MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**
**DEPARTAMENTUL CALCULATOARE**

# My Health Center
# –
# APLICAȚIE WEB PENTRU MONITORIZAREA ȘI EVALUAREA NUTRIȚIEI ȘI ACTIVITĂȚILOR FITNESS

LUCRARE DE LICENŢĂ

Absolvent: **Andreea-Adriana PRODAN**

Coordonator ştiinţific: **As. Ing. Cosmina IVAN**

**2019**

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**
**DEPARTAMENTUL CALCULATOARE**

DECAN,
**Prof. dr. ing. Liviu MICLEA**

DIRECTOR DEPARTAMENT,
**Prof. dr. ing. Rodica POTOLEA**

Absolvent: **Andreea-Adriana PRODAN**

# My Health Center
# –
# APLICAȚIE WEB PENTRU MONITORIZAREA ȘI EVALUAREA NUTRIȚIEI ȘI ACTIVITĂȚILOR FITNESS

1. **Enunţul temei:** Proiectul își propune definirea, proiectarea și implementarea unei platforme online care să faciliteze urmărirea unui stil de viață sănătos, în manieră proprie, individuală sau cu ajutorul unui antrenor sau nutriționist.

2. **Conţinutul lucrării:** Introducere, Obiectivele proiectului, Studiu bibliografic, Analiză și fundamentare teoretică, Proiectare de detaliu și implementare, Proiectarea datelor, Testare și validare, Manual de instalare și utilizare, Concluzii și dezvoltări, Bibliografie.

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**:

5. **Data emiterii temei:** 1 noiembrie 2018

6. **Data predării:** 8 iulie 2019

Absolvent: _____

Coordonator ştiinţific: _____

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**
**DEPARTAMENTUL CALCULATOARE**

# Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a)_____
_____,
legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării
_____
_____
_____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data

_____

Nume, Prenume

_____

Semnătura

# Cuprins

# Capitolul 1. Introducere – Contextul proiectului

## 1.1. Contextul general al proiectului

Obiceiurile nesănătoase continuă tot mai mult să contribuie la creșterea obezității, în prezent aproape 30% din populația lumii fiind obeză sau supraponderală [1]. Similar, o dietă săracă poate duce la creșterea riscului de boli de inimă, hipertensiune arterială sau diabet de tip doi [2]. Prin alegerea inteligentă a alimentelor consumate, combinată cu activitatea fizică, dieta poate ajuta la prevenirea acestor probleme de sănătate precum și menținerea unei sănătăți generale sănătoase. Astfel a apărut nevoia a mai multor nutriționiști, dieteticieni și antrenori personali.

Deși monitorizarea folosind hârtia și creionul funcționează, utilizarea unei aplicații poate fi mai eficientă. Conform unui studiu publicat în Journal of Nutrition Education and Behavior [3], deși nu există o diferență din punct de vedere al eficacității ajutorului în pierderea greutății, participanții care și-au monitorizat dieta cu aplicație smartphone sau funcție memo au tins să persiste în studiu și au ratat mai puține zile în care să introducă datele, în comparație cu cei care și-au monitorizat dieta folosind hârtia și creionul. Astfel tot mai multe persoane încep să folosească aplicații pe acest domeniu, pentru a-și monitoriza alimentația și activitățile fizice.

## 1.2. Contextul specific al proiectului

Odată cu creșterea numărului de aplicații care ușurează urmărirea unei alimentații sănătoase și urmărirea efectuării activităților zilnice, a crescut și necesitatea găsirii unei aplicații care să ofere o modalitate cât mai ușoară de a realiza aceste lucruri. Deși în prezent există multe astfel de aplicații, nici una nu reușește să acopere toate problemele în întregime. Nevoile persoanelor variază, precum și preferințele acestora, iar odată cu introducerea unui număr mare de funționalități crește și nivelul de complexitate al utilizării aplicației. Luând acest lucru în considerare, trebuie să se urmărească cu atenție alegerea funcționalităților sistemului, astfel încât să cuprindă toate elementele esențiale care pot ajuta utilizatorii să își atingă scopul, fără însă a-l aglomera, pentru a nu copleși utilizatorii.

Menținerea unui stil de viață sănătos include atât alimentația cât și activitatea fizică. Asigurarea acestora se poate face individual, de către persoana care își propune acest obiectiv, sau printr-o colaborare cu un specialist în aceste domenii, cum ar fi antrenorul, nutriționistul și dieteticianul. Luând acest lucru în considerare, acest proiect va urmări integrarea specialiștilor, ca utilizatori separați, cu care clienții să poată colabora virtual. În același timp, nu vor fi neglijați nici utilizatorii care doresc să își facă monitorizarea singuri, lucru pe care platforma îl va permite, oferind în același timp și evaluări cu sfaturi pentru a le veni în ajutor.

Un aspect important este prezentarea tuturor soluțiilor și metodelor într-o manieră simplă, care să ajute persoanele să rămână consecvente. Cum în prezent fiecare este

foarte ocupat cu activitățile zilnice, o astfel de aplicație ar trebui sa vină cu o soluție simplă care să poată fi utilizată pe o perioadă lungă de timp.

Astfel s-a propus dezvoltarea unei platforme care să reușească să cuprindă toate aceste aspecte. Se va urmări, odată, oferirea unui mijloc de monitorizare a alimentației, care va avea funcțiile clasice ale aplicațiilor în nutriție și fitness, cum ar fi posibilitatea căutării datelor nutriționale ale alimentelor, salvarea personalizată a acestora, crearea istoricului alimentației și al planului de antrenament și vizualizarea progresului. Pe lângă acestea, se vor introduce grafice și diagrame pentru a ajuta utilizatorul să vizualizeze ușor datele.

Totodată, aceast proiect va urmări oferirea posibilității de a colabora cu o un antrenor sau nutriționist, prin intermediul unei pagini de chat și prin posibilitatea schimbului de planuri și date între aceștia.

## 1.3. Structura lucrării pe capitole

**Introducere** – descrie contextul problemei și necesitatea dezvoltării aplicațiilor în fitness și nutriție, o prezentare a conceptelor importante, precum și contextul proiectului.

**Obiectivele proiectului** – prezintă principalul obiectiv al proiectului și obiectivele secundare care vor trebui să fie atinse în cadrul dezvoltării acestui proiect

**Studiul bibliografic** – prezintă sistemele similare cu cel propus în acest proiect, cu o analiză a funcționalităților principale implementate de acestea, cu scopul de a realiza la final o comparație între aceste sisteme și proiectul propus. Tot aici se face o analiză a formulelor necesare în calcularea caloriilor și a macronutrienților necesari zilnic unei persoane și a calculării caloriilor arse în timpul unui exercițiu fizic.

**Analiză și fundamentare teoretică** – prezintă ecuațiile folosite, cazurile de utilizare, cerințele funcționale și nonfuncționale, cerințele resurselor necesare în dezvoltarea acestui sistem.

**Proiectare de detaliu și implementare** – se prezintă în detaliu arhitectura conceptuală a sistemului propus, API-urile utilizate, diagrama bazei de date, diagrama de deployment, diagrame de secvențe ale operațiilor complexe.

**Testare și validare** – prezintă modul în care au fost testate componentele aplicației și scenarii de test.

**Manual de instalare și utilizare** – prezintă instrumentele necesare pentru a putea rula acest proiect și pașii necesari care trebuie parcurși pentru instalare și utilizare, cu imagini captate din aplicație.

**Concluzii și dezvoltări ulterioare** – prezintă observațiile finale la care s-a ajuns în urma dezvoltării proiectului și dezvoltările ce ar putea fi aduse proiectului ulterior.

# Capitolul 2. Obiectivele Proiectului

În acest capitol se prezintă obiectivul principal al proiectului din tema aleasă, care se va urmări pe parcursul dezvoltării aplicației, urmat de obiectivele secundare, care vor trebui de asemenea să fie îndeplinite.

## 2.1. Obiectivele principale

Chiar dacă sesiunile față în față cu un client sunt fără îndoială importante, posibilitatea colaborării de la distanță poate fi utilă în monitorizarea progresului clientului și susținerea și motivarea acestuia. Fără nici o platformă de acest gen, un nutriționist, spre exemplu, trebuie să creeze un plan într-un fișier în format digital pe care să îl trimită clientului. Datele necesare le obține din surse diferite, iar calculele trebuie să le facă pe hârtie sau prin intermediul Excel-ului. Periodic vor fi necesare înregistrări ale rezultatelor pentru a obține un istoric al progresului. Clientul pe de altă parte va trebui să își noteze cât de exact a urmat programul. Printr-un singur exemplu se poate observa utilitatea unei platforme care să acopere aceste necesități.

Deși pe piață există multe aplicații în nutriție și fitness, nici una nu se concentrează pe oferirea unei căi virtuale de a colabora cu un un antrenor, nutriționist sau dietetician. Singura opțiune pe care o prezintă este de a alege un antrenor oferit de către aplicație. Mai mult, acestea merg fie pe domeniul nutriției, fie pe cel al fitness-ului, foarte puține cuprinzându-le pe ambele.

Principalul obiectiv al proiectului este dezvoltarea unei aplicații web care să ofere o cale de colaborare virtuală între un client și un antrenor sau nutriționist. Platforma va permite antrenorului și nutriționistului să creeze planuri de antrenament și nutriție pe care să le dea clientului. Aceștia vor avea la dispoziție instrumente care să îi ajute la obținerea planurilor personalizate prin oferirea a mai multor funcționalități, precum obținerea datelor unui aliment și calcularea automată a caloriilor. Pentru a putea monitoriza progresul clienților, aceștia vor putea vizualiza alimentația și antrenamentul clienților. Deoarece sunt persoane care poate doresc să își urmărească singuri alimentația și activitatea fizică, aplicația le va oferi această posibilitatea, precum o fac alte aplicații de nutriție și fitness deja existente.

## 2.2. Obiective secundare

Primul **obiectiv** secundar este de a obține o platformă care să acopere nevoile ambelor categorii de utilizatori, luând în considerare nivelul de cunoștințe al fiecărei categorii. Cum nu toți utilizatorii au cunoștințe complexe în domeniul nutriției și a fitness-ului, platforma va trebui să simplifice întregul proces de monitorizare a alimentației și a activităților fizice și să le ofere indicații în scopul obținerii obiectivelor propuse. Aflați la polul opus, antrenorii și nutriționiștii au un set de cunoștințe de specialitate în domeniul nutriției sau al fitness-ului. Din acest motiv platforma va trebui să le ofere opțiuni avansate care să îi ajute în analizarea progresului clienților proprii, fără

însă a pierde din vedere obiectivul de a obține un design simplu, care să le ușureze munca. Astfel se vor introduce diagrame și grafice pentru a reprezenta date care doar textual sau prin reprezentare tabelară ar putea fi prea greu de interpretat de către unii utilizatori.

Al **doilea obiectiv** secundar este de a permite utilizatorului să aleagă între cel puțin două modalități pentru calcularea caloriilor și macronutrienților necesari într-o zi pentru client. Există mai multe ecuații pentru obținerea acestor valori, fiecare luând în calcul anumite date. Majoritatea folosesc date care se pot obține ușor, cum ar fi înălțimea, greutatea sau vârsta, însă există ecuații care se consideră a oferi valori mai precise care au nevoie de procentul de grăsime și activitatea fizică. Cum procentul de grăsime corporală se obține folosind un dispozitiv special, un aspect care trebuie luat în calcul este oferirea posibilității clientului de a obține datele necesare pentru a le introduce în aplicație în vederea calculării valorilor. Luând toate acestea în calcul, este necesară alegerea a cel puțin două ecuații, una care folosește primul grup de date și alta care folosește al doilea.

Al **treilea obiectiv** secundar este cel de a evita un design aglomerat care poate da impresia unei aplicații prea complexe pentru unii utilizatori noi sau poate îngreuna utilizarea acesteia. Aplicațiile care încearcă să cuprindă atât domeniul nutriției cât și al fitness-ului și să aibă multe funcționalități sunt întâmpinate de un dezavantaj, și anume obținerea unei platforme care este complexă din punct de vedere al utilizării sau care are un design înghesuit. Astfel va trebui să se urmărească obținerea unui design plăcut, aerisit, în care utilizatorul poate accesa ușor datele de care are nevoie și care nu conține operații care necesită efectuarea unui număr mare de pași. Prin îndeplinirea acestui obiectiv, utilizatorii noi nu vor fi copleșiți când folosesc pentru prima dată aplicația și le va fi mai ușor să se obișnuiască cu aceasta.

Un alt **obiectiv** important este portabilitatea. În acest sens se va alege proiectarea platformei ca aplicație web, în loc de desktop sau mobile. Aplicația va putea însă să fie utilizată și de pe telefoane inteligente sau de pe tablete, prin intermediul unui browser, permițând utilizarea acesteia în deplasare. Datorită unui ecran mai mare, acesta va permite totodată aranjarea ușoară elementelor pe pagină pentru a obține un efect aerisit, cu o dimensiune suficient de mare pentru o vizibilitate bună. O aranjare ușoară rezultă în obținerea unui design care să ofere un acces rapid la orice funcționalitate.

# Capitolul 3. Studiu Bibliografic

În prima parte a acestui capitol se va prezenta o analiză a eficienței utilizării aplicațiilor în monitorizarea dietei și a activităților fizice zilnice și de ce sunt mai recomandate decât monitorizarea clasică, urmat de o descriere a tipurilor de aplicație din domeniile pe care proiectul curent dorește să le integreze într-una singură. În a doua parte se vor prezenta aplicații similare și se va realiza o analiză a funcționalităților a acestora, puse în comparație cu platforma propusă în acest proiect. Scopul acestei analize este de a identifica caracteristicile esențiale care stau la baza platformei propuse.

## 3.1. Aplicațiile în monitorizarea dietei și activității fizice

O aplicație de nutriție sau fitness este o aplicație care poate fi descărcată pe orice telefon mobil, în cazul aplicațiilor mobile, sau pot fi utilizate de pe un browser web, în cazul aplicațiilor web. Acestea pot fi utilizate de oriunde de către utilizator pentru a-și urmări progresul. Aceste aplicații pot realiza diverse funcții, cum ar fi permiterea utilizatorilor de a-și stabili obiectivele de fitness, urmărirea aportului caloric, căutarea unor noi idei de antrenament. Acestea pot fi utilizate pentru promovarea schimbării în direcția unui comportament sănătos, cu antrenamente personalizate, sfaturi de fitness, planuri de nutriție și posibilitatea distribuirii progresului pe rețelele de socializare. Unele aplicații pot funționa împreună cu dispozitive ce pot fi purtate accesând datele de pe dispozitiv, cum ar fi un dispozitiv ce urmărește pașii făcuți de-a lungul zilei. Alte aplicații utilizează elemente de joc și crează competiții între utilizatori pentru a-i ajuta să fie mai motivați.

