UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Dezvoltarea aplicațiilor pentru dispozitivele mobile folosind React Native. Un prototip de aplicație pentru iOS și Android

propusă de

Mihaela Radu

Sesiunea: *iulie, 2019*

Coordonator științific

Conf. Dr. Cristian Gațu

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Dezvoltarea aplicațiilor pentru dispozitivele mobile folosind React Native. Un prototip de aplicație pentru iOS și Android

Mihaela Radu

Sesiunea: *iulie, 2019*

Coordonator științific

Conf. Dr. Cristian Gațu

Avizat,

Îndrumător Lucrare

Titlul, Numele şi prenumele _____

Data _____ Semnătura _____

DECLARAŢIE
privind originalitatea conţinutului lucrării de licenţă/disertaţie

Subsemntatul(a) ……

domiciliul în ……..

născut(ă) la data de ……..…., identificat prin CNP ……..……..……..., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de ……. specializarea ……, promoţia ……,

declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul:

_____

_____

elaborată sub îndrumarea _____, pe care urmează să o susţin în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă/disertaţie să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării fasificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „Dezvoltarea aplicaţiilor pentru dispozitivele mobile folosind React Native. Un prototip de aplicaţie pentru iOS şi Android”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iaşi, să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *data*

Absolvent *Mihaela Radu*

_____

(semnătura în original)

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, *Mihaela Radu.*

Încheierea acestui acord este necesară din următoarele motive:

Iași, *data*

Decan *Adrian Iftene*

_____

(semnătura în original)

Absolvent *Mihaela Radu*

_____

(semnătura în original)

Cuprins

Introducere

Motivație

Această lucrare de licență are ca scop oferirea unei aplicații ce vrea să vină în ajutorul persoanelor care doresc să învețe să gătească, să încerce rețete noi ori cunoscute. Totodată vine și în ajutorul persoanelor ce sunt lipsite de idei în materie de combinat ingrediente sau care pur și simplu caută o metodă convenabilă pentru a găti în lipsă de timp.

Gradul de noutate

În momentul de față pe piața actuală nu există o diversitate foarte mare în materie de aplicții bazate pe recomandarea de rețete ce au la bază recunoașterea imaginii. Ceea ce este accesibil în momentul de față permite doar căutarea manuală a rețetelor pe baza unor cuvinte, categorii, bucătării specifice anumitor țări etc.. Pe baza recunoașterii imaginii există posibilitatea doar de a ține evidența caloriilor și a unui jurnal a ceea ce mâncă utilizatorul.

Prin prezenta lucrare se dorește îmbinarea cautării anumitor rețete pe baza ingredientelor cu recunoașterea imaginilor, integrat și cu un sistem de recomandări bazat pe utilizările anterioare, dar și posibilitatea de a alcătui o listă de cumpărături cu ingredientele lipsă, pentru a crea o aplicție mult mai facilă și fezabilă pentru timpurile pe care le trăim și în care timpul este un factor important.

Metodologia folosită

Din dorința de a avea o modalitate mai ușoară și la îndemână în orice moment, de a găsi rețete destul de accesibile ca și timp, dar ca și pret s-a născut ideea acestei aplicații. Având în vedere că în acest moment smartphone-urile sunt la îndemâna oricui și sunt printre cele mai folosite dispozitive de pe piață s-a ajuns la concluzia că ar fi cea mai bună opțiune realizarea unei aplicații pentru mobil[1].

Problema cu care se confruntă majoritatea persoanelor de vârstă mijlocie este în primul rând cea a timpului. Din dorința de a se menține un mod de viață sănătos se observă tendința de a găti din ce în ce mai mult acasă. Iar pentru a-și îndepli scopul se observă o disponibilitate spre a încerca lucruri noi care fac viața mai ușoară.

[1] Conform studiului anual al adoptarii tehnologiei Forrester Research

Metodologia folosită în dezvoltarea aplicației constă în îmbinarea mai multor tehnologii moderne și anume ReactNative - ce oferă dezvoltarea mai rapidă, capacitatea de reutilizare a componentelor și ușurința de a fi întreținute; Expo SDK împreună cu Camera Expo ce permite accesul la funcționalitatea sistemului (lucruri cum ar fi camera foto, notificări push, contacte etc.; Clarifai împreună cu Spoonacular ce ajută la identificarea ingredientelor cel din urmă oferind sugestii de rețete pe baza lor.

Obiectivele generale

Obiectivele generale ale aplicației sunt de a îmbina o serie de tehnologii de actualitate ce vin în ajutorul rezolvării problemei întâmpinate.

În realizarea proiectuli se vor pune în evidență avantajele oferite de recunoașterea imaginilor precum: dezvoltarea de aplicații inovatoare și interesante, adăugarea de valoare plus - în special pentru aplicațiile mobile - făcând lumea fizică un loc de joacă interactiv, implicarea utilizatorului. Recunoașterea imaginii oferă aplicațiilor oportunități uriașe de a captiva, deoarece tehnologia permite extinderea dincolo de limitele dispozitivului mobil și în lumea fizică a utilizatorului. Aplicațiile pot oferi ceva mai tangibil, ceea ce permite crearea unei conexiuni mai puternice și mai emotive cu utilizatorii.

Totodată prin sistemul de recomandare bazat pe experiența anterioară a utilizatorului în cadrul aplicației putem furniza sugestii de produse adecvate. Consumatorii ajung să folosească mai mult aplicația atunci când sunt făcute recomandări individuale. Aceștia ajung să petreacă mai mult timp parcurgând recomandare după recomnadare fără a fi nevoie să efectueze o nouă căutare.

Toate acestea sunt realizate cu ajutorul framework-ului React Native ce facilitează implementarea de aplicații mobile pentru mai multe platforme (ex. Android, IOS) asigurând viteza și agilitatea.

Descrierea soluției

Procesul prin care s-a implementat soluția ese următorul: cu ajutorul camerei telefonului prin intermediul librăriei Camera Expo, ce va transforma imaginea în format base64, putem realiza o fotografie a ingredientelor ce le avem la îndemână. Pe baza fotografiei vor fi afișate utilizatorului sugestii de rețete bazate pe ingredientele din imagine, identificate cu ajutorul serviciului de recunoaștere a imaginilor Clarifai și cu ajutorul API-ului Spoonacular ce ne pune la dispoziție o gamă largă de rețete.

Pe baza datelor furnizate în urma utilizărilor anterioare se vor putea face recomandări specifice utilizatorului, prin intermediul algoritmului de recomandare ce se folosește de grafuri ce analizează ratingul serviciilor sau produselor oferite de către un utilizator cu ratingul de la alți utilizatori sau a anterioarelor utilizări pentru a răspunde la întrebări similare cu predicții și sugestii. Mai multe detalii vor fi oferite în al treilea capitol al lucrării.

Tehnologii folosite

Tehnologiile folosite (limbaje de programare, biblioteci, framework-uri etc.) pentru crearea arhitecturii și a aplicției sunt:

- Pentru partea de **server** folosesc:
  - **NodeJS**
- Pentru partea de **client** folosesc:
  - **JavaSript**
  - **ReactJs**
  - **ReactNative**
  - **Spoonacular API**
  - **Camera Expo**
  - **SASS&CSS3**
  - **Clarifai API**
- Pentru partea de dezvoltare folosesc**:**
  - **Expo (SDK)**

Abordări anterioare a lucrării

În momentul actual pe piață există un număr de aplicații mobile sau web cu ajutorul cărora putem descoperi rețete cât mai diverse, bazate pe anumite cuvinte cheie, cum ar fi ingrediente, tipuri de bucătării, anumite diete.

Aplicațiile ce merită menționate sunt:

- *„SuperCook”* – Acesta este un site simplu și eficient și nu este nevoie descărcarea sau instalarea unei aplicații pe telefon. Se începe prin selectarea ingredientelor, pe care le avem deja, din mai multe categorii (cum ar fi carne, condimente și lactate). Pe măsură ce sunt adăugate ingredientele disponibile, SuperCook sugerează rețete, actualizând rezultatele pentru fiecare element nou ce este inclus. [1]
- *„Epicurious”* – Se pot căuta idei de rețete făcându-se căutarea după un anumit ingredient, sugestiile includ fotografii ale felurilor de mâncare finisate și videoclipuri instructive pentru a ajuta pe parcursul preparării. Se pot face căutări și după restricții dietetice, cum ar fi "fără gluten", "fără lapte", "keto prietenos" și așa mai departe.

