

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 IA-31

Кизим Є. К.

Київ 2025

## **Зміст**

|  |           |
|--|-----------|
| <b>Теоретичні відомості.....</b>                                     | <b>4</b>  |
| <b>Хід роботи.....</b>   | <b>5</b>  |
| <b>Діаграма варіантів використання (Use-Cases Diagram).....</b>      | <b>5</b>  |
| <b>Сценарії використання .....</b>                                   | <b>6</b>  |
| <b>1. Варіант використання: Додавання витрати вручну .....</b>       | <b>6</b>  |
| <b>2. Варіант використання: Створення періодичного платежу .....</b> | <b>7</b>  |
| <b>3. Варіант використання: Перегляд статистики витрат.....</b>      | <b>8</b>  |
| <b>Діаграма класів.....</b>  | <b>9</b>  |
| <b>Опис класів .....</b>   | <b>10</b> |
| <b>Структура бази даних .....</b>                                    | <b>11</b> |
| <b>Структура класів .....</b>  | <b>14</b> |
| <b>Контрольні питання.....</b>                                       | <b>14</b> |
| <b>Висновки .....</b>  | <b>19</b> |
| <b>Репозиторій .....</b>   | <b>19</b> |

**Мета:** Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

**Завдання:**

1. Ознайомитися з короткими теоретичними відомостями.
2. Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
3. Спроєктувати діаграму класів предметної області.
4. Вибрати 3 варіанти використання та написати за ними сценарії використання.
5. 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.
6. Нарисувати діаграму класів для реалізованої частини системи.
7. Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

**Моя тема:**

**27. Особиста бухгалтерія** (state, prototype, decorator, bridge, flyweight, SOA)

Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

## Теоретичні відомості

UML (Unified Modeling Language) — це універсальна мова графічного моделювання, яку використовують для опису, проєктування та документування програмних систем і бізнес-процесів. Вона дозволяє створювати модель системи на різних рівнях деталізації: від загального бачення її призначення до логічної структури та фізичного втілення. В об'єктно-орієнтованому аналізі та проєктуванні система подається як сукупність різних представлень, кожне з яких відображає певну частину її будови або поведінки.

Одним із ключових інструментів UML є діаграми, що подають модель у вигляді формалізованих графічних елементів. Серед найпоширеніших типів — діаграми варіантів використання, класів, послідовностей, станів, діяльності, компонентів і розгортання. Кожен вид зосереджується на певному аспекті системи: її статичній структурі, логіці взаємодій, життєвому циклі об'єктів чи фізичній інфраструктурі. Діаграма варіантів використання (Use Case Diagram) показує систему з позиції користувача. На ній зображають акторів — зовнішніх учасників — та варіанти використання, тобто функції або сервіси, які система їм надає. Взаємозв'язки між акторами й варіантами описуються відношеннями: асоціація вказує на факт взаємодії, узагальнення демонструє наслідування властивостей, а відношення «extend» дає змогу розширювати поведінку базового сценарію за певних умов. Для кожного варіанта використання зазвичай формують текстовий опис: передумови, основний сценарій, альтернативні потоки та очікуваний результат. Разом діаграми та сценарії утворюють чітке, структуроване представлення функціональних вимог до системи, що забезпечує однакове розуміння між усіма учасниками розробки.

