



UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENȚĂ: Informatică

LUCRARE DE LICENȚĂ

COORDONATOR:
Lect. Dr. Bonchiș Cosmin

ABSOLVENT:
Baraboi Gabriel

TIMIȘOARA
2022

**UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENȚĂ: Informatică**

Smart Cooker – Machine Learning Application

COORDONATOR:
Lect. Dr. Bonchiș Cosmin

ABSOLVENT:
Baraboi Gabriel

**TIMIȘOARA
2022**

Abstract

In my bachelor's thesis, I suggest a method for recommending recipes that uses image recognition to identify food ingredients, particularly fruits and vegetables. I created an app for the Android operating system that can identify culinary ingredients from pictures and provides the user with a list of recipes they may cook using the ingredients they have submitted. To identify ten categories of food ingredients, I created a convolutional neural network model and trained it for image classification. I achieved an outstanding 92.82% accuracy rate. Based on recognized ingredients, I created an algorithm to suggest recipes.

Cuprins

1 Introducere	6
1.1 Motivație	6
1.2 Obiectivele lucrării	6
1.3 Soluție și contribuție autor	6
1.4 Structura lucrării	7
2 Descrierea Problemei	8
2.1 General	8
2.2 Recunoașterea ingredientelor	8
2.3 Propunerea rețetelor	8
2.4 Abordări existente	8
3 Arhitectura și implementarea aplicației	9
3.1 Arhitectura aplicației	9
3.2 Machine Learning	9
3.2.1 Arhitectura	10
3.2.2 Tehnologii utilizate	11
3.3 Frontend	11
3.3.1 Structura și arhitectura	11
3.3.2 Tehnologii utilizate	12
3.3.3 Interfața utilizator	13
3.4 Backend	15
3.4.1 Arhitectura	15
3.4.2 Tehnologii utilizate	16
3.4.3 Structura	17
3.5 Baza de date	18
3.5.1 Baza de date bazată pe documente	18
3.5.2 MongoDB	19
3.5.3 MongoDB Atlas	19
3.5.4 Indecși	19
4 Detalii de implementare	20
4.1 Detectarea ingredientelor	20
4.2 Recomandarea rețetelor	22
5 Concluzii și direcții viitoare de dezvoltare	26
5.1 Sumar al rezultatelor	26
5.2 Discutarea rezultatelor obținute	26
5.3 Direcții viitoare	27

Capitolul 1

Introducere

1.1 Motivație

Lucrarea mea de licență are ca scop simplificarea procesului de preparare a bucatelor, și anume prin recomandarea rețetelor culinare ce se bazează pe produsele pe care le are utilizatorul în bucătăria sa. Există situații când nu mai ai idei pentru ceea ce ai dori să mănânci și pierzi mult timp navigând pe internet cu speranța de a găsi rețeta perfectă. Aplicația pe care am dezvoltat-o rezolvă problema respectivă, oferind posibilitatea de a fotografia produsele de care dispui, analiza fiecare ingredient și recomanda ce ai putea prepara din ele.

Soluția propusă de mine este reprezentată de o aplicație mobilă bazată pe stack-ul MERN (MongoDB, ExpressJS, React Native și NodeJS), acesta este un stack de dezvoltare modern, și este folosit de o mulțime de companii.

1.2 Obiectivele lucrării

Obiectul principal al sistemului propus este de a ajuta utilizatorii să decidă ce pot găti cu ingredientele disponibile. Utilizatorul ar putea folosi aplicația și în timpul cumpărăturilor, nu doar când se află acasă. Folosind acest sistem, utilizatorii ar putea să-și construiască un plan despre ceea ce vor putea găti în acea săptămână, în baza recomandărilor aplicației.

1.3 Soluție și contribuție autor

Contribuția mea la rezolvarea problemei, constă în dezvoltarea aplicației pentru sistemul Android, și învățarea rețelei neuronale convolutionale. Am dezvoltat aplicația mobilă folosind stack-ul MERN, am creat un API, pentru comunicarea clientului cu serverul. Am folosit o rețea neuronală pentru a o învăța să recunoască unele ingrediente culinare. Detaliile legate de implementarea frontend și backend, arhitectura sistemului, recomandarea rețetelor și recunoașterea ingredientelor se pot găsi la capitolele 3 și 4.

1.4 Structura lucrării

Structura lucrării este următoare:

1. În primul capitol, „Introducere”, este prezentată problema și motivația pentru care se realizează acest proiect. Pe lângă asta, se specifică obiectivele și contribuția autorului.
2. Al doilea capitol, constă din descrierea problemei, și împărțirea ei în două părți. Pe lângă asta, se analizează aplicațiile similare existente pe piață.
3. Al treilea capitol „Arhitectura și implementarea aplicației”, constă din descrierea arhitecturii proiectului. Se prezintă arhitectura frontend-ului și backend-ului, tehnologiile utilizate și multe altele.
4. Capitolul patru se folosește pentru a prezenta unele detalii de implementare a aplicației.
5. În ultimul capitol se realizează un sumar al rezultatelor obținute, sunt prezentate direcțiile viitoare pentru acest proiect.

Capitolul 2

Descrierea Problemei

2.1 General

Problema poate fi împărțită în două părți, una fiind problema recunoașterii ingredientelor, și alta propunerea rețetelor.

2.2 Recunoașterea ingredientelor

Pentru a rezolva această problemă, voi lua un model pre-antrenat a unei rețele neuronale conluvționale, și îl voi antrena pentru a detecta ingredientele introduse de utilizator. Inițial modelul va fi învățat să detecteze doar câteva tipuri de ingrediente.

2.3 Propunerea rețetelor

În dependență de ingrediente ce vor fi identificate după prelucrarea imaginii, aplicația va propune o listă de rețete. Pentru a elabora lista, inițial în baza de date vor fi introduse o mulțime de rețete. Lista cu ingredientele detectate va fi comparată cu fiecare rețetă în parte, pentru a găsi cele mai bune rețete după potrivirea ingredientelor.

2.4 Abordări existente

În ultimul timp, detectarea ingredientelor sau propunerea rețetelor au început să primească atenție din ce în ce mai mare

Kawano, Y. și K. Yanai [1], au prezentat o abordare pentru smartphone, un sistem de recunoaștere în timp real a bucatelor. Au folosit două metode de recunoaștere a imaginilor în timp real, una este o combinație dintre bag-of-feature și histograma culorilor. Iar a doua metodă, este descriptorul HOG și descriptorul de patch de culoare Fisher Vector. Ei au atins o precizie de 79.2%.

Capitolul 3

Arhitectura și implementarea aplicației

3.1 Arhitectura aplicației

Această aplicație este formată din mai multe părți, client (frontend), server (backend), baza de date și rețea neuronală convoluțională.

Frontend-ul, este dezvoltat cu librăria React Native, care este un framework pentru a crea aplicații native pe Android și iOS. Clientul comunică cu serverul prin intermediul protocolului HTTP, cu ajutorul rutelor REST create de API-ul serverului.

Backend-ul rulează prin intermediul mediului de execuție NodeJS, iar API-ul este creat cu ajutorul framework-ului ExpressJS. Serverul interacționează cu scriptul Python, care recunoaște imaginile transmise de client. Pe lângă asta, backend-ul mai interacționează și cu baza de date.

