



**WYŻSZA SZKOŁA
INFORMATYKI I ZARZĄDZANIA**
z siedzibą w Rzeszowie

**Pracownia wytwarzania
oprogramowania 2**

Projekt i implementacja aplikacji mobilnej ExpensesTracker

**Iwona Lalak
w59803**

2019-06-10

1. Ogólny opis projektu

Podczas pracy nad tym projektem zaprojektowano i stworzono aplikację mobilną ExpensesTracker, której zadaniem jest śledzenie wydatków danego użytkownika. Aplikacja oferuje podgląd aktualnego salda konta, zarządzanie wpisami oraz kategoriami oraz przegląd zapisanych danych. Oparta jest o React Native, bibliotekę do tworzenia natywnych aplikacji mobilnych oraz o bazę danych SQLite.

Na cele projektu stworzono dwie grafiki: ikonę aplikacji oraz logo aplikacji pojawiające się podczas otwierania się aplikacji.



Ikona aplikacji

expenses
tracker



Logo aplikacji

2. Cel i zastosowanie projektu

Celem projektu było stworzenie aplikacji mobilnej korzystając z obecnie używanych i popularnych technologii, jakie są dostępne na rynku IT. Aplikacja miała ponadto być użyteczna oraz zaprojektowana zgodnie z wytycznymi UX/UI. Postanowiono stworzyć aplikację do zarządzania wydatkami, to jest notowania zmian finansowych oraz śledzenia aktualnego salda konta.

3. Funkcjonalności aplikacji

Dla użytkownika zostały przygotowane następujące funkcjonalności:

- podgląd aktualnego salda konta oraz ostatnich 7 wpisów
- podgląd wszystkich wpisów z podziałem na miesiące
- dodanie wpisu (łącznie z określeniem kwoty, typu, kategorii, daty oraz opcjonalnej notatki)
- edycja wpisu
- usunięcie wpisu
- przegląd wszystkich kategorii
- dodanie kategorii (łącznie z określeniem nazwy kategorii, jej ikony oraz koloru)
- edycję i usunięcie kategorii (prócz kategorii Nieznane)
- usunięcie za jednym przyciskiem wszystkich wpisów

4. Zaprojektowanie aplikacji

Jak wcześniej wspomniano, starano się użyć popularnych obecnie technologii wśród programistów, do stworzenia aplikacji mobilnej. Wybrano React Native ze względu na szybki i dość prosty sposób tworzenia aplikacji, zarówno na system Android jak i iOS. Posiada on również duże wsparcie, zarówno ze strony twórcy (Facebook), jak i od społeczności. Jako bazę wybrano SQLite ponieważ zapewnia podstawowe funkcjonalności, jakie były potrzebne do stworzenia projektu.

Przed stworzeniem widoków starano się zaprojektować je w tradycyjny sposób, to jest rozrysowując każdą scenę na kartce. Dzięki temu interfejs jest przemyślany i spójny.

5. Implementacja aplikacji

Na początku zaimplementowano sceny oraz routing aplikacji, następnie uzupełniano o elementy sceny, a następnie je programowano, działając z lokalnymi danymi. Gdy aplikacja była gotowa i przetestowano jej działanie na lokalnych danych, dodano SQLite oraz stworzono kontrolery umożliwiające kontakt aplikacji z bazą danych. Na samym końcu stworzono grafiki używając oprogramowania GIMP oraz dodano je do aplikacji, ustawiając ikonę oraz tak zwany splash screen - ekran startowy. Do implementacji aplikacji używano środowiska programistycznego Intelij Idea.

Poniżej przedstawiono i omówiono niektóre elementy aplikacji.

Baza danych

Baza danych składa się z dwóch tabel: categories oraz posts. Połączone są ze sobą kluczem obcym w relacji jeden do wielu. Poniżej przedstawiono właściwości kolumn. Klucze główne są autoinkrementowane.

