Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 2

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування»

Виконала:
студентка групи ІА-33
Самойленко Анастасія

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

Київ 2025

Вступ

Проєктування програмного забезпечення є одним із ключових етапів життєвого циклу системи, оскільки воно визначає структуру, функціональність та взаємодію компонентів майбутньої програми. Одним із найпоширеніших інструментів для моделювання систем є UML (Unified Modeling Language) – уніфікована мова моделювання, яка дозволяє наочно відображати архітектуру, поведінку та взаємозв’язки об’єктів. Використання UML-діаграм забезпечує зрозуміле і формалізоване представлення системи, що спрощує аналіз вимог, комунікацію між розробниками та користувачами, а також подальшу реалізацію проєкту.

У даній лабораторній роботі увага зосереджена на засвоєнні основ побудови UML-діаграм. Зокрема, розглядається вибір інструменту для створення моделей, побудова діаграм варіантів використання, які описують функціональні можливості системи з точки зору користувачів, а також створення сценаріїв для цих варіантів. Окрім цього, важливою частиною роботи є розробка діаграм класів предметної області, що дозволяє формалізувати структуру об’єктів і визначити їх зв’язки.

Таким чином, виконання роботи сприятиме формуванню практичних навичок моделювання програмних систем, що є невід’ємною складовою професійної підготовки фахівців у сфері інформаційних технологій.

Теоретичні відомості

Вступ до мови UML

UML (Unified Modeling Language) – уніфікована мова моделювання, що застосовується для опису, візуалізації та документування програмних систем. Вона містить набір діаграм, які дозволяють моделювати як статичні, так і динамічні аспекти системи, роблячи проєктування більш структурованим та зрозумілим.

Діаграма варіантів використання (Use-Case Diagram)

Діаграма варіантів використання описує функціональність системи з точки зору користувачів і показує, які дії можуть виконувати актори та як вони взаємодіють із системою. Вона є одним з перших кроків моделювання вимог до програмного забезпечення.

Актор (Actor)

Актор – це зовнішній користувач або система, яка взаємодіє з розроблюваною системою. Це може бути людина, інше програмне забезпечення чи технічний пристрій, що має визначені ролі.

Варіанти використання (Use Case)

Варіант використання описує функціональність, яка має цінність для актора. Він представляє завершену дію або сценарій взаємодії між користувачем і системою.

Відносини на діаграмі варіантів використання

Основними відносинами є: асоціація – зв’язок між актором і варіантом використання; include – включення одного сценарію до іншого; extend – розширення поведінки варіанту за певних умов; generalization – узагальнення між акторами або варіантами використання.

Сценарії використання

Сценарій використання є текстовим описом послідовності дій, що виконуються актором і системою для досягнення певної мети. Він деталізує діаграму варіантів використання.

Діаграми класів

Діаграма класів описує структуру системи у вигляді класів, їхніх атрибутів, методів і зв’язків. Вона показує статичну модель предметної області та взаємозв’язки між об’єктами.

Логічна структура бази даних

Логічна структура БД визначає таблиці, їх атрибути, ключі та зв’язки між сутностями без урахування фізичної реалізації. Це абстрактний рівень, що забезпечує коректність і узгодженість даних.

Проєктування бази даних

Проєктування БД включає визначення сутностей і їхніх атрибутів, встановлення зв’язків, нормалізацію та створення схеми, яка забезпечує ефективне зберігання, пошук та обробку інформації.

Хід роботи

1. Для початку проаналізуємо обрану тематику – «Особиста бухгалтерія». Система особистої бухгалтерії дає змогу користувачу ефективно контролювати свої фінансові потоки, вести облік доходів і витрат, планувати бюджет, формувати звіти та здійснювати синхронізацію з банківськими рахунками. Основними користувачами є звичайний користувач. На основі аналізу було побудовано Use Case діаграму (рис. 1), яка відображає головні варіанти використання системи.