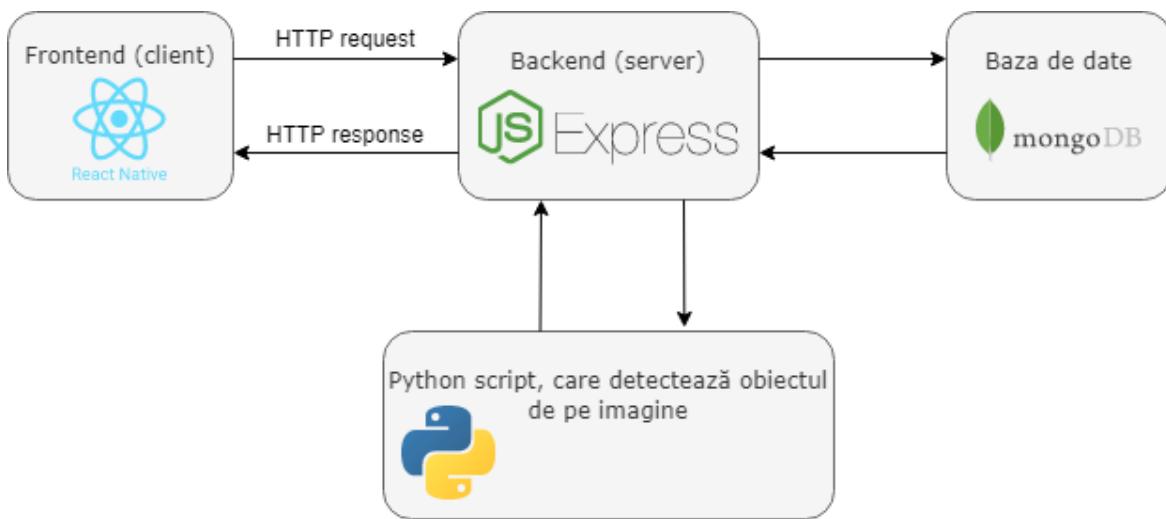


Figura 3.1: Arhitectura aplicației

3.2 Machine Learning

Pentru detectarea automată a ingredientelor de pe imaginile încărcate de utilizator, trebuie să folosesc Machine Learning. Inițial am încercat să folosesc soluția de la

google, care se numește Google Vision, care este un model pre-antrenat de clasificare a imaginilor, și nu numai. Doar că după o mulțime de teste, am ajuns la concluzia că acest instrument, deseori nu poate detecta și clasifica ingredientul corect.

Pentru a păsi peste această problemă, am fost nevoie să cauți alte soluții pentru problema mea, și am ales să antrenez o rețea neuronală convecțională. Am ales un model pre-antrenat pe un set de date foarte mare, și l-am folosit pentru a antrena pe un set de date mai mic.

3.2.1 Arhitectura

Am folosit Transfer Learning pentru instruirea modelului meu. Transfer Learning înseamnă reutilizarea unui model pre-antrenat pentru un set de date mai mic. Ca model pre-antrenat, am folosit GoogLeNet. Transfer Learning este extrem de populară pentru că poți antrena rețele neuronale cu date relativ puține. Rețeaua pre-antrenată poate clasifica imaginile în 1000 de categorii, dar eu voi avea nevoie doar de câteva categorii, în dependență de numărul de ingrediente. Însă, modelul pre-antrenat nu poate clasifica imaginile doar după categoriile de care avem nevoie noi. Am creat un model nou pentru acest scop.

Setul de date

Pentru a învăța modelul nostru să detecteze ingredientele de care avem nevoie, a trebuit să creez un set de date. Am folosit setul de date fruit360 [3], de unde am luat imagini pentru 10 ingrediente diferite. Am avut în total 10262 imagini pentru antrenarea modelului.

Modelul GoogLeNet

GoogLeNet [9] este o rețea neuronală convecțională dezvoltată de către google și mai multe universități. Arhitectura rețelei are o adâncime de 22 de straturi. Modelul a fost conceput pentru a ține cont de eficiență de calcul, adică să poată fi rulată pe dispozitive cu resurse de calcul reduse. Acest model preia imagini de dimensiunea 224x224 cu canalele RGB. În figura 3.2 este reprezentată arhitectura modelului.

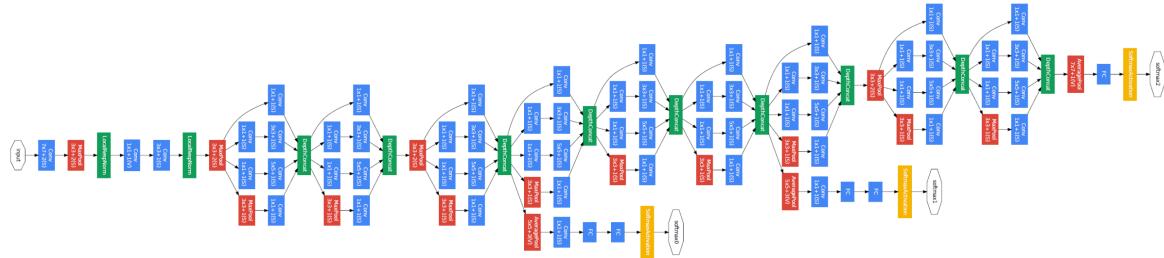


Figura 3.2: Straturile rețelei GoogLeNet

3.2.2 Tehnologii utilizate

PyTorch

PyTorch [7] este un framework open-source, bazat pe librăria Torch și limbajul de programare Python. Se folosește pentru a crea aplicații cu Computer Vision, Machine Learning.

Torchvision

Torchvision [8] este o librărie PyTorch, care constă din seturi de date, modele pre-antrenate, transformări ale imaginilor pentru Computer Vision. Consta din mai multe instrumente, cum ar fi: rețele de instruire pentru detectarea obiectelor, clasificarea imaginilor, clasificarea video, și altele. În acest proiect am folosit torchvision pentru clasificarea imaginilor.

Pillow

Pillow este o librărie bazată pe Python, care ne aduce mai multe funcționalități de procesare a imaginilor.