## Хід роботи

### Діаграма варіантів використання (Use-Cases Diagram)

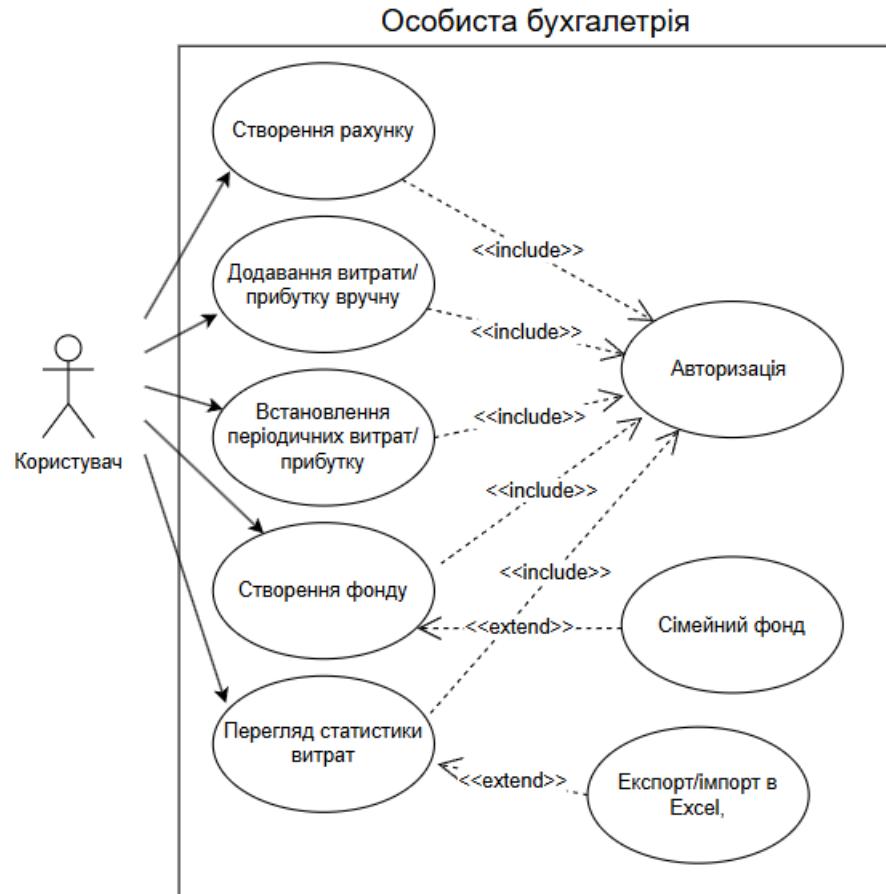


Рисунок 1 – Діаграма варіантів використання

Прямоокруглою рамкою позначено систему «**Особиста бухгалтерія**».

Основні сценарії використання системи такі:

- Створення рахунку
- Додавання витрат/прибутку вручну
- Встановлення періодичних витрат/прибутку
- Створення фонду
- Перегляд статистики витрат

Усі основні сценарії потребують Авторизації, що забезпечує безпеку доступу до особистих фінансових даних користувача. Сценарій «Створення фонду» може

бути розширений сценарієм «Сімейний фонд», що означає можливість створення спільного фонду для кількох користувачів.

Також сценарій «Перегляд статистики витрат» розширюється варіантом Експорт/імпорт в Excel, що дозволяє користувачу зберігати або імпортувати фінансові дані у форматі Excel для подальшого аналізу

## **Сценарії використання**

### **1. Варіант використання: Додавання витрати вручну**

- Передумови – Користувач авторизований у системі.
- Постумови – У разі успішного виконання, створюється нова транзакція витрати, а баланс відповідного рахунку зменшується на суму витрати. У протилежному випадку стан системи не змінюється.
- Взаємодіючі сторони – Користувач, Система.
- Короткий опис – Цей варіант використання описує базовий процес ручного введення користувачем даних про здійснену витрату.
- Основний перебіг подій:
  1. Користувач обирає в меню опцію "Додати витрату".
  2. Система відображає форму для введення даних.
  3. Користувач заповнює обов'язкові поля: suma, data, категорія витрати, опис витрати та рахунок з якого списано кошти.
  4. Користувач натискає кнопку "Зберегти".
  5. Система перевіряє коректність введених даних, створює запис про витрату та оновлює баланс рахунку.
- Винятки:
  - Виняток №1: Некоректна suma. Якщо в полі "Сума" користувач вводить нечислове або від'ємне значення, система виводить повідомлення про помилку та просить виправити дані, не закриваючи форму.

- Примітки – Поле "Дата" може заповнюватися автоматично (поточна дата).  
Поле "Опис" є необов'язковим для заповнення.