Au fost identificate două categorii de resurse care intră în aria de interes a scopului acestui proiect: aplicații în fitness și aplicația în nutriție.

O aplicație pentru nutriție este o aplicație care oferă o bază de date cu alimente și produse, în vederea urmăririi alimentației. Unele pot oferi și posibilitatea urmăririi consumului de apă și a adăugării de rețete.

Aplicația în fitness oferă o bază de date care poate să conțină strict doar exerciții fizice sau și alimente, produse și rețete, pentru a acoperi urmărirea unui stil de viață sănătos din toate punctele de vedere. Tot în aceste aplicații se pot adăuga și împărtășirea progresului cu alți utilizatori și integrarea anumitor dispozitive.

Monitorizarea personală constă în monitorizarea dietei și a activității fizice astfel încât persoana care realizează această sarcină să fie conștientă de comportamentele actuale. Prin această metodă, în atingerea unui obiectiv cum ar fi pierderea în greutate, persoana poate vizualiza exact cantitatea și ceea ce consumă și durata și calitatea antrenamentului fizic, ceea ce ajută în a determina un motiv explicit pentru care obiectivul nu a fost atins. Prin identificarea problemelor din analiza înregistrărilor din monitorizare, mai departe se pot căuta soluții pentru a îndrepta aceste probleme.

Pentru a stabili dacă monitorizarea este întradevăr eficientă, s-au consultat mai multe studii, unul dintre acestea fiind [4], în care s-a descoperit faptul că persoanele cu înregistrări de monitorizare considerate a fi cele mai complete au pierdut semnificativ

mai multă greutate decât cele care au avut înregistrări mai puțin complete, iar pierderea în greutate a fost mai mare în săptămânile cu un grad mai mare de monitorizare.

Deși monitorizarea tradițională, pe hârtie, dă rezultate, în ultimii ani aplicațiile realizate pentru acest scop au câștigat popularitate fiind adoptate tot mai mult. Pentru a determina dacă într-adevăr astfel de aplicații sunt mai eficiente s-au consultat mai multe studii. În unul dintre acestea, publicat în The Journal of the American Medical Informatics Association [5], dintre personele care au participat la studiu, cele care au folosit o aplicație pentru a-și monitoriza dieta și activitatea fizică au exersat mai frecvent și, deși frecvența în monitorizarea dietei nu a diferit față de cei care au monitorizat pe hârtie, cantitatea de calorii consumată a fost mai redusă.

Alte motive pentru care aplicațiile din aceste categorii par mai eficiente pot fi faptul că acestea oferă mai multe date vizuale și feedback care pot motiva persoanele în a rămâne consecvente. De asemenea, timpul necesar pentru a introduce datele într-o aplicație poate fi mai rapid, datorită salvării datelor alimentelor pentru introducerea lor imediată și/sau citirea codului de bare. În plus, multe dintre calcule, cum ar fi totalul de calorii și macro consumați și cantitatea rămasă care mai trebuie consumată, sunt realizate automat de către aplicații, scutind astfel complet persoanele de acest lucru.

În concluzie, utilizarea unei aplicații pentru a monitoriza dieta și activitatea fizică este utilă. Șansele ca persoanele să rămână consecvente în menținerea monitorizării sunt mai mare dacă se folosește o aplicație.

## 3.2. Sisteme similare

Deoarece pe piață se află zeci de astfel de aplicații, s-au ales pentru comparație cele mai utilizate în acest moment: MyFitnessPal, Fitocracy și FitnessBliss. Fiecare are o anumită caracteristică din genul acestor aplicații, prin care se caracterizează. MuFitnessPal dispune de cea mai mare bază de date, având peste 6 milioane de alimente și produse alimentare, cu informații foarte detaliate despre conținutul acestora, de la numărul de calorii, la proteine și vitamine. Un lucru care a reușit să se deosebească de celelalte aplicații este posibilitatea de a scana codul de bare al unui produs, pentru a-i obține datele și a-l salva în baza de date. Fitocracy se caracterizează prin elementele de joc pe care le are introduse, transformând pentru utilizatori urmărirea dietei și antrenamentul într-un joc sau o competiție între alți utilizatori, un lucru care pe unii îi poate motiva să rămână consecvenți. FitnessBliss se concentrează pe antrenament, având o bază de date cu exerciții variate, grupate pe categorii de grupe de mușchi și echipament.

### *3.2.1. MyFitnessPal*[1]

Este o aplicație mobilă și web care urmărește dieta și exercițiile pentru a determina aportul caloric optim și nutrienții pentru obiectivele utilizatorilor, fiind cea mai utilizată aplicație de acest gen. MyFitnessPal oferă și o versiune cu plată care oferă posibilitatea reglării cantității de macro necesar zilnic, vizualizarea colesterolului. De asemenea aplicația afișează patru grafuri și profiluri, unul cu macronutrienți, care sunt

[1] MyFitnessPal: https://play.google.com/store/apps/details?id=com.myfitnesspal.android&hl=en

grăsimea, carbohidrații, proteinele și caloriile, un profil cu sănătatea inimii, care cuprinde grăsimea, sodiumul, colesterolul și caloriile, un graf cu carbohidrați, zaharuri, fibre și calorii, sau un rezumat particularizat în care se pot alege trei elemente nutritive.

### *3.2.2. Fitocracy*[2]

Este un joc online și o rețea socială care urmărește utilizarea gamificării pentru a ajuta utilizatorii să-și îmbunătățească condiția fizică. Utilizatorii trebuie să își înregistreze activitatea de exercițiu prin selectarea dintr-o colecție de activități cum ar fi obținerea în greutate și alergarea. Punctele sunt acordate pe baza beneficiului estimat al activității. Utilizatorii trebuie să atingă un anumit număr de puncte pentru a-și ridica nivelul.

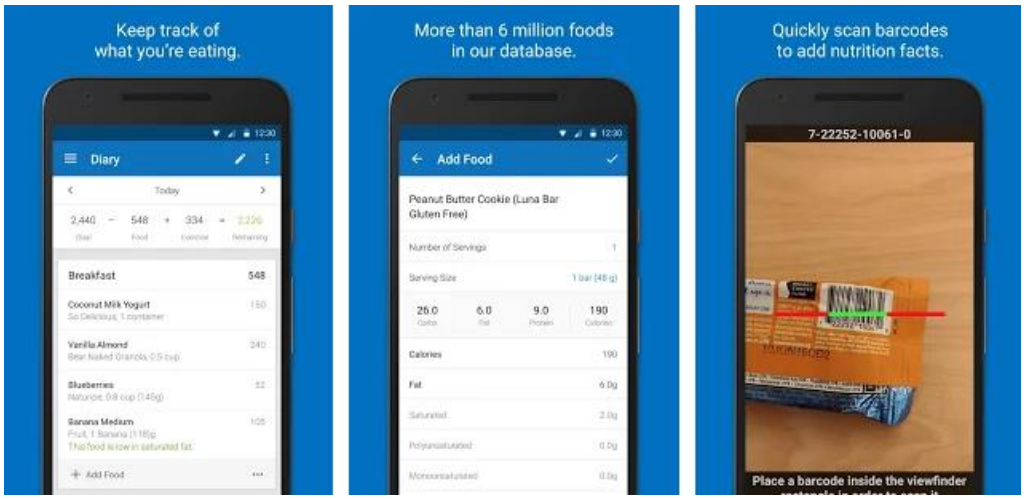



Figura 3.1 Interfața MyFitnessPal

Aplicația oferă utilizatorilor provocări pentru a obține puncte suplimentare, care constau dintr-un set de exerciții. Anumite etape mai importante sunt recompensate cu ecusoane odată ce sunt atinse. Aplicația oferă, de asemenea, o rețea de socializare care permite utilizatorilor să urmărească alți utilizatori, să vizualizeze și să comenteze antrenamentele lor și să se înscrie în grupuri specifice intereselor proprii.

Fitocracy oferă și o versiune cu plată care oferă pe lângă funcțiile de mai sus, un antrenor personal de care aplicația dispune.

[2] Fitocracy: https://www.fitocracy.com/

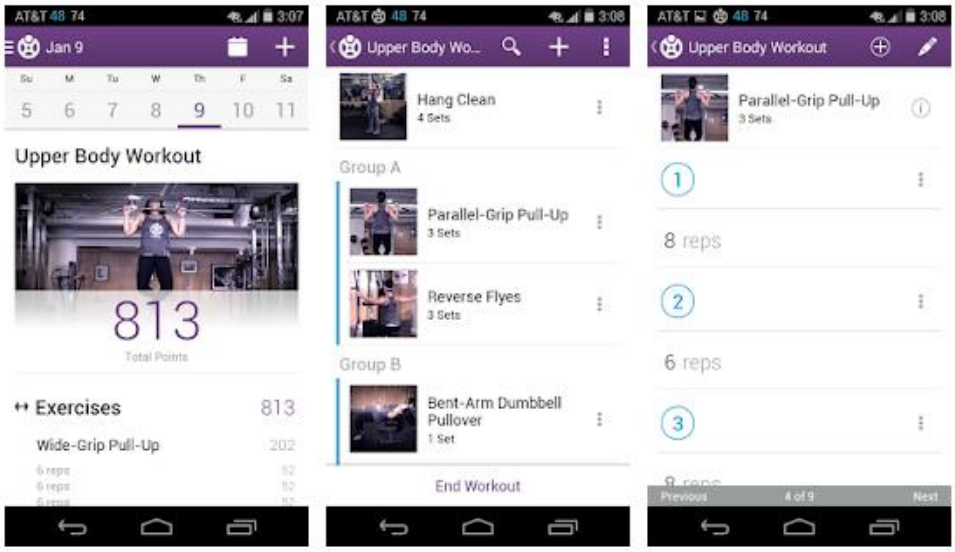



Figura 3.2 Interfața Fitocracy

### *3.2.3. FitnessBliss*[3]

FitnessBliss este o aplicație web care oferă acces la o listă diversă de exerciții, fiecare având atașat o ilustrație animată sau un video care arată modul corect în care trebuie executat exercițiul. Aplicația mai oferă crearea, printarea și urmărirea de grafuri cu rutinele de antrenament, dar și vizualizarea unui istoric al greutății ridicate. Pentru a găsi exercițiile dorite mai eficient, aplicația dispune de o filtrare aplicată pe grupa de mușchi pe care este activată și echipamentul utilizat.

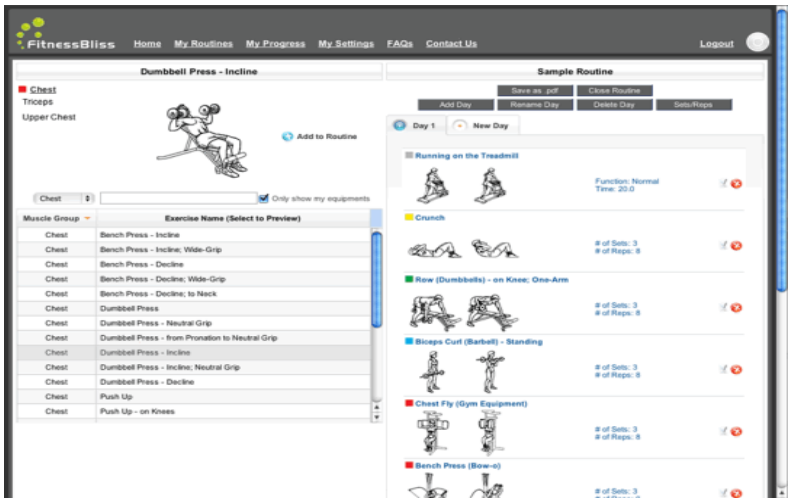



Figura 3.3 Interfața FitnessBliss

[3] FitnessBliss: https://www.fitnessbliss.com/en/

În urma acestor analize s-a urmărit realizarea unei aplicații similare care să conțină de asemenea un element principal în jurul căruia să se construiască restul funcțiilor, și anume comunicarea și colaborarea cu antrenori și nutriționiști.

Mai jos s-a realizat o comparație, din perspectiva funcționalităților oferite pentru utilizator, între aplicațiile prezentate mai sus. Obiectivul a fost determinarea funcționalităților care sunt cuprinse în cel puțin două aplicații care ar indica necesitatea lor și în sistemul propus, dar și a funcționalităților care nu se regăsesc în nici una dintre aceste aplicații și care ar putea fi încadrate în sistemul propus.

În tabelul 3.1 sunt comparate funcționalitățile observate din punct de vedere al monitorizării. În tabelul 3.2 sunt comparate funcționalitățile observate din punct de vedere al datelor al datelor oferite despre conținutul alimentelor. În tabelul 3.3 sunt comparate funcționalitățile oferite pentru monitorizarea exercițiilor fizice.

