### UNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

# „Pocket Planner”

# Aplicație Web Pentru Gestionarea Finanțelor Personale

### Absolvent

### Narcis-Alexandru Fănică

**Coordonator științific**

**Marius Mihăilescu**

**București, iunie 2024**

#### Rezumat

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vitae eros sit amet sem ornare varius. Duis eget felis eget risus posuere luctus. Integer odio metus, eleifend at nunc vitae, rutrum fermentum leo. Quisque rutrum vitae risus nec porta. Nunc eu orci euismod, ornare risus at, accumsan augue. Ut tincidunt pharetra convallis. Maecenas ut pretium ex. Morbi tellus dui, viverra quis augue at, tincidunt hendrerit orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis sollicitudin nunc. Sed sollicitudin purus dapibus mi fringilla, nec tincidunt nunc eleifend. Nam ut molestie erat. Integer eros dolor, viverra quis massa at, auctor.

#### Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vitae eros sit amet sem ornare varius. Duis eget felis eget risus posuere luctus. Integer odio metus, eleifend at nunc vitae, rutrum fermentum leo. Quisque rutrum vitae risus nec porta. Nunc eu orci euismod, ornare risus at, accumsan augue. Ut tincidunt pharetra convallis. Maecenas ut pretium ex. Morbi tellus dui, viverra quis augue at, tincidunt hendrerit orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis sollicitudin nunc. Sed sollicitudin purus dapibus mi fringilla, nec tincidunt nunc eleifend. Nam ut molestie erat. Integer eros dolor, viverra quis massa at, auctor.

1. **INTRODUCERE 4**
   1. [Lorem ipsum](#_heading=h.3znysh7) 4
   2. [Dolor sit amet](#_heading=h.tyjcwt) 5
2. **CONCEPTE TEORETICE 6**
3. **UTILIZAREA TEHNOLOGIILOR ALESE 7**
4. **PREZENTAREA APLICAȚIEI 8**

[**Bibliografi**](#_heading=h.lnxbz9)**e**

**Capitolul 1 - Introducere**

## Motivație

În era digitală, gestionarea finanțelor personale a devenit o necesitate pentru a menține o evidență clară a veniturilor și cheltuielilor. Aplicațiile de gestionare a finanțelor personale oferă utilizatorilor un control mai bun asupra finanțelor lor, ajutându-i să-și atingă obiectivele financiare. Pocket Planner este o astfel de aplicație, dezvoltată pentru a facilita gestionarea tranzacțiilor, bugetelor și conturilor bancare într-un mod eficient și sigur.

## Preliminarii

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

## Nevoi pe care le satisface aplicația

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

* Un mod centralizat de a gestiona tranzacțiile și conturile bancare.
* Funcționalități de categorisire automată a tranzacțiilor folosind algoritmi de machine learning.
* Sincronizarea automată a tranzacțiilor bancare prin integrarea cu serviciul Go Cardless.
* Posibilitatea de a partaja informații financiare relevante cu alți utilizatori 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.
* YNAB (You Need A Budget): Focusată pe planificarea bugetelor și educația financiară.
* Personal Capital: Combină gestionarea finanțelor personale cu sfaturi de investiții.

## Ce aduce nou?

Pocket Planner 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 Naive Bayes pentru categorisirea automată a tranzacțiilor pe baza istoricului și preferințelor utilizatorului.
* **Funcționalitatea de parteneriat:** Permite utilizatorilor să colaboreze în gestionarea anumitor categorii financiare, facilitând gestionarea bugetelor comune.
* **Pipeline de date pentru actualizarea tranzacțiilor:** Un sistem automatizat, construit cu Python și Celery, verifică periodic și actualizează tranzacțiile pentru a menține aplicația actualizată fără a încărca excesiv serverele.

