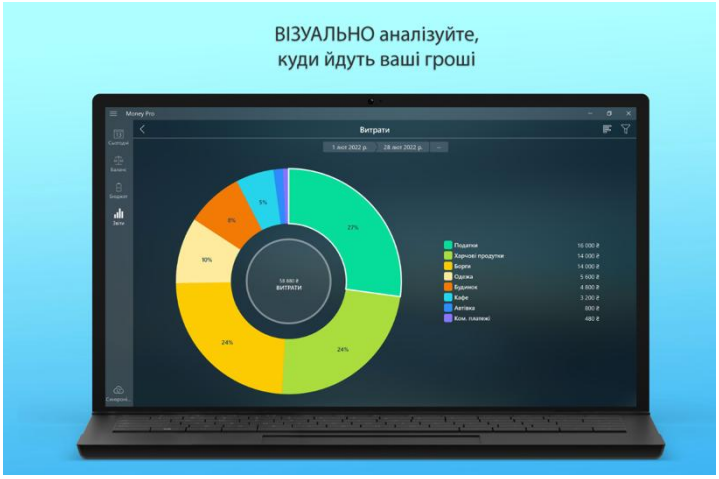

Рисунок 2.5 – візуальний аналіз витрат та доходів

У застосунку передбачено різнокольорові теми оформлення, які дозволяють користувачеві персоналізувати інтерфейс відповідно до власних уподобань. (рис. 2.6).

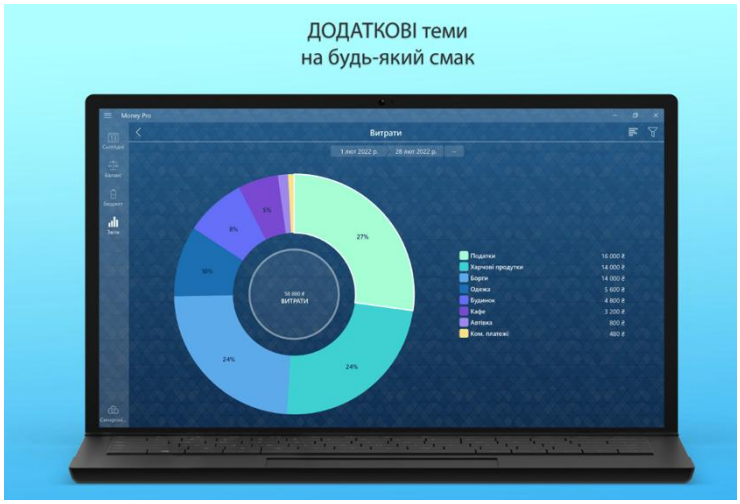

Рисунок 2.6 – додаткові різнокольорові теми

У застосунку Money Pro функція підключення банків призначена для спрощення обліку фінансів і автоматизації внесення даних. (рис. 2.7).

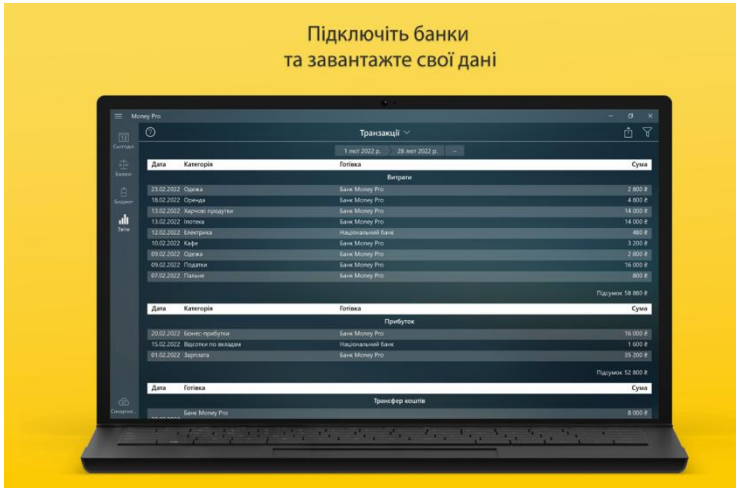

Рисунок 2.7 – можливість завантажити дані з банку

Переваги:

1. Дозволяє вести детальний облік доходів і витрат, планувати бюджет і рахунки, що робить її зручною як для особистого, так і для сімейного використання.
2. Можна працювати з необмеженою кількістю рахунків, фіксувати регулярні платежі та розподіляти одну транзакцію на кілька категорій.
3. Є звіти, аналітика, календар, пошук й інші інструменти для контролю фінансів.
4. Підтримка декількох валют, захист паролем, резервне копіювання даних і кілька профілів.

Недоліки:

1. Базові функції доступні безкоштовно, але багато корисного: синхронізація, банківські підключення, розширені звіти вимагають підписки PLUS або GOLD.
2. Для повноцінного автоматичного імпорту транзакцій з банків потрібна саме підписка GOLD.
3. Користувачі критикують часті підвищення вартості підписок та перехід на повністю підписну модель, через що колишні безкоштовні можливості можуть стати платними.
4. За відгуками, інколи трапляються повільне завантаження, збої або проблеми із синхронізацією, особливо при підключенні банківських рахунків.
5. Деяким користувачам інтерфейс здається не надто інтуїтивним на початку, і налаштування може зайняти час.

2.2. Огляд програми Spending Tracker для обліку фінансових витрат та доходів

Moneygraph - це програмний застосунок для персонального фінансового обліку, доступний у Microsoft Store для Windows. Програма призначена

для контролю особистих фінансів: обліку витрат, доходів та аналізу використання коштів. Вона допомагає користувачам відстежувати свої фінансові операції, планувати бюджет і краще розуміти власні витрати. (рис. 2.8).

Рисунок 2.8 – Головний екран програми

Програма має зручний інтерфейс, який дозволяє швидко додавати фінансові операції без складних налаштувань. (рис. 2.9). Завдяки регулярному обліку витрат користувач може зручно переглядати контролювати бюджет і знаходити можливості для економії коштів. (рис. 2.10).

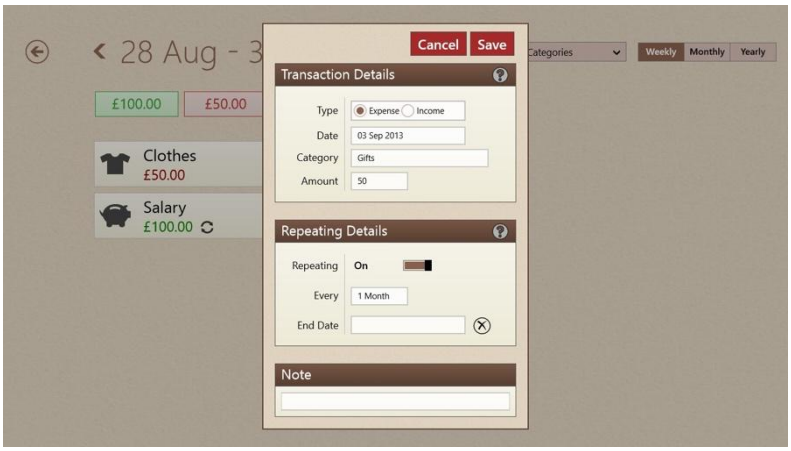

Рисунок 2.9 – Процес створення нової транзакції

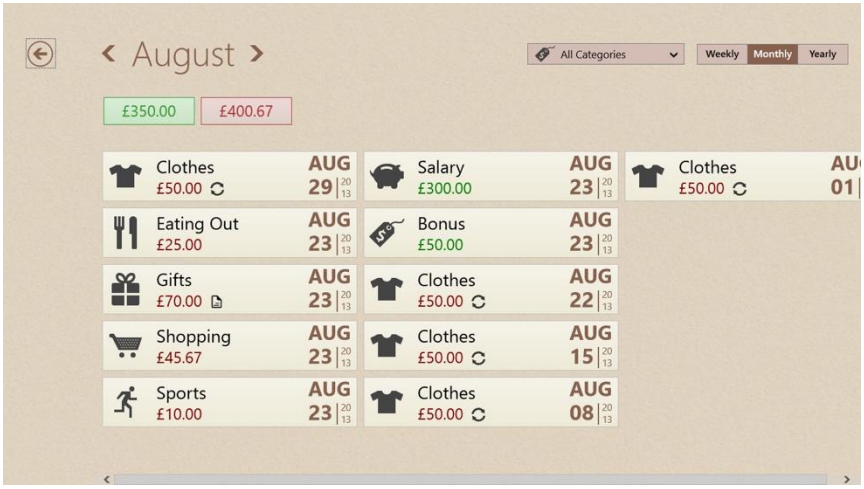

Рисунок 2.10 – Режим перегляду транзакцій

У програмі Spending Tracker візуалізація витрат та доходів реалізована у вигляді графіків і статистичних звітів, які дозволяють швидко зрозуміти фінансовий стан користувача та структуру витрат. (рис. 2.11).

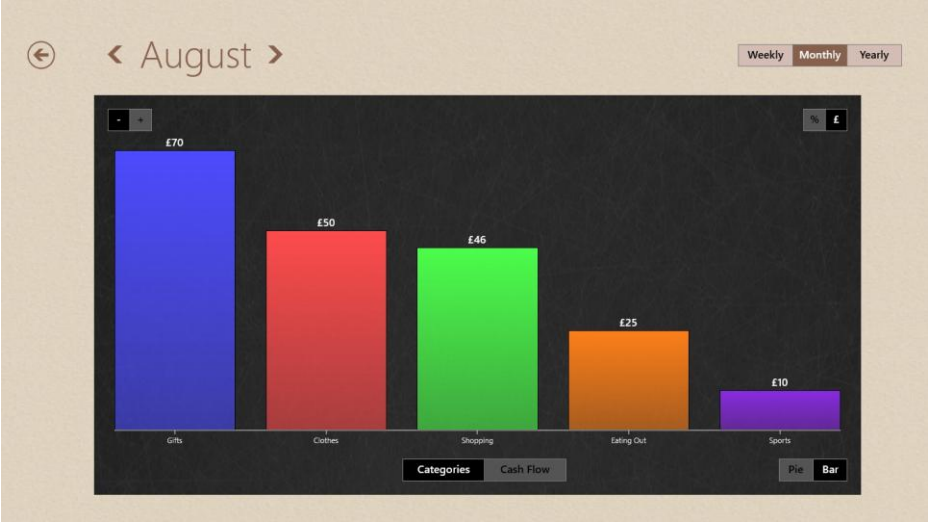

Рисунок 2.11 – Візуалізація витрат та доходів

Функція Summary can be viewed in Snapped mode у програмі Spending Tracker означає, що користувач може переглядати короткий підсумок своїх фінансових даних у режимі прикріпленого вікна. (рис. 2.12).

