# Documentație Tehnică: MoneyManagementApp


**Student:** Andrei Marcu
**Student:** Bodea George
**Student:** Cosma Cosmin
**Student:** Antonio Chirtes **Grupa**: 30233

**Link Repository: https://github.com/marcuandrei1/MoneyManagementApp**

# 1. Descriere Generală

**MoneyManagementApp** este o aplicație desktop de gestiune financiară personală, bazată pe principiul contabil de **partidă dublă (double-entry bookkeeping)**. Proiectul este inspirat funcțional din aplicația open-source **GnuCash**, având ca scop oferirea unei interfețe moderne și simplificate pentru urmărirea fluxurilor financiare.

Spre deosebire de aplicațiile simple de "Expense Tracking", care scad doar banii dintr-un buget, acest sistem asigură integritatea datelor: orice sumă care iese dintr-un cont trebuie să intre în altul (ex: Banii ies din *Cont Curent* și intră în *Cheltuieli Mâncare*).

# 2. Arhitectura Sistemului

Aplicația este construită pe o arhitectură stratificată, decuplată, folosind tehnologii moderne pentru fiecare componentă.

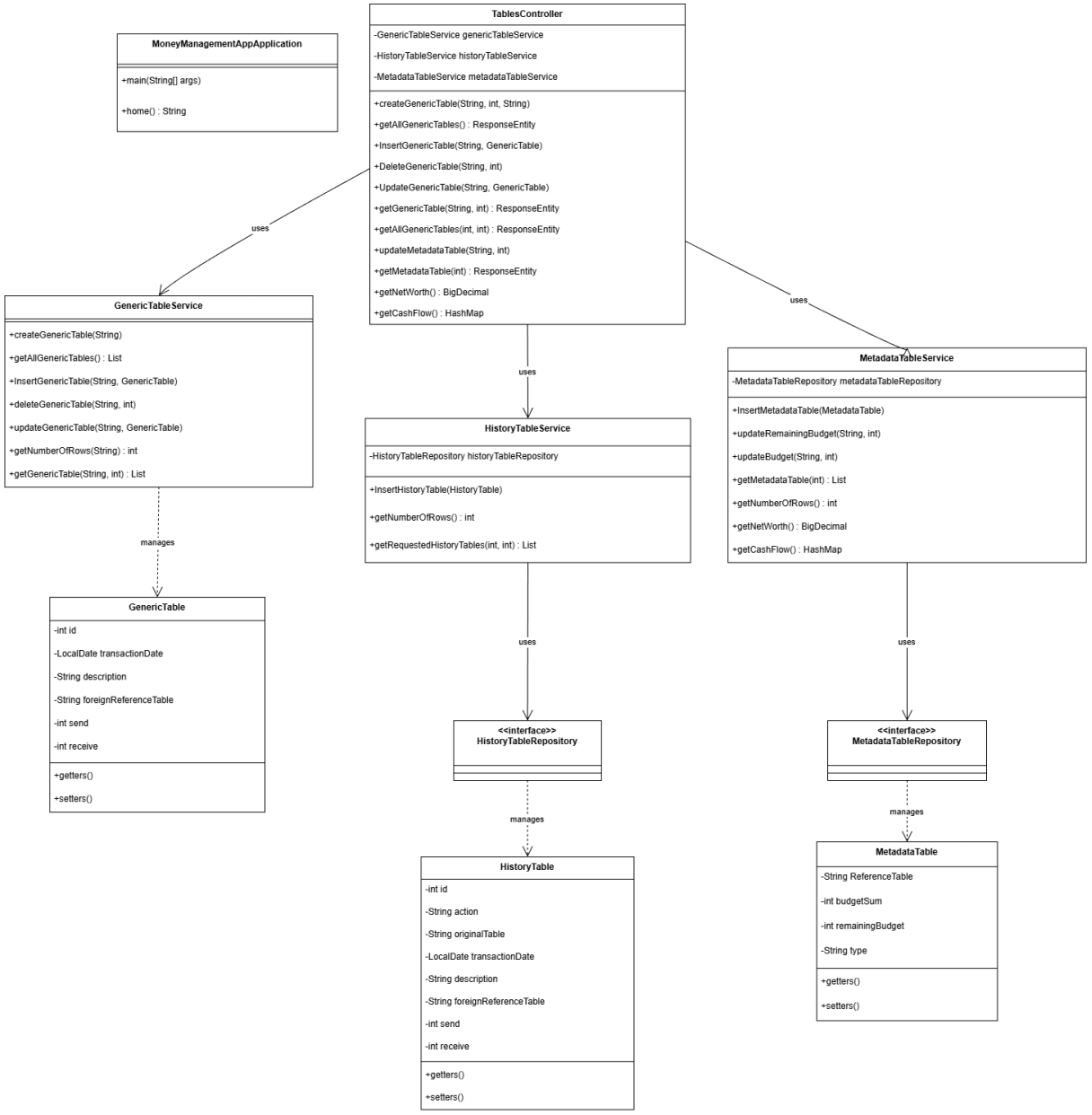
## 2.1 Backend (Java & Spring Boot)