## **2. Варіант використання: Створення періодичного платежу**

- Передумови – Користувач авторизований у системі. У системі існує хоча б один рахунок.
- Постумови – У системі створено нове правило для автоматичного списання коштів за заданим розкладом. У разі невдачі нове правило не створюється.
- Взаємодіючі сторони – Користувач, Система.
- Короткий опис – Цей варіант використання дозволяє користувачу налаштувати автоматичне створення транзакції витрати або прибутку (наприклад, орендна плата, зарплата, плата за інтернет), яка буде повторюватися через певні проміжки часу.
- Основний перебіг подій:
  1. Користувач переходить у розділ "Періодичні платежі" та обирає опцію "Створити новий".
  2. Система відображає форму для налаштування періодичного платежу.
  3. Користувач вводить назву та опис платежу (наприклад, "Оплата за квартиру"), суму та обирає категорію.
  4. Користувач обирає рахунок, до якого буде застосовуватися платіж.
  5. Користувач налаштовує розклад: вказує дату першого платежу, періодичність (наприклад, щомісяця, щотижня) та, за бажанням, дату закінчення.
  6. Користувач підтверджує створення періодичного платежу.
  7. Система перевіряє коректність введених даних, зберігає правило та інформує користувача про успішне створення.
- Винятки:
  - Виняток №1: Некоректні дані. Якщо на кроці 7 система виявляє помилку (наприклад, дата початку пізніша за дату закінчення або сума не є числовим значенням), вона виводить повідомлення про

помилку з проханням виправити відповідні поля, не закриваючи форму.

- Примітки – Поле "Дата" може заповнюватися автоматично (поточна дата). Поле "Опис" є необов'язковим для заповнення.

### **3. Варіант використання: Перегляд статистики витрат**

- Передумови – Користувач авторизований у системі. У системі існують записи про витрати за поточний місяць.
- Постумови – Система відображає зведену інформацію про витрати. Стандарти у системі не змінюються.
- Взаємодіючі сторони – Користувач, Система.
- Короткий опис – Користувач переглядає звіт про свої витрати за поточний місяць, згрупованими за категоріями.
- Основний перебіг подій:
  1. Користувач переходить до розділу "Статистика" або "Звіти".
  2. Система за замовчуванням збирає всі дані про витрати за поточний календарний місяць.
  3. Система відображає на екрані звіт де кожен рядок відповідає певній категорії витрат ("Продукти", "Комунальні платежі" тощо) та показує її частку в загальній сумі.
- Винятки:
  - Виняток №1: Відсутні дані. Якщо за поточний місяць не було зроблено жодної витрати, система замість звіту виводить повідомлення: "За цей період витрат не знайдено".
- Примітки – Відсутні.