## Structura lucrării

Lucrarea este structurată pentru a oferi o imagine completă și detaliată asupra dezvoltării și funcționalităților aplicației Pocket Planner. Structura include:

* **Capitolul 1 - Introducere:** Prezentarea motivației, contextului, obiectivelor și inovațiilor aduse de Pocket Planner.
* **Capitolul 2 - Utilizarea tehnologiilor alese:** Descrierea tehnologiilor utilizate în dezvoltarea aplicației, precum Angular, NestJS, Tailwind CSS și altele.
* **Capitolul 3 - Arhitectura sistemului:** Detalierea arhitecturii backend-ului și frontend-ului aplicației, explicând modul de organizare și funcționare a componentelor.
* **Capitolul 4 - Prezentarea aplicației:** Descrierea interfeței utilizatorului, funcționalităților principale și a modului de utilizare a aplicației (acest capitol va fi dezvoltat ulterior).
* **Capitolul 5 - Concluzii și perspective:** Analiza impactului aplicației, concluziile dezvoltării și perspectivele de viitor pentru extinderea funcționalităților

# Capitolul 2 – Concepte Teoretice

### 2.1. 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, care necesită reîncărcarea întregii pagini pentru fiecare acțiune, SPAs încarcă resursele necesare la început și folosesc JavaScript pentru a manipula DOM-ul și a face cereri asincrone către server (AJAX), 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 AJAX 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.

### 2.2. TypeScript

TypeScript este un limbaj de programare dezvoltat de Microsoft, care extinde JavaScript prin adăugarea de tipuri statice. TypeScript compilează 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 și îmbunătățind claritatea codului.
* **Instrumente de dezvoltare:** TypeScript oferă suport avansat pentru instrumente de dezvoltare, cum ar fi IntelliSense, refactorizare automată și completare de cod.
* **Compatibilitate:** Codul TypeScript este compilat în JavaScript, ceea ce înseamnă că poate fi utilizat oriunde este acceptat JavaScript.
* **OOP:** Suportă concepte de programare orientată pe obiecte, cum ar fi clase, interfețe și moștenire, facilitând scrierea de cod modular și reutilizabil.

#### 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.

### 2.3. 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 a le crea în interiorul său. DI este larg utilizat în framework-urile moderne, inclusiv Angular și NestJS.

#### Avantaje ale Dependency Injection:

* **Separarea preocupărilor:** DI separă comportamentul aplicației de modul în care instanțele sunt create și rezolvate, rezultând un cod mai curat și mai ușor de întreținut.
* **Facilitează testarea:** Dependențele pot fi ușor înlocuite cu mock-uri sau stubs în timpul testării, făcând testarea unităților de cod 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 furnizori de servicii, asigurând un cod modular și ușor de întreținut.

constructor(

private nordigenService: NordigenService,

private accountsService: BankAccountsService,

private transactionsService: BankTransactionsService,

) {}

### 

### 2.3. Dependency Injection (DI)

Swagger este un framework open-source utilizat pentru construirea, documentarea și consumarea 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()