- *„CalorieMama”* - Combinând tehnologia de învățare profundă și tehnologia de clasificare a imaginilor, această aplicație scanează conținutul din farfurie, indicând ingredientele și calculând numărul total de calorii - toate dintr-o singură fotografie. [2]

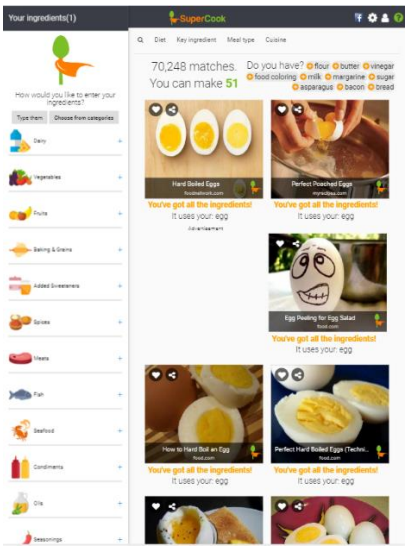


Figura 1. Interfața aplicației SuperCook

[Sursa:
https://www.supercook.com/#/recipes]

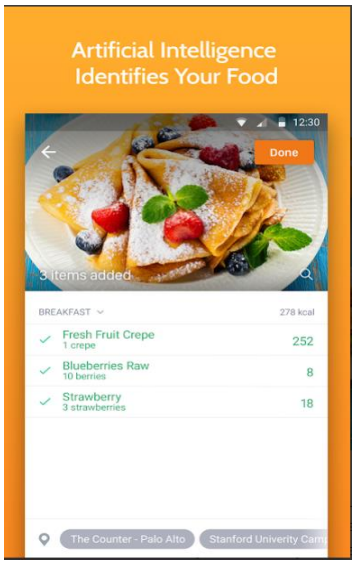


Figura 2. Interfața aplicației CalorieMamma

[Sursa:
https://play.google.com/store/apps/details?id=com.azumio.android.caloriesbuddy&hl=en]

Structura lucrării

În această secțiune este prezentat un conspect al lucrării cu punctele cheie ale fiecărui capitol și legătura dintre ele. Structura acestei documentații este formată din trei capitole.

În primul capitol vom face o scurtă descriere a tuturor tehnologiilor folosite (limbaje de programare, biblioteci, framework-uri etc.) pentru crearea arhitecturii și a aplicației. De asemenea vom prezenta și arhitectura folosită.

Urmând ca în capitolul doi să trecem la descrierea algoritmului de recomandare bazat pe grafuri împreună cu un scurt exemplu.

În capitolul trei vom descrie pe larg și succint modelele arhitecturale și de proiectare folosite în acest proiect cu câteva exemple ce demonstrează soluția proiectului, flow-urile principale plus alte concepte ce merită menționate, folosite în implementare.

Ultima parte a lucrării va conține concluziile finale asupra arhitecturii implementate, provocările la care ne putem aștepta pe viitor și îmbunătățirile care se pot aduce la această soluție.

Capitolul I

Descrierea tehnologiilor

Pe partea de server se folosește **Node.js** ce permite dezvoltarea de aplicații web fiind o platformă construită pe motorul V8 al Chrome și se bazează pe un model bazat pe evenimente, iar operațiile de input/output sunt neblocante bazându-se pe callback-uri.[3]

În cadrul părții de client al aplicației se folosesc o suită de tehnologii actuale pe care le găsim de asemenea și în dezvoltarea unor produse existente pe piață (ex. Instagram, Facebook, Skype, Medium, Pinterest, Tesla, Uber etc.) precum:

- **JavaScript** - este un limbaj de programare de nivel înalt, dinamic, netipizat, bazat pe obiecte, multi-paradigmă, interpretat și rapid cu ajutorul căruia se pot dezvolta multiple aplicații. Folosit pentru a crea pagini web interactive și pentru a furniza programe online, incluzând jocuri video;[4]
- **ReactJs** - bibliotecă JavaScript pentru construirea de interfețe. Este menținută de Facebook, Instagram și o comunitate de dezvoltatori și corporații individuale. *React* poate fi utilizat în dezvoltarea de aplicații de o singură pagină și de aplicații mobile. Obiectivul principal este acela de a oferi viteză, simplitate, modularitate și scalabilitate;[5]
- **SASS & CSS3** – SASS este un limbaj de scripting care este interpretat sau compilat în CSS. *CSS3* constă dintr-o serie de selectori și pseudo-selectori care grupează regulile care se aplică acestora. SassScript oferă următoarele mecanisme: variabile, nesting, mixins și moștenire bazată pe selectori.[6]

ReactNative

ReactNative[7] este un framework ce construiește o ierarhie a componentelor UI pentru a construi codul JavaScript. Dispune de un set de componente pentru platformele iOS și Android pentru a construi o aplicație mobilă cu aspect natural. ReactJS, pe de altă parte, este o bibliotecă open source JavaScript pentru a crea interfețe utilizator.

Cu toate acestea, ReactNative și ReactJS sunt dezvoltate de Facebook utilizând aceleași principii de proiectare, cu excepția interfețelor de proiectare. Deoarece utilizează același cod pentru crearea aplicațiilor ReactNative iOS sau a aplicațiilor ReacrNative Android și aplicațiilor web, trebuie doar să cunoașteți HTML, CSS și JavaScript.

Aplicațiile scrise cu ReactNative sunt compilate într-un cod scris nativ, ceea ce îi permite să lucreze nu numai pe ambele sisteme de operare, ci și să funcționeze în același mod pe ambele platforme fără niciun decalaj. Alte beneficii ale utilizării framework-ului includ dezvoltarea mai rapidă, capacitatea de reutilizare a componentelor și ușurința de a fi întreținute.

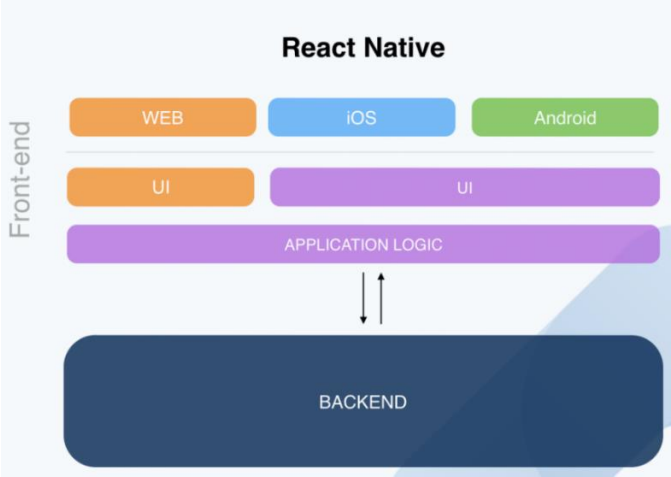


Figura 3. Arhitectura React Native

[**Sursa:** https://www.icapps.com/blog/5-key-advantages-react-native]

Recipe - Food – Nutrition API

Recipe - Food – Nutrition API [8] oferă un motor de căutare semantic pentru rețete folosind interogări în limbaj natural. Se pot calcula automat informațiile nutriționale pentru orice rețetă, analiza costurile rețetei, vizualiza lista de ingrediente, găsi rețete pentru ceea ce avem în frigider sau rețete bazate pe diete speciale, clasifica rețetele în anumite categorii, calcula cantitățile sau se poate crea un plan de masă integral.

Clarifai

Clarifai API [9] oferă recunoașterea imaginilor și a videoclipurilor ca serviciu. API-ul este construit în jurul unei idei simple: trimiterea unor mesaje de intrare (o imagine sau un videoclip) către serviciu și returnarea predicțiilor.

Tipul de predicție se bazează pe modelul care este utilizat. De exemplu, dacă utilizăm metoda de introducere a datelor prin intermediul unui model "alimentar", predicțiile pe care le returnează vor conține concepte pe care modelul "alimente" le cunoaște.

Figura 4. Modelul de predicție

[**Sursa:** https://www.clarifai.com/developer/guide]

Tehnologii folosite pentru partea de dezvoltare (development)

Expo

Expo SDK [10] este un set de biblioteci scrise nativ pentru fiecare platformă care permite accesul la funcționalitatea sistemului (lucruri cum ar fi camera foto, notificări push, contacte, stocare locală și alte aplicații hardware) din JavaScript. SDK este conceput pentru a elimina cât mai mult posibil diferențele între platforme, ceea ce face ca proiectele să fie portabile, deoarece pot funcționa în orice mediu nativ care conține SDK-ul Expo.

Expo furnizează, de asemenea, componente UI pentru a gestiona o varietate de cazuri de utilizare pe care aproape toate aplicațiile le vor conține, dar care nu sunt incluse în framework-ul React Native, de ex. iconițe, și multe altele.

Camera Expo

Camera Expo [10] este o componentă React care permite previzualizarea camerei frontale sau cea din spate a dispozitivului. Totodată se pot accesa parametrii camerei precum zoom, focalizare automată, balans de alb și modul bliț care sunt reglabile. Cu ajutorul camerei Expo, se pot face fotografii și se pot înregistra videoclipuri care sunt salvate în memoria cache a aplicației. Mai mult, componenta este, de asemenea, capabilă să detecteze fețele și codurile de bare care apar în previzualizare.

Arhitectura soluției

React este o bibliotecă JavaScript pentru construirea interfețelor de utilizator web. Utilizează un DOM(Document Object Model) virtual, o abstractizare a HTML

DOM. Eficiența React-ului rezidă în capacitatea sa de a actualiza și re-redacta ceea ce s-a schimbat în loc să redevină din nou întreaga pagină.

Structura aplicațiilor cu ReactNative [11] nu este diferită de structura aplicațiilor bazate pe React, prin urmare este recomandată următoarea structură de fișiere:

- Crearea fișierului „*app*” în root-ul proiectului
- Fișierul „*app*” trebuie să conțină următoarele fișiere:
    - *assets* – ce ar trebui să mai conțină următoarele 3 fișiere – fonts, icons, images
    - *components* - aici vor fi plasate toate componentele React partajate. De obicei, aceste componente sunt cele pe care le numim "dummy", care nu au logică și pot fi reutilizate cu ușurință în cadrul aplicației.
    - *views* - acestea sunt ecranele noastre de aplicații, cele pe care navigăm. Sunt de asemenea, componente React, dar sunt cele pe care le numim containere, pentru că ele conțin starea lor proprie.
    - *modules* - există bucăți de cod care nu au o parte de vizualizare corespunzătoare (JSX). Exemple tipice sunt modulul culori (conține toate culorile aplicației) și modulul utils (conține funcții utilitare care sunt reutilizate).
    - *services* - acestea sunt funcțiile care realizează apelurile către API-uri.