## Діаграма класів

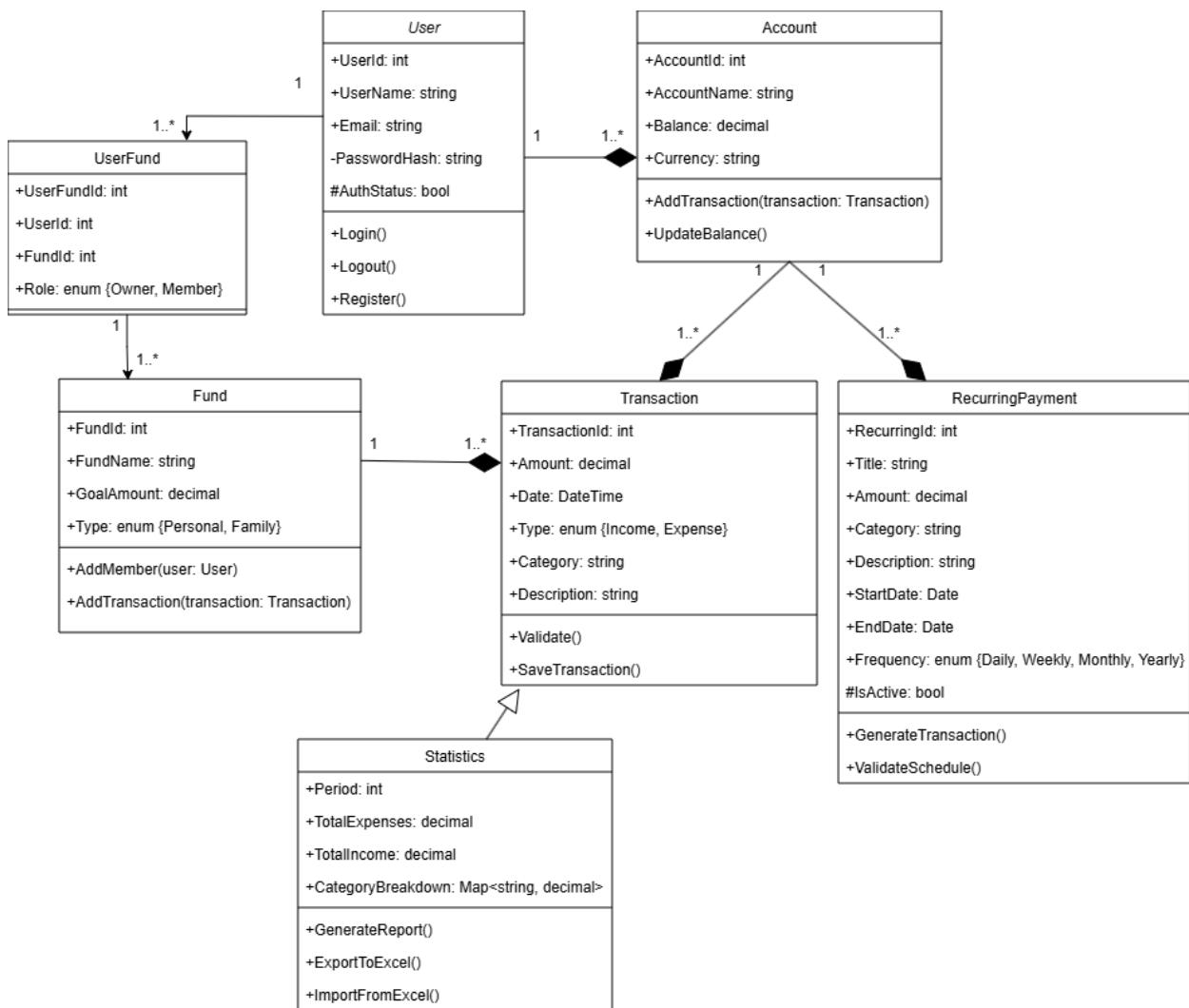


Рисунок 2 – Діаграма класів

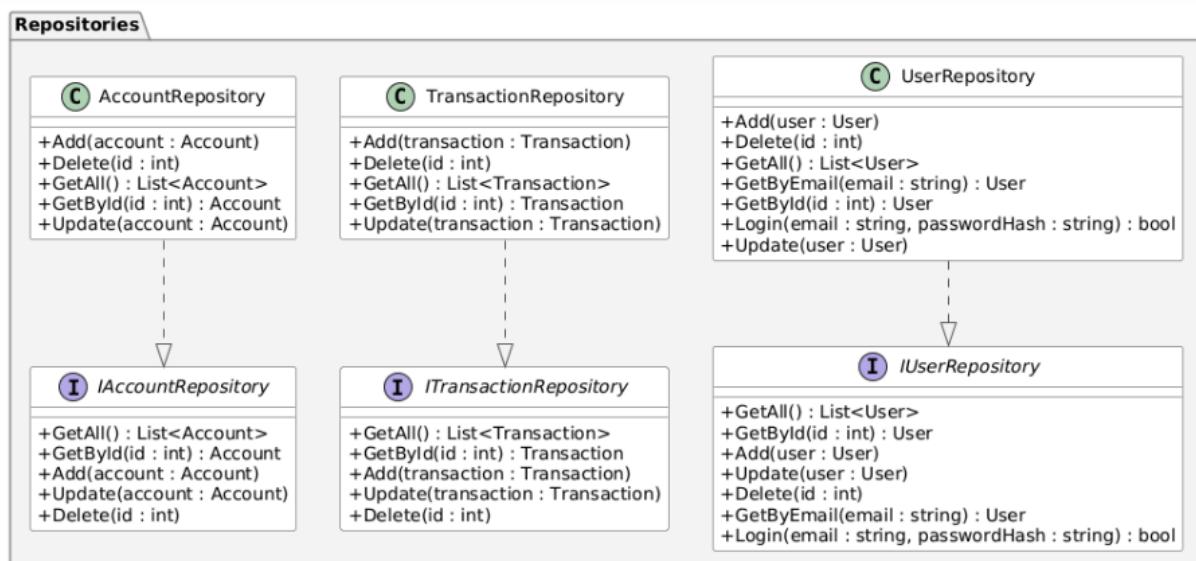


Рисунок 2.1 – Діаграма класів та інтерфейсів які виконують шаблон  
Repository

**Опис класів:**

- **Account.cs:** Клас, що представляє банківський рахунок користувача. Містить інформацію про баланс, валюту та назву рахунку, а також методи для оновлення балансу та додавання транзакцій.
- **Fund.cs:** Клас, що описує спільний фонд користувачів (наприклад, сімейний або персональний). Містить дані про цільову суму, тип фонду та методи для додавання учасників і транзакцій.
- **RecurringPayment.cs:** Клас для збереження даних про регулярні платежі. Містить інформацію про періодичність, суму, категорію та опис платежу, а також методи для генерації транзакцій і перевірки розкладу.
- **Statistics.cs:** Клас, що відповідає за аналітику та статистику фінансів користувача. Зберігає дані про доходи, витрати, розподіл по категоріях та має методи для створення звітів і експорту/імпорту даних в Excel.
- **Transaction.cs:** Клас, що описує окрему фінансову операцію (дохід або витрату). Містить суму, дату, категорію, опис і тип транзакції, а також методи для перевірки та збереження операції.
- **User.cs:** Клас, що представляє користувача системи. Містить особисті дані (ім'я, email, пароль), статус автентифікації та методи для входу, виходу та реєстрації.
- **UserFund.cs:** Проміжний клас, що описує зв'язок між користувачем і фондом. Використовується для реалізації ролей користувачів (власник або учасник фонду).

- **IAccountRepository.cs**: Інтерфейс, що визначає стандартні CRUD-операції для роботи з рахунками (отримання, додавання, оновлення, видалення).
- **AccountRepository.cs**: Клас, що реалізує інтерфейс IAccountRepository. Здійснює безпосередню взаємодію з базою даних для управління об'єктами Account.
- **ITransactionRepository.cs**: Інтерфейс для стандартних CRUD-операцій із транзакціями.
- **TransactionRepository.cs**: Реалізація інтерфейсу ITransactionRepository. Відповідає за збереження та обробку транзакцій у базі даних.
- **IUserRepository.cs**: Інтерфейс, що описує методи для роботи з користувачами, включаючи пошук за email та перевірку входу.
- **UserRepository.cs**: Клас, що реалізує IUserRepository, забезпечуючи повний набір операцій для керування користувачами системи.