Nazwa tabeli: categories				Nazwa tabeli: posts			
Nazwa	Typ danych	Primary Key	Foreign Key	Nazwa	Typ danych	Primary Key	Foreign Key
1 c_id	INTEGER			1 p_id	INTEGER		
2 c_name	VARCHAR (50)			2 p_date	DATE		
3 c_icon	VARCHAR (50)			3 p_note	VARCHAR (100)		
4 c_icongroup	VARCHAR (50)			4 p_amount	DOUBLE		
5 c_color	VARCHAR (10)			5 p_type	CHAR		
				6 category_id	INTEGER		

Połączenie aplikacji z bazą danych

W pliku Database.js stworzono singleton do otwierania połączenia z bazą danych. Wybrano singleton, ponieważ wystarczy jedna instancja obiektu aby wywoływać połączenia z różnych kontrolerów.

```
var SQLite = require('react-native-sqlite-storage')

export default class Database{
  static instance = null;

  Database(){}

  static getInstance(){
    if(this.instance === null){
      this.instance = SQLite.openDatabase({name: 'database.db', createFromLocation: '~database.db'})
      console.log('create instance')
    }
    console.log('return instance')
    return this.instance
  }
}
```

Funkcja wysyłająca zapytanie do bazy

W pliku kontrolera (poniżej PostsController.js) pobrano instancję bazy danych oraz wykonano zapytanie w transakcji zwracające odpowiedź pozytywną (resolve) bądź negatywną (reject). Użyto konstrukcji asynchronicznej operacji Promise.

```
import Database from "../Database";
var db = Database.getInstance();

export default {
  getLastPosts() {
    return new Promise((resolve, reject) => {
      let data = [];
      db.transaction((tx) => {
        tx.executeSql('SELECT * FROM posts INNER JOIN categories ON posts.category_id = categories.c_id ORDER BY posts.p_date DESC LIMIT 7', [], (tx, results) => {
          for (let i = 0; i < results.rows.length; i++) {
            data.push(results.rows.item(i))
          }
          resolve({
            data: data,
            msg: 'Get all posts successfully',
            ok: true,
          });
        }, (error) => {
          reject({
            data: [],
            msg: `${error.message}`,
            ok: false,
          });
        });
      });
    });
  },
};
```

W zależności od zdefiniowanego zapytania SQL i parametrów wejściowych, funkcja w kontrolerze mogła przybierać formy funkcji get, insert, update, delete bądź złożonych. Przykładem złożonej funkcji zawierającej dwie transakcje jest usuwanie kategorii. Usuwając kategorię należało stworzyć taki mechanizm, który by przepisał posty należące do usuwanej kategorii na kategorię Nieznane. Poniższy kod realizuje taką funkcjonalność.

```

deleteCategory(id) {

  return new Promise((resolve, reject) => {
    if (!id) {
      reject({
        msg: 'ID cannot be null',
        ok: false,
      })
    } else {
      db.transaction((txOuter) => {
        txOuter.executeSql(
          'UPDATE posts SET category_id=0 WHERE category_id=?',
          [id],
          (txOuter, resultOuter) => {

            console.log(resultOuter)
            db.transaction((tx) => {

              tx.executeSql(
                'DELETE FROM categories WHERE c_id=?',
                [id],
                (tx, result) => {
                  console.log(result)
                  if (result.rowsAffected > 0) {
                    resolve({
                      msg: 'Category deleted successfully',
                      ok: true
                    })
                  }
                  else {
                    reject({
                      msg: 'Category delete failed',
                      ok: false
                    })
                  }
                },
                (error) => {

                  console.log(error)

                  reject({
                    msg: `${error.message}`,
                    ok: false
                  })
                }
              )
            })
          }
        );
      });
    }
  });
}

```

```

    });
  },
  (error) => {

    console.log(error)

    reject({
      msg: `${error.message}`,
      ok: false
    })
  });
})
}
});
},

```

Wywołanie metody znajdującej się w kontrolerze

Wywołanie metody sprowadza się do oprogramowania interakcji użytkownika z danym elementem aplikacji. W przypadku usunięcia kategorii jest to zaprogramowanie na zdarzenie onPress przycisku usuwania kategorii, przywołania okna potwierdzającego operację, w przypadku zatwierdzenia wywołania metody w kontrolerze oraz zwrócenia odpowiedzi użytkownikowi. Jeśli odpowiedź jest twierdząca, następuje wyświetlenie komunikatu sukcesu oraz przejście do innego widoku aplikacji, w przypadku odpowiedzi negatywnej użytkownik zostaje poinformowany o tym, że nie można było wykonać ów akcji.

