КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет Комп'ютерних наук та кібернетики
Кафедра Теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра

За спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА АВТОМАТИЗОВАНОГО ТРЕКЕРА ТА АГРЕГАТОРА ВИТРАТ З ПРИВ'ЯЗАНИХ БАНКІВСЬКИХ КАРТ

Виконав студент 4-го курсу
Циронін Павло Олегович Підпис:_____

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Панченко Тарас Володимирович Підпис:_____

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент Підпис:_____

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«01» червня 2022 р.,
протокол № 10
Завідувач кафедри
М. С. Нікітченко Підпис:_____

Київ - 2022 рік

РЕФЕРАТ

Обсяг роботи 43 сторінки, 26 ілюстрацій, 5 таблиць, 12 використаних джерел.

API ОНЛАЙН-БАНКІВ, ВИПИСКИ ВИТРАТ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, КЛІЄНТ-СЕРВЕРНИЙ ЗАСТОСУНОК, РОЗРОБКА API, РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ.

Об’єктом дослідження даної роботи є витрати користувача, зроблені та зареєстровані за допомогою онлайн-банків та усі витрати, що були зроблені будь-яким стороннім способом.

Метою роботи є розробка додатку, що допоможе користувачам визначитись зі статтями витрат, які могли б бути мінімізовані та отримати розуміння того, яка сума грошей дійсно йде на важливі речі, і значно спростити рутинний процес перенесення витрат з історії переказів онлайн-банку в трекер витрат.

Методи розробки: розробка клієнт-серверного застосунку. Інструменти розробки: Microsoft SQL Server, Microsoft SQL Server Management Studio(SSMS), Microsoft Visual Studio 2022, ASP.NET Web API, ORM-технологія Entity Framework Core, Postman, Fiddler, Xamarin Forms.

Результат роботи: проаналізовано API сучасних банківських систем та реалізовано функціонал отримання виписок витрат від двох з найпопулярніших онлайн-банків в Україні для зручності ведення обліку, розроблено API-застосунок з гнучкою архітектурою та мобільний додаток для зручної роботи з витратами користувача.

Система відрізняється від найближчих популярних аналогів наявністю кастомізації категорій витрат та в той же час автоматизацією процесу перенесення банківських витрат в додаток.

Система може застосовуватись для розвитку фінансової грамотності користувачів, а саме в ролі застосунку для ведення обліку витрат. Завдяки простому інтерфейсу цільовою авдиторією такого трекера можуть бути як люди, що ніколи раніше не вели облік витрат або користувались тільки вбудованою статистикою онлайн-банків, так і досвідчені користувачі, що шукають більш зручну альтернативу ручному веденню такого обліку.

ЗМІСТ

ВСТУП

В сучасному світі проблема контролю своїх фінансів стоїть як ніколи гостро. Ведення обліку власних грошових витрат є важливою умовою досягнення фінансової грамотності будь-якою людиною. В таких обставинах важливо звертати увагу як на свої регулярні витрати, так і на непередбачувані.

Для кращого виставлення пріоритетів і, як наслідок, економії грошей були винайдені трекери витрат, куди записується кожна витрата, зроблена користувачем. Як правило, такі додатки надають опції перегляду статистики затрачених грошей по категоріям, які були додані користувачем. Але у абсолютної більшості подібних додатків є один значний спільний недолік – рутинність перенесення витрат з історії банківських переказів у систему обліку витрат. Що стосується вбудованих в клієнт-банк сервісів по отриманню статистики по витратам, вони, як правило, не мають жодної опції кастомізації категорій витрат та зовсім не підходять людям, що користуються декількома банківськими картами. Більше того, витрати зроблені готівкою в таким системах не беруться до уваги, що призводить в кінці кінців до результуючої статистики, що не відповідає реальним витрати користувача.

Метою даного проекту є розробка застосунку, який би міг допомогти користувачам з комфортом визначитись зі статтями витрат, що могли б бути мінімізовані та отримати приблизне уявлення про те, яка сума грошей дійсно є їх необхідним прожитковим мінімумом та, що не менш важливо, зрозуміти, на які категорії іде надто багато коштів, з метою більш ефективного розподілення фінансів в майбутньому.

Завданнями даної роботи є аналіз доступних API онлайн-банків, імплементація функціоналу для взаємодії з ними, розробка API-сервісу для роботи з витратами та їх категоріями, взаємодії з банками та отримання

помісячної статистики витрат по категоріям і розробка мобільного застосунку для зручної взаємодій з системою.

Об’єктом дослідження даної роботи є витрати користувача, зроблені та зареєстровані за допомогою онлайн-банків та усі витрати, що були зроблені будь-яким іншим способом.

Методи розробки: розробка клієнт-серверного застосунку. Інструменти розробки: Microsoft SQL Server, Microsoft SQL Server Management Studio(SSMS), Microsoft Visual Studio 2022, ASP.NET Web API, ORM-технологія Entity Framework Core, Postman, Fiddler, Xamarin Forms.

Система може застосовуватись для розвитку фінансової грамотності користувачів, а саме як застосунок для ведення обліку витрат. Завдяки простому інтерфейсу цільовою авдиторією такого трекера можуть бути як люди, що ніколи раніше не вели облік витрат або користувались тільки вбудованою статистикою онлайн-банків, так і досвідчені користувачі, що шукають більш зручну альтернативу ручному веденню такого обліку.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ НА РИНКУ СИСТЕМ

1.1 Android-додаток «1Money» Error! Reference source not found.

Додаток[1], який був основним джерелом ідей та досвіду використання для даного проекту(рис. 1 та 2). Має опції кастомізації категорій, підкатегорій(платна версія), відображає статистику помісячно по категоріям. Не має опції прив'язування банківських карт для отримання історії переказів.

Рисунок 1 – головна сторінка 1Money

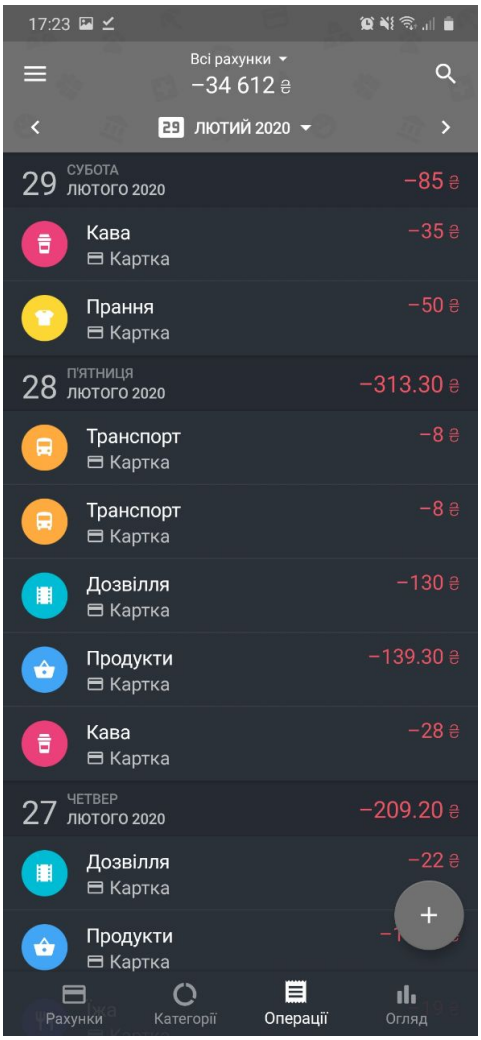

Рисунок 2 – історія витрат в 1Money

1.2 Android-додаток «Expense Manager»

На відміну від попереднього додатку, дана система[2] має поденну статистику, але будь-який з варіантів статистики доступний тільки користувачам платної версії. Не має прив’язки до карт.

Рисунок 3 - головна сторінка Expense Manager

Рисунок 4 - історія витрат в Expense Manager

1.3 Android-додаток «Hurdlr»

Цей додаток[3] має велику кількість функцій, він фіксує пройдені кілометри, допомагає рахувати податки(актуально для американських користувачів) та має опцію ведення історії витрат. Розглянемо саме останню функцію. На відміну від попередніх додатків, Hurdlr дозволяє розділяти

витрати на бізнес і витрати на побут. Тим не менш, через свою багатофункціональність та захараженість інтерфейсу(рис. 5) є менш цікавий цільовій аудиторії створеної системи.

