**ZEROWASTE - DOCUMENTAȚIE**

1. **INTRODUCERE** .............................................................................................................. 2
2. Scopul aplicației ...................................................................................................... 2
3. Publicul țintă ........................................................................................................... 2
4. Tehnologii utilizate .................................................................................................. 2
5. **ARHITECTURA SISTEMULUI** .......................................................................................... 3
6. Descriere generală .................................................................................................. 3
7. Diagrame ................................................................................................................. 3
8. Dependențe ............................................................................................................ 4
9. **INSTALARE ȘI CONFIGURARE** ........................................................................................ 6
10. Pași pentru instalare ................................................................................................ 6
11. Comenzi pentru rulare locală .................................................................................. 7
12. **STRUCTURA ȘI IMPLEMENTAREA CODULUI** ................................................................. 8
13. Backend ................................................................................................................... 8
14. Server ................................................................................................................... 8
15. Modul AI .............................................................................................................. 9
16. Frontend ............................................................................................................... 11
17. **FUNCȚIONALITĂȚI** ....................................................................................................... 21
18. **SECURITATE** ................................................................................................................. 22
19. **TESTARE** ...................................................................................................................... 23
20. **JURNALIZARE** .............................................................................................................. 23

**O imagine care conține text, Grafică, siglă, Font

Descriere generată automat**

**Aplicație realizată de**

*Pătroi Cristiana-Gabriela*

*Scurtu-Rodriguez Kevin-George*

*Stolniceanu Denisa-Elena*

*Stoica Grigore-Darius*

*Rus Vasti*

*Stan Ionuț-Raul*

***An universitar 2024-2025***

1. **INTRODUCERE**

***a. Scopul aplicației***

Aplicația ZeroWaste a fost dezvoltată cu scopul de a combate risipa alimentară, oferind utilizatorilor instrumente eficiente pentru gestionarea alimentelor din gospodărie. Aceasta promovează utilizarea inteligentă a resurselor, prevenind expirarea produselor prin notificări și generarea de rețete personalizate care integrează alimentele disponibile.

Aplicația permite urmărirea produselor alimentare prin înregistrarea informațiilor esențiale, cum ar fi data de expirare, data deschiderii și numărul recomandat de zile pentru consum după deschidere. Prin funcționalitățile sale, aplicația facilitează atât gestionarea individuală, cât și colaborarea în gospodărie.

***b. Publicul țintă***

Aplicația se adresează unui public larg, incluzând utilizatori individuali și familii care doresc să își gestioneze eficient alimentele și să reducă risipa. Este utilă în special pentru persoanele care:

* Au un interes în sustenabilitate și reducerea risipei alimentare.
* Doresc să primească notificări personalizate despre produsele care se apropie de expirare.
* Au alergii sau preferințe alimentare specifice și doresc rețete adaptate nevoilor lor.
* Își împart responsabilitățile de gestionare a alimentelor cu alte persoane din gospodărie.

***c. Tehnologii utilizate***

Pentru dezvoltarea aplicației ZeroWaste, a fost utilizat un set modern de tehnologii, adaptat cerințelor specifice ale unei aplicații mobile și integrării eficiente între frontend și backend:

* **Frontend**
* **React Ionic**: tehnologia utilizată pentru construirea interfeței utilizatorului, care permite dezvoltarea de aplicații mobile hibride, utilizând un singur cod sursă.
* **Backend**
* **Django**: framework-ul principal utilizat pentru dezvoltarea serverului backend, care permite gestionarea logicii aplicației, a serviciilor REST și a autentificării securizate.
* **Daphne**: utilizat pentru a asigura suport WebSocket în cadrul serverului Django.
* **Celery**: bibliotecă de procesare asincronă, utilizată pentru gestionarea task-urilor programate, cum ar fi trimiterea notificărilor personalizate.
* **Redis**: serverul de mesagerie utilizat ca broker pentru Celery.

1. **ARHITECTURA SISTEMULUI**

***a. Descriere generală***

Aplicația este construită pe o arhitectură client-server care separă logica de interfață de logica de afaceri și procesarea datelor.