Wywołanie funkcji z confirmem dla użytkownika na zdarzenie onPress przycisku

```

<ButtonBottomPanelComponent
  onPressSave={() => this.onPressSave()}
  onPressDelete={() => this.onPressDelete()}
  onPressBack={() => Actions.pop()}
  editMode={this.props.editMode}
/>

```

Funkcja z confirmem dla użytkownika

```

onPressDelete() {
  Alert.alert(
    "Czy na pewno chcesz usunąć kategorię "+this.props.category.c_name+"?",
    "Wpisy należące do tej kategorii zostaną przypisane do kategorii Nieznane",
    [
      {text:"NIE", style:'cancel'},
      {text:"TAK", onPress: ()=>this.deleteCategory()}
    ]
  )
}

```


Funkcja wywołująca metodę kontrolera

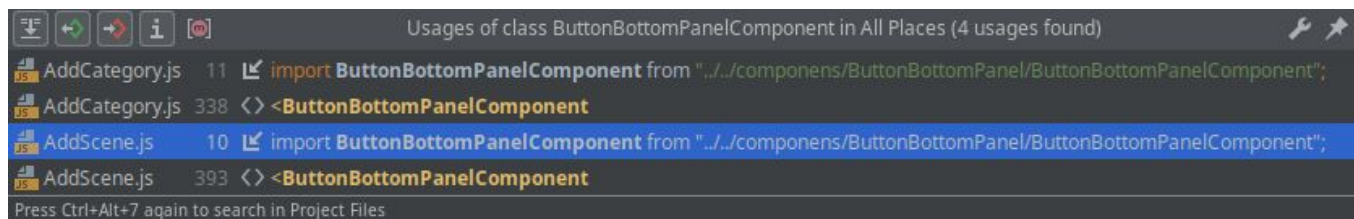
```
deleteCategory(){
  CategoriesController.deleteCategory(this.props.category.c_id)
    .then(
      function (response) {
        if(response.ok){
          showMessage({
            message: "Pomyślnie usunięto kategorie",
            type: "success",
            position: "center",
            icon: 'success'
          });

          setTimeout(function () {
            Actions.push("CategoriesScene")
          },1000)
        }
      }.bind(this))
    .catch(function (err) {
      console.log(err)

      showMessage({
        message: "Wystąpił błąd - nie można usunąć kategorii",
        type: "danger",
        position: "center",
        icon: 'danger'
      });
    }.bind(this))
}
```

Komponenty w React Native

Wcześniej przytoczono komponent `ButtonBottomPanelComponent.js` zawierający zestaw dwóch przycisków (jednego typu “usuń” oraz drugiego typu “zapisz”). React zarówno w wersji Native oraz webowej posiada możliwość tworzenia i wykorzystywania wielokrotnie komponentów, przez co znacznie wzrasta zarówno czytelność kodu jak i szybkość tworzenia aplikacji. Tworzenie aplikacji w React JS i React Native można porównać do budowania jej z małych klocków, z czego dany klocek może być jednocześnie używany w wielu miejscach. Takim przykładem jest powyżej wspomniany komponent: używany jest w widoku formularza danej kategorii oraz formularza danego wpisu, w przypadku wywołania na rekordzie opcji edycji.



Utils. czytelność kodu

Podczas tworzenia aplikacji starano się opierać na zasadzie czystego kodu. Kod jest pisany w języku angielskim, nazwy zmiennych i funkcji odzwierciedlają rzeczywiste przeznaczenie danego elementu. Ponadto wykorzystano możliwość tworzenia tak zwanych utils, czyli zestawu funkcji, zmiennych definiowanych globalnie i gotowych do użycia tak, aby nie powielać kodu. Przykładem jest zdefiniowanie funkcji formatującej liczbę do formatu pieniężnego, to jest 00.00,00 zł . Jest ona wywoływana podczas wyświetlania kwoty wpisu bądź dla sald: aktualnego oraz miesięcznego.

```
import acc from "accounting-js";

export default {
  format(amount){
    return acc.formatMoney(amount,{ symbol: "zł", precision: 2, thousand: ".", decimal: ",", format: "%v %s"})
  }
}
```

Zdefiniowano również główne style (kolory) używane w aplikacji. Dzięki temu wystarczy zmienić kolor w jednym miejscu, zamiast w kilkunastu bądź kilkudziesięciu.

