

Dokumentacja

1. Wprowadzenie

Aplikacja **Finance Tracker** jest prostym, lecz funkcjonalnym narzędziem desktopowym służącym do zarządzania finansami osobistymi. Umożliwia rejestrowanie wpływów i wydatków, wyświetlanie bieżącego salda oraz generowanie statystyk w formie wykresów. Projekt powstał w celu zaprezentowania umiejętności tworzenia aplikacji w technologii JavaFX oraz wykorzystania lokalnej bazy danych SQLite.

2. Zakres i cele projektu

Celem aplikacji jest:

- Ułatwienie codziennego monitorowania przychodów i wydatków
- Zapewnienie szybkiego podglądu salda i rozkładu kosztów
- Możliwość definiowania i kontroli limitów wydatków w wybranym okresie (dziennym, tygodniowym, miesięcznym)
- Prosty, intuicyjny interfejs oparty na JavaFX

Projekt skupia się na następujących aspektach:

1. **Rejestracja transakcji** – dodawanie opisu, kwoty (dodatnie = przychód, ujemne = wydatek), daty oraz kategorii.
 2. **Przechowywanie danych** – użycie lokalnej bazy SQLite do trwałego zapisu transakcji oraz limitów.
 3. **Prezentacja danych** – tabela transakcji, karty podsumowania, filtracja listy, wykresy kołowe.
 4. **Obsługa limitów** – zapobieganie dodawaniu transakcji przekraczających ustalony limit.
-

3. Technologie i narzędzia

Technologia / Narzędzie	Wersja / Uwagi
JDK	Java 17
JavaFX	17 (moduły javafx-controls, javafx-fxml)
SQLite	JDBC (org.sqlite.JDBC)
FXML	Apache FXML do opisu UI
Scene Builder (opcjonalnie)	Do projektowania plików FXML
IDE	IntelliJ IDEA / Eclipse / NetBeans
System kontroli wersji	Git (repozytorium lokalne / GitHub)

- **JavaFX** odpowiada za warstwę prezentacji (UI).
 - **FXML** (w pliku `main.fxml`) definiuje strukturę interfejsu.
 - **SQLite** (via JDBC) jest lokalną bazą danych, do której zapisywane są transakcje i limity.
 - **CSS** (`style.css`) stylizuje aplikację zgodnie z ciemną kolorystyką.
-

4. Architektura aplikacji

4.1 Warstwy aplikacji

Aplikacja składa się z dwóch głównych warstw:

1. Warstwa prezentacji (UI)

- Plik FXML (`main.fxml`), kontroler JavaFX (`MainController.java`), styl CSS (`style.css`).
- Odpowiedzialna za wyświetlanie danych, obsługę zdarzeń kliknięć, aktualizację widoków.

2. Warstwa dostępu do danych (DAO / Database)

- Klasa `Database.java` (funkcje: `connect()`, `isLimitExceeded()`, `setLimit()`).
- Klasa `Transaction.java` (model danych dla pojedynczej transakcji).
- Użycie JDBC do komunikacji z lokalnym plikiem bazy SQLite.

Schemat powiązań:

```
pgsql
KopiujEdytuj
Main (start aplikacji)
└─> MainController (obsługa UI)
    └─> Transaction (model)
        └─> Database (DAO: tworzenie tabel, zapytania, limity)
            └─> view/main.fxml + style.css
```

4.2 Schemat bazy danych

W projekcie używana jest jedna tabela:

```
sql
KopiujEdytuj
CREATE TABLE IF NOT EXISTS transactions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    amount REAL NOT NULL,
    date TEXT NOT NULL,
    category TEXT NOT NULL
);
```

- **id** – klucz główny (AUTO_INCREMENT)
- **title** – opis transakcji (tekst)
- **amount** – kwota (pozytywna = przychód, negatywna = wydatek)
- **date** – data transakcji w formacie YYYY-MM-DD
- **category** – kategoria (tekst)

Dodatkowo:

- Wartości limitu nie są przechowywane w osobnej tabeli. Zamiast tego klasa `Database` wykorzystuje osobne mechanizmy (np. w pliku konfiguracyjnym lub inna tabela; patrz implementacja w `Database.java`).