* **Frontend**: Implementat cu **React Ionic**, gestionează interfața utilizatorului și comunicarea cu serverul prin intermediul **API-urilor REST** și **WebSocket**. Frontend-ul este proiectat ca o aplicație mobilă hibridă, compatibilă atât cu Android, cât și cu iOS.
* **Backend**: Serverul backend este dezvoltat cu **Django**, expunând endpoint-uri **API REST** prin **Django REST Framework**. **Daphne** este utilizat pentru a gestiona comunicația bidirecțională prin **WebSocket**, necesară pentru funcționalități în timp real.
* **Database**: Aplicația utilizează **SQLite** ca sistem de gestionare a bazei de date. Aceasta stochează datele utilizatorilor, informațiile despre alimente și rețetele.

***b. Diagrame***

Diagrama domeniului:

O imagine care conține text, diagramă, chitanță, număr

Descriere generată automat

Diagrama cazurilor de utilizare:

O imagine care conține text, captură de ecran, Artă de copii, diagramă

Descriere generată automat

***c. Dependențe***

* **Frontend**
* **React Ionic**: framework pentru dezvoltarea interfeței utilizatorului, care necesită instalarea pachetelor *Node.js* și a managerului de pachete npm (Node Package Manager)
* **Axios**: utilizat pentru gestionarea *cererilor HTTP* către serverul backend
* **React Router**: realiează navigarea între paginile aplicației
* **Ionicons**: set de pictograme
* **Capacitor**: framework utilizat pentru construirea pachetelor native Android și iOS;
* **Backend**
* **Django**: framework principal pentru dezvoltarea serverului backend
* **Django REST Framework (DRF)**: crearea și gestionarea *API-urilor de tip REST*
* **Daphne**: *server ASGI* pentru suport *WebSocket*
* **Celery**: gestionarea task-urilor asincrone, utilizând *Redis* ca *broker de mesaje*
* **Redis**: server de mesagerie pentru distribuirea task-urilor și gestionarea sesiunilor
* **SQLite**: baza de date utilizată pentru stocarea locală a datelor
* **backend-ul SMTP** **integrat în Django**: trimiterea email-urilor de verificare, configurat prin modulul *django.core.mail.backends.smtp.EmailBackend*
* **django-cors-headers**: configurarea conexiunii între frontend și backend
* **spaCy**: analiza și procesarea bonurilor fiscale, utilizată pentru a identifica și extrage automat produsele alimentare
* **Django Channels**: gestionarea comunicațiilor asincrone și în timp real, cum ar fi notificările sau colaborarea între utilizatori pe listele de produse
* **Django Environs**: gestionarea variabilelor de mediu

Dependențele sunt gestionate prin:

* **Frontend**: managerul *npm* pentru instalarea și actualizarea pachetelor
* **Backend**: *pip* și *pipenv* pentru instalarea librăriilor Python din fișierul *requirements.txt*

1. **INSTALARE ȘI CONFIGURARE**

***a. Pași pentru instalare***

* ***Frontend***

1. Se instalează Node.js

2. Se clonează repository-ul Github folosind comenzile:

git clone [https://github.com/kriss1809/ZeroWasteFrontEnd.git](https://github.com/kriss1809/ZeroWasteFrontEnd.git%20)

cd ZeroWasteFrontEnd

3. După preluarea proiectului, se vor rula următoarele comenzi în terminal:

npm install  
npm install -g @ionic/cli

* ***Backend***

1. Se instalează Python (cel puțin versiunea 3.8)

2. Se clonează repository-ul Github folosind comenzile:

git clone <https://github.com/kevinscrg/ZeroWasteBackEnd.git>

cd ZeroWasteBackEnd

3. După preluarea proiectului, se vor rula următoarele comenzi în terminal:

pip install -r requirements.txt

python manage.py makemigrations  
python manage.py migrate

* ***Modul AI***

1. Se instalează Python (cel puțin versiunea 3.8)

2. Se clonează repository-ul Github folosind comenzile:

git clone <https://github.com/kriss1809/ZeroWasteAI.git>

cd ZeroWasteAI

3. După preluarea proiectului, se vor rula următoarele comenzi în terminal:

pip install spacy

python -m spacy download en\_core\_web\_md

pip install pandas

pip install numpy

pip install scikit-learn

pip install opencv-python

pip install openpyxl

pip install websocket-client

***b. Comenzi pentru rulare locală***

Înainte de rulare, trebuie modificate adresele IP în proiectele pentru frontend și backend în funcție de adresele IP locale. Se recomadă folosirea Visual Studio Code atât pentru frontend, cât și pentru backend.