### Структура бази даних

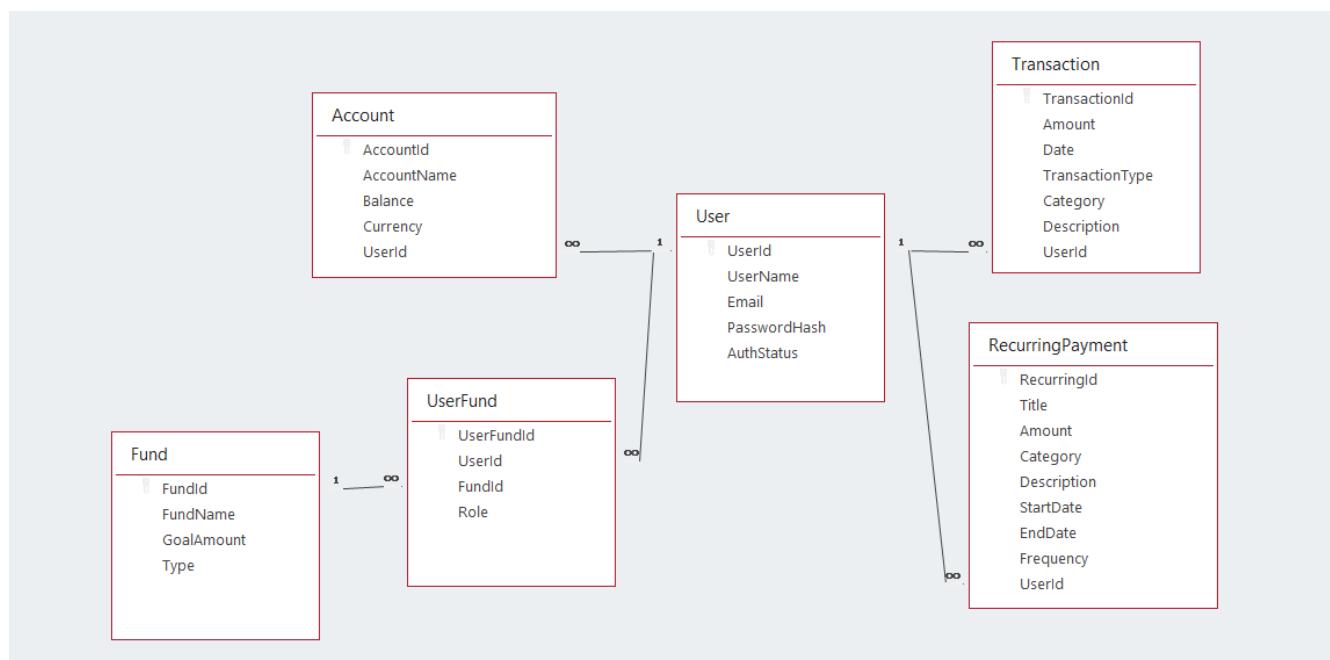


Рисунок 3 – Структура бази даних

Таблиця User Зберігає інформацію про окремих користувачів системи.

- UserId (PK): Унікальний ідентифікатор користувача.
- UserName: Ім'я користувача.
- Email: Електронна пошта користувача.
- PasswordHash: Хеш пароля користувача.
- AuthStatus: Статус автентифікації (наприклад, активний, неактивний).

Таблиця Account Зберігає інформацію про фінансові рахунки користувачів

- AccountId (PK): Унікальний ідентифікатор рахунку.
- AccountName: Назва рахунку.
- Balance: Поточний баланс рахунку.
- Currency: Валюта рахунку.
- UserId (FK): Посилання на користувача, якому належить рахунок (зв'язок "один-до-багатьох": один користувач може мати багато рахунків).

Таблиця Fund Зберігає інформацію про фінансові цілі або фонди (наприклад, накопичення на відпустку).

- FundId (PK): Унікальний ідентифікатор фонду.
- FundName: Назва фонду/цілі.
- GoalAmount: Цільова сума накопичення.
- Type: Тип фонду.

Таблиця UserFund Забезпечує зв'язок "багато-до-багатьох" між User та Fund, дозволяючи кільком користувачам брати участь у спільному фонді.

- UserFundId (PK): Унікальний ідентифікатор зв'язку.
- UserId (FK): Зовнішній ключ, що посилається на User.
- FundId (FK): Зовнішній ключ, що посилається на Fund.
- Role: Роль користувача у цьому фонді (наприклад, "власник", "учасник").

Таблиця Transaction Зберігає інформацію про окремі фінансові транзакції (доходи/витрати).

- TransactionId (PK): Унікальний ідентифікатор транзакції.
- Amount: Сума транзакції.
- Date: Дата транзакції.
- TransactionType: Тип транзакції (наприклад, "дохід", "витрата").
- Category: Категорія транзакції.
- Description: Опис транзакції.
- UserId (FK): Посилання на користувача, який здійснив транзакцію.

Таблиця RecurringPayment Зберігає інформацію про регулярні платежі.

- RecurringId (PK): Унікальний ідентифікатор регулярного платежу.
- Title: Назва регулярного платежу.
- Amount: Сума платежу.
- Category: Категорія.
- Description: Опис.
- StartDate: Дата початку.
- EndDate: Дата завершення.
- Frequency: Частота платежу (наприклад, "місячно", "щотижня").
- UserId (FK): Посилання на користувача, якому належить платіж.

Ця таблиця є ключовою для відстеження майбутніх зобов'язань.