Partea de server este responsabilă pentru logica de business și persistența datelor.

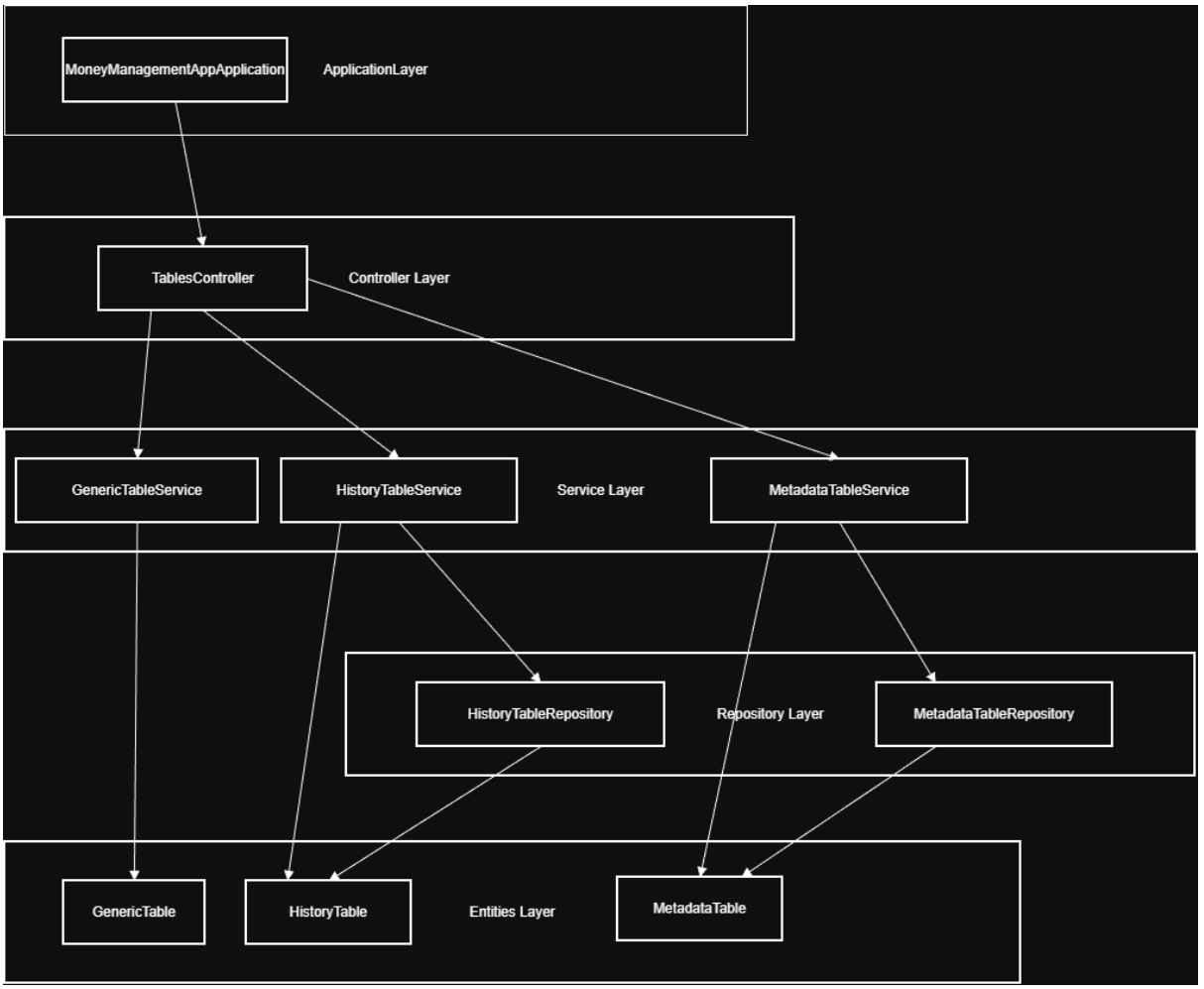
- **Limbaj:** Java 17.
- **Framework:** Spring Boot (v3.5.x) - ales pentru rapiditatea dezvoltării și gestionarea automată a dependențelor.
- **Build Tool:** Gradle - folosit pentru gestionarea bibliotecilor.
- **Baza de Date:** MySQL - stocare relațională.
- **ORM:** Hibernate/JPA - maparea obiectelor Java la tabelele bazei de date.
- **Rol:** Expune un API pe care interfața grafică le apelează pentru a crea conturi sau a înregistra tranzacții.

