Bazy Danych

Kierunek	Termin
Automatyka i Robotyka	*Czwartek 15.15 - 16.55*
Imię, nazwisko, numer albumu,grupa	**Data**
Mateusz Szlachetko 259370 *Jakub Zając 259362* *Anna Kopij 259336*	*26 czerwca 2023*
Temat Projektu	**Prowadzący**
Baza danych do zarządzania budżetem domowym.	*dr inż. Roman Ptak*

Spis treści

1 Wstęp

1.1 Cel projektu

Celem projektu jest stworzenie aplikacji bazodanowej do zarządzania budżetem domowym.

1.2 Opis projektu

Aplikacja w swojej docelowej formie, ma umożliwić i ułatwić zarządzanie budżetem domowym. W tym celu użytkownicy powinni mieć możliwość podglądu swojego budżetu, wpłacania zarobionych środków, czyli w ogólności kontrolę dochodów, a także możliwość śledzenia wydatków. Jeżeli miesięczny budżet będzie stabilny i użytkownicy będą mogli pozwolić sobie na oszczędzanie środków, aplikacja umożliwi im także planowanie celów na które mogą te środki przeznaczyć. Będą istniały 2 rodzaje użytkowników: dorosły i dziecko. Dorosły będzie miał dostęp do pełnej funkcjonalności: będzie widział cały budżet, a dziecko tylko ograniczony fragment budżetu nazwany kieszonkowym.

2 Analiza wymagań

2.1 Wymagania funkcjonalne

- Dwa rodzaje użytkowników:
  - Dorosły
  - Dziecko
- Dorosły:
  - Zarządzanie dochodami
  - Zarządzanie wydatkami
  - Kategoryzowanie wydatków
  - Kategoryzowanie dochodów
  - Przegląd historii wydatków
  - Przegląd historii wpływów
  - Przegląd wydatków w danym czasie (np. w miesiącu, roku)
  - Przegląd wydatków z danej kategorii
  - Ustalanie celów finansowych (tworzenie wishlisty)
  - Tworzenie podbudżetu na oszczędności (możliwość przekazywania do niego pieniędzy z głównego budżetu i z powrotem)
  - Podgląd oszczędności (wgląd w podbudżet z oszczędnościami)

- Dziecko:
  - Podgląd jedynie swojego podbudżetu (kieszonkowego)
  - Zarządzenie dochodami (oprócz kieszonkowego od rodziców, możliwość okazjonalnych wpłat np. prezent od babci)
  - Zarządzanie swoimi wydatkami
  - Kategoryzowanie swoich wydatków
  - Przegląd historii swoich wydatków
  - Przegląd historii swoich wpływów (kieszonkowe, wpłaty z prezentów)
  - Przegląd swoich wydatków w danym czasie (np. w miesiącu, roku)
  - Przegląd swoich wydatków z danej kategorii
  - Ustalanie własnych celów finansowych (tworzenie wishlisty)

  Uwaga: dziecko nie ma wglądu do głównego budżetu dorosłych.

2.2 Wymagania niefunkcjonalne

2.2.1 Wykorzystywane technologie i narzędzia

- System zarządzania relacyjną bazą danych (DBMS): SQLite
- Język programowania, technologie: Python 3, Angular
- Aplikacja webowa (hostowana lokalnie) z graficznym interfejsem użytkownika

2.2.2 Wymagania dotyczące rozmiaru bazy danych

Przewidywania ilości przechowywanych danych:

- Użytkownicy: kilka-kilkanaście
- Budżet: kilka (Główny i kieszonkowe dzieci)
- Wpływy miesięcznie: kilka-kilkanaście
- Wydatki miesięcznie: kilkadziesiąt-kilkaset
- Kategorie wydatków: kilkadziesiąt
- Kategorie wpływów: kilka

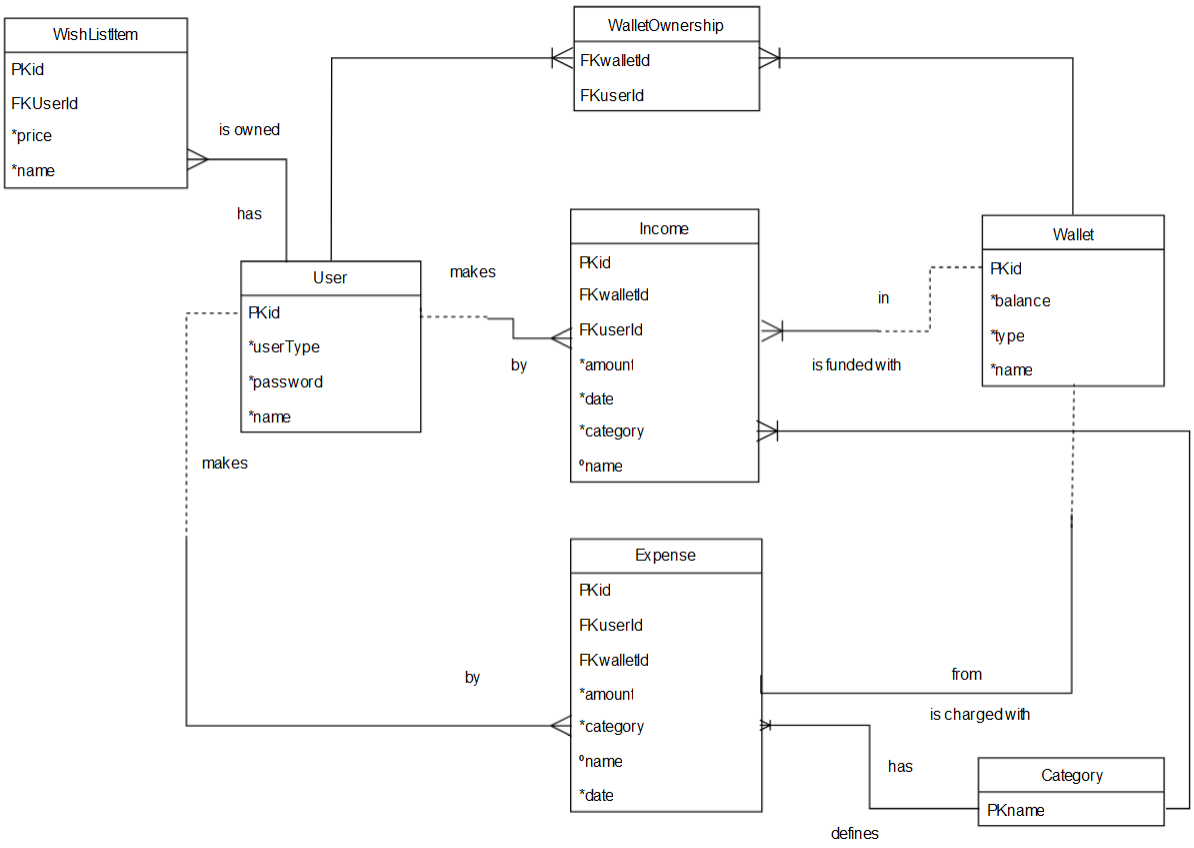
2.2.3 Wymagania dotyczące bezpieczeństwa systemu