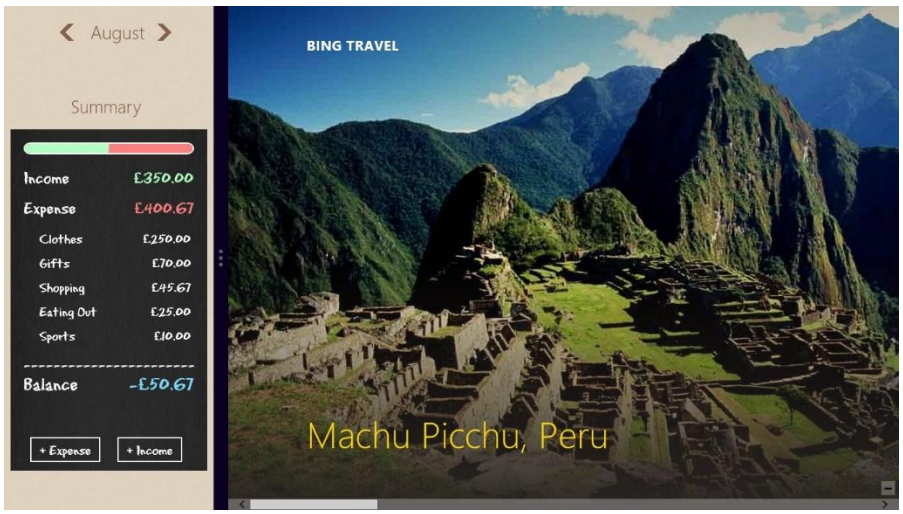

Рисунок 2.12 – Функція Summary can be viewed in Snapped mode

Переваги програми

1. Простий та зрозумілий інтерфейс. Програма має інтуїтивно зрозумілий дизайн, що дозволяє користувачам швидко освоїти її функції та без труднощів додавати записи про доходи та витрати.
2. Швидке введення фінансових операцій. Користувач може оперативно додавати інформацію про витрати та доходи, що робить програму зручною для щоденного використання.
3. Категоризація витрат. Програма дозволяє розподіляти витрати за категоріями, харчування, транспорт, житло, розваги, що допомагає краще аналізувати структуру бюджету.
4. Візуалізація фінансових даних. Дані відображаються у вигляді графіків і діаграм, що дозволяє швидко оцінити співвідношення доходів і витрат та визначити найбільш витратні категорії.
5. Підсумкова статистика. Застосунок формує короткі фінансові підсумки за певний період і дозволяє переглядати їх навіть у компактному режимі Snapped mode, що зручно під час роботи з іншими програмами.
6. Планування бюджету. Користувач може встановлювати фінансові обмеження та контролювати дотримання бюджету.

Недоліки програми

1. Обмежений функціонал у порівнянні з професійними фінансовими системами. Програма орієнтована переважно на особистий бюджет, тому може не мати розширених функцій фінансового аналізу або бухгалтерського обліку.
2. Можлива відсутність інтеграції з банківськими сервісам. Дані зазвичай потрібно вводити вручну, що може займати додатковий час.
3. Обмежена аналітика. Хоча застосунок має графіки та звіти, їх можливості можуть бути простішими порівняно з професійними системами фінансового менеджменту.
4. Залежність від ручного введення даних. Точність фінансового аналізу безпосередньо залежить від того, наскільки регулярно користувач вносить інформацію про свої доходи та витрати.

Застосунок Spending Tracker є зручним інструментом для контролю особистих фінансів. Його основними перевагами є простота використання, швидке введення операцій, наочна візуалізація витрат та можливість перегляду фінансових підсумків. Водночас програмі притаманні певні обмеження, зокрема менша функціональність порівняно з професійними фінансовими системами та необхідність ручного введення даних. Загалом застосунок добре підходить для повсякденного контролю бюджету та аналізу витрат.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. CRUD операції для фінансових операцій бази даних застосунку

Організаціям, які працюють з даними клієнтів, фінансовими операціями, медичною інформацією та іншими важливими відомостями, необхідні надійні системи для довготривалого зберігання інформації. Зазвичай такі дані впорядковуються у базах даних — структурованих електронних сховищах. Найпоширенішим типом є реляційні бази даних, у яких інформація подається у вигляді таблиць із рядками та стовпцями, що можуть бути пов’язані між собою за допомогою ключів.

CRUD — це скорочення, яке позначає чотири базові операції, потрібні для роботи з даними в системах постійного зберігання: створення Create, читання Read, оновлення Update та видалення Delete.

Модель CRUD широко застосовується в роботі з базами даних, зокрема у системах керування реляційними базами Oracle, MySQL, PostgreSQL, а також у NoSQL-рішеннях, таких як MongoDB, Apache Cassandra чи AWS DynamoDB.

Типова CRUD-архітектура складається з трьох основних компонентів:

API або серверної частини, яка обробляє запити;

бази даних, що зберігає та надає доступ до інформації;

інтерфейсу користувача UI, через який користувач взаємодіє із системою.

CRUD-операції використовуються для роботи з даними, які зберігаються постійно та не втрачаються після вимкнення системи. Це відрізняє їх від операцій з інформацією, що зберігається в оперативній пам’яті або кеші.

Користувачі можуть виконувати CRUD-операції як за допомогою програмного коду, так і через графічний інтерфейс. Подібні дії можливі і з

файлами наприклад, створення, редагування чи видалення документа, однак у контексті CRUD мова йде саме про операції з записами в базах даних, а не з окремими файлами.

Create

Операція створення дозволяє додавати нові записи до бази даних. У SQL вона відповідає команді INSERT. Запис — це окремий рядок таблиці, а стовпці є його атрибутами. Користувач може додати новий рядок, але змінювати структуру таблиці додавати атрибути зазвичай має право лише адміністратор.

Read

Операція читання дає змогу отримувати та переглядати інформацію з бази даних. Пошук може здійснюватися за ключовими словами або шляхом фільтрації за певними умовами. При цьому дані не змінюються.

Update

Операція оновлення використовується для внесення змін до вже існуючих записів. Вона може застосовуватися як до одного запису, так і до групи записів, залежно від заданих критеріїв.

Delete

Операція видалення дозволяє прибирати непотрібні записи з бази даних. Видалення може бути жорстким дані повністю стираються або м’яким запис позначається як видалений, але фізично залишається в базі.

Що таке тестування CRUD?

Тестування CRUD у програмному забезпеченні полягає в перевірці коректності обробки даних під час їх створення, читання, оновлення та видалення. Воно включає перевірку точності значень, правильності відображення інформації та відсутності помилок на серверній стороні. Наприклад, тестувальник може перевіряти, чи правильно імпортовані дані, чи коректно вони редагуються та чи відображаються зміни у звітах.

Існує два основні підходи до тестування баз даних:

самостійне написання та виконання SQL-запитів;

перевірка у співпраці з розробником, інженером з автоматизації або досвідченим тестувальником.

Приклади тестування CRUD-функцій

Create

Наприклад, додавання нового облікового запису співробітника. Тестувальник перевіряє відображення запису в інтерфейсі та його наявність у базі даних.

Додатково перевіряються:

1. шифрування чутливих даних паролів;
2. обробка помилок при дублюванні даних наприклад, email;
3. багатокористувацькі сценарії;
4. рівні доступу для різних ролей;
5. різні варіанти введення даних;
6. реакція системи на збої.

Read

Перевіряється можливість перегляду створеного запису та відповідність даних інформації в базі.

Додаткові сценарії:

1. перегляд одного або кількох записів;
2. пошук неіснуючого запису;
3. тестування функції пошуку;
4. перевірка доступу відповідно до ролей;
5. обробка помилок.

Update

Тестується зміна даних у записі та збереження результатів у базі.

Додаткові перевірки:

1. масове оновлення записів;
2. порушення унікальних обмежень;
3. різні комбінації введених даних;
4. відповідність оновлень правам доступу;

5. поведінка системи при збоях.

Delete

Перевіряється видалення запису та результат у базі даних.

Додаткові сценарії:

1. видалення кількох записів одночасно;
2. часткове видалення інформації;
3. перевірка ланцюжка Create–Update–Delete;
4. повторне створення та видалення одного запису;

коректна обробка помилок. [4]

3.2. Покроковий алгоритм роботи коду застосунку

Після запуску програмного забезпечення створюється екземпляр головної форми, у конструкторі якої ініціалізуються всі елементи інтерфейсу користувача та заповнюється випадаючий список типів фінансових операцій значеннями «Прибуток» і «Витрата». Далі формується універсальний шлях до файлу бази даних у папці «Мої документи» та створюється рядок підключення до SQL LocalDB. Після цього викликається метод перевірки наявності бази даних: якщо файл бази відсутній, автоматично створюється нова база даних і таблиця для збереження фінансових записів. Після ініціалізації бази даних здійснюється завантаження всіх наявних записів з таблиці у компонент DataGridView для їх відображення користувачу, а також обчислюється загальний фінансовий баланс шляхом підсумовування сум усіх операцій з урахуванням їх типу, де значення з категорією «Прибуток» додаються до балансу, а значення з категорією «Витрата» віднімаються.

Під час додавання нового запису користувач вводить ідентифікатор операції, обирає дату, тип операції, вводить суму та опис, після чого натискає кнопку додавання. Програма виконує перевірку коректності введених даних, зокрема правильність формату ідентифікатора, вибір типу операції та числове значення суми. У разі успішної перевірки формується SQL-запит для вставки нового запису в базу даних, який виконується через підключення до SQL LocalDB. Після додавання запису оновлюється таблиця відображення даних, повторно обчислюється загальний баланс, очищуються всі поля введення та виводиться повідомлення про успішне виконання операції.

У випадку видалення запису користувач вводить ідентифікатор операції та ініціює процес видалення відповідною кнопкою. Програма перевіряє коректність введеного ідентифікатора, після чого виконує SQL-запит на видалення запису з бази даних. Якщо запис знайдено та видалено,

відбувається оновлення таблиці з даними, перерахунок балансу та очищення полів введення, про що користувач отримує відповідне повідомлення; якщо ж запис не існує, система повідомляє про це. Протягом усього часу роботи додатка баланс відображається в реальному часі та завжди відповідає актуальному стану даних у базі, а завершення роботи програми здійснюється коректно через кнопку виходу з попереднім закриттям усіх ресурсів.

Крок 1. Запуск програми.

1. При відкритті форми:
2. ініціалізуються елементи інтерфейсу;
3. у ComboBox додаються значення «Прибуток» і «Витрата»;
4. формується шлях до файлу бази даних;
5. створюється база даних і таблиця, якщо їх ще немає;
6. завантажуються всі записи в DataGridView.

Крок 2. Створення бази даних.

1. Програма перевіряє:
2. чи існує файл БД;
3. якщо ні — створює БД;
4. перевіряє наявність таблиці Expenses;
5. якщо таблиці немає — створює її.

Крок 3. Введення даних користувачем.

1. Користувач вводить ID, суму, опис і дату;
2. Перед додаванням перевіряється коректність ID;
3. перевіряється, чи введена сума;
4. перевіряється, чи вибрана категорія.