## 2.1.1 Diagrame Backend

- **Diagrama de clase:**



- **Diagrama de pachete:**




## 2.2 Frontend (React & Interfață Utilizator)

Interfața cu utilizatorul este construită ca o aplicație React modernă, utilizând componente funcționale și Hooks pentru gestionarea stării.

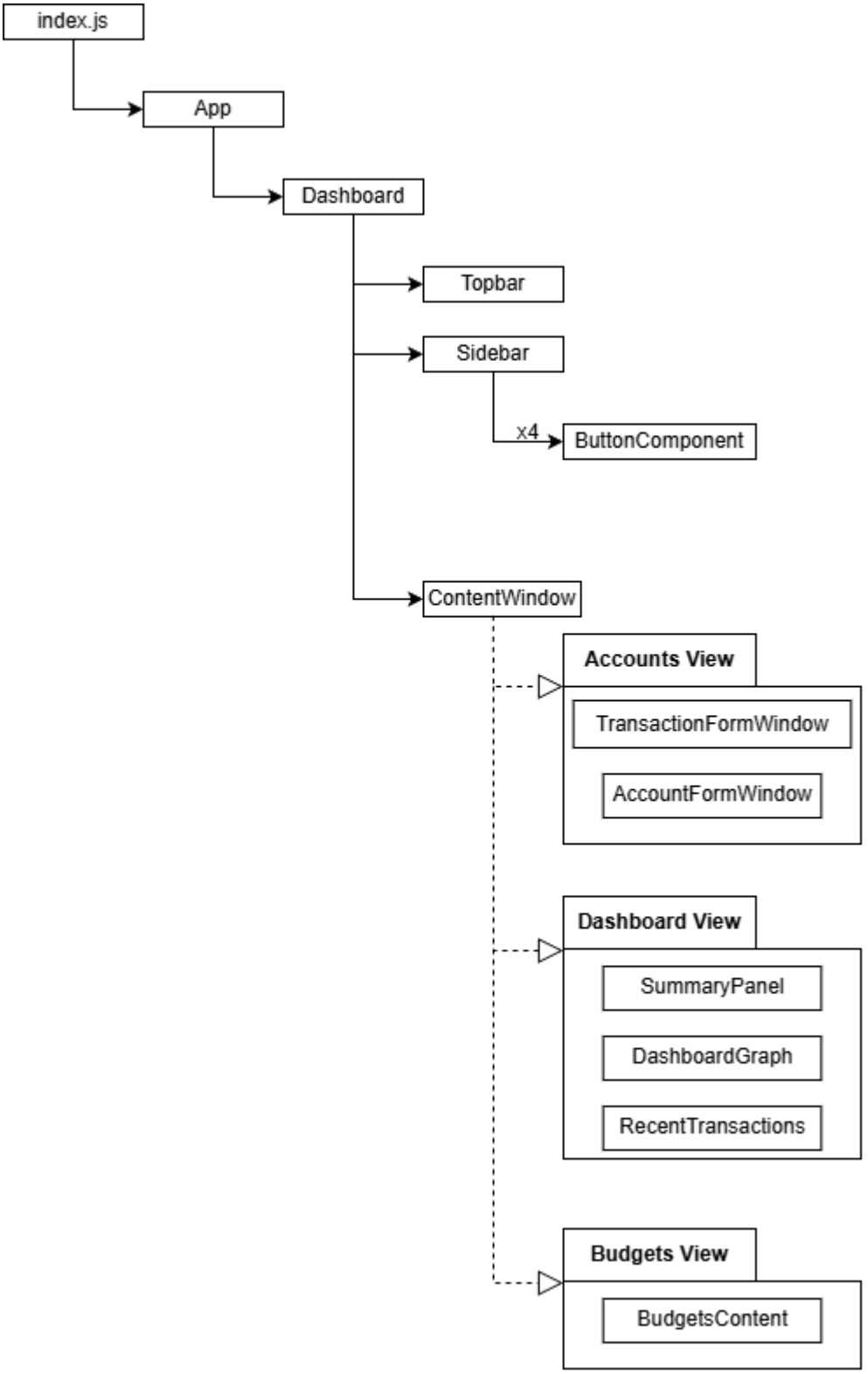
- **Framework: React.js**. Arhitectura este bazată pe componente funcționale (ex: `Dashboard`, `AccountsContent`) și utilizarea hook-urilor standard (`useState`, `useEffect`, `useRef`) pentru logica reactivă.
- **Design & Stilizare (Abordare Hibridă):**
    - **CSS Custom:** Pentru layout-ul general, ferestre modale și stilizarea specifică a aplicației s-au folosit fișiere CSS dedicate (ex: `transactionFormWindow.css`, `sidebarStyle.css`), oferind un control fin asupra aspectului vizual.
    - **Material UI (MUI):** S-au integrat componente specifice din biblioteca Material UI pentru elementele complexe de interfață care necesitau funcționalități