Aplikacja będzie uruchamiana lokalnie, dlatego nie wymaga szczególnej dodatkowej autoryzacji. Przewidywana forma zabezpieczenia to login i hasło dla każdego użytkownika.

3 Projekt systemu

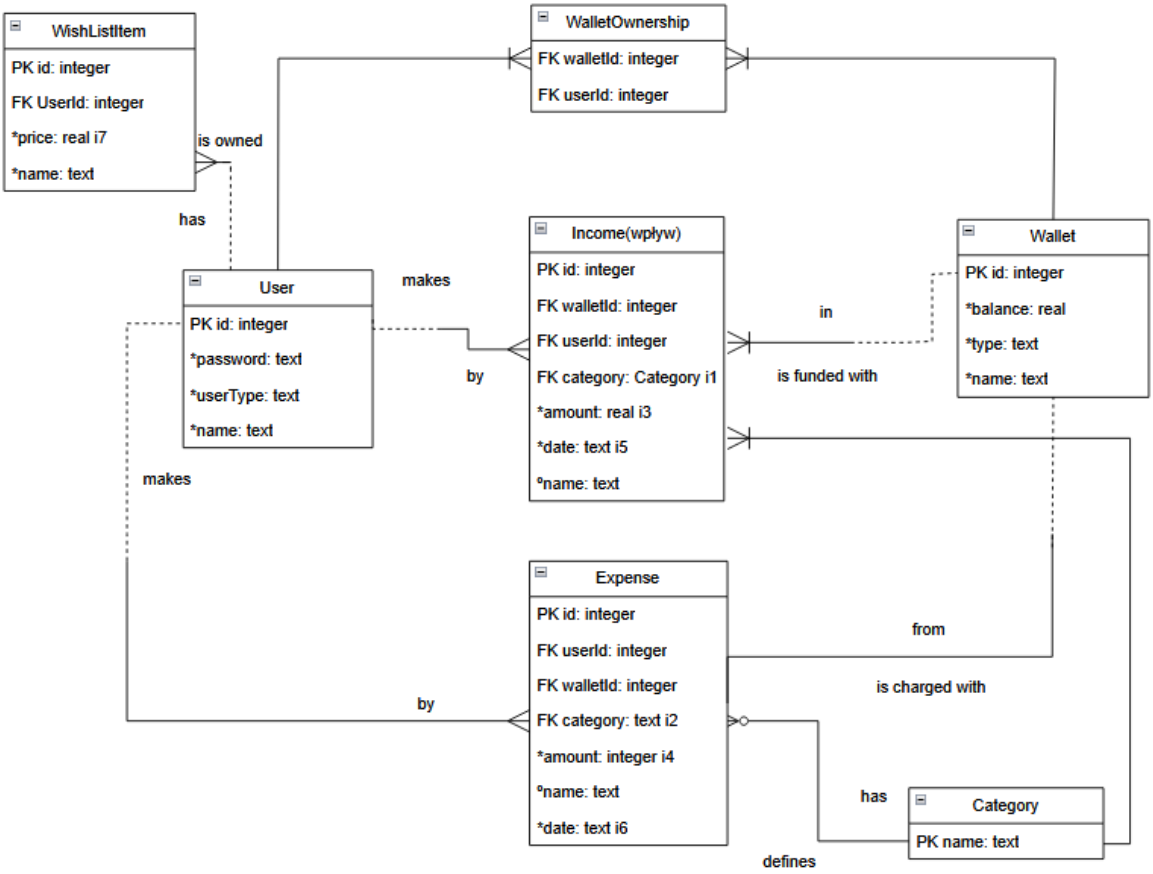
3.1 Projekt bazy danych

3.1.1 Model logiczny



Rysunek 1: Model logiczny znormalizowany

3.1.2 Model fizyczny



Rysunek 2: Model fizyczny zgodny z SQLite

3.2 Projekt wybranych funkcji systemu

3.2.1 Sekwencje

1. UserId: zaczyna się od 1, co 1
2. WalletId zaczyna się od 1, co 1
3. ExpenseId dla każdego portfela zaczyna się od 1, co 1
4. IncomeId dla każdego portfela zaczyna się od 1, co 1
5. WishListId dla każdej listy zaczyna się od 1, co 1

3.2.2 Procedury

1. AddExpense - wykonuje zbiór instrukcji do utworzenia wydatku, przy parametrach wywołania: id, userId, walletId, amount, category, name, date
2. AddIncome - wykonuje zbiór instrukcji do utworzenia przychodu, przy parametrach wywołania: id, userId, walletId, amount, category, name, date
3. CreateWallet - tworzy portfel, przy parametrach: id, balance, type
4. CreateUser - tworzy użytkownika przy parametrach: id userType, password, name
5. AddWishListItem - wykonuje zbiór instrukcji do dodania nowego obiektu, przy parametrach wywołania: id, userId, price, name

3.2.3 Trigery

1. Po dodaniu do tabeli "Income" o kwocie "amount" i danego "walletId" wyzwala zasilenie "balance" "Walleta" o podanym id.
2. Po dodaniu do tabeli "Expense" o kwocie "amount" i danego "walletId" wyzwala zasilenie "balance" "Walleta" o podanym id.