Pentru o vizualizare mai bună asupra modului în care ReactNative funcționează avem următoarea diagramă:

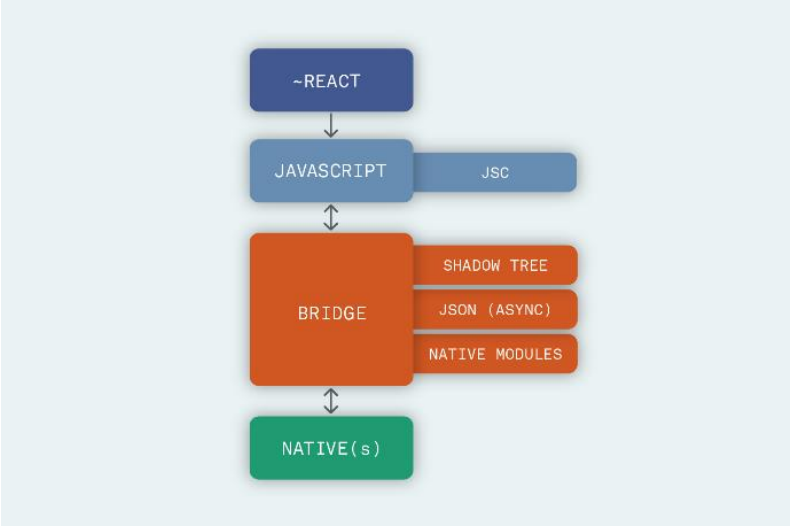


Figura 5. Interpretarea fișierelor în React Native

[**Sursa:** https://formidable.com/blog/2019/react-codegen-part-1]

După cum se poate observa, există patru secțiuni principale: codul React care este scris (care este foarte asemănător cu omologul său web), JavaScript care este interpretat din ceea ce scrieți, o serie de elemente colectiv cunoscute sub numele de "The Bridge" și partea nativă. Structura fișierelor din cadrul aplicției curente poate fi găsită la finalul documentului la secțiunea Anexe.

Concluzii

În cadrul acestui capitol am prezentat, pe scurt, aspecte legate de tehnologiile cu ajutorul cărora a fost implementat proiectul curent de licență împreună cu arhitectura lui.

În capitolul următor voi face o descriere detaliată a algoritmului de recomandare bazat pe grafuri împreună cu prezentarea unui exemplu în introducere.

Capitolul II

Algoritm de recomandare

Introducere

Sistemele recomandate au devenit unul dintre instrumentele importante din comerțul electronic. Acestea combină ratingul serviciilor sau produselor oferite de către un utilizator cu ratingul de la alți utilizatori pentru a răspunde la întrebări similare cu predicții și sugestii. Pentru a explica acest lucru în mod corespunzător, să luăm un exemplu. Luăm în considerare diferitele genuri ale muzicii. Acestea includ rock, jazz, metal, death metal, alternative, blues, clasic etc.

Acum, să presupunem că o anumită persoană Rareș îi place rock, metal și jazz. Prietenei lui, Sara, îi place rock, death metal și muzica clasică în timp ce celuilalt prieten Radu îi place rock, metal și alternative. Folosim tehnica pentru a vedea genurile comune de muzică între Rareș și prietenii săi. Vedem că atât Sarei, cât și lui Radu le place rock-ul împreună cu Rareș. Totuși, lui Radu îi place și metal, care este în comun cu gusturile lui Rareș, dar nu și cu ale Sarei care preferă death metal.

Deci, îl vom selecta pe Radu ca persoană a cărei genuri vor fi recomandate lui Rareș. Observăm că lui Radu îi place și alternative. Deoarece atât Rareș, cât și Radu au două genuri în comun, există o mare șansă ca și lui Rareș să îi placă alternative, dar probabil că nu este conștient de acesta. Deci, genul alternative este recomandat lui Rareș. Aici, merită menționat faptul că preferințele Sarei death metal și muzica clasică ar putea fi de asemenea recomandate, dar din moment ce Radu are două genuri în comun, există o probabilitate mai mare ca Rareș să prefere alternative decât death metal și muzica clasică.

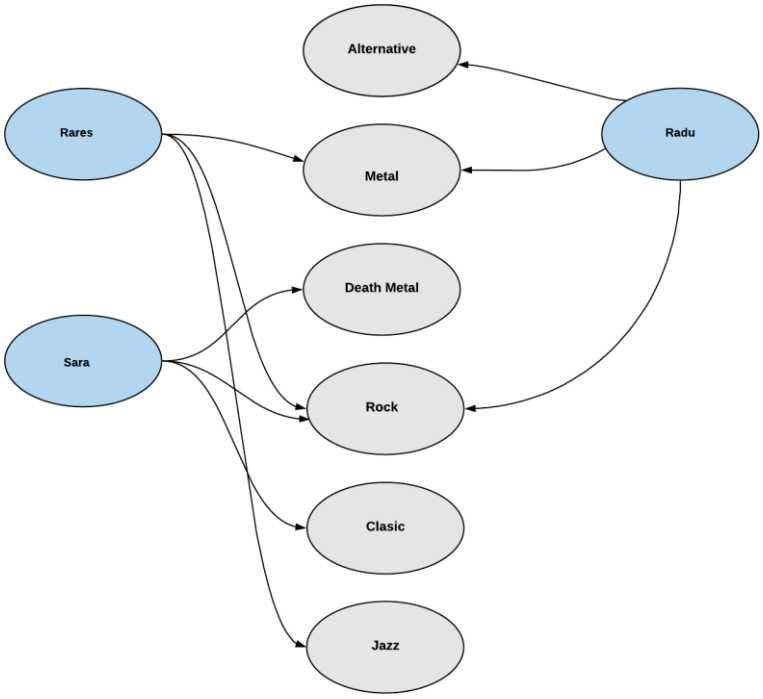


Figura 6. Reprezentarea grafică a exemplului genurilor muzicale

Vom crea un grafic cu noduri ce conțin elementele pe care au fost făcute evaluări în cazul nostru genurile muzicale și utilizatorii reprezentați la rândul lor tot prin noduri. Pe baza evaluărilor efectuate de utilizatori asupra diferitelor elemente pe care le creăm, se creează marginile. Rezultatul este un graf bipartit, care este, de asemenea, numit un graf hamac datorită formei sale. Folosind acest grafic, putem decide asupra recomandărilor care trebuie făcute utilizatorilor. Este afișat un exemplu de grafic bazat pe exemplul utilizat în imaginea de mai sus.[12]

Algoritm de recomandare bazat pe grafuri

Grafurile, pur și simplu, sunt o modalitate de a modela relații complexe între multe obiecte. Un graf există ca o colecție de noduri și muchii. Un nod este doar un punct de date abstract - ar putea reprezenta orice. O persoană, un computer, o clădire, o intersecție sau orice lucru din jurul nostru. O muchie conectează două noduri și poate fi direcționată (informațiile curg doar într-o direcție).

Implementarea acestor tipuri de grafuri constă într-un număr mare de operații, dar acest lucru se poate simplifica cu ajutorul prototipurilor din JavaScript (JavaScript Prototypes) ce permit definirea de metode cu ușurință pentru toate instanțele unui anumit obiect. Avantajul este că metoda este aplicată prototipului, deci este stocată doar în memorie, o singură dată, fiecare instanță a obiectului putând avea acces la ea.

Crearea structurilor de date și a grafului

În continuare se va putea observa integrarea prototipului specific sintaxei JavaScript în cadrul soluției. Totodată se vor descrie și clasele cu ajutorul cărora s-a realizat implementarea algoritmului.

Mai întâi, se va defini un prototip de bază pentru comportamentul partajat între noduri și muchii. Se va numi *Unit*. Vrem ca *Unit* să poată specifica o entitate și să aibe proprietăți și metode legate de setarea și redarea acestor proprietăți.

```
class Unit {
  constructor(entity, properties) {
    this.entity = entity + "";
    this.load(properties || {});
  }

  load(properties) {
    let p = Object.create(null);

    Object.keys(properties).forEach(function(v) {
      p[v] = properties[v];
    });
    this.properties = p;
    return this;
  }

  set(property, value) {
    return (this.properties[property] = value);
  }

  unset(property) {
    return delete this.properties[property];
  }

  has(property) {
    return Object.prototype.hasOwnProperty.call(this.properties, property);
  }

  get(property) {
    return this.properties[property];
  }

  toString() {
    return [this.constructor.name, " (", this.entity, " ", JSON.stringify(this.properties), ")", ].join("");
  }
}
```

Figura 7. Implementarea clasei Unit

Următorul pas este definirea clasei *Node* unde să urmărim toate muchiile, cele de intrare și cele de ieșire (pentru direcționalitate). De asemenea, se specifică o metodă de deconectare *Unlink* pentru a elimina toate marginile conectate.

```
class Node extends Unit {
  constructor(entity, properties) {
    super(entity, properties);
    this.edges = [];
    this.inputEdges = [];
    this.outputEdges = [];
  }

  unlink() {
    let edges = this.edges;

    for (let i = 0, len = edges.length; i < len; i++) {
      edges[i].unlink();
    }

    return true;
  }
}
```

