### UNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

# „PocketPlanner”

# Aplicație Web Pentru Gestionarea Finanțelor Personale

### Absolvent Narcis-Alexandru Fănică

**Coordonator științific**

**Lect. Univ. Dr. Marius Iulian Mihailescu**

**București, iunie 2024**

#### Rezumat

PocketPlanner este o aplicație web de gestionare a finanțelor personale dezvoltată pentru a oferi utilizatorilor ajutorul de care au nevoie pentru a-și înțelege obiceiurile financiare. Frontend-ul aplicației este realizat folosind Angular și TailwindCSS, oferind o interfață modernă și responsive. Backend-ul este dezvoltat cu NestJS și Mongoose, asigurând o arhitectură solidă și scalabilă. Utilizatorii se pot autentifica folosind contul Google prin OAuth, iar tranzacțiile bancare sunt importate automat prin API-ul Go Cardless și sincronizate periodic prin intermediul unui pipeline de date construit cu Celery. Aplicația oferă funcționalități precum vizualizarea tranzacțiilor, crearea de bugete, categorisirea automată a tranzacțiilor și notificări în timp real și partajarea categoriilor cu un partener. PocketPlanner integrează, de asemenea, un algoritm de machine learning de tip Decision Tree pentru categorisirea automată a tranzacțiilor bancare importate.

#### Abstract

PocketPlanner is a web application for personal finance management developed to help users understand their financial habits. The frontend of the application is built using Angular and Tailwind CSS, providing a modern and responsive interface. The backend is developed with NestJS and Mongoose, ensuring a robust and scalable architecture. Users can log in using their Google account through OAuth, and bank transactions are automatically imported via the Go Cardless API and periodically synchronized through a data pipeline built with Celery. The application offers functionalities such as transaction viewing, budget creation, automatic transaction categorization, real-time notifications, and category sharing with a partner. PocketPlanner also integrates a Decision Tree machine learning algorithm for the automatic categorization of imported bank transactions.

# Cuprins

[Capitolul 1 – Introducere 5](#_Toc169202445)

[1.1 Motivație 5](#_Toc169202446)

[1.2 Preliminarii 5](#_Toc169202447)

[1.3 Nevoi pe care le satisface aplicația 5](#_Toc169202448)

[1.4. Aplicații asemănătoare 5](#_Toc169202449)

[1.5. Ce aduce nou? 5](#_Toc169202450)

[1.4 Structura lucrării 5](#_Toc169202451)

[Capitolul 2 – Concepte Teoretice 7](#_Toc169202452)

[2.1 Single Page Application (SPA) 7](#_Toc169202453)

[2.2 TypeScript 7](#_Toc169202454)

[2.3 Dependency Injection (DI) 8](#_Toc169202455)

[2.4 Swagger 9](#_Toc169202456)

[2.5 Modelul de Date Document 11](#_Toc169202457)

[2.6 OAuth și Autentificarea cu Google 12](#_Toc169202458)

[2.7 Machine Learning și Algoritmi de Clasificare 14](#_Toc169202459)

[2.8 Task-uri Asincrone și Sistemele de Cozi de Mesaje 15](#_Toc169202460)

[2.9 Open Banking 15](#_Toc169202461)

[Capitolul 3 – Utilizarea Tehnologiilor alese 17](#_Toc169202462)

[3.1 Angular 17](#_Toc169202463)

[3.2 TailwindCSS 18](#_Toc169202464)

[3.2.1 Caracteristici definitorii: 18](#_Toc169202465)

[3.3 Librării de componente: Spartan UI și Daisy UI 18](#_Toc169202466)

[3.3.1 Spartan UI 18](#_Toc169202467)

[3.3.2 Daisy UI 19](#_Toc169202468)

[3.4 NestJS 20](#_Toc169202469)

[3.4.1 Arhitectura Modulară 20](#_Toc169202470)

[3.4.2 Integrări și extensibilitate 20](#_Toc169202471)

[3.4.3 Scheduling 21](#_Toc169202472)

[3.5 Mongoose 21](#_Toc169202474)

[3.5.1 Caracteristici definitorii 22](#_Toc169202475)

[3.6 Python și Celery 22](#_Toc169202476)

[3.6.1 Celery 22](#_Toc169202477)

[3.6.2 Caracteristici definitorii 22](#_Toc169202478)

[3.7 Algoritmul Decision Tree 23](#_Toc169202479)

[3.7.1 Caracteristici definitorii 24](#_Toc169202480)

[3.7.2 Implementare în PocketPlanner 24](#_Toc169202481)

[3.8 Go Cardless pentru Importarea Datelor Bancare 25](#_Toc169202482)

[Capitolul 4 - Prezentarea aplicației 28](#_Toc169202483)

[4.1 UI/UX Design 28](#_Toc169202484)

[4.2 Design Responsive 29](#_Toc169202485)

[4.3 Notificări 30](#_Toc169202486)

[4.4 Autentificare 31](#_Toc169202487)

[4.4.1 Login și Sign-Up 31](#_Toc169202488)

[4.4.2 Autentificare cu Google (OAuth) 32](#_Toc169202489)

[4.4.3 Actualizarea informațiilor inițiale 32](#_Toc169202490)

[4.5 Dashboard 33](#_Toc169202491)

[4.6 Gestionarea Conturilor 34](#_Toc169202492)

[4.6.1 Conturi Cash 34](#_Toc169202493)

[4.6.2 Sincronizare Bancară 35](#_Toc169202494)

[4.6.3 Sincronizare automată prin pipeline 36](#_Toc169202495)

[4.7 Categorii 36](#_Toc169202496)

[4.7.1 Gestionarea Categoriilor 36](#_Toc169202497)

[4.7.2 Parteneriate 37](#_Toc169202498)

[4.8 Tranzacții 39](#_Toc169202499)

[4.8.1 Tabela de Tranzacții: 40](#_Toc169202500)

[4.8.2 Gestionarea tranzacțiilor cash: 40](#_Toc169202501)

[4.8.3 Gestionarea tranzacțiilor bancare: 41](#_Toc169202502)

[4.8.4 Grafice: 42](#_Toc169202503)

[4.9 Bugete 43](#_Toc169202504)

[4.9.1 Funcționalități principale: 43](#_Toc169202505)

[4.9.2 Grafice: 44](#_Toc169202506)

[Capitolul 5 - Concluzii și Perspective 46](#_Toc169202507)

[5.1 Concluzii 46](#_Toc169202508)

[5.2 Perspective de Dezvoltare 46](#_Toc169202509)

[5.2.1 Funcționalități Adiționale 46](#_Toc169202510)

[5.2.2 Îmbunătățiri de Performanță și Securitate 47](#_Toc169202511)

[Bibliografie 48](#_Toc169202512)

# Capitolul 1 – Introducere

## Motivație

Analiza și gestionarea finanțelor personale a fost dintotdeauna un subiect de interes pentru mine. În era digitală, o aplicație ce facilitează acest proces a devenit o necesitate pentru a menține o evidență clară a veniturilor și cheltuielilor. Aceste aplicații oferă utilizatorilor un control mai bun asupra finanțelor lor, ajutându-i să-și atingă obiectivele financiare. Experimentând cu diferitele aplicații din această nișă am realizat că există anumite elemente ce lipsesc sau ce ar putea fi îmbunătățite pentru a crea o soluție ușor de utilizat, coerentă și cu toate funcționalitățile de care ar putea avea nevoie un utilizator de la o astfel de aplicație. PocketPlanner intenționează să rezolve aceste aspecte, dezvoltată pentru a facilita gestionarea tranzacțiilor, bugetelor și conturilor bancare într-un mod facil de utilizat, eficient și sigur.

## Preliminarii

PocketPlanner este o aplicație web pentru gestionarea finanțelor personale, dezvoltată folosind o suită de tehnologii moderne pentru a oferi o experiență de utilizare optimă. Aplicația include funcționalități precum gestionarea tranzacțiilor (atât cash cât și bancare), categorii de tranzacții, conturi cash și bancare și bugete lunare pentru fiecare categorie. Integrând servicii externe și un pipeline de date pentru actualizarea automată a conturilor tranzacțiilor bancare și utilizând algoritmi de machine learning pentru categorisirea acestor tranzacții, PocketPlanner reprezintă un instrument puternic pentru utilizatorii săi.

## Nevoi pe care le satisface aplicația

Aplicația PocketPlanner răspunde nevoii de organizare și control asupra finanțelor personale, oferind:

* Un mod centralizat de a gestiona tranzacțiile și conturile cash și bancare.
* Funcționalități de categorisire manuală a tranzacțiilor, precum și automată folosind algoritmi de machine learning ce învață din preferințele fiecărui utilizator.
* Sincronizarea automată a tranzacțiilor bancare prin integrarea cu serviciul Go Cardless.
* Posibilitatea de a partaja informații financiare relevante, precum cheltuieli și bugete cu un alt utilizator prin funcția de parteneriat.