3.3 Frontend

3.3.1 Structura și arhitectura

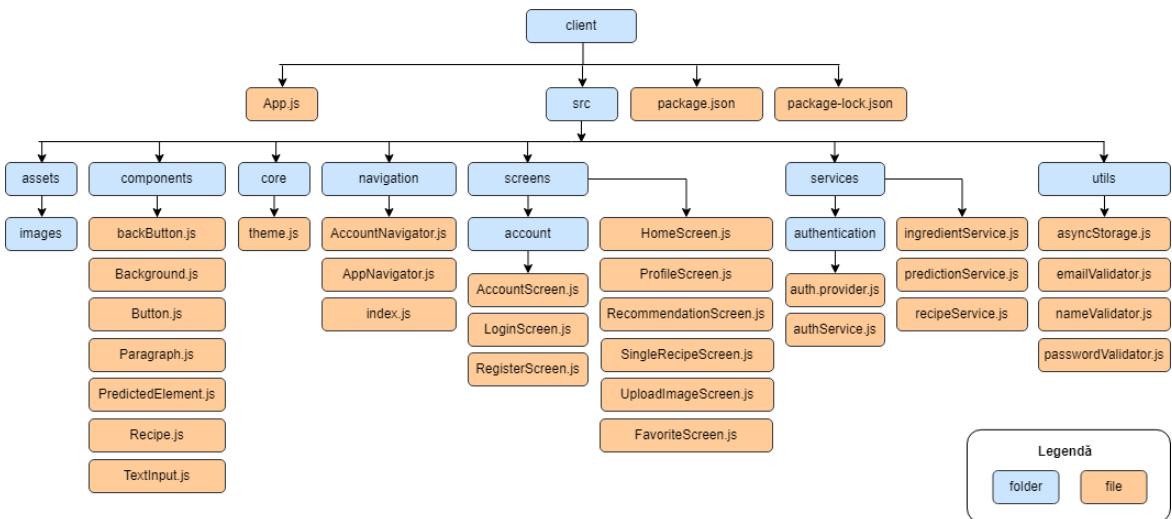


Figura 3.3: Structura părții de frontend

Structura frontend-ului este una foarte simplă, gândită pentru a respecta cât mai mult principiul separation of concerns. Astfel, partea clientului are această structură:

- App.js este fișierul care conține principala componentă a aplicației. Aici se ran-dează componentele aplicației, se realizează navigarea folosind pachetul react-navigation.

- Fișierele package.json și package-lock.json, dețin metadatele relevante pentru proiect și sunt folosite pentru gestionarea dependențelor proiectului, scripturilor de rulare, versiuni și multe altele.
- .gitignore, este fișierul care îi spune lui git ce foldere/fișiere să ignore atunci când încărcați proiectul pe GitHub.
- src, este folderul principal al aplicației, care conține proiectul în sine, este împărțit în mai multe foldere:
 - assets, este folderul care conține toate imaginile, fonturi, etc.
 - components, este folderul care conține toate componentele comune ale aplicației, care pot fi utilizate de mai multe ori (cum ar fi fundalul, un buton, un paragraf, etc.)
 - navigation, este folderul unde se află fișierele principale pentru navigare. Avem două fișiere de genul, AccountNavigator.js, și AppNavigator.js, ele sunt accesate în dependență dacă suntem logați sau nu.
 - * AccountNavigator.js răspunde de rutile utilizatorului atunci când nu este logat (cum ar fi înregistrarea, autentificarea, etc.)
 - * AppNavigator.js răspunde de rutile utilizatorului atunci când este logat. Acest navigator este creat din două componente, StackNavigator și BottomTabNavigator, primul conține toate paginile aplicației, al doilea conține paginile meniului care apare jos în aplicație.
 - screens, este folderul care conține toate paginile aplicației.
 - services, este folderul care conține toate fișierele care se folosesc pentru comunicarea cu serverul, cu ajutorul pachetului axios. În aceste fișiere se creează funcțiile pentru operațiile CRUD (create, read, update, delete). Ele sunt exportate cu scopul de a fi utilizate în alte componente ale aplicației. De asemenea, aici se află funcția care transmite către server token-ul, pentru a verifica dacă utilizatorul are dreptul pentru a accesa o resursă.
 - utils, este folderul care conține funcțiile utile pentru aplicație, cum ar fi funcțiile care validează parola, email-ul sau numele utilizatorului, atunci când se înregistrează sau loghează

3.3.2 Tehnologii utilizate

React Native

React Native este un framework JavaScript, conceput pentru a construi aplicații pe mai multe platforme precum iOS și Android utilizând exact aceeași bază de cod.

React Native folosește JavaScript pentru a compila interfața de utilizator a aplicației, dar utilizând vizualizarea nativă a sistemului de operare. Pentru funcții mai complexe, permite implementarea codului în limbajele native ale sistemului de operare (Swift și Objective-C pentru iOS sau Java și Kotlin pentru Android). Printre avantajele acestui framework, se numără:

- **Compatibilitate** - cu acest framework, 95% de cod este compatibil atât cu Android, cât și cu iOS, este nevoie să creezi doar o aplicație, și în final, sunt

create două aplicații. Acest lucru te ajută să economisești o grămadă de timp. În plus, mențenanța și actualizările sunt efectuate în același timp pentru ambele aplicații.

- **Performanță** - aplicațiile React Native funcționează aproape exact ca o aplicație nativă. Deoarece limbajul de programare este optimizat pentru dispozitivele mobile, aceste aplicații sunt destul de performante. Pe lângă asta, React Native nu utilizează doar CPU, dar profită și de GPU.
- **Bazat pe componente** - deoarece este creat pe ReactJS, conține toate caracteristicile lui, care permite să creezi o interfață utilizator complexă bazată pe componente.

Axios

Axios este client HTTP pentru JavaScript, capabil să facă request-uri din browser și NodeJS. Printre funcționalități se numără:

- Interceptarea request-urilor și response-urilor
- Crearea de request-uri HTTP și XMLHttpRequests
- Suport pentru Promise API
- Transformare automată pentru datele JSON

React Navigation

Deoarece React Native nu are un API pentru navigare, aşa cum ne permite un browser web, avem nevoie de o librărie. React navigation ne ajută să creăm structura de navigare a aplicației. Pe lângă asta, această librărie ne permite să navigăm între mai multe ecrane împreună cu gesturile și animațiile iOS și Android.

Async Storage

Async Storage este o librărie asincronă, necriptată, care ne permite să păstrăm datele offline ale aplicației. Persistența datelor se realizează într-un sistem de stocare cheie-valoare. Această aplicație ne ajută să salvăm unele date chiar după ce restartăm aplicație, cum ar fi datele utilizatorului logat.

3.3.3 Interfața utilizator

În figura 3.4 este ecranul de pornire a aplicației. Pe această pagină utilizatorul trebuie să aleagă cum se va autentifica în aplicație. Primul buton te duce pe alt ecran, pe care îl putem vizualiza în figura 3.5, pe acel ecran utilizatorul trebuie să introducă datele pentru a se autentifica. Al doilea buton este logarea pentru guest, adică unde nu ai nevoie de un cont, doar că acești utilizatori nu vor avea toate facilitățile aplicației, de exemplu să adauge rețetele favorite într-o listă, pentru a le avea salvate în telefon. Si cel de-al treilea buton ne duce pe pagina din figura 3.6, pe acest ecran, utilizatorul trebuie să introducă datele pentru a se înregistra în aplicație, după înregistrare, el se poate autentifica în aplicație.