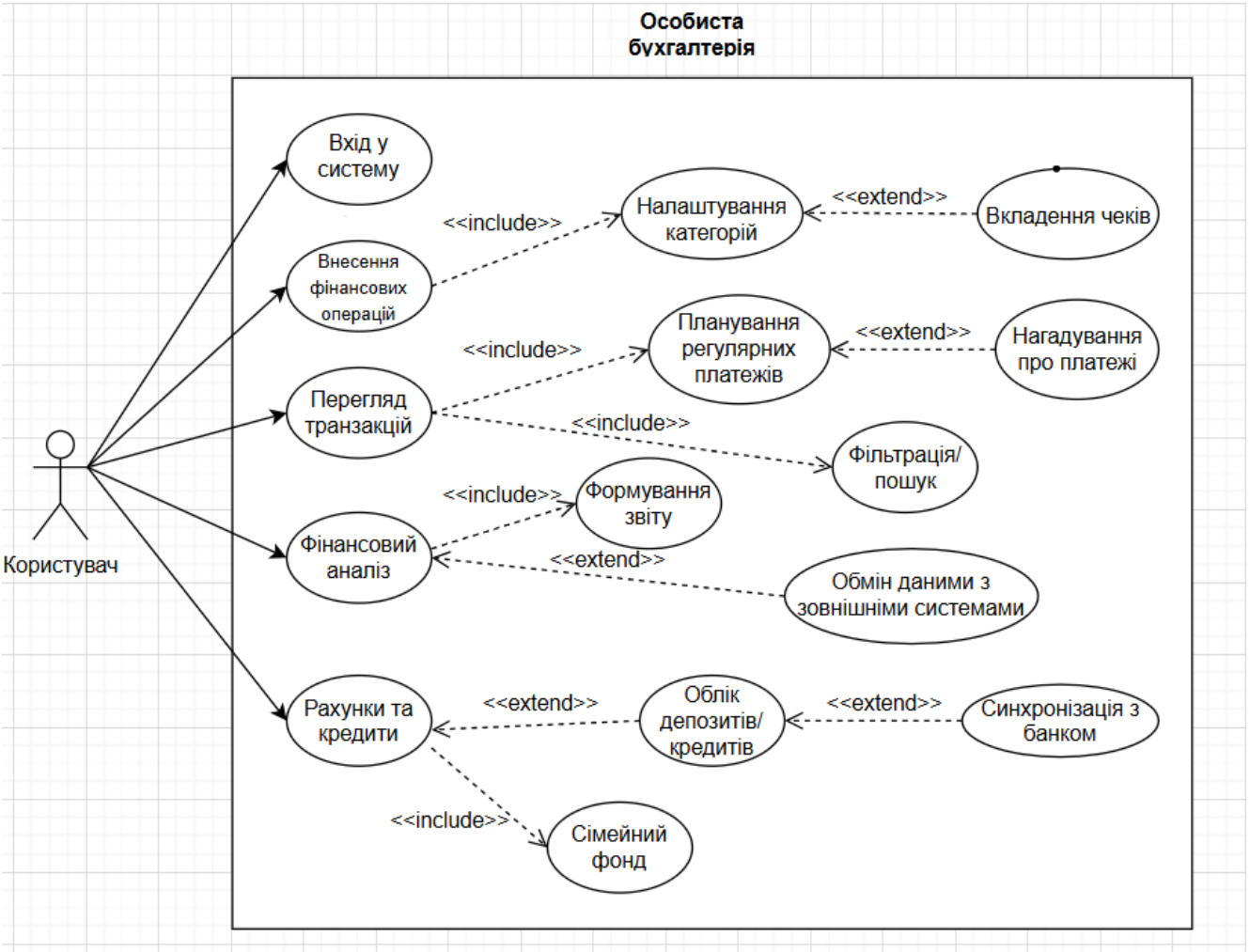

Рисунок 1 – Use Case для користувача

2. Проаналізувавши тематику системи «Особиста бухгалтерія» та розроблену діаграму варіантів використання (Use Case Diagram), було побудовано діаграму класів (рис.2), яка відображає основні сутності, зв’язки між ними та логіку роботи застосунку.
   На діаграмі класів система поділена на два логічні рівні: Domain (Сутності) та Application (Сервіси). У доменній частині відображені ключові об’єкти предметної області — транзакції, рахунки, категорії, регулярні платежі,