3.2.4 Indeksy

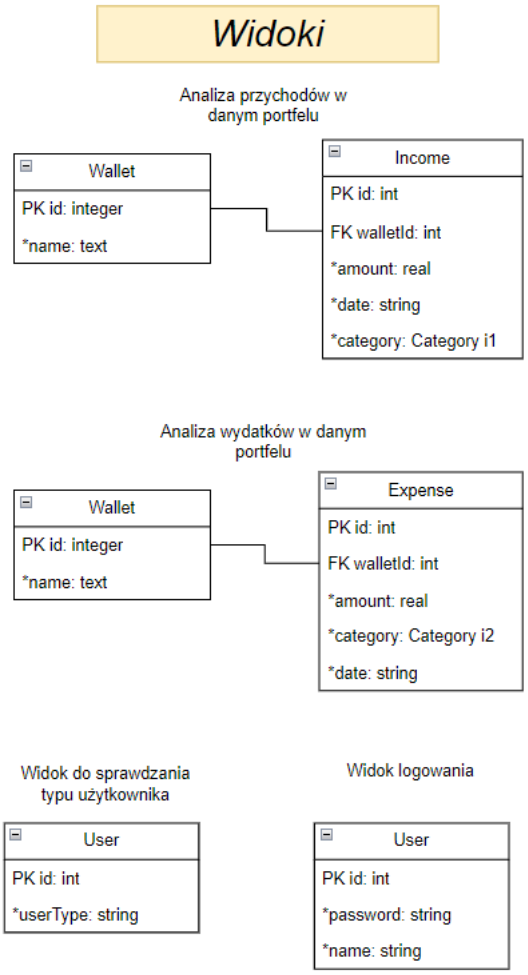
Widoczne na modelu fizycznym.
Ascendentne:

1. ix expense category
2. ix expense date
3. ix income category
4. ix income date
5. ix wishlistitem price

Descendentne:

1. ix expense amount
2. ix income amount

3.3 Widoki



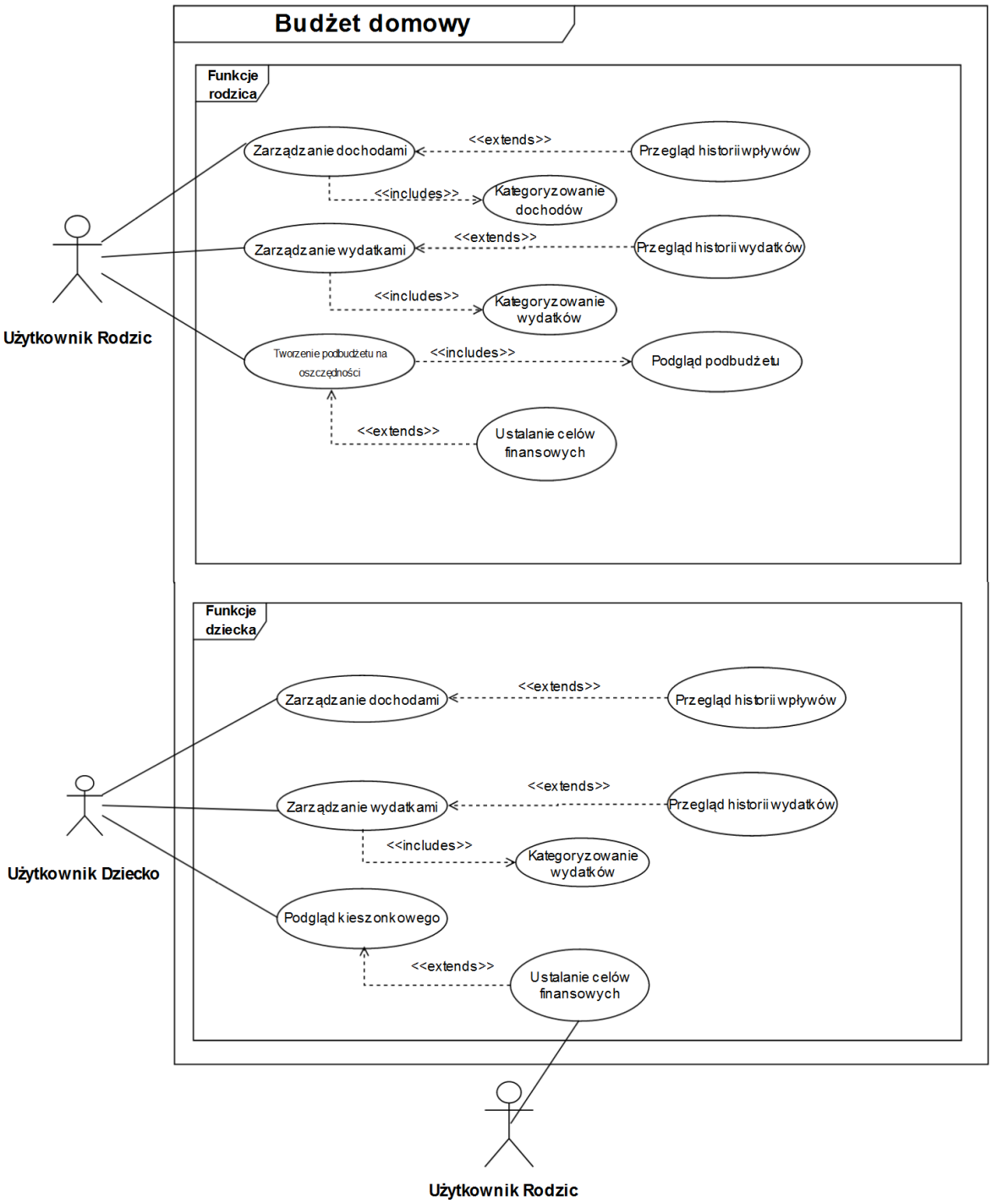
Rysunek 3: Caption

3.3.1

Wszyscy użytkownicy mają uprawnienia ALL.