* ***Frontend***

Pentru pornirea aplicației, se va executa în terminal comanda **ionic serve**.

***Posibilă eroare:***

*The file ...\npm\ionic.ps1 cannot be loaded because script execution is disabled on this system.*

***Soluție:***

Se deschide Powershell în modul administrator și se rulează **Set-ExecutionPolicy RemoteSigned**, apoi se poate rula din nou **ionic serve** pentru a porni aplicația.

* ***Backend***

Pentru pornirea aplicației, se vor executa în terminale comenzile:

**Terminal 1:**

cd zerowaste

daphne -p **nr\_port** zerowaste.asgi:application

**Terminal 2**:

**Linux:** sudo service redis-server start

**Windows:** redis-server.exe

**Terminal 3**:

cd zerowaste

celery -A zerowaste.celery\_app worker -l info --pool=solo

În final, se rulează proiectul dintr-un mediu de dezvoltare.

* ***Modul AI***

Se rulează proiectul dintr-un mediu de dezvoltare. (ex: PyCharm)

1. **STRUCTURA ȘI IMPLEMENTAREA CODULUI**

***a. Backend***

***i. Server***

***1. Structura proiectului***

Proiectul backend este organizat modular în mai multe aplicații Django, fiecare având rolul de a gestiona o entitate specifică. Această arhitectură modulară permite o organizare clară, scalabilitate și întreținere facilă.

***/zerowaste*** este directorul principal al proiectului Django și include configurațiile generale.

**Fișiere principale**:

* **settings.py**: conține configurările principale ale proiectului (aplicații instalate, baze de date, middleware-uri, configurații REST API și JWT)
* **urls.py**: folosie pentru definirea rutelor principale ale proiectului
* **wsgi.py și asgi.py**: sunt fișiere de configurare pentru serverul WSGI/ASGI, necesare pentru rularea aplicației
* **celeryconfig.py**: conține configurările pentru Celery (task-uri asincrone)

**Aplicația api** are ca scop gestionareautilizatorilor și a autentificării.

* **models.py**: definește modelul de utilizator personalizat, clasa pentru rețete și conexiunea dintre ele
* **serializers.py**: serializatoare pentru transformarea datelor Python în JSON și invers
* **views.py**: endpoint-uri pentru autentificare, ieșire din cont, creare cont, ștergere cont, verificare email, schimbare parolă, parolă uitată, vizualizare colaboratori, actualizare (pentru setările din profil – alergii, preferințe, notificări, dark mode) și pentru operații efectuate pe rețete (ex: filtrare, căutare, refresh)
* **urls.py**: rutele API pentru operațiile enumerate mai sus
* **consumers.py**: gestionarea comunicării WebSocket pentru notificări

**Aplicația product** are ca scopgestionarea produselor utilizatorului

* **models.py**: definirea modelelor pentru produs și pentru lista de produse
* **views.py**: endpoint-uri pentru operații pe lista de produse, incluzând gestionarea bonurilor fiscale primite pentru a extrage alimentele (procesare făcută prin Celery)
* **serializers.py**: serializatoare pentru transformarea datelor Python în JSON și invers
* **urls.py**: rutele specifice pentru operațiile aferente
* **services/receipt\_processing.py**: procesarea bonurilor fiscale folosind OCR
* **services/tasks.py**: task-uri asincrone pentru adăugarea produselor din bonuri (folosind Celery) și notificarea utilizatorului prin WebSocket