Figura 8. Implementarea clasei Node

Mai departe, după implementarea celor două clase de bază trebuie să creăm funcționalitățile ce se vor ocupa de conectarea nodurilor între ele. Acestea se vor realiza în clasa *Edge*, unde se vor memora un inputNode, un outputNode și dacă muchia este sau nu o legătură duplex (bidirecțională).

```
class Edge extends Unit {
  constructor(entity, properties) {
    super(entity, properties);

    this.inputNode = null;
    this.outputNode = null;
    this.duplex = false;
    this.distance = 1;
  }

  // link a specific node in a certain direction
  _linkTo(node, direction) {
    if (direction <= 0) {
      node.inputEdges.push(this);
    }
    if (direction >= 0) {
      node.outputEdges.push(this);
    }
    node.edges.push(this);
    return true;
  }

  // link two nodes, optionally make edge bidirectional (duplex)
  link(inputNode, outputNode, duplex) {
    this.unlink();
    this.inputNode = inputNode;
    this.outputNode = outputNode;
    this.duplex = !!duplex;

    if (duplex) {
      this._linkTo(inputNode, 0);
      this._linkTo(outputNode, 0);
      return this;
    }
    this._linkTo(inputNode, 1);
    this._linkTo(outputNode, -1);
    return this;
  }
```

Figura 9. Implementarea clasei Edge

Pe lângă funcționalitățile de link ale nodurilor în cadrul clasei *Edge* se vor mai regăsi și următoarele metode *setDistance* – ce va seta distanța de traversare, *setWeight* – ce va seta importanța muchiei fiind reprezentată de raportul 1/distanța, *oppositeNode* – va returna nodul opus dat de un nod de pornire, *unlink* – ce va elimina conexiunile dintre noduri și va deconecta muchiile. Implementarea completă a clasei *Edge* se poate găsi la sfârșitul acestei lucrări la secțiunea Anexe.

În continuare după definirea claselor, cu ajutorul lor se va crea graful după cum se poate observa în imaginea de mai jos:

```
//Graph implementation

// Create nodes...
let user = `${username.name}`;
let recipe = `${recipe.name}`;
let joe = new Node("person");
joe.set("name", user);

let recipe = new Node("recipe");
recipe.set("name", recipe);

// Create edge...
let likes = new Edge("likes");

// link them!
likes.link(user, recipe);

// add more nodes...
let notch = new Node("person", user);
let created = new Edge("created").link(user, recipe);
```

Figura 10. Crearea grafului

Maparea pe comportamentul utilizatorului

Acum, odată ce structurile de date de bază ale graficului au fost implementate, putem memora comportamentul utilizatorilor într-un graf. În cadrul aplicației avem trei moduri de bază prin care utilizatorul poate interacționa cu o înregistrare: să vizualizeze o rețetă, să favorizeze o rețetă, dar și opțiunea de a vedea rețetele cele mai apreciate. Se vor folosi aceste trei acțiuni ca și muchii și vom defini două tipuri de noduri, utilizatori și înregistrare. Graful va fi ansamblat astfel:

```
//assemble our graph
let users = getUsers(); // abstract function to get user data
let views = getViews();
let favorites = getFavorites();

// quick and dirty O(n) function to get a node by id
function getNodeById(nodes, id) {
  return nodes.filter(function(node) {
    return node.get("id") === id;
  })[0];
}

users = users.map(function(user) {
  return new Node("user", user);
});

views = views.map(function(view) {
  return new Edge("view").link(
    getNodeById(users, view.user_id),
    getNodeById(listings, view.listing_id)
  );
});

favorites = favorites.map(function(favorite) {
  return new Edge("favorite").link(
    getNodeById(users, favorite.user_id),
    getNodeById(listings, favorite.listing_id)
  );
});
```

Figura 11. Asamblarea grafului

Înainte de a începe să oferim recomandări, trebuie să alocăm greutăți (sau distanțe) fiecărui comportament al utilizatorului. Se dorește ca acțiunile mai importante să fie ponderate mai favorabil. "Greutatea" și "distanța" sunt inverse între ele, o greutate mai mare echivalează cu o distanță mai mică.

În cadrul soluției curente se vor stabili distanțele deoarece acestea sunt mai ușor de folosit atunci când se traversează grafuri. Întrucât favoritele sunt cele mai importante, le vom stabili la o distanță de 1, aprecierile la 2 și vizualizările la 3.

```
//set distances
views.forEach(function(view) {
  view.setDistance(3);
});

likes.forEach(function(request) {
  request.setDistance(2);
});

favorites.forEach(function(favorite) {
  favorite.setDistance(1);
});
```

Figura 12. Setarea distanțelor

Distanța este calculată ca lungimea totală a muchiilor dintre cele două noduri. Ceea ce se întâmplă pentru a genera recomandări este traversarea grafului din exterior de la utilizator și găsirea tuturor celor mai apropiate favorite în ordinea în care apar. Apoi pe baza favoritelor găsite se vor recomanda rețete asemănătoare. Pentru parcurgerea grafului creat se va utiliza *UnitGraph* care este o librărie special creată pentru grafuri, ce oferă un wrapper peste graful creat cu ajutorul claselor definite anterior, ce permite găsirea celor mai apropiate noduri pe un graf sau urmărirea celei mai scurte rute dintre două noduri.

```
// get the closest 'favourites' nodes, at a minimum depth (distance) of 3
let results = graph.closest(node, {
  compare: function(node) {
    return node.entity === "favourites";
  },
  minDepth: 3,
  count: 100,
});

// results is now an array of Paths, which are each traces from your starting node to your result node...
let resultNodes = result.map(function(path) {
  return path.end();
});
```

Figura 13. Preluarea celor mai apropiate noduri favorite

Se observă că valoarea proprietății minDepth din interiorul graph.closest () este 3. S-a atribuit această valoare pentru a împiedica recomandarea unor rețete favorite pe care utilizatorul le are marcate deja (distanțele de 1 și respectiv 2). La o distanță

de cel puțin 3, se garantează recomandarea de rețete favorite pe care nu le-a mai văzut anterior.

Concluzii

În cadrul acestui capitol am prezentat, pe scurt, o introducere în ceea ce înseamnă un algoritm de recomandare. Ca mai apoi să aflăm cum se poate implementa unul, oferind exemple de implentare. În capitolul următor vom face o descriere amănunțită a părții de implementare a soluției împreună cu metodele folosite.

Capitolul III

Elemente de arhitectură și implementare

În acest capitol se va regăsi prezentarea arhitecturii generale a aplicației și modul în care s-au implementat cele mai importante componente, printre care se numără partea de recunoaștere a ingredientelor, cea de redare și de sortare a ingredientelor, afișarea rețetelor în urma filtrării lor cu ajutorul algoritmului de recomandare (modulul cel mai important pentru orice aplicatie de recomandări), urmând partea în care se poate crea o listă de cumpărături cu ingredientele lipsă.

Arhitectura generală

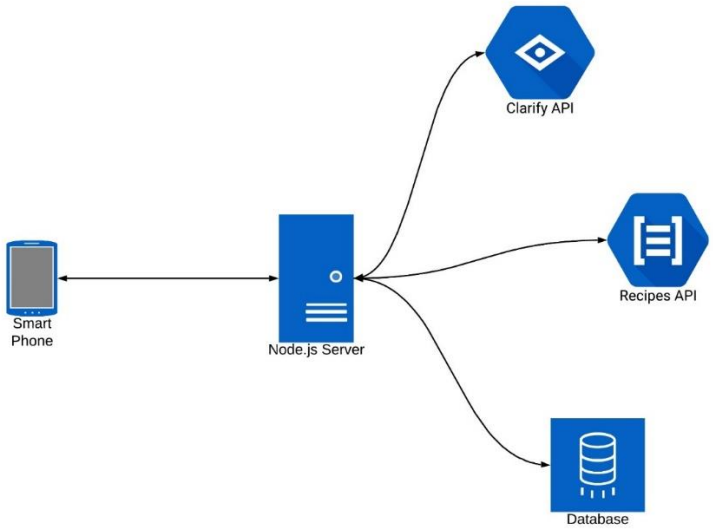


Figura 14. Arhitectura generală a proiectului

Arhitectura este formată din trei module principale și anume: identificarea ingredientelor, recomandarea rețetelor și creearea de liste de cumpărături precum și a rețetelor personalizate. După cum îi spune și numele *identificarea ingredientelor* este partea din aplicație care ajută la recunoașterea ingredientelor ce se regăsesc în poza realizată cu camera telefonului. Componenta de *recomandări* se ocupă de partea de redare a rețetelor utilizatorului în funcție de preferințele lui folosindu-se de un algoritm de recomandări ce îmbunătățește conținutul cu fiecare utilizare a aplicației.

În final componenta de *creare de liste* de cumpărături și de rețete personalizate unde se realizeză o filtrare a ingredientelor, ce nu sunt comune, dintre cele care au fost identificate din imagine și a celor conținute de rețetă și se crează o listă de cumpărături cu cele lipsă. Pentru cea din urmă opțiune se poate crea propria rețetă cu ingrediente și instrucțiuni.

Modulele aplicației și interacțiunea dintre ele

Aplicația *ChefAssistant* se bazează pe strânsa legatură dintre modulele prezentate anterior după cum urmează: dispozitivul mobil trimite poza în format base64 către server de unde se apelează API-ul Clarifai, după primirea răspunsului se oferă posibilitatea clientului de a selecta ingredientele pe baza cărora dorește să caute rețete, după care se va trimite lista ingredientelor către server. Serverul va face un apel către API-ul Spoonacular ce va oferi ca și răspuns rețetele ce conțin ingredientele trimise.