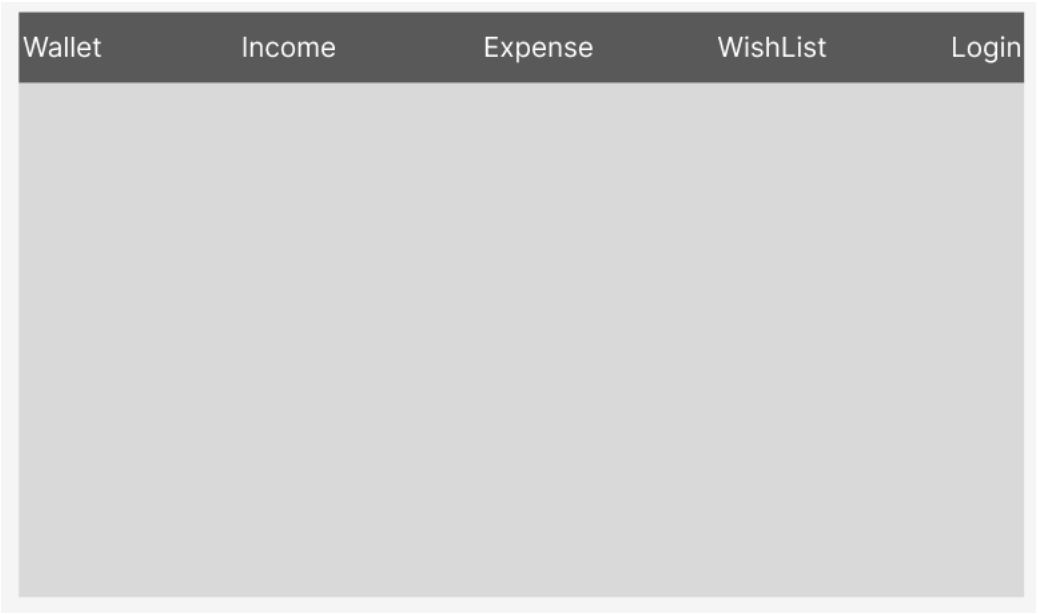
3.4 Projekt aplikacji użytkownika

3.4.1 Diagram przypadków użycia



Rysunek 4: Diagram przypadków użycia

3.4.2 Interfejs graficzny i struktura menu



Rysunek 5: Uproszczony widok strony głównej

Rysunek 6: Uproszczony widok portfela

Rysunek 7: Uproszczony widok strony wpłat

4 Implementacja systemu baz danych

4.1 Tworzenie tabel

Na podstawie modelu, napisaliśmy skrypty, których listingi pokazujemy poniżej. Skrypty zostały napisane w poddialekcie SQLite.

