

# Sprawozdanie – Aplikacja do zarządzania finansami

Przedmiot: Systemy urządzeń mobilnych

Autorzy: Jakub Opiełka, Dominik Szczepanik, Wojciech  
Wróbel

Kierunek: Informatyka dla inżynierów

Semestr: I

## 1. Wstęp

Aplikacja została stworzona jako nowoczesne narzędzie do zarządzania finansami osobistymi. Umożliwia użytkownikowi łatwe śledzenie przychodów i wydatków, analizowanie danych oraz planowanie budżetu miesięcznego.

## 2. Opis funkcjonalności aplikacji

Aplikacja pozwala na:

- dodawanie i przeglądanie transakcji (przychody i wydatki),
- ustawienie budżetu miesięcznego,
- generowanie raportów finansowych,
- wizualizację danych (wykresy: kołowy, słupkowy, liniowy),
- filtrowanie transakcji po miesiącach,
- działanie w trybie offline i automatyczną synchronizację danych.

### 3. Opis techniczny krok po kroku

#### Dodawanie i zapis transakcji

Formularz pozwala użytkownikowi dodać przychód lub wydatek, który trafia do backendu przez TransactionService. Jeśli użytkownik jest offline, dane trafiają do localStorage i są zsynchronizowane później.

Moje transakcje

Przychód

Kategoria

0

22.05.2025

Zapisz

#### Fragment kodu

```
2 |
3 // POST a new transaction
4 router.post('/', async (req, res) => {
5   try {
6     const newTransaction = new Transaction(req.body);
7     await newTransaction.save();
8     res.status(201).json(newTransaction);
9
10    const sub = getSubscription();
11    if (sub) {
12      const payload = JSON.stringify({
13        title: '👉 Nowa transakcja',
14        body: `${newTransaction.type === 'income' ? '💰 Przychód' : '💸 Wydatek'} 📄 ${newTransaction.amount} zł`
15      });
16
17      await webpush.sendNotification(sub, payload);
18      console.log('📢 Wysłano powiadomienie push');
19    }
20  } catch (err) {
21    console.error('❌ Błąd zapisu transakcji:', err);
22    res.status(500).json({ error: 'Wewnętrzny błąd serwera' });
23  }
24 });
```

## Wyświetlanie listy transakcji

TransactionListComponent pobiera dane z backendu lub z localStorage, gdy brak internetu. Transakcje są sortowane i pokazywane w formie listy.


Lista transakcji	
6/14/25	💎 dsfsd – 234 zł
5/30/25	💎 impreza – 100 zł
5/23/25	🍷 Dniówka – 600 zł
5/16/25	💎 Sport – 50 zł
5/15/25	🍷 Transport – 22 zł
5/14/25	🍷 dd – 22 zł
5/14/25	🍷 inne – 22 zł

## Fragment kodu

```
16
17 // GET all transactions
18 router.get('/', async (req, res) => {
19   const transactions = await Transaction.find().sort({ date: -1 });
20   res.json(transactions);
21 });
22
23 // POST a new transaction
```


## Ustawianie miesięcznego budżetu


BudgetComponent umożliwia ustawienie budżetu miesięcznego. Wartość jest zapisywana do backendu, a w trybie offline – lokalnie.


 **Miesięczny budżet**


Ustaw budżet:


3000

 Dzisiaj: 22.05.2025

 Pozostałe dni w miesiącu: 9

 Wydano: 2834 zł

 Pozostało: 166 zł

 Możesz wydawać około **18.44 zł** dziennie.

## Fragment kodu

```
mobilliProjekt > backend > routes > budget.js > router.post('/') callback > updated
1  const express = require('express');
2  const router = express.Router();
3  const Budget = require('../models/budget');
4
5  // GET budget for given month
6  router.get('/:month', async (req, res) => {
7    const { month } = req.params;
8    const budget = await Budget.findOne({ userId: 'demo', month });
9    res.json(budget || { amount: 0 });
10 });
11
12 // SET/UPDATE budget
13 router.post('/', async (req, res) => {
14   const { month, amount } = req.body;
15   const updated = await Budget.findOneAndUpdate(
16     { userId: 'demo', month },
17     { amount, updatedAt: new Date() },
18     { upsert: true, new: true }
19   );
20   res.json(updated);
21 });
22
23 router.post('/test', (req, res) => {
```

## Synchronizacja danych offline/online

Zarówno TransactionService, jak i BudgetService posiadają logikę zapisywania danych lokalnie oraz ich synchronizacji, gdy aplikacja odzyska połączenie z internetem. Wykrycie stanu online wykonywane jest przez navigator.onLine.