Tabel 3.1 Comparare funcționalități pentru monitorizarea alimentației

| **funcționalități /nume platformă** | **MyFitnessPal** | **Fitocracy** | **FitnessBliss** | **Sistemul propus** |
|---|---|---|---|---|
| date cu alimente și produse de consum | ✓ | ✗ | ✗ | ✓ |
| importare /creare rețete | ✓ | ✗ | ✗ | ✓ |
| înregistrarea unui produs prin scanarea codului de bare | ✓ | ✗ | ✗ | ✗ |
| calculator de calorii pe baza datelor personale | ✓ | ✗ | ✗ | ✓ |
| jurnal prin împărțirea pe mesele principale | ✓ | ✗ | ✗ | ✓ |
| urmărirea consumului de apă | ✓ | ✗ | ✗ | ✗ |
| setarea unui obiectiv | ✓ | ✗ | ✗ | ✓ |
| comunicare cu alți utilizatori | ✓ | ✓ | ✓ | ✗ |
| vizualizare progres prin grafuri | ✓ | ✗ | ✗ | ✓ |
| elemente de joc | ✗ | ✓ | ✗ | ✗ |
| exportare date | ✗ | ✓ | ✗ | ✓ |
| nutriționist /dietetician | ✗ | ✗ | ✗ | ✓ |

Tabel 3.2 Compararea datelor oferite pentru alimente

| **funcționalități nutriție /nume platformă** | **MyFitnessPal** | **Fitocracy** | **FitnessBliss** | **Sistemul propus** |
|---|---|---|---|---|
| calorii | ✓ | ✗ | ✗ | ✓ |
| macronutrienți | ✓ | ✗ | ✗ | ✓ |
| nutrienți | ✓ | ✗ | ✗ | ✓ |
| alergeni | ✗ | ✗ | ✗ | ✓ |
| antrenor | ✗ | ✓ | ✓ | ✓ |

Tabel 3.3 Comparare funcționalități pentru monitorizarea antrenamentului

| **funcționalități antrenament /nume platformă** | **MyFitnessPal** | **Fitocracy** | **FitnessBliss** | **Sistemul propus** |
|---|---|---|---|---|
| atașări imagini/video | ✗ | ✓ | ✓ | ✓ |
| creare exerciții | ✓ | ✓ | ✓ | ✓ |
| grupare pe grupe de mușchi | ✗ | ✗ | ✓ | ✓ |
| îndrumare exerciții | ✗ | ✗ | ✗ | ✓ |
| înregistrare pași | ✓ | ✗ | ✗ | ✗ |

Din ceea ce s-a putut observa, toate aplicațiile analizate au un singur tip de utilizator și anume cel care utilizează platforma în scopul urmăririi alimentației și/sau al antrenamentului. Antrenorii care pot fi oferiți sunt colaboratori ai aplicațiilor. Acest lucru duce la anumite constrângeri pentru utilizator. În primul rând dacă un utilizator are deja un antrenor cu care ar dori să poată colabora la distanță, acesta nu va putea utiliza platforma în acest scop, ci va trebui să aleagă un antrenor oferit de aplicație. Al doilea dezavantaj este lipsa unui nutriționist. Nici o aplicație nu oferă un nutriționist. Acestea menționează faptul că antrenorul poate să ofere și un plan de dietă personalizat, pe lângă exerciții. Deși antrenorii au cunoștințe de bază în nutriție, acestea pot să nu fie suficiente încât să asiste un client care de exemplu are probleme de alimentație.

Aplicația își propune acoperirea acestei funcționalități prin adăugarea unui nou tip de utilizator care să cuprindă antrenorul și nutriționistul. În loc să se creeze câte un tip de utilizator pentru fiecare rol, s-a ales unificarea acestora, aceștia având funcții comune.

# Capitolul 4. Analiză şi Fundamentare Teoretică

## 4.1. Calcularea caloriilor și macronutrienților

Un lucru necesar este calcularea numărului de calorii și macronutrienți necesari într-o zi, pe baza datelor personale. În prezent există multe ecuații pentru acest scop, cele mai cunoscute fiind Ecuația Harris-Benedic, Ecuația Organizației Mondiale a Sănătății și Ecuația Mifflin-M.D.-St. Jeor. Dintre toate acestea Mifflin-M.D.-St. Jeor este recomandată de profesioniștii din domeniul nutriției și s-a constatat că este cea mai fiabilă în estimarea energiei consumate [6].

Ecuațiile menționate mai devreme folosesc ca date masa corporală, înălțimea, vârsta, genul și activitatea fizică. În cazul tuturor ecuațiilor care folosesc aceste date, rezultatele pentru persoanele obeze sau foarte slabe nu sunt corecte. Persoanele foarte slabe au puțină grăsime și mult țesut muscular care arde mai multe calorii chiar și în repaus. Persoanele obeze, pe de altă parte, au mai multă grăsime și puțin țesut activ, necesitând foarte puțină energie pentru a se menține. Astfel pentru aceste tipuri de persoane greutatea totală a grăsimii trebuie luată în considerare în acuație.[7] Acest lucru o face ecuația Katch-McArdle.

Cum poate nu toate persoanele doresc să folosească greutatea totală a grăsimii, se vor introduce două ecuații în sistem, Mifflin-M.D.-St. Jeor, implicit, și Katch-McArdle dacă între timp se dorește o precizie mai bună.

În calcularea numărului de calorii și macronutrienți se va folosi implicit ecuația Mifflin-M.D.-St. Jeor. Primul pas constă în calcularea energiei consumate pasiv (Resting Energy Ependiture – REE), pe baza următoarei ecuații:

$$P = \left( \frac{10.0m}{1\ \text{kg}} + \frac{6.25h}{1\ \text{cm}} - \frac{5.0a}{1\ \text{year}} + s \right) \frac{\text{kcal}}{\text{day}}$$

Figura 4.1 Ecuația Mifflin-M.D.-St. Jeor

(m=masă, h=înălțime, a=vârstă, s=+5 pentru femei și -161 pentru bărbați)

Mai departe trebuie calculată energia consumată totală, care cuprinde și energia consumată când persoana este activă (TDEE). Rezultatul se obține înmulțind REE cu un factor care depinde de tipul de activitate:

- sedentar: REE x 1.2
- activitate ușoară: REE x 1.375
- activitate moderată: REE x 1.55
- foarte activ: REE x 1.725

Ultimul pas este luarea în considerare a scopului. Dacă se dorește menținerea greutății, numărul de calorii va fi exact TDEE. Dacă se dorește pierderea în greutate, se scade din TDEE cel mult 20%. Dacă se dorește creșterea în greutate, se adaugă cel mult 20%:

- menținere: TDDE = TDDE

- pierdere în masă: TDDE = TDEE – (TDDE x .20)
- câștig de masă: TDDE = TDEE + (TDDE x .20)

Pentru aplicarea ecuației Katch-McArdle este nevoie de doi factori, masa corporală musculară (Lean Body Mass) și Rata Metabolică Bazală (BMR), care este înlocuitorul lui TDEE pasiv din Mifflin-M.D.-St. Jeor:

Lean Body Mass = Body Weight - (Body Weight * Body Fat %)

BMR = 370 + (21.6 x Lean Body Mass(kg) )

Cum nu toți utilizatorii au aparatura necesara pentru a-și calcula procentajul de grăsime, se adaugă opțiunea de a o calcula pe baza ecuațiilor:

lean body mass (bărbați) = (masa * 1.082) + 94.42 – talia * 4.15

lean body mass (femei) = (masa * 0.732) + 8.987 + încheietura / 3.140 - talia * 0.157 – șolduri * 0.249 + antebraț * 0.434

masa de grăsime = masa – lean body mass

procentajul de grăsime (BFP) = masa de grăsime / masa

Toate aceste formule și detalii despre acestea se pot găsi pe omni CALCULATOR[4].

Mai departe se calculează TDEE bazat pe activitate și pe scop, exact ca și în cazul Mifflin-M.D.-St. Jeor.

Macronutrienții se determină la fel în ambele cazuri.

Pentru proteine, recomandarea actuală de către Recommended Dietary Allowance este de 0.8g per kg [8].

Pentru grăsimi sunt diferite opinii, variind între 20% și 30% din totalul de calorii. Se alege astfel valoarea cea mai mică, 20%.

Cantitatea de carbohidrați reprezintă restul rămas din calorii, după scăderea numărului de calorii din totalul de proteine și de grăsimi. Cum 1g de proteine, respectiv carbohidrați are 4 calorii și 1 gram de grăsime are 9 calorii, se obține:

carbohidrați (g) = [TDEE – 4 x proteine(g) - 9 x grăsimi(g)] / 4

## 4.2. Determinarea caloriilor arse

Pentru a determina totalul de calorii arse se vor folosi două metode, una care să folosească ritmul cardiac și alata care să nu necesite această informație.

O metodă pentru a estima cât mai precis câte calorii sunt arse pe perioada unei activități fizice este folosirea unei formule publicată în Journal of Sports Sciences[5]. Acest jurnal oferă o formulă pentru fiecare gen:

barbați:

Calories Burned = [(Age x 0.2017) - (Weight x 0.09036) + (Heart Rate x 0.6309) - 55.0969] x Time / 4.184.

femei:

[4] omni CALCULATOR: How to calculate body fat (https://www.omnicalculator.com/health/body-fat#how-to-calculate-body-fat)

[5] Journal of Sports Sciences: Prediction of Energy Expenditure from Heart Rate Monitoring During Submaximal Exercise (http://www.braydenwm.com/cal_vs_hr_ref_paper.pdf)

Calories Burned = [(Age x 0.074) - (Weight x 0.05741) + (Heart Rate x 0.4472) - 20.4022] x Time / 4.184.

Pentru această formulă trebuie să se obțină ritmul cardiac. Acesta se obține tot prin folosirea unui monitor ritm cardiac care, cum s-a menționat mai sus, poate să nu fie mereu foarte precis. O altă problemă care poate apărea este lipsa unui astfel de dispozitiv. Astfel, pentru a nu forța utilizatorul să depindă de un astfel de dispozitiv, platforma va oferi posibilitatea de a decide o a doua metodă de determinare a caloriilor arse.

Deși pentru determinarea caloriilor arse pe perioada unui exercițiu cea mai cunoscută metodă este prin folosirea unui monitor ritm cardiac, uneori rezultatele oferite de acesta nu sunt precise. O problemă este faptul că partea din spate a ceasului sau a dispozitivului de transmisie trebuie să rămână în contact permanent cu pielea de pe încheietura mâinii sau a pieptului. Dacă nu face contact, nu se calculează. Contactul poate fi întrerupt, iar monitorizarea oprită atunci când persoana transpiră, din cauza pierderii contactului. Astfel, numărul caloriilor arse poate fi puțin mai mic decât ceea ce s-ar calcula manual.

A doua metodă pentru a estima câte calorii sunt arse pe perioada unei activități fizice este de a folosi MET (echivalent metabolic). Cercetătorii au atribuit valori MET pentru orice tip de activitate fizică, astfel nu este greu găsirea valorilor. Pentru acest proiect, valorile au fost luate de pe pagina oficială a profesorului Gary A. Chase, de la Universitatea Pacific Lutheran[6].

Calories Burned = [MET x 3.5 x Weigth (kg) / 200] * Time (minutes)

Când se monitorizează exercițiile fizice de forță, se notează mai multe date care nu sunt folosite direct pentru a obține alte valori, ci pentru a putea progresa și a putea observa în timp progresul. Acestea pot varia de la persoană la alta, însă cele mai comune sunt greutatea folosită, numărul de seturi și numărul de repetiții. Prin greutatea folosită se va putea vedea după o perioadă de timp dacă aceasta a crescut, ceea ce indică un progres, sau dacă a rămas constant, sau chiar a scăzut. Prin urmărirea numărului de seturi și repetiții se poate de asemenea urmări progresul: menținerea constantă sau scăderea numărului de seturi sau repetiții, fără a crește greutatea utilizată, poate indica un declin, precum nevoia reorganizării planului sau a introducerii unei perioade de recuperare.

Astfel aceste date vor putea fi introduse atunci când se introduce un exercițiu din această categorie în jurnal. Prin această opțiune, antrenorul va putea atunci când crează un plan de antrenament să specifice exact ce greutate va trebui clientul să folosească și câte seturi a câte repetiții trebuie să execute. Desigur, și un utilizator fără un antrenor poate să menționeze aceste date pentru monitorizarea mai precisă a progresului.

[6] Listă cu valori MET pentru diverse exerciții fizice (https://community.plu.edu/~chasega/met.html)

## 4.3. Determinarea actorilor

În monitorizarea nutriției și a exercițiilor fizice prin colaborare pot fi încadrați mai mulți specialiști, și anume nutriționistul, dieteticianul, antrenorul șiu fizioterapeutul. Pentru simplificare toți acești specialiști vor fi cuprinși în aplicație ca un singur tip de utilizator, numit coach. Chiar și așa aplicația tot va trebui să cuprindă toate funționalitățile de bază de care acești specialiști au nevoie pentru a-și putea asista clientul. Din acest motiv s-a analizat fiecare specialist în parte pentru a se determina punctele care le au în comun și care ar fi nevoile fiecăruia.

Un nutriționist este o persoană care deține un certificat în nutriție. Aceștia dețin cunoștințe în studiul alimentelor și știința nutriției, principalul lor rol fiind acela de a învăța clienții despre nutriția generală și de a oferi supraveghere în nutriția acestora. Ceea ce nu le este permis nutriționiștilor este diagnosticarea tulburărilor de alimentație sau ajutarea la planificarea meselor pentru gestionarea simptomelor de probleme de sănătate, aceste funcții revenindu-i dieteticianului.

Prin urmare, dieteticianul trebuie să aibă acces la lista personală a alimentelor a clientului și să poată să introducă noi alimente pe care clientul le poate consuma, dar și la jurnalul cu alimentele consumate și planul de nutriție și posibilitatea de a crea un plan de nutriție folosind alimentele din lista clientului.

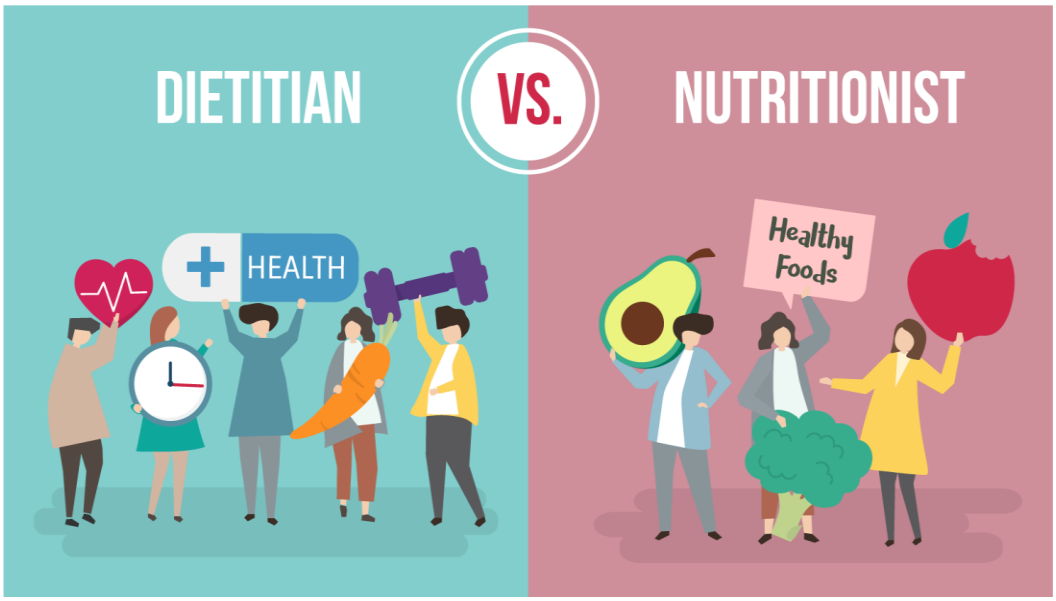



Figura 4.2 Dietetician vs nutriționist

În timp ce pentru a deveni un nutriționist există multe școli care oferă astfel de cursuri, pentru a putea deține titlul de dietetician trebuie urmată o școală specifică care oferă certificat de dietetician și să finalizeze un stagiu sau un program de practică în acest domeniu înainte ca aceștia să poată practica ca și dieteticieni calificați. Rolul lor este de a urmări prepararea și servirea mâncării, crearea unor diete noi, participarea și realizarea de studii și educarea persoanelor la obiceiuri nutriționale sănătoase. Datorită studiilor avansate, aceștia pot colabora cu medici pentru a oferi intervenții medicale nutriționale.