Uwaga: jeśli w przyszłości pojawi się potrzeba przechowywania limitów i historii limitów, można dodać tabelę:

```
sql
KopiujeDytuj
CREATE TABLE IF NOT EXISTS limits (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    amount REAL NOT NULL,
    period TEXT NOT NULL -- np. "Daily", "Weekly", "Monthly"
);
```

W obecnej wersji limit jest przechowywany w jednym wierszu lub np. w osobnym pliku.

5. Opis funkcjonalności

5.1 Główny widok transakcji

- Po uruchomieniu aplikacji użytkownik widzi:
 1. **Pasek górny:** nazwa aplikacji (“📊 Finance Tracker”) oraz aktualna data.
 2. **Pasek boczny (sidebar):** przyciski-nawigacja:
 - [] Powrót do widoku głównego (lista transakcji).
 - [] Przejście do statystyk.
 - [] Przejście do ustawień limitu wydatków.
 3. **Główna sekcja:**
 - Formularz dodawania nowej transakcji (opis, kwota, kategoria).
 - Karty podsumowania (“All Accounts”, “Income This Month”, “Expenses This Month”).
 - Tabela transakcji (Title, Amount, Date, Category, Delete).
 - Pasek filtrów (opis, data + przyciski “Filter”, “Clear”).

Widok główny umożliwia szybki podgląd historii finansów i natychmiastowe dodanie kolejnej pozycji.

5.2 Dodawanie i usuwanie transakcji

1. Dodawanie transakcji:

- Użytkownik wypełnia pola:
 - **Description** (opis, np. “Zakupy RTV”).
 - **Amount** (np. -250.00 dla wydatku lub 500.00 dla przychodu).
 - **Category** (lista rozwijana z opcjami: “Jedzenie”, “Transport”, “Rozrywka”, “Zdrowie”, “Inne”).
- Po kliknięciu przycisku **“Add”** wykonywana jest metoda `handleAddTransaction()`:
 - Sprawdzenie poprawności wprowadzonych danych (puste pola, niepoprawna kwota).
 - Pobranie bieżącej daty `LocalDate.now()`.
 - Utworzenie obiektu `Transaction`.
 - Sprawdzenie limitu (jeśli wydatki w danym okresie przekraczają ustawiony limit → wyświetlenie alertu i przerwanie).
 - Zapis transakcji w tabeli SQL → odświeżenie tabeli w UI → aktualizacja podsumowań.
 - Wyczyszczenie formularza.

2. Usuwanie transakcji:

- W kolumnie **“Delete”** każda transakcja posiada przycisk kosza (🗑).
- Kliknięcie go wywołuje `deleteTransaction(transaction)` → usuwa rekord z bazy → odświeża tabelę i podsumowania.

5.3 Filtrowanie transakcji

- W dolnej części widoku głównego znajduje się **pasek filtrów**:
 - Pole tekstowe **“Description filter”**: wpisanie fragmentu opisu wyszukuje pasujące transakcje (on-the-fly, bez konieczności zatwierdzania).
 - Pole **DatePicker** do wyboru daty.
 - Przycisk **“Filter”**: wywołuje `onFilter()`, który na podstawie wpisanego tekstu i/lub wybranej daty tworzy listę filtrowanych transakcji i wyświetla je w tabeli.

- Przycisk “**Clear**”: wywołuje `onClear()`, czyści pola filtra i przywraca pełną listę transakcji.

Zachowanie filtrowania:

- Filtrowanie po opisie jest nieczułe na wielkość liter.
 - Jeśli żadne pole filtra nie jest wypełnione → wyświetlane są wszystkie transakcje.
-

5.4 Podsumowania finansowe


- Bezpośrednio nad tabelą widoczne są **Karty podsumowania**:
 1. **All Accounts (PLN)** – całkowity bilans: suma wszystkich przychodów minus suma wszystkich wydatków.
 2. **Income This Month** – suma wszystkich transakcji z wartością dodatnią w bieżącym miesiącu.
 3. **Expenses This Month** – suma wszystkich transakcji z wartością ujemną (wartość bezwzględna) w bieżącym miesiącu.
 - Metoda `updateSummaryLabels()` liczy iteracyjnie wartości na podstawie listy `transactions` (wszystkie rekordy załadowane z bazy) i ustawia tekst w odpowiednich etykietach.
-