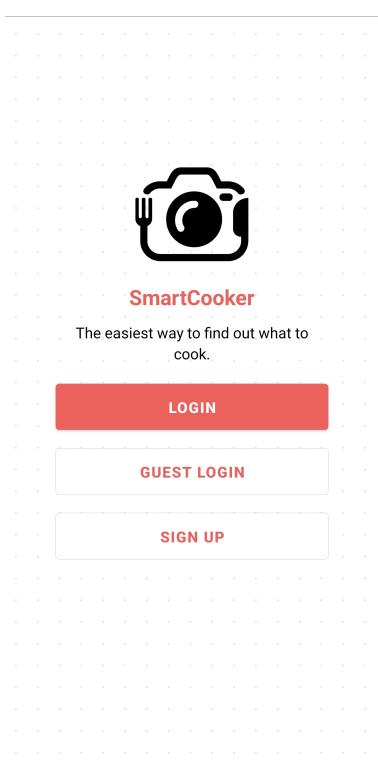


Figura 3.4: Ecranul de start

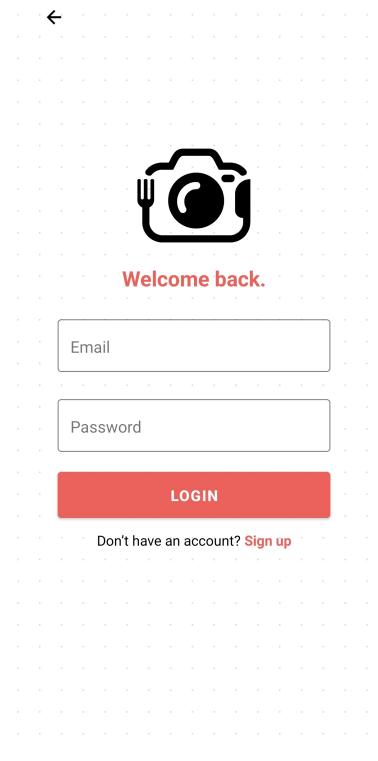


Figura 3.5: Ecranul de logare

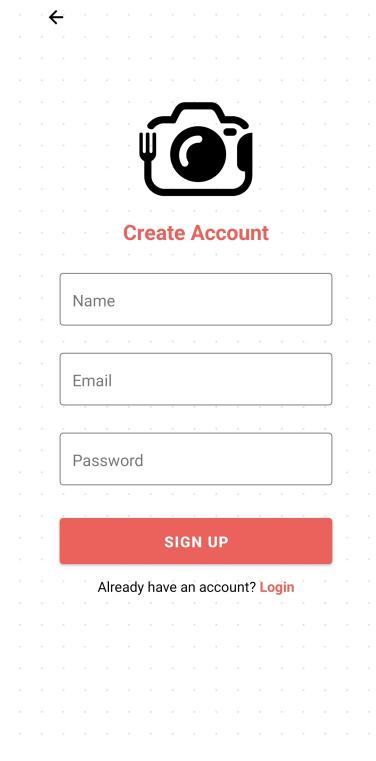


Figura 3.6: Ecranul de înregistrare

În figura 3.7 este reprezentat primul ecran pe care îl întâlnește un utilizator după logare, aici în mod aleatoriu se aleg 20 de rețete din baza de date și se afișează. Fiecare rețetă conține un titlu, descriere și două butoane, înima este pentru a adăuga această rețetă în lista de favorite (utilizatorii guest nu au acest buton), al doilea buton este pentru a deschide rețeta într-o pagină nouă (figura 3.8).

Pe acestă pagină se afișează toate datele despre rețete:

- titlul - numele rețetei
- numărul de calorii, grăsimi, etc.
- toate ingredientele ale rețetei
- instrucțiunile după care poți să gătești

În figura 3.9 este reprezentat ecranul de încărcare a imaginilor, în partea de jos sunt 4 butoane:

- take - deschide camera telefonului pentru a face o poză
- select - deschide galeria telefonului mobil, pentru a alege o imagine
- clear - șterge toate imaginile alese
- search - se deschide pagina cu rețetele recomandate, bazat pe ingredientele introduse

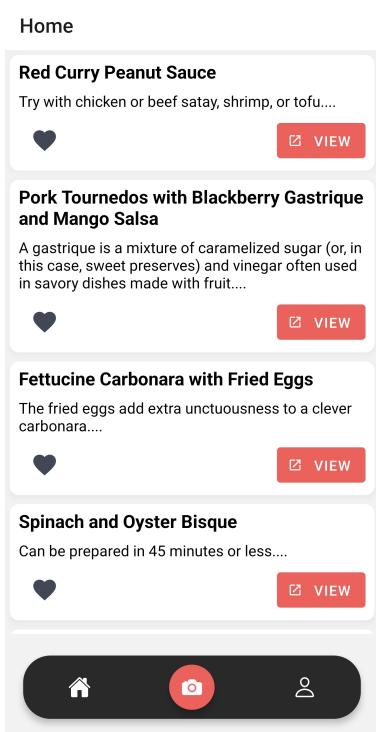


Figura 3.7: Ecranul de start după logare

Figura 3.8: Pagina unei rețete

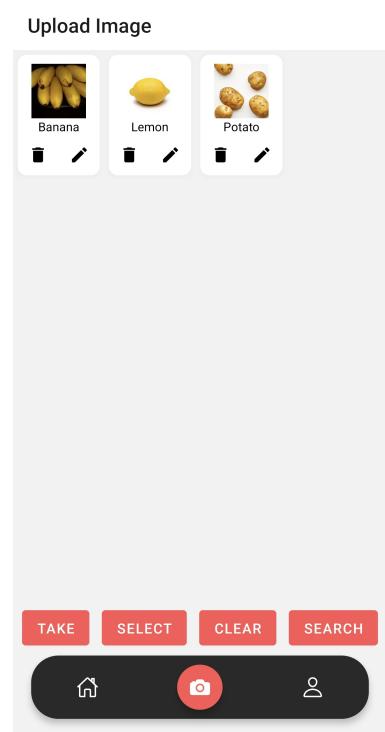


Figura 3.9: Pagina de încărcare a ingredientelor

În partea de sus a ecranului sunt aranjate în rând imaginile încărcate de utilizator, care deja au un label primit de la server, în urma procesului de detectare a ingredientului. Fiecare imagine are două butoane: de stergere și de editare. În caz că rețeaua neuronală greșește, utilizatorul poate modifica label-ul.

3.4 Backend

3.4.1 Arhitectura

Backend-ul constă dintr-un REST API, care comunică cu frontend-ul prin intermediul protocolului HTTP. Acesta este folosit pentru a interacționa cu baza de date MongoDB și de a aplica business logic pentru a oferi clientului datele pe care le-a cerut. De asemenea, serverul este securizat de un strat, care nu permite utilizatorului să acceseze datele fără permisiune. Serverul poate fi împărțit în trei stări prin care trece un request.

Primul strat pe care îl întâlnește un request către server, este stratul de securitate. Acesta se asigură că utilizatorul are dreptul să ceară anumite date, sau să efectueze o anumită operațiune. Aceste verificări sunt realizate cu ajutorul middleware-urilor, care pot să verifice dacă token-ul furnizat de request, este valid sau dacă nu a expirat. Prin acest token se verifică dacă utilizatorul are rolul necesar pentru a efectua această operațiune, dacă token-ul invalid sau expirat, serverul trimite ca response statusul 401 (Unauthorized) sau 403 (Forbidden).

Dacă clientul are dreptul de a accesa ruta, atunci request-ul ajunge în controller-ul respectiv. Acest strat se numește business logic, aici este accesată și baza de date dacă

este necesar, astfel accesând și cel de al treilea strat, de date. Acest strat constă din modele. Odată ce request-ul este prelucrat, datele adunate sunt trimise înapoi către client ca răspuns.