Deci, spre deosebire de nutriționist, rolul de dietetician este mai reglementat și orientat pe problemele de sănătate, permițându-i să lucreze inclusiv în spitale.

Cum în acest proiect accentul se pune pe menținerea unui stil de viață sănătos și foarte puțin pe problemele de sănătate, dieteticianul este integrat ca nutriționist.

În figura 4.2[7] este prezentată vizual diferența dintre dietetician și nutriționist.

Există diverse certificate pentru a putea deveni antrenor personal, fiecare având titluri diferite. Cele mai comune titluri sunt ‘antrenor personal’, ‘antrenor de fitness’ sau ‘instructor de fitness’. Cerințele care trebuie îndeplinite pentru a deveni un antrenor variază de la o țară la alta. În principiu trebuie terminată o școală sau un curs specializat în acest domeniu, iar la final este necesară trecerea unui examen pentru obținerea certificatului. Rolul unui antrenor de fitness este de a instrui și motiva o persoană sau grupuri de persoane, cu orice nivel de experiență, în activități fizice, care pot include activități cardiovasculare, de rezistență și de flexibilitate. Datorită studiilor parcurse aceștia au cunoștințe generale de alimentație și de fitness, pentru a putea prescrie și instrui exerciții, însă nu pot spre exemplu să asiste persoane care sunt accidentate sau au dizabilități care le împiedică mult efectuarea exercițiilor, acest rol revenindu-i fizioterapeutului.

Antrenorul va trebui să aibă poată în primul rând să creeze un plan de antrenament pentru client și să poată să îi vizualizeze progresul. De multe ori antrenorii, pe lângă planul de antrenament, le pot crea clienților și un plan de dietă sau le pot recomanda ce alimente ar trebui sau nu să consume pentru a-și atinge scopul, într-ucât cât timp alimentația nu este corectă rezultatele pot să nu se vadă. În plus, antrenorul trebuie să poată vedea jurnalul cu ceea ce a consumat clientul pentru a vedea dacă motivul pentru care nu se poate vedea nici un progres nu este datorat dietei pe care clientul o urmează. Astfel, antrenorul trebuie să aibă acces la funcționalitățile nutriționistului. În același mod, nutriționistul poate avea nevoie de funcționalitățile atribuite antrenorului, deoarece nutriționistul ar putea să îi recomande clientului exerciții fizice sau activități pe care acesta să le realizeze pentru a-i ușura atingerea scopului.

Așa cum la un nutriționist există și dieteticianul cu titlu cu calificare, și la antrenor este fizioterapeutul. Precum dieteticianul, acesta trebuie să termine cursuri de specializate și să treacă un examen. Rolul principal al unui fizioterapeut este de a ajuta pacienții cu probleme fizice rezultate din accidentări, dizabilități sau îmbătrânire să își recapete mobilitatea. Motivul pentru care fizioterapeutul nu este inclus în astfel de aplicați este datorat faptului că acesta trebuie să își asiste clientul față în față pentru a putea să-l îndrume în mișcări și pentru a se asigura în general că fiecare exercițiu se face într-o poziție cât mai corectă și în funcție de problemele clientului.

În figura 4.3[8] este prezentată vizual diferența între fizioterapeut și antrenor personal.

[7] https://eduadvisor.my/articles/major-differences-between-a-dietitian-and-a-nutritionist/
[8] http://social.asianphysiotherapy.com/insta/health-fitness-AsianPhysiotherapy-1dse0ghjpl5khby/

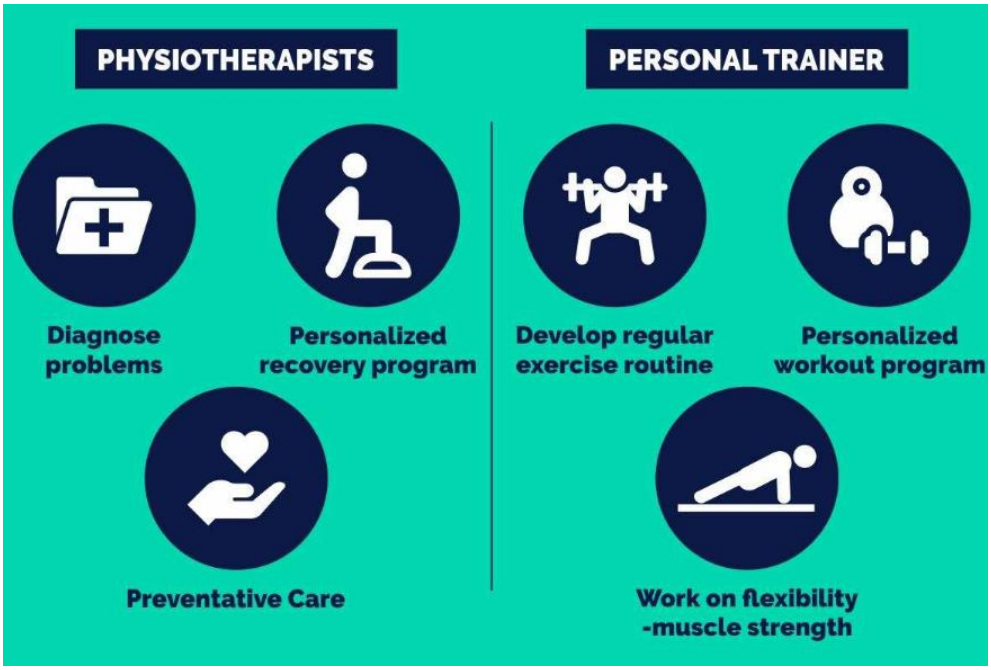



Figura 4.3 Antrenor vs fizioterapeut

## 4.4. Cazuri de utilizare

În figurile de mai jos sunt prezentate cazurile de utilizare pentru cele două tipuri de actori, client și antrenor/nutriționist, urmate de o descriere detaliată a cazurilor de utilizare principale identifcate în cadrul aplicației. Identificarea s-a realizat în funcție de gradul de complexitate al operației. Pentru a putea încadra toate cazurile de utilizare în figură, acestea au fost încadrate în forme dreptunghiulare în loc de forme ovale. De asemenea, operațiile comune, care se aplică atât clientului cât și antrenorului și nutriționistului, sunt introduse în ambele categorii de utilizatori deoarece aceste operații, deși la bază au același scop, diferă într-o anumită măsură între ele. De exemplu, pentru operația "înregistrare", datele care trebuie introduse diferă în funcție de tipul utilizatorului.

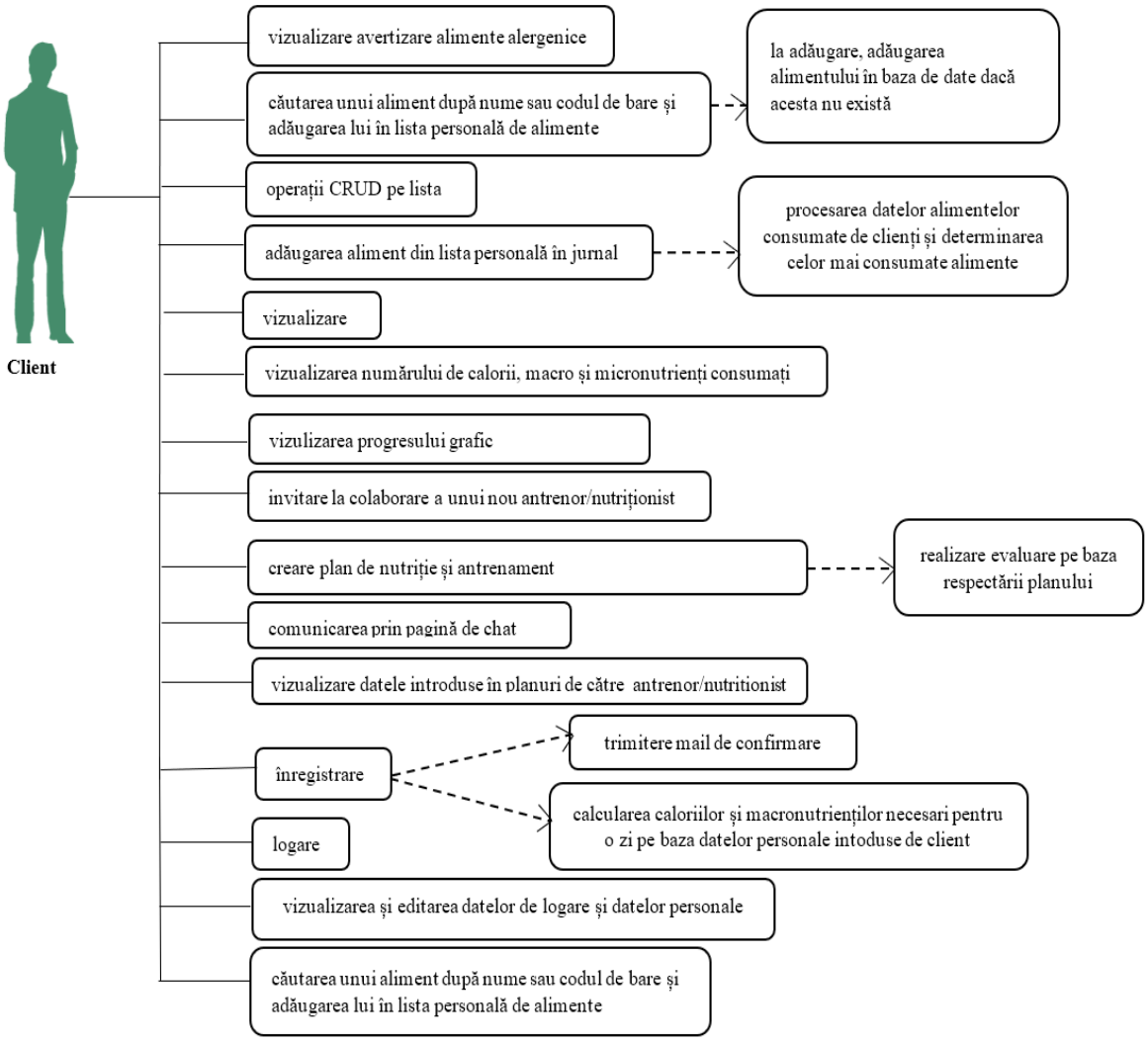



Figura 4.4 Cazuri de utilizare pentru Client

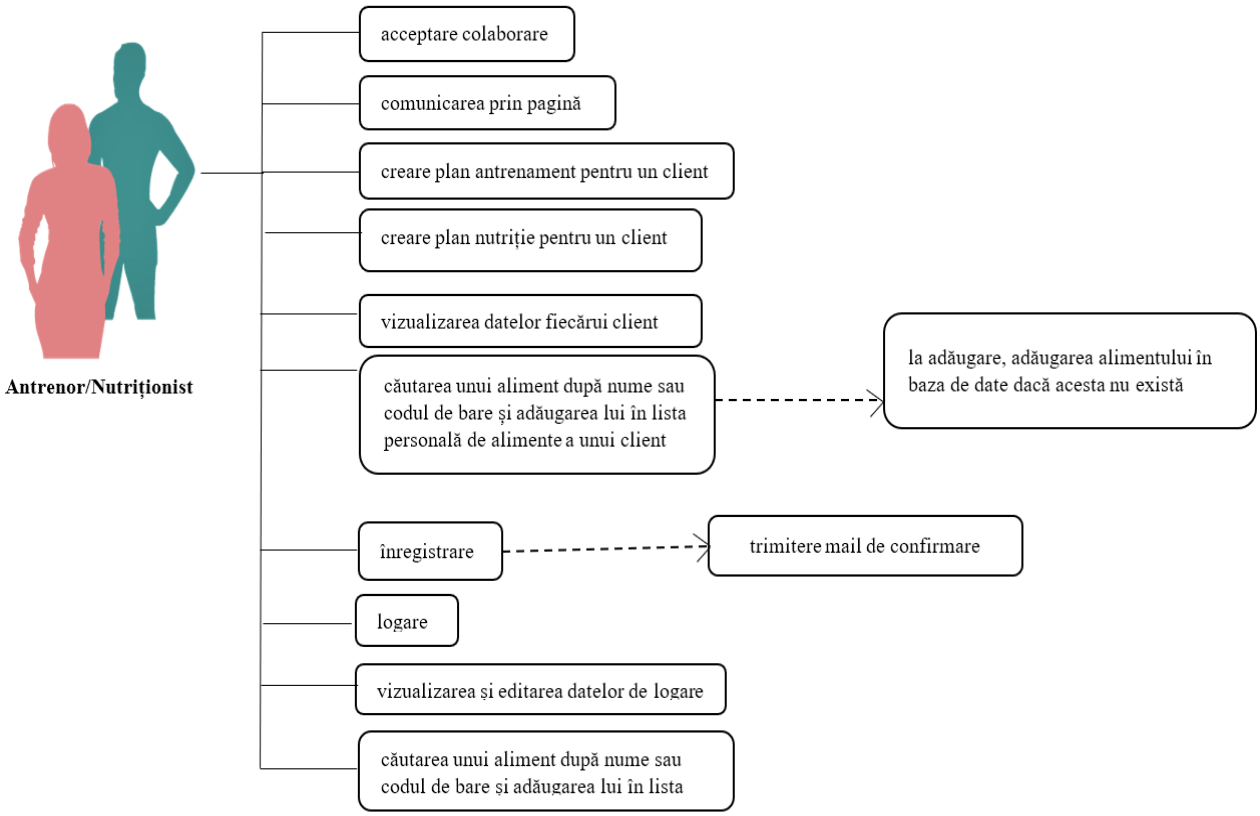



Figura 4.5 Cazuri de utilizare pentru Antrenor/Nutriționist

### *4.4.1. Adăugarea unui aliment nou în lista de alimente*

**Nume cazului de utilizare:** adăugarea aliment în lista personală de alimente
**Descriere:** Utilizatorul dorește să adauge un nou aliment în lista personală de alimente, în urma căutării lui în pagina Add Food.
**Actori:** client
**Precondiții:**

- Utilizatorul trebuie să fie conectat la internet;
- Utilizatorul trebuie să fie autentificat în aplicație ca și client;
- Utilizatorul trebuie să selecteze din lista de rezultate alimentul pe care dorește să-l adauge;
- Denumirea alimentului trebuie să aibă dimensiunea de maxim 100 de caractere și minim un caracter;
- Form-urile corespunzătoare valorilor nutriționale ale alimentului trebuie să permită introducerea doar a numerelor pozitive.

