Національний технічний університет України «КПІ»
Факультет інформатики та обчислювальної техніки
Кафедра Інформаційних систем та технологій

Лабораторна робота №3

з дисципліни « Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

на тему: « Проектування REST веб-API»

Виконала:
студентка гр. ІС-11
Гаврильчик Яна
Викладач:
Бардін В.

2023 рік

Завдання:

Теоретична частина:

1. Ознайомитися з основами створення REST веб-API та методологією C4 для відображення архітектури системи.
2. Ознайомитися з основами створення ER-діаграм для представлення структури бази даних.

Практична частина:

1. З дотриманням вимог REST-у спроектувати веб-API для обраної(згідно варіанту) доменної області, використовуючи методологію C4 для створення діаграми архітектури системи.
2. Створити ER-діаграму для DAL (Data Access Layer), яка відображатиме структуру бази даних веб-API.
3. Оформити спроєктоване рішення у вигляді звіту до лабораторної роботи.

Варіант:

4	Гаманець. Керування власним бюджетом та фінансами	1. Власний бюджет складається з декількох рахунків, які поповнюються за заданими статтями прибутку. 2. Гроші цих рахунків можуть бути переведені з одного на інший, можуть витрачатись за заданими статтями витрат. 3. Підсумовуючи витрати та прибутки, можливо отримати інформацію, скільки було витрачено/отримано загалом/за певною статтею по заданому рахунку. **Функціональні вимоги:** 1. Ведення власного бюджету; 2. Отримання звітної інформації по рахунках.

Хід виконання роботи:

1. C4 модель – набір ієрархічних абстракцій (програмні системи, контейнери, компоненти та код).

Тобто у результаті повинно бути чотири види діаграм, хоча доволі популярною практикою є обмеження у використанні тільки діаграм контексту та контейнеру, які дають достатньої цінності архітектурі.

Почнемо з діаграми контексту, яка допомагає побачити загальну картину архітектури:

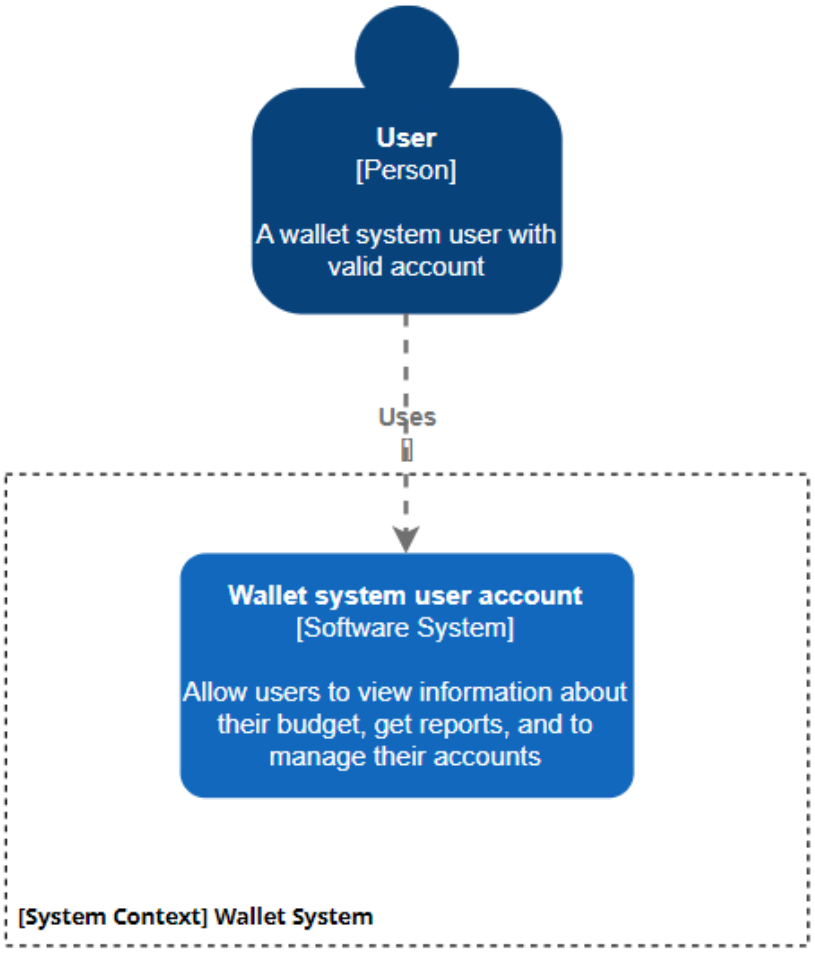

Рис.1 – Context diagram

У системі “Гаманець” буде один тип користувача “Користувач”, який матиме змогу зареєструватись у системі та використовувати її для ведення свого бюджету та керування фінансами.

Наступним кроком буде створення діаграми контейнерів. Вона показує високорівневу архітектуру ПЗ та розподіл відповідальності між нею. Вона також показує основні технологічні рішення та те, як контейнери взаємодіють один з одним:

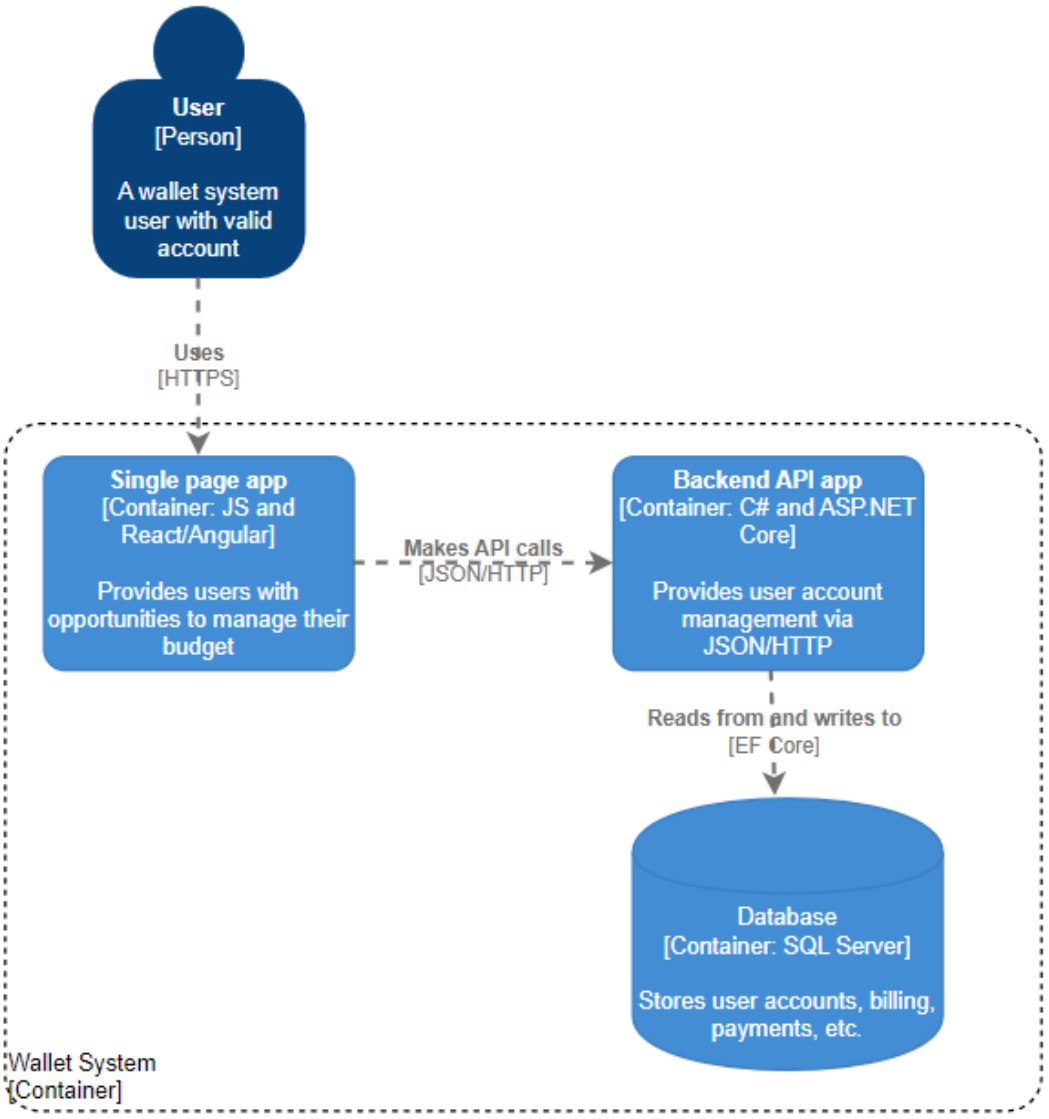

Рис.2 – Container diagram

Система керування бюджету буде складатися з трьох основних контейнерів:

1. Веб-застосунок, який надає користувачу функціонал для керування свого бюджету через веб-браузер.
2. Серверна програма, яка реалізує функціонал REST API та приймає запити від браузера.
3. База даних, яка зберігає моделі основних сутностей, які будуть задіяні у програмі.

Далі я збільшую масштаб і розкладаю кожен контейнер далі, щоб визначити основні структурні блоки та їх взаємодію. Діаграма компонентів показує, як контейнер складається з кількох компонентів, що таке кожен із цих компонентів, їхні обов'язки та деталі технології/реалізації:

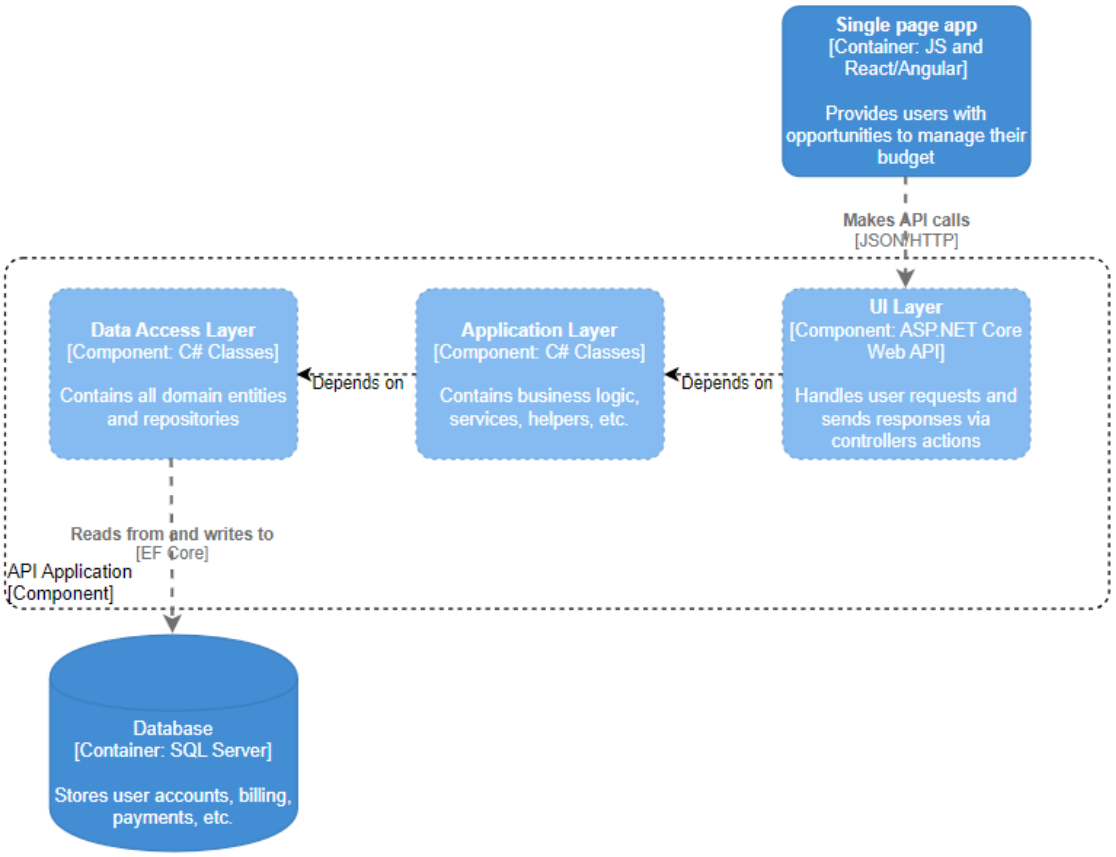

Рис.3 – Component diagram

Для розробки застосунку буде використано N-layer архітектуру, яка поділяє застосунок на 3 шари (layers):

a. **User Interface Layer** (Шар представлення): Цей рівень відповідає за взаємодію з користувачем та представлення інформації. Він містить графічний інтерфейс та обробку введення. Тут знаходяться контролери.
b. **Application Layer** (Шар бізнес-логіки): Цей рівень містить бізнес-логіку програми, яка визначає логіку операцій та обробку даних. Містить інтерфейси сервісів та їхню імплементацію.
c. **DAL** (Шар доступу до даних): Цей рівень відповідає за доступ до даних та взаємодію з базою даних.

Останнім етапом є створення Code diagram. Доменний шар буде реалізовано відповідно до сутностей бази даних, які успадковуються від класу EntityBase. Також тут міститься реалізація дженерікового репозиторію RepositoryBase<T> та контекст бази даних ApplicationContext, який в свою чергу містить набір даних усіх сутностей. Для комунікації з БД використовується EF Core.
Шар бізнес-логіки містить інтерфейси сервісів та їх імплементацію: ILoginService, IAuthService, IBudgetService, IReportService. Кожен сервіс містить певний набір функцій, які він виконує. За допомогою них плануються наступні базові запити, які зможе посилати користувач до застосунку:

1. **Додавання рахунку:**
    - Метод: POST
    - URL: /api/accounts
    - Параметри запиту: Новий об'єкт рахунку (назва рахунку, початковий баланс тощо)
    - Опис: Створення нового рахунку для користувача.
2. **Додавання транзакції:**
    - Метод: POST

- URL: /api/transactions
- Параметри запиту: Новий об'єкт транзакції (тип транзакції, сума, опис, рахунок тощо)
- Опис: Додавання нової транзакції до обраного рахунку.

3. Переказ коштів:

- Метод: POST
- URL: /api/transactions/transfer
- Параметри запиту: Відправник, отримувач та сума
- Опис: Переказ коштів з одного рахунку на інший або за вказаною статтею витрат

4. Отримання загальної суми прибутків для певного рахунку:

- Метод: GET
- URL: /api/reports/income/total/{accountId}
- Параметри запиту: accountId - ідентифікатор рахунку
- Опис: Повертає загальну суму прибутків для вказаного рахунку.

5. Отримання загальної суми витрат для певного рахунку:

- Метод: GET
- URL: /api/reports/expenses/total/{accountId}
- Параметри запиту: accountId - ідентифікатор рахунку
- Опис: Повертає загальну суму витрат для вказаного рахунку.

6. Отримання суми прибутків за певною статтею витрат для певного рахунку:

- Метод: GET
- URL: /api/reports/income/by-category/{accountId}/{category}
- Параметри запиту: accountId - ідентифікатор рахунку, category - категорія статті прибутку

- Опис: Повертає суму прибутків для вказаної категорії статті прибутку на вказаному рахунку.

7. Отримання суми витрат за певною статтею витрат для певного рахунку:

- Метод: GET
- URL: /api/reports/expenses/by-category/{accountId}/{category}
- Параметри запиту: accountId - ідентифікатор рахунку, category - категорія статті витрат
- Опис: Повертає суму витрат для вказаної категорії статті витрат на вказаному рахунку.