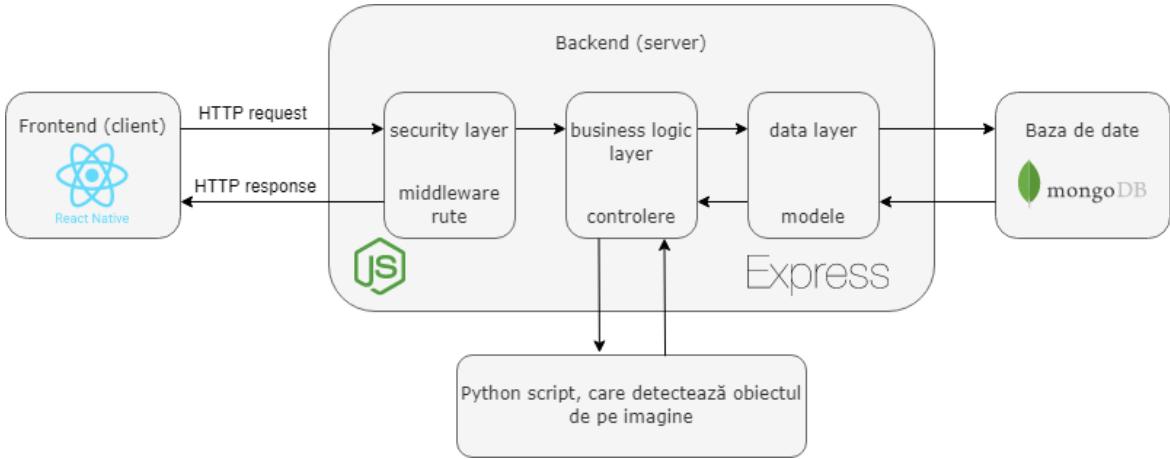


Figura 3.10: Arhitectura backend-ului

3.4.2 Tehnologii utilizate

NodeJS

NodeJS [5] este un mediu de execuție JavaScript cross-platform, care permite să ruleze cod JavaScript în afara unui browser web. NodeJS le permite dezvoltatorilor să creeze script-uri cu ajutorul limbajului JavaScript, care vor rula pe server pentru a produce conținut dinamic al paginii web înainte ca utilizatorul să vadă pagina.

ExpressJS

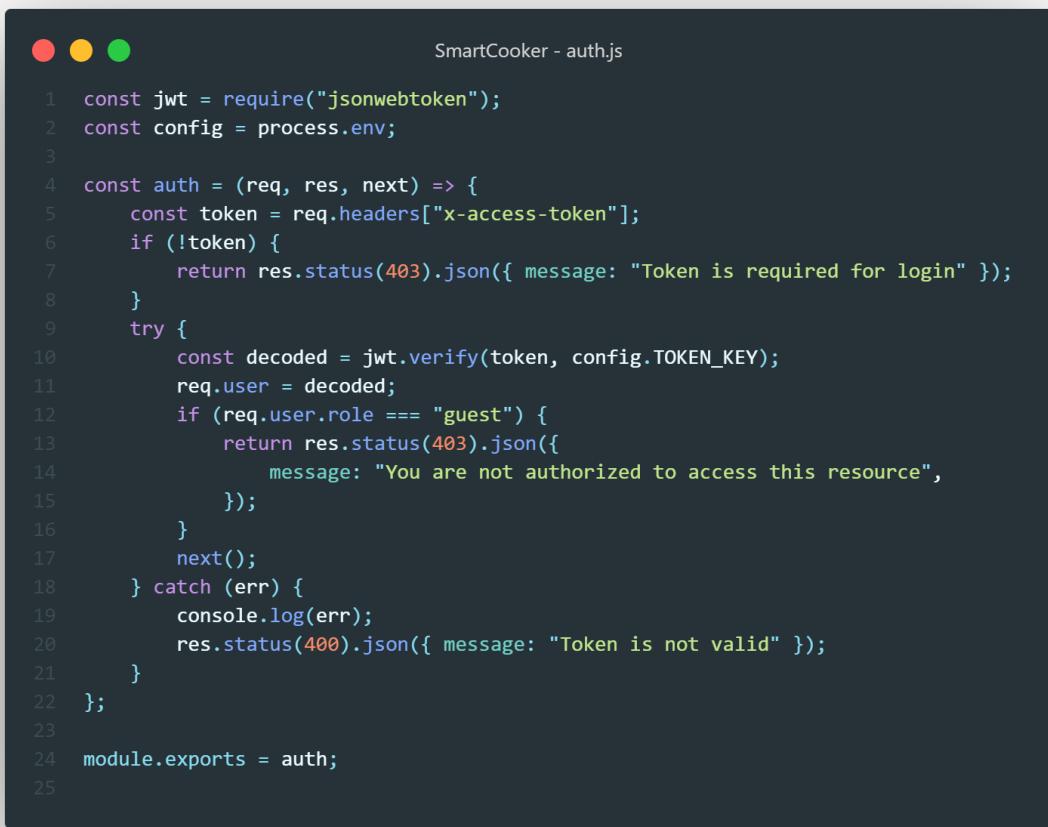
ExpressJS [6], este un framework web pentru NodeJS. Este conceput pentru construirea de aplicații web și API-uri. Dispune de o mulțime de metode HTTP, și middleware-uri care ne ajută să creăm un API eficient și securizat.

Rutele transmit request-urile și informația atașată către funcțiile controller potrivite. În figura 3.12 putem vedea fragmentul de cod de creare a rutelor pentru autentificare.

Controllerele sunt funcțiile care sunt apelate în dependență de ruta accesată de către utilizator. Acestea au ca parametri două obiecte:

1. **req** - este request-ul HTTP, care poate să conțină mai multe tipuri de date, cum ar fi: body, query, parametrii și headerele HTTP
2. **res** - este response-ul HTTP pe care funcția apelată îl returnează utilizatorului.

Middleware-urile sunt funcții care sunt executate în primul rând când vine un request către server, pentru a realiza anumite verificări. Dacă toate verificările au trecut cu succes, request-ul este transmis spre controller-ul potrivit. Middleware-urile pot modifica obiectele din request/response, și pot apela următorul middleware din stack. În fragmentul de cod 3.11, putem vedea middleware-ul care verifică dacă utilizatorul nu are un token invalid sau expirat.