După primirea răspunsului, se va trece prin algoritmul de recomandare în partea de server, trimițând apoi clientului cele mai bune recomandări bazate pe numărul de aprecieri și după rețetele favorite ale utilizatorului. După ce se alege de către utilizator rețeta ce va fi gătită se va salva alegerea lui în baza de date ca pe viitor să se poată îmbunătăți sistemul de recomandare în vederea oferirii celor mai bune alegeri.

Flow-ul aplicației

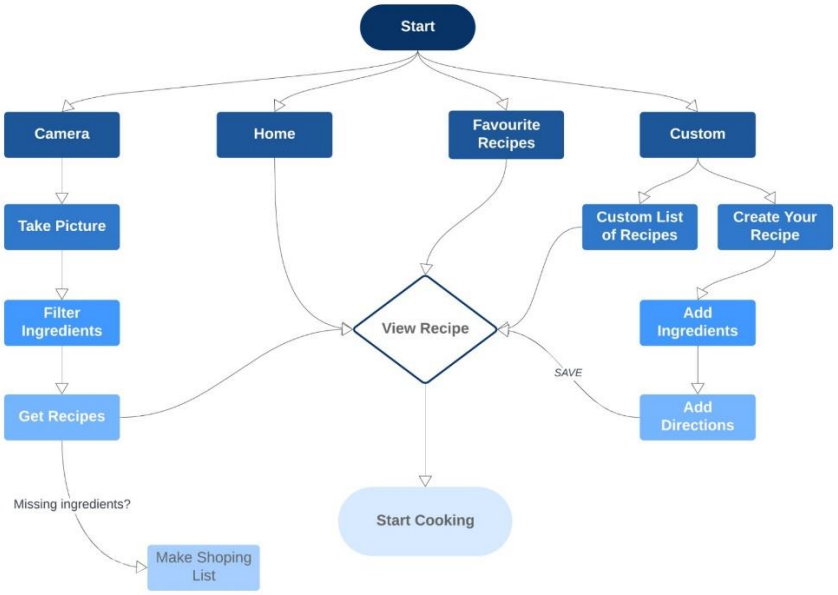


Figura 15. Flow-ul complet al aplicației

Principalul flow al aplicației se poate descrie după cum urmează. La lansarea aplicației vom avea prezent în partea de jos a ecranului un meniu ce va fi compus din următoarele tab-uri *Home, Recipes, Favourite, Shopping List* și *Camera* ce este accesibilă din cadrul tab-ului Home în partea sângă sus. În momentul selectării *Camerei* vom putea face o fotografie a ingredientelor, după care vom avea posibilitatea de a filtra ingredientele identificate și ulterior ni se vor afișa rețetele sugerate. Urmând apoi prin selectarea unei rețete să îi putem vedea totalitatea

ingredientelor împreună cu instrucțiunile pentru gătit. După vizualizarea întregii rețete avem opțiunea de a crea o listă de cumpărături ce va conține ingredientele pe care nu le avem momentan la îndemână.

La selectarea opțiunii *Acasa (Home)* vom putea vedea rețetele recomandate de algoritmul nostru bazate pe cele adăugate ca și favorite având posibilitatea de a selecta una dintre acestea fără a mai face o poză nouă. În cadrul *Rețete favorite (Favourite)* ni se vor pune la dispoziție rețetele marcate ca și favorite. Conținutul tab-ului *Shopping List* va fi format din ingredientele lipsă din cadrul unei rețete.

Modulul Camera

Pentru identificarea ingredientelor după realizarea fotografiei, ea este transformată în format base64 care ulterior va fi trimisă componentei *IngredientsScreen* de unde se va apela API-ul Clarifai. În cadrul modului Camera se realizează lasarea camerei telefonului numai în cazul oferirii permisiunii de către utilizator. Fără acordarea permisiunii utilizatorul nu va putea fotografia ingredientele.

```
render() {
  const { hasCameraPermission } = this.state;

  if (hasCameraPermission === null) {
    return <Container />;
  } else if (hasCameraPermission === false) {
    return <Text>No access to camera</Text>;
  } else {
    return (
      <Container style={styles.container}>
        <Camera
          style={styles.camera}
          type={this.state.type}
          ref={ref => {
            this.camera = ref;
          }}
        >
          <Container style={styles.cameraactions}>
            <Button large transparent style={styles.button}>
              <Icon style={styles.icon} name="ios-square-outline" />
            </Button>
            <Button
              large
              transparent
              style={styles.button}
              onPress={this.takePicture}
            >
              <Icon style={styles.cameraicon} name="ios-camera" />
            </Button>
            <Button
              large
              transparent
              style={styles.button}
              onPress={this.switchCamera}
            >
              <Icon style={styles.icon} name="ios-reverse-camera" />
            </Button>
          </Container>
        </Camera>
      </Container>
    );
  }
}
```

Figura 16. Redarea camerei pe baza permisiunii

Identificarea ingredientelor

Pentru prezicerea ingredientelor ne folosim de *Clarifai* API. El va returna o listă de concepte cu probabilități corespunzătoare cu privire la modul în care aceste concepte se regăsesc în imagine. Când facem o predicție prin intermediul API-ului, trebuie să precizăm ce model vrem să utilizăm. Acesta poate fi un model pre-construit al lui Clarifai sau un model personalizat instruit de catre noi. Un model conține un grup de concepte. Un model va "vedea" doar conceptele pe care le conține. De asemenea, putem specifica mai mulți parametri pentru predicții.

Imaginile pot fi trimise către API în două moduri via url sau via bytes. În cadrul acestei aplicații vom folosi cea din urmă variantă după cum urmează, în cadrul componentei *IngredientsScreen* vom defini cheia pentru conectarea la API ca mai apoi să realizăm primul apel.

```
import Clarifai from "clarifai";

import { CLARIFAI } from "../../constants/Config";
import IngredientList from "../../components/IngredientList/IngredientList";

const clarifaiApp = new Clarifai.App({
  apiKey: CLARIFAI.API_KEY,
});
```

Figura 16. Conectarea la API-ul Clarifai prin cheia specifică

```
getIngredients() {
  const { navigation } = this.props;
  const base64 = navigation.getParam("base64", "");

  clarifaiApp.models.predict(Clarifai.FOOD_MODEL, { base64 }).then(
    res => {
      this.setState({
        ingredients: res.outputs[0].data.concepts,
        loading: false,
      });
    },
    err => {
      console.error("Clarifai error: ", err);
    }
  );
}
```

Figura 17. Prezicerea răspunsului pe baza imaginii

După cum se poate observa și din cod modulul folosit în cazul aplicației noastre este cel de mâncare (FOOD_MODEL). Ca mai apoi răspunsul primit de la predicție să aibe urmatoarea formă:

```
"outputs": [
  {
    "created_at": "2019-05-19T17:04:46.773830331Z",
    "data": {
      "concepts": [
        {
          "app_id": "main",
          "id": "ai_NDbbpCv1",
          "name": "vegetable",
          "value": 0.96186477
        },
        {
          "app_id": "main",
          "id": "ai_mPjqBnPk",
          "name": "carrot",
          "value": 0.9588791
        },
        {
          "app_id": "main",
          "id": "ai_3fJXxTPQ",
          "name": "sausage",
          "value": 0.95605826
        },
```


Figura 18. Prima parte a răspunsului cu predicție mare


```
        {
          "app_id": "main",
          "id": "ai_0wh0dJkQ",
          "name": "sweet",
          "value": 0.37237075
        },
        {
          "app_id": "main",
          "id": "ai_0s603rs2",
          "name": "tuber",
          "value": 0.36426216
        },
        {
          "app_id": "main",
          "id": "ai_dmQ7SN9X",
          "name": "frankfurters",
          "value": 0.34405565
        }
      ]
    },
```


Figura 19. Ultima parte a răspunsului cu predicție mică

Filtrarea ingredientelor în urma predicţiei

După predicția oferită pe baza imaginii utilizatorul va putea selecta ingredientele ce se potrivesc cu cele din imaginea realizată. Vom parcurge array-ul de ingrediente afișând denumirea ingredientului și procentajul predicției împreună cu un checkbox.