avansate, cum ar fi `Pagination` și `TablePagination` pentru tabelele de date.

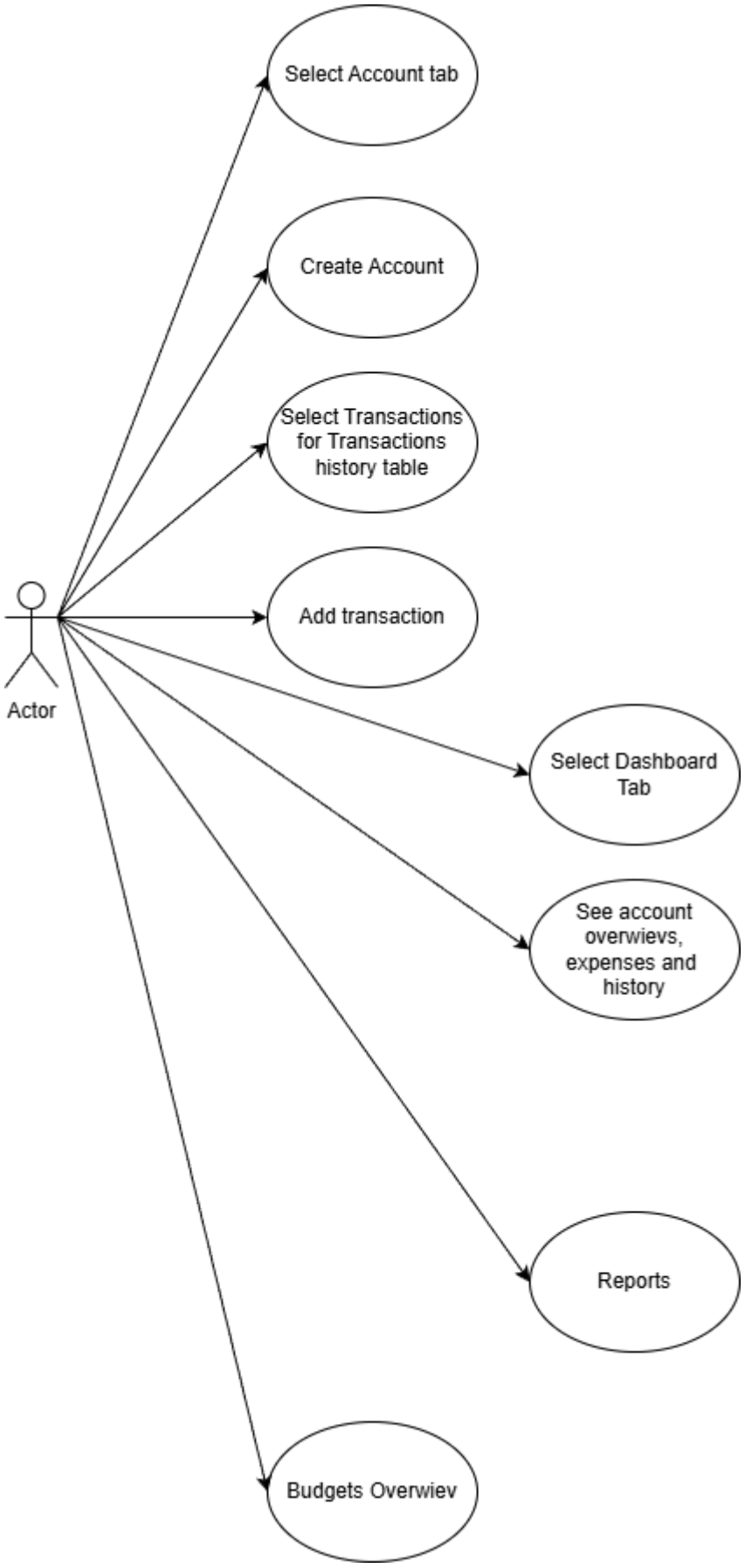
- **Vizualizare Date (Charts):**
  - S-a utilizat **`@mui/x-charts`** (componenta `BarChart`) pentru a reda graficul de "Monthly Cash Flow" în Dashboard.
  - Pentru secțiunea de Bugete, s-a implementat o logică vizuală personalizată care calculează dinamic culoarea barelor de progres (de la verde la roșu) în funcție de raportul dintre suma cheltuită și cea alocată.
- **Comunicare cu Backend-ul:** Interacțiunea cu API-ul Java se realizează prin **Fetch API** (nativ JavaScript). Cererile HTTP sunt asincrone (`async/await`) și gestionează datele JSON primite de la endpoint-uri precum `/tables/getNetWorth` sau `/tables/createTable`.
- **Navigație & State Management:** Aplicația funcționează ca un Single Page Application (SPA), însă navigația internă este gestionată prin **Conditional Rendering**. Componenta principală `Dashboard.js` menține un state activ (`activeView`), schimbând dinamic conținutul afișat (Accounts, Dashboard, Budgets) fără a reîncărca pagina.
- **Editare Interactivă:** Tabelele permit editarea datelor direct în celulă (Inline Editing) și salvarea automată la pierderea focusului (`onBlur`), optimizând experiența utilizatorului.

### 2.2.1 Diagrame Frontend

- **Diagrama ierarhică a Componentelor:**



- **Diagrama UseCase:**



### 2.3 Desktop Integration (Electron)

Deși aplicația este web-based, **Electron** este folosit ca un "wrapper" pentru a o transforma într-o aplicație desktop nativă.

- **Rol:** Electron creează o fereastră de browser care încarcă aplicația React.
- **Avantaj:** Permite rularea aplicației ca un executabil (.exe) independent de browserul utilizatorului.

## 3. Principii de Funcționare: Double-Entry Bookkeeping

În contextul aplicației, fiecare tranzacție (Transaction) are cel puțin două "picioare":

1. **Sursă:** De unde pleacă banii (Credit).
2. **Destinație:** Unde ajung banii (Debit).

Exemplu:

Dacă utilizatorul cumpără o cafea de 15 RON:

- Contul Active:Portofel scade cu 15 RON.
- Contul Cheltuieli:Cafea crește cu 15 RON.
  Astfel, suma totală din sistem rămâne mereu echilibrată.