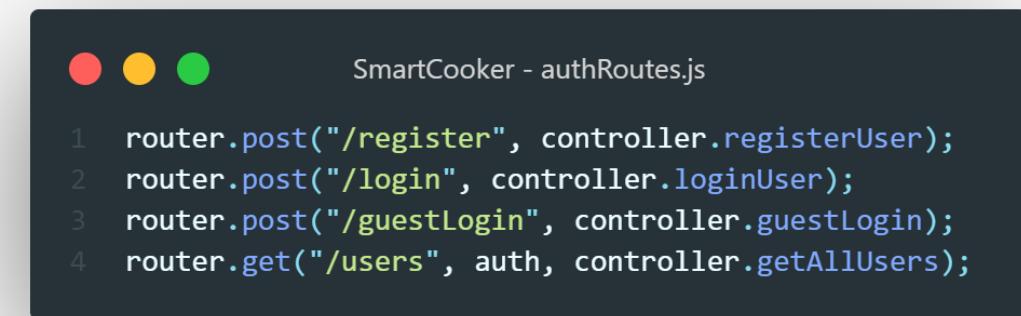
```

SmartCooker - auth.js

1  const jwt = require("jsonwebtoken");
2  const config = process.env;
3
4  const auth = (req, res, next) => {
5      const token = req.headers["x-access-token"];
6      if (!token) {
7          return res.status(403).json({ message: "Token is required for login" });
8      }
9      try {
10          const decoded = jwt.verify(token, config.TOKEN_KEY);
11          req.user = decoded;
12          if (req.user.role === "guest") {
13              return res.status(403).json({
14                  message: "You are not authorized to access this resource",
15              });
16          }
17          next();
18      } catch (err) {
19          console.log(err);
20          res.status(400).json({ message: "Token is not valid" });
21      }
22  };
23
24  module.exports = auth;
25

```

Figura 3.11: Middleware folosit pentru a verifica token-ul de acces



```

SmartCooker - authRoutes.js

1 router.post("/register", controller.registerUser);
2 router.post("/login", controller.loginUser);
3 router.post("/guestLogin", controller.guestLogin);
4 router.get("/users", auth, controller.getAllUsers);

```

Figura 3.12: Rutele folosite pentru autentificare

3.4.3 Structura

Structura backend-ului este împărțită în următoarele foldere:

- index.js, este fișierul entry-point a serverului. În acest fișier, serverul este initializat, se setează url-urile pentru rutele existente, se atribuie un port serverului.

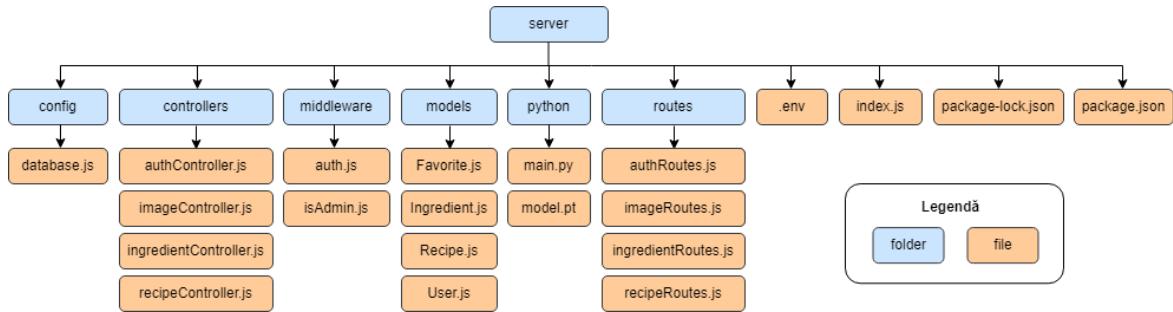


Figura 3.13: Structura părții de backend

- Fișierele package.json și package-lock.json, dețin metadatele relevante pentru proiect și sunt folosite pentru gestionarea dependențelor proiectului, scripturilor de rulare, versiuni și multe altele.
- .gitignore, este fișierul care îi spune lui git ce foldere/fișiere să ignore atunci când încărcați proiectul pe GitHub.
- .env, acest fișier conține datele sensibile, cum ar fi token-uri, chei pentru API-uri, și altele.
- config, acest folder conține fișierul de configurare a bazei de date.
- controllers, acest folder conține toate controller-ele serverului.
- middleware, acest folder conține toate middleware-urile aplicației.
- models, acest folder conține toate modelele ale bazei de date.
- python, acest folder conține script-ul python care detectează ingredientul din imagine, și modelul pre-antrenat a rețelei neuronale convoluționale.
- routes, acest folder conține toate rutele REST API ale aplicației.

3.5 Baza de date

Pentru această aplicație am folosit o bază de date bazată pe documente (NoSQL).

3.5.1 Baza de date bazată pe documente

O bază de date orientată document se bazează pe stocarea datelor pe baza documentelor. Acest tip de bază de date sunt considerate non-relaționale sau NoSQL. Sunt utilizate foarte mult pentru aplicațiile web și big data.

Aceste baze de date oferă următoarele avantaje:

- sunt extrem de flexibile, deoarece pot stoca și combina orice tip de date, structurate și nestructurate, pe când bazele de date relaționale pot stoca datele doar într-un mod structurat
- permit să actualizați dinamic schema fără a provoca întrerupere în funcționarea aplicației

3.5.2 MongoDB

MongoDB [4] este un sistem de gestionare a bazelor de date care funcționează cu un model de date bazat pe documente. Spre deosebire de SGBD-urile relaționale, MongoDB nu necesită tabele, scheme sau un limbaj de interogare separat. Informațiile sunt stocate sub formă de documente sau colecții.

Dezvoltatorii poziționează produsul ca o legătură intermediară între DBMS clasic și NoSQL. MongoDB nu folosește scheme precum bazele de date relaționale, ceea ce îmbunătățește performanța generală a sistemului.

3.5.3 MongoDB Atlas

În acest proiect am folosit MongoDB Atlas, care este versiunea de MongoDB din cloud, oferită sub formă de serviciu. Aceasta versiune oferă un set de avantaje:

- Baza de date este hostată în cloud, ceea ce elimină orice nevoie de a gestiona propriile servicii.
- Datele sunt stocate pe mai multe servere, ceea ce oferă uptime, chiar dacă nu funcționează un server.
- Actualizările bazei de date se realizează automat, ceea ce înseamnă ca primim noile features instant.

3.5.4 Indecși

MongoDB oferă mai multe tipuri de indecși, care ne ajută la execuția eficientă a interogărilor. Indecșii care mi-au fost utili în dezvoltarea aplicației mele sunt indecșii unici. Aceștia asigură faptul că, câmpurile indexate nu stochează date duplicate. MongoDB creează în mod implicit un index de unicitate pentru câmpul `_id` pentru fiecare colecție. Am folosit acest index pentru colecția de utilizatori, deoarece fiecare trebuie să aibă un email unic.

Capitolul 4

Detalii de implementare

4.1 Detectarea ingredientelor

Pentru a detecta ingredientele avem nevoie de un model pre-antrenat pe categoriile de care avem nevoi noi. Odată ce am antrenat o rețea neuronală, am exportat modelul antrenat, pe care îl vom folosi ulterior pentru a detecta imaginile încărcate de utilizatori

Utilizatorul când încarcă o imagine, ea este transformată în base64 și este transmisă printr-o rută către server. Aici m-am întâmpinat cu o problemă, serverul este în JavaScript, dar script-ul care detectează imaginea folosind modelul pre-antrenat este în limbajul Python. Inițial am încercat să rescriu scriptul în JavaScript, doar că nu am găsit librăriile necesare pentru a folosi un model pre-antrenat Torchvision pentru NodeJS.

Pentru rezolvarea acestei probleme, am folosit funcția spawn, care creează un proces copil, căruia îi putem transmite argumente, adică să-i transmitem imaginea în base64. Odată ce imaginea ajunge la script, el rulează și returnează un număr, care reprezintă id-ul clasei (tipului ingredientului), din baza de date, de unde luăm și numele ingredientului, care este transmis înapoi spre client, și se afișează.