Acest lucru se realizează după cum se poate observa în imaginea de mai jos:

```
  render() {
    const { ingredients } = this.props;

    return (
      <Container style={styles.container}>
        <Content>
          {ingredients.map(ingredient => (
            <ListItem key={ingredient.id}>
              <Body>
                <Text>{ingredient.name}</Text>
              </Body>
              <CheckBox
                checked={this.checkIfItemSelected(ingredient.name)}
                onPress={() => this.onCheckBoxPress(ingredient.name)}
                color="green"
              >
                <Body>
                  <Text>{ingredient.name}</Text>
                </Body>
              </CheckBox>
            </ListItem>
          ))}
        </Content>
        <Button
          primary
          full
          iconLeft
          disabled={!this.state.selectedIngredients.length}
          onPress={() =>
            this.props.onGetRecipesPress(this.state.selectedIngredients)
          }
        >
          <Icon name="ios-list-box" />
          <Text>Get Recipes</Text>
        </Button>
      </Container>
    );
  }
}

export default withLoading(IngredientList);
```

Figura 20. Redarea listei de ingrediente

Afișarea listei de rețete

Pentru redarea listei de rețete ne folosim de avantajul oferit de framework-ul React Native de a crea componente cât mai modulare ce permit reutilizarea lor. Prin crearea acestor componente reutilizabile codul este mult mai lizibil, fără a fi nevoie de a o încărca cu funcționalități multiple, ea știind să facă doar un singur lucru. Acest tip de componentă poartă denumirea de componentă pură, care indiferent de input-ul primit va returna mereu același lucru, în cazul nostru o listă de Card-uri.

```
class RecipeList extends Component {
  render() {
    const {
      recipes,
      favouriteRecipes,
      onRecipePress,
      onFavouritePress,
    } = this.props;

    return (
      <Content>
        {recipes.map(recipe => {
          return (
            <RecipeCard
              key={recipe.id}
              recipe={recipe}
              onRecipePress={onRecipePress}
              isFavourite={isFavouriteRecipe(recipe, favouriteRecipes)}
              onFavouritePress={onFavouritePress}
            />
          );
        })}
      </Content>
    );
  }
}

export default withLoading(RecipeList);
```

Figura 21. Componeta de redare a listei de rețete

După cum se poate observa din implementarea prezentată anterior *RecipeList* randează o altă componentă *RecipeCard* care la rândul ei este tot o componentă pură, știind să randeze pe baza datelor primite de la *RecipeList* un Card cu diferite CardItem-uri. Acestea sunt componente React Native cu ajutorul cărora redăm într-un mod vizual plăcut lista de rețete. Implementarea completă a componentei *RecipeCard* se regăsește la finalul documentației în secțiunea Anexe.

Creearea listei de cumpărături

După alegerea rețetei dorite pentru a o putea găti avem opțiunea de a crea o listă de cumpărături în cazul în care o parte din ingredientele rețetei nu se regăsesc printre cele identificate în imagine. Acest lucru se va realiza astfel:

Favorite

În cadrul paginii de favorite se vor regăsi rețetele marcate de utilizator ca fiind favorite. Această marcare este folosită de asemenea și în cadrul algoritmului de recomandare, pe baza lor oferindu-se rețete asemănatoare cu cele ce se regăsesc în această pagină. Pentru a realiza acest lucru este necesară cearea unui nou array în care se vor regăsi rețetele marcate de utilizator în urma unei filtrări. Acest array va fi stocat pe state-ul general aplicației deoarece în momentul adăugării unei noi rețete toate componentele ce se folosesc de acest array să fie înștiințate în momentul unei schimbări și pentru favorizarea re-randării componentelor.

S-a dorit ca în momentul închiderii aplicației aceste rețete să rămână stocate în memoria telefonului pentru a nu se pierde în momentul unei noi lansări a aplicației. Acest lucru a putut fi realizat cu ajutorul *Context API* - ce oferă o modalitate de a transmite datele prin arborele componentelor, fără a fi nevoie să fie transmis manual în fiecare nivel prin proprietatea props.

```
export const StoreContext = React.createContext({
  recipes: [],
  favouriteRecipes: [],
  similarRecipes: [],
  setRecipes: () => {},
  onFavouritePress: () => {},
});
```

Figura 22. Crearea Contextului

```
onFavouritePress(recipe) {
  this.setState(
    ({ favouriteRecipes }) => {
      if (isFavouriteRecipe(recipe, favouriteRecipes)) {
        return {
          favouriteRecipes: favouriteRecipes.filter(({ id }) => {
            return id !== recipe.id;
          }),
        };
      }

      return {
        favouriteRecipes: favouriteRecipes.concat(recipe),
      };
    },
    () => {
      const { similarRecipes, favouriteRecipes } = this.state;
      if (!similarRecipes.length && favouriteRecipes.length) {
        this.getSimilarRecipes();
      }
    }
  );
}
```

Figura 23. Marcarea ca și favorită a unei rețete

Elemente de interfață

Elementele de interfață prezentate în această lucrare de licență sunt: pagina de redare a camerei, vizualizarea ingredientelor și a recomandărilor de rețete, dar și vizualizarea rețetei în totalitate. Pe baza alegerii făcute de către utilizator în privința rețetei pe care o va găti vom putea antrena algoritmul de recomandări pentru a deveni mai bun și pentru a putea oferi rezultate cât mai precise.

Prezentarea aplicației

Pentru o mai bună înțelegere a conceptelor expuse anterior în cele ce urmează voi prezenta aplicația întro manieră vizuală cu explicații aferente acolo unde este nevoie.

La lansarea aplicației se afișează ecranul *Home*, care va fi populat cu rețete recomandate dacă avem rețete adăgate la favorite sau în situația opusă când ne va fi afișat un mesaj ce ne informează că ar trebui să marcăm măcar o rețetă ca și favorită pentru a avea sugestii.

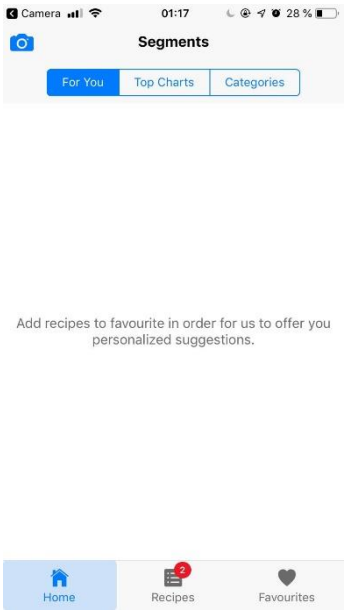


Figura 24. Ecranul Home fără recomandări

Figura 25. Ecranul Home cu recomandări

Pentru a putea face o poză ingredientelor trebuie lansată camera cu ajutorul butonului din partea stângă sus a ecranului. În *ecranul de Cameră* avem posibilitatea de a schimba camera principală cu cea frontală și de a face poza.

Figura 26. Camera

După realizarea pozei vom fi redirecționați către *ecranul de ingrediente*, ce vor avea afișat și procentajul de predicție. Aici se vor putea selecta indredientele dorite, iar când avem cel puțin un ingredient selectat butonul Get Recipes, devine activ și prin intermediul lui se primesc sugestii de rețete. Tot în cadrul acestui ecran avem posibilitatea de a ne intoarce la Cameră prin butonul de back, din stânga ecranului sau de a renunța la a mai face o poză prin butonul cancel, care face redirecționarea către Home.

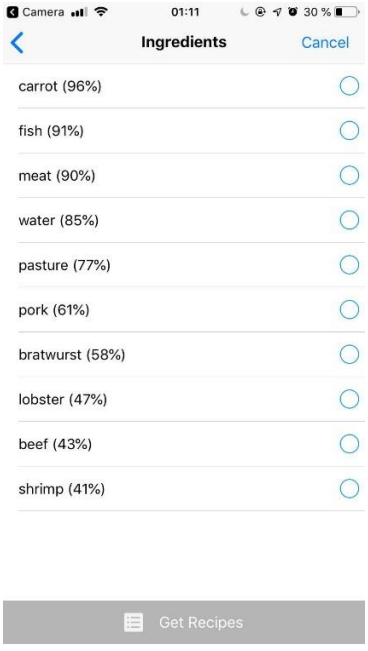


Figura 27. Ecranul Ingredients fără nicio selecție

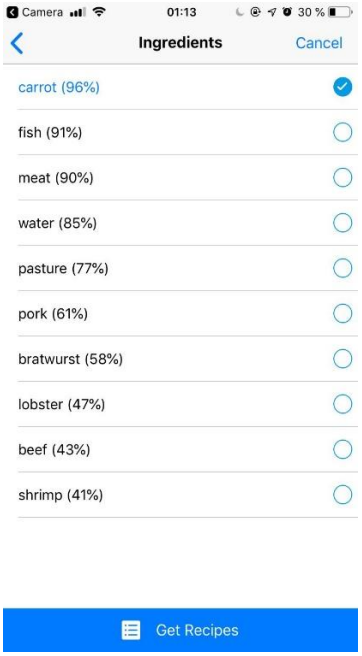


Figura 28. Ecranul Ingredients cu selecție

Rețetele primite ca și răspuns în urma apelului la API, vor putea fi vizualizate în ecranul *Recipes* care va fi populat doar după ce se trece prin precedentele două prezentate, altfel se va afișa un mesaj ce va informa utilizatorul că este necesară o poză pentru a vedea sugestii de rețete.

Rețetele vor fi afișate sub formă de Card-uri. Fiecare Card având opțiunea de a adăuga rețeta respectivă la favorite, reprezentată printr-o inimă. În momentul selectării unui Card se va face redirecționarea către ecranul de detalii a rețetei.