**Migrations** include fișierele generate automat pentru modificări în schema bazei de date.

***2. Fluxul de date***

1. Cererile *http* sunt preluate în *views.py*, unde sunt procesate și datele sunt trimise către frontend prin endpoint-uri REST.
2. WebSocket-urile sunt utilizate pentru trimiterea notificărilor în timp real
3. Task-urile lungi, precum procesarea bonurilor fiscale, sunt delegate lui Celery (acestea sunt executate în fundal)
4. Fișierul *receipt\_processing.py* implementează funcționalitatea pentru prelucrarea imaginilor folosind OCR

***3. Gestionarea autentificării și delogării***

Aplicația backend utilizează *Django REST Framework* (DRF) și *JWT* (JSON Web Tokens) prin *rest\_framework\_simplejwt* pentru a gestiona autentificarea și delogarea utilizatorilor.

Modelul *User* extinde *AbstractBaseUser* și adaugă câmpuri suplimentare specifice aplicației. Metoda tokens() din cadrul modelului utilizează *RefreshToken.for\_user()* pentru a genera token-urile de acces și refresh JWT.

***LoginSerializer*** și ***LogoutSerializer***validează și gestionează datele primite (de la *LoginView* și *LogoutView*) pentru autentificare și delogare.

Metoda *validate()* din cadrul *LoginSerializer* verifică dacă utilizatorul există și dacă parola este corectă folosind check\_password. În cazul unui cont al cărui email nu a fost verificat (is\_verified == False), returnează o eroare.Dacă autentificarea este validă, returnează token-urile JWT (acces și refresh).

Metoda *save()* din cadrul *LogoutSerializer* adaugă token-ul în lista nevalidă (blacklist) utilizând *RefreshToken.blacklist()* pentru a preveni reutilizarea.

***ii. Modul AI***

Modulul AI este responsabil pentru prelucrarea și analiza datelor legate de rețete și preferințele utilizatorilor. Este organizat astfel încât să integreze mai multe funcționalități prin procese sincrone și asincrone utilizând WebSocket-uri și algoritmi de procesare a datelor.

***1. Structura proiectului***

Aplicația conține un fișier *.py* și un director cu un fișier *.xlsx* din care sunt extrase rețetele:

* **Main.py**
* **Files/recipesAllergensPreferences.xlsx**

***2. Fluxul de date și descrierea funcționalităților***

**a. Preluarea datelor**

Datele sunt citite dintr-un fișier Excel (*Files/recipesAllergensPreferences.xlsx*), care conține informații despre rețete (precum alergeni, preferințe alimentare, timp de gătire, etc.)

**b. Prelucrarea datelor**

Datele sunt filtrate și prelucrate pe baza criteriilor utilizatorului cu ajutorul funcțiilor:

* **filter\_recepies**: filtrează rețetele eliminându-le pe cele care conțin alergenii specificați de utilizator, le selectează pe cele care corespund preferințelor sale alimenatre și apoi le filtrează după timp, tip și dificultate (filtrare iterativă, se returnează un *DataFrame*)
* **use\_expiring\_ingredients**: prioritizează rețetele care conțin ingrediente ce expiră în curând și utilizează un sistem de calcul pentru a potrivi și sorta rețetele în funcție de relevanță (normalizare listă, calcul prioritate, sortare după prioritate)
* **recomendations**: generează recomandări pe baza rețetelor preferate de utilizator, utilizând tehnici de procesare a textului și calculul similarității cosinus (vectorizare ingrediente utilizând *MultiLabelBinarizer*, procesare text ingrediente utilizând *CountVectorizer*, calcul *similaritate cosinus*, sortare rețete)

**c. Răspunsul către client**

Datele procesate sunt trimise utilizatorului prin *WebSocket*, utilizând mesaje *JSON* care includ id-urile rețetelor filtrate și mesaje de status.

***3. Integrarea Websocket-urilor***

**a. Ascultarea mesajelor**

Funcția *websocket\_listener* ascultă mesaje de la server și le adaugă într-o coadă (queue) pentru procesare.