Listing 1: Listing skryptów

```
/*
Sekcja tworzenia tabel
*/

--- Tabela zawierajaca uzytkownikow i informacje o nich ---
CREATE TABLE IF NOT EXISTS "User" (
   "id"  INTEGER NOT NULL,
   "name"  TEXT NOT NULL,
   "password" TEXT NOT NULL,
   "userType" TEXT NOT NULL,
   PRIMARY KEY("id")
);

--- Tabela opisujaca kategorie wydatku lub przychodu ---
CREATE TABLE IF NOT EXISTS "Category" (
   "name"  TEXT NOT NULL,
   PRIMARY KEY("name")
);

--- Tabela zawierajaca wydatki i informacje o nich ---
CREATE TABLE IF NOT EXISTS "Expense" (
   "id"  INTEGER NOT NULL,
   "walletId" INTEGER NOT NULL,
   "userId" INTEGER NOT NULL,
   "amount" REAL NOT NULL,
   "date"  TEXT NOT NULL,
   "category" TEXT NOT NULL,
   "name"  TEXT,
   FOREIGN KEY("category") REFERENCES "Category"("name"),
   FOREIGN KEY("walletId") REFERENCES "Wallet"("id"),
   FOREIGN KEY("userId") REFERENCES "User"("id"),
   PRIMARY KEY("id")
);

--- Tabela zawierajaca przychody i informacje o nich ---
CREATE TABLE IF NOT EXISTS "Income" (
   "id"  INTEGER NOT NULL,
   "walletId" INTEGER NOT NULL,
   "userId" INTEGER NOT NULL,
   "amount" REAL NOT NULL,
   "date"  TEXT NOT NULL,
   "category" TEXT NOT NULL,
   "name"  TEXT,
   FOREIGN KEY("category") REFERENCES "Category"("name"),
   FOREIGN KEY("walletId") REFERENCES "Wallet"("id"),
   FOREIGN KEY("userId") REFERENCES "User"("id"),
   PRIMARY KEY("id")
);
```

```
--- Tabela zawierajaca portfele i informacje o nich ---
CREATE TABLE IF NOT EXISTS "Wallet" (
   "id"  INTEGER NOT NULL,
   "balance"  REAL NOT NULL,
   "type"  TEXT NOT NULL,
   "name"  TEXT NOT NULL,
   PRIMARY KEY("id")
);

--- Tabela posredniczaca opisujaca powiazanie miedzy uzytkownikiem a portfelem ---
CREATE TABLE IF NOT EXISTS "WalletOwnership" (
   "walletId" INTEGER NOT NULL,
   "userId" INTEGER NOT NULL,
   FOREIGN KEY("userId") REFERENCES "User"("id"),
   FOREIGN KEY("walletId") REFERENCES "Wallet"("id")
);

--- Tabela zawierajaca przedmioty z listy zyczen i informacje o nich---
CREATE TABLE IF NOT EXISTS "WishlistItem" (
   "id"  INTEGER NOT NULL,
   "userId" INTEGER NOT NULL,
   "name"  TEXT NOT NULL,
   "price" REAL NOT NULL,
   PRIMARY KEY("id"),
   FOREIGN KEY("userId") REFERENCES "User"("id")
);


/*
    Sekcja tworzenia indeksow
*/

--- Indeks na atrybucie "amount" w tabeli "Expense"---
CREATE INDEX IF NOT EXISTS "ix_expense_amount" ON "Expense" (
   "amount" DESC
);

--- Indeks na atrybucie "category" w tabeli "Expense"---
CREATE INDEX IF NOT EXISTS "ix_expense_category" ON "Expense" (
   "category"
);

--- Indeks na atrybucie "date" w tabeli "Expense"---
CREATE INDEX IF NOT EXISTS "ix_expense_date" ON "Expense" (
   "date"
);

--- Indeks na atrybucie "amount" w tabeli "Income"---
CREATE INDEX IF NOT EXISTS "ix_income_amount" ON "Income" (
   "amount" DESC
);

--- Indeks na atrybucie "category" w tabeli "Income"---
CREATE INDEX IF NOT EXISTS "ix_income_category" ON "Income" (
   "category"
);

--- Indeks na atrybucie "date" w tabeli "Income"---
CREATE INDEX IF NOT EXISTS "ix_income_date" ON "Income" (
```

```
    "date"
);

--- Indeks na atrybucie "price" w tabeli "WishlistItem"---
CREATE INDEX IF NOT EXISTS "ix_wishlistitem_price" ON "WishlistItem" (
    "price"
);

/*
    Sekcja tworzenia widokow
*/

--- Skrypt tworzacy widok usprawniajacy procedure logowania, wybierajacy z tabeli "User"
    atrybuty id, name, password ---
CREATE VIEW IF NOT EXISTS v_login AS
    SELECT
        id, name, password
    FROM
        User;

--- Skrypt tworzacy widok usprawniajacy procedure sprawdzania typu uzytkownika, wybierajacy z
    tabeli "User" atrybuty id, name, userType ---
CREATE VIEW IF NOT EXISTS v_user_type AS
    SELECT
        id, name, userType
    FROM
        User;

--- Skrypt tworzacy widok zlozony usprawniajacy procedure sprawdzania wplywow do danego portfela
    ---
CREATE VIEW IF NOT EXISTS v_wallet_incomes AS
    SELECT
        Wallet.id, Wallet.name, Income.id, Income.userId, Income.amount, Income.date,
            Income.category
    FROM
        Income
    JOIN
        Wallet ON Wallet.id = Income.walletId;

--- Skrypt tworzacy widok zlozony usprawniajacy procedure sprawdzania wydatkow z danego portfela
    ---
CREATE VIEW IF NOT EXISTS v_wallet_expenses AS
    SELECT
        Wallet.id, Wallet.name, Expense.id, Expense.userId, Expense.amount, Expense.date,
            Expense.category
    FROM
        Expense
    JOIN
        Wallet ON Wallet.id = Expense.walletId;

/* Ponizsza czesc kodu tworzy uzywane w programie wyzwalacze, ktore automatyzuja dzialanie bazy
    danych poprzez wyzwalanie stosownych polecen. */

-- Tworzy wyzwalacz aktualizujacy saldo portfela po otrzymaniu przychodu
CREATE TRIGGER IF NOT EXISTS TR_update_wallet
    AFTER INSERT ON Income
  BEGIN
    UPDATE Wallet SET balance = (balance + new.amount) WHERE id = new.walletId;
  END;
```

```
-- Tworzy wyzwalacz aktualizujacy saldo portfela po modyfikacji przychodu
CREATE TRIGGER IF NOT EXISTS TR_modify_income
    AFTER UPDATE ON Income
  BEGIN
    UPDATE Wallet SET balance = (balance - old.amount + new.amount) WHERE id = new.walletId;
  END;

-- Tworzy wyzwalacz aktualizujacy saldo portfela po usunieciu przychodu
CREATE TRIGGER IF NOT EXISTS TR_delete_income
    AFTER DELETE ON Income
  BEGIN
    UPDATE Wallet SET balance = (balance - old.amount) WHERE id = old.walletId;
  END;

 -- Tworzy wyzwalacz aktualizujacy saldo portfela po dodaniu wydatku
CREATE TRIGGER IF NOT EXISTS TR_add_expense
    AFTER INSERT ON Expense
  BEGIN
    UPDATE Wallet SET balance = (balance - new.amount) WHERE id = new.walletId;
  END;

 -- Tworzy wyzwalacz aktualizujacy saldo portfela po modyfikacji wydatku
CREATE TRIGGER IF NOT EXISTS TR_modify_expense
    AFTER UPDATE ON Expense
  BEGIN
    UPDATE Wallet SET balance = (balance + old.amount - new.amount) WHERE id = new.walletId;
  END;

 -- Tworzy wyzwalacz aktualizujacy saldo portfela po usunnieciu wydatku
CREATE TRIGGER IF NOT EXISTS TR_delete_expense
    AFTER DELETE ON Expense
  BEGIN
    UPDATE Wallet SET balance = (balance + old.amount) WHERE id = old.walletId;
  END;

-- Tworzy wyzwalacz usuwajacy polaczenie portfel-uzytkownik po usunieciu uzytkownika
CREATE TRIGGER IF NOT EXISTS TR_delete_user
    AFTER DELETE ON User
  BEGIN
   DELETE FROM WalletOwnership WHERE userId = old.id;
  END;

-- Tworzy wyzwalacz usuwajacy polaczenie portfel-uzytkownik po usunieciu portfela
CREATE TRIGGER IF NOT EXISTS TR_delete_wallet
    AFTER DELETE ON Wallet
  BEGIN
   DELETE FROM WalletOwnership WHERE walletId = old.id;
  END;
```

4.2 Implementacja mechanizmów przetwarzania danych

Brak możliwości utworzenia procedur składowanych, ze względu na brak wsparcia przez SQLite, planowane rozwiązanie to funkcje napisane bezpośrednio w pythonie.