сімейний фонд та банківські рахунки, які описують структуру та взаємозв'язки фінансових даних користувача. У частині сервісів представлено класи, що реалізують бізнес-логіку системи: TransactionService, SchedulerService, IntegrationService та ReportGenerator. Вони відповідають за операції з обліку доходів і витрат, планування регулярних платежів, імпорт та експорт даних, а також формування аналітичних звітів. Така структура забезпечує модульність системи, спрощує підтримку та розширення функціоналу.

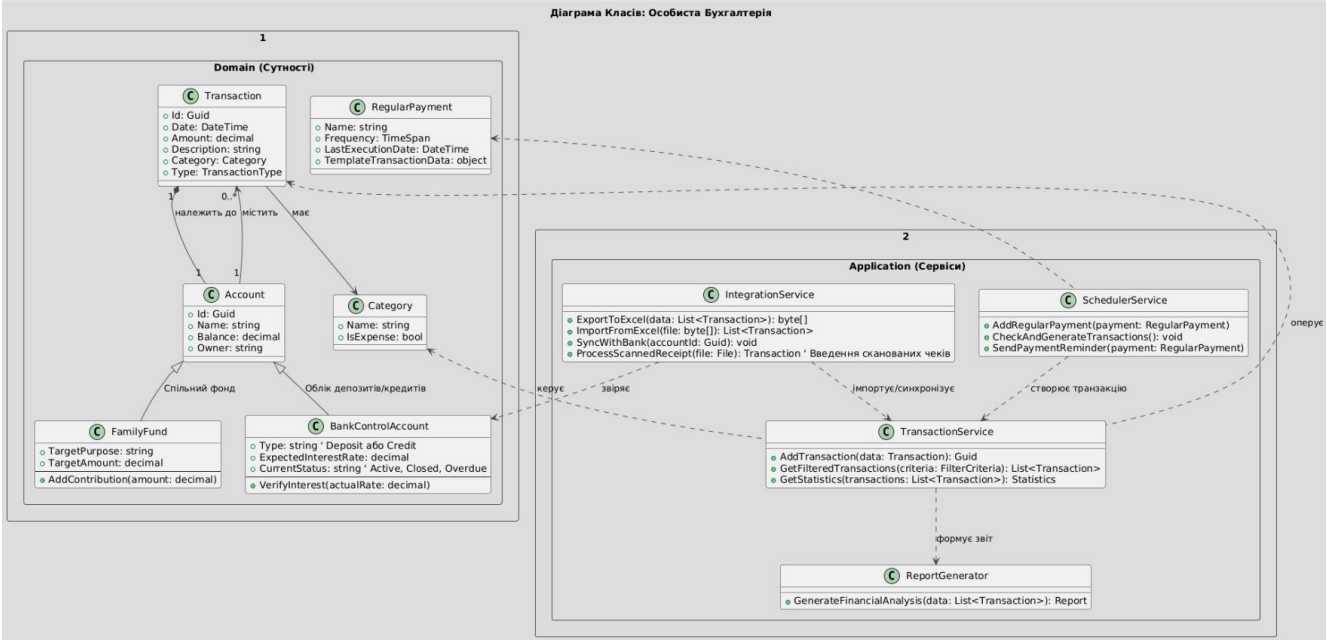

Рисунок 2 – Діаграма класів

3. На основі діаграми варіантів використання (Use Case Diagram) було сформовано три основні сценарії взаємодії користувача із системою:
    - Додавання нової транзакції.
    - Формування фінансового звіту.
    - Налаштування регулярного платежу.