Крок 4. Додавання запису.

1. дані вставляються в таблицю Expenses;
2. оновлюється таблиця на екрані;
3. запускається перерахунок балансу;
4. поля введення очищаються.

Крок 5. Розрахунок балансу.

1. прибуток - сума додається;
2. витрата - сума віднімається;
3. результат відображається в полі балансу.

Крок 6. Видалення запису.

1. запис з вказаним ID видаляється з БД;
2. таблиця оновлюється;
3. баланс перераховується.

Крок 7. Закриття програми.

При натисканні кнопки закриття програма закривається.

Реалізований алгоритм роботи програми дозволяє коректно виконувати додавання, перегляд і видалення фінансових записів із постійним оновленням відображення даних у таблиці та автоматичним перерахунком загального балансу з урахуванням типу операції. Використання перевірки введених даних підвищує надійність роботи системи та запобігає помилкам користувача. Програмне забезпечення забезпечує актуальність фінансової інформації в реальному часі, стабільну роботу з SQL LocalDB та коректне завершення роботи, що робить його зручним і практичним інструментом для персонального обліку доходів і витрат.

3.3. Загальна UML діаграма роботи застосунку

Загальна UML діаграма роботи застосунку керування фінансами (рис. 3.1)

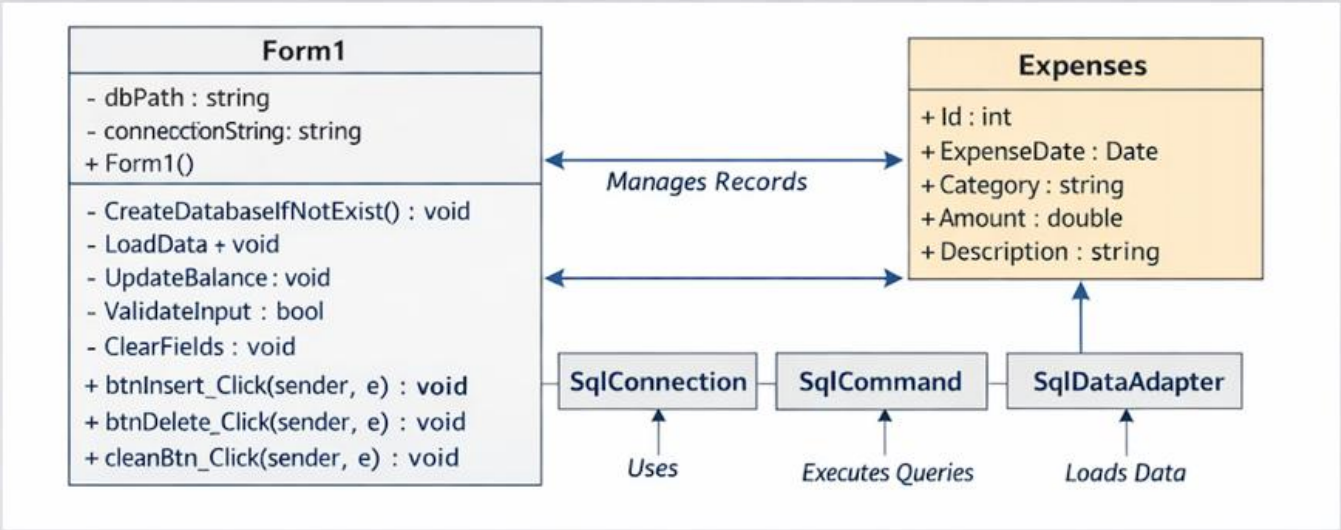

Рисунок 3.1 – загальна UML діаграма

3.4. UML діаграма роботи бази даних застосунку

UML діаграма роботи бази даних застосунку керування фінансами (рис. 3.2)

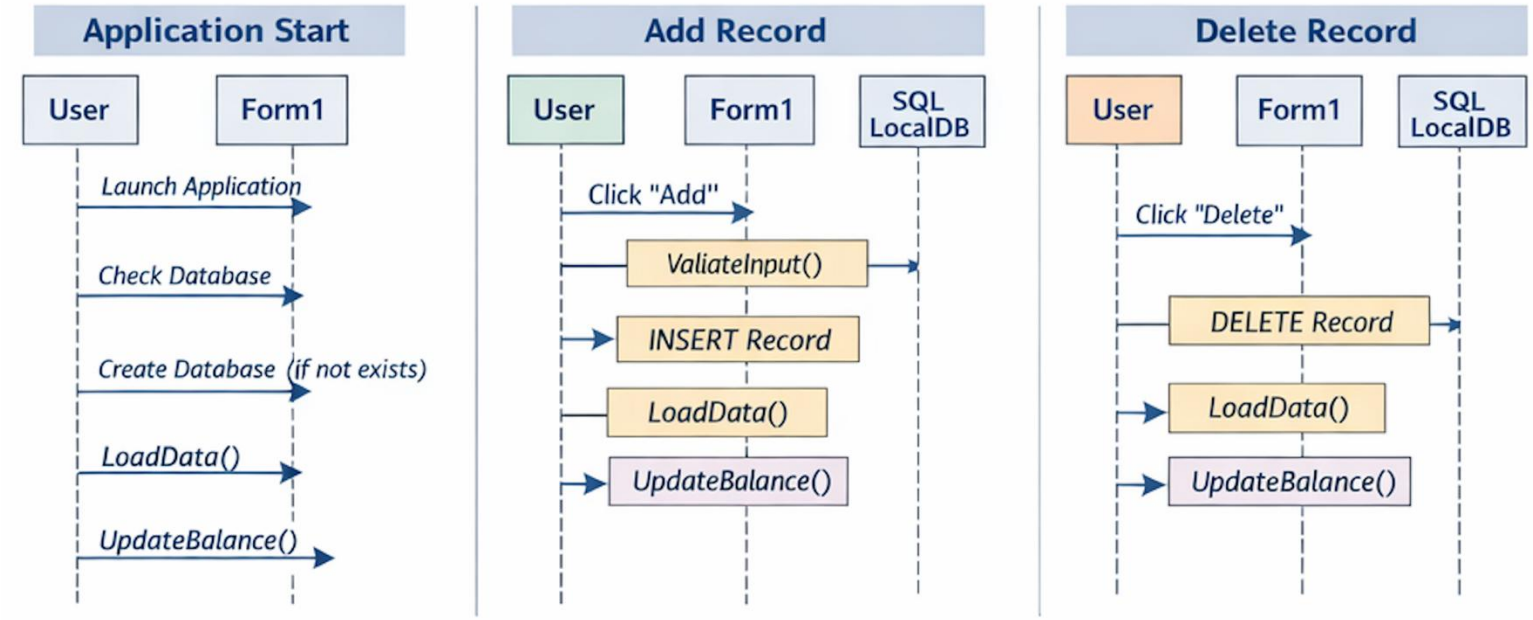

Рисунок 3.2 – UML діаграма роботи бази даних

4. ПРАКТИЧНА ЧАСТИНА

4.1. Обґрунтування вибору програмних засобів

Для розробки програмного забезпечення для обліку фінансових витрат та доходів було обрано мову програмування C# у поєднанні з платформою .NET, що обумовлено рядом технічних, функціональних та практичних переваг. По-перше, C# є об’єктно-орієнтованою мовою програмування, що дозволяє створювати чітко структуровані програмні продукти з високим рівнем повторного використання коду, можливістю наслідування, інкапсуляції та поліморфізму. Це спрощує підтримку, модернізацію та масштабування програмного забезпечення, що особливо важливо для систем, які працюють з фінансовими даними та потребують надійності й стабільності.

По-друге, платформа .NET надає широкий спектр готових бібліотек та компонентів для реалізації різних аспектів системи. Зокрема, .NET включає інструменти для роботи з базами даних, організації взаємодії між клієнтською та серверною частинами, обробки подій та реалізації бізнес-логіки. Це дозволяє суттєво скоротити час розробки та підвищити якість програмного продукту. Крім того, платформа підтримує багатопоточність, що забезпечує швидку та ефективну обробку великих обсягів фінансових даних, а також надійність роботи програми при виконанні складних операцій.

По-третє, використання C# дозволяє ефективно створювати сучасний користувацький інтерфейс за допомогою Windows Forms або WPF, що забезпечує інтуїтивно зрозумілу взаємодію користувача з системою. Зручний інтерфейс є ключовим фактором для ефективного ведення обліку фінансів, оскільки він дозволяє швидко вводити, редагувати та аналізувати дані, формувати звіти та отримувати оперативну інформацію про фінансовий стан.

Також важливим фактором вибору є інтеграція C# та Visual Studio як середовища розробки. Visual Studio надає потужні засоби для налагодження коду, тестування, профілювання продуктивності та управління версіями проекту. Це дозволяє розробнику оперативно виявляти та усувати помилки, забезпечувати стабільність роботи програми та підтримувати високий рівень якості програмного забезпечення.

Крім того, C# забезпечує високу продуктивність та безпеку програмного коду, підтримує сучасні принципи об’єктно-орієнтованого програмування і дозволяє застосовувати різні підходи до організації обробки даних, включно з асинхронними операціями та роботою з потоками. Ці можливості роблять C# оптимальним вибором для створення систем обліку фінансів, що потребують надійності, точності розрахунків та збереження цілісності даних.

Таким чином, вибір мови програмування C# та платформи.NET обґрунтований поєднанням простоти розробки, високої функціональної гнучкості, надійності, безпеки та можливостей створення сучасного, ефективного та зручного програмного забезпечення для обліку фінансових витрат і доходів. Використання цих технологій забезпечує реалізацію всіх поставлених завдань, дозволяє автоматизувати облік фінансів і створює умови для подальшого розвитку та модернізації програмного продукту.