**Fluxul normal:**

1. Utilizatorul modifică datele alimentului care sunt editabile (nume, număr calorii, macronutrienți).
2. Utilizatorul apasă butonul „ADD”.

3. Sistemul validează datele introduse.
4. Sistemul verifică dacă alimentul este luat din baza de date.
5. Sistemul salvează alimentul în baza de date.
6. Utilizatorul este redirecționat în pagina Food List, pentru a vedea noul aliment adăugat în listă.

**Fluxul în caz de excepție:**

3.a. Numele alimentului nu este valid
3.a.i Se oprește fluxul.
3.a.ii Se afișează un mesaj de avertizare.
4.a. Alimentul se afă în baza de date
4.a.i Se continuă fluxul.
4.b. Alimentul nu se află în baza de date
4.b.i Sistemul adaugă alimentul în baza de date.
4.b.ii Se continuă fluxul.

**Postcondiții:**

- Alimentul este adăugat în baza de date;
- Dacă numărul de calorii și macronutrienți a alimentului nu au fost specificate, au valoarea setată la zero;
- Utilizatorul este redirecționat în pagina cu lista cu alimente personale, Food List;
- În pagina Food List se poate vedea noul aliment adăugat.

### *4.4.2. Adăugarea unui aliment nou în lista de alimente a unui client*

**Nume cazului de utilizare:** adăugarea aliment în lista personală de alimente a unui client
**Descriere:** Utilizatorul dorește să adauge un nou aliment în lista de alimente a unui client, în urma căutării lui în baza de date.
**Actori:** antrenor/nutriționist
**Precondiții:**

- Utilizatorul trebuie să fie conectat la internet;
- Utilizatorul trebuie să fie autentificat în aplicație ca și antrenor/nutriționist;
- Utilizatorul trebuie să selecteze clientul cui dorește să adauge alimentu;
- Utilizatorul trebuie să selecteze din lista de rezultate alimentul pe care dorește să-l adauge;
- Denumirea alimentului trebuie să aibă dimensiunea de maxim 100 de caractere și minim un caracter;
- Form-urile corespunzătoare valorilor nutriționale ale alimentului trebuie să permită introducerea doar a numerelor pozitive.

**Fluxul normal:**

1. Utilizatorul selectează clientul căruia dorește să-i adauge alimentul.
2. Utilizatorul selectează operația de adăugare a unui nou aliment.
3. Utilizatorul modifică datele alimentului care sunt editabile (nume, număr calorii, macronutrienți).
4. Utilizatorul apasă butonul „ADD”.
5. Sistemul validează datele introduse.

6. Sistemul verifică dacă alimentul este luat din baza de date.
7. Sistemul salvează alimentul în baza de date.
8. Utilizatorul este redirecționat în pagina Food List, pentru a vedea noul aliment adăugat în listă.

**Fluxul în caz de excepție:**

5.a. Numele alimentului nu este valid
5.a.i Se oprește fluxul.
5.a.ii Se afișează un mesaj de avertizare.
6.a. Alimentul se afă în baza de date
6.a.i Se continuă fluxul.
6.b. Alimentul nu se află în baza de date
6.b.i Sistemul adaugă alimentul în baza de date.
6.b.ii Se continuă fluxul.

**Postcondiții:**

- Alimentul este adăugat în baza de date;
- Dacă numărul de calorii și macronutrienți a alimentului nu au fost specificate, au valoarea setată la zero;
- Utilizatorul este redirecționat în pagina cu lista cu alimente ale clientului, Food List;
- În pagina Food List se poate vedea noul aliment adăugat.

### *4.4.3. Evaluarea consumului*

În acest caz există două fluxuri, primul realizat de subsistemul principal, la selectarea clientului de a trimite datele din planul nutrițional de pe ziua curentă în jurnal și al doilea realizat de subsistemul secundar, Consumer, care prelucrează datele primite de la primul subsistem.

**Nume cazului de utilizare:** trimiterea planului nutrițional în jurnal
**Descriere:** Utilizatorul trimite datele din planul nutrițional de pe ziua curentă, din pagina Meal Plan, în jurnal, în urma căruia datele sunt trimise și la subsistemul secundar, Consumer, pentru a fi pelucrate pentru evaluarea clientului.
**Actori:** client și subsistemul principal, Sistem
**Precondiții:**

- Utilizatorul trebuie să fie conectat la internet;
- Utilizatorul trebuie să fie autentificat în aplicație ca și client;
- Data aleasă petnru a vizualiza planul nutrițional trebuie săfie data curentă, altfel nu trebuie să fie afișat butonul de trimitere a datelor în jurnal.

**Fluxul normal:**

1. Utilizatorul apasă butonul „Add To Diary”.
2. Sistemul extrage din baza de date intrările din planul nutrițional de pe ziua curentă.
3. Sistemul ia pe rând fiecare intrare.
   3.a. Verifică pentru fiecare intrare dacă este setată ca fiind consumată.
   3.a.i alimentul este consumat

Alimentul este salvat în jurnal.

3.a.ii alimentul nu este consumat

Se continuă la următorul pas.

3.b. Șterge din baza de date intrarea din planul nutrițional.

4. Sistemul trimite lista cu intrări spre al doilea subsistem
5. Utilizatorul este redirecționat în pagina Food List, pentru a vedea noul aliment adăugat în listă.

**Fluxul în caz de excepție:**

**Postcondiții:**

- Alimentele setate ca fiind consumate sunt adăugate în jurnal, în baza de date, și sunt vizibile în pagina Diary;
- Toate datele din planul nutrițional de pe ziua curentă sunt șterse, atât din baza de date cât și din pagina Meal Plan;
- Utilizatorul este redirecționat în pagina cu jurnal, Diary.

**Nume cazului de utilizare:** evaluarea consumului

**Descriere:** Subsistemul secundar, Consumer, ia din coadă datele trimise de la subsistemul principal și le prelucrează pentru evaluarea clientului.

**Actori:** subsistemul Consumer

**Precondiții:**

- Consumer ascultă coada pentru mesaje.

**Fluxul normal:**

7. Consumer detectează un mesaj pus în coadă.
8. Consumer ia mesajul și extrage lista din mesaj.
9. Consumer verifică dimensiunea listei

3.a. lista nu este nulă

3.a.i. Consumer calculează suma totală de calorii și macro, a tuturor intrărilor din listă, și suma celor care sunt marcate ca fiind consumate.

3.a.ii. Consumer caută în baza de date dacă există o intrare pentru evaluare pe ziua curentă

3.a.ii.1 Este găsită o intrare.

Se însumează sumele la datele existente găsite.

3.a.ii.2 Nu este găsită o intrare.

Se crează o nouă evaluare, unde se pun sumele obținute.

3.a.iii Consumer salvează evaluarea în baza de date.

3.b. lista este nulă

3.b.i Nu se face nimic.

**Fluxul în caz de excepție:**

**Postcondiții:**

- În baza de date se află o nouă intrare pentru evaluare pentru ziua curentă, dacă nu a existat, cu sumele obținute de Consumer sau se află cea existentă cu sumele însumate valorilor inițiale.

## 4.5. Cerințe funcționale

Mai jos sunt prezentate toate cerințele funționale ale platformei, grupate pe actori și pagina în care se află fiecare. În tabelul 4.1 se află cerințele funcționale ale clientului, iar în tabelul 4.2 cerințele funcționale ale antrenorului și nutriționistului. În tabelul 4.3 sunt prezentate cerințele funcționale ale subsistemelor care au loc în urma acțiunilor actorilor.

Tabel 4.1 Cerințe funcționale pentru client

| Actor | Acțiune | Pagina |
|---|---|---|
| **Client** | Înregistrare în aplicație ca și client, prin confirmare email. | LogIn |
| | Introducerea datelor personale la momentul înregistrării. | |
| | Intrarea în aplicație prin email și parolă. | |
| | Vizualizarea caloriilor și macronutrienților necesari pentru o zi. | Profile |
| | Editarea datelor de logare, datelor personale și a caloriilor și macronutrienților. | |
| | Selectarea între două formule pentru calculele realizate de aplicație. | |
| | Calcularea procentajului de grăsime pe baza unei formule. | |
| | Selectarea sistemului de măsurare. | |
| | Căutarea unui aliment după nume sau codul de bare. | Add Food |
| | Vizualizarea datelor alimentului căutat, modificarea lor și salvarea alimentului în lista personală de alimente . | |
| | Avertizarea clientului asupra alimentelor la care este alergic. | |
| | Adăugarea unui nou aliment, care nu există în baza de date. | |
| | Vizualizarea alimentelor salvate în lista personală. | Food List |
| | Vizualizarea, pentru fiecare aliment salvat în lista personală, a caloriilor, macronutrienților, valorilor recomandate de fibre, sodiu și zahăr, pentru o cantitate introdusă manual de la tastatură. | |
| | Adăugarea unui aliment din lista personală în Diay. | |
| | Ștergerea unui aliment din lista personală. | |
| | Vizualizarea alimentelor consumate pe o zi, cu totalul de calorii, macro, fibre, sodiu și zahăr, organizat pe mese. | Diary |
| | Vizualizarea numărului de calorii, macro și micronutrienți consumați din totalul necesar zilnic, grafic și prin procentaj. | |
| | Vizulizarea progresului într-un linechart, pe o perioadă selectată, cu posibilitatea modificării formatului datei. | Progress |
| | Crearea unui plan de nutriție pentru o zi. | Meal Plan |
| | Vizualizarea planurilor pentru fiecare zi. | |
| | Bifarea alimentelor consumate din plan. | |
| | Adăugarea alimentelor consumate din planul zilei curente în Diary. | |
| | Căutarea unui exercițiu după nume. | Exercises |
| | Vizualizara datelor unui exercițiu. | |
| | Adăugarea unui exercițiu la exerciții efectuate. | |
| | Vizualizarea planului de antrenament creat de A/N. | |
| | Vizualizarea exercițiilor efectuate. | Diary |
| | Introducerea greutății pentru ziua curentă. | Weight |
| | Vizualizarea progresului într-un linechart, pe o perioadă selectată. | |
| | Adăugarea unui A/N la colaborare. | Collaborations |
| | Comunicarea cu A/N într-o pagină de chat. | Chat |

Notare: Antrenor/Nutriționist = A/N

Tabel 4.2 Cerințe funcționale ale antrenorului/nutriționistului

| Actor | Acțiune | Pagina |
|---|---|---|
| **A/N** | Comunicarea cu clienții. | Chat |
| | Creare plan antrenament pentru un client. | Create Plan |
| | Creare plan nutriție pentru un client. | |
| | Vizualizarea progresului fiecărui client. | Progress |

Tabel 4.3 Cerințe funcționale ale subsistemelor

| | Acțiune |
|---|---|
| **Subsistem** | Trimitere email de confirmare a parolei. |
| | Calcularea caloriilor și macronutrienților necesari pentru o zi pe baza datelor personale intoduse de client la înregistrare. |
| | Adăugarea alimentului dacă acesta nu există în baza de date. |
| **Subistem client** | Trimiterea unui mail cu o evaluare a respectării planului |
| | Procesarea datelor alimentelor consumate de clienți pentru a determina cele mai consumate alimente. |
| | Recomandarea clientului, pe baza evaluarii, a alimentelor care să le consume. |

În tabelul 4.3, cele două subsisteme fac parte din sistemul server pe care aplicația le va folosi. Primul subsistem este cel care va servi clientul, iar cel de-al doilea, numit Consumer, se va ocupa de procesările adiționale de date.

## 4.6. Cerințe non-funcționale

Pentru ca sistemul să ofere toate cerințele cerute acesta trebuie să respecte anumite constrângeri care sunt descrise prin cerințe non-funcționale. Pentru acest proiect, cerințele non-funcționale sunt următoarele:

**Performanța**

Performanța acoperă calitățile sistemului legate de capacitatea unui sistem de a-și îndeplini funcțiile necesare în condiții precizate pentru o anumită perioadă de timp.

Pentru îmbunătățirea timpului se introduce DTO (Data Transfer Object), pentru a nu aduce decât datele necesare pe front-end și nu se vor folosi doar funcțiile prestabilite din ORM, scăzând astfel volumul mare de date.

Conform studiului efectuat de Jakob Nielsen [9] este recomandat ca timpul de răspuns să fie de 1.0 secunde astfel încât fluxul de gândire al utilizatorului să rămână neîntrerupt. Pentru a păstra atenția utilizatorului asupra sarcinii, limita este de 10 secunde. Astfel, timpul de răspuns pentru operații trebuie să fie de 1.0 secunde pentru majoritatea și de 3 secunde pentru operațiile CRUD care trebuie să aducă o cantitate mare de date.

Pentru o accesare mai rapidă a datelor se vor folosi două sisteme de baze de date, una relațională și alta non-relațională. Datele dernomalizate sunt plasate în baza de date nonrelaționată care oferă un acces mai rapid.

### Utilizabilitate

Aceasta acoperă calitățile sistemului legată de ușurința cu care un utilizator poate învăța să opereze, să pregătească intrări și să interpreteze ieșirile unui sistem sau al unei componente.

Aplicația va trebui să poată fi folosită cu ușurință de fiecare tip de utilizator. Cum nu toate persoanele au cunoștințe multe în domeniul fitness-ului și a nutriției, termenii specializați și formulele folosite precizate în interfața utilizator vor fi însoțite de un *tooltip* care să ofere o explicație. Proiectul va conține diagrame pentru reprezentarea progresului. Toate aceste diagrame vor trebui să fie simple și să menționeze semnificația datelor de pe axe, astfel încât aceasta să fie ușor de citit.