## Aplicații asemănătoare

Există mai multe aplicații de gestionare a finanțelor personale disponibile pe piață, fiecare cu caracteristici distincte. Printre acestea se numără:

* Mint: Oferă gestionarea bugetelor, urmărește cheltuielile și oferă sugestii de economisire. Dezavantajul acesteia este acoperirea bancară limitată, clienții din România neavând acces la datele lor bancare.
* YNAB (You Need A Budget): Focusată pe planificarea bugetelor și educația financiară, însă poate fi excesiv de complexă pentru utilizatorii noi.
* Wallet: Oferă funcționalități de gestionare a conturilor, tranzacțiilor și bugetelor și oferă o acoperire bancară ce include majoritatea băncilor din România, dar interfața este greu de folosit și nu oferă categorisire automată a tranzacțiilor.

## Ce aduce nou?

PocketPlanner aduce câteva inovații importante în domeniul gestionării finanțelor personale:

* **Integrarea cu API-ul Go Cardless:** Aceasta permite sincronizarea automată a conturilor și tranzacțiilor bancare, oferind o actualizare continuă a datelor financiare fără intervenția utilizatorului.
* **Algoritm de machine learning:** Utilizarea unui model Decision Tree pentru categorisirea automată a tranzacțiilor pe baza istoricului și preferințelor utilizatorului.
* **Pipeline de date pentru actualizarea tranzacțiilor:** Un sistem automatizat, construit cu Python și Celery, verifică periodic și actualizează soldurile conturilor și tranzacțiile pentru a menține aplicația actualizată fără a încărca excesiv serverele.
* **Funcționalitatea de parteneriat:** Permite utilizatorilor să creeze un parteneriat cu un alt utilizator prin care să partajeze anumite categorii de tranzacții, facilitând urmărirea bugetelor comune.

## Structura lucrării

Lucrarea este satructurată pentru a oferi o imagine completă și detaliată asupra dezvoltării și funcționalităților aplicației PocketPlanner. Structura include:

* **Capitolul 1 - Introducere:** Prezentarea motivației, contextului, obiectivelor și inovațiilor aduse de PocketPlanner.
* **Capitolul 2 – Concepte teoretice:** Prezentarea conceptelor teoretice utilizate de diferitele module și tehnologii ale aplicației.
* **Capitolul 3 – Utilizarea tehnologiilor alese:** Descrierea tehnologiilor utilizate în dezvoltarea aplicației, precum Angular, NestJS, TailwindCSS, Mongoose, etc.
* **Capitolul 4 - Prezentarea aplicației:** Descrierea interfeței aplicației, funcționalităților și fluxurilor principale și a modului de utilizare a aplicației.
* **Capitolul 5 - Concluzii și perspective:** Concluziile dezvoltării și perspectivele de viitor pentru extinderea funcționalităților.

# Capitolul 2 – Concepte Teoretice

## 

## Single Page Application (SPA)

Single Page Applications (SPA) sunt aplicații web care interacționează cu utilizatorul printr-o singură pagină HTML, actualizându-se dinamic pe măsură ce utilizatorul interacționează cu aplicația. Spre deosebire de aplicațiile web tradiționale, multi-pagină, care necesită reîncărcarea întregii pagini pentru fiecare acțiune, aplicațiile single-page încarcă resursele necesare la început și folosesc JavaScript pentru a manipula DOM-ul și a face cereri asincrone către server, îmbunătățind astfel experiența utilizatorului și performanța aplicației.

#### Avantaje ale SPA:

* **Performanță îmbunătățită:** Single Page Applications reduc timpul de încărcare al paginilor, deoarece majoritatea resurselor sunt încărcate o singură dată la început.
* **Experiență de utilizare fluidă:** Utilizatorii beneficiază de o interacțiune mai rapidă și fără întreruperi, fără reîncărcări complete ale paginii.
* **Navigare mai rapidă:** Prin utilizarea apelurilor asincrone către server pentru a încărca datele, doar părțile necesare ale paginii sunt reîncărcate, ceea ce face ca navigarea să fie foarte rapidă.
* **Arhitectură modulară:** Permite dezvoltarea unor aplicații cu o arhitectură bine definită, unde front-end-ul și back-end-ul sunt clar separate.

## TypeScript

TypeScript este un limbaj de programare dezvoltat de Microsoft, reprezentând un superset al JavaScript, extinzându-l prin adăugarea de tipuri statice. Codul TypeScript este compilat în JavaScript, fiind complet compatibil cu proiectele JavaScript existente.

#### Avantaje ale TypeScript:

* **Tipare statice:** Permite dezvoltatorilor să definească tipurile variabilelor, parametrilor și funcțiilor, reducând astfel erorile ce pot apărea la rulare și îmbunătățind claritatea codului.
* **Suport în mediile de dezvoltare:** TypeScript este bine integrat în toate mediile integrate de dezvoltare, precum VS Code sau Webstorm, oferind funcționalități de care dezvoltatorii au nevoie, cum ar fi IntelliSense, refactorizare automată și completare de cod.
* **Compatibilitate:** Codul TypeScript este compilat în JavaScript, ceea ce înseamnă că poate fi în orice mediu în care poarte rula JavaScript, precum browser-ul sau alte runtime-uri Javascript, precum node.js sau deno.
* **OOP:** Suportă concepte de programare orientată pe obiecte, cum ar fi clase, interfețe și moștenire, facilitând scrierea de cod modular și reutilizabil și făcându-l o alegere ideală pentru framework-urile ce au la bază programarea orientată pe obiect.

#### Utilizări în Angular și NestJS:

* **Angular:** Angular este scris în TypeScript, beneficiind de tipare statice și facilitățile de dezvoltare oferite de TypeScript.
* **NestJS:** NestJS este un framework de back-end care utilizează TypeScript pentru a asigura o scriere clară și tipizată a codului server-side.

## Dependency Injection (DI)

Dependency Injection (DI) este un design pattern utilizat pentru a gestiona dependențele dintre obiecte, permițând injectarea de dependențe necesare unui obiect din exterior în loc de crearea lor în interiorul obiectului. Dependency Injection este larg utilizat în framework-urile ce au la bază programarea orientată pe obiect, precum Angular și NestJS.

#### Avantaje ale Dependency Injection:

* **Separation of concerns:** Dependency Injection separă comportamentul aplicației de modul în care instanțele claselor sunt create și rezolvate, rezultând un cod mai curat și mai ușor de întreținut.
* **Facilitează testarea:** Dependențele pot fi ușor înlocuite cu mock-uri în timpul testării, făcând testarea unitară a codului mai ușoară și mai eficientă.
* **Reutilizarea codului:** Dependențele pot fi reutilizate în diferite părți ale aplicației fără a fi necesară refactorizarea codului.

#### Implementare în Angular și NestJS:

* **Angular:** Angular folosește DI pentru a injecta servicii în componente, directive, pipe-uri și alte servicii, facilitând gestionarea și reutilizarea codului.
* **NestJS:** NestJS utilizează DI pentru a injecta servicii în controlere, middleware și alți provideri, asigurând un cod modular și ușor de întreținut.

constructor(

private *nordigenService*: NordigenService,

private *accountsService*: BankAccountsService,

private *transactionsService*: BankTransactionsService,

) {}

### 

## Swagger

Swagger este un framework open-source utilizat pentru construirea, documentarea și testarea API-urilor RESTful. Swagger permite dezvoltatorilor să descrie structura API-urilor lor astfel încât dezvoltatorii ce au nevoie să le utilizeze să le poată înțelege. Documentația generată automat de Swagger oferă o interfață interactivă prin care dezvoltatorii pot explora și testa API-urile.

#### Caracteristici:

* **Documentație Interactivă:** Swagger oferă o interfață web interactivă care permite utilizatorilor să exploreze și să testeze endpoint-urile API direct din documentație.
* **Standardizare:** Utilizând Swagger, API-urile sunt documentate într-un format standardizat, ușor de înțeles și de utilizat de către alte aplicații și dezvoltatori.
* **Generare automată:** Swagger poate genera documentația automat pe baza adnotărilor din cod, reducând astfel efortul manual necesar pentru documentare.

#### Integrarea Swagger în NestJS:

* **Instalare și configurare:** Modulul dedicat @nestjs/swagger oferă suport pentru documentarea API-ului utilizând Swagger prin integrarea acestuia în modului principal al aplicației.
* **Adnotări:** Dezvoltatorii adaugă adnotări Swagger în controlerele și DTO-urile (Data Transfer Objects) din aplicație pentru a descrie structura și funcționalitatea endpoint-urilor API-ului.
* **Generare Documentație:** Swagger generează documentația API pe baza adnotărilor, oferind o interfață interactivă la un endpoint dedicat, de obicei /api.

export class AppModule {

static configureSwagger(*app*) {

const options = new DocumentBuilder()

.setTitle('Thesis API')

.setVersion('1')

.addBearerAuth()

.build();

const document = SwaggerModule.createDocument(*app*, options);

SwaggerModule.setup('api', *app*, document);