4.2. Опис програмної реалізації

Програмне забезпечення для обліку фінансових витрат та доходів розроблено на базі мови програмування C# [9, 10]. Для розробки програми використано Visual Studio, це середовище підтримує розширення, які додають нові функції, підсвічування синтаксису, автодоповнення коду та інтеграцію з системами контролю версій, такими як Git. (рис. 4.1)

Рисунок 4.1 – Microsoft Visual Studio

Спочатку створюється новий проект, потім розробляється вікно входу у систему з пін-кодом для додаткової безпеки. (рис. 4.2)

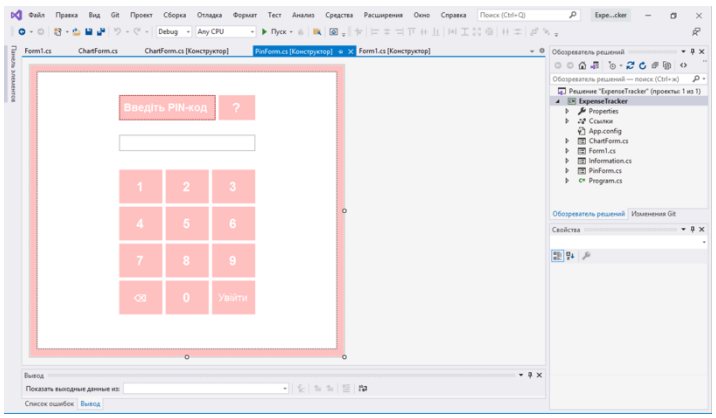

Рисунок 4.2 – Дизайн вікна з пін-кодом

Після розміщення елементів вікна пін-коду, необхідно запрограмувати кожну кнопку і зробити обробку кожної можливої помилки. (рис. 4.3)

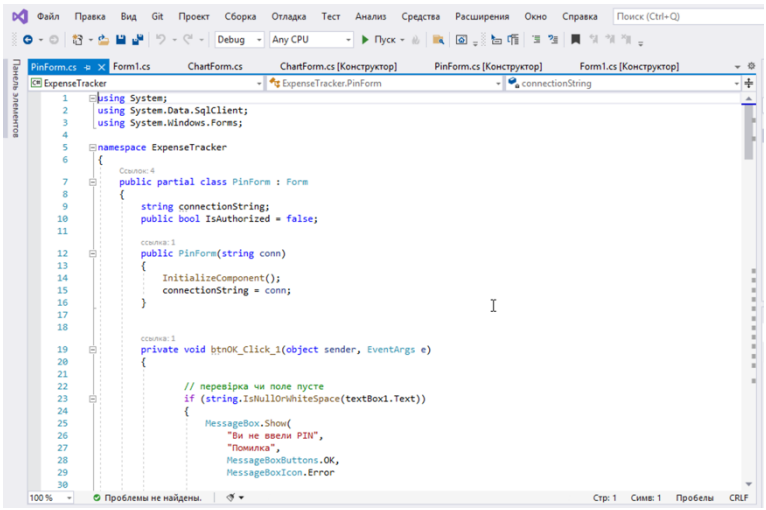

Рисунок 4.3 – Обробка помилок при введенні пін-коду

Також необхідно запрограмувати, щоб при першому запуску програма зберегла пін-код до бази даних SQL, а потім зрівнювала введений користувачем пін-код з тим що збережений в базі. (рис. 4.4)

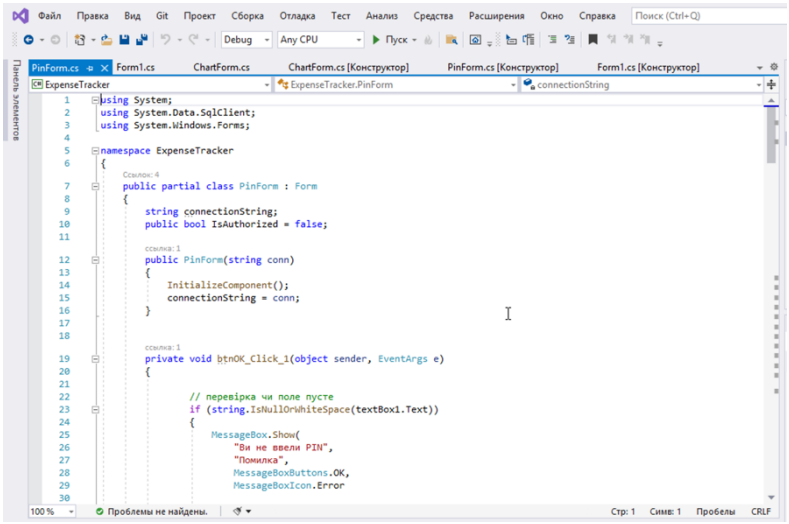

Рисунок 4.4 – Програмування запису пін-коду до бази даних

Додатково у вікні входу в систему створено кнопку “Довідка”. При натисканні користувач зможе ознайомитись детальніше як працює пін-код в системі. (рис. 4.5)

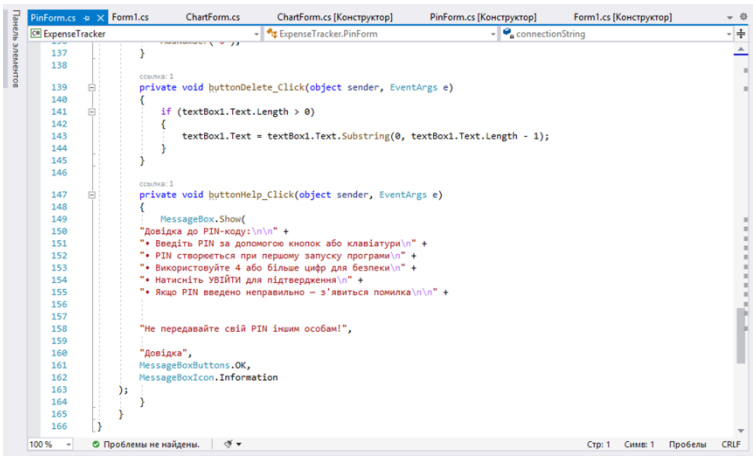

Рисунок 4.5 – Програмування вікна довідки

Після вікна з пін кодом необхідно розробити головний екран застосунку обліку витрат та доходів. (рис. 4.6)

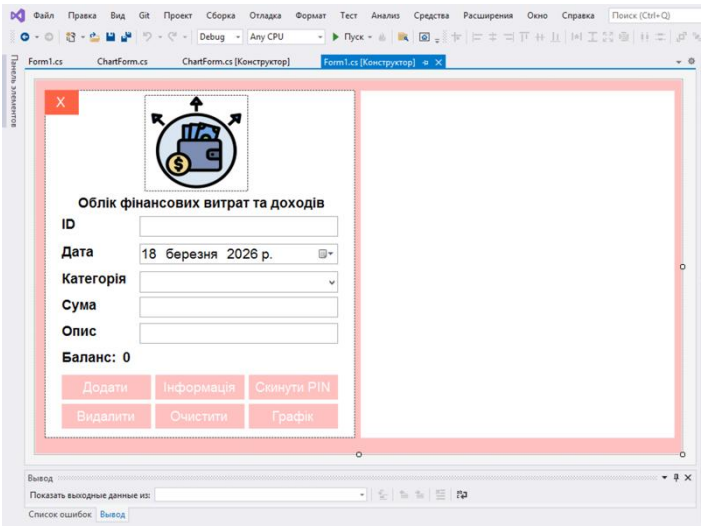

Рисунок 4.6 – Головний екран застосунку

Після створення головної форми з основними елементами, необхідно прописати код створення таблиці бази даних. (рис. 4.7)

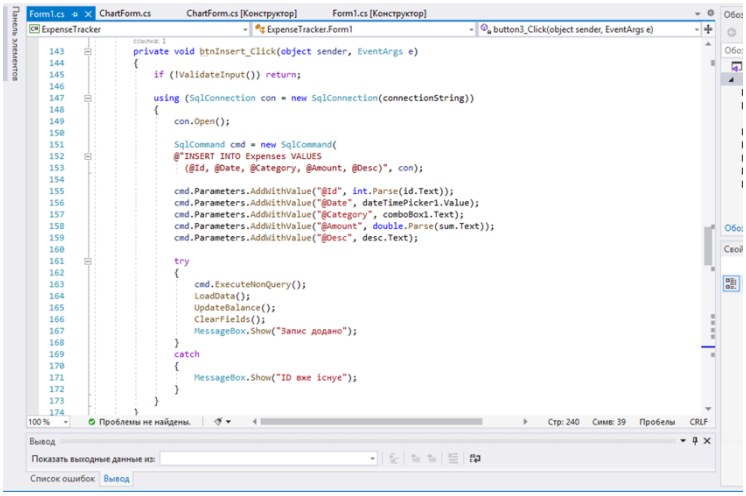

Рисунок 4.7 – Написання коду створення таблиці

Щоб забезпечити збереження даних користувача, необхідно налаштувати підключення до локальної SQL-бази даних. (рис. 4.8)

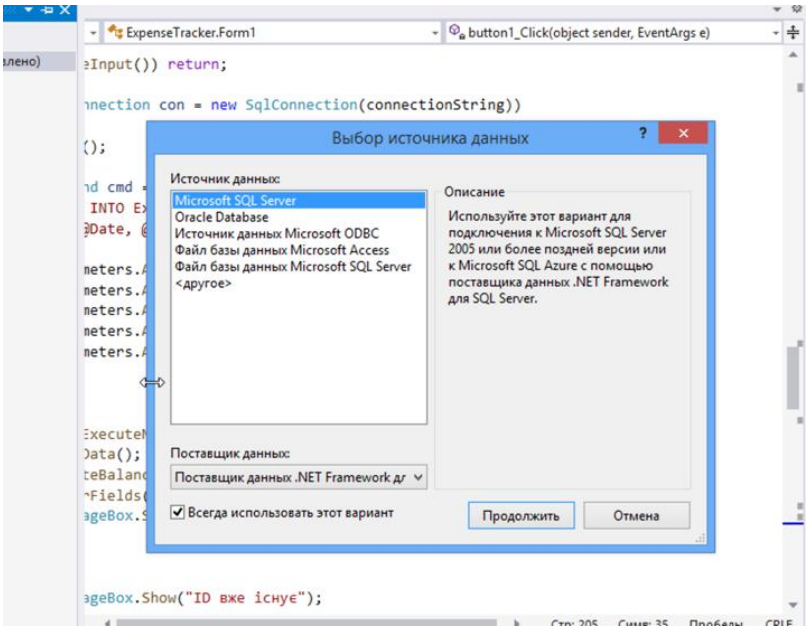

Рисунок 4.8 – Підключення до локальної бази даних SQL

Також було реалізовано обробку можливих помилок застосунку. (рис. 4.9)

```
ссылка: 1
private bool ValidateInput()
{
    if (!int.TryParse(id.Text, out _))
    {
        MessageBox.Show("ID має бути числом");
        return false;
    }

    if (comboBox1.SelectedIndex == -1)
    {
        MessageBox.Show("Оберіть тип операції");
        return false;
    }

    if (!double.TryParse(sum.Text, out _))
    {
        MessageBox.Show("Сума має бути числом");
        return false;
    }

    return true;
}
```

Рисунок 4.9 – Обробка помилок

Для правильного підрахунку балансу необхідно розробити механізм, який буде розрізняти де прибуток а де витрата. (рис. 4.10)