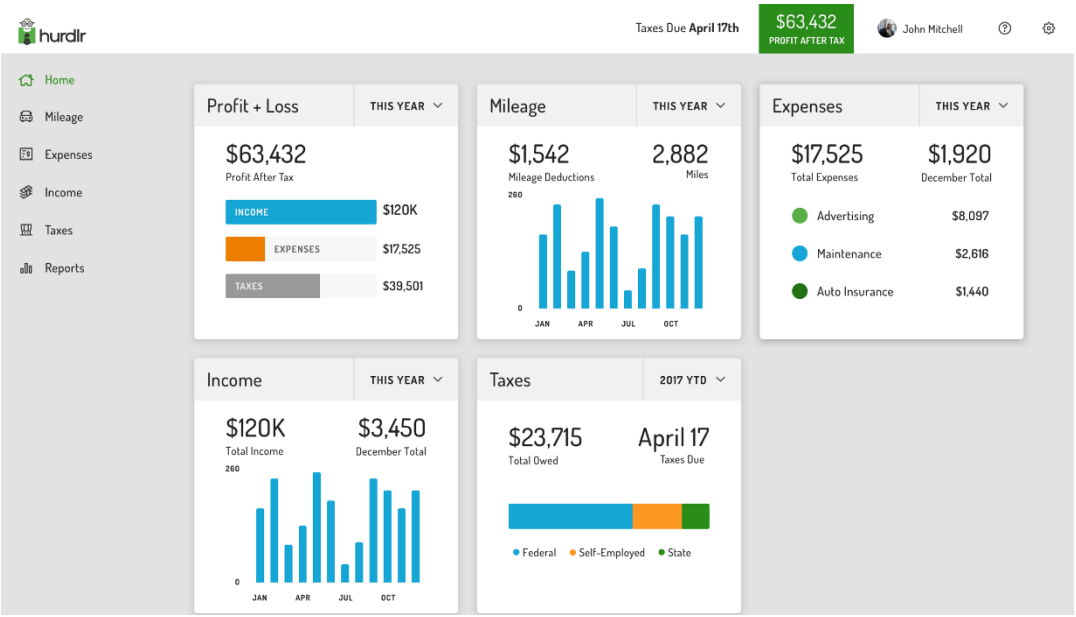

Рисунок 5 - головна сторінка Hurdlr

1.4 Android-додаток CoinKeeper

На відміну від попередніх конкурентів CoinKeeper[4] має опцію прив’язки банківських карт та запам’ятовування вказаних категорій витрат та пропонування автоматичного виставлення категорії згідно отримувача переказу. Додаток є в основі російським, тому підтримує тільки такі банки: Tinkoff, Alfa-bank, СберБанк. Для українського користувача даний продукт не надає більшого функціоналу ніж попередні перелічені трекери і загалом не цікавить.

Рисунок 6 - Головна сторінка додатку CoinKeeper

РОЗДІЛ 2. ВИМОГИ, ПРИЗНАЧЕННЯ ТА ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Вимоги до системи в цілому

Система повинна надавати функціонал для створення та редагування категорій, додавання та видалення інформації про банківські аккаунти та записи витрат. Обов’язковою умовою використання системи є авторизованість користувача, тому вона має надавати можливість реєстрації.

Система не вимагає від користувача вказувати інформацію про свої банківські рахунки для реєстрації. В додатку передбачено ручне додавання всіх витрат без прив’язки до банківських карт.

2.2 Технічні вимоги

Користувацький застосунок компілюється як під Android так і під IOS.

На довільні некоректні дії користувача, пов’язані з введенням невірних даних, не заповненням обов’язкових полів введення в формах та інші, які можуть бути оброблені системою, генеруються відповідні повідомлення про помилки англійською мовою, в межах загального дизайну додатку.

Джерелом даних для серверної частини системи є інформаційна система Microsoft SQL Server.

2.3 Призначення системи

Система була призначена для полегшення ведення обліку витрат, що сприяє кращому розумінню користувачами їх пріоритетів в питанні регулярних витрат.

2.4 Цілі створення системи

Система створена з метою отримання інформації про витрати користувача і побудови статистики виходячи з отриманих даних. Реалізація можливості проаналізувати свої витрати з максимальною зручністю використання системи є головною метою роботи.

Use-case діаграму цільового функціоналу можна побачити на рисунку 7.

Рисунок 7 - Use-Case-Діаграма проекту

РОЗДІЛ 3. ОПИС ОРГАНІЗАЦІЇ ІНФОРМАЦІЙНОЇ БАЗИ

3.1 Логічна структура бази даних

В проекті використовується БД Microsoft SQL Server. Підключення до якої реалізовано завдяки ORM Entity Framework

Діаграма робочої бази даних зображена на рисунку 8.

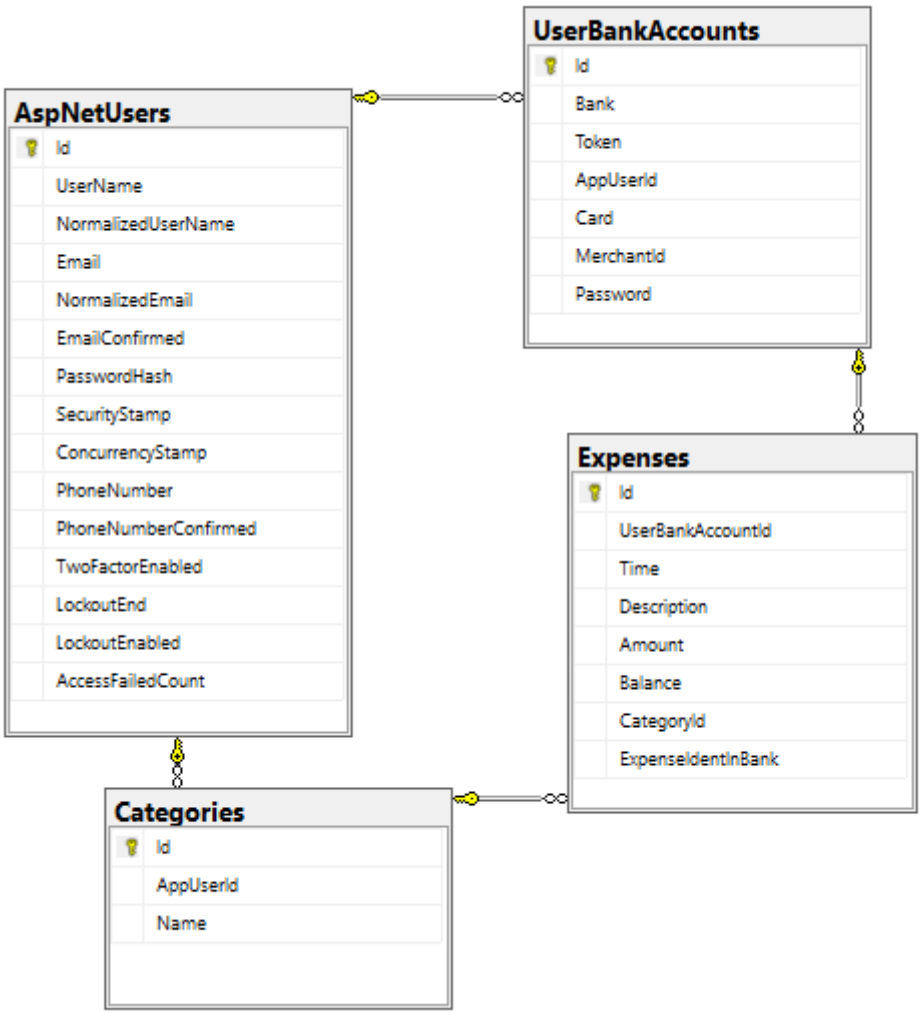

Рисунок 8 - Схема бази даних проекту

3.2 Опис таблиць

В базі даних використовується чотири наступні таблиці:

Таблиця 1 - Опис таблиць бази даних

Таблиця	**Опис**
Categories[Таблиця 2]	Таблиця з категоріями записів.
UserBankAccounts[Таблиця 3]	Таблиця з даними про банківські аккаунти користувачів.
Expenses[Таблиця 4]	Таблиця з записами занесених витрат.
AspNetUser[Таблиця 5]	Автоматично згенерована таблиця ASP.NET Identity, що містить всю необхідну інформацію про користувача.

Таблиця 2 - Атрибути таблиці Categories

Атрибут	**Тип**	**Опис**
Id	integer	Ідентифікатор
Name	nvarchar(50)	Назва категорії
AppUserId	integer	Ідентифікатор користувача, якому належить категорія

Таблиця 3 - Атрибути таблиці UserBankAccounts

Атрибут	**Тип**	**Опис**
Id	integer	Ідентифікатор
Bank	nvarchar(50)	Назва банку

Token	nvarchar(50)	Токен, що надається банком для доступу до мерчанта
AppUserId	integer	Ідентифікатор користувача, якому належить категорія
Card	nvarchar(50)	Номер карти
MerchantId	nvarchar(50)	Ідентифікатор мерчанта в банківській системі
Password	nvarchar(50)	Пароль до мерчанта

Таблиця 4 - Атрибути таблиці Expenses

Атрибут	**Тип**	**Опис**
Id	integer	Ідентифікатор
Amount	money	Сума витрати
CategoryId	integer	Ідентифікатор категорії, якій належить поточний запис
UserBankAccountId	integer	Ідентифікатор Банківського аккаунта, з якого було зроблено переказ коштів
Time	date	Дата та час додавання витрати
Description	nvarchar(50)	Опис переказу в банківській системі
Balance	money	Баланс на банківській карті після завершення переказу
ExpenseIdentifierInBank	nvarchar(50)	Запис, що однозначно ідентифікує витрату в банківській системі. Генерується безпосередньо в коді бізнес логіки. Для monobank отримується з API одразу готовий ідентифікатор, для privatbank генерується співставленням опису, часу та суми витрати.

Таблиця 5 - Атрибути таблиці AspNetUser

Атрибут	**Тип**	**Опис**
Id	nvarchar(50)	Ідентифікатор

Email	nvarchar(50)	Унікальна пошта користувача
UserName	nvarchar(50)	Ім’я користувача
PasswordHash	nvarchar(50)	Хеш пароля користувача

В даному розділі були упущені автоматично згенеровані таблиці ASP.NET Identity такі як AspNetRoleClaims, AspNetRoles, AspNetUserClaims, AspNetUserLogins, AspNetUserRoles та AspNetUserTokens оскільки вони є автоматично згенерованими таблицями Identity фреймворку і не відіграють ролі в контексті даного додатку. Деталі щодо цих таблиць та їх застосування можна переглянути на офіційному сайті документації Microsoft [5].