}

}

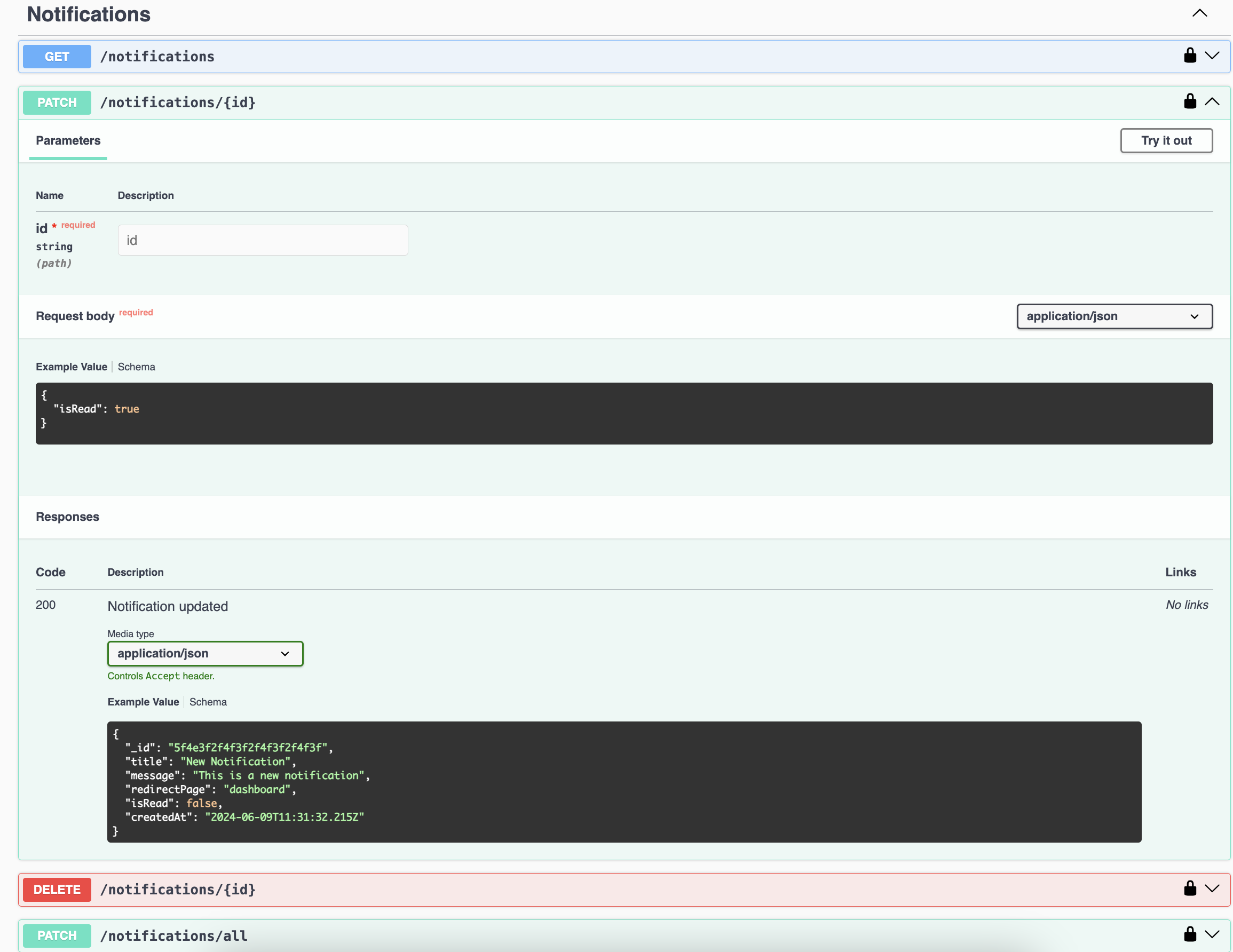


Figura 1: Interfața Swagger

## Modelul de Date Document

Un model de date document este o abordare de stocare a datelor în care datele sunt stocate și gestionate în structuri asemănătoare documentelor, precum JSON sau BSON. MongoDB este un exemplu popular de bază de date documentară. Acest model este intuitiv pentru dezvoltatorii aplicației și foarte flexibil, permițând modelului de date să evolueze cu ușurință odată cu nevoile aplicației.

#### Caracteristici:

* **Flexibilitate:** Structurile de date sunt flexibile și permit stocarea de date nestandardizate, fără strictețea impusă de o bază de date relațională.
* **Scalabilitate:** Sistemele de baze de date documentare sunt de foarte scalabile, fiind capabile să gestioneze volume mari de date distribuite pe mai multe servere.
* **Interogare complexă:** Permite interogări complexe și agregări de date, suportând operațiuni de tip CRUD (Create, Read, Update, Delete) foarte eficiente.

**Avantajele MongoDB în comparație cu bazele de date relaționale:**

* **Schema flexibilă:** MongoDB permite adăugarea și eliminarea de câmpuri în documente fără a afecta alte documente din colecție. Această flexibilitate este ideală pentru aplicațiile care necesită schimbări rapide ale structurii de date.
* **Performanță:** MongoDB este optimizat pentru stocarea și accesarea rapidă a datelor nestandardizate, ceea ce îl face potrivit pentru aplicații cu cerințe de performanță ridicată.
* **Scalabilitate:** MongoDB poate scala orizontal prin distribuirea datelor pe mai multe servere (sharding), asigurând gestionarea eficientă a unor volume mari de date.
* **Agregare avansată:** MongoDB oferă un framework puternic de agregare, care permite realizarea de interogări și procesări de date complexe.
* **Manipulare nativă a datelor JSON:** MongoDB stochează datele în format BSON (o extensie binară a JSON), facilitând integrarea cu aplicațiile care folosesc JSON pentru transferul de date.

## OAuth și Autentificarea cu Google

OAuth (Open Authorization) este un protocol de autorizare care permite utilizatorilor să permită aplicațiilor terțe să acceseze resursele lor de pe un alt site web fără a împărtăși credențiale lor (parole). OAuth este utilizat pe scară largă pentru autentificarea cu furnizori de identitate precum Google, Github, și alții.

#### Caracteristici:

* **Securitate:** OAuth oferă un mod sigur pentru aplicații de a accesa resursele utilizatorilor fără a compromite securitatea credențialelor.
* **Comoditate:** Utilizatorii pot folosi conturile existente pentru a se autentifica în alte aplicații, reducând nevoia de a crea și reține noi seturi de acreditări.
* **Scalabilitate:** Permite dezvoltatorilor să construiască aplicații care pot accesa în mod sigur resursele protejate ale utilizatorilor.

#### Implementarea OAuth cu Google în PocketPlanner:

* **Autentificare:** Utilizatorii PocketPlanner se pot autentifica folosind contul lor Google. Procesul începe prin redirecționarea utilizatorului către pagina de autentificare Google.
* **Autorizare:** După autentificare, utilizatorul este întrebat dacă dorește să permită PocketPlanner să acceseze informațiile sale de profil.
* **Token de acces:** Dacă utilizatorul acceptă, Google emite un access token care este utilizat de PocketPlanner pentru a accesa resursele ce necesită autorizare.

@Injectable()

export class GoogleStrategy extends PassportStrategy(Strategy, 'google') {

constructor(

@InjectModel(User.name) private *userModel*: Model<User>,

) {

*super*({

clientID: process.env.GOOGLE\_CLIENT\_ID,

clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,

callbackURL: 'http://localhost:3000/auth/google/callback',

scope: ['email', 'profile'],

});

}

async validate(

*accessToken*: string,

*refreshToken*: string,

*profile*: any,

*done*: VerifyCallback,

): Promise<any> {

const { name, emails, id } = *profile*;

const user = await *this*.userModel.findOne({ googleId: id });

if (!user) {

const user = await *this*.userModel.create({

googleId: id,

email: emails[0].value,

username: name.givenName,

});

}

done(null, user);

}

}

## Machine Learning și Algoritmi de Clasificare

Machine Learning (ML) este o subset al inteligenței artificiale care permite sistemelor să învețe și să facă predicții pe baza datelor. O ramură esențiale a ML este utilizarea algoritmilor de clasificare pentru a categorisi datele în funcție de anumite caracteristici.

#### Decision Tree:

Algoritmul Decision Tree este un algoritm de clasificare precis și intuitiv, care folosește un arbore de decizie pentru a efectua predicții pe baza caracteristicilor de intrare.

* **Principiu de funcționare:** Algoritmul construiește un model de arbore de decizie, unde fiecare nod intern reprezintă un test pe o caracteristică, fiecare ramură reprezintă rezultatul testului, iar fiecare nod frunză reprezintă o clasă prezisă.
* **Avantaje:** Decision Tree este ușor de înțeles și interpretat și poate gestiona atât date numerice cât și categorice.
* **Performanță:** Spre deosebire de alți algoritmi, precum regresia logistică, algoritmul Decision Tree poate captura relații non-liniare între caracteristici și etichete de clasă, oferind adesea performanțe bune pe date complexe.

#### TF-IDF Vectorizer:

TF-IDF (Term Frequency-Inverse Document Frequency) este o tehnică de reprezentare a textului utilizată în procesarea limbajului natural. TF-IDF reflectă cât de important este un cuvânt pentru un document și este folosit pentru a transforma textul în vectori numerici ce pot fi apoi procesați de algoritmii de machine learning.

#### Principiul de funcționare:

* **Term Frequency (TF):** Măsoară frecvența unui termen într-un document. Este calculat ca numărul de apariții ale termenului într-un document, împărțit la numărul total de termeni din document.
* **Inverse Document Frequency (IDF):** Măsoară importanța unui termen în întregul corpus. Este calculat ca logaritmul numărului total de documente împărțit la numărul de documente care conțin termenul respectiv.
* **TF-IDF:** Este produsul dintre TF și IDF, rezultând o valoare care indică importanța unui termen în contextul unui document și al întregului corpus.