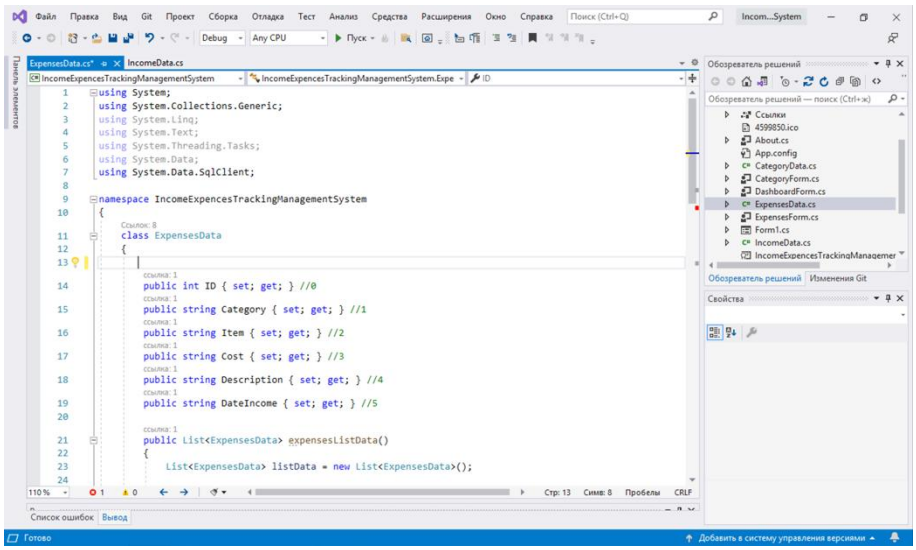

Рисунок 4.10 – Розробка механізму підрахунку балансу

Додатково, необхідно створити інформаційне вікно з загальною інформацією про програмний продукт. (рис. 4.11)

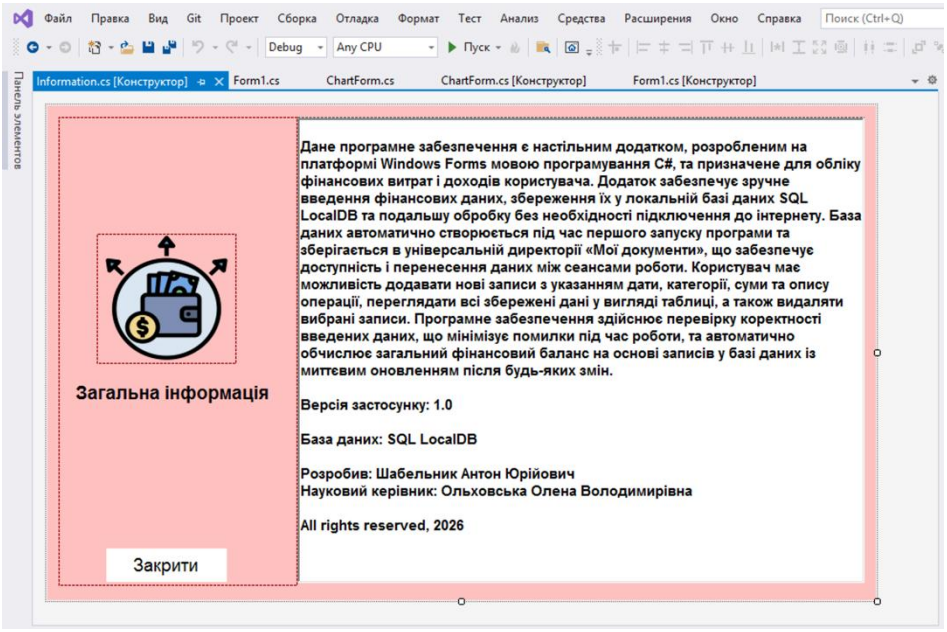

Рисунок 4.11 – Інформаційне вікно

Для візуалізації витрат та доходів було запрограмовано на окремому вікні графік витрат та доходів. (рис. 4.12)

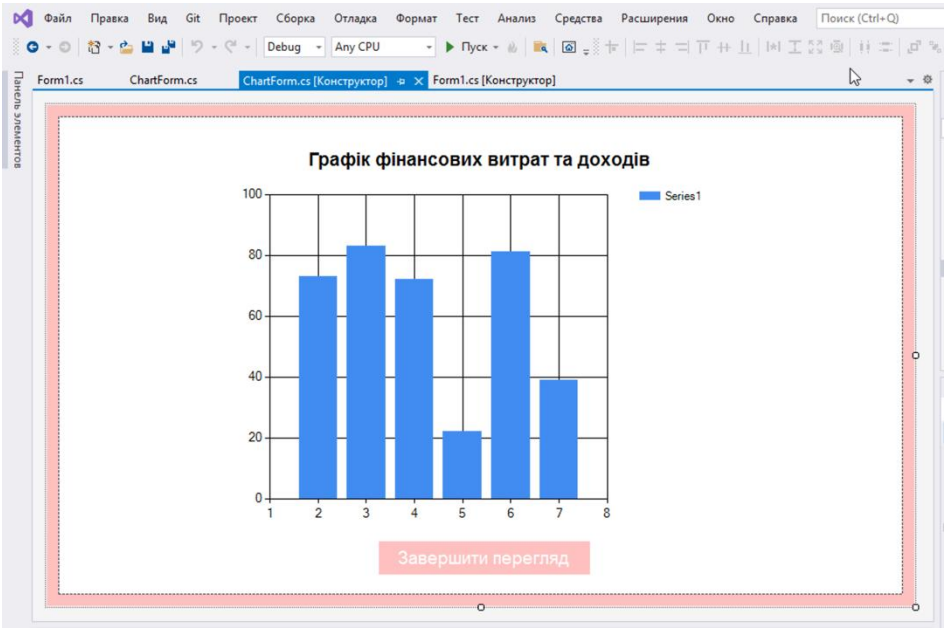

Рисунок 4.12 – Графік витрат та доходів

Вся інформація для графіку витрат та доходів підтягується з локальної бази даних SQL, у яку користувач раніше додав свої витрати та доходи, додатково прописаний функціонал у програмному коді застосунку. (рис. 4.13)

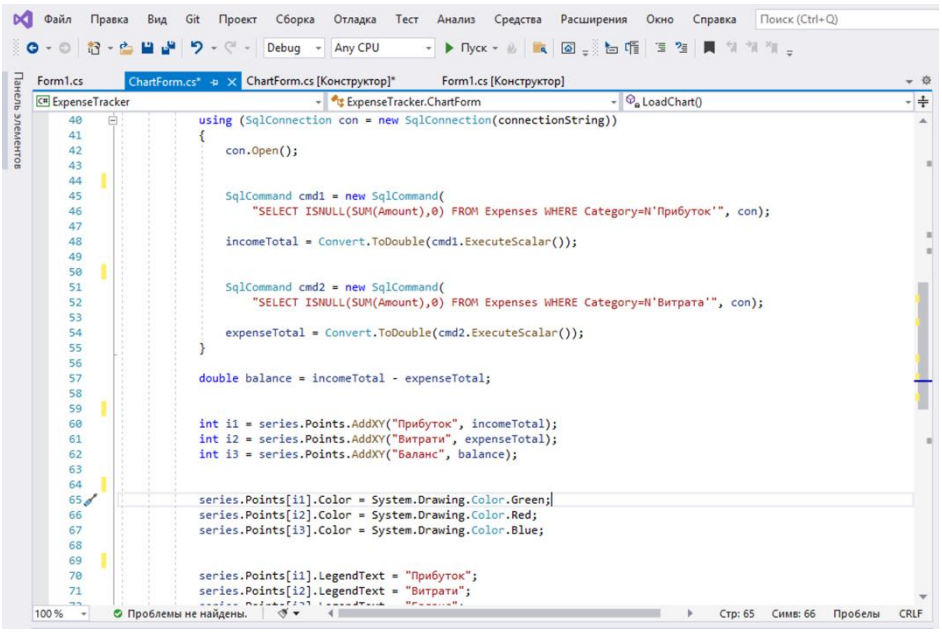

Рисунок 4.13 – Реалізація відображення записів з бази даних у графіку витрат та доходів

Програмний код було створено в середовищі розробки Microsoft Visual Studio 2019 із застосуванням мови програмування C# та технології Windows Forms. Водночас Visual Studio 2019 забезпечила наявність інструментів для візуального проєктування інтерфейсу, автоматичного створення коду компонентів форми та зручної обробки подій. У застосунку використано стандартні бібліотеки .NET Framework, зокрема простір імен System.Data.SqlClient для взаємодії з базою даних SQL Server LocalDB, що дозволило реалізувати функції додавання, редагування, видалення й відображення даних у програмі.

4.3. Опис роботи застосунку для обліку фінансових витрат та доходів

Після запуску застосунку для обліку фінансових доходів і витрат користувач бачить екран з кнопками і полем для введення пін-коду. Користувачу необхідно придумати пін-код та увести його, програма цей код запам’ятає і далі він буде використовуватись для входу в систему. (рис. 4.14)

Рисунок 4.14 – Вікно пін-коду

Для додаткової довідки необхідно натиснути кнопку з знаком питання. (рис. 4.15)

Рисунок 4.15 – Вікно довідки

Також оброблено можливі помилки, якщо користувач не увів пін-код і натиснув кнопку “Увійти”, виведеться помилка. (рис. 4.16)

Рисунок 4.16 – Обробка помилок

Після того як користувач придумав пін-код та перший раз його увів, система збереже код до бази даних і виведе додаткове підтвердження у вигляді повідомлення. (рис. 4.17)

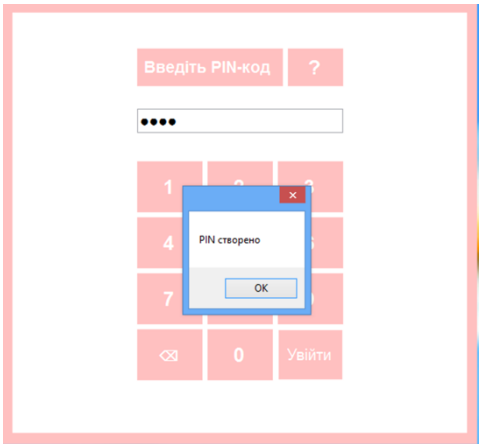

Рисунок 4.17 – Збереження пін-коду

Після збереження пін-коду користувач бачить головний екран застосунку, на якому відображаються записи з бази даних разом із полями введення та елементами керування. У правій частині інтерфейсу розташована таблиця з даними, а в лівій — поля для введення інформації та кнопки керування. (рис. 4.18)

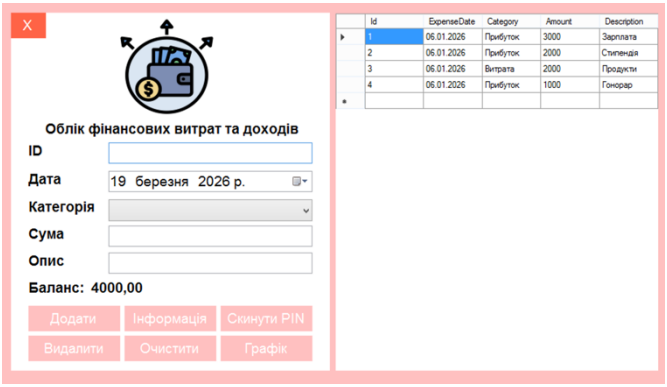

Рисунок 4.18 – Головне вікно програми

Якщо користувач натисне кнопку “Інформація” у лівій частині вікна, він отримає загальну інформацію. (рис. 4.19)