8. Отримання загальної інформації про рахунок:

- Метод: GET
- URL: /api/accounts/{accountId}/summary
- Параметри запиту: accountId - ідентифікатор рахунку
- Опис: Повертає загальну інформацію про рахунок, включаючи загальну суму прибутків та витрат.

9. Отримання списку рахунків:

- Метод: GET
- URL: /api/accounts
- Параметри запиту: Відсортовані за певним критерієм
- Опис: Отримання списку всіх рахунків користувача.

10. Оновлення інформації про рахунок:

- Метод: PUT
- URL: /api/accounts/{id}
- Параметри запиту: Ідентифікатор рахунку та нові дані (назва, баланс тощо)
- Опис: Оновлення інформації про рахунок.

11. Видалення рахунку:

- Метод: DELETE
- URL: /api/accounts/{id}
- Параметри запиту: Ідентифікатор рахунку
- Опис: Видалення рахунку користувача.

12. Оновлення інформації про транзакцію:

- Метод: PUT
- URL: /api/transactions/{id}
- Параметри запиту: Ідентифікатор транзакції та нові дані (сума, тип, опис тощо)
- Опис: Оновлення інформації про транзакцію.

13. Видалення транзакції:

- Метод: DELETE
- URL: /api/transactions/{id}
- Параметри запиту: Ідентифікатор транзакції
- Опис: Видалення транзакції користувача.

Шар користувацького інтерфейсу містить наступні контролери: LoginController, AuthController, BudgetController, ReportController.

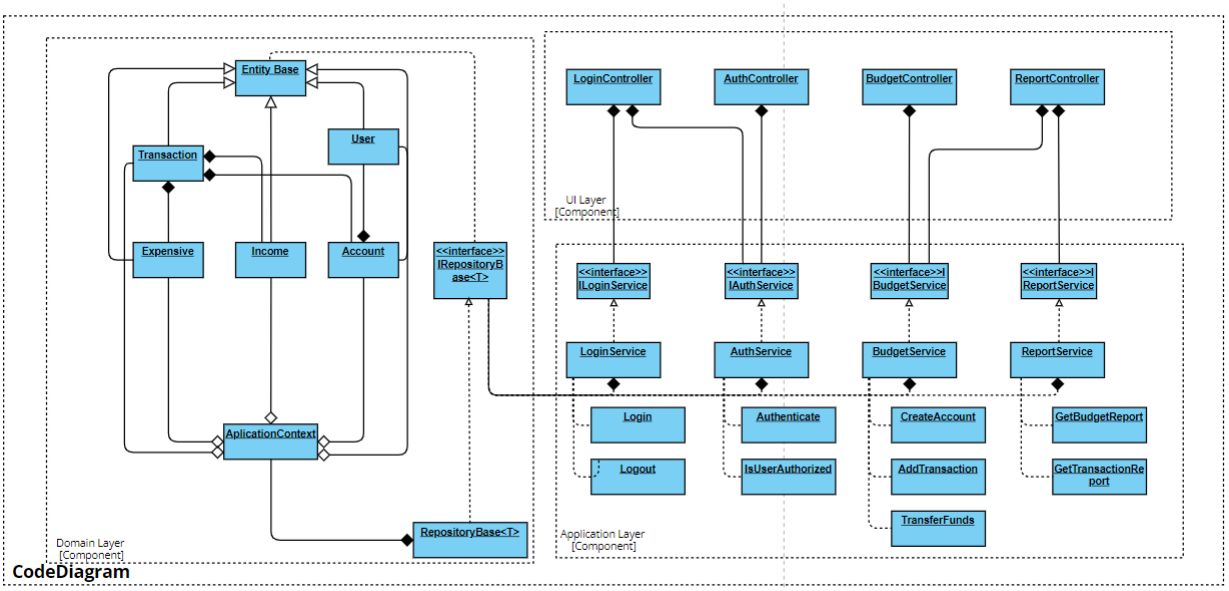

Рис.4 – Code diagram

2. Створена ER-діаграма для DAL має наступні сутності: User, Account, Transaction, Income, Expense. В свою чергу між сутностями присутні наступні зв’язки: User → Account (one to many), Account → Transaction (one to many), Transaction → Income (one to one), Transaction → Expense (one to one).