### Fragmenty kodu

#### Budget service (frontend )

```
@Injectable({ providedIn: 'root' })
export class BudgetService {
  private readonly API_URL = 'http://localhost:4000/api/budget';
  private readonly STORAGE_KEY = 'monthlyBudget';
  private readonly PENDING_KEY = 'pending-budget';

  constructor(private http: HttpClient, private localStorage: LocalStorageService) {}

  private getMonthKey(): string {
    const now = new Date();
    return `${now.getFullYear()}-${(now.getMonth() + 1).toString().padStart(2, '0')}`;
  }

  loadBudget(): Observable<number> {
    const month = this.getMonthKey();

    if (!navigator.onLine) {
      console.warn('⚠ Offline ☐ używam lokalnego budżetu');
      const local = this.localStorage.load<{ amount: number, month: string }>(this.STORAGE_KEY);
      return of(local?.amount || 0);
    }

    return this.http.get<{ amount: number }>(`${this.API_URL}/${month}`).pipe(
      tap(data => this.localStorage.save(this.STORAGE_KEY, { ...data, month })),
      map(data => data.amount),
      catchError(() => {
        console.warn('⚠ Błąd pobierania budżetu ☐ fallback do localStorage');
        const local = this.localStorage.load<{ amount: number, month: string }>(this.STORAGE_KEY);
        return of(local?.amount || 0);
      })
    );
  }
};
```

#### Transaction service (backend)

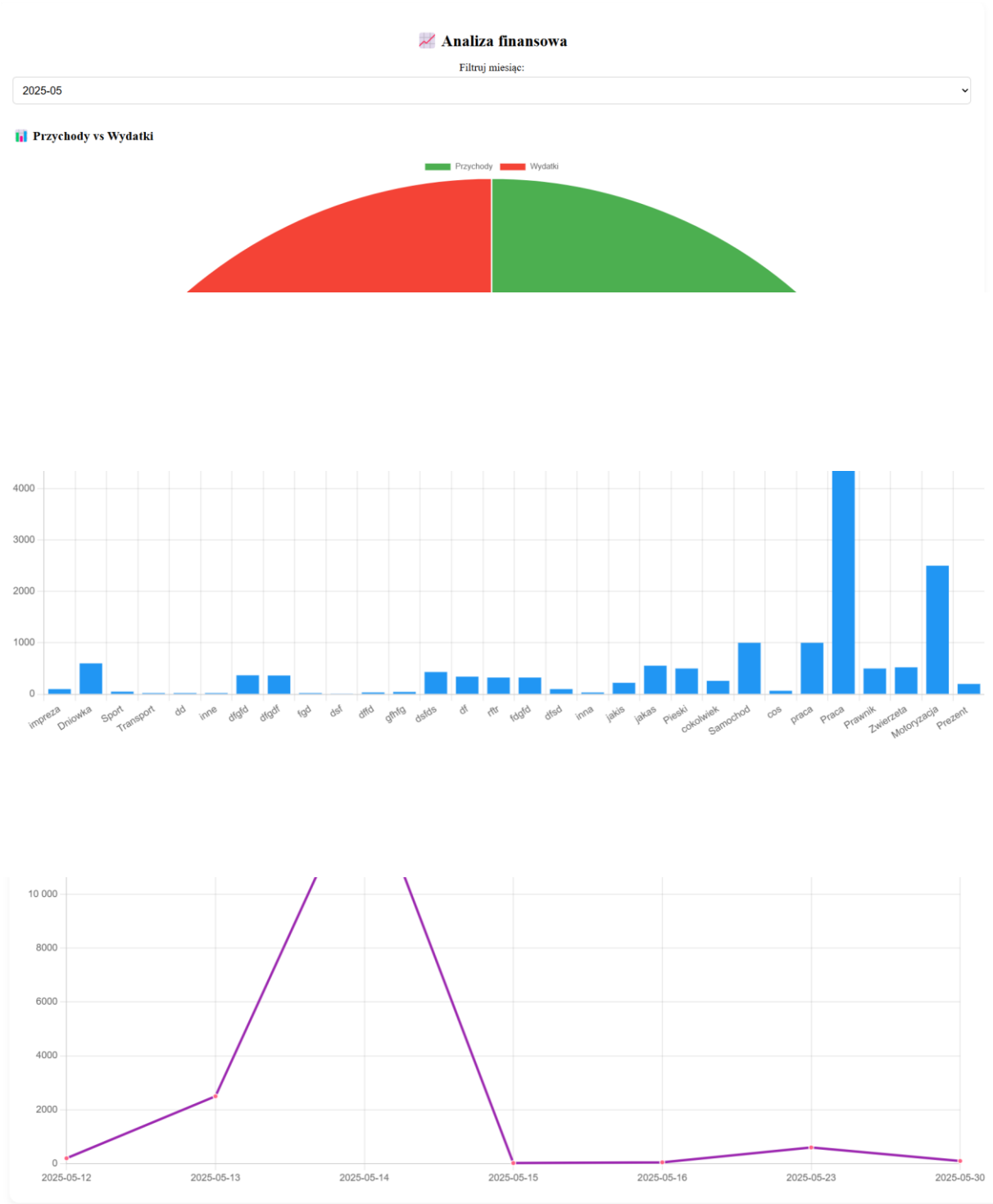
```
loadTransactions(): void {
  if (!navigator.onLine) {
    console.warn('⚠ Offline ☐ używam danych z localStorage');
    const local = this.localStorage.load<Transaction[]>(this.LOCAL_KEY) || [];
    this.transactionsSubject.next(local);
    return;
  }

  this.http.get<Transaction[]>(this.API_URL).pipe(
    tap(data => {
      this.localStorage.save(this.LOCAL_KEY, data);
      this.transactionsSubject.next(data);
    }),
    catchError(() => {
      console.warn('⚠ Błąd pobierania ☐ fallback do localStorage');
      const local = this.localStorage.load<Transaction[]>(this.LOCAL_KEY) || [];
      this.transactionsSubject.next(local);
      return of([]);
    })
  ).subscribe();
}

addTransaction(tx: Transaction): Observable<Transaction> {
  if (!navigator.onLine) {
    console.warn('🛑 Brak połączenia ☐ zapisuję lokalnie (offline)');
    const pending = this.localStorage.load<Transaction[]>(this.PENDING_KEY) || [];
    this.localStorage.save(this.PENDING_KEY, [...pending, tx]);
  }
}
```