4.3 Implementacja uprawnień i innych zabezpieczeń

Ze względu na SQLite, każdy użytkownik ma dostęp do całej bazy danych. Implementacja uprawnień musi być rozwiązana programowo.

4.4 Testowanie bazy danych na przykładowych danych

```
1   INSERT INTO Category (name) VALUES('Insurance');
Execution finished without errors.
Result: query executed successfully. Took 1ms, 1 rows affected
At line 1:
INSERT INTO Category (name) VALUES('Insurance');
```

Rysunek 8: Udana próba wprowadzenia danych do tabeli

```
1   INSERT INTO Category (name) VALUES('Car');
Execution finished with errors.
Result: UNIQUE constraint failed: Category.name
At line 1:
INSERT INTO Category (name) VALUES('Car');
```

Rysunek 9: Nieudana próba wprowadzenia danych do tabeli

```
1   INSERT INTO Wallet (balance,type,name)
2   VALUES
3      (NULL,'joint','Main budget')
Execution finished with errors.
Result: NOT NULL constraint failed: Wallet.balance
At line 1:
INSERT INTO Wallet (balance,type,name)
VALUES
   (NULL,'joint','Main budget')
```

Rysunek 10: Nieudana próba zaktualizowania danych

```
1   DELETE FROM WalletOwnership WHERE walletId = 2 AND userId = 2
Execution finished without errors.
Result: query executed successfully. Took 1ms, 1 rows affected
At line 1:
DELETE FROM WalletOwnership WHERE walletId = 2 AND userId = 2
```

Rysunek 11: Udana próba usunięcia wiersza tabeli

```
1   DELETE FROM WalletOwnership WHERE walletId = 0 AND userId = 0
Execution finished without errors.
Result: query executed successfully. Took 0ms, 0 rows affected
At line 1:
DELETE FROM WalletOwnership WHERE walletId = 0 AND userId = 0
```

Rysunek 12: Nieudana próba usunięcia wiersza tabeli

```
1   UPDATE WishlistItem SET price=1000 WHERE name = 'Trip'

Execution finished without errors.
Result: query executed successfully. Took 0ms, 1 rows affected
At line 1:
UPDATE WishlistItem SET price=1000 WHERE name = 'Trip'
```

Rysunek 13: Udana próba zaktualizowania danych

```
1   UPDATE WishlistItem SET price=100 WHERE name = 'nie_istniejace'

Execution finished without errors.
Result: query executed successfully. Took 0ms, 0 rows affected
At line 1:
UPDATE WishlistItem SET price=100 WHERE name = 'nie_istniejace'
```

Rysunek 14: Nieudana próba zaktualizowania danych

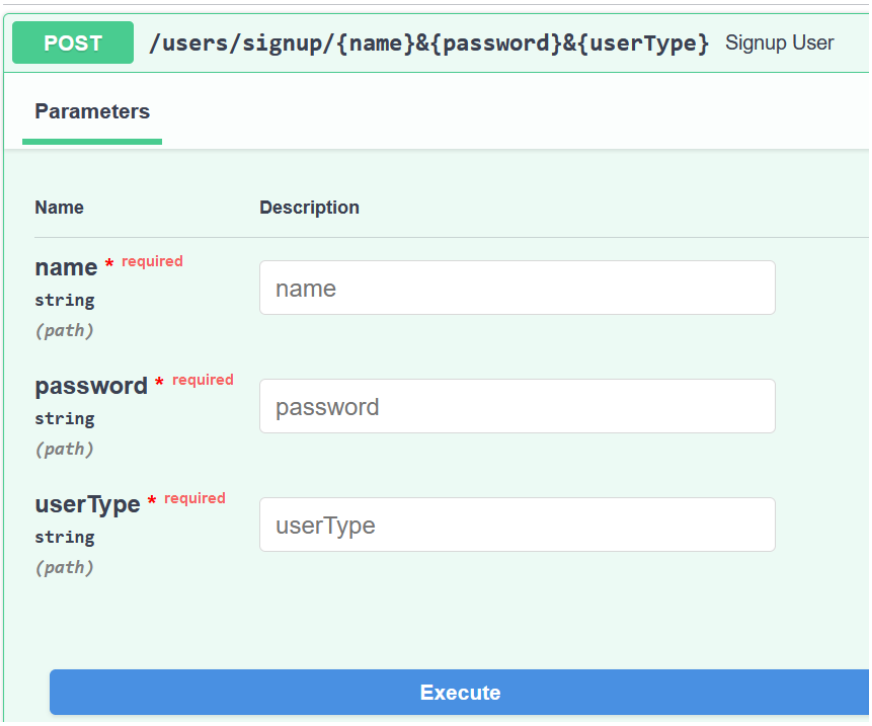
5 Implementacja i testy aplikacji

5.1 Implementacja

Implementacja aplikacji została wykonana w formie RESTapi, a dokumentacja jest dostępna przy użyciu SWAGGER'a.

5.2 Instrukcja użytkowania aplikacji

Aby zalogować się do aplikacji dostępowej należy wykorzystać endpoint rejestracji użytkownika podając nazwę, hasło i rodzaj użytkownika.