**b. Procesarea cererilor**

Funcția *process\_request* gestionează fiecare cerere primită prin WebSocket:

* Extrage informațiile relevante din payload (de exemplu: alergeni, preferințe, produse care expiră).
* Apelează funcțiile de filtrare și recomandare pentru a prelucra cererea.
* Trimite înapoi rezultatele sub formă de JSON prin WebSocket.

**c. Gestionarea thread-urilor**

Modulul utilizează threading și *ThreadPoolExecutor* pentru a procesa mai multe cereri simultan. Funcția *process\_requests\_worker* este un worker care procesează cererile din coadă până la închiderea aplicației.

***b. Frontend***

* 1. ***Structura directoarelor și a fișierelor***

Aplicația este structurată pe mai multe directoare care organizează codul și resursele în funcție de funcționalitate:

**Directoare principale**

* **.vscode/**: conține configurările specifice editorului Visual Studio Code
* **cypress/**: include fișierele și configurările pentru testele de integrare utilizând framework-ul Cypress
* **node\_modules/**: conține toate pachetele și dependențele instalate utilizând npm
* **public/**: conține fișierele publice, accesibile direct de utilizator:
  + - **images/**: resurse de tip imagine utilizate în aplicație
    - **favicon.png**: iconița aplicației pentru browser
    - **manifest.json**: fișier de configurare pentru aplicația web progresivă (PWA).
* **src/**: conține codul sursă al aplicației, având mai multe subdirectoare și fișiere:
  + - **components/**: include componentele reutilizabile din interfața utilizatorului, cum ar fi formulare, liste, butoane, etc.
    - **entities/**: stochează definiții sau modele ale entităților utilizate în aplicație, cum ar fi structurile de date.
    - **pages/**: conține fișierele care definesc paginile principale ale aplicației.
    - **services/**: include funcții sau clase utilizate pentru a interacționa cu API-urile backend-ului
    - **theme/**: conține fișiere css utilizate pentru stilizarea aplicației
    - **App.tsx** și **main.tsx**: punctele principale de intrare în aplicație. (*App.tsx* definește structura aplicației, iar *main.tsx* inițializează aplicația)
  1. ***Fluxul de date***
  2. Datele sunt preluate din backend prin *apeluri HTTP* sau notificări *WebSocket.*

**Exemple:**

* Lista de produse este preluată folosind *GetProductList().*
* Lista de rețete este încărcată progresiv cu *GetRecipes(limit, offset)* pentru a afișa rețetele în mod paginat.
  1. Datele preluate sunt stocate în contexte, fiecare context gestionând o secțiune specifică (*ProductListContext* pentru produse, *RecipesContext* pentru rețete, *AuthContext* pentru autentificare).
  2. Componentele din pagini accesează aceste date prin hooks (*useContext*).
  3. Orice modificare a stării în Context API declanșează re-renderizarea componentelor relevante.

***3. Componentele aplicației***

Directorul *src/components/* conține componentele reutilizabile din aplicație, care sunt folosite pentru a construi interfața utilizatorului. Aceste componente sunt organizate astfel încât să fie reutilizabile și să se ocupe de părți specifice ale interfeței sau funcționalităților.