5.5 Ustawienia limitów

- Użytkownik może określić maksymalny dopuszczalny poziom wydatków w wybranym okresie (**Daily, Weekly, Monthly**).
- Po kliknięciu ikony ⚙ w pasku bocznym zostaje wywołana metoda `showLimitSettings()`, która ukrywa główny widok i wyświetla panel limitów.
- W panelu limitów:
 1. Pole **Amount (PLN)** – wprowadzenie kwoty limitu (typ `Double`).
 2. ComboBox **limitPeriodComboBox** – wybór okresu: “Daily”, “Weekly” lub “Monthly”.
 3. Przycisk “**Save limit**” → `handleSaveLimit()` → próba konwersji tekstu do `double` i zapis do bazy/metody `Database.setLimit(amount, period)`.
 4. Przycisk “**Back**” → `handleReturnToMain()` → powraca do widoku głównego (ukrywa `limitSettingsBox`).
- W metodzie dodawania transakcji (`handleAddTransaction`) przed wstawieniem nowego wiersza sprawdzane jest: `Database.isLimitExceeded(t)` → jeśli limit został przekroczony, wyświetlany jest `Alert` i dodanie jest blokowane.

Uwaga: szczegóły implementacji przechowywania limitów (np. w osobnej tabeli lub w pliku konfiguracji) znajdują się w klasie `Database.java`.

5.6 Statystyki (wykresy i dane)

- Po kliknięciu ikony  w pasku bocznym wywoływana jest metoda `showStats()`.
 - Widok `statsBox` (ukrywany domyślnie) wyświetla:
 1. Nagłówek **“Statistics”** oraz etykietę `statsBalance` (bilans: suma przychodów minus sumy wydatków).
 2. Panel szybkich statystyk:
 - **Stats Income** – całkowita suma przychodów.
 - **Stats Expenses** – całkowita suma wydatków.
 - **Stats Count** – liczba wszystkich transakcji.
 3. **PieChart** – wykres kołowy przedstawiający strukturę wydatków według kategorii. Tylko wydatki (wartości ujemne w bazie).
 4. `categoryStatsBox` – lista etykiet w formacie “Kategoria: wartość PLN” obok wykresu.
 5. Przycisk **“Back”** → `handleReturnToMain()`.
 - Logika: w `showStats()`:
 1. `getTransactions()` pobiera wszystkie rekordy.
 2. Iteracja po liście: sumowanie przychodów i wydatków, budowanie mapy `categoryMap` (kategoria → łączna wartość wydatków).
 3. Obliczenie bilansu.
 4. Wypełnienie odpowiednich etykiet (tekst).
 5. Utworzenie listy `ObservableList<PieChart.Data>` na podstawie `categoryMap`.
 6. Ustawienie danych w `expensePieChart.setData(pieData)`.
 7. Dodanie opisów kategorii w `categoryStatsBox`.
-

6. Instrukcja instalacji i uruchomienia

6.1 Wymagania systemowe

- System operacyjny: Windows / macOS / Linux (Java 17 powinna być poprawnie zainstalowana).
- Java Development Kit (JDK 17).
- Dostęp do lokalnego systemu plików (tworzenie pliku SQLite).
- (Opcjonalnie) Scene Builder do podglądu i edycji plików FXML.

6.2 Konfiguracja projektu

1. Pobranie kodu źródłowego

- Rozpakuj archiwum `FinanceTracker.zip` w wybranej lokalizacji.

2. Import do IDE

- Uruchom IntelliJ IDEA (lub Eclipse/NetBeans).
- Wybierz “Import Project” → wskaż folder `FinanceTracker`.
- Projekt powinien być typu Maven albo Gradle (w zależności od ustawień). Jeśli to zwykły projekt JavaFX: utwórz nowy projekt i skopiuj foldery `src/main/java` i `src/main/resources`.

3. Dodanie bibliotek

- Upewnij się, że w ustawieniach projektu na classpathie są:
 - JavaFX SDK 17 (m.in. `javafx-controls`, `javafx-fxml`).
 - Sterownik SQLite JDBC (np. `sqlite-jdbc-3.x.x.jar`).
- Jeśli projekt używa pliku `pom.xml` (Maven): w sekcji `<dependencies>` dodaj:

```
xml
KopiujEdytuj
<dependency>
  <groupId>org.xerial</groupId>
  <artifactId>sqlite-jdbc</artifactId>
  <version>3.41.2.1</version>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-controls</artifactId>
  <version>17.0.2</version>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-fxml</artifactId>
  <version>17.0.2</version>
</dependency>
```

- Jeżeli nie używasz systemu budowania, ręcznie dodaj JAR-y do ścieżki kompilacji.