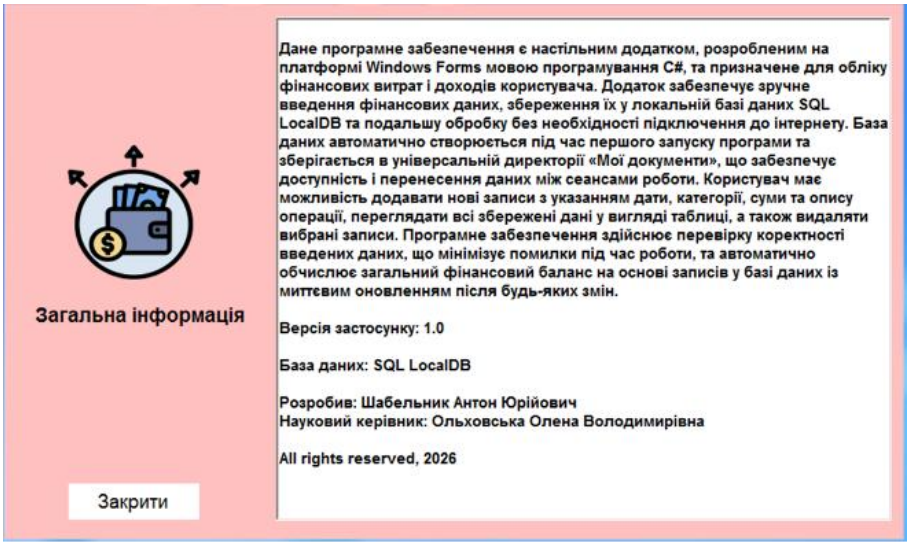

Рисунок 4.19 – Загальна інформація

Щоб додати запис з витратою або доходом, необхідно спочатку заповнити всі поля у лівій частині екрану. (рис. 4.20)

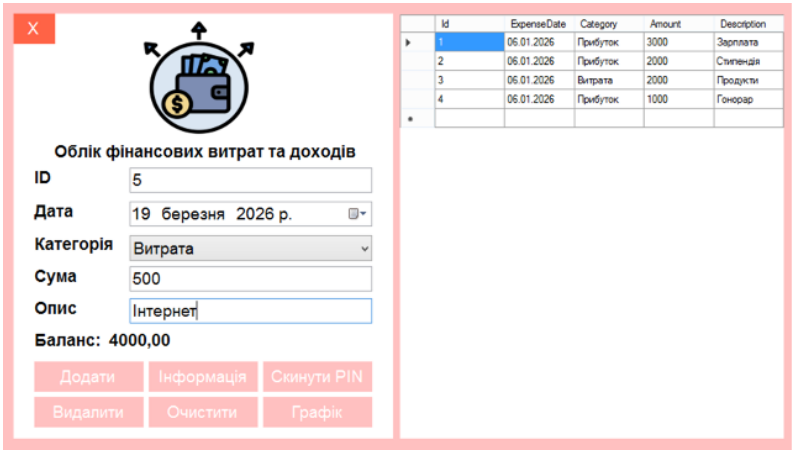

Рисунок 4.20 – Заповнення полів з витратою

Якщо не заповнити поле, додатково оброблено цю помилку і застосунок виведе вікно помилки. (рис. 4.21)

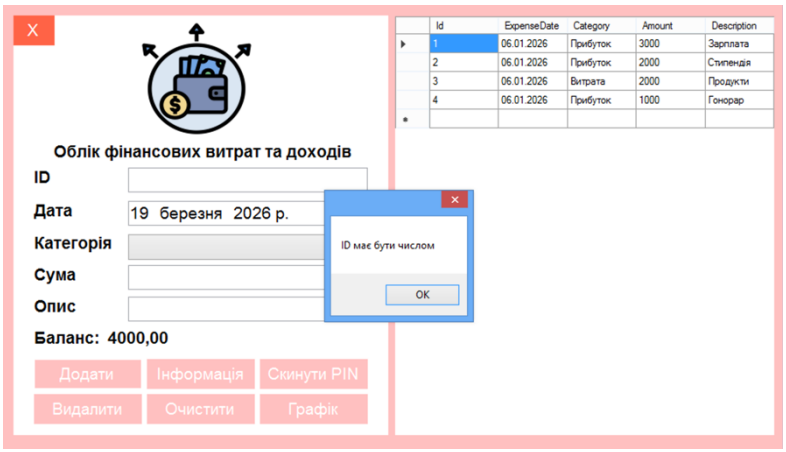

Рисунок 4.21 – Вікно помилки

Якщо всі поля у лівій частині заповнені, запис буде успішно додано до бази даних і буде відображатися у правій частині екрану. (рис. 4.22)

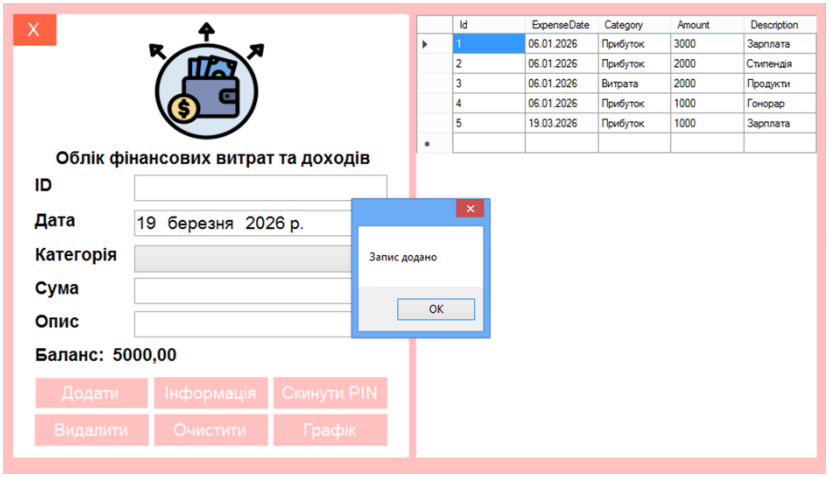

Рисунок 4.22 – Запис додано

Якщо користувачу потрібно видалити запис, спочатку треба увести ID запису, потім натиснути кнопку “Видалити”, обраний запис видалиться. (рис. 4.23)

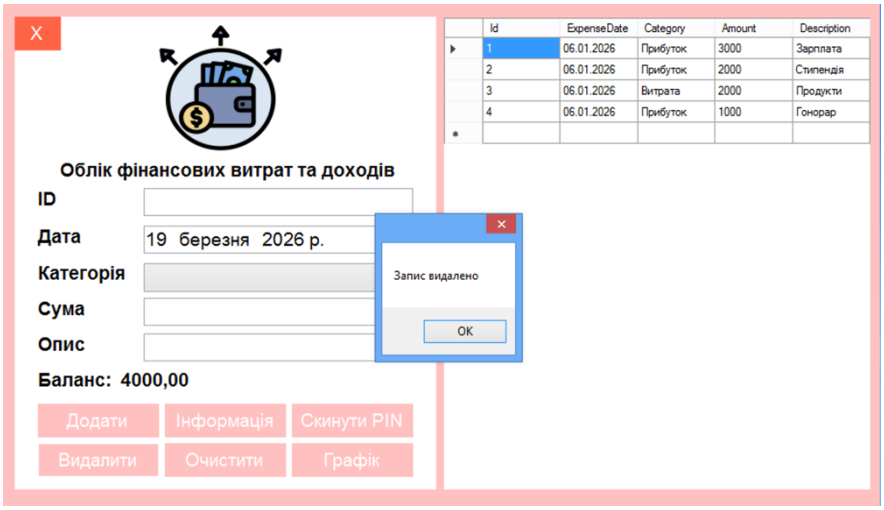

Рисунок 4.23 – Видалення запису

Якщо користувач заповне поле ID і не вибере категорію – застосунок додатково нагадає про необхідність вибору типу операції. (рис. 4.24)

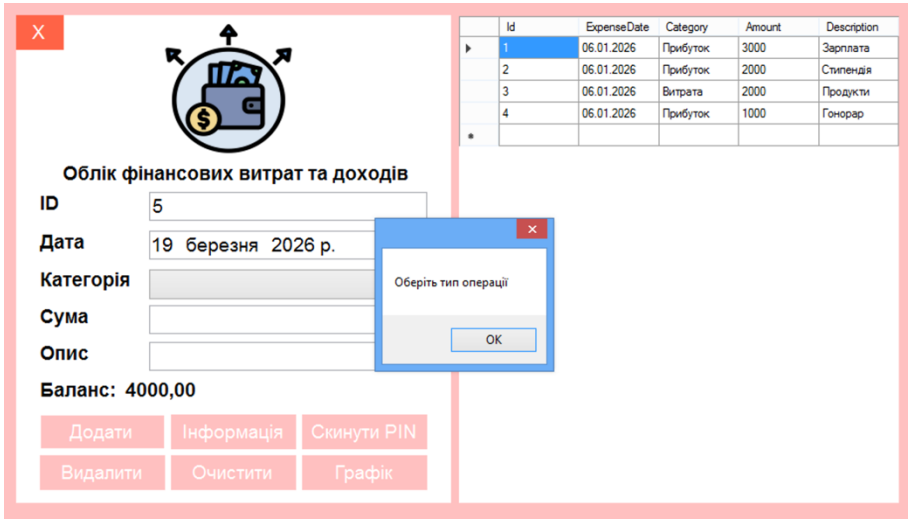

Рисунок 4.24 – Сповіщення нагадування

Для того щоб швидко очистити всі поля, користувачу потрібно лише натиснути кнопку “Очистити”, всі поля будуть швидко очищені. (рис. 4.25)

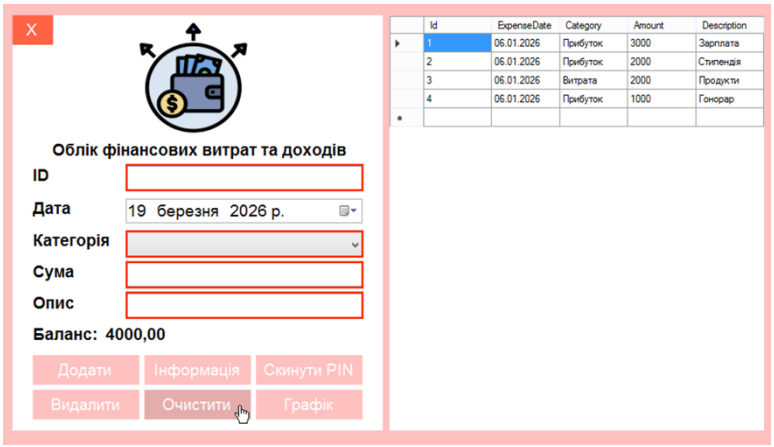

Рисунок 4.25 – Очищення полів

Якщо користувачу необхідно поміняти або скинути поточний пін-код входу у застосунок, на головному екрані потрібно натиснути кнопку “Скинути PIN”, після цього збережений у базу даних пін-код видалиться і при вході в систему потрібно буде вписати новий пін-код. (рис. 4.26)