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Клієнт-серверний підхід

Система побудована за допомогою клієнт-серверного підходу. З логічної точки зору, розділ обов'язків відбувається наступним чином: клієнт-частина відповідає за формування інтерфейсу, відображення даних у зручній та доступній для клієнта формі, зручній для читання, сприйняття, відтворення і будь-якої взаємодії: клієнт може мати можливість змінити чи видалити певну інформацію, а також додати нові елементи.

При цьому клієнт-частина сервісу не займається зберіганням жодної інформації про користувача, за винятком особливих потреб, на кшталт кешування або збереження файлів-cookie. Однак основна інформація про користувача, його права доступу і можливості, а також про весь вміст веб-сторінки або додатку, зберігається саме на сервері. Для перегляду та будь-якої взаємодії з інформацією, клієнт має отримати команду від користувача, опрацювати її та сформувати відповідний запит на сервер.

Сервер, у свою чергу, має постійно очікувати на запит. Його завдання при отриманні запиту – якнайшвидше обробити його та надати відповідь, яка буде свідчити про успішність або неуспішність операції, запит на яку зробила клієнт-частина сервісу. Також у відповіді на запит часто міститься інформація, яку необхідно надати користувачу у відповідь на запит. У сучасному програмному забезпеченні для реалізації серверної частини проекту найбільш поширеною практикою є використання підходу REST API, в тому числі такий підхід використовувався в даному проекті.

4.2 тришарова архітектура сервера

Задля гнучкості розробки, можливості якіснішого тестування та ізоляції окремих компонент системи, в ній була використана тришарова архітектура.

Часто при використанні підходу REST при розробці програмного забезпечення серверна частина має досить специфічну архітектуру відому як тришарова архітектура. Її розподіляють на три основні шари(рис. 9) – шар API, шар бізнес-логіки, та шар безпосередньої роботи з даними.

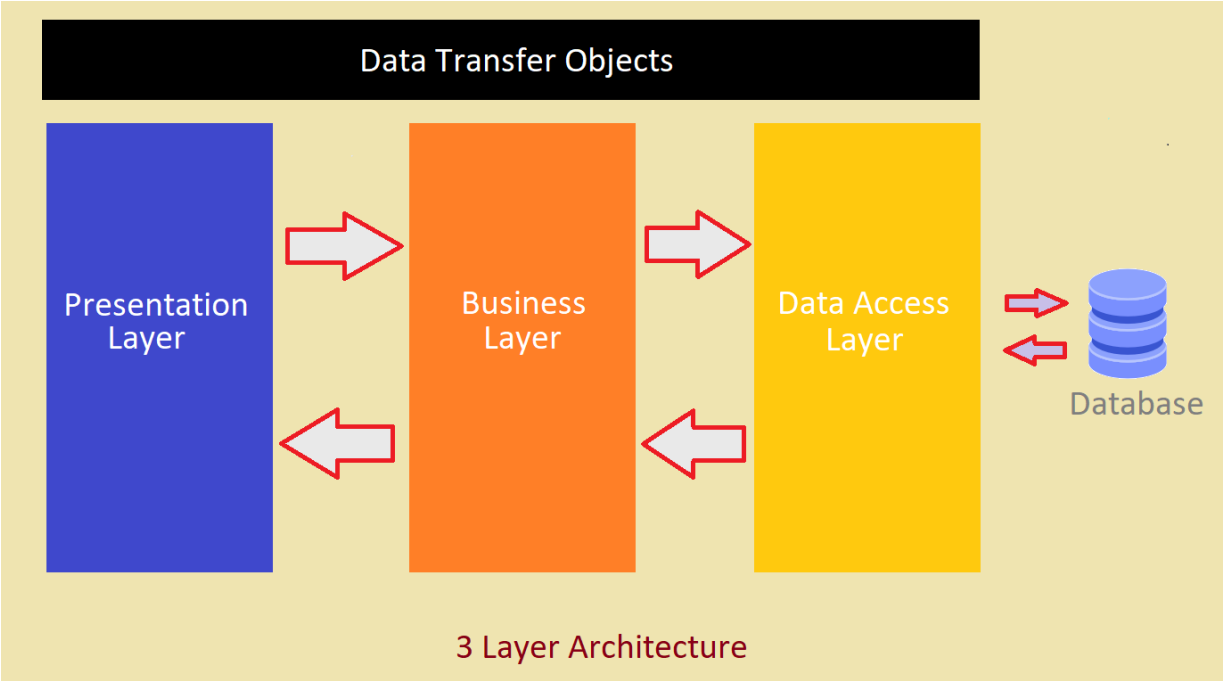

Рисунок 9. Структура застосунку з 3-шаровою архітектурою

Першим шаром, з яких складається серверна частина додатку є шар API (скорочено від application programming interface) – він реалізує отримання запитів безпосередньо від клієнтської частини сервісу, їх обробку (парсинг), та формування необхідної задачі для серверної частини сервісу. Часто в великих та добре масштабованих проектах, де обробкою запитів займається велика кількість машин, цьому кроку іноді передує балансувальний елемент(load balancer). Він розподіляє запити між різними серверами зі спільним функціоналом, щоб зменшити навантаження на систему загалом.

Останнім шаром є шар безпосередньої роботи з даними. Він реалізує підключення до бази даних і роботу з нею. Робота з базою може передбачати як прості запити, що реалізують систему доступу CRUD (Create, Read, Update, Delete), так і більш складні, композиційні запити, що можуть рахувати певні статистичні дані, або отримувати інформацію про специфіковані набори елементів. Реалізація кожного з таких запитів відбувається на основі потреб клієнта, і досить часто побудова цього шару відбувається тільки після того, як буде змодельований загальний вигляд клієнтської частини, визначені її можливості, і потреби відповідно до цих можливостей.

Також розповсюдженою практикою є підключення проміжного шару, який пов’язаний з бізнес-логікою або обробкою даних. Дані, отримані при роботі з базою даних, перебувають у необробленому вигляді, і можуть викликати складнощі з розумінням у користувача. Саме тому шар бізнес-логіки займається обробкою цих даних і перетворенням їх в такий вигляд, в якому вони будуть готові до відправки на сторону клієнта шаром API. Також бізнес-логіка може включати в себе найрізноманітніші функції – ведення обліку кількості та типів запитів, блокування запитів, що не мають бути виконані через обмеження в доступі користувача. Окрім того, що даний шар здійснює обробку даних, які передаються з бази даних на клієнт-частину, він також приймає рішення про запити, які необхідні для формування відповіді. Це пов’язано з тим, що сутності, якими оперує база даних, можуть відрізнятися від аналогічних сутностей, якими оперує клієнтська частина сервісу і сам користувач. У зв’язку з цим, для формування відповіді у потрібному форматі серверу інколи необхідно сформувати кілька запитів та надати відповідь, що отримана з їх комбінування – і задача цієї комбінації також лягає на шар бізнес-логіки.

Приклад реалізації такої архітектури в розробленій системі можна переглянути на рисунку 10. В даному випадку «DAL»(Data Access Level) - це шар доступу до бази даних, в якому реалізовано шаблони проектування

«Repository» та «Unit Of Work» [6]. «BLL»(Business Logic Level) містить в собі всю бізнес логіку проекту, включно з логікою роботи з API банків та відповідає за узгодження моделей даних, що використовуються шаром доступу до даних на шаром взаємодії з користувачем. «MoneyTracker» шо є проектом API серверного додатку в даному випадку виступає шаром взаємодії з клієнтським застосунком.

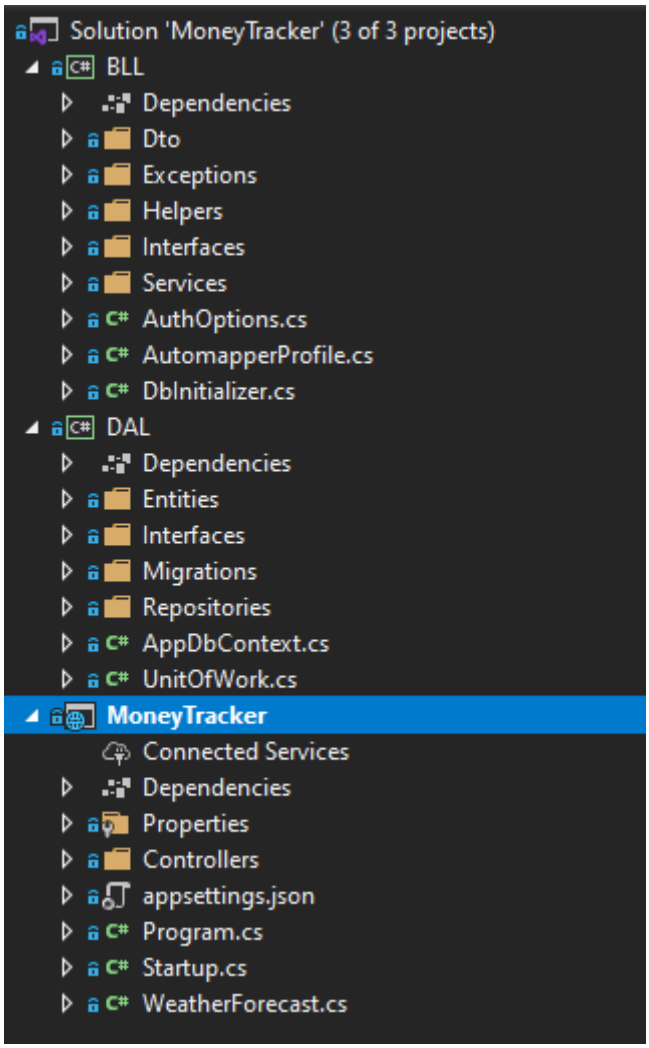

Рисунок 10 - Структура проекту серверного додатку

Дотримання такого розподілу не є обов'язковим, і на малих проектах всі ці функції можуть виконуватися одним класом чи сервісом. Однак використання такого розподілу обов'язків між різними класами чи сервісами є слідуванням одним з основних принципів об'єктно-орієнтовного програмування – інкапсуляції, що суттєво спрощує розробку, тестування і підтримку сутностей, кожна з яких має свою конкретну функцію. Відступ від