6.3 Uruchomienie aplikacji

1. W IDE otwórz klasę `Main.java` (lub odpowiednią klasę startową w pakiecie `com.financetracker.financetracker`).
 2. Upewnij się, że VM options zawierają ścieżki do modułów JavaFX, np. (IntelliJ):

```
swift
KopiujEdytuj
--module-path /ścieżka/do/javafx-sdk-17/lib --add-modules
javafx.controls,javafx.fxml
```
 3. Uruchom program (Run As → Java Application).
 4. Aplikacja powinna utworzyć lokalną bazę SQLite (plik `finance_tracker.db` w katalogu roboczym).
 5. Po pierwszym uruchomieniu wszystkie tabele zostaną automatycznie utworzone.
-

7. Struktura plików i pakietów

```
css
KopiujEdytuj
FinanceTracker/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── com/
│   │   │   │   └── financetracker/
│   │   │   │       └── financetracker/
│   │   │   │           ├── Main.java
│   │   │   │           ├── MainController.java
│   │   │   │           ├── Transaction.java
│   │   │   │           └── Database.java
│   │   └── resources/
│   │       ├── main.fxml
│   │       └── style.css
├── README.md (opcjonalnie)
└── FinanceTracker.zip (archiwum projektowe)
```

- **Main.java** – klasa uruchamiająca aplikację (metoda `start(Stage primaryStage)`).
- **MainController.java** – kontroler JavaFX, powiązany z `main.fxml`.
- **Transaction.java** – model danych dla transakcji (pola: id, title, amount, date, category).
- **Database.java** – klasa obsługi bazy danych (połączenie, tworzenie tabel, zapytania, limity).
- **main.fxml** – definicja struktury UI.
- **style.css** – plik stylów CSS.

8. Opis najważniejszych klas i modułów

8.1 MainController

- **Odpowiedzialność:** logika widoku głównego, obsługa zdarzeń UI, komunikacja z warstwą DAO.
- **Główne metody:**
 - `initialize()` – konfiguracja tabeli, połączenie z bazą, załadowanie danych.
 - `handleAddTransaction(ActionEvent)` – walidacja, sprawdzenie limitu, zapis do bazy, odświeżenie.
 - `addDeleteButtonToTable()` – dodanie przycisku “🗑️” w każdym wierszu tabeli.
 - `updateSummaryLabels()` – obliczenie bilansu, przychodów i wydatków w bieżącym miesiącu.
 - `onFilter(ActionEvent) / onClear(ActionEvent)` – filtrowanie i czyszczenie filtrów.
 - `showLimitSettings() / handleSaveLimit()` – obsługa panelu limitów.
 - `showStats()` – wyświetlanie statystyk (obliczenie sum, przygotowanie wykresu).
 - `handleReturnToMain()` – powrót do głównego widoku.

8.2 Transaction

- **Rola:** reprezentuje pojedynczą transakcję.
- **Pola:**
 - `int id`
 - `String title` (opis)
 - `double amount`
 - `String date` (format ISO: YYYY-MM-DD)
 - `String category`
- **Konstruktor:**
 - `Transaction(int id, String title, double amount, String date, String category)`
 - `Transaction(String title, double amount, String date, String category)` (bez ID, do wstawiania)

- **Gettery/Settery** – `getId()`, `getTitle()`, `getAmount()`, `getDate()`, `getCategory()`.

8.3 Database

- **Rola:** połączenie z lokalną bazą SQLite, metody CRUD, obsługa limitów.
- **Główne metody:**
 - `static Connection connect()` – zwraca obiekt `Connection` do pliku bazy `finance_tracker.db`.
 - `static boolean isLimitExceeded(Transaction t)` – sprawdza, czy dodanie transakcji w bieżącym okresie nie przekroczy limitu zdefiniowanego przez użytkownika.
 - `static void setLimit(double amount, String period)` – zapis/aktualizacja limitu (okres: "Daily", "Weekly", "Monthly").
 - (ewentualnie) `static double getCurrentLimit(String period)` – pobranie zapisanego limitu dla danego okresu.
- **Inicjalizacja:** w `createTableIfNotExists()` tworzona jest tabela `transactions`, jeśli nie istnieje.