   Кожен сценарій відображає конкретну послідовність дій користувача та системи, спрямовану на досягнення певної мети: ведення обліку фінансових операцій, аналіз доходів/витрат і автоматизацію повторюваних платежів.

Сценарій 1: Додавання нової транзакції (дохід / витрата)

Передумови:

- Користувач увійшов у систему.
- Створено хоча б один рахунок і категорію.
- Система працює у штатному режимі.

Постумови:

- У базі даних з'являється новий запис транзакції.

- Баланс рахунку оновлюється відповідно до типу транзакції (дохід/витрата).
- Система зберігає оновлену аналітику.

Взаємодіючі сторони:
Користувач, Система.

Короткий опис:
Користувач додає нову фінансову операцію, вказуючи тип, суму, категорію, опис і дату. Система перевіряє дані, зберігає їх у базі та оновлює баланс рахунку.

Основний потік подій:

1. Користувач відкриває розділ «Транзакції».
2. Обирає опцію «Додати транзакцію».
3. Система запитує: тип операції (дохід/витрата), рахунок, категорію, суму, дату, опис.
4. Користувач заповнює поля та натискає «Зберегти».
5. Система перевіряє коректність введених даних (позитивна сума, вибрані рахунок і категорія).
6. Якщо перевірка успішна, система створює об’єкт Transaction і додає його до вибраного Account.
7. Баланс рахунку оновлюється, а транзакція з’являється у списку операцій.
8. Система показує повідомлення «Транзакцію успішно додано».

Винятки:

- Некоректна сума (нуль або від’ємне значення) → повідомлення «Сума некоректна».
- Не вибрано рахунок або категорію → система просить уточнити.
- Відсутній зв’язок із базою → повідомлення «Помилка збереження даних».

Примітки:
Може бути викликано з основного меню або через коротке посилання «+ Додати транзакцію» на головному екрані.

Сценарій 2: Планування регулярного платежу (зарплата, оренда, кредит)

Передумови:

- Користувач авторизований.
- Існує хоча б один рахунок і категорія.
- Система синхронізована за часом.

Постумови:

- Створено запис у таблиці RegularPayment.
- Збережено шаблон транзакції для автоматичного створення.
- Налаштовано нагадування або автозапуск транзакцій.

Взаємодіючі сторони:
Користувач, Система.

Короткий опис:
Користувач створює регулярний платіж (наприклад, оренда чи заробітна плата).

Система зберігає шаблон, додає його до розкладу й автоматично створює транзакції у зазначені дати.

Основний потік подій:

1. Користувач відкриває розділ «Регулярні платежі».
2. Обирає опцію «Додати новий платіж».
3. Система пропонує ввести назву, тип (дохід/витрата), суму, періодичність, дату початку, рахунок і категорію.
4. Користувач заповнює поля й натискає «Підтвердити».
5. Система перевіряє дані (усі поля заповнені, коректна дата, позитивна сума).
6. Система створює об’єкт RegularPayment і додає його до бази даних.
7. Запускається сервіс SchedulerService, який планує майбутні транзакції.
8. Система відображає повідомлення «Регулярний платіж створено».

Винятки:

- Некоректна дата → повідомлення «Дата повинна бути в майбутньому».
- Відсутня періодичність → система просить уточнити інтервал.
- Помилка при збереженні → система пропонує повторити.

Примітки:

При кожному запуску застосунку система перевіряє, чи настав час для автоматичного виконання регулярних платежів.

Сценарій 3: Формування фінансового звіту

Передумови:

- Користувач увійшов у систему.
- Є транзакції у вибраному періоді.

Постумови:

- Сформовано фінансовий звіт у вигляді таблиці або діаграми.
- Звіт можна зберегти, експортувати або надрукувати.

Взаємодіючі сторони:

Користувач, Система.

Короткий опис:

Користувач формує фінансовий звіт за вибраний період. Система виконує вибірку з бази даних, обчислює підсумки та будує графічне відображення доходів і витрат.