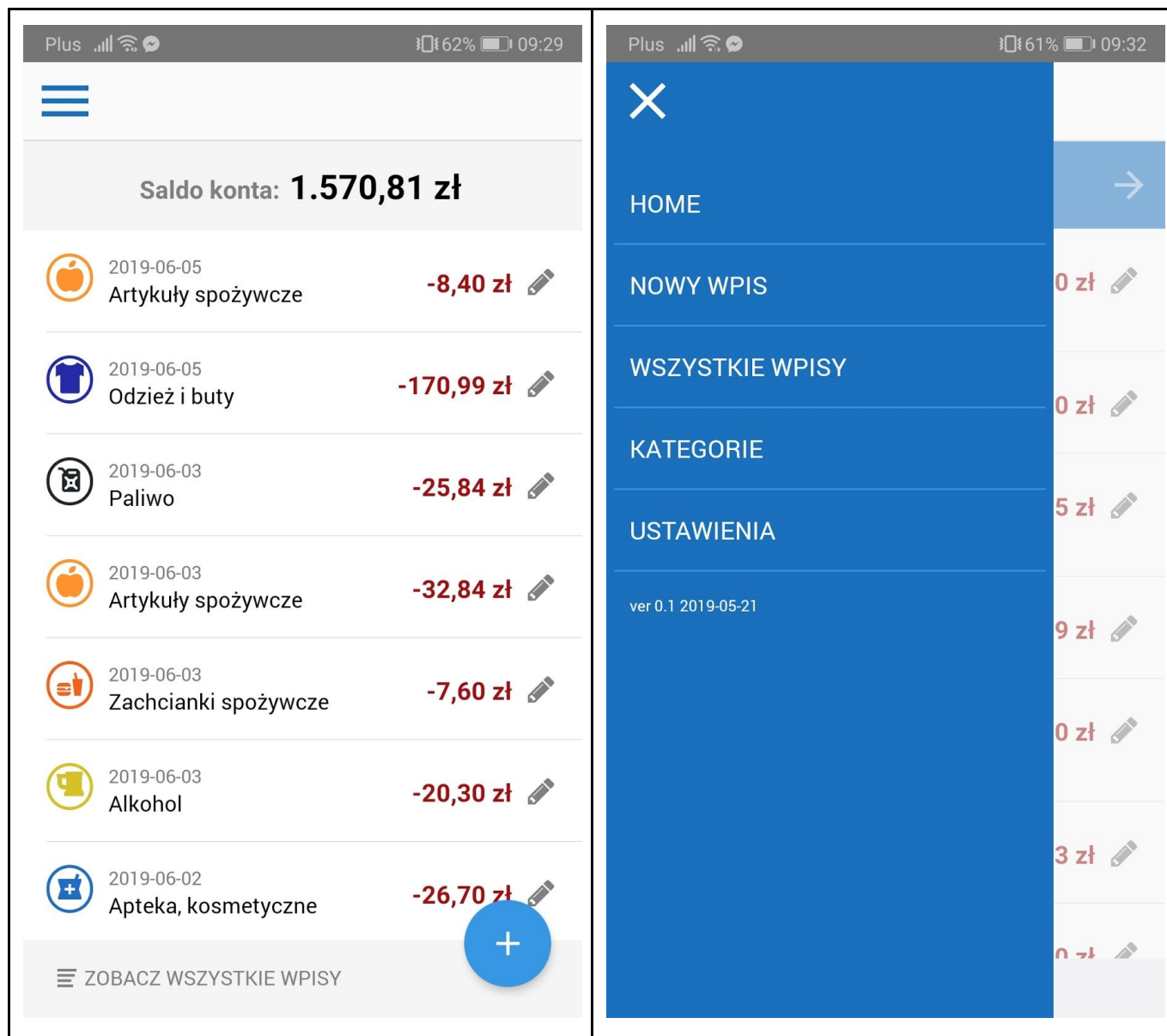
```
1 export default {
2   background_light_color: "whitesmoke",
3   main_color_lighter: '#3797e4',
4   main_color: '#1b70be',
5   main_color_darker: '#195ca3',
6
7   add_color_float_btn: '#3797e4',
8
9   green_lighter: "#65c231",
10  green_medium: "#4b892a",
11
12  red_lighter: "#bf363c",
13  red_medium: "#9e1115",
14 }
```