## Структура класів

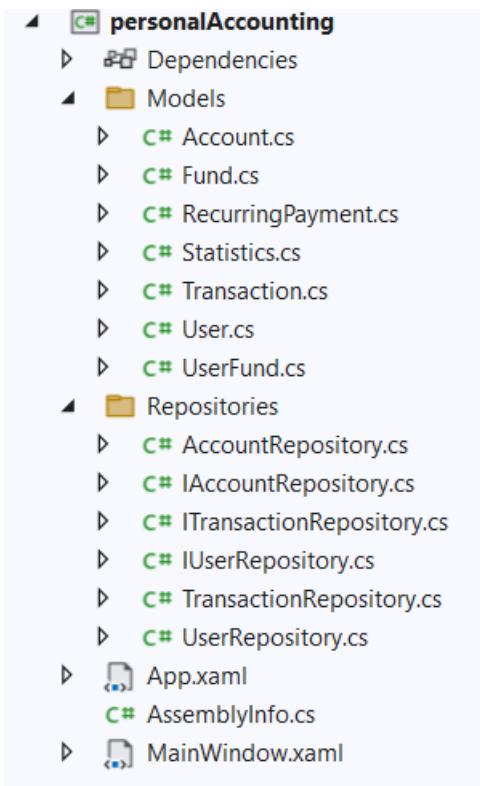


Рисунок 4 – Структура класів

Структура проекту чітко відображає архітектурні принципи, зокрема розділення відповідальності, шляхом логічного групування класів у спеціалізовані каталоги. Ключові папки *Models* та *Repositories* вказують на застосування шаблону, де сутності предметної області (такі як *User*, *Account*, *Transaction*, тощо) відокремлені від логіки доступу до даних. Каталог *Models* містить ці основні об'єкти. Водночас, папка *Repositories* реалізує патерн "Repository", інкапсулюючи взаємодію з базою даних через інтерфейси (*IUserRepository*, *IAccountRepository*, *ITransactionRepository*) та їхні конкретні реалізації.

## Контрольні питання

### 1. Що таке UML?

Мова UML є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем.

## **2. Що таке діаграма класів UML?**

Діаграми класів використовуються при моделюванні програмних систем. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру. Діаграма класів не відображає динамічної поведінки об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

## **3. Які діаграми UML називають канонічними?**

Нотація канонічних діаграм є основним засобом розробки моделей мовою UML. У нотації мови UML визначено такі види діаграм: варіантів використання (use case diagram); класів (class diagram); кооперації (collaboration diagram); послідовності (sequence diagram); станів (statechart diagram); діяльності (activity diagram); компонентів (component diagram) та розгортання (deployment diagram). Перелічені діаграми є невід'ємною частиною графічної нотації мови UML.

## **4. Що таке діаграма варіантів використання?**

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється. Діаграма варіантів використання – це вихідна концептуальна модель проєкованої системи, вона не описує внутрішню побудову системи.

## **5. Що таке варіант використання?**

Варіант використання служить для опису служб, які система надає актору. Інакше кажучи кожен варіант використання визначає набір дій, здійснюваний системою під час діалогу з актором. Кожен варіант використання являє собою послідовність дій, який повинен бути виконаний системою, що проєктується при взаємодії її з відповідним актором, самі ці дії не відображаються на діаграмі.

## **6. Які відношення можуть бути відображені на діаграмі використання?**

Існують такі відносини: асоціації, узагальнення, залежність (складається з включення та розширення):

**Асоціація (association)** – узагальнене, невідоме ставлення між актором та варіантом використання.

**Відношення узагальнення (generalization)** – показує, що нащадок успадковує атрибути у свого прямого батьківського елементу. Тобто, один елемент моделі є спеціальним або окремим випадком іншого елемента моделі. Може застосовуватися як до акторів, так і до варіантів використання.

**Відношення залежності (dependency)** визначається як форма взаємозв'язку між двома елементами моделі, призначена для специфікації тієї обставини, що зміна одного елемента моделі призводить до зміни деякого іншого елемента.

**Відношення включення (include)** – окремий випадок загального відношення залежності між двома варіантами використання, при якому деякий варіант використання містить поведінку, визначену в іншому варіанті використання.

**Відношення розширення (extend)** – показує, що варіант використання розширює базову послідовність дій та вставляє власну послідовність. У цьому на 25 відміну типу відносин «включення» розширена послідовність може здійснюватися залежно від певних умов.

## **7. Що таке сценарій?**

Сценарії використання – це текстові уявлення тих процесів, які відбуваються при взаємодії користувачів системи та самої системи. Вони є чітко формалізованими, покроковими інструкціями, що описують той чи інший процес у термінах кроків досягнення мети. Сценарії використання однозначно визначають кінцевий результат.