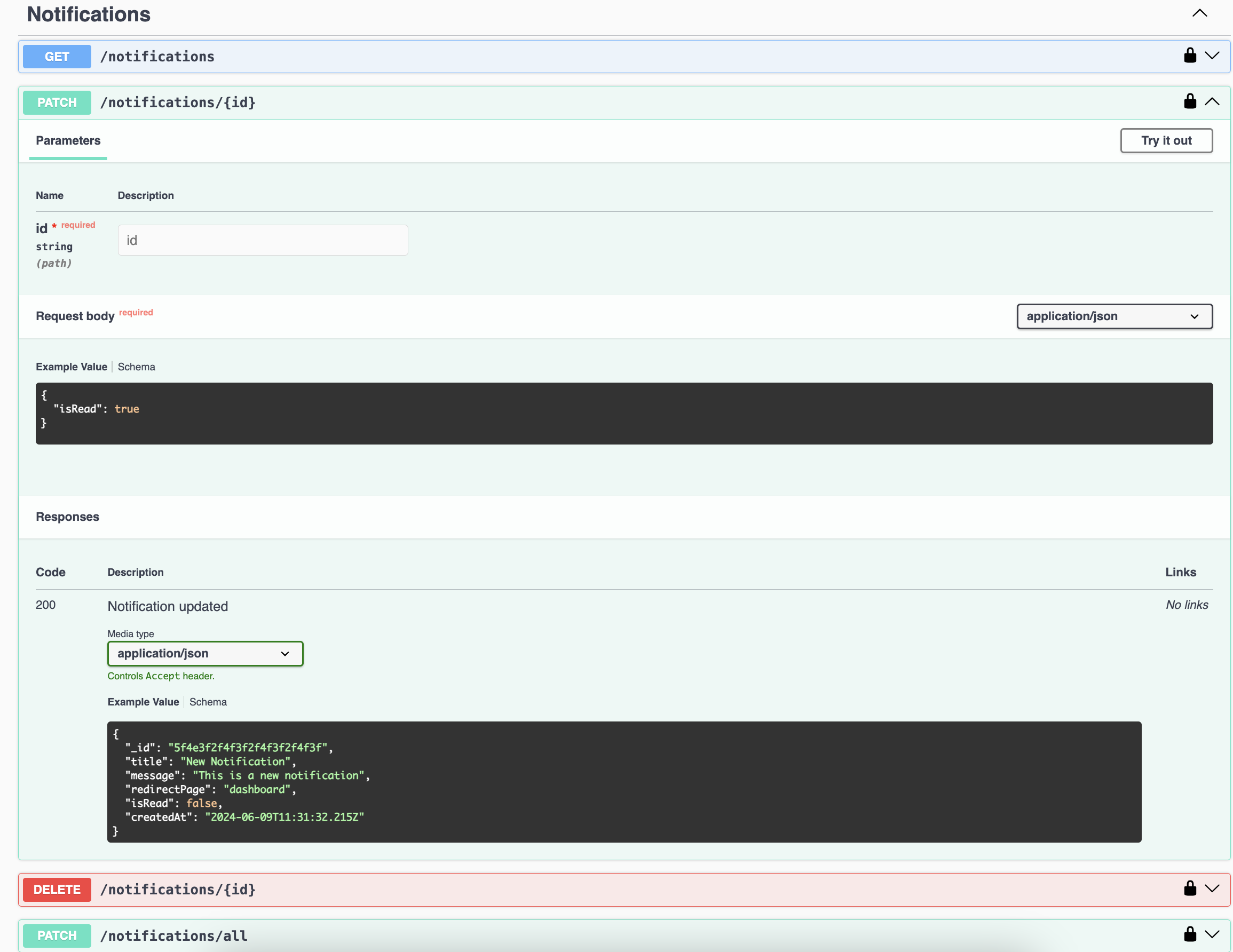
.build();

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

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

}

}



### 2.4. 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 foarte flexibil și permite stocarea unor structuri de date complexe și nestandardizate.

#### Caracteristici:

* **Flexibilitate:** Structurile de date sunt flexibile și permit stocarea de date nestandardizate, fără schema strictă a bazelor de date relaționale.
* **Scalabilitate:** Sistemele de baze de date documentare sunt de obicei foarte scalabile, fiind capabile să gestioneze mari volume 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.
* **Replicație și disponibilitate:** MongoDB suportă replicația datelor prin seturi de replici, asigurând redundanță și disponibilitate ridicată în cazul defectării unui nod.
* **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.

### 2.5. 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 acreditările lor (parole). OAuth este utilizat pe scară largă pentru autentificarea cu furnizori de identitate precum Google, Facebook, și alții.

#### Caracteristici:

* **Securitate:** OAuth oferă un mod sigur pentru aplicații de a accesa resursele utilizatorilor fără a compromite securitatea acreditărilor.
* **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 Pocket Planner:

* **Autentificare:** Utilizatorii Pocket Planner 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 Pocket Planner să acceseze informațiile sale de profil.
* **Token de acces:** Dacă utilizatorul consimte, Google emite un token de acces care este utilizat de Pocket Planner pentru a accesa resursele autorizate.
* **Integrare în aplicație:** Token-ul de acces este folosit pentru a obține informațiile utilizatorului (cum ar fi adresa de email și numele) și pentru a finaliza procesul de autentificare.

@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);

}

}

### 2.6. Machine Learning și Algoritmi de Clasificare

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

#### Multinomial Naive Bayes:

Algoritmul Multinomial Naive Bayes este un algoritm de clasificare simplu, dar eficient, utilizat adesea pentru clasificarea textului și alte tipuri de date discrete.