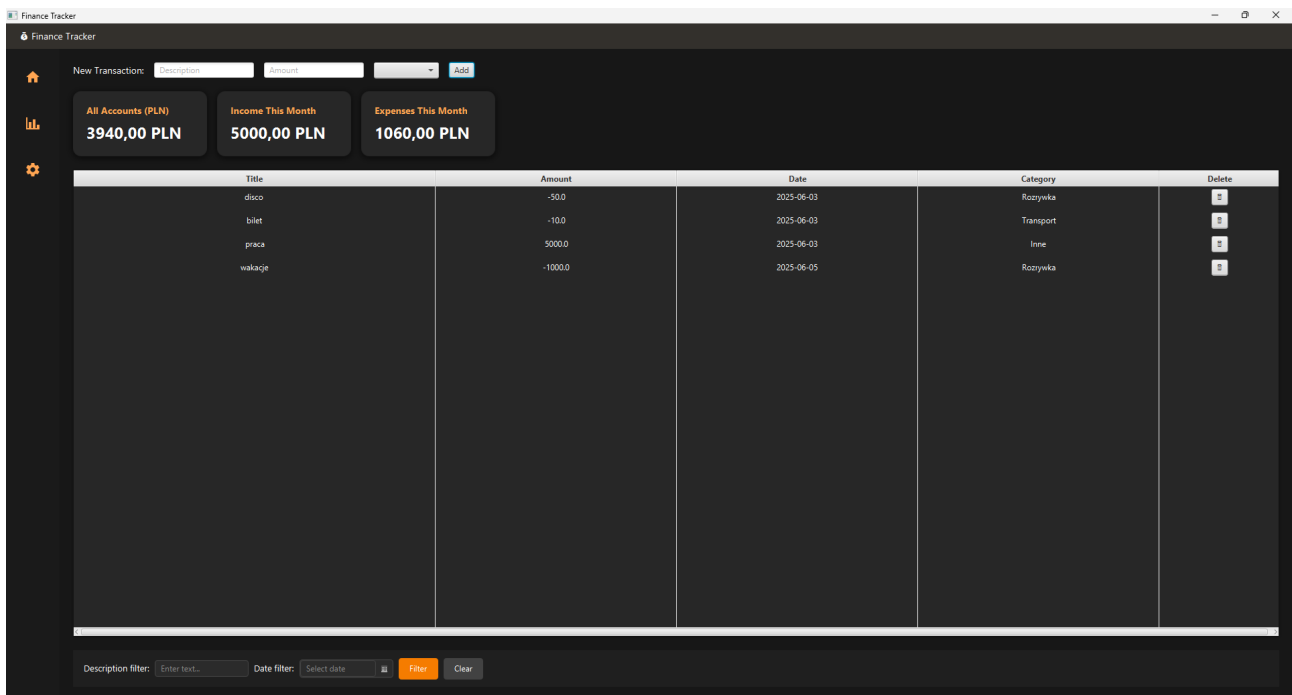
8.4 Main (klasa startowa)

- Zawiera metodę `public void start(Stage primaryStage):`
 1. Ładuje plik FXML:
`FXMLLoader.load(getClass().getResource("main.fxml"))`.
 2. Ustawia scenę (`Scene scene = new Scene(root)`).
 3. Dołącza plik stylów CSS:
`scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm())`.
 4. Wyświetla okno (`primaryStage.setScene(scene);`
`primaryStage.show()`).

9. Scenariusze użytkowania / Przykładowe zrzuty ekranu

9.1 Widok główny

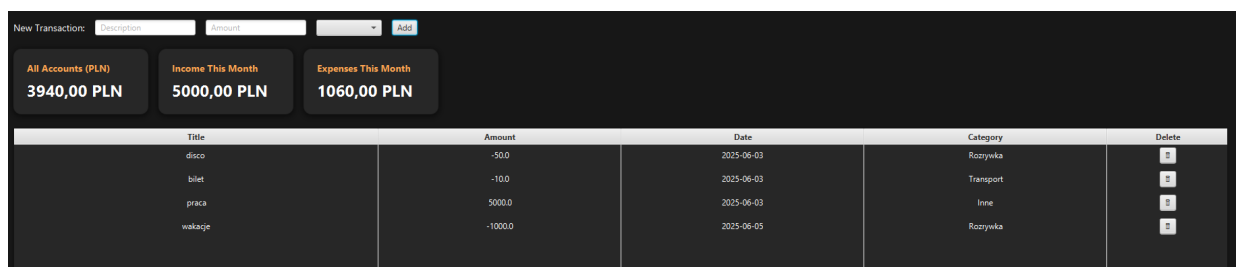
Po uruchomieniu aplikacji wyświetla się ekran podzielony na: pasek górny, lewą nawigację i środkową część z formularzem, kartami podsumowania, tabelą i filtrem.