În figura 4.1, putem observa fragmentul de cod care reprezintă modelul a unui ingredient în baza de date. Acest model are mai multe câmpuri:

- model_id - reprezintă id-ul clasei ingredientului din modelul antrenat
- name - reprezintă numele ingredientului

În figura 4.2 este script-ul care primește de la server imaginea în base64, și returnează id-ul clasei detectate.



SmartCooker - Ingredient.js

```

1 const mongoose = require("mongoose");
2
3 var IngredientSchema = new mongoose.Schema({
4     model_id: { type: String, required: true },
5     name: { type: String, required: true, minlength: 2 },
6     display_name: { type: String, required: true, minlength: 2 },
7 });
8
9 module.exports = mongoose.model("Ingredient", IngredientSchema);

```

Figura 4.1: Modelul de ingrediente



SmartCooker - main.py

```

1 checkpoint = torch.load("python/model.pt")
2 model = googlenet(pretrained=True)
3 num_ftrs = model.fc.in_features
4 model.fc = nn.Linear(num_ftrs, 10)
5 model.load_state_dict(checkpoint)
6 model.eval()
7
8 def predict_image(image_path):
9     transformation = transforms.Compose([
10         transforms.Resize((224, 224)),
11         transforms.ToTensor(),
12         transforms.Normalize([0.6941, 0.5490, 0.4671],
13                             [0.2903, 0.3704, 0.4086])
14     ])
15     image_tensor = transformation(image_path)
16     image_tensor = image_tensor.unsqueeze_(0)
17     image_tensor.cpu()
18
19     input = Variable(image_tensor)
20     output = model(input)
21
22     index = output.data.numpy().argmax()
23     return index
24
25 im_b64 = sys.argv[1]
26 im_b64 = base64.b64decode(im_b64)
27 im_b64 = BytesIO(im_b64)
28 im_b64 = Image.open(im_b64)
29 image = im_b64.convert('RGB')
30 prediction = predict_image(image)
31 print(prediction)
32 sys.stdout.flush()

```

Figura 4.2: Script-ul Python care detectează ingrediente

4.2 Recomandarea rețetelor

Pentru a recomanda rețete utilizatorilor atunci când ei încarcă imagini cu ingrediente, avem nevoie de un set de date cu multe rețete. Am folosit Epicurious [2], un set de date cu 20000 de rețete în formă de obiecte JSON. Fiecare rețetă este formata din mai multe chei-valori, cum ar fi:

- title, care este numele rețetei.
- directions, care este o listă de instrucțiuni despre cum trebuie preparată această bucătă
- ingredients, este lista de ingrediente de care ai nevoie pentru a prepara această mâncare

Ingredientele din aceste rețete conțin o mulțime de informații redundante, care ne vor crea greutăți pentru algoritmul de căutare, de exemplu greutățile ingredientelor. Deci, ar trebui să scăpăm de aceste informații, și să creăm lista de ingrediente pentru fiecare rețetă în parte.

Inițial, am găsit pe Wikipedia o sursă, care conține o lista de valori standard de gătit, cum ar fi lingurițe, grame, etc. Aceste cuvinte trebuie eliminate din ingrediente, pe lângă ele am eliminat cele mai comune cuvinte din limbă, numite „cuvintele stop”, aceasta am realizat cu ajutorul librării NLTK.

Există și alte cuvinte în ingrediente care sunt inutile, de exemplu ingredientele comune pentru rețete, cum ar fi apă, sare, uleiul. Majoritatea oamenilor au ulei în casă, de aia ar deveni enervant să introduci de fiecare dată în aplicație aceste date. Pe lângă asta, am scăpat de punctuație, accente, am transformat totul în minuscule și am scăpat de caracterele unicode.

În fragmentul de cod din figura 4.5, putem observa funcția ingredient_parser, care procesează toate ingredientele noastre, și returnează o listă de ingrediente care poate fi folosită pentru recomandare.

Lista cu ingredientele procesate este adăugată la fiecare rețetă în documentul JSON.

Pentru a încărca toate rețetele în baza de date, am creat un model pe partea serverului, și un script Python care deschide fișierul JSON, și încarcă pe rând fiecare obiect în baza de date. Modelul poate fi văzut în figura 4.4. Script-ul poate fi văzut în figura 4.3



SmartCooker - main.py

```

1 import json
2 from pymongo import MongoClient, InsertOne
3
4 client = MongoClient("url")
5 db = client.smartcooker
6 collection = db.recipes
7 requesting = []
8
9 with open("file.json", "r") as f:
10     data = json.load(f)
11     for jsonObj in data:
12         requesting.append(InsertOne(jsonObj))
13
14 result = collection.bulk_write(requesting)
15 client.close()
16

```

Figura 4.3: Script-ul Python care importă toate rețetele în baza de date



SmartCooker - Recipe.js

```

1 const mongoose = require("mongoose");
2
3 var RecipeSchema = new mongoose.Schema({
4     title: { type: String, required: true, minlength: 2 },
5     directions: [{ type: String, required: true, minlength: 2 }],
6     ingredients: [{ type: String, required: true, minlength: 2 }],
7     ingredients_clean: [{ type: String, required: true, minlength: 2 }],
8     categories: [{ type: String, required: true, minlength: 2 }],
9     desc: { type: String, required: true, minlength: 2 },
10    fat: { type: Number, required: true },
11    calories: { type: Number, required: true },
12    protein: { type: Number, required: true },
13    sodium: { type: Number, required: true },
14    rating: { type: Number, required: true },
15 });
16
17 module.exports = mongoose.model("Recipe", RecipeSchema);
18

```

Figura 4.4: Modelul rețetei



SmartCooker - ingredients_parser.py

```
1 def ingredient_parser(ingredients):
2     measures = ['teaspoon', 't', 'tsp.', ... 'kilo']
3     words_to_remove = ['oil', 'a', ... 'breast']
4     ingr = ['apple', ... 'banana']
5
6     translator = str.maketrans('', '', string.punctuation)
7     lemmatizer = WordNetLemmatizer()
8     ingred_list = []
9     for i in ingredients:
10         i.translate(translator)
11         i = i.replace(',', '')
12         items = re.split(' ', i)
13         items = [word for word in items if word.isalpha()]
14         items = [word.lower() for word in items]
15         items = [unidecode.unidecode(word) for word in
16                  items]
17         items = [lemmatizer.lemmatize(word) for word in items]
18         items = [word for word in items if word not in measures]
19         items = [word for word in items if word not in words_to_remove]
20         if items:
21             clean = ' '.join(items)
22             for t in ingr:
23                 if t in clean:
24                     clean = "".join(
25                         ["" if not ele == t else ele for ele in clean.split()])
26             if len(clean) > 0:
27                 ingred_list.append(clean)
28     ingred_list = list(dict.fromkeys(ingred_list))
29     return ingred_list
30
```