#### Avantaje:

* **Relevanță:** TF-IDF acordă greutate mai mare termenilor rari și relevanți și greutate mai mică termenilor comuni.
* **Normalizare:** Ajută la normalizarea datelor text, facilitând utilizarea acestora în algoritmi de învățare automată.
* **Simplicitate:** Este ușor de implementat și interpretat.

## Task-uri Asincrone și Sistemele de Cozi de Mesaje

Task-urile asincrone sunt operațiuni care sunt executate în fundal, fără a bloca fluxul principal de execuție. Sistemele de cozi de mesaje, precum Celery, sunt utilizate pentru a gestiona și a distribui aceste task-uri către workeri care le execută.

#### Caracteristici:

* **Distribuție:** Task-urile sunt distribuite către workeri care le execută în fundal, permițând astfel execuția paralelă și asigurând scalabilitatea aplicației.
* **Reîncercare:** Task-urile eșuate pot fi reîncercate automat după un anumit interval de timp, asigurând fiabilitatea execuției.
* **Monitorizare:** Permite monitorizarea și gestionarea cozii de mesaje și a task-urilor, oferind instrumente pentru vizualizarea și gestionarea acestora în timp real.

## Open Banking

Open Banking reprezintă o abordare în industria financiară care promovează partajarea securizată și autorizată a datelor financiare ale utilizatorilor prin intermediul unor API-uri standardizate. Acest lucru permite dezvoltatorilor de aplicații să construiască servicii financiare inovatoare cu ușurință, facilitând transferul securizat al informațiilor financiare între diverse platforme și aplicații.

#### Caracteristici:

* **Acces Deschis la Date:** Utilizatorii pot permite aplicațiilor terțe să acceseze datele lor financiare în mod securizat, prin intermediul unor API-uri standardizate.
* **Securitate și Confidențialitate:** Open Banking utilizează protocoale de securitate avansate pentru a proteja datele utilizatorilor și pentru a asigura confidențialitatea acestora.
* **Inovație în Servicii Financiare:** Permite dezvoltatorilor să creeze aplicații și servicii financiare inovatoare, cum ar fi agregatoare de conturi, instrumente de bugetare și soluții de plată.

#### Integrarea Open Banking în PocketPlanner prin Serviciul Go Cardless:

* **Acces la Date Bancare:** Serviciul Go Cardless Account Data reprezintă un REST API pentru agregarea datelor bancare oferite prin Open Banking de către băncile din Uniunea Europeană și permite PocketPlanner să acceseze datele conturilor bancare și ale tranzacțiilor utilizatorilor, oferindu-le o vedere de ansamblu asupra situației lor financiare.
* **Sincronizare Automată:** Utilizatorii pot conecta mai multe conturi bancare la PocketPlanner, iar aplicația va sincroniza automat tranzacțiile prin intermediul pipeline-ului de date.
* **Analiză și Categorisire:** Datele bancare împreună cu preferințele utilizatorilor de categorisire sunt utilizate pentru a analiza și categorisi tranzacțiile ulterioare prin intermediul algoritmilor de machine learning, oferindu-le utilizatorilor informații valoroase despre cheltuielile și veniturile lor.

# Capitolul 3 – Utilizarea Tehnologiilor alese

## Angular

Angular este un framework open-source pentru dezvoltarea aplicațiilor web de tip SPA, dezvoltat de Google. Angular facilitează dezvoltarea aplicațiilor complexe printr-o arhitectură bazată pe componente organizate în module și oferă suport pentru data binding bidirecțional al datelor, dependency injection, rutare și multe alte funcționalități.

* + 1. **Caracteristici definitorii:**
* **Modularitate:** Angular permite împărțirea aplicației în module distincte, fiecare având propriile componente, servicii și alte resurse.
* **Data Binding:** Angular suportă binding bidirecțional al datelor, sincronizând automat modelul de date și view-ul.
* **Dependency Injection:** Angular utilizează un sistem de injectare a dependențelor, ce permite reutilizarea serviciilor în diferitele module ale aplicației, îmbunătățind flexibilitatea aplicației și facilitând abilitatea de testare a acesteia.
* **Directives:** Angular oferă directive pentru a extinde funcționalitatea elementelor HTML existente sau pentru a crea elemente HTML personalizate.

@ViewChild('categoryDialog') categoryDialog!: HlmDialogComponent;

@ViewChild('accountDialog') accountDialog!: HlmDialogComponent;

constructor(

private *authService*: AuthService,

private *categoriesService*: CategoriesService,

private *router*: Router

) {}

ngOnInit() {

*this*.loadCategories();

}

loadCategories() {

*this*.categoriesService.getDefaultCategories().subscribe({

next: (*categories*) => {

*this*.categories = *categories*;

},

error: (*err*) => {

console.error('Failed to load categories', *err*);

},

});

}

## TailwindCSS

TailwindCSS este un framework CSS utilitar care permite dezvoltatorilor să construiască interfețe responsive și personalizabile într-un mod rapid și eficient. TailwindCSS evită abordarea convențională de stilizare a unei aplicații web prin scrierea de CSS și oferă clase utilitare ce pot fi aplicate în mod direct diferitelor elementelor aplicației.

### Caracteristici definitorii:

* **Utility-First:** TailwindCSS oferă o gamă largă de clase utilitare predefinite pentru a aplica stiluri direct în HTML, oferind dezvoltatorului abilitatea de a configura aspecte precum așezarea în pagină, dimensiunea elementelor, fonturi, culori și efecte.
* **Personalizare:** Framework-ul este extrem de configurabil, permițând personalizarea temelor, paletelor de culori și a altor stiluri globale.
* **Responsive Design:** TailwindCSS facilitează crearea de layout-uri responsive prin utilizarea de breakpoint-uri pentru diferite dimensiuni ale ecranului ce permit aplicarea condițională a diferitelor clase utilitare.

<div class="flex flex-col md:flex-row justify-between gap-4">

<div class="md:w-8/12 w-full">

<div class="flex flex-col justify-between gap-4 sm:flex-row">

## Librării de componente: Spartan UI și Daisy UI

Pentru a accelera procesul de dezvoltare și a asigura un design uniform și modern, PocketPlanner utilizează librăriile de componente Spartan UI și Daisy UI, ambele bazate pe TailwindCSS.

### Spartan UI

Spartan UI oferă un set de componente UI predefinite, stilizate și nestilizate, construite cu Angular, care pot fi integrate ușor în aplicațiile dezvoltate cu TailwindCSS. Avantajul oferit de către Spartan UI față de alte librării asemănătoare de componente îl reprezintă faptul că utilizatorul are acces deplin la codul sursă al componentei și îl poate modifica pentru a se potrivi nevoilor aplicației sale.

export class HlmButtonDirective {

public readonly userClass = input<ClassValue>('', { alias: 'class' });

private readonly \_settableClass = signal<ClassValue>('');

protected \_computedClass = computed(() =>

hlm(buttonVariants({ variant: *this*.\_variant(), size: *this*.\_size() }), *this*.\_settableClass(), *this*.userClass()),

);

setClass(*value*: ClassValue) {

*this*.\_settableClass.set(*value*);

}

private readonly \_variant = signal<ButtonVariants['variant']>('default');

@Input()

set variant(*variant*: ButtonVariants['variant']) {

*this*.\_variant.set(*variant*);

}

private readonly \_size = signal<ButtonVariants['size']>('default');

@Input()

set size(*size*: ButtonVariants['size']) {

*this*.\_size.set(*size*);

}

}

### 

### Daisy UI

Daisy UI este o altă librărie de componente care oferă o colecție variată de elemente UI stilizate, compatibile cu TailwindCSS, facilitând astfel crearea unei interfețe utilizator atractive și coerente. Componentele oferite de Daisy UI sunt create în totalitate cu HTML și CSS, fiind astfel independente de framework-ul de Frontend utilizat și ușor de integrat în orice aplicație dezvoltată cu TailwindCSS

<div class="card w-full max-w-sm shadow-2xl bg-base-100">

## NestJS

NestJS este un framework progresiv pentru construirea aplicațiilor server-side eficiente și scalabile, utilizând TypeScript. Inspirat de arhitectura modulară a Angular, NestJS oferă un mediu de dezvoltare coerent și puternic, bazat pe Typescript și ce oferă dezvoltatorilor toate instrumentele necesare pentru construirea unei aplicații complexă și sigură.

### Arhitectura Modulară

NestJS folosește o arhitectură modulară pentru a organiza codul într-un mod care să fie ușor de gestionat și extins. Fiecare funcționalitate sau grup de funcționalități poate fi organizată într-un modul separat, care include toate componentele necesare (module importate, controlere, servicii, entități, etc.).

@Module({

imports: [

JwtModule.register({

global: true,

secret: process.env.JWT\_SECRET,

signOptions: { expiresIn: '1d' },

}),

MongooseModule.forFeature([{ name: 'User', schema: UserSchema }]),

CategoriesModule,

AccountsModule,

],

controllers: [AuthController],

providers: [AuthService, GoogleStrategy],

})

export class AuthModule {}

### 

### Integrări și extensibilitate

NestJS oferă suport integrat pentru numeroase librării utile în dezvoltarea unui backend robust, printre cele utilizate in PocketPlanner regăsindu-se Mongoose, Swagger, JWT, validare, scheduling și OAuth. În plus, NestJS oferă suport pentru middleware, interceptori și guards, facilitând extinderea funcționalităților de bază.