|  |  |
| --- | --- |
| **AccountSettings.tsx**  Permite utilizatorului să vizualizeze adresa de email asociată contului cu care este autentificat, să schimbe parola sau să șteargă contul.  Este utilizat în pagina de profil. |  |
| **AddItem.tsx**  Formular pentru adăugarea unui produs nou în lista de produse. Include câmpuri pentru numele produsului, data de expirare, data deschiderii și numărul de zile recomandat pentru consum.  Este utilizat în pagina principală. |  |
| **CollaboratorsModal.tsx**  Modal care afișează lista colaboratorilor care au acces la lista de produse curentă.  Este utilizat în pagina principală. |  |
| **ItemCard.tsx**  Reprezintă un card individual pentru un produs, ce afișează informații precum numele produsului și data de expirare și pune la dispoziție butoane pentru consumarea/eliminarea unui produs  Este utilizat în pagina principală. |  |
| **Menu.tsx**  Componentă pentru meniul principal de navigare al aplicației. Include linkuri către paginile principale (profilul utilizatorului, lista de produse și rețetele)  Este utilizat în toate cele 3 pagini principale. |  |
| **ProfileSettings.tsx**  Permite utilizatorului să configureze notificările (zile și ore), alergiile și preferințele sale alimentare, precum și modul întunecat. Include o funcționalitate pentru obținerea unui cod pentru lista sa (pe care îl poate copia), precum și pentru aderarea la o listă având un cod dat.  Este utilizată în pagina de profil. |  |
| **RecipeCard.tsx**  Reprezintă un card individual pentru o rețetă, ce include informații precum denumirea rețetei, timpul de preparare, și nivelul de dificultate, dar și două butoane pentru a da like/dislike unei rețete.  Este utilizat în pagina dedicată rețetelor. |  |
| **ThemeContext.tsx**  Componentă pentru gestionarea temei aplicației (mod întunecat/lumină).  Este utilizat în toate paginile aplicației. |  |
| **UploadReceiptModal.tsx**  Permite utilizatorului să încarce un bon fiscal pentru a identifica automat produsele alimentare și a le adăuga în lista de produse.  Este utilizat în pagina principală. |  |

***4. Paginile aplicației***

Directorul *src/pages/* conține fișiere ce reprezintă paginile aplicației, ce includ diverse componente.

|  |  |
| --- | --- |
| **Home.tsx**  Pagina principală a aplicației, în care poate fi vizualizată lista produselor din gospodărie. În cadrul acestei pagini, utilizatorul poate:  - modifica/șterge/marca drept consumat un produs existent  - adăuga manual un produs  - încărca un bon fiscal pentru ca produsele alimentare să fie extrase automat  - consulta lista colaboratorilor listei  - naviga către alte pagini |  |
| **Login.tsx**  Pagina de login este reprezentată de un formular care cere emailul și parola pentru a realiza autentificarea într-un cont existent. Dacă utilizatorul nu are un cont, poate crea unul nou. În cazul în care utilizatorul uită parola, are posibilitatea de a cere resetarea ei. |  |
| **NotFound.tsx**  Pagină afișată în momentul în care utilizatorul accesează o resursă inexistentă. |  |
| **Profile.tsx**  Pagina de profil a utilizatorului, unde acesta poate beneficia de funcționalitățile din componentele AccountSettings și ProfileSettings. De asemenea, utilizatorul se poate loga utilizând un buton existent în această pagină și poate naviga către alte pagini. |  |
| **Recipes.tsx**  Pagina dedicată rețetelor, în care este afișată o listă de rețete (sortate în funcție de preferințele utilizatorului pe baza aprecierilor și pe baza produselor din lista sa care se apropie de termenul de expirare). În această pagină, utilizatorul poate:  - căuta o rețetă  - filtra rețetele după anumite criterii  - reîncărca lista de rețete (pentru a vedea modificările realizate în ultima oră în urma adăugării de produse / aprecierii de rețete noi)  - naviga către alte pagini |  |
| **SetNewPassword.tsx**  Pagină ce permite resetarea parolei utilizatorului după ce acesta a indicat faptul că și-a uitat parola și a accesat link-ul primit pe email. |  |
| **Signup.tsx**  Pagina de signup este reprezentată de un formular care cere un email valid și o parolă pentru crearea unui cont. În urma completării acestui formular, utilizatorul primește un email ce conține un link pentru validarea adresei de email. |  |
| **SuccessfullyCreatedAccount.tsx**  Pagină ce anunță utilizatorul că email-ul său a fost verificat, iar contul a fost creat cu succes și poate continua logându-se în cont. |  |
| **SuccessfullyDeletedAccount.tsx**  Pagină ce anunță utilizatorul că și-a șters contul cu succes în urma introducerii parolei sale. |  |

***5. Entitățile aplicației***

Directorul *src/entities/* conține cele 3 entități utilizate în aplicație (Product, User, Recipe):

|  |  |  |
| --- | --- | --- |
| export interface Product {  id: number;  name: string;  best\_before: string;  opened: string;  consumption\_days: string;  } | export interface User {  email : string;  preferred\_notification\_hour : string;  preferences: string[];  allergies: string[];  notification\_day: number;  dark\_mode: boolean;  } | export interface Recipe {  id: number;  name: string;  link: string;  image: string;  difficulty: number;  time: number;  recipe\_type: string;  rating: boolean|null;  } |

***6. Serviciile aplicației***

Directorul *src/services/* conține fișiere ce fac conexiunea între backend și frontend utilizând cereri HTTP și comunicare prin intermediul Websocket-urilor.