[Zrzut ekranu 1: Widok główny (main screen)]

9.2 Dodawanie nowej transakcji

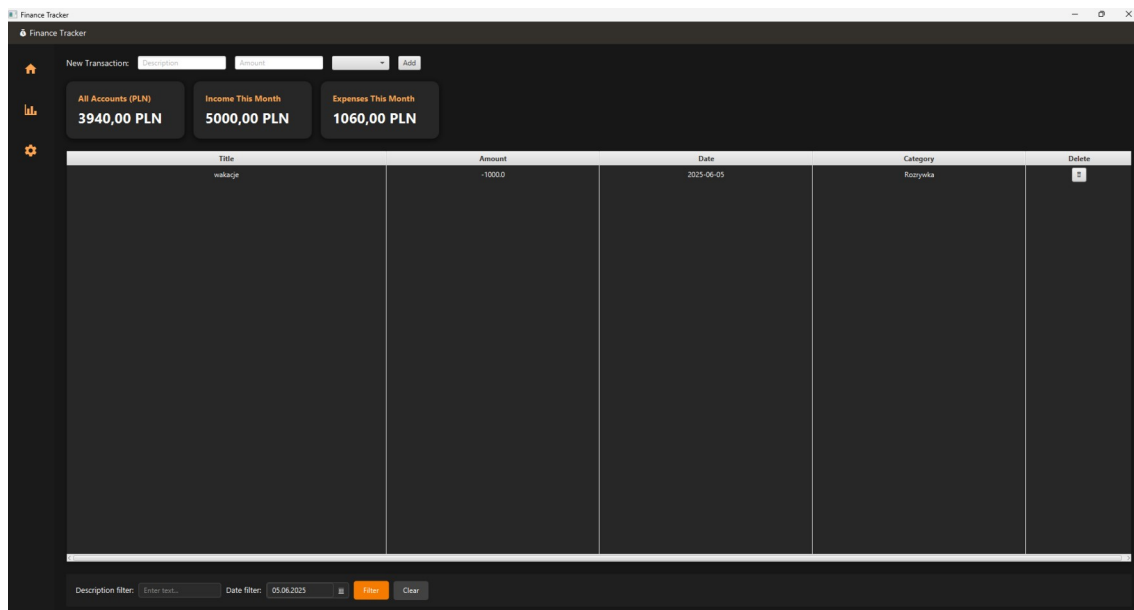
1. Użytkownik wpisuje opis, kwotę i wybiera kategorię.
2. Po kliknięciu **“Add”** transakcja pojawia się w tabeli i aktualizowane są karty podsumowania.



[Zrzut ekranu 2: Formularz dodawania transakcji + efekt w tabeli]