Cum aplicația este o aplicație web, aceasta va putea fi accesată ușor de oricine are conexiune la internet și un browser web.

### Openess

Un sistem care respectă această proprietate folosește interfețe, componente și librării bine definite, publice care permit flexibilitate și extensibilitate.

Se vor folosi doar librării open source, care sunt frecvent dezvoltate. Se vor alege API-uri care sunt bine documentate și care asigură o funcționalitate pe termen lung și care pot asigura asistență în cazul apariției unei probleme.

Pentru utilizarea API-urilor se vor defini metode de apel generice, care vor permite înlocuirea ușoară a orcărui API.

Se va alege arhitectura Layered Architecture pentru a realiza modificări care să nu afecteze în lanț alte componente.

### Eterogenitate

Cerința eterogenității este de a integra componente eterogene, adică componente care diferă prin hardware, sistemul de operare, rețele și limbaje de programare utilizate.

Deși soluția propusă este dezvoltarea unei aplicații web, ulterior se poate dori extinderea ei și ca aplicație mobile. Pentru a nu fi necesară o dezvoltare de la zero a întregii aplicației, sistemul se va separa în front-end și back-end, structurarea va fi pe nivele, iar pentru comunicare se vor folosi servicii REST. Astfel, pentru extinderea pe dispozitive mobile va fi necesară dezvoltarea doar a unui nou front-end.

### Securitate

Cerințele non-funcționale de securitate acoperă calitățile sistemului în cauză cu asigurarea siguranței, confidențialității și integrității. Include software-ul, hardware-ul, comunicațiile, accesul utilizatorilor și dispozitivele.

În cadrul acestui proiect, utilizatorii vor trebui să se autentifice prin email și parolă.

Accesul la date se va face pe baza unui rol. Astfel un tip de utilizator, cum ar fi clientul, nu va putea realiza operațiile celorlate tipuri de utilizator, antrenor și administrator.

## 4.7. Baza de date

În alegerea bazei de date care să se utilizeze în proiect, pentru început, s-a realizat o simplă analiză prin luarea unui exemplu pentru a se determina cât de mult ar putea crește numărul de date care trebuie stocate. Dacă, de exemplu, se presupune că un utilizator ar introduce în jur de 10 alimente (înregistrări) pe zi, adică 3650 de înregistrări pe an, pe utilizator. În acest caz, dacă aplicația ar avea 100 de mii de utilizatori, s-ar ajunge la 365 milioane de înregistrări.

Cea mai mare parte a datelor care sunt introduse într-un număr mare și des nu sunt puternic relaționate. Acestea sunt frecvent accesate, atât pentru afișare cât și pentru a realiza statistici cu progresul utilizatorului, ceea ce necesită ca colectarea lor să se realizeze rapid. Din acest motiv, pentru aceste date, s-a ales utilizarea unei baze de date NoSQL (Not only SQL) care sunt cele mai potrivite pentru cazul în care datele trebuie să fie colectate foarte repede și la un volum mare pentru a le analiza. Restul datelor vor fi stocate într-o bază de date SQL.

Pentru alegerea celei mai potrivite baze de date SQL și NoSQL, s-a făcut o scurtă analiză a acelor mai des utilizate dintre acestea, MySQL, Microsoft SQL și PostgreSQL, respectiv MongoDB și Cassandra.

MySQL are viteză puțin mai mare, oferă mecanisme avansate, precum InnoDB, util dacă se efectuează multe instrucțiuni INSERT, și MyISAM, util dacă trebuie să se efectueze multe instrucțiuni SELECT și UPDATE. Ca securitate, oferă protecție securizată prin criptare, utilizând TLS (Transport Layer Security). Dimensiunea maximă a bazei de date este teoretic nelimitată, însă ar fi undeva la 256TB, iar dimensiunea maximă a tabelei este de 256TB pentru MySAM și de 64TB pentru InnoDB. Scalabilitatea și fiabilitatea sunt  sunt în continuă îmbunătățire. Se utilizează funcționalitatea multiprocesării simetrice (SMP) pentru a atribui sarcinile mai logic, gestionând astfel sarcinile mai eficient. Această optimizare asigură că procesul de funcționare este independent, indiferent de tipul de procesor pe care MySQL rulează [10].

PostgreSQL a fost inițial proiectat pentru Linux. Acesta oferă protecție securizată prin criptare, utilizând SSL (Secure Sockets Layer) și a fost conceput pentru integritate ridicată a datelor și utilizării solemne. Codul sursă poate fi modificat după propria nevoie, permițând ștergerea sau adăugarea la acesta. Dimensiunea maximă a bazei de date este nelimitată, iar dimensiunea maximă a tabelei este de 32TB.

Microsoft SQL Server rulează exclusiv pentru Windows și necesită licență. Acesta asigură scalabilitatea ridicată, precum opțiunea de filtrare a datelor bazată pe rânduri. De asemenea, acesta nu blochează baza de date în timpul copierii datelor. Caracteristica permite utilizatorilor să salveze și să restaureze cantități uriașe de date fără a acorda timp și efort suplimentar. O altă caracteristică este aceea că nu permite niciun proces să acceseze sau să manipuleze fișierele sau binarele bazei sale de date. Aceasta

cere utilizatorilor să execute anumite funcții sau să manipuleze fișiere executând o instanță, eliminând astfel posibilitatea ca un hacker să acceseze sau să manipuleze date direct.

MongoDB utilizează documente sub format JSON, grupate grupate în colecții, care la rândul lui este format din perechi câmp:valoare. Backup-ul este complet gestionat și continuu, astfel se salvează automat o copie cu fiecare modificare făcută datelor. O altă caracteristică este point-in-time recovery, ceea ce înseamnă că baza de date poate fi adusă la starea la care a fost la oricare moment din timp, de la ultimul backup efectuat. Scalarea este suportată, precum și interogările care se pot face pe orice câmp din document și suportă indexarea. Alte caracteristici sunt autentificarea, autorizarea, criptarea SSL, crearea de roluri, citirea rapidă, stare consistentă și generarea automată de grafuri.

În Cassandra, structura de bază utilizată pentru a stoca date se bazează pe utilizarea de coloane și este compusă din spațiu de chei, familii de coloane, rânduri și coloane. Oferă autentificare și autorizare, suportă criptare și interogări doar pe chei. Starea este în cele din urmă adusă într-o stare consistentă. O caracteristică importantă este scrierea rapidă, comform studiului realizat de DATASTAX [11].

În concluzie, între MySQL și PostgreSQL nu există diferențe majore, ambele potrivindu-se pentru aplicație. SQL Server este bun pentru aplicații de dimensiuni mari care necesită înaltă securizare. Aplicația propusă nu atinge dimensiuni mari și nici nu necesită securitate înaltă. În plus, SQL Server este un sistem comercial, iar în cadrul dezvoltării acestei aplicații se dorește utilizarea tehnologiilor open source. Astfel s-a ales MySQL.

Între MongoDB și Cassandra, prima realizează citirea mai rapidă, în timp ce a doua realizează o scriere și modificare mai rapidă. De asemenea, MongoDB suportă scalarea. Astfel, cum datele vor fi des accesate pentru citire, va fi necesară o scalare ușoară și generarea de grafuri pentru analiză, pentru crearea bazei de date non-relaționale s-a ales MongoDB.

## 4.8. Resurse necesare

Pentru realizarea platformei vor fi necesare următoarele resurse:

2. Baze de date: MySQL, MongoDB;
3. Development Kits: JDK 1.8;
4. Frameworks: Spring MVC, Spring Data, Spring Boot, Hibernate (Object-Relational Mapper), Angular 6, Bootstrap
5. Build Automation: Gradle
6. Architectural Styles: REST
7. IDEs: IntelliJ, Netbeans
8. Hardware:
    - Sistem bazat pe arhitectura x64 (64 de biti);
    - Memorie RAM: 8 GB RAM;
    - Procesor: frecventa minima 2.0 GHz, 2 nuclee.

## 4.9. Arhitectura conceptuală generală

Sistemul este împărțit în două aplicații, pe modelul client-server.

Pentru partea de client se folosește Angular, care generează paginile dinamice web pentru utilizatori și trimite cereri REST prin HTTP spre server sau apeluri spre API-ul Chatckit, pentru pagina de chat.

Pe partea de server se folosește Tomcat ca server și Spring ca framework. Organizarea arhitecturală este pe trei nivele, conform impunerii framework-ului Spring. Acesta preia cererile de la client și îi returnează rezultatul cerut. Pentru obținerea datelor nutriționale a unui produs alimentar, în cazul în care alimentul nu se găsește în baza de date, se folosește adițional un API (Application Program Interface). Pentru trimiterea de mail utilizatorilor se folosește serviciul RabbitMQ.

Deoarece anumite operații necesită un timp mai lung pentru procesare, iar numărul datelor care trebuie să fie procesate poate fi mare, se adaugă un subsistem,

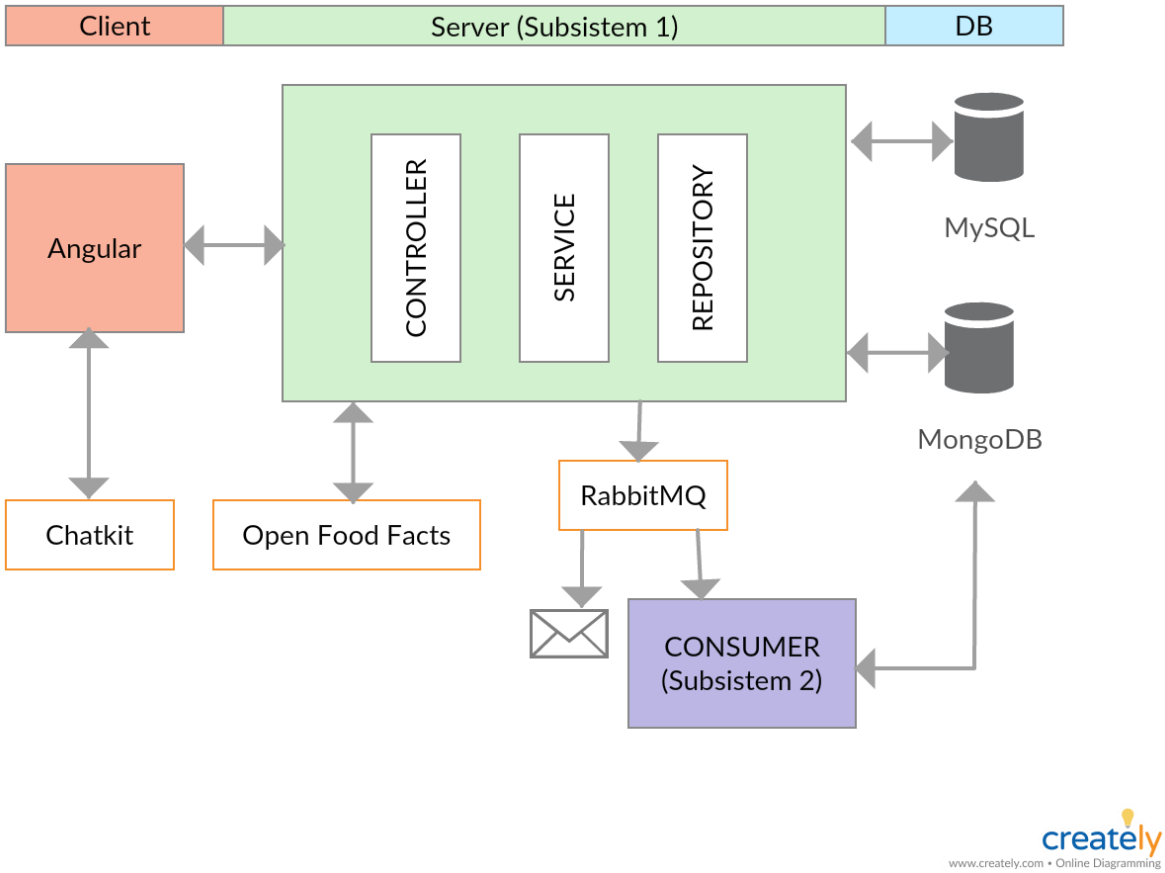



Figura 4.6 Arhitectura conceptuală generală

CONSUMER, care să realizeze aceste operații. Astfel clientul nu va trebui să aștepte până când se termină complet toate operațiile. Pentru comunicarea cu acest microservicu se folosesc cozile oferite de serviciul RabbitMQ.

Persistența datelor se face pe două baze de date, MySQL și MongoDB.

# Capitolul 5. Proiectare de Detaliu și Implementare

## 5.1. Stil arhitectural ales

Sistemul este împărțit în două aplicații, pe baza modelului distribuit de comunicare client-server, atât pentru o mai bună separare a front-end de back-end, dar și pentru introducerea posibilității de a extinde ușor aplicația și pe platforma mobile. Partea de front-end, clientul, realizează cereri serverului pentru servicii pe care le oferă sau pentru a obține date, în timp ce pe back-end server-ul preia cererile, realizează serviciul cerut și/sau obține datele cerute pe care le returnează clientului. Comunicarea dintre cele două componente se face prin apeluri REST (Representational State Transfer).

REST este un stil arhitectural utilizat pentru a furniza standarde între sistemele de pe web pentru a facilita comunicarea între ele și pentru a permite implementarea clientului și a serverului independent. REST utilizează HTTP (Hyper Text Transfer Protocol) pentru comunicarea cu resursele de pe internet. Trimiterea de date între front-end și back-end se face prin obiecte JSON (JavaScript Object Notation). JSON este o sintaxă care poate fi folosită pentru a stoca și trimite date.

Partea de back-end este separată în două subsisteme, Subsistem1 care comunică cu clientul și are rolul de server, și un subsistem secundar, Consumer, care este folosit pentru a realiza operații pe un număr mare de date care consumă mult timp. Comunicarea dintre acestea se face folosind cozile din RabbitMQ. Pentru ambele subsisteme, ca framework, se folosește Spring Framework Architecture. Acesta este o arhitectură organizată pe nivele (layered architecture) care consistă din mai multe module, fiecare construit peste containerul său principal. Aceste module oferă tot de ceea ce este nevoie pentru dezvoltarea aplicațiilor. Dezvoltatorul poate alege ce caracteristici oferite de acestea dorește și poate elimina modulele care nu le folosește. Pentru această aplicație se folosește cea mai simplă și des întâlnită formă de arhitectură, organizată pe patru nivele: presentation, business logic, persistence și database.