Figura 4.5: Funcția care procesează ingrediente

Algoritmul de recomandare a rețetelor este destul de simplu, atunci când utilizatorul încarcă ingredientele, baza de date returnează toate rețetele care conțin cel puțin un ingredient din lista utilizatorului. După asta, rețetele sunt sortate după o variabilă care se calculează folosind numărul total de ingrediente din rețetă, și numărul de ingrediente care coincid cu lista din rețetă. Dacă sunt două ingrediente cu aceeași valoare, atunci se sortează după numărul de ingrediente care coincide cu lista rețetei. În figura 4.6 observăm fragmentul de cod a funcției de sortare.

```
SmartCooker - recipeController.js

1  function sortRecipes(recipes, ingredients) {
2      const results = recipes
3          .map((recipe) => {
4              let matching_count = 0;
5              let matching_ratio = 0;
6              for (let i = 0; i < ingredients.length; i++) {
7                  if (recipe.ingredients_clean.includes(ingredients[i])) {
8                      matching_count++;
9                  }
10                 matching_ratio =
11                     matching_count / recipe.ingredients_clean.length;
12             }
13             return {
14                 ...recipe._doc,
15                 matching_count,
16                 matching_ratio,
17             };
18         })
19         .sort((a, b) => {
20             let ratioDiff = b.matching_ratio - a.matching_ratio;
21             if (ratioDiff === 0) {
22                 return b.matching_count - a.matching_count;
23             }
24             return ratioDiff;
25         });
26         return results;
27     }
}
```

Figura 4.6: Funcția de sortare a ingredientelor

Capitolul 5

Concluzii și directii viitoare de dezvoltare

5.1 Sumar al rezultatelor

Am dezvoltat o aplicație mobilă, creată cu ajutorul stack-ului MERN, care permite utilizatorilor să încarce imagini cu produsele pe care le au în frigider, și să primească o listă de rețete recomandate de sistem, pe care le pot pregăti folosind ingredientele introduse. Printre cele mai importante rezultate obținute în urma dezvoltării acestui proiect se numără:

- Crearea și învățarea rețelei neuronale convoluțională
- Dezvoltarea arhitecturii aplicației (frontend și backend)
- Dezvoltarea aplicației pentru sistemul de operare Android

5.2 Discutarea rezultatelor obținute

Cerințele de bază pe care le-am gândit la începutul proiectului au fost implementate. Pentru a măsura precizia de detectare a ingredientelor, am rulat un test format din 3427 imagini și am primit următoarele rezultate.

ID	Ingredient	Precizie
0	Apple	98.33%
1	Avocado	96.12%
2	Banana	98.19%
3	Eggplant	77.56%
4	Lemon	64.63%
5	Onion	88.51%
6	Orange	100.00%
7	Pepper	100.00%
8	Potato	75.33%
9	Tomato	96.14%

Tabela 5.1: precizia pentru fiecare ingredient

În tabela 5.1, putem observa precizia pentru fiecare ingredient în parte. Din 3427 de imagini, modelul a detectat corect 3181, adică precizia totală este 92.82%. După părerea mea, am primit un rezultat foarte bun.

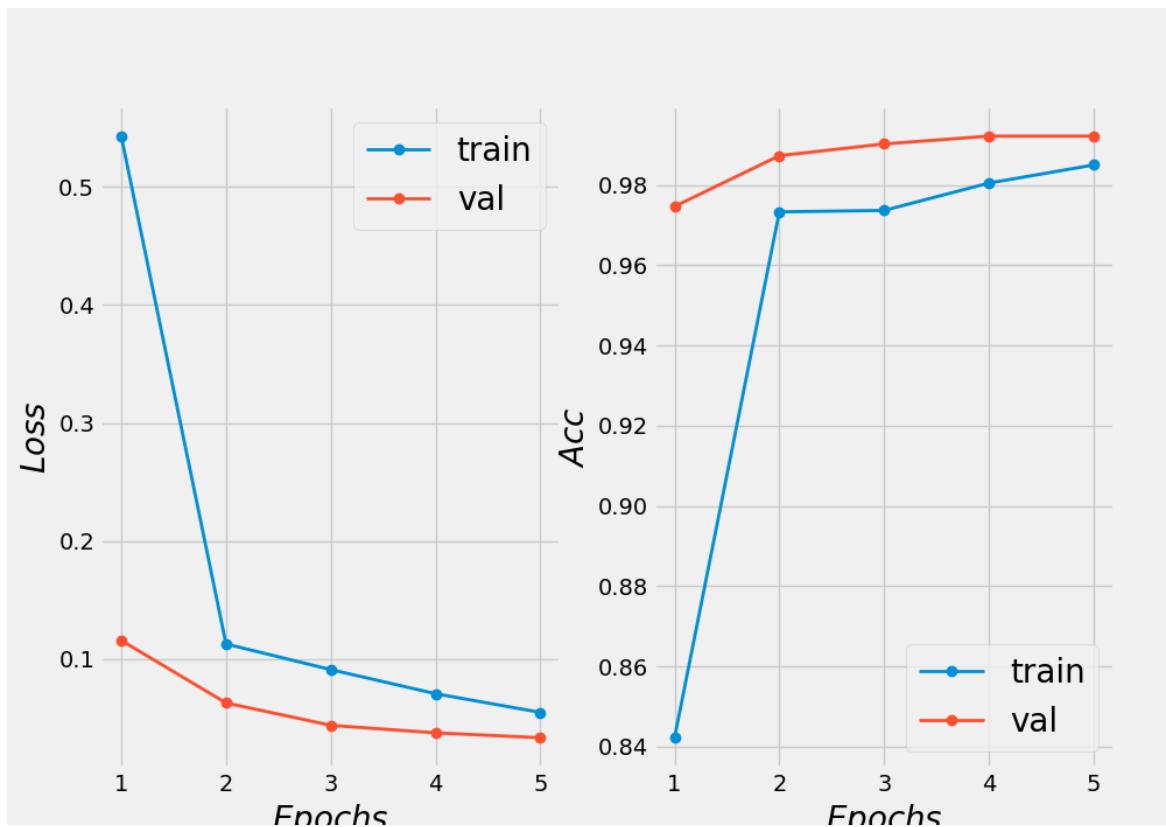


Figura 5.1: Precizia și eroarea după învățare

În figura 5.1 este o diagramă care ne arată precizia și eroarea după ce am antrenat modelul de 5 ori, observăm că datele sunt mai bune proporțional cu numărul de încercări.

5.3 Directii viitoare

M-am gândit la următoarele direcții viitoare pentru aplicația mea:

- Dezvoltarea aplicației și pentru alte sisteme de operare, de exemplu pentru iOS
- Îmbunătățirea algoritmilor de căutare a rețetelor după ingredientele introduse
- Învățarea rețelei neuronale să recunoască mult mai multe ingrediente

Bibliografie

- [1] https://www.researchgate.net/publication/271659249_FoodCam_A_real-time_food_recognition_system_on_a_smartphone
- [2] Epicurious - Recipes with Rating and Nutrition <https://www.kaggle.com/datasets/hugodarwood/epirecipes>
- [3] A dataset with 90380 images of 131 fruits and vegetables <https://www.kaggle.com/datasets/moltean/fruits>
- [4] MongoDB Documentation <https://www.mongodb.com/docs/>
- [5] NodeJS Documentation <https://nodejs.org/en/docs/>
- [6] ExpressJS Guide <https://expressjs.com/en/guide/routing.html>
- [7] PyTorch Documentation <https://pytorch.org/docs/stable/index.html>
- [8] Torchvision Documentation <https://pytorch.org/vision/stable/index.html>
- [9] GoogLeNet convolutional neural network <https://www.mathworks.com/help/deeplearning/ref/googlenet.html>