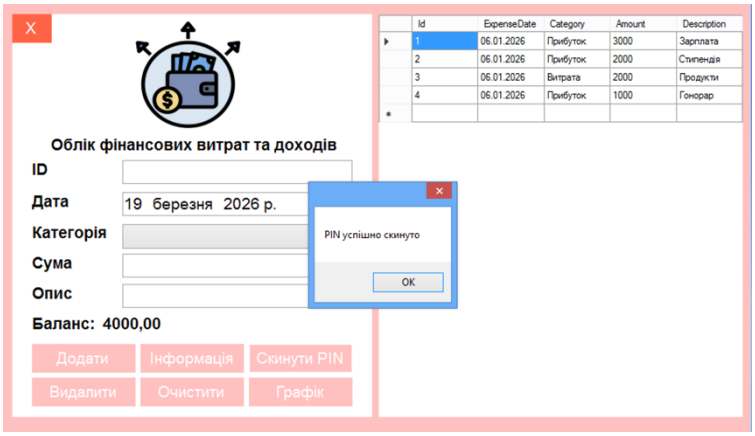

Рисунок 4.26 – Скидання пін-коду

Додатково, якщо користувач натисне на головному екрані кнопку “Графік”, йому відкриється візуальний графік фінансових витрат та доходів. На графіку відображаються додані користувачем прибутки та витрати а також баланс. (рис. 4.27)

Рисунок 4.27 – Графік витрат та прибутків

Додатково програмне забезпечення протестовано на різних конфігураціях комп’ютерів з метою перевірки стабільності роботи, сумісності та коректності виконання основних функцій. (рис. 4.31)

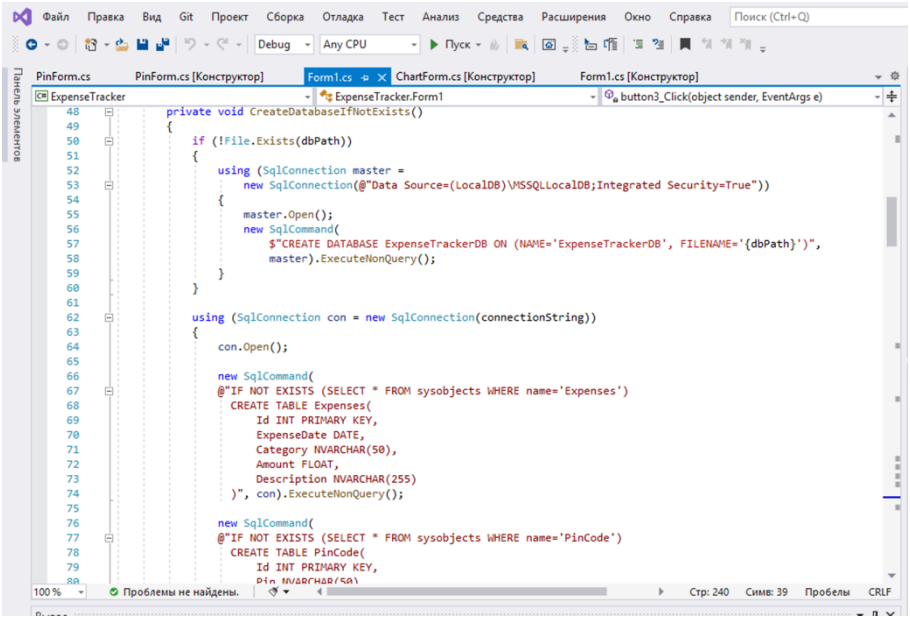

Рисунок 4.31 – Тестування програмного забезпечення

Робота програми можлива лише за умови встановленого програмного забезпечення локальної SQL-бази даних. (рис. 4.32)

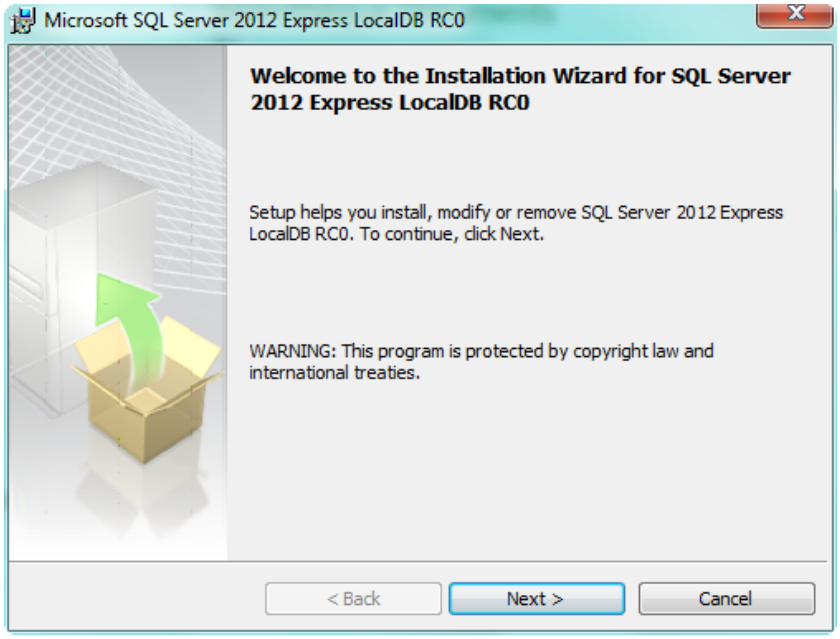

Рисунок 4.32 – SQL local database

У результаті розробки було створено настільний застосунок для обліку фінансових витрат та доходів, який забезпечує зручне введення, збереження, перегляд і видалення фінансових операцій. Програма автоматично створює локальну базу даних, зберігає її в універсальному шляху та гарантує збереження даних між запусками. Реалізовано перевірку коректності введених даних, наочне відображення записів у таблиці та автоматичний розрахунок загального балансу з урахуванням прибутків і витрат. Розроблений застосунок є простим у використанні, надійним та може бути використаний як основа для подальшого розширення функціоналу системи фінансового обліку.

ВИСНОВКИ

Отже, у ході виконання роботи було проведено комплексний аналіз предметної області, що дозволив визначити основні вимоги до програмного забезпечення для обліку фінансових витрат та доходів. Було встановлено, що ефективне управління фінансами потребує автоматизації процесів введення, збереження та обробки даних, а також наявності зручного користувацького інтерфейсу та можливостей аналітики.

Розробка архітектури системи дозволила побудувати логічну структуру програмного продукту, що забезпечує ефективну взаємодію між модулем обліку даних, базою даних і користувацьким інтерфейсом, а також гарантує масштабованість та зручність подальшої модернізації. Було реалізовано програмне забезпечення мовою C# з використанням об’єктно-орієнтованого підходу, що забезпечило структурованість коду, можливість повторного використання компонентів та легкість підтримки програми.

Система забезпечує ведення обліку фінансових операцій, включно з функціоналом створення, редагування та видалення записів, розрахунком балансу доходів і витрат.

Проведене тестування підтвердило коректність роботи програмного забезпечення, надійність збереження даних та відповідність поставленим вимогам. Розроблений користувацький інтерфейс є інтуїтивно зрозумілим і дозволяє користувачу швидко виконувати необхідні дії з фінансовими даними.

Таким чином, виконана робота підтвердила доцільність використання автоматизованих систем обліку фінансів, продемонструвала практичну ефективність застосування мови програмування C# та об’єктно-орієнтованого підходу, а також дозволила створити функціональний інструмент для контролю та аналізу фінансового стану користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ольховська О.В., Черненко О. О. методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня бакалавра / О. О. Черненко., Ольховська О.В. – Полтава : ПУЕТ, 2025. – 58 с.
2. Що таке CRUD? Його функції, переваги та приклади [Електронний ресурс]. – Режим доступу: https://highload.tech/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/
3. Огляд програмного забезпечення Money Pro [Електронний ресурс]. – Режим доступу: https://apps.microsoft.com/detail/9nqw069mjts2?hl=uk-UA&gl=US
4. Огляд застосунку Spending Tracker [Електронний ресурс]. – Режим доступу: https://apps.microsoft.com/detail/9wzdncrfhmw8?hl=uk-UA&gl=UA
5. Розробка програмного забезпечення [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/5118185
6. Системне програмне забезпечення [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/2426774/page:10/
7. C# Підручник [Електронний ресурс]. – Режим доступу: https://www.google.com/url?sa=t&source
8. Категорія: Мова програмування C# [Електронний ресурс]. – Режим доступу: https://abitap.com/category/c/#google_vignette
9. What is C# and use cases of C#? [Електронний ресурс]. – Режим доступу: https://www.devopsschool.com/blog/what-is-c#-net-and-use-cases-of-c#-net/
10. Створення програми в Visual Studio 2019 [Електронний ресурс]. – Режим доступу: https://learn.ztu.edu.ua/mod/page/view.php?id=9976
11. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання: ДСТУ 7.1-2006. – [Чинний від 2007-07-01]. – К. : Держспоживстандарт України, 2007. – 47 с.