@Injectable()

export class AuthGuard implements CanActivate {

constructor(private *jwtService*: JwtService) {}

async canActivate(*context*: ExecutionContext): Promise<boolean> {

const request = *context*.switchToHttp().getRequest();

const token = *this*.extractTokenFromHeader(request);

if (!token) {

throw new UnauthorizedException();

}

try {

const payload = await *this*.jwtService.verifyAsync(token, {

secret: process.env.JWT\_SECRET,

});

request['user'] = payload;

} catch {

throw new UnauthorizedException();

}

return true;

}

private extractTokenFromHeader(*request*: Request): string | undefined {

const [type, token] = *request*.headers.authorization?.split(' ') ?? [];

return type === 'Bearer' ? token : undefined;

}

}

### Scheduling

### NestJS permite gestionarea task-urilor programate prin decoratorul @Cron. Acesta poate fi folosit pentru a programa task-uri să ruleze la intervale specifice. În cadrul PocketPlanner, funcția de scheduling este utilizată în 2 locuri: actualizarea zilnică a ratelor de exchange și actualizarea lunară a bugetelor utilizatorilor.

@Cron(CronExpression.EVERY\_DAY\_AT\_MIDNIGHT)

async updateCurrencyExchangeRates() {

const baseCurrency = 'RON';

const currencies = await *this*.currencyModel.find().exec();

const currencyNames = currencies.map((*currency*) => *currency*.name);

const rates = await *this*.getRates(currencyNames, baseCurrency);

rates[`${baseCurrency}${baseCurrency}`] = 1;

for (const currency of currencies) {

const exchangeRate = rates[`${baseCurrency}${currency.name}`];

currency.exchangeRate = exchangeRate;

await currency.save();

}

console.log('Updated exchange rates:', rates);

}

## Mongoose

Mongoose este o librărie de modelare a datelor pentru MongoDB și Node.js. Oferă un API puternic și ușor de integrat în backend-ul aplicației pentru a interacționa cu MongoDB, facilitând definirea schemelor și validarea datelor.

### Caracteristici definitorii

* **Schema-based:** Mongoose permite definirea schemelor de date care sunt utilizate pentru validarea și modelarea documentelor MongoDB.
* **Middleware:** Oferă suport pentru middleware prin pre și post hooks, permițând executarea logicii personalizate înainte și după diferite operațiuni ce interacționează cu baza de date.
* **Validare:** Include mecanisme robuste de validare a datelor în procesul de actualizare a acestora, asigurând integritatea datelor stocate.

export type CurrencyDocument = mongoose.HydratedDocument<Currency>;

@Schema()

export class Currency {

@Prop({ required: true, enum: CurrencyEnum })

name: CurrencyEnum;

@Prop({ required: true })

exchangeRate: number;

}

export const CurrencySchema = SchemaFactory.createForClass(Currency);

## Python și Celery

PocketPlanner utilizează Python pentru a implementa un pipeline de date care verifică periodic și actualizează soldurile conturilor și tranzacțiile bancare ale utilizatorilor și pentru a antrena modelele de tip Decision Tree ce sunt utilizate pentru categorisirea automată a tranzacțiilor noi. Celery este folosit pentru gestionarea task-urilor asincrone în acest pipeline.

### Celery

Celery este o librărie open-source pentru gestionarea task-urilor asincrone, folosind cozi de mesaje pentru a distribui task-urile către workerii care le execută în fundal.

### Caracteristici definitorii

* **Distribuție:** Permite distribuirea task-urilor pe mai mulți workeri, scalând astfel capacitatea de procesare.
* **Reîncercare automată:** Task-urile eșuate pot fi reîncercate automat după un anumit interval de timp.
* **Monitorizare:** Oferă instrumente pentru monitorizarea task-urilor și gestionarea cozii de mesaje.

app = Celery("tasks")

app.config\_from\_object("celeryconfig")

@app.task

def update\_all\_users\_data():

users = db.users.find()

for user in users:

requisitions = db.requisitions.find({"user": user["\_id"]})

for requisition in requisitions:

accounts = db.bankaccounts.find(

{

"user": user["\_id"],

"requisition": requisition["\_id"],

"isDeleted": False,

}

)

for account in accounts:

update\_user\_data(user["\_id"], account["\_id"], requisition["\_id"])

broker\_url = "redis://localhost:6379/0"

beat\_schedule = {

"update-user-data-every-hour": {

"task": "tasks.update\_all\_users\_data",

"schedule": crontab(*minute*="0", *hour*="\*"), *# Every hour*

},

"retrain-models-daily": {

"task": "tasks.retrain\_models",

"schedule": crontab(*minute*=0, *hour*=0), *# Every day at midnight*

},

}

### 

## Algoritmul Decision Tree

Pentru categorisirea automată a tranzacțiilor, PocketPlanner utilizează un model de machine learning bazat pe algoritmul Decision Tree. Acest algoritm este eficient pentru probleme de clasificare și este utilizat pentru a învăța tiparele din tranzacțiile categorisite ale utilizatorului pentru a prezice categoriile în care se vor încadra tranzacțiile noi.

### Caracteristici definitorii

* **Simplicitate și eficiență:** Algoritmul Decision Tree este simplu de implementat și foarte eficient pentru clasificarea textului și a altor date discrete.
* **Performanță:** Este cunoscut pentru performanțele sale bune chiar și cu seturi de date mici.
* **Scalabilitate:** Poate gestiona ușor un număr mare de caracteristici, fiind potrivit pentru clasificarea tranzacțiilor financiare.

### Implementare în PocketPlanner

Algoritmul Decision Tree este implementat în pipeline-ul de date pentru a categorisi noile tranzacții pe baza istoricului de tranzacții al utilizatorului. Pentru fiecare conexiune la o instituție bancară pe care utilizatorul o are, un model de machine learning va fi antrenat folosind tranzacțiile utilizatorului din acea conexiune, urmând ca noile tranzacții din conexiunea respectivă să fie categorisite automat pe baza modelului, îmbunătățind astfel experiența utilizatorului prin automatizarea procesului de categorisire.

def train\_requisition\_model(*user\_id*, *requisition\_id*):

transactions = db.transactions.find(

{

"user": *user\_id*,

"requisition": *requisition\_id*,

"isDeleted": {"$ne": True},

"category": {"$exists": True},

}

)

data = [(t["details"], str(t["category"])) for t in transactions]

if not data:

return None

descriptions, categories = zip(\*data)

model = make\_pipeline(TfidfVectorizer(), DecisionTreeClassifier())

model.fit(descriptions, categories)

model\_path = f"user\_models/{*user\_id*}\_{*requisition\_id*}\_model.joblib"

os.makedirs(os.path.dirname(model\_path), *exist\_ok*=True)

joblib.dump(model, model\_path)

def classify\_transaction(*user\_id*, *requisition\_id*, *details*):

model\_path = f"user\_models/{*user\_id*}\_{*requisition\_id*}\_model.joblib"

if not os.path.exists(model\_path):

return None

model = joblib.load(model\_path)

category = model.predict([*details*])[0]

print(category)

return category

## Go Cardless pentru Importarea Datelor Bancare

Go Cardless Account Data este o platformă care permite aplicațiilor să colecteze informații despre conturile și tranzacțiile bancare ale utilizatorilor lor prin intermediul Open Banking. În contextul aplicației PocketPlanner, API-ul Go Cardless este utilizat pentru a importa date bancare ale utilizatorilor, facilitând gestionarea finanțelor personale într-un mod automatizat și centralizat.

#### Caracteristici:

* **Acces la Date Bancare:** API-ul Go Cardless permite aplicației să acceseze informații despre conturile bancare ale utilizatorilor și să importe tranzacțiile bancare.
* **Securitate și Confidențialitate:** Go Cardless folosește protocoale de securitate avansate pentru a proteja datele utilizatorilor, asigurând confidențialitatea și integritatea acestora.
* **Acoperire globală:** API-ul este oferit de Go Cardless oferă integrarea cu peste 300 de instituții bancare, asigurând acoperirea necesară astfel încât fiecare utilizator al aplicației să aibă acces la datele sale bancare.

#### Implementarea Go Cardless în PocketPlanner:

1. **Autentificare și Autorizare:** Utilizatorii se autentifică în PocketPlanner și apoi pot crea o conexiune bancară selectând din interfața aplicației instituția la care doresc să se conecteze.
2. **Conectare la conturi bancare:** Utilizatorii își dau consimțământul în pagina de autentificare specifică instituției la care se conectează și selectează conturile la care doresc să ofere acces aplicației.
3. **Importarea tranzacțiilor și a conturilor:** După finalizarea procesului de oferire a consimțământului, utilizatorul este redirecționat în aplicație unde se identifică faptul că statusul conexiunii a devenit activ, iar aplicația importă conturile și tranzacțiile bancare ale utilizatorului.
4. **Actualizări ulterioare:** Utilizând pipeline-ul de date, soldurile conturilor și tranzacțiile bancare ce aparțin acestora sunt actualizate periodic, iar tranzacțiile sunt categorisite automat utilizând algoritmul Decision Tree, aceste actualizări fiind trimise in backend-ul aplicației pentru stocarea lor în baza de date și afișarea lor în interfață.