таких правил є можливим, однак потребує зваженого і вмотивованого рішення, за відсутності ж підстав діяти інакше рекомендується послуговуватися вищеописаними парадигмами.

4.3 Реалізація взаємодії з API банків

Оскільки всі банківські API мають різні інтерфейси було прийнято рішення для кожного банку створити свій клас-хелпер, основна задача якого полягає в тому, щоб маючи данні про банківський аккаунт отримати виписки витрат з карти користувача.

Для додання аккаунту кожного з банків в систему, користувачу необхідно пройти процес створення мерчанта[7] за допомогою якого надалі система зможе взаємодіяти з API банку та отримувати виписки по витратам. Процес оформлення мерчанта в онлайн банках загалом простий і дозволяє користувачу самому переконатись, що система не буде використовувати інформацію про банківський аккаунт в цілях виводу грошей чи розповсюдження небажаної для користувача інформації, оскільки в процесі відкриття мерчанта користувач сам вказує всі права, що матиме система при його використанні. В даному випадку необхідні права обмежуються отриманням виписки переказів.

4.3.1 monobank API

API monobank[8] представляє собою RESTful API, що для авторизації мерчанта використовує токен(його треба додати в хедери запиту), що знаходиться в особистому кабінеті користувача та ідентифікатор рахунку.

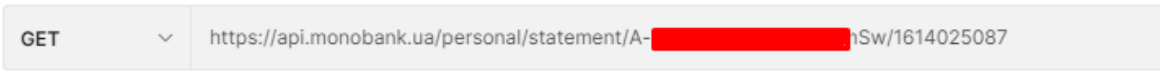

Рисунок 11 - рядок GET-запиту до API monobank

4.3.2 Privat24 API

В випадку API PrivatBank спосіб авторизації, протокол взаємодії та інформація про витрати значно відрізняються від monobank. По-перше, Privat використовує передачу повідомлень у вигляді XML. По-друге, для авторизації мерчанта використовується підпис, що створюється шляхом хешування алгоритмами SHA1 та MD5 інформації про пароль мерчанта та частини запиту, що буде відправлятись[9]. Формула отримання підпису на мові програмування PHP виглядає наступним чином: $sign=sha1(md5($data.$password)).

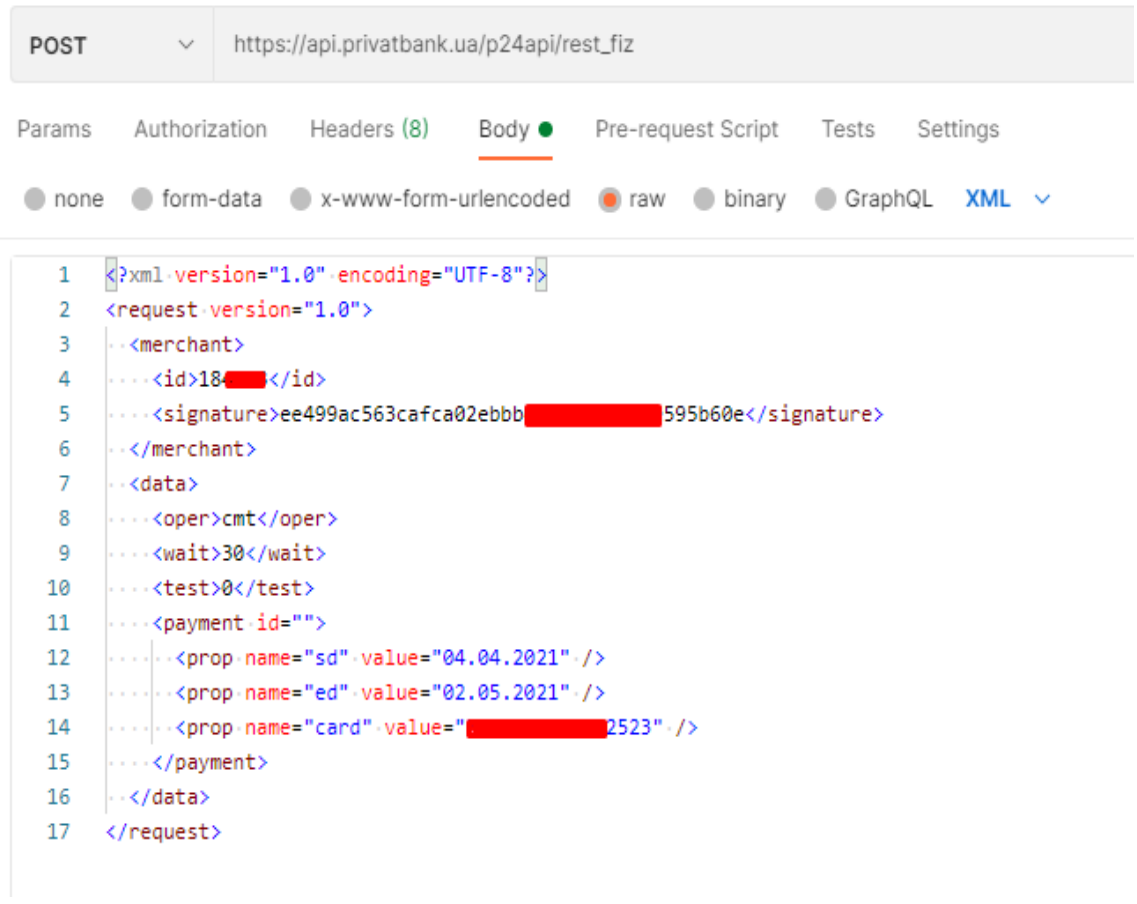

Рисунок 12 - рядок та тіло POST-запиту до API Privat 24

4.4 Огляд використаних технологій

Для побудови бази даних було використано Microsoft SQL Server Management Studio. Для створення серверного застосунку - Microsoft Visual Studio 2022 та технологія ASP.NET Web API. Для клієнт-додатку: Xamarin Forms. Для зручнішого використання бази даних використовувалась ORM-технологія Entity Framework Core. В порівнянні з технологією Dapper, EF Core програє в швидкості, а код, згенерований EF Core потрібно підтримувати самому розробнику. Тим не менш вона є найбільш розповсюдженою серед подібних технологій для .NET-застосунків і пропонує досить низький поріг входу для нових розробників. Для зіставлення Моделей шару DAL та API та приведення інформації про витрати, отримані від банків, до спільного вигляду, використовувалась бібліотека AutoMapper. Для відображення діаграм в застосунку Xamarin Forms було використано бібліотеку Microcharts. В якості системи контролю версій було використано Git і GitHub як веб-сервіс для цих цілей. В якості СКБД було використано Microsoft SQL Server Management Studio(SSMS).

РОЗДІЛ 5. ІНСТРУКЦІЯ КОРИСТУВАЧА

5.1 Авторизація в системі

Для входу в систему користувач має бути зареєстрований, тому при першому запуску додатку демонструється наступна сторінка з можливістю зареєструватися(рис. 13) або увійти під існуючим логіном(рис. 14) якщо користувач вже реєструвався в системі протягом роботи з мобільним додатком:

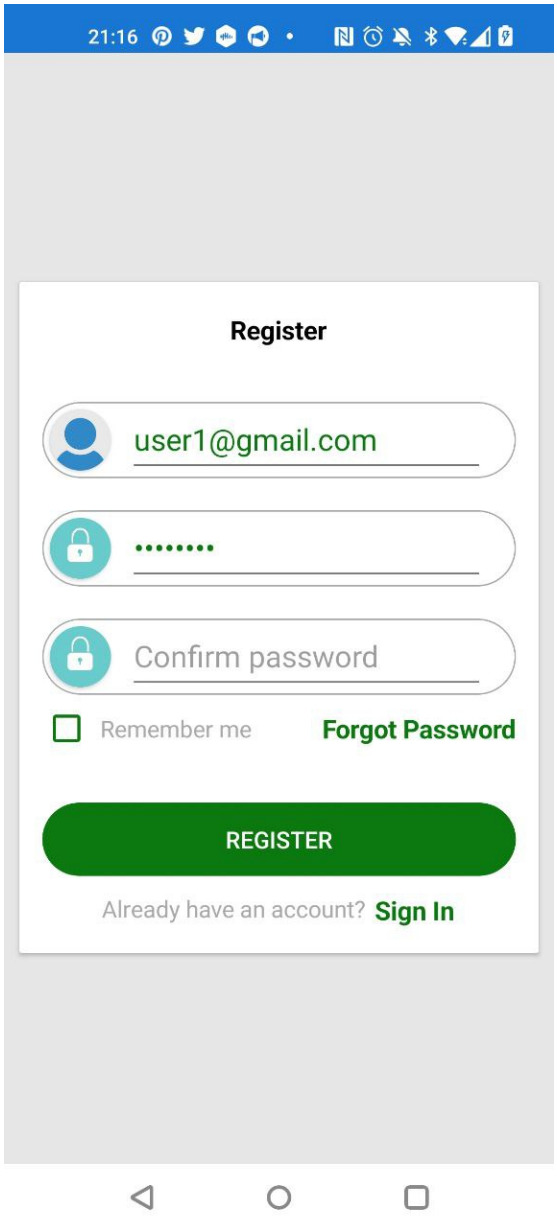

Рисунок 13- Сторінка реєстрації

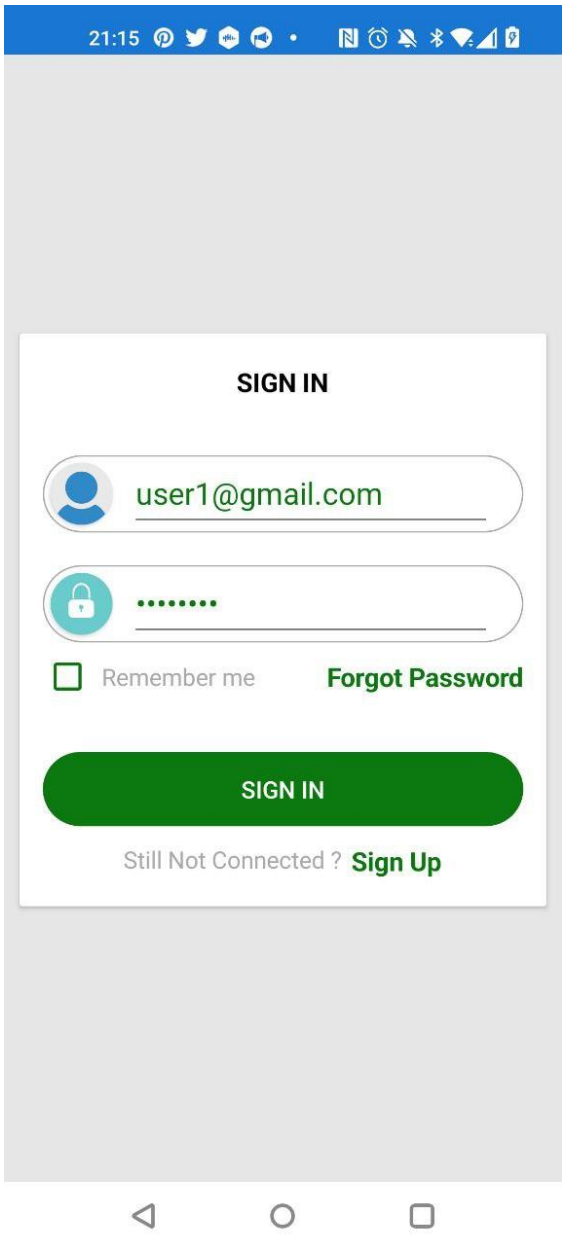

Рисунок 14 - Сторінка входу в систему

5.2 Історія витрат

При вході в систему користувача зустрічає головна сторінка(рис. 15), на якій демонструється список останніх категоризованих банківських витрат разом з власноруч доданими витратами. На кожному елементі списку відображається опис, сума, категорія та час витрати.

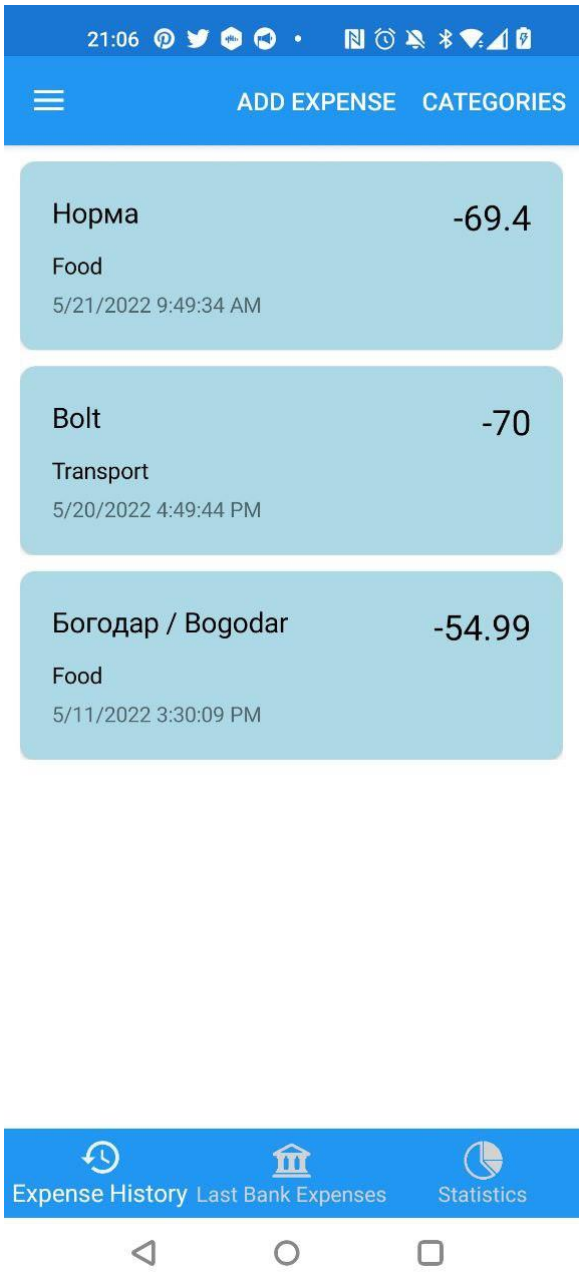

Рисунок 15- Головна сторінка з історією витрат

При натисканні на будь-яку витрату, її можна видалити(рис.16).

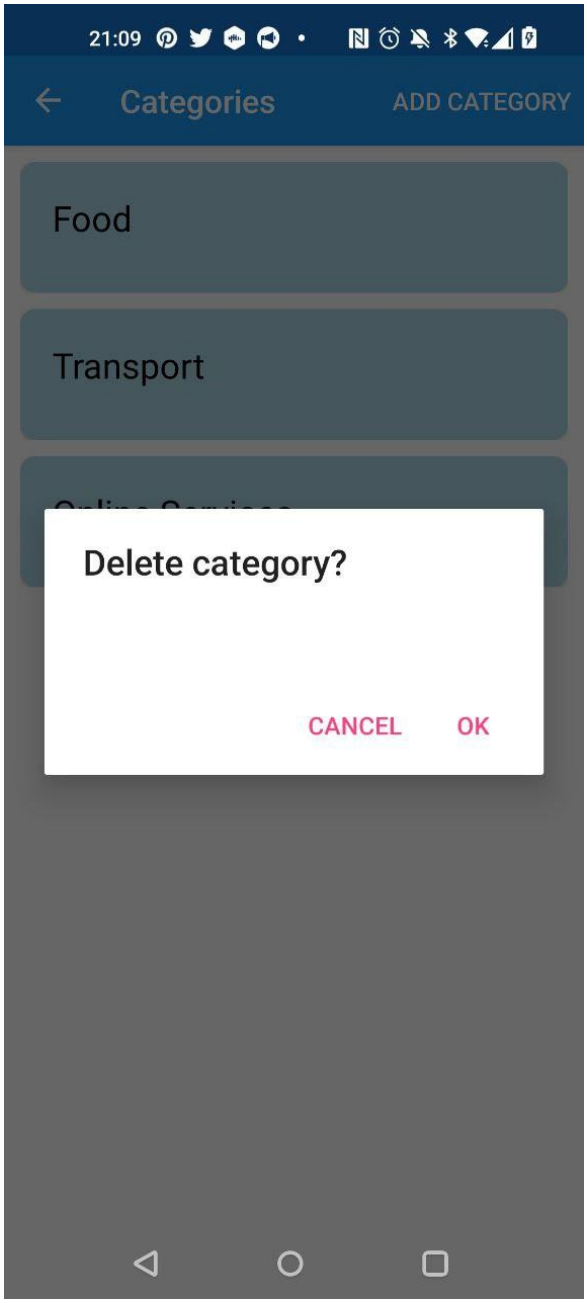

Рисунок 16 - Діалогове вікно з підтвердженням видалення витрати

5.3 Робота з банківськими витратами

При переключенні на другу вкладку в нижньому меню, користувач направляється на сторінку з останніми банківськими витратами(рис. 17) з його доданих карт, що ще не були категоризовані.