12. Microsoft Visual Studio [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
13. development with Visual Studio [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/?view=vs-2022
14. Get Started [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/cs/cs_getstarted.php
15. Visual Studio [Електронний ресурс]. – Режим доступу: https://www.halvorsen.blog/documents/programming/csharp/csharp.php
16. Створення програми в Visual Studio 2019 [Електронний ресурс]. – Режим доступу: https://learn.ztu.edu.ua/mod/page/view.php?id=9976
17. Microsoft Visual Studio [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
18. Навіщо потрібні програми для обліку [Електронний ресурс]. – Режим доступу: https://smartoblik.com/nav%D1%96shcho_potr%D1%96bn%D1%96_prohramy_dlya_obliku/
19. Об'єктно-орієнтоване програмування (Advanced Encryption Standard instructions) [Електронний ресурс]. – Режим доступу: https://vseosvita.ua/test/elementy-teorii-obiektno-oriientovanoho.
20. Методологія та організація ведення обліку фінансів [Електронний ресурс]. – Режим доступу: https://magazine.faaf.org.ua/metodologiya-ta-organizaciya-vedennya-obliku-v-umovah-avtomatizacii.html

ДОДАТОК А. КОД ПРОГРАМИ

```
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
            this.panel1 = new System.Windows.Forms.Panel();
            this.comboBox1 = new System.Windows.Forms.ComboBox();
            this.button2 = new System.Windows.Forms.Button();
            this.label7 = new System.Windows.Forms.Label();
            this.balance = new System.Windows.Forms.Label();
            this.cleanBtn = new System.Windows.Forms.Button();
            this.button1 = new System.Windows.Forms.Button();
            this.btnDelete = new System.Windows.Forms.Button();
            this.btnInsert = new System.Windows.Forms.Button();
            this.dateTimePicker1 = new System.Windows.Forms.DateTimePicker();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.desc = new System.Windows.Forms.TextBox();
            this.Label1 = new System.Windows.Forms.Label();
            this.sum = new System.Windows.Forms.TextBox();
            this.id = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.dataGridView1 = new System.Windows.Forms.DataGridView();
            this.panel1.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();

((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).BeginInit();
            this.SuspendLayout();
            //
            // panel1
            //
            this.panel1.BackColor = System.Drawing.Color.White;
            this.panel1.Controls.Add(this.comboBox1);
            this.panel1.Controls.Add(this.button2);
            this.panel1.Controls.Add(this.label7);
            this.panel1.Controls.Add(this.balance);
            this.panel1.Controls.Add(this.cleanBtn);
            this.panel1.Controls.Add(this.button1);
            this.panel1.Controls.Add(this.btnDelete);
            this.panel1.Controls.Add(this.btnInsert);
```

```
            this.panel1.Controls.Add(this.dateTimePicker1);
            this.panel1.Controls.Add(this.pictureBox1);
            this.panel1.Controls.Add(this.desc);
            this.panel1.Controls.Add(this.Label1);
            this.panel1.Controls.Add(this.sum);
            this.panel1.Controls.Add(this.id);
            this.panel1.Controls.Add(this.label2);
            this.panel1.Controls.Add(this.label3);
            this.panel1.Controls.Add(this.label6);
            this.panel1.Controls.Add(this.label4);
            this.panel1.Controls.Add(this.label5);
            this.panel1.Location = new System.Drawing.Point(13, 14);
            this.panel1.Name = "panel1";
            this.panel1.Size = new System.Drawing.Size(438, 491);
            this.panel1.TabIndex = 0;
            //
            // comboBox1
            //
            this.comboBox1.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.comboBox1.FormattingEnabled = true;
            this.comboBox1.Location = new System.Drawing.Point(134, 256);
            this.comboBox1.Name = "comboBox1";
            this.comboBox1.Size = new System.Drawing.Size(280, 30);
            this.comboBox1.TabIndex = 25;
            //
            // button2
            //
            this.button2.BackColor = System.Drawing.Color.Tomato;
            this.button2.Cursor = System.Windows.Forms.Cursors.Hand;
            this.button2.FlatAppearance.BorderSize = 0;
            this.button2.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.button2.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.button2.ForeColor = System.Drawing.Color.White;
            this.button2.Location = new System.Drawing.Point(0, 0);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(48, 35);
            this.button2.TabIndex = 24;
            this.button2.Text = "X";
            this.button2.UseVisualStyleBackColor = false;
            //
            // label7
```

```
            //
            this.label7.AutoSize = true;
            this.label7.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label7.Location = new System.Drawing.Point(20, 366);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(87, 22);
            this.label7.TabIndex = 23;
            this.label7.Text = "Б а л а н с :";
            //
            // balance
            //
            this.balance.AutoSize = true;
            this.balance.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.balance.Location = new System.Drawing.Point(106, 366);
            this.balance.Name = "balance";
            this.balance.Size = new System.Drawing.Size(21, 22);
            this.balance.TabIndex = 22;
            this.balance.Text = "0\r\n";
            //
            // cleanBtn
            //
            this.cleanBtn.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(255)))), ((int)(((byte)(192)))),
((int)(((byte)(192)))));
            this.cleanBtn.Cursor = System.Windows.Forms.Cursors.Hand;
            this.cleanBtn.FlatAppearance.BorderSize = 0;
            this.cleanBtn.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.cleanBtn.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.cleanBtn.ForeColor = System.Drawing.Color.White;
            this.cleanBtn.Location = new System.Drawing.Point(218, 445);
            this.cleanBtn.Name = "cleanBtn";
            this.cleanBtn.Size = new System.Drawing.Size(126, 35);
            this.cleanBtn.TabIndex = 21;
            this.cleanBtn.Text = "Оч и с т и т и ";
            this.cleanBtn.UseVisualStyleBackColor = false;
            this.cleanBtn.Click += new System.EventHandler(this.cleanBtn_Click);
            //
            // button1
            //
```

```
            this.button1.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(255)))), ((int)(((byte)(192)))),
((int)(((byte)(192)))));
            this.button1.Cursor = System.Windows.Forms.Cursors.Hand;
            this.button1.FlatAppearance.BorderSize = 0;
            this.button1.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.button1.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.button1.ForeColor = System.Drawing.Color.White;
            this.button1.Location = new System.Drawing.Point(218, 404);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(126, 35);
            this.button1.TabIndex = 20;
            this.button1.Text = "І нфо р ма ц і я";
            this.button1.UseVisualStyleBackColor = false;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // btnDelete
            //
            this.btnDelete.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(255)))), ((int)(((byte)(192)))),
((int)(((byte)(192)))));
            this.btnDelete.Cursor = System.Windows.Forms.Cursors.Hand;
            this.btnDelete.FlatAppearance.BorderSize = 0;
            this.btnDelete.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.btnDelete.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.btnDelete.ForeColor = System.Drawing.Color.White;
            this.btnDelete.Location = new System.Drawing.Point(86, 445);
            this.btnDelete.Name = "btnDelete";
            this.btnDelete.Size = new System.Drawing.Size(126, 35);
            this.btnDelete.TabIndex = 19;
            this.btnDelete.Text = "В и д а л и т и";
            this.btnDelete.UseVisualStyleBackColor = false;
            this.btnDelete.Click += new System.EventHandler(this.btnDelete_Click);
            //
            // btnInsert
            //
            this.btnInsert.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(255)))), ((int)(((byte)(192)))),
((int)(((byte)(192)))));
            this.btnInsert.Cursor = System.Windows.Forms.Cursors.Hand;
            this.btnInsert.FlatAppearance.BorderSize = 0;
```

```
            this.btnInsert.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.btnInsert.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.btnInsert.ForeColor = System.Drawing.Color.White;
            this.btnInsert.Location = new System.Drawing.Point(86, 404);
            this.btnInsert.Name = "btnInsert";
            this.btnInsert.Size = new System.Drawing.Size(126, 35);
            this.btnInsert.TabIndex = 18;
            this.btnInsert.Text = "Додати";
            this.btnInsert.UseVisualStyleBackColor = false;
            this.btnInsert.Click += new System.EventHandler(this.btnInsert_Click);
            //
            // dateTimePicker1
            //
            this.dateTimePicker1.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.dateTimePicker1.Location = new System.Drawing.Point(134, 217);
            this.dateTimePicker1.Name = "dateTimePicker1";
            this.dateTimePicker1.Size = new System.Drawing.Size(280, 29);
            this.dateTimePicker1.TabIndex = 12;
            //
            // pictureBox1
            //
            this.pictureBox1.BackColor = System.Drawing.Color.White;
            this.pictureBox1.Image =
((System.Drawing.Image)(resources.GetObject("pictureBox1.Image")));
            this.pictureBox1.Location = new System.Drawing.Point(141, 7);
            this.pictureBox1.Name = "pictureBox1";
            this.pictureBox1.Size = new System.Drawing.Size(146, 136);
            this.pictureBox1.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.StretchImage;
            this.pictureBox1.TabIndex = 1;
            this.pictureBox1.TabStop = false;
            //
            // desc
            //
            this.desc.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.desc.Location = new System.Drawing.Point(134, 329);
            this.desc.Name = "desc";
            this.desc.Size = new System.Drawing.Size(280, 29);
            this.desc.TabIndex = 11;
```

```
            //
            // Label1
            //
            this.Label1.AutoSize = true;
            this.Label1.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.Label1.Location = new System.Drawing.Point(43, 148);
            this.Label1.Name = "Label1";
            this.Label1.Size = new System.Drawing.Size(358, 22);
            this.Label1.TabIndex = 0;
            this.Label1.Text = "Облік фінансових витрат та доходів";
            //
            // sum
            //
            this.sum.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.sum.Location = new System.Drawing.Point(134, 292);
            this.sum.Name = "sum";
            this.sum.Size = new System.Drawing.Size(280, 29);
            this.sum.TabIndex = 10;
            //
            // id
            //
            this.id.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.id.Location = new System.Drawing.Point(134, 178);
            this.id.Name = "id";
            this.id.Size = new System.Drawing.Size(280, 29);
            this.id.TabIndex = 7;
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label2.Location = new System.Drawing.Point(20, 178);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(29, 22);
            this.label2.TabIndex = 2;
            this.label2.Text = "ID";
            //
            // label3
            //
```

```
            this.label3.AutoSize = true;
            this.label3.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label3.Location = new System.Drawing.Point(20, 217);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(55, 22);
            this.label3.TabIndex = 3;
            this.label3.Text = "Да т а ";
            //
            // label6
            //
            this.label6.AutoSize = true;
            this.label6.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label6.Location = new System.Drawing.Point(20, 329);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(59, 22);
            this.label6.TabIndex = 6;
            this.label6.Text = "Оп и с ";
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label4.Location = new System.Drawing.Point(20, 255);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(101, 22);
            this.label4.TabIndex = 4;
            this.label4.Text = "К а т е г о р і я ";
            //
            // label5
            //
            this.label5.AutoSize = true;
            this.label5.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label5.Location = new System.Drawing.Point(20, 292);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(59, 22);
            this.label5.TabIndex = 5;
            this.label5.Text = "Су ма ";
            //
            // dataGridView1
            //
```

```
            this.dataGridView1.AutoSizeColumnsMode =
System.Windows.Forms.DataGridViewAutoSizeColumnsMode.Fill;
            this.dataGridView1.BackgroundColor = System.Drawing.Color.White;
            this.dataGridView1.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
            this.dataGridView1.Location = new System.Drawing.Point(457, 15);
            this.dataGridView1.Name = "dataGridView1";
            this.dataGridView1.Size = new System.Drawing.Size(397, 490);
            this.dataGridView1.TabIndex = 1;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(255)))),
((int)(((byte)(192)))), ((int)(((byte)(192)))));
            this.ClientSize = new System.Drawing.Size(867, 517);
            this.Controls.Add(this.dataGridView1);
            this.Controls.Add(this.panel1);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
            this.Name = "Form1";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Form1";
            this.panel1.ResumeLayout(false);
            this.panel1.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.Label Label1;
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.TextBox id;
        private System.Windows.Forms.TextBox sum;
        private System.Windows.Forms.TextBox desc;
```

```
        private System.Windows.Forms.DateTimePicker dateTimePicker1;
        private System.Windows.Forms.Button btnDelete;
        private System.Windows.Forms.Button btnInsert;
        private System.Windows.Forms.DataGridView dataGridView1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button cleanBtn;
        private System.Windows.Forms.Label balance;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.ComboBox comboBox1;
    }
}
```