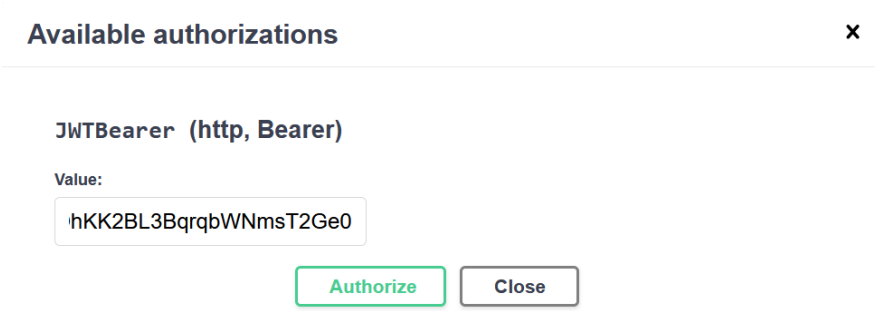


Rysunek 15: Endpoint rejestracji użytkownika

Jeżeli konto jest już utworzone należy użyć endpointu "login". Po zatwierdzeniu danych użytkownik otrzyma swój token.

Rysunek 16: Token użytkownika po zalogowaniu

Otrzymany token należy wkleić do okna ”Authorize” na górze strony.



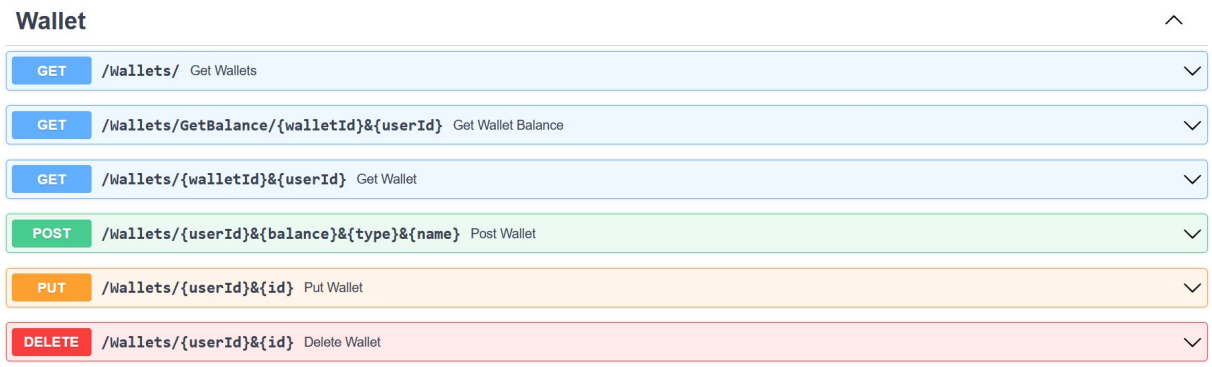
Rysunek 17: Autoryzacja za pomocą tokena

Po wykonaniu tych operacji reszta endpointów jest gotowa do użycia zgodnie z możliwościami dostępu do tabel danego użytkownika.

5.3 Omówienie wybranych rozwiązań programistycznych

5.3.1 Implementacja interfejsu dostępu do bazy danych

- Korzystanie z portfela
  Użytkownik może:
  1. Metodą GET /Wallets/ wyświetlić wszystkie portfele.
  2. Metodą GET /Wallets/GetBalance/ wyświetlić stan konta określonego portfela.
  3. Metodą GET /Wallets/ wyświetlić informacje o konkretnym portfelu.
  4. Metodą POST /Wallets/ dodać nowy portfel.
  5. Metodą PUT /Wallets/ zmodyfikować określony portfel.
  6. Metodą DELETE /Wallets/ usunąć określony portfel.

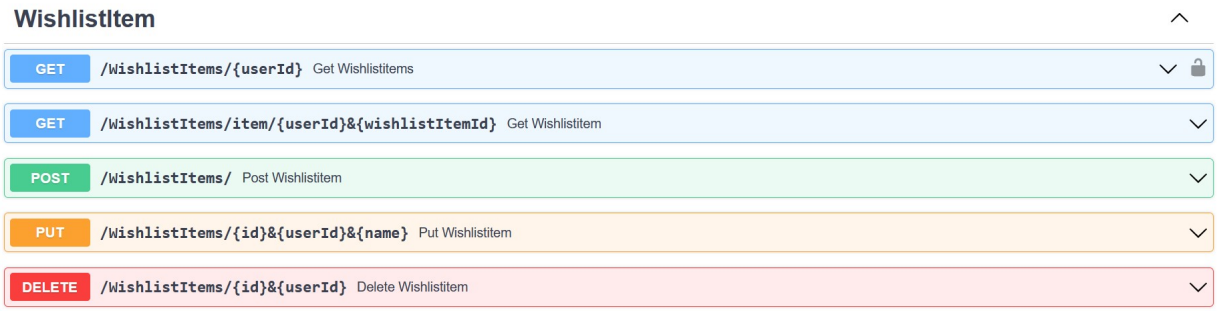


Rysunek 18

- Rejestracja i logowanie
  Użytkownik może:
  1. Metodą POST /users/signup/ zarejestrować się tworząc konto.
  2. Metodą POST /users/login/ zalogować się na swoje konto.