*authProvider*, *ProductListProvider*, *RecipesProvider* sunt contexte care permit aplicației să aibă acces (de oriunde, fără duplicarea codului sau introducerea de parametri suplimentari) la funcționalitățile (de pe backend):

* legate de contul utilizatorului (împreună cu stocarea utilizatorului logat)
* listei de produse a utilizatorului curent
* listei de rețete a utilizatorului curent

*WebSocketProvider* permite aplicației să comunice cu serverul prin mesaje în mod bidirecțional, astfel încât propagarea schimbărilor să fie realizată în timp real. De exemplu, atunci când un utilizator adaugă sau modifică un produs în lista sa, aceste modificări sunt trimise automat către server, iar serverul notifică toți colaboratorii listei partajate prin mesaje WebSocket. Acest mecanism asigură sincronizarea în timp real între utilizatori și permite gestionarea eficientă a datelor partajate.

***7. Stilizarea aplicației***

Directorul *src/theme/* conține fișiere *.css* împărțite în funcție de pagini/componente, ce oferă un aspect vizual plăcut interfeței. De asemenea, conține fișierul *.css* cu stiluri pentru modul întunecat.

***8. Navigarea între pagini***

Navigarea este implementată folosind React Router integrat cu Ionic React, asigurând o gestionare ușoară a rutelor și o experiență de utilizare fluidă. Structura rutelor este centralizată în componenta App.tsx, care definește toate paginile aplicației și fluxurile de navigare.

Aplicația Ionic este inițializată cu setupIonicReact(), iar componentele IonApp și IonReactRouter sunt utilizate pentru a configura structura de bază.

Navigarea este gestionată cu ajutorul IonRouterOutlet, care permite comutarea între paginile aplicației în funcție de rutele definite.

Rutele sunt configurate folosind componentele Route și Switch din React Router. Exemple de rute definite:

* ***/home*** - pentru pagina principală a utilizatorului logat
* ***/recipes*** - pentru afișarea rețetelor
* ***/login*** - pentru autentificare
* ***/signup*** - pentru înregistrare
* ***/profile*** - pentru setările contului utilizatorului
* ***/set-new-password*** - pentru resetarea parolei
* ***/successfully-created-account*** și ***/successfully-deleted-account*** - pentru pagini de confirmare
* ***/*** - redirecționează către /login dacă utilizatorul nu este autentificat
* ***\**** - pagină NotFound pentru rute inexistente

Aplicația folosește ThemeProvider pentru a aplica tema luminoasă sau întunecată la nivel global. Schimbarea modului (luminos/întunecat) este gestionată din context (ThemeContext) și aplicată pe elementul <body> folosind clasa CSS dark-mode.

***9. Gestionarea autentificării și delogării***

**Procesul de autentificare**:

* Utilizatorul introduce email-ul și parola în formularul de autentificare.
* Cererea este trimisă către backend folosind funcția loginUser din *services/apiClient.tsx* printr-o cerere *HTTP POST.*
* Backend-ul validează credențialele și returnează două token-uri:
  + **Token de acces**: folosit pentru autorizarea cererilor la API.
  + **Token de reîmprospătare**: utilizat pentru obținerea unui nou token de acces când cel curent expiră.

**Stocarea token-urilor:**

* Token-ul de acces este stocat în *sessionStorage*, deoarece are o durată de viață limitată și trebuie eliminat când utilizatorul închide aplicația.
* Token-ul de reîmprospătare este stocat în *localStorage* pentru a persista între sesiuni.