9.3 Filtrowanie listy

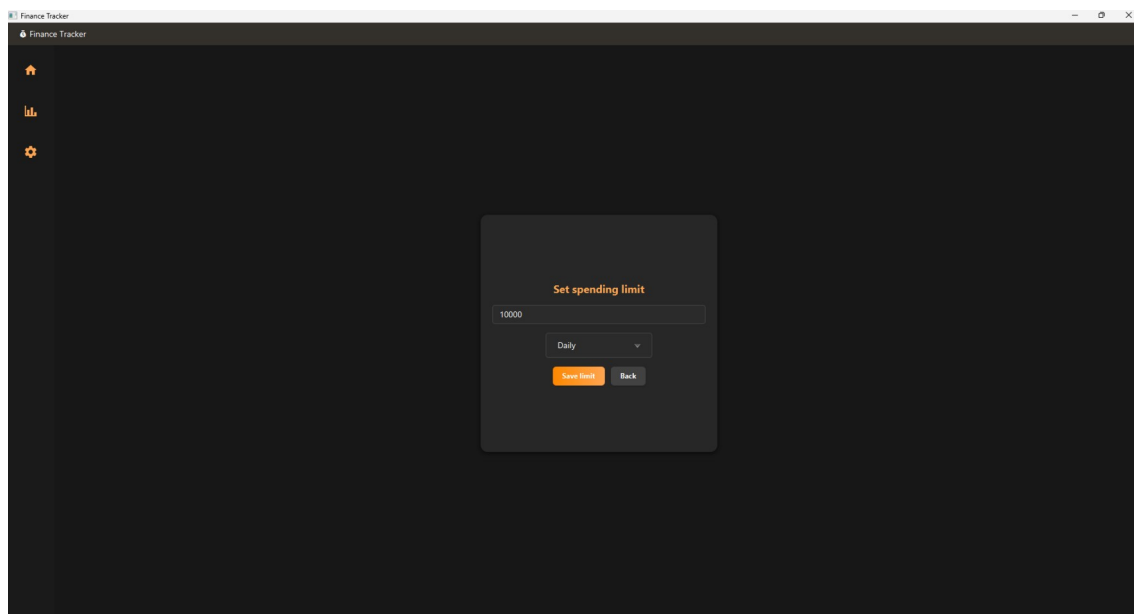
1. Wpisanie fragmentu opisu w polu **“Description filter”** ogranicza wyświetlane wiersze.
2. Wybór daty w kalendarzu (DatePicker) filtruje transakcje tylko z określonego dnia.
3. Kliknięcie **“Filter”** włącza filtr, a **“Clear”** przywraca wszystkie wiersze.



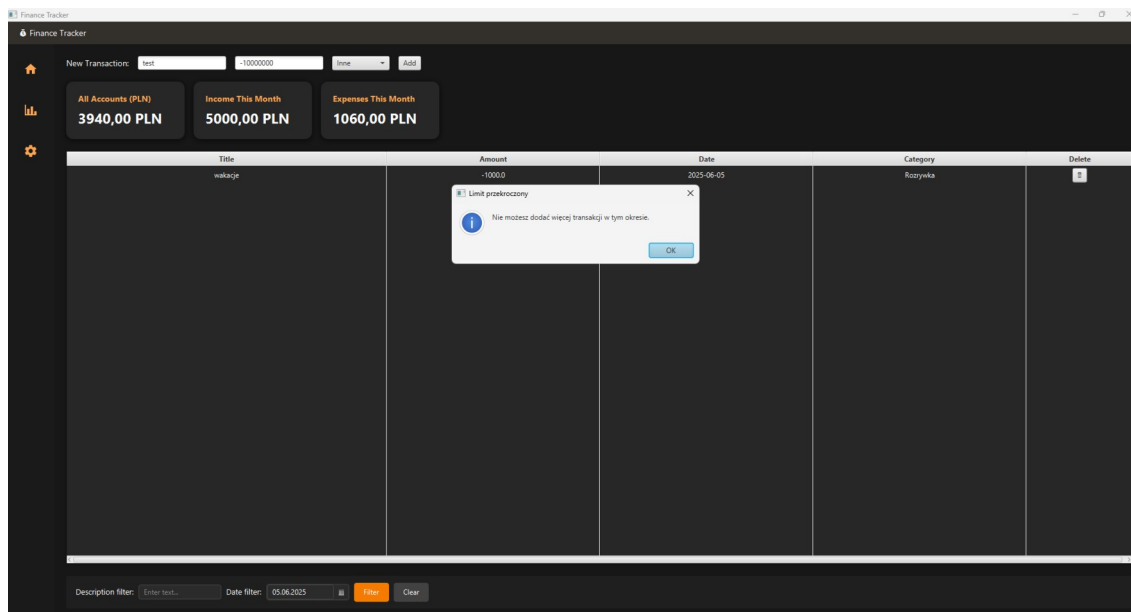
[Zrzut ekranu 3: Przykład działania filtrów (lista ograniczona do jednego dnia)]

9.4 Ustawianie limitów

1. Kliknięcie ikony ⚙️ (sidebar) otwiera panel “Set spending limit”.
2. Użytkownik wybiera okres (“Daily”, “Weekly”, “Monthly”) oraz podaje kwotę.
3. Kliknięcie “Save limit” zapisuje dane.
4. Próba dodania transakcji, która przekracza zapisany limit, wyświetla Alert i blokuje zapis.