Nivelele dintr-o astfel de arhitectură sunt definite ca fiind *închise*. Acest lucru înseamnă că o cerere este transmisă de la un nivel adiacent la alt nivel adiacent lui. Deci, de exemplu, o cerere din niveleul Controller nu poate să fie transmisă direct spre nivelul Repository, ci trebuie să treacă prima dată prin nivelul Service. Prin izolarea nivelelor o modificare efectuată într-un nivel va afecta doar nivelele adiacente cu acesta. Fiecare nivel al modelului arhitectural are un rol și o responsabilitate specifice în cadrul aplicației și formează o abstracție în jurul muncii care trebuie realizată pentru a satisface o anumită cerere. Astfel, de exemplu, nivelul Controller nu trebuie să știe cum anume se efectuează operațiile, ci doar ce operații trebuie efectuați și care este ordinea lor, iar nivelul Service nu trebuie să știe de unde provin datele de la client, cum se transmit datele spre client sau cum anume se obțin datele din baza de date. Prin definirea clară a componentelor și limitarea scopului acestora, acest tip de organizare ușurează construirea și clasificarea pe roluri și atribuirea responsabilității eficient. Tot această abordare facilitează dezvoltarea și testarea aplicației, precum și întreținerea ei. Acest lucru va permite, de exemplu, schimbarea bazei de date, necesitând doar actualizarea nivelului de persistență.

Alte obiecte folosite în sunt entitățile și DTO-urile. O entitate (model) reprezintă un tabel din baza de date relațională sau un document din baza de date NoSQL și fiecare instanță a entității corespunde unei intrări din baza de date. Un DTO este un obiect care

este transmis în afara aplicației, clientului. Acesta conține doar o parte din datele entității, și anume cele de care clientul are nevoie.

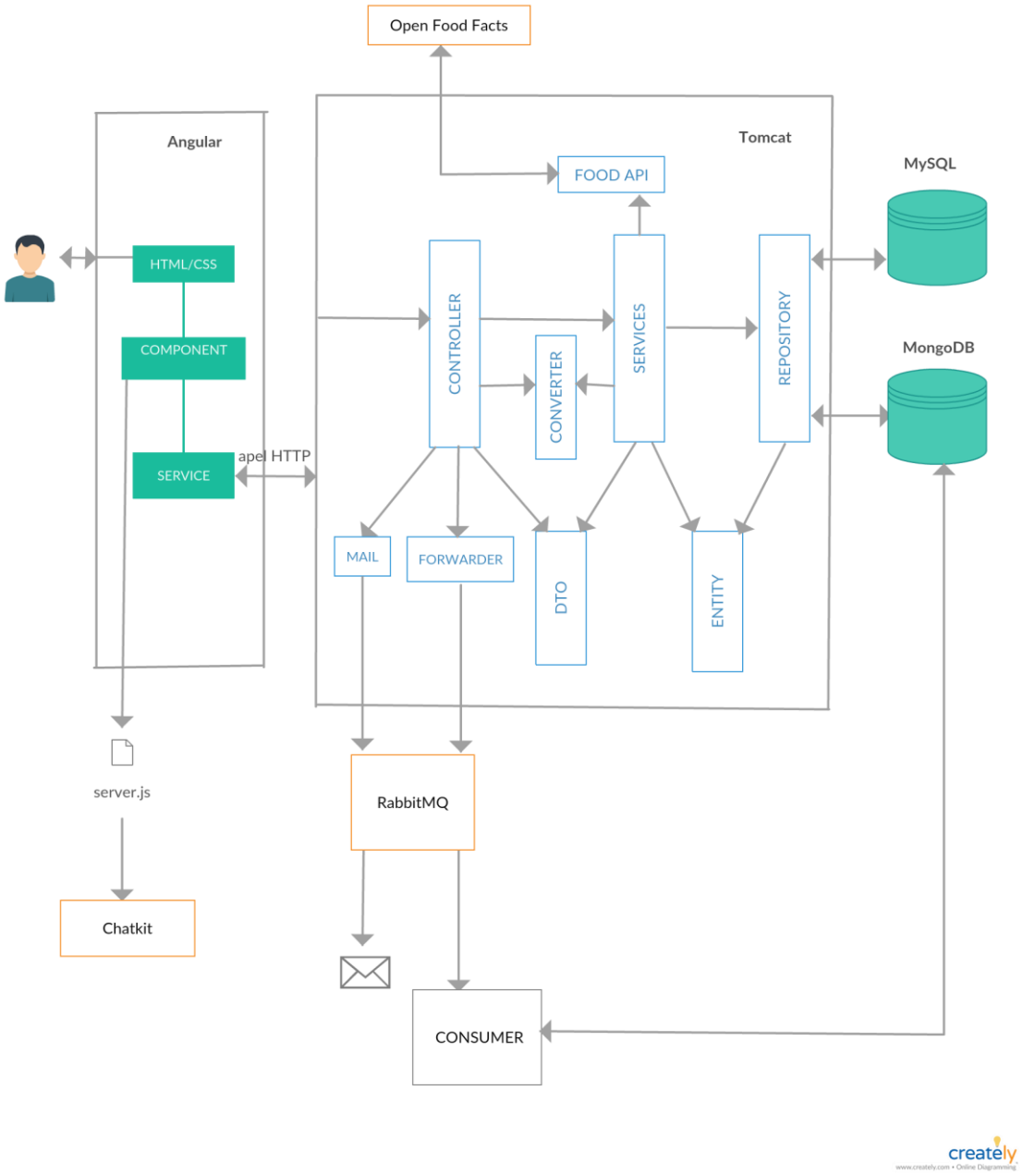



Figura 5.1 Arhitectura conceptuală

Pentru partea de client se folosește Angular 6, care, în esență este bazat pe framework-ul MVC. Componentele, serviciile și directivele pot fi considerate controlerul, template-ul este view-ul, iar modelul este același din framework.

În figura 5.1 este prezentată arhitectura conceptuală a întregii aplicații. Tot în figură se poate observa utilizarea cozilor oferite de RabbitMQ pentru a trimite email, API-ul Open Food Facts pentru a obține date despre alimente și Chatkit care a fost folosit pentru a implementa pagina de chat.

## 5.2. Arhitectura back-end

Primul nivel, *Presentation Layer*, este nivelul care expune funcționalitatea aplicației ca API capabil să gestioneze cererile REST HTTP. Acesta preia informațiile primite, le transmite mai departe nivelului următor de business logic, de la care primește un răspuns pe care îl transimte clientului. Aici se află clasele controller din pachetul "controller". Pentru a mapa URL-urile la metodele din controller se folosește adnotarea din Spring @RequestMapping. Spring suportă cinci tipuri de adnotări, pentru tipurile de cereri HTTP GET, POST, PUT, DELETE și PATCH. Acestea sunt @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping. @GetMapping este folosit pentru a prelua cereri de tip GET, @PostMapping este folosit pentru a prelua cereri de tip POST, etc. Parametrul utilizat în @RequestMapping este sufixul din URL-ul metodei. În metode, ca parametrii, se dau variabile din calea URL, adnotat cu @PathVariable, și obiectul DTO (Data Transfer Object) care este primit de la client, adnotat prin @RequestBody.

Al doilea nivel, *Business Logic Layer*, reprezintă nivelul logic de business a aplicației Spring. Acesta translatează obiectele DTO în entități și invers și efectuează operații complexe, cum ar fi validarea datelor primite de la *Controller* și realizarea efectivă a calculelor din cadrul operațiilor. Cum implementarea trebuie să fie ascunsă de interacțiunile externe, s-au folosit interfețele. Toate aceste clase se află în pachetul "service".

DTO-urile, aflate în pachetul "dto", sunt folosite deoarece nu se dorește expunerea întregului model clientului, acesta având nevoie doar de anumite date din model. De asemenea, trimiterea unui model cu atribute nule ar fi costisitoare ca timp de transfer. Astfel, pentru fiecare entitate există o clasă dto corespondentă. Sunt și cereri de la client care necesită trimiterea datelor de la mai multe modele, caz în care s-au construit DTO-uri care să cuprindă toate aceste date, evitând necesitatea trimiterii mai multor cereri din partea clientului pentru a obține toate datele. Convertirea de la model la DTO și invers se face separat, de clase specializare, numite *Converter*, care se afă tot în nivelul de business, în pachetul "converter". Fiecare model are atribuit o clasă converter, implementând cele două metode, pentru cele două tipuri de convertire, din interfața *Converter*. Interfața este folosită și în acest caz pentru a ascunde implementarea, dar și pentru a asigura faptul că fiecare clasă de convertire va avea cele două metode implementate.

Deoarece în multe cazuri trebuie transmisă o colecție de DTO-uri, pentru a nu fi necesară scrierea de afiecare dată a unui *for* sau *stream*, s-a creat o clasă abstractă, *AbstractConverter*, care implementează cele două metode de convertire pentru colecții. Folosind stream-urile din Java8, aceasta aplică pe fiecare model din colecția dată operația de converitre, pe care le returnează înapoi sub noua formă tot într-o colecție.

Mai jos, în figura 5.3, este ilustrat un exemplu pentru o clasă de convertire corespunzătoare modelului User, UserConverter, cu legăturile dintre clasele generice.

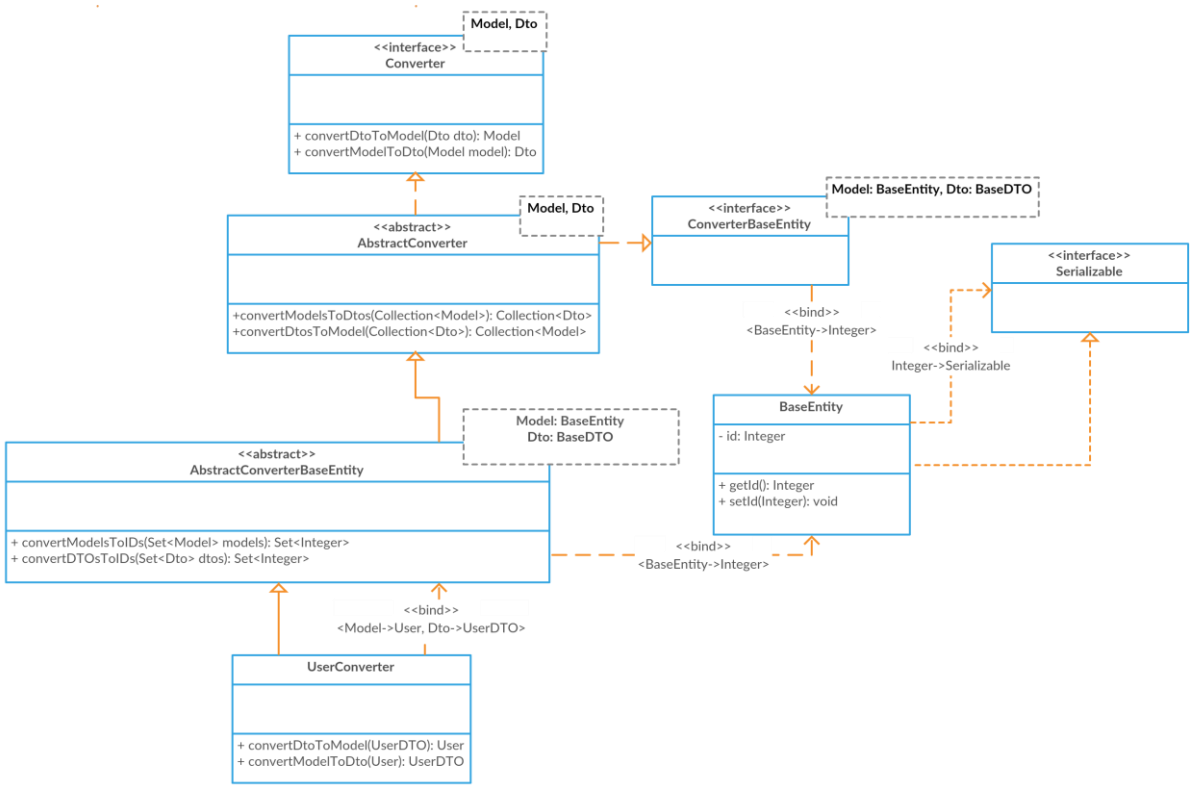



Figura 5.3 Legătura dintre clasele generice pentru convertire

Așa cum se poate vedea în figură, *UserConverter* extinde clasa abstractă de convertire, *AbstractConverterBase* indirect prin clasa *AbstractConverterBaseEntity* pentru a avea metodele specifice pentru colecții. *AbstractConverterBaseEntity* conține două metode care extrag id-urile din modelele din colecția transmisă ca parametru. Tot indirect, *UserConverter* implementează interfața *Converter* cu cele două metode de convertire. *BaseEntity* conține atributul id al entității, cu meotdele *getter* și *setter* pentru acest atribut. Prin folosirea acestei clase, nu va mai trebui să se specifice în fiecare entitate id-ul cu constrângerile asupra lui. Această clasă, de asemenea, implementează interfața *Serializable*, astfel DTO-urile pot fi serializate pentru a fi transmise spre client.

Tot în nivelul business logic se află clasele din pachetele “api”, “batch”, “config”, “exception” și “mail”. Pachetul “api” conține clasele care au rolul de a apela API-urile. Prin punerea metodelor care fac apeluri spre API-uri în clase separate, API-ul utilizat poate fi oricând schimbat ușor, fiind necesară doar modificarea metodei care realizează apelul, ceea ce returnează metoda rămânând neschimbat, astfel și metodele din alte clase care o apelează. Pachetul “batch” conține clasele folosite pentru a trimite date spre al doilea subsistem. Pachetul “config” conține clasele de excepții folosite pentru cazul în care o cerere nu se poate realiza cu succes. Excepțiile sunt aruncate în clasele de service în cazul în care este întâlnită o problemă, urmând să fie tratată în clasa controller. Pachetul “config” conține clasele de configurații și de autentificare și generare de token.

Al treilea nivel, Repository, este cel responsabil cu comunicarea cu baza de date. Acesta conține interfețele Repository, aflate în pachetul "repository", care facilitează accesul la baza de date, pentru accesarea și modificarea datelor.