## 4. Funcționalități Cheie

### 4.1 Gestiunea Conturilor (Planul de Conturi)

Utilizatorul poate crea o ierarhie de conturi.

- **Tipuri de conturi:** Active (Bancă, Cash), Venituri (Salariu, Cadouri), Cheltuieli (Chirie, Mâncare).
- Aplicația permite operațiuni CRUD (Create, Read, Update, Delete) asupra conturilor.

### 4.2 Registrul de Tranzacții

Interfața principală unde se introduc datele.

- Adăugarea unei tranzacții necesită specificarea datei, descrierii și a celor două conturi implicate.
- Validări: Nu se poate salva o tranzacție dacă suma debitată nu este egală cu cea creditată.

### 4.3 Rapoarte Vizuale

Folosind biblioteca `@mui/x-charts`, aplicația generează grafice pentru o interpretare rapidă a datelor:

- **Bar Chart:** Cheltuieli lunare grupate pe categorii.

### 4.4 Monitorizarea Bugetelor (Budgets)

Această funcționalitate permite utilizatorilor să își seteze limite de cheltuieli pentru diferite categorii (ex: Mâncare, Transport).

- **Vizualizare Progresivă:** Fiecare buget este reprezentat printr-un "card" care afișează suma alocată versus suma rămasă.
- **Feedback Vizual Dinamic:** Bara de progres își schimbă culoarea automat în funcție de gradul de epuizare a bugetului. Aceasta trece gradual de la **verde** (fonduri suficiente) la **roșu** (buget aproape epuizat), oferind o avertizare vizuală imediată.
- **Editare Rapidă:** Utilizatorul poate modifica suma alocată direct din interfață (click-to-edit), fără a deschide ferestre suplimentare.
- **Iconițe Inteligente:** Aplicația asociază automat o pictogramă relevantă fiecărui buget, analizând numele acestuia (ex: un buget numit "Energy" va primi automat o iconiță de fulger).

# 5. Concluzii

MoneyManagementApp demonstrează utilizarea unui stack tehnologic modern (Java + React + Electron) pentru a rezolva o problemă clasică de contabilitate. Arhitectura aleasă permite extinderea ușoară a funcționalității (de exemplu, adăugarea exportului în PDF sau sincronizarea în cloud pe viitor).

# 6. Ce a facut fiecare

Andrei Marcu :
Am lucrat impreuna cu Cosmin Cosma pe frontend. Am facut diferite componente fiecare, iar concret as putea zice ca am facut:

- DashBoard-ul
- Graficele de la Dashboard
- Panelurile si componentele de pe pagina de Dashboards
- Styling pentru fiecare componenta
- Sidebar, iar apoi a fost retusat de Cosmin

Personal vreau sa specific ca chiar am acoperit fiecare unu pe altu. Eu am facut anumite componente pe care dupaia Cosmin le-a retusat daca eu nu am mai avut chef de ele.

Cosma Cosmin:
Am venit cu ideea proiectului și m-am asigurat ca funcționalitatea sa fie corectă.

Dezvoltatori backend: Chirteș Adrian Antonio și Bodea George

Am lucrat împreună cu Bodea George pe backend, unde am discutat cerințele aplicației și am proiectat cu acesta arhitectura aplicației.

Am realizat diferite componente fiecare, ținând cont de cerințele de pe front end. În mod concret am participat la:

-realizarea ierarhiei de clase

-dezvoltarea diverselor module pentru realizarea nevoilor unui utilizator

-implementarea și legarea unei baze de date în care pot fi adăugate tabele în mod dinamic

-adaugarea tabelelor necesare pentru menținerea unui istoric al tranzacțiilor

-implementarea unei opțiuni pentru a permite utilizatorului să seteze diferite bugete pentru aplicație

Din punct de vedere al modului de lucru, consider că activitatea s-a desfășurat în colaborare. Împreună am proiectat și implementat funcționalitățile de bază, urmând ca fiecare să expandeze asupra acestora, menținând un plan ce surprinde ce a fost implementat, ce schimbări au fost aduse diferitelor module și cerințele ce urmează să fie implementate.