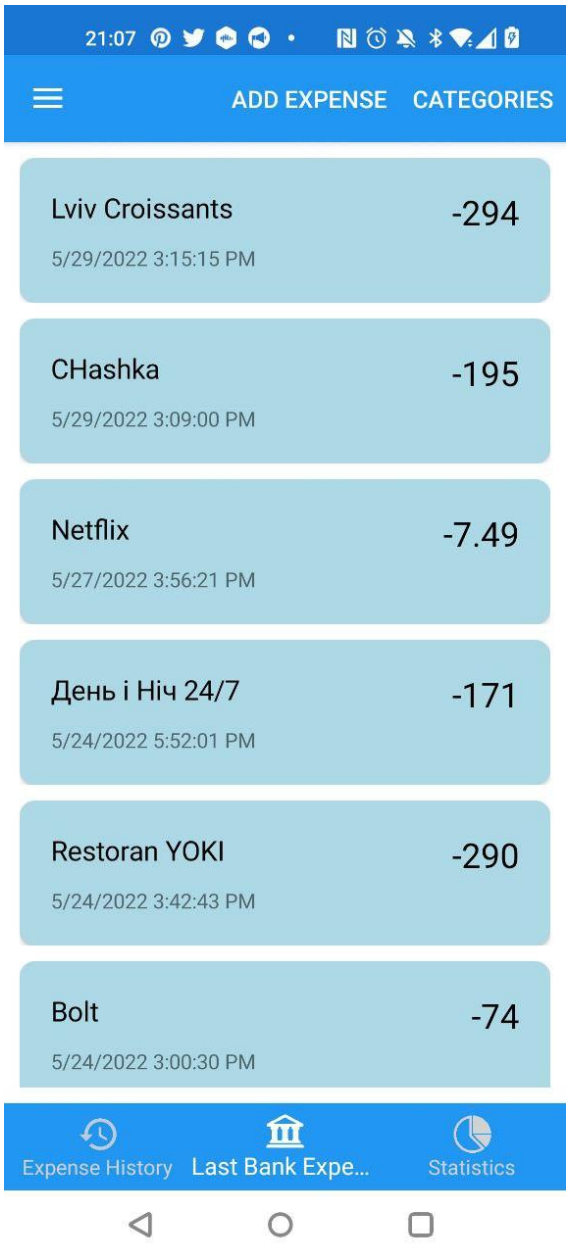

Рисунок 17- Сторінка з останніми банківськими витратами користувача

При натисканні на будь-який запис банківської витрати з’являється діалогове вікно для вказання категорії витрати для внесення її в історію витрат(рис. 18).

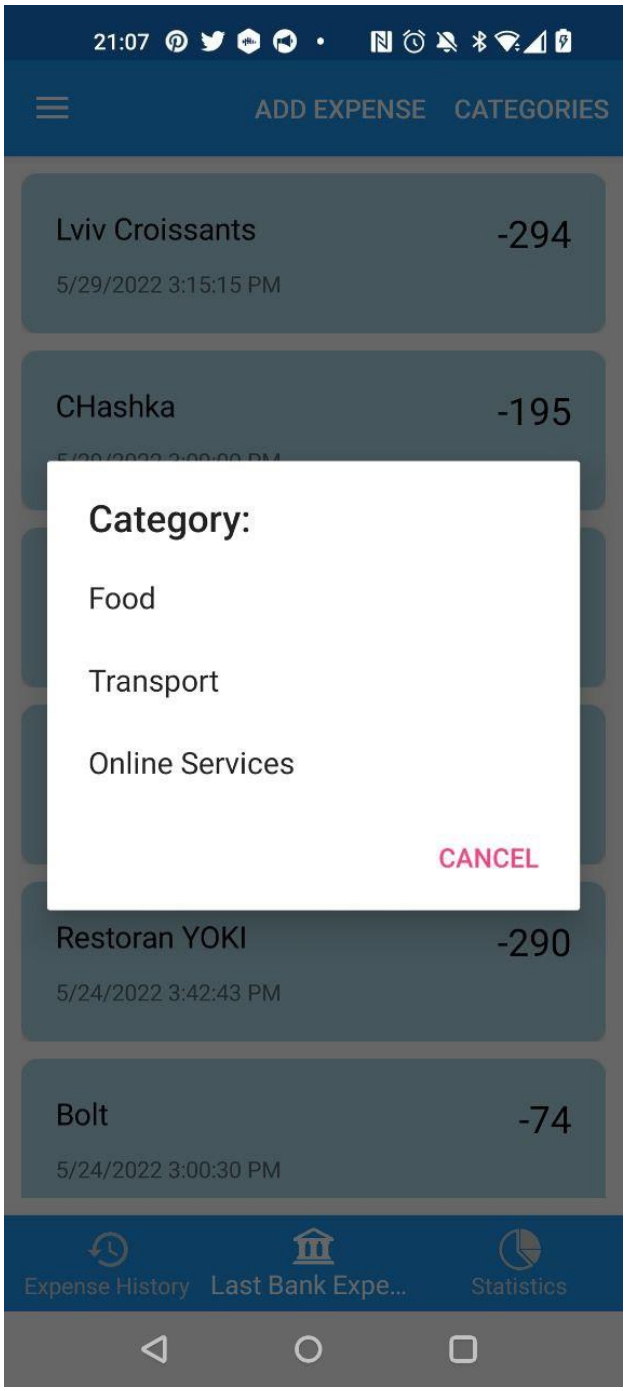

Рисунок 18 - Діалогове вікно для вказання категорії банківської витрати

5.4 Статистика витрат

При переключенні на третю вкладку(рис. 19) в нижньому меню, користувач направляється на сторінку, де відображена місячна статистика категоризованих банківських та доданих власноруч витрат по категоріям.

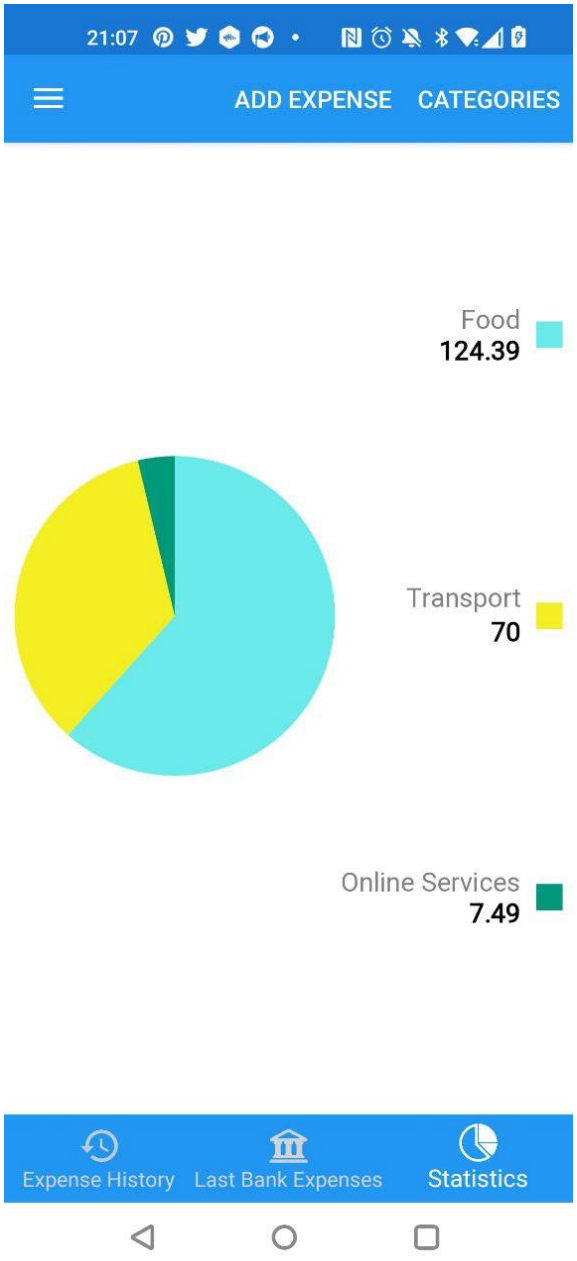

Рисунок 19 - сторінка зі статистикою витрат користувача

5.5 Додавання витрат власноруч

Кнопка «Add expense» на навігаційній панелі направляє користувача на сторінку ручного додавання нової витрати(рис. 20). На ній необхідно вказати суму витрати, категорію витрати та дату витрати.