Analiza danych i wykresy

ChartAnalysisComponent generuje wykresy (kołowy, słupkowy, liniowy) na podstawie danych transakcji. Możliwe jest filtrowanie po miesiącu. Wykresy są responsywne i czytelne także na telefonach.



## Fragment kodu

```
selector: 'app-chart-analysis',
imports: [CommonModule, FormsModule, NgChartsModule],
template: `
<div class="card analysis">
<h2 class="card-title">📊 Analiza finansowa</h2>
<label class="form-label">
  Filtruj miesiąc:
  <select [(ngModel)]="selectedMonth" (change)="updateCharts()" class="form-control">
    <option *ngFor="let month of availableMonths" [value]="month">{{ month }}</option>
  </select>
</label>
<h3 class="chart-header">📈 Przychody vs Wydatki</h3>
<canvas baseChart [type]="pie" [data]="pieChartData" [options]="chartOptions"></canvas>
<h3 class="chart-header">📊 Kategorie</h3>
<canvas baseChart [type]="bar" [data]="barChartData" [options]="chartOptions"></canvas>
<h3 class="chart-header">📈 Trend dzienny</h3>
<canvas baseChart [type]="line" [data]="lineChartData" [options]="chartOptions"></canvas>
</div>
`
})
export class ChartAnalysisComponent {
  selectedMonth: string = this.formatMonth(new Date());
  availableMonths: string[] = [];

  pieChartData: ChartData<'pie', number[], string[]> = {
    labels: [['Przychody'], ['Wydatki']],
    datasets: [{ data: [0, 0], backgroundColor: ['#4caf50', '#f44336'] }]
  };

  barChartData: ChartData<'bar', number[], string[]> = {
```

## Raport miesięczny

ReportComponent zlicza sumaryczne przychody, wydatki i saldo na podstawie danych transakcji. Wartości są wizualnie wyróżnione kolorem w zależności od wyniku.


📊 Raport finansowy	
📈 Przychody:	15441 zł
📊 Wydatki:	2834 zł
📈 Saldo:	12607 zł

## Fragment kodu

```
9  template: `
10  <div class="card report">
11    <h2 class="card-title">📊 Raport finansowy</h2>
12    <div class="report-item">
13      <span>📈 Przychody:</span>
14      <strong>{{ income }} zł</strong>
15    </div>
16    <div class="report-item">
17      <span>📊 Wydatki:</span>
18      <strong>{{ expense }} zł</strong>
19    </div>
20    <div class="report-item balance" [ngClass]="{ negative: balance < 0, positive: balance >= 0 }">
21      <span>📈 Saldo:</span>
22      <strong>{{ balance }} zł</strong>
23    </div>
```

## Responsywny interfejs użytkownika

Wszystkie komponenty zostały dostosowane do wyświetlania na urządzeniach mobilnych. Wykorzystano elastyczne kontenery, media queries oraz jednolitą estetykę komponentów.