* **Principiu de funcționare:** Se bazează pe aplicarea teoremei lui Bayes cu o presupunere de independență între caracteristici. Fiecare caracteristică contribuie în mod independent la probabilitatea de clasificare.
* **Avantaje:** Este rapid și eficient, necesită cantități mici de date de antrenament și funcționează bine chiar și cu seturi de date sparse. Este foarte potrivit pentru probleme de clasificare text, cum ar fi analiza sentimentelor și clasificarea emailurilor.
* **Performanță:** În ciuda presupunerii sale simple de independență, Multinomial Naive Bayes oferă performanțe competitive cu algoritmi mai complecși pentru multe aplicații practice.

### 2.7. 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.

### 2.8. Open Banking

Open Banking reprezintă un concept și o practică în industria financiară care promovează accesul deschis la datele financiare ale utilizatorilor prin intermediul unor API-uri securizate. Acest lucru permite dezvoltatorilor de aplicații să construiască servicii financiare inovatoare, oferind utilizatorilor o experiență îmbunătățită și mai multe opțiuni pentru gestionarea finanțelor personale.

#### 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 Pocket Planner 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 Pocket Planner 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 Pocket Planner, iar aplicația va sincroniza automat tranzacțiile în timp real.
* **Analiză și Categorisire:** Datele bancare sunt utilizate pentru a analiza și categorisi tranzacțiile utilizatorilor, oferindu-le informații valoroase despre cheltuielile și veniturile lor.

# Capitolul 3 – Utilizarea Tehnologiilor alese

### 3.1. 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 și oferă suport pentru data binding bidirecțional, dependency injection și multe alte funcționalități.

#### 3.1.1. Caracteristici definitorii

* **Modularitate:** Angular permite împărțirea aplicației în module distincte, fiecare având propriile componente, servicii și alte resurse.
* **Data Binding:** Angular suportă data binding bidirecțional, sincronizând automat modelul de date și view-ul.
* **Dependency Injection:** Angular utilizează un sistem de injectare a dependențelor, îmbunătățind testabilitatea și flexibilitatea aplicației.
* **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);

},

});

}

### 3.2. TailwindCSS

TailwindCSS este un framework CSS utilitar care permite dezvoltatorilor să construiască interfețe responsive și personalizabile rapid și eficient. TailwindCSS se bazează pe o abordare de stilizare folosind clase utilitare, evitând astfel scrierea de cod CSS suplimentar.

#### 3.2.1. Caracteristici definitorii

* **Utility-First:** TailwindCSS oferă o gamă largă de clase utilitare predefinite pentru a aplica stiluri direct în HTML.
* **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 clase utilitare pentru diferite dimensiuni ale ecranului.

<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">

### 3.3. 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.

#### 3.3.1. 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 il 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);

}

}

#### 3.3.2. 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">

### 3.4. 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.

#### 3.4.1. Caracteristici definitorii

* **Arhitectură modulară:** NestJS permite structurarea aplicației în module distincte, îmbunătățind organizarea și mentenanța codului.
* **Integrări puternice:** Oferă suport integrat pentru Mongoose (pentru gestionarea bazelor de date MongoDB), Swagger (pentru documentarea API-urilor), validare, scheduling și OAuth.
* **Extensibilitate:** NestJS permite utilizarea de middleware, interceptori și guards pentru a extinde funcționalitățile de bază ale aplicației.

@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 {}

### 3.5. Mongoose

Mongoose este o librărie de modelare a datelor pentru MongoDB și Node.js. Oferă un API elegant și puternic pentru a interacționa cu MongoDB, facilitând definirea schemelor și validarea datelor.

#### 3.5.1. Caracteristici definitorii

* **Schema-based:** Mongoose permite definirea schemelor de date care sunt utilizate pentru validarea și modelarea documentelor MongoDB.
* **Middleware:** Oferă suport pentru middleware pre și post hooks, permițând executarea logicii personalizate înainte și după operațiuni de bază de date.
* **Validare:** Include mecanisme puternice de validare a datelor, asigurând integritatea acestora.

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);

### 3.6. Python și Celery

PocketPlanner utilizează Python pentru a implementa un pipeline de date care verifică periodic și actualizează tranzacțiile bancare. Celery este folosit pentru gestionarea task-urilor asincrone în acest pipeline.