A screenshot of a phone

Description automatically generated

Figura 2: Endpoint-urile oferite de API-ul Go Cardless

# Capitolul 4 - Prezentarea aplicației

## UI/UX Design

Design-ul UI/UX este esențial pentru a asigura o experiență de utilizare plăcută și eficientă. PocketPlanner folosește o temă minimalistă, cu culori vibrante și o interfață intuitivă.

#### Elemente de Design:

* **Tematica de culori:** Aplicația utilizează o paletă de culori vibrantă pentru a evidenția diferitele elemente ale interfeței utilizator. Culorile primare includ nuanțe de albastru, roz, turcoaz și gri.
* **Tipografie:** Fonturi clare și lizibile sunt utilizate pentru a asigura că utilizatorul poate citi cu ușurință textul tuturor componentelor aplicației
* **Componente UI:** Aplicația folosește componente pre-stilizate din librăriile Spartan UI și Daisy UI, bazate pe TailwindCSS, pentru a asigura consistență în estetica aplicației și o aparență minimalistă și modernă.

#### Elemente de Îmbunătățire a Experienței Utilizatorului:

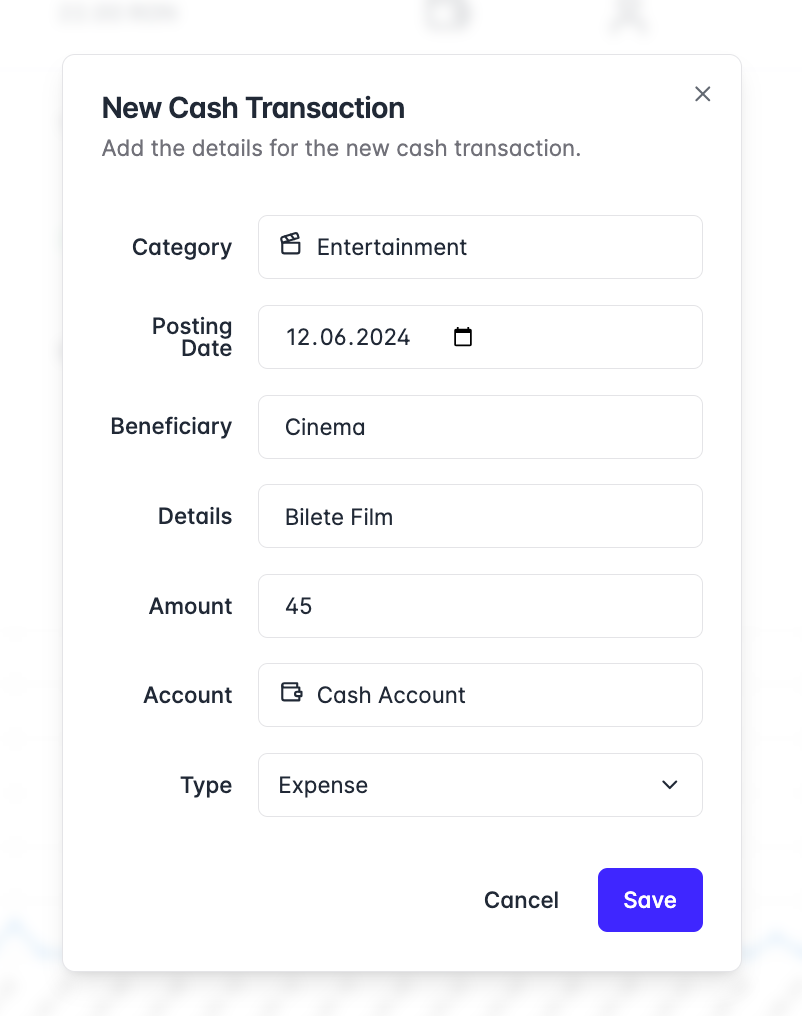
* **Tabele Responsive:** Tabelele din PocketPlanner sunt proiectate să fie responsive, adaptându-se automat la dimensiunile diferite ale ecranelor pentru a asigura o experiență plăcută indiferent de tipul device-ului pe care este folosită aplicația, iar acestea oferă funcții de paginare, sortare și filtrare pentru a face obținerea informațiilor esențiale cât mai facilă pentru utilizatori.

A screenshot of a white screen

Description automatically generated

Figura 3: Tabelă responsive

* **Popup-uri de Tip Dialog:** Aplicația folosește popup-uri de tip dialog pentru a oferi utilizatorilor un mod cât mai eficient de a interacționa cu datele sale, acestea fiind utilizate în fluxurile de adăugat, actualizat și șters tranzacții, conturi, categorii, bugete, inițierea și acceptarea unui parteneriat, categorisirea unei tranzactii și altele.

A screenshot of a computer

Description automatically generated

Figura 4: Popup-uri de tip dialog

## Design Responsive

Pentru a asigura o experiență de utilizare optimă pe toate dispozitivele, PocketPlanner utilizează break point-urile TailwindCSS pentru a crea un design responsive. Acest lucru permite aplicației să se adapteze la diferite dimensiuni de ecran, de la telefoane mobile la desktop-uri.

#### Break Point-uri TailwindCSS:

* **sm:** 640px și mai mult
* **md:** 768px și mai mult
* **lg:** 1024px și mai mult
* **xl:** 1280px și mai mult
* **2xl:** 1536px și mai mult

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

## Notificări

Notificările joacă un rol crucial în informarea utilizatorilor despre activitățile importante din aplicație. PocketPlanner oferă un sistem de notificări care informează utilizatorii despre actualizările importante și acțiunile relevante.

#### Tipuri de Notificări:

* **Actualizări de Categorii Partajate:** Utilizatorii sunt informați când partenerii lor actualizează categorii partajate, modifică lista de categorii partajate, adăugând noi categorii partajate, sau mutând categorii partajate în personale.
* **Nou Parteneriat:** Notifică utilizatorii atunci când primesc o solicitare de parteneriat sau când un partener a acceptat sau a refuzat o solicitare de parteneriat.
* **Tranzacții Noi:** Alertează utilizatorii despre noile tranzacții bancare în conturile lor importate prin intermediul data pipeline-ului.
* **Actualizări ale Soldurilor:** Notifică utilizatorii când solduri conturilor lor bancare a fost actualizat prin intermediul data pipeline-ului.

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generated

Figura 5: Notificări

## Autentificare

### Login și Sign-Up

În PocketPlanner, utilizatorii trebuie să se autentifice pentru a accesa funcționalitățile aplicației. Aceștia pot fie să creeze un cont nou (sign-up), fie să se autentifice cu un cont existent (login). Procesul de autentificare implică introducerea adresei de email și a parolei.

**Funcționalități principale:**

* **Formular de login:** Utilizatorii introduc adresa de email și parola pentru a se autentifica.
* **Formular de sign-up:** Utilizatorii pot crea un cont nou completând adresa de email, numele și parola.

**Validări:**

* Ambele formulare conțin validări pentru câmpurile ce trebuie completate, precum verificarea conținutului să nu fie gol, email-ul să respecte formatul standard, iar termenii și condițiile să fie acceptați.

A screenshot of a login form

Description automatically generatedA screenshot of a login form

Description automatically generated

Figura 6: Autentificare

### Autentificare cu Google (OAuth)

Pentru a facilita procesul de autentificare, PocketPlanner permite utilizatorilor să se autentifice folosind contul Google prin OAuth. Acest lucru asigură o autentificare rapidă și sigură.

**Flux de autentificare cu Google:**

1. Utilizatorul apasă pe butonul „Login with Google”.
2. Este redirecționat către pagina de autentificare Google.
3. După autentificare, utilizatorul este redirecționat înapoi în aplicație, în pagina “Dashboard” și este autentificat automat.

### Actualizarea informațiilor inițiale

Pentru finalizarea procesului de sign-up, utilizatorii noi trec printr-un flux de creare a primelor lor categorii în care vor fi încadrate tranzacțiile, selectând din cele predefinite. De asemenea, acesta își va seta suma totală de bugetat și își va crea primul său cont cash din care să poată face tranzacții prin selectarea numelui, soldului și valutei acestui cont.

A screenshot of a phone

Description automatically generatedA screenshot of a registration form

Description automatically generated

Figura 7: Finalizarea înregistrării

## Dashboard

Dashboard-ul oferă o vedere de ansamblu asupra stării financiare a utilizatorului, prezentând informații cheie despre cele mai recente tranzacții și bugete ale utilizatorului, precum și grafice pentru a facilita urmărirea cheltuielilor și bugetelor

**Informații prezentate:**

* **Tranzacții recente:** O listă cu ultimele tranzacții înregistrate, incluzând detalii precum categoria, data, suma, beneficiarul și contul asociat.
* **Grafic de cheltuieli:** Grafic de tip „pie” care ilustrează distribuția cheltuielilor pe categorii, oferind utilizatorilor o perspectivă vizuală asupra modului în care își gestionează finanțele.
* **Bugete curente:** O listă cu cele mai recent actualizate bugete, care arată cât din buget a fost cheltuit și cât este disponibil pentru fiecare categorie.
* **Grafic de bugete: Grafic de tip** „**pie” care ilustrează cât din suma totală bugetată a fost alocat pentru fiecare buget.**

A screenshot of a computer

Description automatically generated

## Gestionarea Conturilor

Utilizatorii pot își pot gestiona în cadrul aplicației atât conturile cash, din care pot adăuga în aplicație si tranzacții cash, cât și pe conturile bancare, PocketPlanner oferind utilizatorului posibilitatea de a-și conecta conturile bancare pentru a urmări soldurile și tranzacțiile acestora în timp real. Această importare a datelor bancare se realizează cu ajutorul API-ului oferit de serviciul Go Cardless.