 **FinTrack**

🏠 Transakcje

💰 Budżet

📊 Raporty

📈 Analiza

📅

**Moje transakcje**

Przychód

Kategoria

0

22. 05. 2025

Zapisz

📋

**Lista transakcji**

6/14/25 | 💎 dsfsd – 234 zł

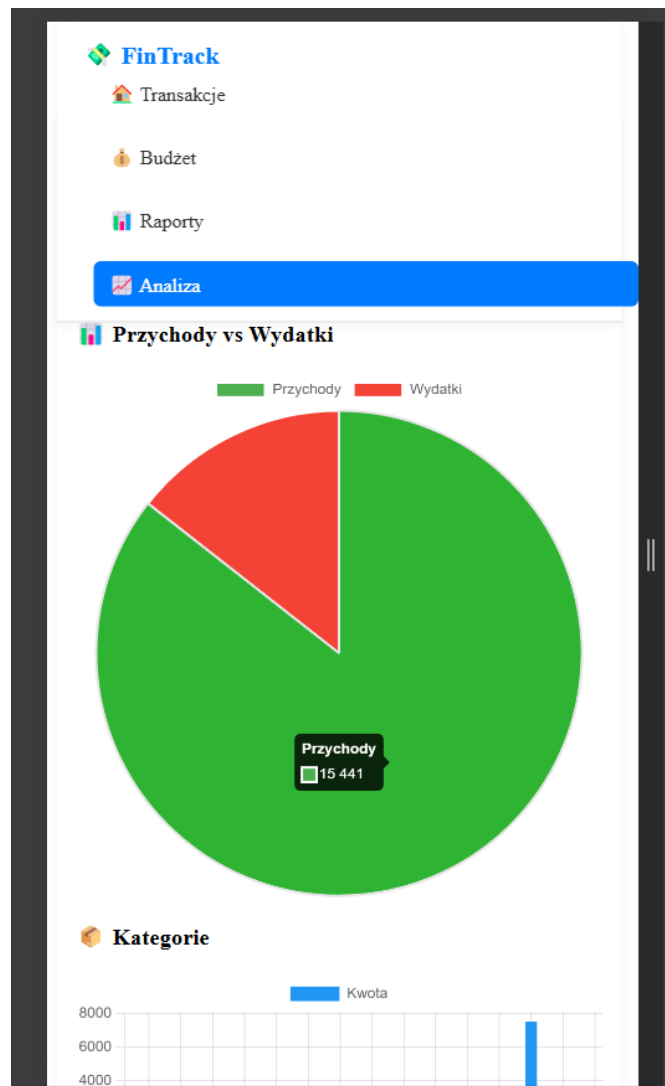
5/30/25 | 💎 impreza – 100 zł

5/23/25 | 💰 Dniówka – 600 zł

5/16/25 | 💎 Sport – 50 zł

5/15/25 | 💰 Transport – 22 zł





Fragment kodu

```
1  .balance.negative {  
2    color: #dc3545;  
3  }  
4  
5  .transaction-list {  
6    list-style: none;  
7    padding: 0;  
8  }  
9  
10 .transaction-list li {  
11   border-bottom: 1px solid #eee;  
12   padding: 8px 0;  
13   font-size: 16px;  
14 }  
15  
16 @media screen and (max-width: 600px) {  
17   .card {  
18     padding: 15px;  
19   }  
20  
21   .card-title {  
22     font-size: 1.2rem;  
23   }  
24 }
```

#### 4. Trudności i ich rozwiązania

Największym wyzwaniem była implementacja powiadomień push. Pomimo wielu prób i poprawnej konfiguracji backendu oraz udzielonej zgody przez przeglądarkę, powiadomienia nie pojawiały się po stronie użytkownika. Ostatecznie zrezygnowano z tej funkcji.

Drugim istotnym problemem było zapewnienie synchronizacji danych między trybem offline a online. Wymagało to stworzenia logiki, która monitoruje stan połączenia sieciowego oraz automatycznie przenosi dane z localStorage do bazy danych po odzyskaniu połączenia.

#### 5. Podsumowanie:

Aplikacja spełnia swoje główne założenia i działa stabilnie w warunkach zarówno online, jak i offline. Mimo rezygnacji z funkcji push, projekt stanowi solidną i funkcjonalną platformę do zarządzania budżetem w sposób przystępny i estetyczny.