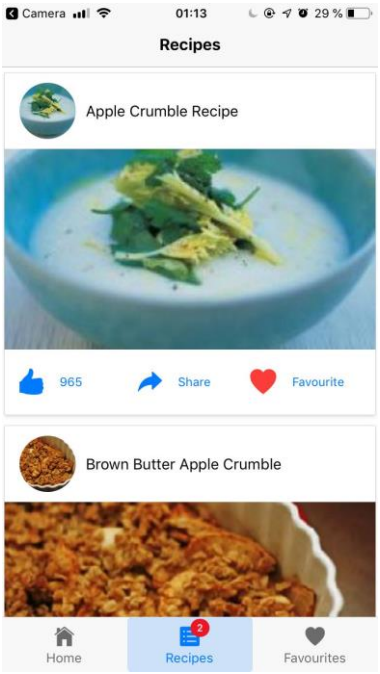


Figura 29. Ecranul Recipes cu rețetă favorită

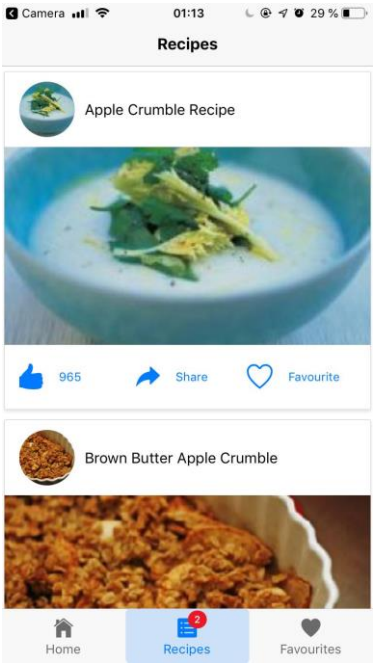


Figura 30. Ecranul Recipes

Figura 31. Ecranul Recipes la lansarea aplicației

În momentul în care ajungem pe *ecranul de detalii ale rețetei*, se vor putea vizualiza poza rețetei împreună cu ingredientele necesare preparării, în ce tipuri de rețete se încadrează(ex. Vegetariene, fără gluten etc.), instrucțiunile de gătire și durata medie de gătire. De asemenea va fi disponibilă și opțiunea creării unei liste de cumpărături ce va conține ingredientele lipsă. Avem posibilitatea de a adăuga rețeta la favorite și din ecranul acesta.

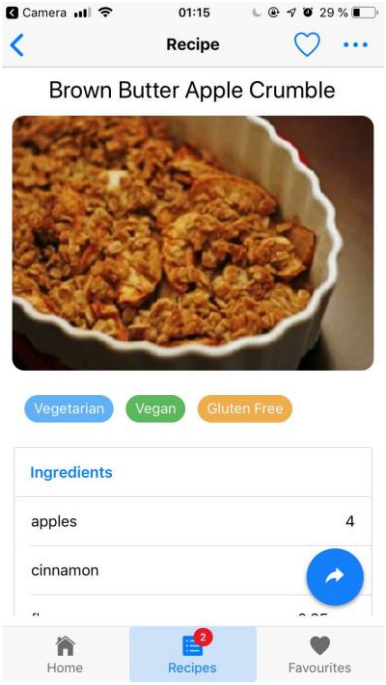


Figura 32. Ecranul detalii rețetă

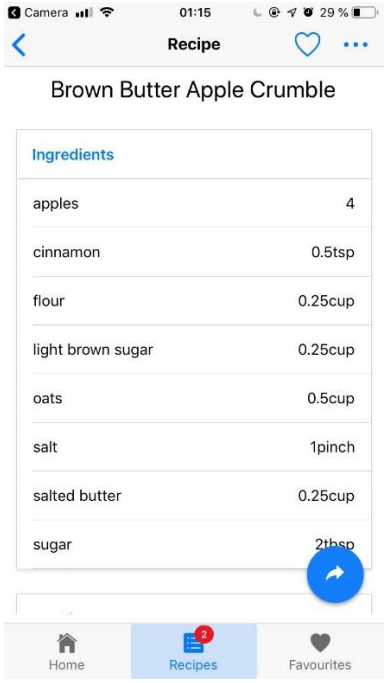


Figura 33. Ecranul detalii rețetă secțiunea ingrediente

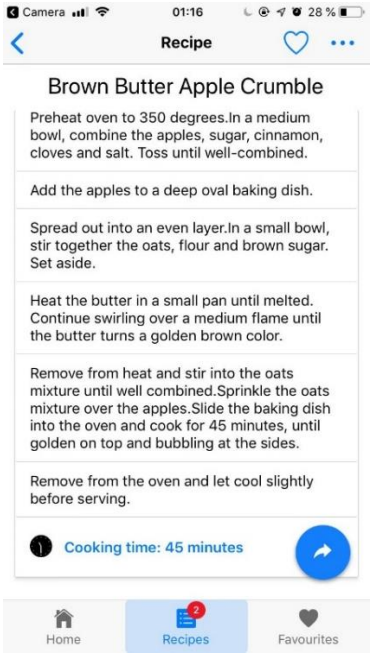


Figura 34. Ecranul detalii rețetă secțiunea instrucțiuni

Ecranul favorite conține lista rețetelor care au fost marcate de utilizator. În cazul în care nu este marcară nicio rețetă ca și favorită va fi afișat un mesaj ce va informa utilizatorul cu ce este necesar pentru a fi populată lista. În momentul în care se elimină o rețetă de la favorite ea va fi eliminată automat din listă. De asemenea prin selectarea rețetei vom fi redirecționați către ecranul de detalii.

Figura 35. Ecranul Favorite fără rețete marcate

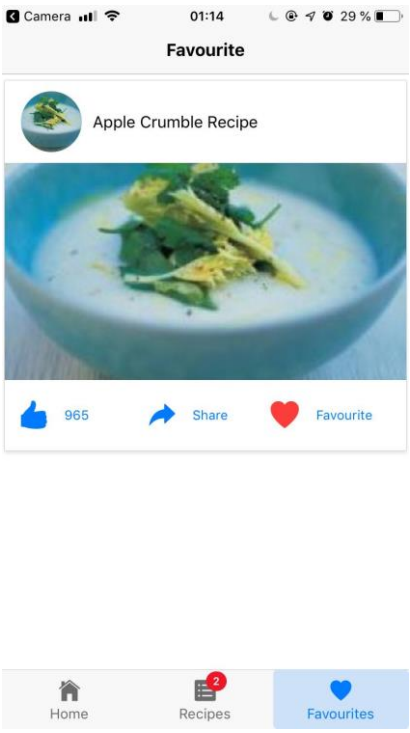


Figura 36. Ecranul Favorite cu rețetă marcată

Din ecranul de *detalii ale rețetei* vom avea opțiunea de a crea o listă de cumpărături *Shopping List* ce va fi de tipul unei liste cu checkbox-uri pentru a selecta produsele achiziționate. În momentul în care nu există niciun produs adăugat în listă se va afișa un mesaj informativ pentru utilizator după cum se poate observa în imaginea de mai jos.

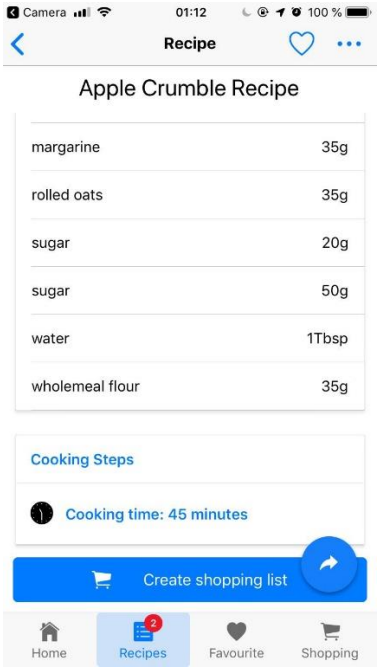
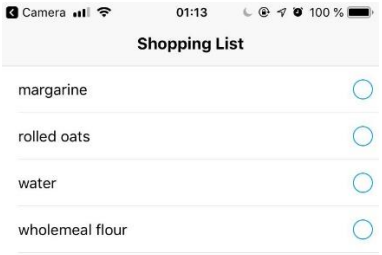


Figura 37. Opțiunea de creare a listei de cumpărături

Figura 38. Lista de cumpărături goală



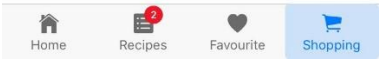
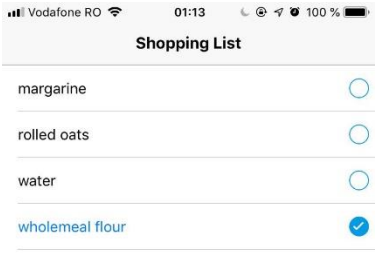


Figura 39. Lista de cumpărături populată



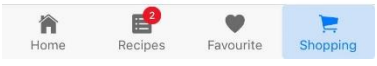


Figura 40. Produs din listă selectat

Concluzii

În acest capitol a fost prezentatată cea mai importantă parte a lucrării de licență și anume arhitectura aplicației punând accent pe modularitate, scalabilitate și

flexibilitate. Aceste trei elemente sunt și cele mai importante atunci când vine vorba despre construirea unei arhitecturi pentru o aplicație, oricare ar fi ea.

Totodată am putut vedea și o descriere mai amănunțită a modulelor aplicației împreună cu relația dintre ele.

Concluzii

Concluzii finale

Arhitectura pe baza căreia a fost construită acestă aplicație împreună cu dezvoltarea algoritmului de recomandare oferă o bază pentru creearea de aplicații asemănătoare. Totodată am avut ca și scop și oferirea unei noi modalități mult mai ușoare de folosit pentru o activitate ce uneori poate parea anevoioasă, gătitul.

Contribuţii

Prin acest proiect ne-am propus realizarea unei aplicații ce este mereu la îndemâna utilizatorului și este foarte ușor de folosit economisind atât timp cât și bani în vederea preparării de rețete cu ingrediente ce le avem la îndemână, dar care sunt și recomandate de un număr mare de persoane.

Această aplicație dezoltată pe arhitectura prezentată în capitolele de de mai sus poate rula atât pe dispozitivele ce rulează IOS cât și pe cele ce rulează Android.

Acest lucru este posibil datorită tehnologiilor moderne folosite pentru dezvoltarea lui precum ReactNative ce permite folosirea aceluiași cod atât pentru IOS cât și pentru Android, împreună cu Expo SDK, Clarifai și Spoonacular.