## **8. Що таке діаграма класів?**

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проектування, показуючи її структуру. Діаграма класів не відображає динамічної поведінки об'єктів зображеніх на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

## 9. Які зв'язки між класами ви знаєте?

Існують такі зв'язки між класами: асоціація, узагальнення (успадкування), агрегація та композиція:

**Асоціація** – найбільш загальний вид зв'язку між двома класами системи. Як правило, вона відображає використання одного класу іншим за допомогою певної якості або поля.

**Узагальнення (успадкування)** на діаграмах класів використовується, щоб показати зв'язок між класом-батьком та класом-нащадком. Воно вводиться на діаграму, коли виникає різновид будь-якого класу, і навіть у випадках, як у системі виявляються кілька класів, які мають подібну поведінку.

**Агрегацію** позначається відношення частина-ціле, коли об'єкти одного класу входять до об'єкта іншого класу. Типовим прикладом таких відносин є списки об'єктів. У цьому випадку список буде виступати агрегатом, а об'єкти, що входять до списку, елементами, що агрегуються.

**Композицію** також позначається відношення частина-ціле, але позначає тісніший зв'язок між елементами, що представляють ціле та частини.

## 10. Чим відрізняється композиція від агрегації?

Композиція і агрегація обидві позначають відношення частина-ціле, але відрізняються силою зв'язку:

В агрегації об'єкти-частини можуть існувати, якщо об'єкт-ціле видаляється, а в композиції об'єкти-частини не можуть існувати за межами об'єкта-цілого.

## **11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів?**

На діаграмах класів вони відрізняються графічним позначенням ромба:

**Агрегація:** Позначається незафарбованим (порожнім) ромбом біля класу, що представляє ціле.

**Композиція:** Позначається зафарбованим (чорним) ромбом біля класу, що представляє ціле.

## **12. Що являють собою нормальні форми баз даних?**

**Нормальна форма** – властивість відношення в реляційній моделі даних, що характеризує його з погляду надмірності, що потенційно призводить до логічно помилкових результатів вибірки або зміни даних. Нормальна форма окреслюється сукупністю вимог, яким має задовольняти ставлення.

Нормалізація призначена для приведення структури БД до виду, що забезпечує мінімальну логічну надмірність, і не має на меті зменшення або збільшення продуктивності роботи або зменшення або збільшення фізичного обсягу бази даних. Кінцевою метою нормалізації є зменшення потенційної суперечливості інформації, що зберігається в базі даних.

## **13. Що таке фізична модель бази даних? Логічна?**

**Фізична модель бази даних** представляє собор набір бінарних даних у вигляді файлів, структурованих та згрупованих згідно з призначенням (сегменти, екстенти та ін.), що використовується для швидкого та ефективного отримання інформації з жорсткого диска, а також для компактного зберігання та розміщення даних на жорсткому диску.

**Логічна модель бази даних** є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що дозволяють власне програмування та використання бази даних.

## **14. Який взаємозв'язок між таблицями БД та програмними класами?**

Взаємозв'язок між таблицями БД та програмними класами реалізується у процесі проєктування. Програмні класи є сутностями (елементами предметної області), а таблиці відображають їх технічну реалізацію та спосіб зберігання і зв'язку.

Розрізняють кілька підходів до зв'язування:

- Одна таблиця – один клас.
- Одна таблиця – кілька класів.
- Один клас – кілька таблиць.

Цей зв'язок зазвичай реалізується за допомогою шаблону Repository, який інкапсулює логіку взаємодії з БД, та шаблону ORM (Object-Relational Mapping), де класи слугують моделями (сховищами даних), що відображають структуру відповідних таблиць.

**Висновки:** Під час виконання лабораторної роботи я успішно обрала та освоїла зручну систему побудови UML-діаграм, що дозволило ефективно провести візуальне моделювання системи. Були розроблені Діаграми Варіантів Використання, які чітко окреслили функціональні межі системи, а також сценарії для опису поведінки користувачів. Також було створено діаграми класів предметної області, що дозволило структурувати ключові сутності, їхні атрибути та зв'язки, що дозволило створити основні класи в додатку.

**Репозиторій:** [посилання](#)