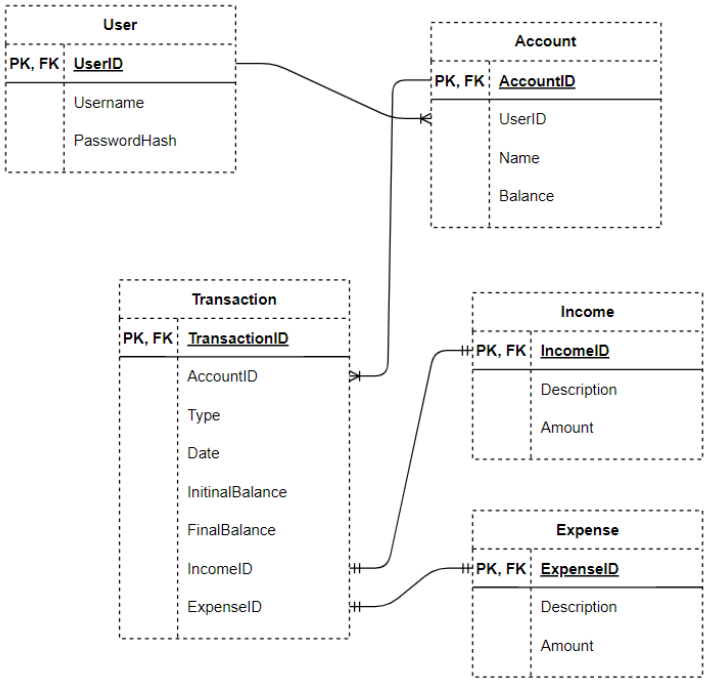

Рис.5 – ER diagram

Таблиця: User

Призначення: містить інформацію про користувача системи та використовує дані таблиці для аутентифікації.

Поля:

- UserID: GUID - унікальний ідентифікатор користувача
- UserName: VARCHAR(20) - Ім’я користувача
- PasswordHash: VARCHAR(30) - Хеш паролю користувача

Таблиця: Account

Призначення: зберігає інформацію про рахунок користувача та використовуєтьсь для веденя бюджету.

Поля:

- AccountID: GUID - унікальний ідентифікатор рахунку
- UserID: GUID - унікальний ідентифікатор користувача
- Name: VARCHAR(20) - Ім’я (опис) рахунку
- Balance: DECIMAL - Баланс рахунку

Таблиця: Transaction

Призначення: зберігає історію фінансових транзакцій, які включають прибуток та витрати на рахунку користувача.

Поля:

- TransactionID: GUID - унікальний ідентифікатор транзакції

- AccountID: GUID - унікальний ідентифікатор рахунку
- Type: ENUM - тип транзакції (прибуток або витрати)
- Date: DateTime - дата транзакції
- InitialBalance: DECIMAL - баланс до здійснення транзакцій
- FinalBalance: DECIMAL - балан опісля здійснення транзакції
- IncomeID: GUID - унікальний ідентифікатор доходу
- ExpensiveID: GUID - унікальний ідентифікатор витрати

Таблиця: Income

Призначення: містить інформацію про фінансові доходи користувача та використовується для їх відстеження.

Поля:

- IncomeID: GUID - унікальний ідентифікатор доходу
- Description: VARCHAR (50) - опис статті доходу
- Amount: DECIMAL - сума доходу

Таблиця: Expensive

Призначення: містить інформацію про фінансові витрати користувача та використовується для їх відстеження.

Поля:

- ExpensiveID: GUID - унікальний ідентифікатор витрати
- Description: VARCHAR (50) - опис статті витрати
- Amount: DECIMAL - сума витрати