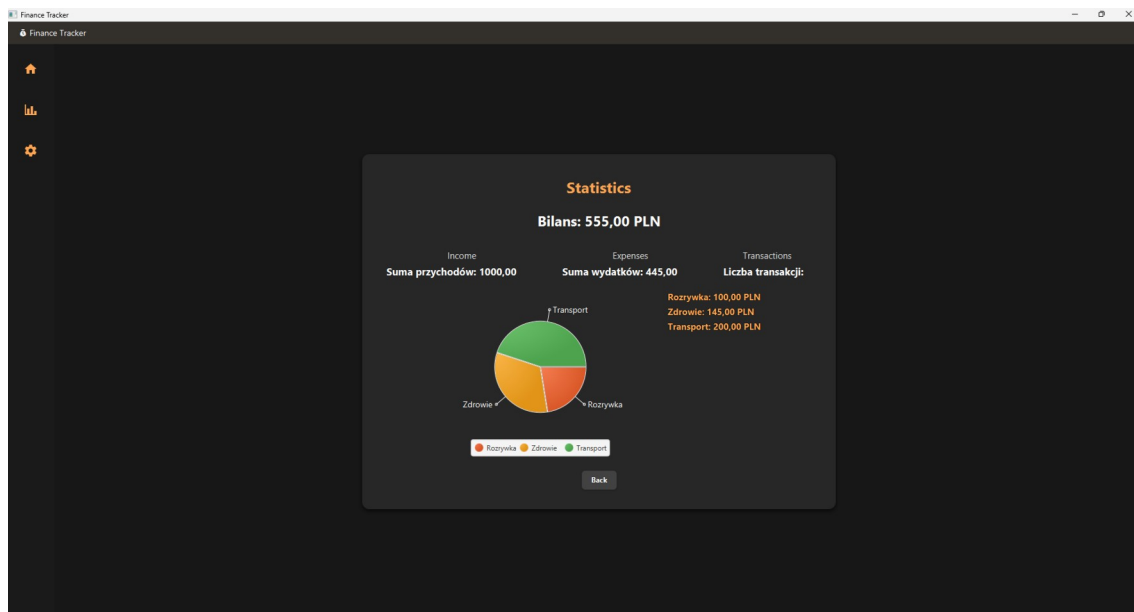
[Zrzut ekranu 4: Widok panelu ustawień limitów]



[Zrzut ekranu 5: Alert informujący o przekroczeniu limitu]

9.5 Widok statystyk

1. Kliknięcie ikony  (sidebar) otwiera panel **“Statistics”**.
2. Wyświetlane są:
 - Suma przychodów, wydatków, bilans i liczba transakcji.
 - Wykres kołowy z rozkładem wydatków wg kategorii.
 - Lista wartości kategorii obok wykresu.
3. Kliknięcie **“Back”** powraca do głównego widoku.



[Zrzut ekranu 6: Widok statystyk z wykresem kołowym]

10. Rozszerzenia i kierunki rozwoju

1. Obsługa wielu walut

- Możliwość definiowania kont w różnych walutach, automatyczna konwersja (np. przy użyciu zewnętrznego API kursów).

2. Eksport/import danych

- Możliwość eksportu transakcji do pliku CSV / XLSX.
- Import transakcji z plików bankowych.

3. Powiadomienia i reminder'y

- Przypomnienie np. w dniu wypłaty (możliwość integracji z kalendarzem).

4. Zaawansowane raporty

- Generowanie raportów PDF (wykresy, podsumowanie roczne).

5. Bezpieczeństwo

- Hasło dostępu do aplikacji, szyfrowanie pliku bazy.

6. Aplikacja mobilna / Webowa

- Wersja responsywna jako aplikacja webowa (np. Spring Boot + React).

7. Synchronizacja z chmurą

- Archiwizacja bazy w chmurze (Dropbox, Google Drive).

8. Planowanie budżetu

- Ustawianie budżetu na kategorie i śledzenie odchyleń.
-

11. Podsumowanie

Projekt **Finance Tracker** przedstawia kompletną, prostą aplikację do zarządzania finansami osobistymi, wykorzystującą JavaFX, SQLite oraz FXML. Dzięki przejrzystemu interfejsowi, filtrowaniu transakcji, graficznym statystykom oraz możliwości definiowania limitów, użytkownik zyskuje szybki wgląd w swoje finanse i kontrolę nad wydatkami. Kod jest modularny i łatwy do rozbudowy – w przyszłości można dodać eksport/import, obsługę wielu walut czy integrację z serwisami zewnętrznymi.

Autorzy: Dominik Pakuła, Paweł Kulesza