**Autentificarea în aplicație:**

* Starea autentificării este gestionată de AuthContext (din authProvider.tsx), care utilizează useState pentru a stoca:
  + Informațiile utilizatorului (profilul).
  + Starea autentificării (isAuthenticated).
  + Token-ul de acces (accessToken).
* La fiecare încărcare a aplicației, AuthProvider verifică existența unui token de acces în sessionStorage. Dacă acesta există, încearcă să preia profilul utilizatorului folosind getUserProfile.

**Reîmprospătarea token-ului de acces:**

* Dacă token-ul de acces expiră, aplicația apelează endpoint-ul */api/token/refresh/* pentru a obține un nou token de acces folosind token-ul de reîmprospătare din localStorage.
* Funcția RefreshAccessToken gestionează acest proces.

**Logout:**

Când utilizatorul dorește să se delogheze, aplicația:

* + Apelează endpoint-ul /logout/ pentru a invalida token-urile pe backend.
  + Șterge token-urile din sessionStorage și localStorage.
  + Resetează starea autentificării și profilul utilizatorului în AuthContext.

1. **FUNCȚIONALITĂȚI**

Conform diagramei cazurilor de utilizare de la punctul 2b, aplicația ZeroWaste oferă următoarele funcționalități clientului:

* Creare cont (cu verificarea emailului)
* Ștergere cont
* Schimbare parolă
* Schimbare parolă uitată
* Logare
* Delogare
* Vizualizare produse din listă
* Adăugare produs în listă
* Modificare produs existent în listă
* Ștergere produs
* Consumare produs
* Căutare produs
* Încărcare bon fiscal
* Vizualizare colaboratori la listă
* Obținere cod listă
* Alăturare la o listă folosind un cod
* Vizualizare profil
* Setare alergeni
* Setare preferințe alimentare
* Setare zile pentru notificări
* Setare oră pentru notificări
* Vizualizare notificări primite
* Vizualizare rețete
* Filtrare rețete
* Căutare rețetă
* Apreciere rețetă

1. **SECURITATE**

**Utilizarea token-urilor JWT:**

* Token-urile sunt utilizate pentru a identifica utilizatorul și a autentifica cererile către backend.
* Token-urile de acces au o durată de viață scurtă, iar cele de reîmprospătare sunt utilizate pentru a evita o reconectare frecventă.

**Stocarea token-urilor:**

* Token-ul de acces este stocat temporar în sessionStorage pentru a preveni expunerea în cazul unui atac XSS.
* Token-ul de reîmprospătare este stocat în localStorage, deoarece trebuie să persiste între sesiuni.

**Cereri HTTP securizate:**

* Fiecare cerere HTTP include token-ul de acces în header-ul Authorization: Bearer <token>.

**Protecția datelor utilizatorului:**

* Backend-ul validează token-urile și verifică autorizațiile utilizatorului înainte de a permite accesul la date.

**Gestionarea erorilor de autentificare:**

* Dacă un token de acces este invalid sau expiră, aplicația încearcă să obțină un token nou. Dacă acest proces eșuează, utilizatorul este delogat automat.

1. **TESTARE**

Pentru serverul dezvoltat pe Backend, se regăsesc fișiere test.py în fiecare aplicație. Acestea se rulează folosind comenzile:

cd zerowaste

python manage.py test

1. **JURNALIZARE**

Toate acțiunile care au loc în cadrul aplicației sunt descrise în fișierul de log *zerowaste.log,* care se găsește în proiectul aplicației Django. Acest fișier este rescris la fiecare rulare a serverului și configurările sale se găsesc în *settings.py.*

*Exemple de linii din fișier:*

INFO 2024-12-30 19:02:34,689 daphne.server Listening on TCP address 192.168.100.186:8000

DEBUG 2024-12-30 19:03:05,857 daphne.http\_protocol HTTP b'OPTIONS' request for ['192.168.100.201', 45304]

DEBUG 2024-12-30 19:03:05,858 daphne.http\_protocol HTTP 200 response started for ['192.168.100.201', 45304]

DEBUG 2024-12-30 19:03:05,858 daphne.http\_protocol HTTP close for ['192.168.100.201', 45304]