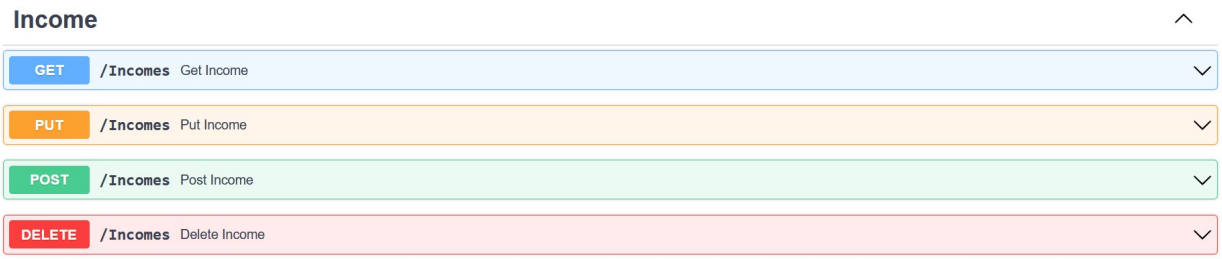
Rysunek 19

- Korzystanie z listy życzeń
  Użytkownik może:
  1. Metodą GET /wishlistItems/ wyświetlić swoją listę życzeń.
  2. Metodą GET /wishlistItems/item/ wyświetlić szczegóły elementu ze swojej listy życzeń.
  3. Metodą POST /wishlistItems/ dodać element do swojej listy życzeń.
  4. Metodą DELETE /wishlistItems/ usunąć element ze swojej listy życzeń.



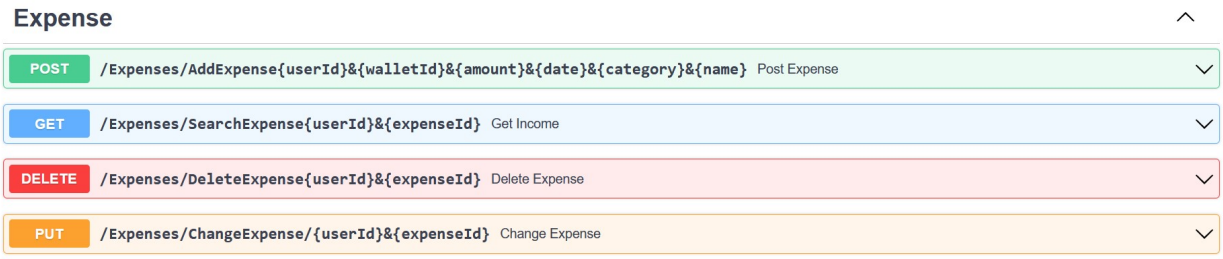
Rysunek 20

- Zarządzanie wpływami pieniężnymi
  Użytkownik może:
  1. Metodą GET /Incomes wyświetlić swój konkretny wpływ pieniężny.
  2. Metodą PUT /Incomes zmodyfikować swój konkretny wpływ pieniężny.
  3. Metodą POST /Incomes dodać swój wpływ pieniężny.
  4. Metodą DELETE /Incomes usunąć swój konkretny wpływ pieniężny.



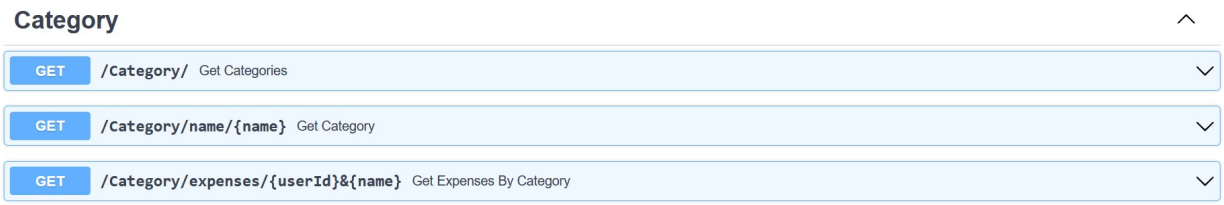
Rysunek 21

- Zarządzanie wydatkami
  Użytkownik może:
  1. Metodą POST /Expenses/AddExpense dodać swój konkretny wydatek pieniężny.
  2. Metodą GET /Expenses/SearchExpense wyświetlić swój wydatek pieniężny.
  3. Metodą DELETE /Expenses/DeleteExpense usunąć swój konkretny wydatek pieniężny.
  4. Metodą PUT /Expense/ChangeExpense zmodyfikować swój konkretny wydatek pieniężny.



Rysunek 22

- Korzystanie z kategorii
  Użytkownik może:
  1. Metodą GET /Category wyświetlić kategorie.
  2. Metodą GET /Category/name wyświetlić kategorię wpływu.
  3. Metodą GET /Category/expenses sprawdzić czy dana kategoria istnieje.

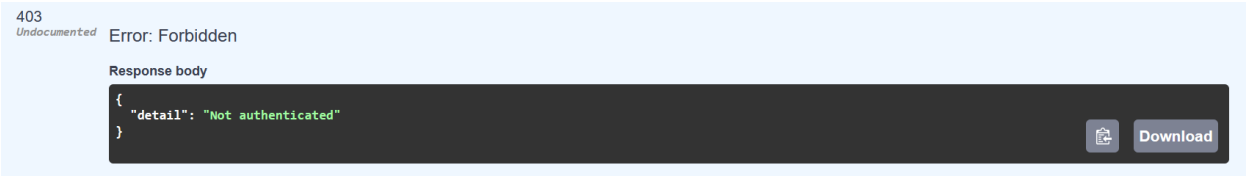


Rysunek 23

5.3.2 Implementacja mechanizmów bezpieczeństwa

Operacje są odrzucane dla użytkowników typu dziecko, jeżeli chcą zasięgnąć informacji o danych rodzica. Dostęp jest również ograniczony dla każdego użytkownika, jeżeli chce wykonać operacje na nie swoim portfelu. Jest to ustalane za pomocą tabeli "WalletOwnerShip"

Poniżej przedstawiona jest sytuacja dla odrzucania zapytania jeżeli, użytkownik jest dzieckiem i chce sprawdzić listę życzeń rodzica.



Rysunek 24: Brak autoryzacji

6 Podsumowanie i wnioski

- Podczas projektu udoskonaliliśmy swoje umiejętności pracy w grupie.
- Finalnie udało się stworzyć działającą aplikację bazodanową do kontroli budżetu, w formie Rest API.
- Tworzenie dokumentacji na bieżąco ułatwiało kolejne kroki przy realizacji projektu.