Основний потік подій:

1. Користувач переходить до розділу «Аналітика і звіти».
2. Обирає тип звіту (доходи, витрати, баланс).
3. Вказує період (початкова й кінцева дати).
4. Система формує запит до бази транзакцій.
5. TransactionService отримує дані та передає їх у ReportGenerator.
6. ReportGenerator створює аналітичний звіт і повертає його системі.
7. Система відображає звіт користувачу у вигляді таблиці чи діаграми.

8. Користувач за потреби натискає «Експортувати в Excel».
9. IntegrationService генерує файл і пропонує зберегти його.

Винятки:

- У періоді відсутні транзакції → повідомлення «Дані для звіту відсутні».
- Помилка експорту → система повідомляє «Не вдалося зберегти файл».

Примітки:

Система може зберігати останні параметри звіту для швидкого повторного формування.

4. На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.

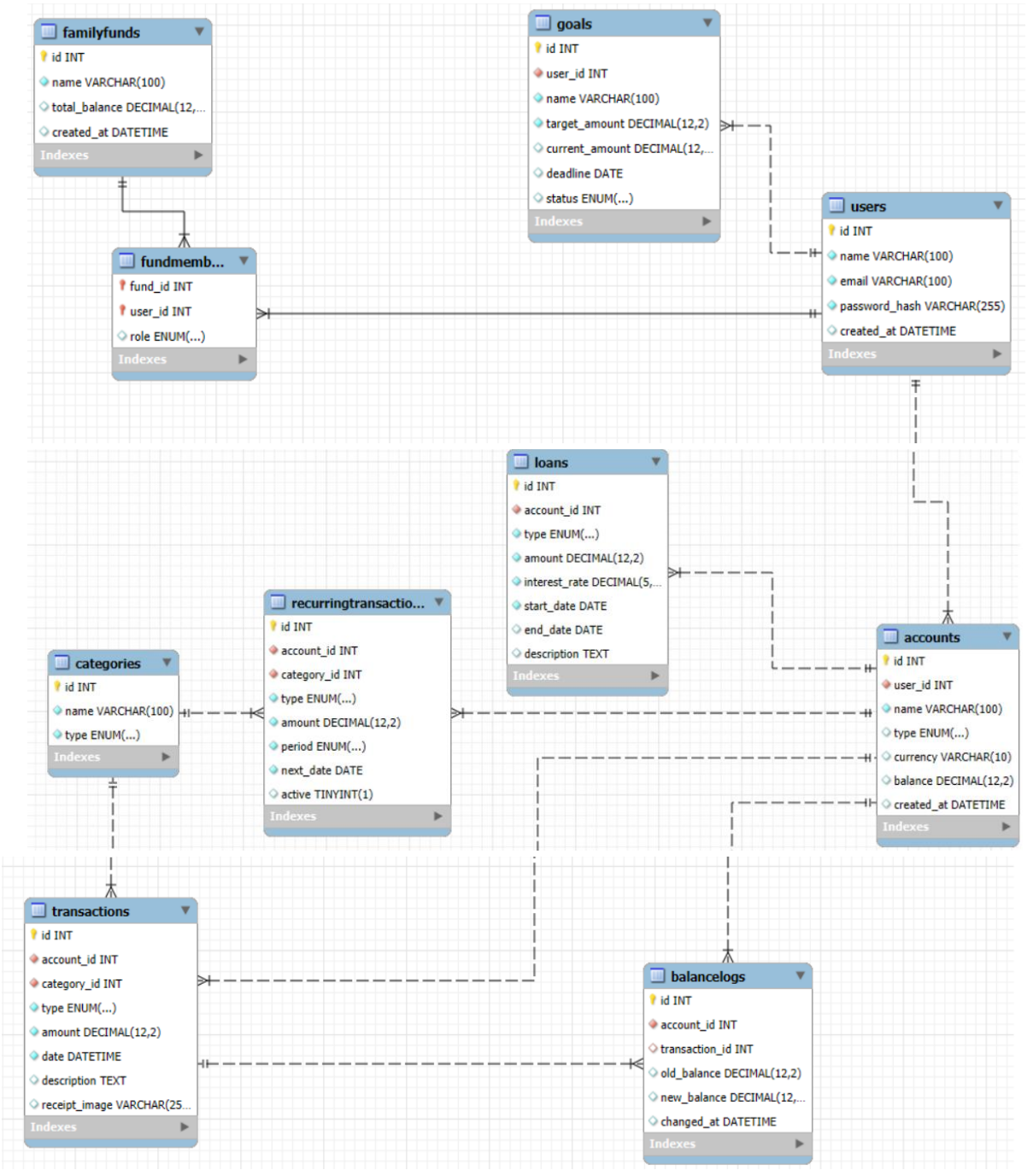

Рисунок 3 – Структура бази даних системи

6. Ми побудували діаграму класів для реалізованої частини системи «Особиста бухгалтерія», яка відображає основну структуру програмного забезпечення та взаємозв’язки між його компонентами.
   На діаграмі показано три основні рівні системи:
   - рівень доменних сутностей (Domain), що описує основні об’єкти предметної області, такі як користувачі, рахунки, транзакції, категорії, фінансові цілі, кредити, періодичні платежі та сімейний фонд;
   - рівень доступу до даних (Repository), який реалізує шаблон Repository для взаємодії з базою даних, забезпечуючи виконання CRUD-операцій (створення, читання, оновлення, видалення);
   - рівень прикладної логіки (Application Services), де реалізовано бізнес-операції системи — облік фінансів, побудову звітів, обробку регулярних транзакцій, управління цілями, а також імпорт і експорт даних.Таким чином, діаграма класів демонструє логічну структуру системи, її модульність, зв’язки між сутностями та механізм взаємодії між рівнями, що забезпечує гнучкість і масштабованість розробленої програми.