6. Przewodnik po aplikacji

Poniżej przedstawiono użytkowanie aplikacji omawiając każdy widok.

Home

Po uruchomieniu aplikacji domyślnie pokaże się widok Home zawierający aktualne saldo konta (czyli sumę wszystkich wydatków i przychodów) oraz ostatnie 7 wpisów. W Home można edytować wpis, przejść do widoku wszystkich wpisów oraz przejść do widoku dodawania nowego wpisu. Obok zrzutu widoku Home przedstawiono również sidebar zawierający nawigację po aplikacji.



W przypadku, jeśli aplikacja nie posiada żadnych wpisów (na przykład uruchamiana jest pierwszy raz), wyświetla się informacja o braku wpisów oraz odpowiedź wskazująca na dodanie pierwszego wpisu.

Nowy wpis

Po przyciśnięciu na widoku Home przycisku dodawania wpisu bądź wybrania z sidebara opcji Nowy wpis pojawia się formularz dodawania wpisu. Wpis może mieć ustaloną kwotę, typ (czy jest to przychód czy wydatek), kategorię (wybór z listy zdefiniowanych w aplikacji), datę oraz opcjonalnie notatkę.

Plus 62% 09:30

DODAJ NOWY WPIS

2.500,00 zł

WYDATEK

PRZYCHÓD

Przychód

Data wpisu: 05 czerwiec 2019

Notatka:

Wypłata

POWRÓT

+ DODAJ

Plus 62% 09:30

EDYTUJ WPIS

25,84 zł

WYDATEK

PRZYCHÓD

Paliwo

Data wpisu: 03 czerwiec 2019

Notatka:

4,89 litra

USUŃ

✓ ZAPISZ

Po dodaniu wpisu można go zobaczyć w widoku Home (jeśli mieści się w ciągu ostatnich 7 wpisów, sortując po dacie wpisu), bądź w widoku wszystkich wpisów.

Wszystkie wpisy

Widok ten zawiera wszystkie wpisy, które są podzielone i wyświetlane wedle miesiąca pobranego z daty wpisu. Używając nawigacji po miesiącach można łatwo przeglądać archiwalne wpisy oraz zobaczyć saldo miesięczne czyli zsumowaną kwotę wszystkich przychodów i wydatków w danym miesiącu.

Plus 61% 09:30	Plus 61% 09:31
CZERWIEC 2019	MAJ 2019
<div>2019-06-05</div> <div>Artykuły spożywcze</div> <div>-8,40 zł</div>	<div>2019-05-31</div> <div>Rozrywka Netflix</div> <div>-13,00 zł</div>
<div>2019-06-05</div> <div>Odzież i buty</div> <div>-170,99 zł</div>	<div>2019-05-30</div> <div>Transport</div> <div>-3,00 zł</div>
<div>2019-06-03</div> <div>Paliwo 4,89 litra</div> <div>-25,84 zł</div>	<div>2019-05-30</div> <div>Artykuły spożywcze Biedronka</div> <div>-25,95 zł</div>
<div>2019-06-03</div> <div>Artykuły spożywcze</div> <div>-32,84 zł</div>	<div>2019-05-30</div> <div>Zachcianki spożywcze</div> <div>-6,99 zł</div>
<div>2019-06-03</div> <div>Zachcianki spożywcze Chrupki i sok</div> <div>-7,60 zł</div>	<div>2019-05-29</div> <div>Restauracja, bar, jedzenie Kabsik</div> <div>-12,00 zł</div>
<div>2019-06-03</div> <div>Alkohol</div> <div>-20,30 zł</div>	<div>2019-05-28</div> <div>Artykuły spożywcze</div> <div>-6,23 zł</div>
<div>2019-06-02</div> <div>Apteka, kosmetyczne</div> <div>-26,70 zł</div>	<div>2019-05-28</div> <div></div> <div>-50,00 zł</div>
<div>ZMIANA: 1.167,98 zł</div>	<div>ZMIANA: 402,83 zł</div>