#### 3.6.1. 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.

#### 3.6.2. 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")

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.accounts.find(

{"user": user["\_id"], "requisition": requisition["requisition"]}

)

for account in accounts:

update\_user\_data(

user["\_id"], account["account"], requisition["requisition"]

)

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

beat\_schedule = {

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

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

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

},

"retrain-models-daily": {

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

"schedule": crontab(minute=0, hour=0),

},

}

### 3.7. Algoritmul Multinomial Naive Bayes

Pentru categorisirea automată a tranzacțiilor, PocketPlanner utilizează un model de machine learning bazat pe algoritmul Multinomial Naive Bayes. Acest algoritm este eficient pentru probleme de clasificare și este utilizat pentru a învăța tiparele din datele de tranzacții ale utilizatorului.

#### 3.7.1. Caracteristici definitorii

* **Simplicitate și eficiență:** Algoritmul Multinomial Naive Bayes 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.

#### 3.7.2. Implementare în PocketPlanner

Algoritmul Multinomial Naive Bayes este implementat în pipeline-ul de date pentru a categorisi noile tranzacții pe baza istoricului de tranzacții al utilizatorului. Acest model învață din datele anterioare și prezice categoriile pentru tranzacțiile viitoare, îmbunătățind astfel experiența utilizatorului prin automatizarea procesului de categorisire.

transactions = db.transactions.find(

{

"userId": user\_id,

"requisitionId": requisition\_id,

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

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

}

)

data = [(t["transactionData"]["details"], t["categoryId"]) for t in transactions]

if not data:

return None

descriptions, categories = zip(\*data)

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

model.fit(descriptions, categories)

## Capitolul 4 - Prezentarea aplicației

### 4.1. Prezentarea detaliată a componentelor

#### 4.1.1. Dashboard

Dashboard-ul aplicației oferă o vedere de ansamblu asupra stării financiare a utilizatorului. Utilizatorii pot vizualiza tranzacțiile recente, bugetele curente și grafice care ilustrează distribuția cheltuielilor pe categorii. Acesta este punctul central de acces al aplicației, de unde utilizatorii pot naviga către alte secțiuni.

**Funcționalități principale:**

* **Tranzacții recente:** O listă cu ultimele tranzacții înregistrate, inclusiv detalii precum data, suma, beneficiarul și contul asociat.
* **Grafice de cheltuieli:** Grafice de tip „pie” care arată distribuția cheltuielilor pe categorii, oferind utilizatorilor o perspectivă vizuală asupra modului în care își gestionează finanțele.
* **Bugete curente:** O secțiune dedicată bugetelor, care arată cât din buget a fost cheltuit și cât este disponibil pentru fiecare categorie.

#### 4.1.2. Gestionarea tranzacțiilor

Utilizatorii pot adăuga, edita și șterge tranzacții financiare. Tranzacțiile pot fi de două tipuri: cash sau bancare. Fiecare tranzacție este categorisită și afișată într-o listă detaliată. Utilizatorii pot filtra tranzacțiile după dată, categorie sau beneficiar.

**Funcționalități principale:**

* **Listă de tranzacții:** Afișează toate tranzacțiile înregistrate, cu detalii precum categoria, data, beneficiarul, descrierea și suma.
* **Filtrare și sortare:** Permite utilizatorilor să filtreze tranzacțiile după criterii specifice și să sorteze lista în funcție de dată sau sumă.
* **Grafice de cheltuieli și venituri:** Grafice de tip „line” și „pie” care ilustrează evoluția cheltuielilor și veniturilor în timp.

#### 4.1.3. Crearea de noi tranzacții

Utilizatorii pot crea tranzacții noi prin completarea unui formular simplu. Acesta include câmpuri pentru categorie, data tranzacției, beneficiar, detalii, sumă și tipul tranzacției (venit sau cheltuială). Formularul este intuitiv și ușor de utilizat, asigurând o înregistrare rapidă a tranzacțiilor.

**Funcționalități principale:**

* **Formular de adăugare tranzacție:** Câmpuri pentru detaliile esențiale ale tranzacției, inclusiv selecția contului și a tipului de tranzacție.
* **Validare:** Asigură că toate câmpurile necesare sunt completate corect înainte de a permite salvarea tranzacției.
* **Categorii personalizate:** Utilizatorii pot selecta dintr-o listă de categorii predefinite sau pot adăuga categorii noi.