### ****Conturi Cash****

Utilizatorii pot adăuga și gestiona conturile lor cash în diferite valute pentru a le permite să își gestioneze și tranzacțiile cash din aceste conturi. Tranzacțiile efectuate din aceste conturi vor fi în valuta contului iar soldul acestor conturi cash poate fi urmărit prin graficul de tip „pie” ce afișează soldurile tuturor conturilor utilizatorului convertite în RON pe utilizând ratele de exchange actualizate zilnic.

A screenshot of a account

Description automatically generatedA screenshot of a cash account

Description automatically generated

Figura 8: Adăugarea și Editarea unui cont cash

### ****Sincronizare Bancară****

Utilizatorii pot conecta conturile bancare pentru a importa automat tranzacțiile, menținând datele actualizate. Fluxul începe prin alegerea băncii de la care utilizatorul dorește să importe conturile și tranzacțiile bancare. După alegerea băncii, aplicația redirecționează utilizatorul către pagina de oferire a consimțământului corespunzătoare băncii respective. Utilizatorul își dă consimțământul și își confirmă identitatea, apoi este redirecționat în aplicație unde este întâmpinat de un popup ce îl informează că sincronizarea datelor bancare este în proces. După finalizarea importului inițial, utilizatorul își poate vedea urmări conturile și tranzacțiile pentru care și-a dat acordul în paginile de conturi, respective tranzacții și dashboard.

A screenshot of a phone

Description automatically generatedA screenshot of a account

Description automatically generated

Figura 9: Crearea unei conexiuni bancare

**A screenshot of a computer

Description automatically generated**

Figura 10: Afișarea conturilor bancare în aplicație

### Sincronizare automată prin pipeline

După importarea inițială a datelor bancare, soldurile conturilor și lista tranzacțiilor utilizatorului este actualizată de către data pipeline-ul aplicației. Acesta rulează în fiecare oră un task de actualizare a datelor bancare ale tuturor utilizatorilor, task ce apelează endpoint-urile de oferite de Go Cardless pentru obținerea soldurilor conturilor și a listelor de tranzacții corespunzătoare acestor conturi, apoi compară informațiile obținute cu cele deja existente în baza de date iar dacă găsește solduri actualizate sau tranzacții noi pentru utilizatorul respective, pipeline-ul apelează endpoint-urile dedicate ale backend-ului pentru actualizarea datelor, iar utilizatorul este notificat de actualizările efectuate. Acest mecanism permite menținerea datelor bancare ale utilizatorului actualizate fără a îngreuna experiența de utilizare cu apeluri către API-uri externe ce pun utilizatorul să aștepte pentru încărcarea datelor.

## Categorii

Utilizatorii pot crea, edita și șterge categorii de cheltuieli și venituri. Fiecare tranzacție, cash sau bancară poate fi categorisită pentru a-i permite utilizatorului să își urmărească tendințele în cheltuieli și venituri. De asemenea, fiecare utilizator poate face parte dintr-un parteneriat cu un alt utilizator iar prin acest parteneriat, cei 2 pot urmări tranzacțiile din categoriile pe care aleg să le partajeze.

### Gestionarea Categoriilor

* **Creare și editare categorii:** Permite utilizatorilor să creeze noi categorii sau să editeze categorii existente pentru a reflecta mai bine obiceiurile lor de cheltuieli.
* **Partajare categorii:** Utilizatorii pot partaja categorii cu partenerul lor, inclusiv ulterior acceptării parteneriatului cu toate categoriile partajate inițial. În plus, pot muta categorii partajate înapoi în cele personale, iar editarea sau ștergerea unei categorii partajate de către un utilizator se va reflecta și în categoriile partenerului său.
* **Filtrare categorii:** Oferă posibilitatea de a filtra categoriile după tip (toate, personal sau partajat).

A screenshot of a computer

Description automatically generated

Figura 11: Afișarea categoriilor în aplicație

### Parteneriate

* **Creare parteneriat:** Utilizatorul ce dorește să inițieze un parteneriat poate face asta apăsând pe butonul „Initiate Partnership”. Acesta va deschide un popup de tip dialog ce va îndemna utilizatorul să introducă email-ul partenerului și să selecteze categoriile pe care dorește să le partajeze. Este validat faptul că partenerul invitat are un email valid și nu se află deja într-un alt parteneriat, iar dacă aceste validări sunt trecute cu succes, parteneriatul este creat cu status „pending” până la acceptarea acestuia.

A screenshot of a phone

Description automatically generated

Figura 12: Inițierea parteneriatului

* **Acceptare parteneriat: Utilizatorul invitat într-un parteneriat este notificat în acest sens prin sistemul de notificări al aplicației, iar la accesarea secțiunii de categorii partajate din cadrul paginii de categorii, un popup de tip dialog se va deschide informând utilizatorul de invitația primită.** Acesta are posibilitatea să accepte sau să refuze parteneriatul iar dacă îl acceptă, statusul parteneriatul se schimbă din „pending” în „confirmed”, iar cei 2 utilizatori pot începe să urmărească tranzacțiile din categoriile partajate împreună.

A screenshot of a computer screen

Description automatically generated

Figura 13: Informarea utilizatorului despre invitația într-un parteneriat

* **Modificarea ulterioară a categoriilor:** După confirmarea unui parteneriat, oricare dintre cei 2 utilizatori ce fac parte din acesta pot modifica lista de categorii partajate, adăugând categorii personale ale lor, ce va crea acea categorie și pentru partener, sau mutând categorii partajate în cele personale, schimbare ce va fi, de asemenea, reflectată în categoriile ambilor utilizatori.

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generated

Figura 14: Modificarea ulterioară a categoriilor partajate

* **Părăsirea unui parteneriat:** În orice moment, oricare dintre cei 2 utilizatori înrolați într-un parteneriat poate părăsi parteneriatul, schimbare ce va muta toate categoriile partajate în personale și va declanșa trimiterea unei notificări către partener.

A screenshot of a phone

Description automatically generated

Figura 15: Părăsirea unui parteneriat

## Tranzacții

Pagina de tranzacții permite utilizatorilor să-și gestioneze tranzacțiile financiare, oferind funcționalități pentru adăugarea, editarea și ștergerea tranzacțiilor cash, categorisirea tranzacțiilor cash și bancare, precum și vizualizarea acestora într-un format detaliat și grafic pentru înțelegerea facilă a obiceiurilor financiare ale utilizatorilor.

### Tabela de Tranzacții**:**

* **Informații Prezentate:** Afișează toate tranzacțiile înregistrate ale utilizatorului (și ale partenerului din categoriile partajate, dacă acestea există), cu detalii precum categoria, data, beneficiarul, descrierea și suma și contul.
* **Filtrare, sortare și paginare:** Permite utilizatorilor să filtreze tranzacțiile după oricare din câmpurile afișate, să modifice numărul de tranzacții afișate in tabelă și să sorteze lista în funcție de data adăugării tranzacției.
* **Tranzacții ale Partenerului:** În cazul în care utilizatorul se află într-un parteneriat confirmat, în tabelă se vor regăsi și tranzacțiile cash și bancare ale partenerului său ce categoriilor partajate, astfel încât ambii utilizatori să aibă o perspectivă detaliată asupra cheltuielilor și surselor de venit partajate.

A screenshot of a computer

Description automatically generated

Figura 16: Tabela de tranzacții

### Gestionarea tranzacțiilor cash**:**

* **Adăugare, actualizare și ștergere:** Utilizatorii pot crea, actualiza și șterge tranzacții cash prin intermediul popup-urilor de tip dialog. Popup-urile conțin validare pentru toate câmpurile iar salvarea modificărilor nu este posibilă până când câmpurile nu sunt completate corect.

A screenshot of a phone

Description automatically generatedA screenshot of a computer screen

Description automatically generated

Figura 17: Adăugarea și stergerea unei tranzacții cash

* **Categorisire:** Tranzacțiile din tabel pot fi categorisite prin intermediul popup-ului ce apare la tap pe categoria tranzacției. Pe măsură ce tranzacțiile sunt categorisite, chart-urile vor fi actualizate pentru a reflecta schimbările.

A screenshot of a phone

Description automatically generated

Figura 18: Categorisirea unei tranzacții

### Gestionarea tranzacțiilor bancare**:**

* **Actualizare Automată:** După importul inițial al datelor pentru o anumită conexiune bancară, lista tranzacțiilor bancare este actualizată în fiecare oră prin intermediul task-ului dedicat definit prin Celery în cadrul data pipeline-ului, utilizatorul fiind notificat de această nouă tranzacție.
* **Categorisire Automată:** În primul import de date bancare al unui utilizator pentru o anumită conexiune, tranzacțiile vor fi necategorisite, urmând ca utilizatorul să le categorisească pentru ca algoritmul specific lui să învețe din preferințele sale. După ce utilizatorul are cel puțin 100 de tranzacții categorisite, algoritmul Decision Tree Classifier va fi antrenat pentru preferințele sale iar următoarele tranzacții specifice acelei conexiuni bancare vor fi importate împreună cu categoria prezisă de algoritm. Utilizatorul poate apoi să modifice aceste categorii, algoritmul fiind reantrenat zilnic prin intermediul unui alt task definit prin Celery în cadrul data pipeline-ului.