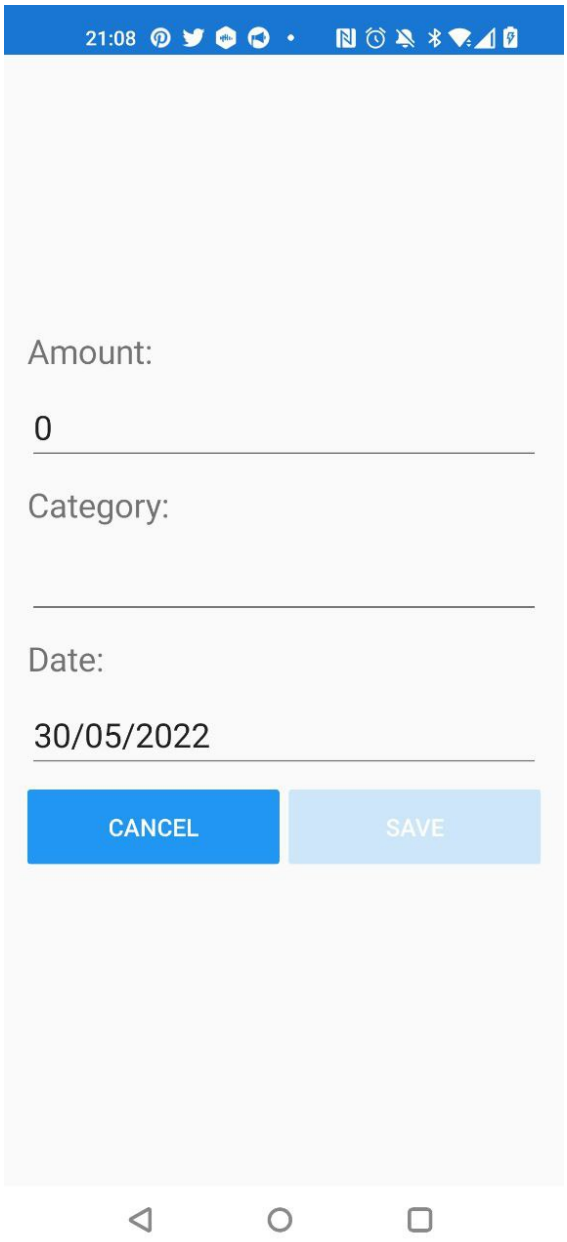

Рисунок 20 - Сторінка для додавання нової витрати

При натисканні на поле для категорії, на екрані демонструється діалогове вікно з пропозицією вибрати одну з раніше створених категорій(рис. 21).

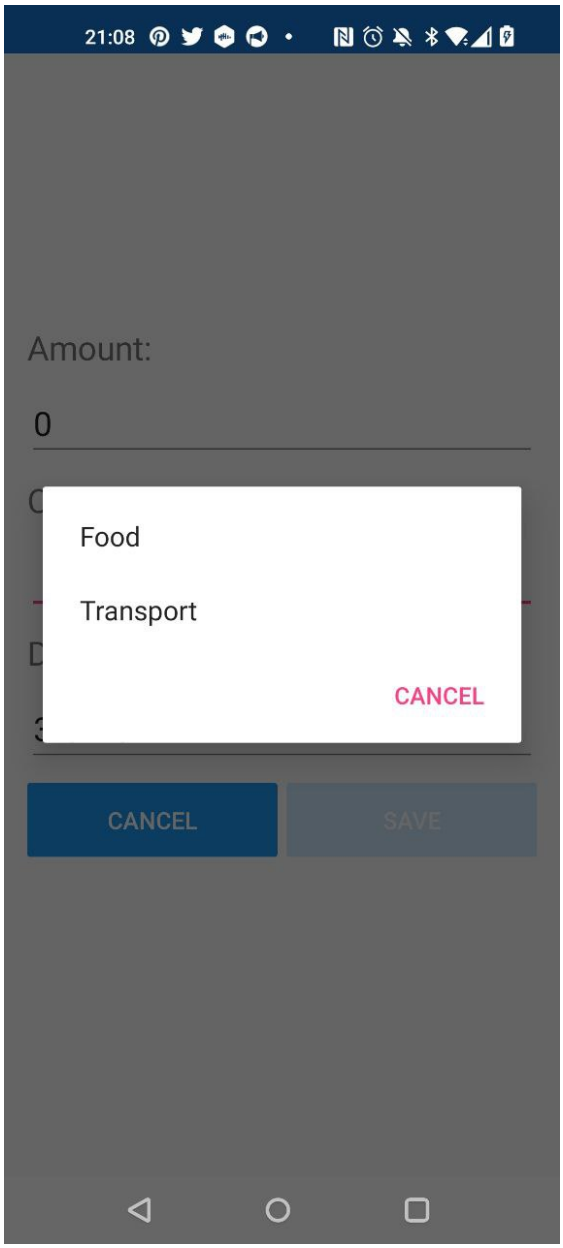

Рисунок 21 - Діалогове вікно для вказання категорії нової витрати

При натисканні на поле для дати, демонструється вікно для вибору дати на календарі(рис. 22). За замовчуванням обрана поточна дата.

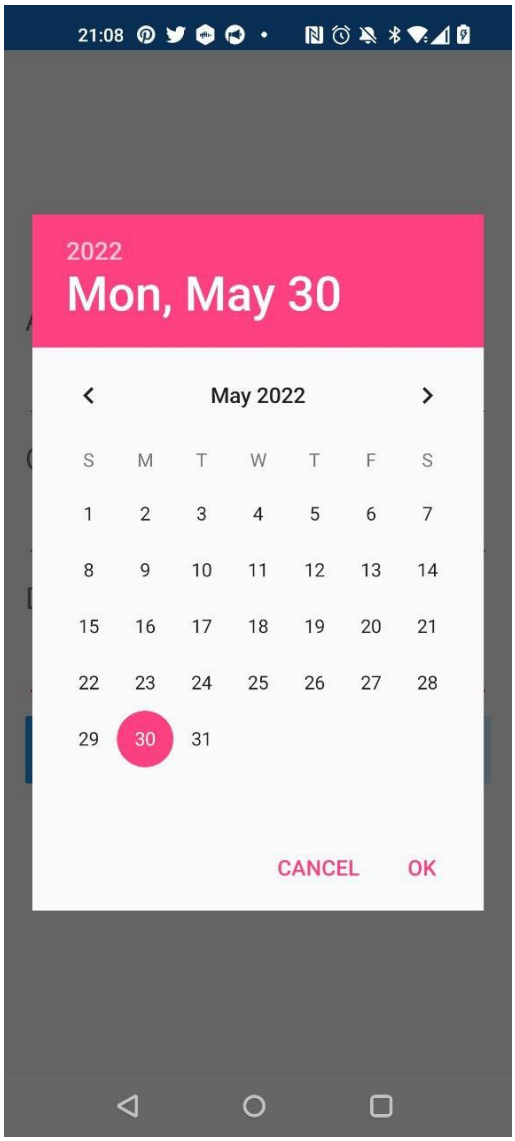

Рисунок 22 - Вікно для вказання дати нової витрати

5.6 Управління категоріями

Кнопка «Categories» навігаційної панелі переносить користувача на сторінку з категоріями(рис. 23).

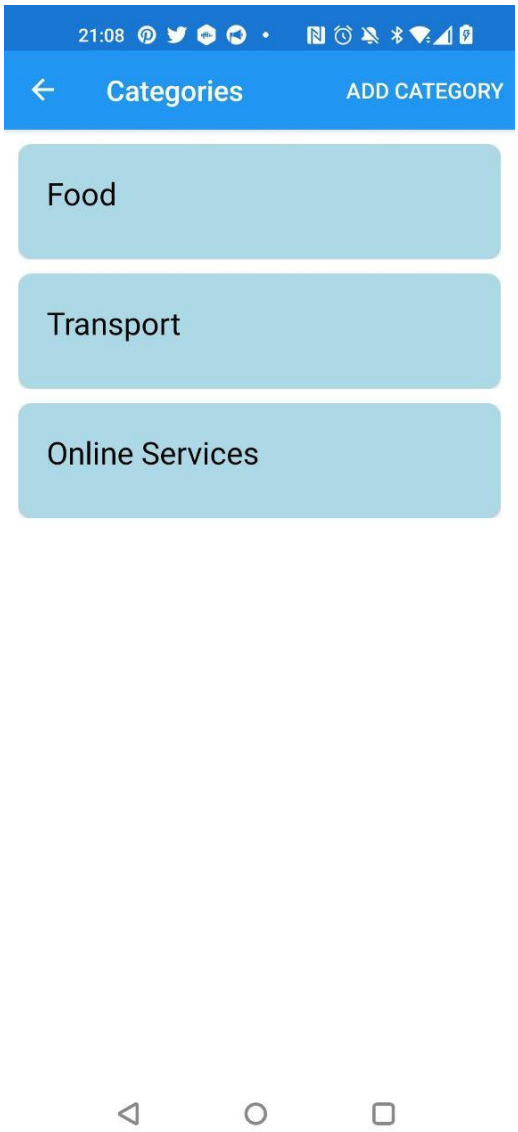

Рисунок 23 - Сторінка з категоріями витрат користувача

Користувач може додати нову категорію натиснувши на кнопку «Add category», де додаток запропонує йому ввести назву категорії(рис. 24).