#### 4.1.4. Gestionarea bugetelor

Utilizatorii pot seta bugete pentru diferite categorii 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. Aceasta ajută la gestionarea eficientă a finanțelor personale și la evitarea depășirii bugetelor.

**Funcționalități principale:**

* **Setare bugete:** Permite utilizatorilor să aloce sume specifice pentru fiecare categorie de cheltuieli.
* **Monitorizare cheltuieli:** Afișează progresul cheltuielilor în raport cu bugetul stabilit, inclusiv sumele cheltuite și disponibile.
* **Partajare bugete:** Utilizatorii pot partaja bugete cu alți utilizatori, facilitând gestionarea comună a finanțelor.

#### 4.1.5. Gestionarea categoriilor

Categorii de cheltuieli și venituri pot fi create, editate și șterse. Utilizatorii pot partaja anumite categorii cu alți utilizatori pentru a urmări împreună anumite cheltuieli sau venituri. Această funcționalitate oferă flexibilitate și personalizare în gestionarea tranzacțiilor.

**Funcționalități principale:**

* **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 specifice cu alți utilizatori, facilitând colaborarea și transparența financiară.
* **Filtrare categorii:** Oferă posibilitatea de a filtra categoriile după tip (personal sau partajat).

#### 4.1.6. Gestionarea conturilor

Utilizatorii pot adăuga și gestiona conturi bancare și conturi cash. Aplicația permite importul automat al tranzacțiilor bancare prin integrarea cu API-ul Nordigen. Aceasta asigură actualizarea constantă a datelor financiare fără intervenția manuală a utilizatorului.

**Funcționalități principale:**

* **Adăugare conturi:** Utilizatorii pot adăuga conturi noi, fie bancare, fie cash, și pot introduce detalii precum numele contului și soldul inițial.
* **Sincronizare bancară:** Utilizatorii pot conecta conturile bancare pentru a importa automat tranzacțiile, menținând datele actualizate.
* **Vizualizare solduri:** Afișează soldurile conturilor în timp real, oferind o imagine clară asupra situației financiare a utilizatorului.

#### 4.1.7. Autentificare și OAuth

Utilizatorii se pot autentifica folosind un cont creat anterior sau prin OAuth cu Google. Acest lucru asigură o experiență de utilizare sigură și convenabilă. Sistemul de autentificare protejează datele utilizatorilor și previne accesul neautorizat.

**Funcționalități principale:**

* **Autentificare tradițională:** Utilizatorii pot crea un cont nou sau se pot autentifica folosind un email și o parolă.
* **OAuth:** Permite autentificarea rapidă și securizată prin contul Google, reducând fricțiunile la înscriere și login.
* **Resetare parolă:** Funcționalitate pentru recuperarea contului în cazul pierderii parolei.

#### 4.1.8. Notificări

Utilizatorii primesc notificări pentru diferite evenimente, cum ar fi actualizările de categorii partajate sau tranzacțiile noi. Notificările sunt afișate într-un panou dedicat și pot fi marcate ca citite. Aceasta asigură că utilizatorii sunt mereu informați despre activitățile recente din contul lor.

**Funcționalități principale:**

* **Notificări în timp real:** Afișează notificări pentru evenimente importante, cum ar fi noi tranzacții, actualizări ale categoriilor sau partajarea bugetelor.
* **Panou notificări:** Oferă o vizualizare centralizată a tuturor notificărilor recente, permițând utilizatorilor să le gestioneze eficient.
* **Marcare ca citite:** Utilizatorii pot marca notificările ca citite pentru a menține panoul organizat.

### 4.2. Structura internă a aplicației

#### 4.2.1. Frontend

Frontend-ul aplicației este dezvoltat folosind Angular, un framework modern care permite crearea de aplicații web dinamice și interactive. Angular este utilizat pentru componentizarea interfeței utilizatorului și pentru gestionarea stării aplicației.

**Caracteristici principale:**

* **Componente reutilizabile:** Angular permite crearea de componente UI reutilizabile, ceea ce reduce duplicarea codului și îmbunătățește mentenabilitatea.
* **Routing:** Angular Router gestionează navigarea între diferitele pagini ale aplicației, asigurând o experiență fluidă pentru utilizatori.
* **Servicii:** Angular oferă un sistem robust de servicii pentru partajarea logicii de afaceri și a datelor între componente.