### Grafice**:**

* **Grafice de tip „pie”:** Pagina de tranzacții conține 2 grafice de tip „pie” ce sumarizează sumele cheltuite și castigate de utilizator în ultimele 3 luni împărțite pe categorii.

A screenshot of a cell phone screen

Description automatically generated

Figura 19: Graficele din pagina de tranzacții

* **Grafic de tip „line”:** În plus, pagina de tranzacții prezintă utilizatorului și un grafic de tip „line” ce îi permite acestuia să urmărească cheltuielile de-a lungul timpului.

A graph with blue lines and numbers

Description automatically generated

Figura 20: Graficul de tip line din pagina de tranzacții

## Bugete

Utilizatorii pot seta bugete pentru categoriile de cheltuieli și monitoriza cheltuielile în raport cu aceste bugete. Fiecare buget este resetat lunar, iar utilizatorii pot vedea cât au cheltuit și cât au rămas din bugetul alocat.

### ****Funcționalități principale:****

* **Setare bugete:** Permite utilizatorilor să aloce sume specifice pentru fiecare categorie de cheltuieli în orice valută.
* **Monitorizare cheltuieli:** Afișează progresul cheltuielilor în raport cu bugetul stabilit. În plus, sumele cheltuite și disponibile convertite în valuta bugetului sunt vizibile la hover peste bara de progres a bugetului respectiv.
* **Partajare bugete:** Bugetele ce aparțin unor categorii partajate vor aparține ambilor utilizatorilor din parteneriat, iar aceștia vor putea urmări cheltuielile amândurora pentru aceste categorii.
* **Resetare lunară:** Fiecare buget are o data de start și o data de resetare calculată automat ca fiind o lună de la setarea bugetului. Aceste date sunt actualizate automat în fiecare lună de către job-ul dedicate definit în backend-ul aplicației pentru a permite utilizatorilor să urmărească cheltuielile relevante pentru aceste categorii.

A screenshot of a phone

Description automatically generated

Figura 21: Tabela de bugete

### Grafice**:**

* **Grafic de tip „pie”:** Pagina de bugete conține un grafic de tip „pie” ce sumarizează cât din suma totală de bugetat a alocat utilizatorul în fiecare din bugetele sale.

A pie chart with text and numbers

Description automatically generated

Figura 22: Grafic de tip pie în pagina de bugete

* **Grafic de tip „bar”:** În plus, pagina de bugete prezintă utilizatorului și un grafic de tip „bar” ce afișează evoluția în timp a sumelor cheltuite din bugetul total al utilizatorului pe parcursul ultimelor 6 luni.

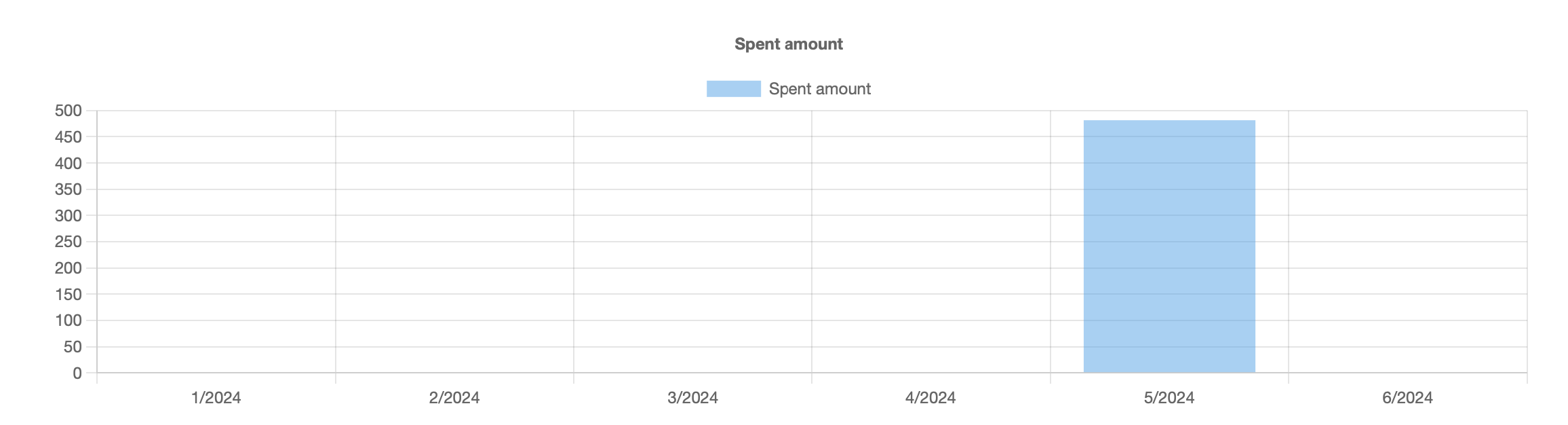


Figura 23: Grafic de tip line în pagina de bugete

# Capitolul 5 - Concluzii și Perspective

## Concluzii

Prin prezenta lucrare am expus conceptul, rolul cât și procesul de realizare al aplicației PocketPlanner, o aplicație web pentru gestionarea finanțelor personale, utilizând tehnologii moderne precum Angular pentru frontend și NestJS pentru backend. Am implementat funcționalități esențiale pentru gestionarea finanțelor personale, incluzând:

* **Autentificare și Autorizare:** Utilizatorii se pot autentifica și autoriza folosind credențiale tradiționale sau prin intermediul contului Google, asigurând un proces de login rapid și securizat.
* **Dashboard Intuitiv:** Dashboard-ul oferă o vedere de ansamblu asupra finanțelor personale, prezentând tranzacții recente, grafice de cheltuieli și bugete curente.
* **Gestionarea Tranzacțiilor:** Utilizatorii pot adăuga, edita și categorisi tranzacții cash și își pot sincroniza tranzacțiile bancare, beneficiind de o interfață intuitivă și ușor de utilizat.
* **Integrarea cu Băncile:** Prin API-ul Go Cardless, utilizatorii pot conecta conturile bancare pentru a importa și sincroniza automat tranzacțiile financiare.
* **Bugete și Categorii Partajate:** Utilizatorii pot crea și gestiona bugete pentru fiecare categorie și pot partaja categorii un partener pentru o gestionare comună a finanțelor.
* **Notificări:** Sistemul de notificări informează utilizatorii despre actualizările importante și acțiunile relevante, asigurându-le că sunt mereu la curent cu activitățile financiare.

## Perspective de Dezvoltare

PocketPlanner a fost conceput pentru a oferi o soluție completă și intuitivă de gestionare a finanțelor personale, însă există anumite direcții de dezvoltare care pot îmbunătăți și extinde funcționalitățile aplicației:

### Funcționalități Adiționale

* **Analiză Predictivă:** Implementarea unor algoritmi de machine learning pentru a oferi utilizatorilor predicții privind cheltuielile viitoare și recomandări pentru bugetare.
* **Plăți și Transferuri:** Integrarea cu servicii de plată pentru a permite utilizatorilor să efectueze plăți și transferuri și să inițieze abonamente și plăți recurente direct din aplicație.
* **Monitorizare Investiții:** Adăugarea funcționalităților pentru a urmări și gestiona portofoliile de investiții ale utilizatorilor.

### Îmbunătățiri de Performanță și Securitate

* **Optimizarea Performanței:** Optimizarea interogărilor bazei de date și a fluxurilor de date pentru a asigura performanțe ridicate chiar și pentru utilizatorii cu volume mari de tranzacții.
* **Securitate Avansată:** Implementarea unor măsuri suplimentare de securitate, cum ar fi autentificarea multifactor (MFA) și integrarea cu alte servicii de OAuth, precum Facebook sau Github.

Prin lista mai sus menționată am demonstrat faptul că, aplicația PocketPlanner are un potențial ridicat. Numeroase funcționalități mai pot fi integrate pentru a transforma acest proiect, ce în momentul actual se află în forma lui de bază, într-o aplicație ce poate ajuta numeroase persoane să își gestioneze cu ușurință finanțele personale.

# Bibliografie

1. URL: <https://angular.io/docs>
2. URL: <https://tailwindcss.com/docs>
3. URL: <https://docs.nestjs.com>
4. URL: <https://mongoosejs.com/docs>
5. URL: <https://swagger.io/docs>
6. URL: <https://docs.celeryq.dev/en/stable/>
7. URL: <https://datatracker.ietf.org/doc/html/rfc6749>
8. URL: <https://developer.gocardless.com>
9. URL: <https://www.typescriptlang.org/docs/handbook/intro.html>
10. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, 2011 - <https://jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf>
11. Quinlan, J. R., "Induction of Decision Trees," Machine Learning, 1986 - <https://link.springer.com/article/10.1007/BF00116251>
12. Karpathy, Andrej, "The Unreasonable Effectiveness of Recurrent Neural Networks," 2015 - <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>
13. European Banking Authority, "Regulatory Technical Standards on Strong Customer Authentication and Secure Communication," 2017 - <https://www.eba.europa.eu/legacy/regulation-and-policy/regulatory-activities/payment-services-and-electronic-money-0>