Utilizatorul are acces la cameră pentru a realiza o fotografie a ingredientelor, apoi are posibilitatea de a selecta anumite ingrediente prezise sau de a adăuga unele lipsă ca mai apoi să își aleagă rețeta preferată din sugestiile oferite cu ajutorul algoritmului de recomandare. Totodată el își va putea crea o listă de cumpărături cu ingredientele lipsă, dacă ele există. Aplicația poate fi folosită și ca un carnet de rețete cu ajutorul opțiunii de rețete personalizate unde utilizatorul va putea crea propriile rețete.

Îmbunătăţiri viitoare

Pe viitor, aplicației i se pot aduce îmbunătățiri pe partea de funcționalități precum posibilitatea de a oferi recomandări de unde ar putea fi achiziționate produsele lipsă din lista de cumpărături. De asemenea posibilitatea de a face comandă printr-un serviciu online a acelor ingrediente.

Este posibilă implementarea de convertire a cantităților din cadrul rețetelor, de a oferi valori nutriționale. Totodată atunci când ești pe o rețetă se poate crea opțiunea să faci o poză la ingredientele pe care le ai și să-ți spună ce ingrediente lipsesc.

Se poate oferi de asemenea posibilitatea ca utilizatorii să își creeze un plan alimentar pe o săptămână împreună cu posibilitatea de a adăuga notificări. Opțiunea de a accesa sfaturi de gătit sau de a fi anunțat când este sezonul unui anumit fruct sau unei anumite legume împreună cu o listă de câteva sugestii de rețete ce pot fi făcute pe baza lor.

Un machine model pentru recunoașterea locală a ingredientelor, să nu mai depindem de un API, algoritm de recomandare mai bun.

Un alt plus de valoare ar putea fi adus de accesibilitatea mai usoară pentru a încuraja și persoanele cu dizabilități să poată folosi aplicația prin comenzi vocale.

Bibliografie

[1] https://www.escoffieronline.com/top-apps-for-finding-recipes-for-ingredients-you-already-have
[2] https://caloriemama.ai/
[3] https://profs.info.uaic.ro/~busaco/teach/courses/wade/presentations/web-nodejs.pdf
[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[5] https://reactjs.org/docs/getting-started.html
[6] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3
[7] https://facebook.github.io/react-native
[8] https://spoonacular.com/food-api
[9] https://www.clarifai.com/developer/guide
[10] https://docs.expo.io/versions/v32.0.0/sdk/camera/
[11] https://medium.com/react-native-training/best-practices-for-creating-react-native-apps-part-1
[12] “Recomandarea sistemelor folosind teoria grafurilor” de Vishal Venkatraman

Anexe

```
class Edge extends Unit {
  constructor(entity, properties) {
    super(entity, properties);

    this.inputNode = null;
    this.outputNode = null;
    this.duplex = false;
    this.distance = 1;
  }

  // link a specific node in a certain direction
  _linkTo(node, direction) {
    if (direction <= 0) {
      node.inputEdges.push(this);
    }
    if (direction >= 0) {
      node.outputEdges.push(this);
    }
    node.edges.push(this);
    return true;
  }

  // link two nodes, optionally make edge bidirectional (duplex)
  link(inputNode, outputNode, duplex) {
    this.unlink();
    this.inputNode = inputNode;
    this.outputNode = outputNode;
    this.duplex = !!duplex;

    if (duplex) {
      this._linkTo(inputNode, 0);
      this._linkTo(outputNode, 0);
      return this;
    }
    this._linkTo(inputNode, 1);
    this._linkTo(outputNode, -1);
    return this;
  }

  // distance for traversal
  setDistance(v) {
    this.distance = Math.abs(parseFloat(v) || 0);
    return this;
  }

  // weight is 1 / distance
  setWeight(v) {
    this.distance = 1 / Math.abs(parseFloat(v) || 0);
    return this;
  }

  // find the opposite node given a starting node
  oppositeNode(node) {
    if (this.inputNode === node) {
      return this.outputNode;
    } else if (this.outputNode === node) {
      return this.inputNode;
    }
    return;
  }

  // unlink edge, remove connections from nodes
  unlink() {
    let pos;
    let inode = this.inputNode;
    let onode = this.outputNode;
    if (!(inode && onode)) {
      return;
    }

    (pos = inode.edges.indexOf(this)) > -1 && inode.edges.splice(pos, 1);
    (pos = onode.edges.indexOf(this)) > -1 && onode.edges.splice(pos, 1);
    (pos = inode.outputEdges.indexOf(this)) > -1 &&
      inode.outputEdges.splice(pos, 1);
    (pos = onode.inputEdges.indexOf(this)) > -1 &&
      onode.inputEdges.splice(pos, 1);

    if (this.duplex) {
      (pos = inode.inputEdges.indexOf(this)) > -1 &&
        inode.inputEdges.splice(pos, 1);
      (pos = onode.outputEdges.indexOf(this)) > -1 &&
        onode.outputEdges.splice(pos, 1);
    }
    this.inputNode = null;
    this.outputNode = null;
    this.duplex = false;
    return true;
  }
}
```

Figura 41. Implementarea clasei Edge

```
import React, { Component } from "react";
import { Image } from "react-native";
import {
  Text,
  Card,
  CardItem,
  Left,
  Right,
  Thumbnail,
  Body,
  Button,
  Icon,
} from "native-base";

import styles from "./styles";

class RecipeCard extends Component {
  constructor(props) {
    super(props);

    this.normalizeImageUrl = this.normalizeImageUrl.bind(this);
  }

  normalizeImageUrl(url) {
    if (url.includes("https://")) {
      return url;
    }

    return `https://spoonacular.com/recipeImages/${url}`;
  }

  render() {
    const { recipe, isFavourite, onFavouritePress, onRecipePress } = this.props;

    const image = this.normalizeImageUrl(recipe.image);
    const favouriteIconStyles = isFavourite
      ? [styles.icon, styles.favourite]
      : styles.icon;
    const favouriteIconName = isFavourite ? "ios-heart" : "ios-heart-empty";

    return (
      <Card>
        <CardItem button onPress={() => onRecipePress(recipe)}>
          <Left>
            <Thumbnail source={{ uri: image }} />
            <Body>
              <Text>{recipe.title}</Text>
            </Body>
          </Left>
        </CardItem>
        <CardItem cardBody button onPress={() => onRecipePress(recipe)}>
          <Image
            source={{ uri: image }}
            style={{ height: 200, width: null, flex: 1 }}
          />
        </CardItem>

        <CardItem>
          <Left transparent>
            <Button transparent>
              <Icon name="ios-thumbs-up" style={styles.icon} />
              <Text>{recipe.likes}</Text>
            </Button>
          </Left>
          <Body>
            <Button transparent style={styles.button}>
              <Icon name="ios-share-alt" style={styles.icon} />
            </Button>
          </Body>
          <Right>
            <Button transparent onPress={() => onFavouritePress(recipe)}>
              <Icon name={favouriteIconName} style={favouriteIconStyles} />
            </Button>
          </Right>
        </CardItem>
      </Card>
    );
  }
}

export default RecipeCard;
```

Figura 42. Implementarea componentei RecipeCard

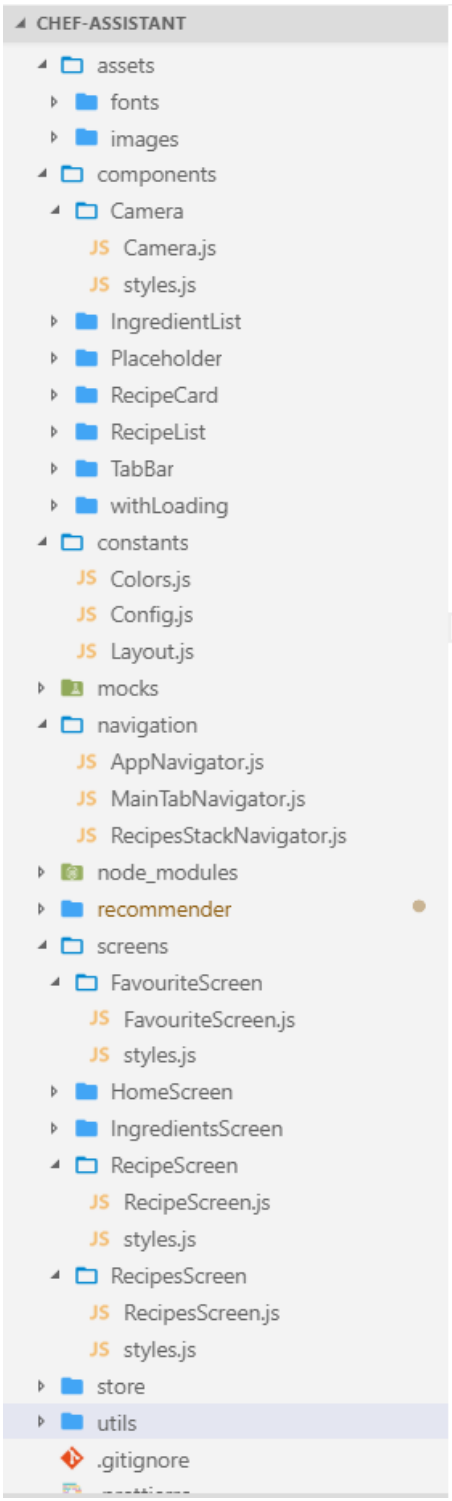


Figura 43. Structura folderelor