#### 4.2.2. Backend

Backend-ul aplicației este construit folosind NestJS, un framework progresiv pentru Node.js care permite dezvoltarea de aplicații server-side scalabile și eficiente. NestJS utilizează Mongoose pentru interacțiunea cu baza de date MongoDB și oferă suport pentru validare, autentificare și documentare API.

**Caracteristici principale:**

* **Arhitectură modulară:** NestJS permite organizarea codului în module, fiecare gestionând funcționalități distincte, cum ar fi autentificarea, gestionarea tranzacțiilor și a conturilor.
* **Interacțiune cu baza de date:** Utilizând Mongoose, NestJS permite definirea de modele pentru documentele MongoDB și oferă funcționalități de CRUD (Create, Read, Update, Delete) eficiente.
* **Documentare API:** NestJS integrează Swagger pentru a genera documentația API, facilitând astfel înțelegerea și utilizarea acestuia de către dezvoltatori.

#### 4.2.3. Baza de date

Aplicația utilizează MongoDB, o bază de date NoSQL care stochează datele în format JSON-like. MongoDB oferă flexibilitate și scalabilitate, fiind potrivită pentru gestionarea datelor complexe și nestructurate.

**Caracteristici principale:**

* **Flexibilitate:** MongoDB permite stocarea de date fără o schemă strictă, ceea ce facilitează modificările rapide și adaptarea la cerințe schimbătoare.
* **Scalabilitate:** MongoDB este proiectată pentru a scala orizontal, permițând distribuirea datelor pe mai multe servere și gestionarea unor volume mari de date.
* **Interogare avansată:** MongoDB oferă un limbaj de interogare puternic care permite realizarea de interogări complexe și agregări.

### 4.3. Utilizarea librăriilor externe

#### 4.3.1. TailwindCSS

TailwindCSS este utilizat pentru stilizarea interfeței utilizatorului. Acest framework CSS utilitar permite crearea rapidă și flexibilă de stiluri, reducând necesitatea scrierii de cod CSS personalizat.

**Caracteristici principale:**

* **Clas Utility:** TailwindCSS oferă o gamă largă de clase utilitare pentru aplicarea de stiluri direct în HTML.
* **Personalizare:** Framework-ul este extrem de configurabil, permițând personalizarea temelor, paletelor de culori și a altor stiluri globale.
* **Design responsive:** Facilitează crearea de layout-uri responsive prin utilizarea de clase utilitare pentru diferite dimensiuni ale ecranului.

#### 4.3.2. 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, care oferă seturi predefinite de componente UI stilizate.

**Caracteristici principale:**

* **Componente predefinite:** Spartan UI și Daisy UI oferă o colecție variată de componente UI, cum ar fi butoane, carduri, formulare și altele, toate stilizate folosind TailwindCSS.
* **Integrare ușoară:** Componentele sunt ușor de integrat în aplicație, reducând timpul necesar pentru dezvoltarea interfeței utilizatorului.
* **Design coerent:** Utilizarea acestor librării asigură un design coerent și modern pentru toate elementele aplicației.

#### 4.3.3. Celery

Pentru gestionarea task-urilor asincrone, PocketPlanner utilizează Celery. Acesta este un sistem de cozi de mesaje care permite distribuirea task-urilor către workeri, asigurând execuția paralelă și fiabilă a operațiunilor.

**Caracteristici principale:**

* **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.

#### 4.3.4. API-ul Nordigen

Integrarea cu API-ul Nordigen permite PocketPlanner să acceseze și să sincronizeze datele conturilor bancare ale utilizatorilor. Acest lucru facilitează importul automat al tranzacțiilor bancare și actualizarea continuă a datelor financiare.

**Caracteristici principale:**

* **Acces la date bancare:** API-ul Nordigen permite accesul securizat la datele conturilor bancare ale 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 în timp real.
* **Analiză și categorisire:** Datele bancare sunt utilizate pentru a analiza și categorisi tranzacțiile utilizatorilor, oferindu-le informații valoroase despre cheltuielile și veniturile lor.