Edycja, usunięcie wpisu

Klikając na pozycję bądź ołówek można przejść do widoku zarządzania istniejącym wpisem, który jest w zasadzie zmodyfikowanym formularzem dodawania wpisu. Aplikacja pozwala na edytowanie wpisu lub jego usunięcie, co jest dodatkowo potwierdzane przez użytkownika w wyskakującym alercie. W przypadku chęci powrotu należy skorzystać z domyślnego przycisku powrotu w Androidzie.


Plus 62% 09:30

EDYTUJ WPIS

25,84 zł

WYDATEK


PRZYCHÓD


 Paliwo

Data wpisu: 03 czerwiec 2019

Notatka:

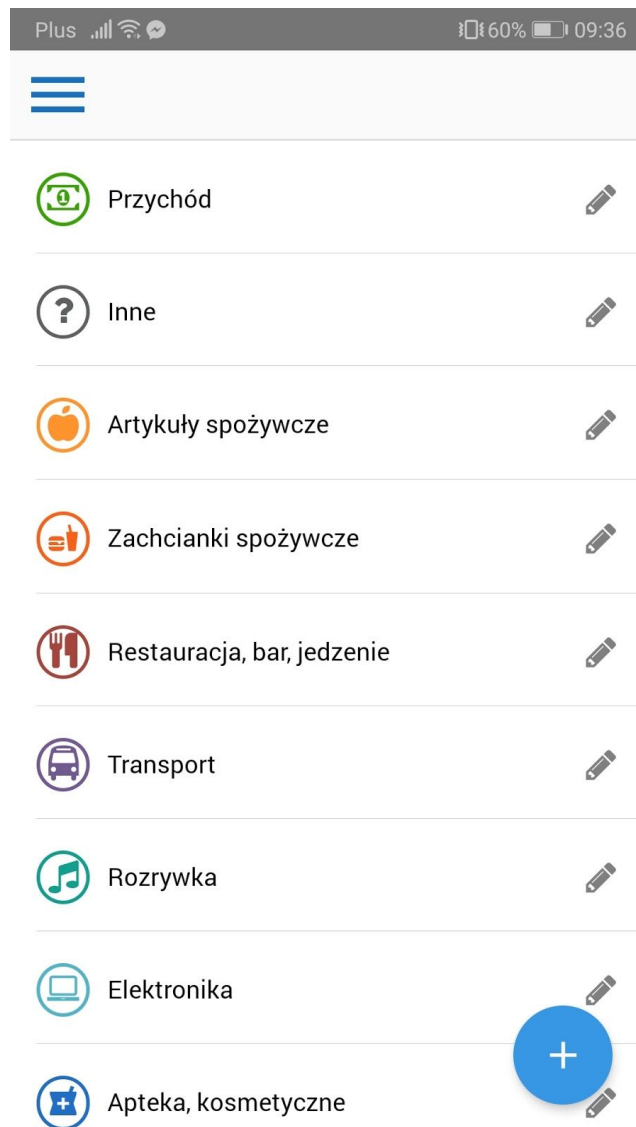
4,89 litra

 USUŃ

 ZAPISZ

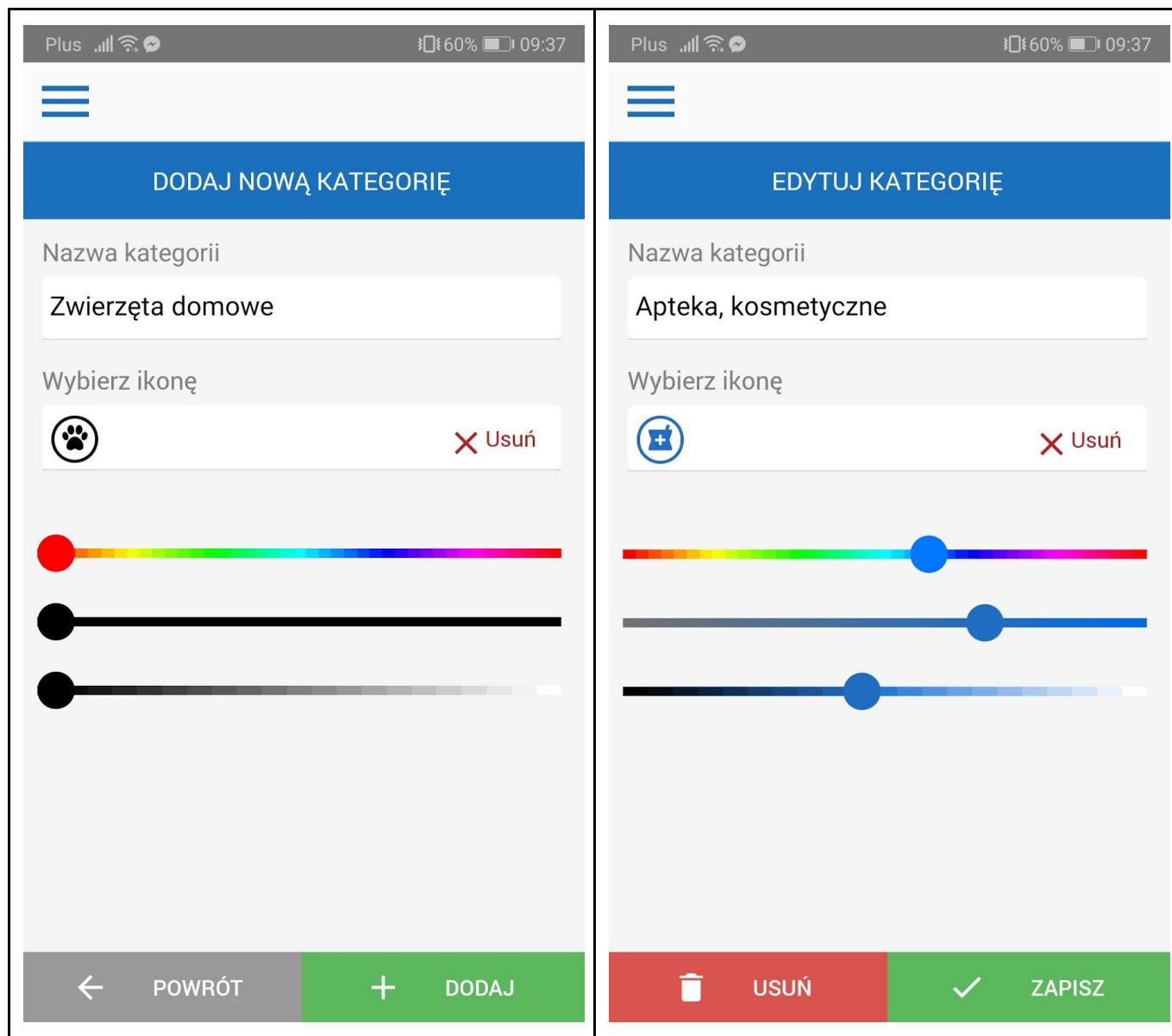
Przegląd kategorii

Aplikacja umożliwia zarządzanie kategoriami w celu zwiększenia wygody korzystania.



Dodawanie, usuwanie, edycja kategorii

Podobnie jak wpisy, kategorie można dodawać, edytować oraz usuwać. Podczas usuwania kategorii aplikacja przypisuje do kategorii Nieznane wszystkie wpisy które posiadały powiązanie do kategorii usuwanej.



Usunięcie wszystkich wpisów

Aplikacja udostępnia opcję usunięcia wszystkich wpisów za pośrednictwem jednego przycisku. Po dodatkowym potwierdzeniu w wyskakującym alercie akcji, baza danych jest czyszczona z wpisów, a saldo naturalnie zeruje się.



WYCZYŚĆ WSZYSTKIE WPISY