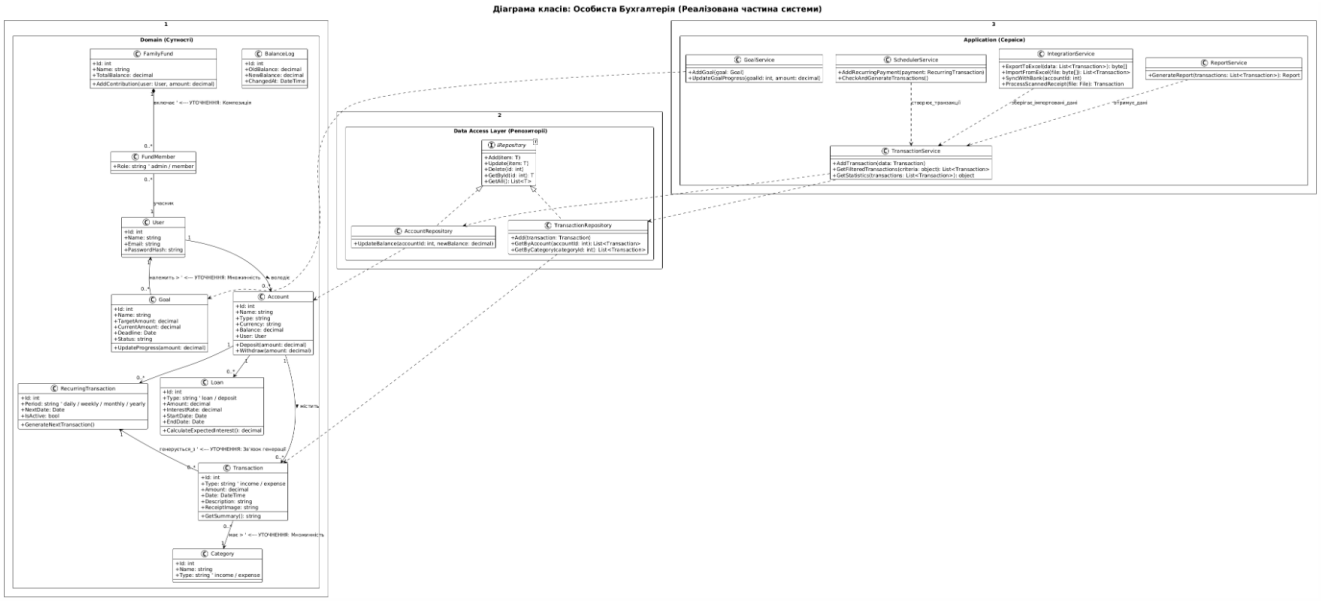

Рисунок 4 – Діаграма класів реалізованої частини системи

Відповіді на контрольні запитання:

1. Що таке UML?
UML — це уніфікована мова моделювання, яка використовується для опису, проектування та документування програмних систем.

2. Що таке діаграма класів UML?
Діаграма класів — це статичне представлення структури системи, яке показує класи, їх атрибути, методи та зв’язки між ними.

3. Які діаграми UML називають канонічними?
До канонічних діаграм UML належать діаграми варіантів використання, класів, об’єктів, послідовностей, станів, діяльності, компонентів і розгортання.

4. Що таке діаграма варіантів використання?
Це діаграма, яка відображає взаємодію користувачів (акторів) із системою через її функціональні можливості — варіанти використання.

5. Що таке варіант використання?
Варіант використання — це окрема функція або сценарій поведінки системи, який приносить користь користувачу.

6. Які відношення можуть бути відображені на діаграмі використання?
На діаграмі використання можуть бути відображені відношення **асоціації**, **узагальнення (generalization)**, **включення (include)** та **розширення (extend)**.

7. Що таке сценарій?
Сценарій — це послідовність дій, що описує конкретний спосіб виконання варіанту використання.

8. Що таке діаграма класів?
Це модель, яка показує основні класи системи, їхні атрибути, методи та типи зв’язків між ними.

9. Які зв’язки між класами ви знаєте?
Основні типи зв’язків: **асоціація**, **агрегація**, **композиція**, **успадкування (generalization)** та **залежність (dependency)**.

10. Чим відрізняється композиція від агрегації?
Композиція означає, що об’єкт не може існувати без власника, а агрегація — що об’єкт може існувати самостійно.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів?
На діаграмі агрегація позначається **порожнім ромбом**, а композиція — **заповненим ромбом** біля класу-власника.

12. Що являють собою нормальні форми баз даних?
Нормальні форми — це правила, за якими структурують таблиці бази даних, щоб уникнути надлишкових даних і забезпечити цілісність.

13. Що таке фізична модель бази даних? Логічна?
Логічна модель описує структуру даних і зв’язки між сутностями, а фізична — конкретну реалізацію цих структур у СУБД (таблиці, типи полів, індекси тощо).

14. Який взаємозв'язок між таблицями БД та програмними класами?
Кожна таблиця бази даних зазвичай відповідає окремому класу програми, а рядки таблиці — це об’єкти цього класу.

Висновки

У ході виконання лабораторної роботи було опрацьовано теоретичні відомості щодо UML-моделювання, структури баз даних та шаблону Repository. Проаналізовано тему «Особиста бухгалтерія» та розроблено діаграму варіантів використання, яка відображає основні функції системи та взаємодію користувача з програмою. Було спроєктовано діаграму класів предметної області, що містить основні сутності, їх атрибути та зв’язки. На основі цієї діаграми створено структуру бази даних із реалізацією логічних зв’язків між таблицями. Класи даних реалізують шаблон Repository, який забезпечує зручну взаємодію між програмною логікою та базою даних. Також було побудовано діаграму класів для реалізованої частини системи, що демонструє структуру коду, реалізацію сервісів, репозиторіїв та основних компонентів програми.