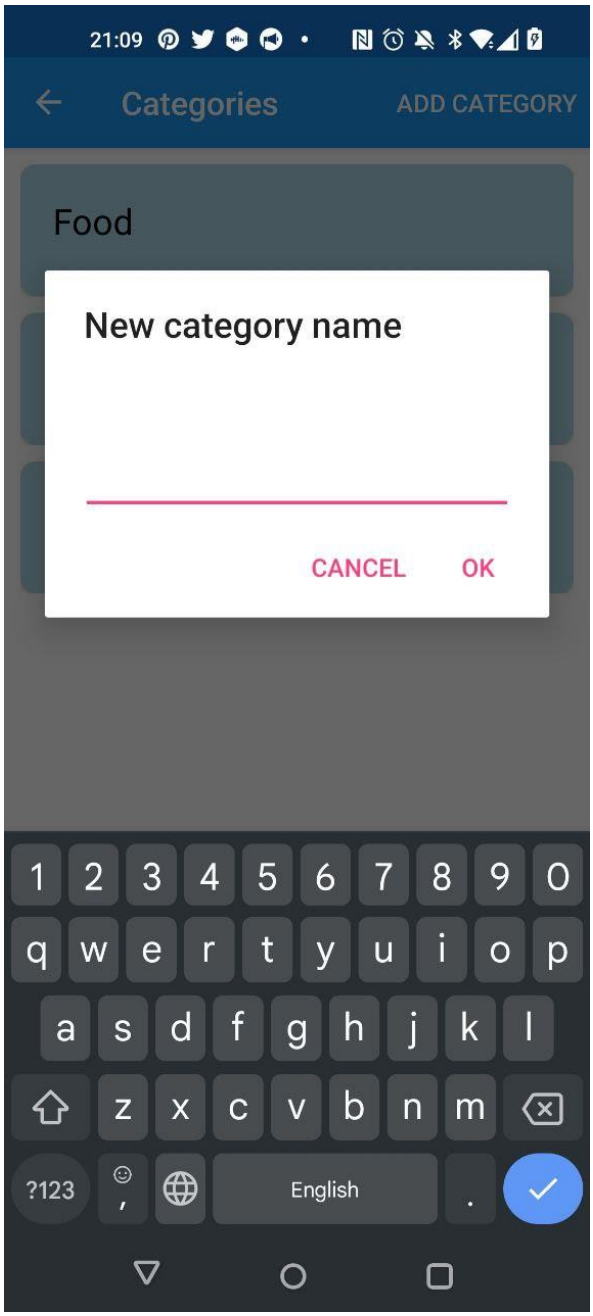

Рисунок 24- Діалогове вікно для вказання імені нової категорії

При натисканні на будь-яку категорію користувач може її видалити(рис. 25).

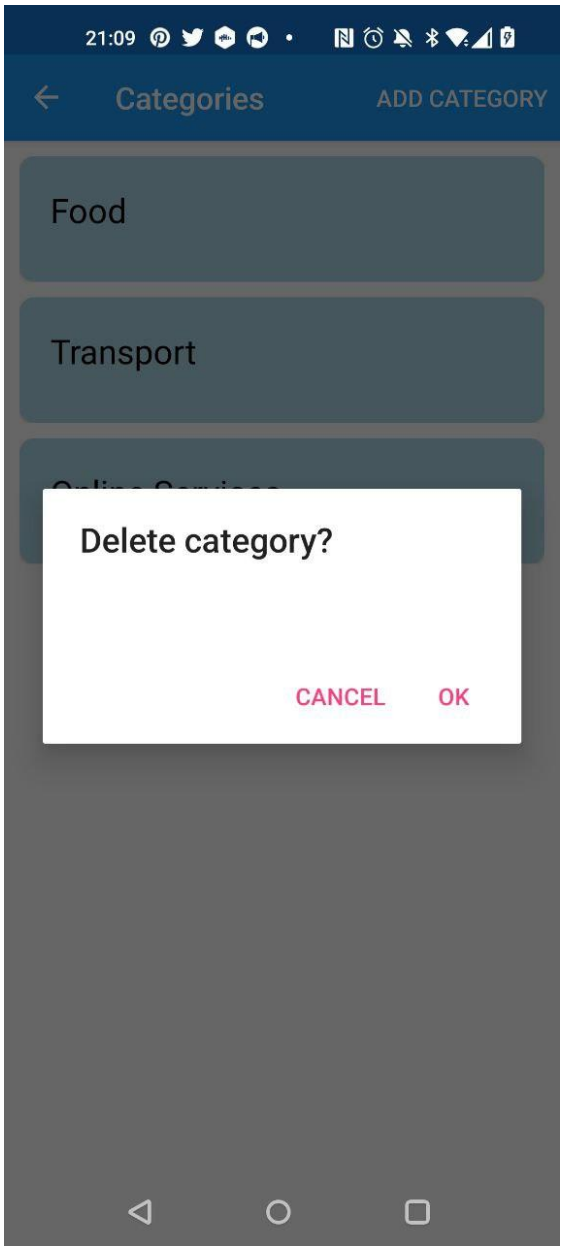

Рисунок 25 - Попередження про видалення категорії

5.7 Вихід з системи

Користувач може вийти з системи натиснувши на кнопку «Log out» на боковій панелі додатку(рис. 26).

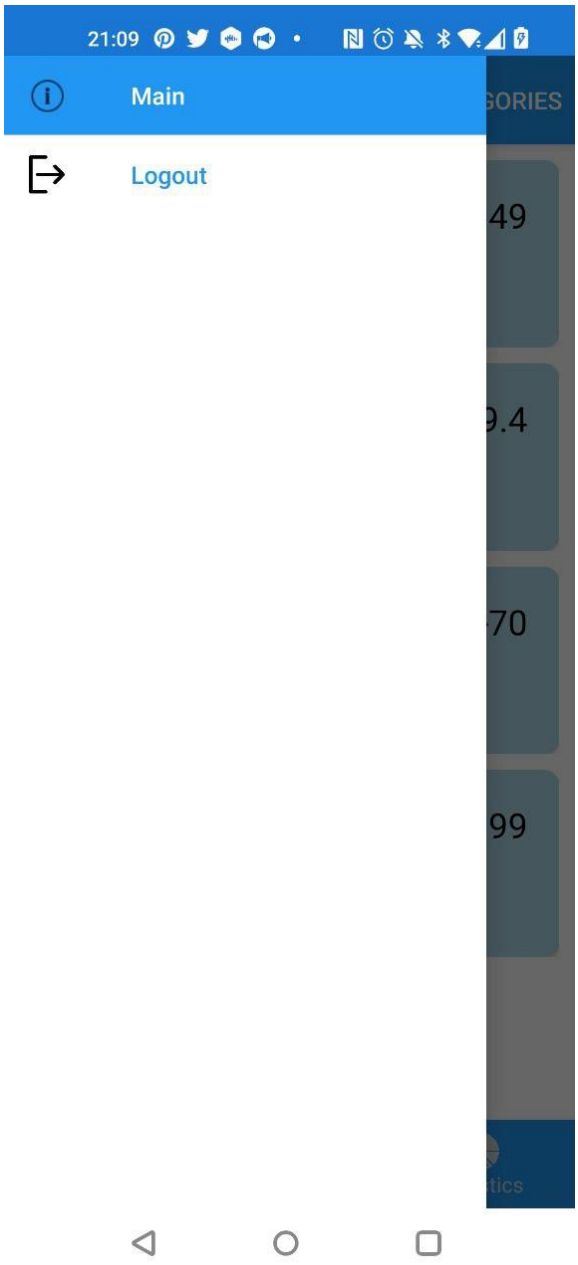

Рисунок 26 - Бокова панель додатку

ВИСНОВКИ

В ході даної роботи було проаналізовано схожі існуючі системи, що вже мали успіх у своїй сфері. Виконано дослідження відкритих API онлайн-банків. Зіставивши популярність банку та функціонал API зроблено вибір на користь певних банків та реалізовано спільну логіку для роботи з транзакціями користувачів в цих банках. На основі цього розроблено клієнт-серверну систему з RESTful API, 3-рівневою архітектурою на стороні серверу та мобільним додатком з використанням Xamarin Forms на стороні клієнта. В системі реалізовано авторизацію за допомогою технології JavaScript Web Token(JWT).

Система відрізняється від найближчих популярних аналогів наявністю кастомізації категорій витрат та в той же час автоматизацією процесу перенесення банківських витрат в додаток.

Система може застосовуватись для розвитку фінансової грамотності користувачів, а саме як застосунок для ведення обліку витрат. Завдяки простому інтерфейсу цільовою авдиторією такого трекера можуть бути як люди, що ніколи раніше не вели облік витрат або користувались тільки вбудованою статистикою онлайн-банків, так і досвідчені користувачі, що шукають більш зручну альтернативу ручному веденню такого обліку.

Потенціал автоматизації згаданого вище рутинного процесу перенесення витрат не обмежується лише отриманням виписок витрат з банків. Деякі системи онлайн-банкінгу також надають можливість нотифікації клієнтського додатку кожен раз коли відбувається транзакція. Завдяки цьому можна реалізувати процес отримання push-повідомлення користувачем кожен раз коли за карточкою була зроблена операція та пропонувати одразу категоризувати дану витрату.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Додаток “1Money” [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=org.pixelrush.moneyiq&hl=ru.
2. Додаток “Expense Manager” [Електронний ресурс] – Режим доступу до ресурсу: https://4pda.ru/forum/index.php?showtopic=444462.
3. Додаток “Hurdlr” [Електронний ресурс] – Режим доступу до ресурсу: https://www.hurdlr.com/.
4. Додаток “CoinKeeper” [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.disrapp.coinkeeper.material&hl=ru&gl=US.
5. Identity model customization in ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-6.0.
6. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application.
7. Investopedia - Merchant Account [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/terms/m/merchant-account.asp#:~:text=A%20merchant%20account%20is%20a,in%20an%20electronic%20payment%20transaction.
8. Monobank API [Електронний ресурс] – Режим доступу до ресурсу: https://api.monobank.ua/docs/.

9. Privat24 API [Електронний ресурс] – Режим доступу до ресурсу: https://api.privatbank.ua/#p24/orders.
10. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Irvine : University of California, 2000. 180 p.
11. GitHub-репозиторій з кодом серверної частини системи [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Tsyronin/MoneyTracker.
12. GitHub-репозиторій з кодом клієнтської частини системи [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Tsyronin/XamApp.