Ultimul nivel, al bazei de date, poate fi accesat doar de nivelul de persistență și conține tabelele și legăturile dintre ele.

În figura 5.4 este ilustrată o porțiune din dependința dintre clase, pentru User. În anexă se află diagrama de clase pentru aceste clase.

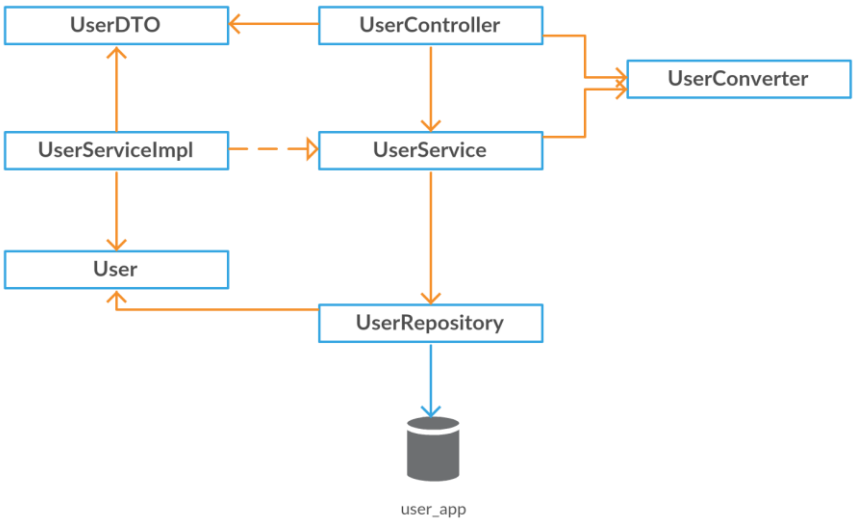



Figura 5.4 Porțiune din dependința dintre clase

În framework-ul Spring se folosesc mai multe adnotări pentru a specifica rolul clasei, bean. În Spring, bean-urile sunt obiectele care formează elementele principale ale aplicației și care sunt gestionate de containerul Spring IoC (Inversion of Control). IoC este un proces în care un obiect definește dependențele sale fără a le crea, delegând această sarcină de a construi dependințele unui container IoC. Definirea dependințelor se realizează prin folosirea adnotării @Autowired. Definirea bean-urilor se face prin adnotările @Component, @Repository, @Service și @Controller. Spring va importa aceste bean-uri în container și va injecta dependințele. Adnotarea @Component marchează clasa ca fiind un bean. Adnotarea @Repository este o specializare a adnotării @Component cu utilizări și funcționalități similare, dar în plus face ca excepțiile unchecked, aruncate de metodele din aceste clase care accesează baza de date, să fie ligibile pentru translație în Spring. Adnotarea @Service este tot o specializare a @Component. Deși aceasta nu aduce nimic în plus este bine să fie folosită în special în arhitectura pe nivele pentru a specifica rolul clasei mai bine. Adnotarea @Controller marchează clasa ca fiind o clasă controller. Cum multe metode din clasele controller returnează un json fără a returna o pagină, trebuie pe lângă @Controller să se adauge adnotarea @ResponseBody la fiecare metodă, pentru a specifica acest lucru. O prescurtare folosită în acest proiect este @RestController care combină adnotăriile @Controller și @ResponseBody. Această adnotare specifică faptul că clasa adnotată corespunzătoare poate gestiona serviciile RESTful WEB.

Multe clase trebuie să aibă metodele *getter*, *setter*, *toString*, dar și constructorul default, fără parametrii și constructorul cu toate argumentele. Scrierea de fiecare dată a

acestora devine prea complicată și încarcă clasa, îngreunând citirea acesteia. Pentru simplificare, s-au folosit adnotările lombok. Prin folosirea adnotărilor @Getter și @Setter, de exemplu, Lobok va genera metodele getter și setter pentru toate câmpurile clasei, iar prin @NoArgsContructor și @AllArgsConstructor, va genera constuctorul default, respectiv construcorul cu toate argumentele. Pentru câmpurile private, care nu se dorește să poată fi modificate prin metode setter, se folosește notarea @Setter(AccessLevel.PROTECTED) în declararea câmpului.

Pentru crearea aplicației este folosit Spring Boot. Acesta este construit peste Spring și permite dezvoltatorului să se concentreze asupra dezvoltării aplicației, eliminând necesitatea de a se preocupa de celelalte aspecte ale ciclului său de viață, de implementare și de management. Ca server se folosește Tomcat care este un server web Java open source. Aplicația se pornește din clasa *UserPortalApplication*, care se află în același loc cu restul pachetelor folosite, pentru a putea fi văzute la rulare. Adnotarea @SpringBootAplication este folosită pentru a declanșa autoconfigurarea și scanarea componentelor. Aceasta este identică cu folosirea adnotărilor @Configuration, @EnableAutoConfiguration și @ComponentScan.

## 5.3. Arhitectura front-end

Versiunea de Angular folosită în acest proiect este Angular 6. Față de AngularJS, aici controllerele sunt înlocuite de componente, servicii și directive. Modelul reprezină în continuare clasa model, care corespunde DTO-ului din back-end și care este folosit pentru a trimite și primi date de la back-end. View a rămas la fel, reprezentând fișierul HTML și CSS care afișează și primește informații de la utilizator. În figura 5.5 este ilustrată arhitectura generală a Angular.

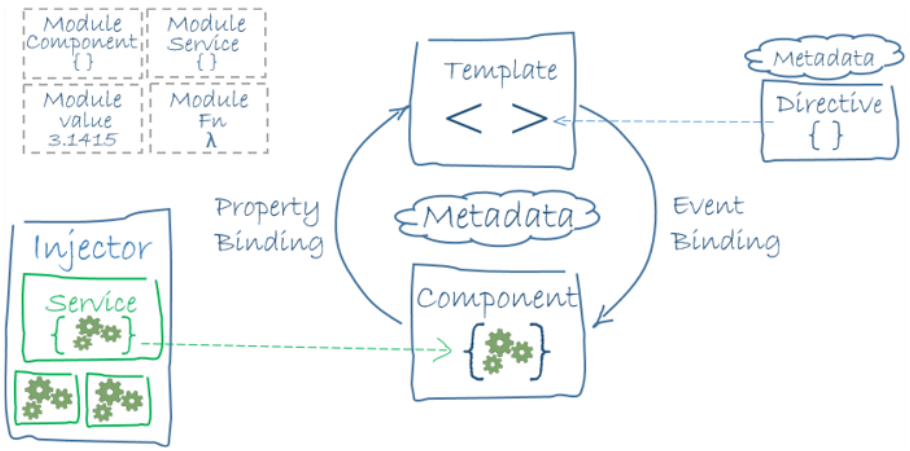



Figura 5.5 Arhitectura Angular

Fiecare componentă controlează o bucată din ecran, numit *view,* reprezentat prin HTML și CSS. Întreaga logică a componentei este definită într-o clasă, care interacționează cu pagina HTML prin proprietăți și metode. Angular crează, actualizează și distruge componentele pe parcurs ce utilizatorul navighează prin aplicație. Clasa este

identificată prin adnotarea @Component, numit *decorator,* care este folosit și pentru a specifica metadata clasei. În metadată, pentru acest proiect, s-au specificat: selector, templateUrl, styleUrls, directives și providers. Selector este un selector CSS care poate fi folosit în HTML pentru a spune lui Angular să creeze și să insereze o instanță a acestei componente. TemplateUrl specifică calea spre fișierul HTML asociat componentei. StyleUrls specifică calea spre fișierele CSS asociat componentei care să fie aplicate pe fișierul HTML. Directives este un tablou de componente sau directive care sunt necesare și sunt folosite pentru a adăuga alte funcționalități. Provider este un tablou în care se specifică serviciile de care componenta este dependentă.

Ca și structură a proiectului, toate fișierele se află în "src/app". Pentru fiecare pagină din aplicație este creat un folder cu patru fișiere: fișierul cu extensie ts în care este definită componenta, fișierele HTML și CSS și fișierul pentru testare a componentei .spec.ts. Fișierele spec sunt teste unit test pentru fișierele sursă. Convenția impusă pentru aplciațiile Angular este ca fiecare fișier .ts să aibă un fișier .spec.ts. Testele sunt executate folosind frameworkul javascript de testare Jasmine prin executorul de teste Karma, la folosirea comenzii "ng test". Pentru o mai bună organizare, folderele au fost grupate după tipul de utilizator care utilizează componenta, în folderul *client* și *coach,* iar cele comune au fost lăsate împreună.

Metodele care realizează apeluri REST sunt puse în servicii, care sunt folosite de componente. Fiecare clasă controller din back-end are asociată o clasă service, cu o metodă de apel pentru fiecare metodă din controller. Astfel, metodele sunt ușor de identificat și modificat. Fiecaree clasă service are pus decoratorul @Injectable() pentru a genera metadatele. Acest lucru va permite adăugare de dependințe prin injecție, altfel o excepție va fi aruncată. Toate fișierele se afă în "src/app/services".

Inițializările și declarațiile sunt puse în metoda *ngOnInit.* Aceasta este apelată doar o dată, după inițializarea legăturilor clasei. Constructorul este folosit strict doar pentru a inițializa clasele membre ale clasei și pentru a injecta dependințele. Motivul este acela că constructorul este apelat înainte ca componenta să fie creată, în timp ce ngOnInit va fi apelat doar după crearea componentei. Metoda aparține interfeței OnInit, astfel clasa va trebui să implementeze această interfață.

Pentru a preveni pierderi de memorie, trebuie să se facă un subscribe înainte ca o componentă care are subscribes să fie distrusă. Acestea se fac în metoda *ngOnDestroy(),* oferită de interfața OnDestroy, care este executată înainte ca componeta să fie distrusă.

Componenta principală din care pornește aplicația este AppComponent, aflată în fișierul app.component.ts. Fiecare componentă, serviciu, directivă, pipe și modul folosit în aplicație trebuie să fie declarate în fișierul app.modules.ts, altfel o eroare va fi aruncată în consola browserului. În fișierul app.routing.modules.ts se află modulul AppRoutingModule în care este configurat ruterul cu Routes. Routes indică ruterului care view să se afișeze pentru fiecare link folosit în aplicație, în path specificându-se URL-ul, iar în component componenta pe care ruterul trebuie să o creeze când se navighează la ruta specificată în path. Imaginile folosite în aplicație se află în "src/assets/images".

Alte două fișiere importante sunt package.json, care conține metadata proiectului, și tslint.json, care conține regulile de cod Typescript.

### Componente child

Nutriționistul și antrenorul trebuie să poată accesa anumite pagini ale clientului pentru a le vizualiza și în unele cazuri modifica. În fiecare componentă destinată clientului primul lucru care se face la inițializare, în *gnOnInit(),* este verificarea dacă datele principale ale utilizatorului, care sunt folosite des în componente, sunt salvate în sesiune. Pe lângă această metodă, restul codului este același și pentru antrenor/nutriționist. Pentru a nu se scrie cod duplicat, pentru fiecare astfel de pagină s-au creat două componente principale părinte, una pentru client și una pentru antrenor/nutriționist. Ambele componente au inserată o componentă child care conține codul comun.

Componente child are un atribut *user* care este dat de componenta părinte și conține clientul a cărui date trebuie luate pentru acea pagină. Astfel componenta părinte pentru client ia direct datele clientului salvate la logarea sa, în timp ce componenta părinte pentru antrenor/nutriționist verifică dacă antrenorul/nutriționistul a selectat un client pentru care să afișeze datele, dacă le găsește le ia și le transmite componetei child, iar în caz contrar îl redirecționează spre pagina principală unde poate să aleagă un client.

Datele clientului sunt ținute în serviciul ColabService, în atribute private. Verificarea existenței datelor se face cu o metodă din serviciu, *isDataValid().* Setarea acestor date se face prin metode de setare implementate tot aici. Nu este definită nici o metodă care să returneze aceste valori pentru a preveni orice accces direct la acestea. În paginile HTML, anumite elemente sunt afișate doar dacă utilizatorul este un client pentru a nu da antrenorului/nutriționistului accesul la anumite operații neautorizate cum ar fi ștergerea unui aliment din lista personală. Pentru a asigura această constrângere, pe partea de server, metodele au specificate care tipuri de utilizatori sunt autorizați să apeleze metoda, tipul fiind obținut cu tokenul care este folosit la fiecare cerere.

### Stocare temporlă a datelor

Anumite date ale utilizatorului sunt folosite des, în majoritatea paginilor. Pentru a nu fi necesară aducerea constantă a acestora, s-a decis ca acestea să fie aduse o singură dată, după logare și să fie ținute în serviciul *UserService*. Accesarea lor se poate face doar prin metode getter și setter. Datele sunt pierdute în cazul unui refresh, astfel, în fiecare metodă *ngOnInit()* se verifică la început dacă datele există pentru a fi readuse dacă este necesar.

Pentru stocarea pe partea clientului a datelor necesare tot timpul utilizării aplicației se pot folosi *cookies, localStorage* și *sessionStorage*. Sesiunile permit stocarea unei cantități mai mare de date spre deosebire de *cookies*, permițând stocarea de imagini, cerință necesară pentru stocarea imaginii de profil a utilizatorului. Astfel acest proiect necesită folosirea de sesiuni. Între cele două sesiuni s-a ales *sessionStorage,* motivul fiind securitatea pe care *localStorage* nu o poate oferi. *localStorage* a fost creat să fie un spațiu de stocare bazat pe cheie/valoare care să permită dezvoltarea de aplicații web Single Page complexe. Orice cod JavaScript poate acesa datele din *localStorage*, acesta neavând o protecție a datelor. Astfel, dacă un atacator poate rula cod JavaScript pe website, va putea să extragă toate datele din *localStorage* și să le trimită în domeniul său. Cum acest proiect folosește Angular care este bazat pe JavaScript, există posibilitatea de a se rula cod JavaScript. Datele care trebuie stocate sunt token-ul, imaginea de profil, email-ul și tipul

utilizatorului, care sunt date sensibile, deci trebuie să se folosească sessionStorage. *sessionStorage* stochează datele tot pe bază de cheie/valoare. La închiderea tabului sesiunea va fi ștearsă automat, iar utilizatorul va trebui să se logheze din nou. Utilizarea sesiunii nu trebuie abuzată însă. Doar datele strict importante trebuie ținute aici, un număr mare de date putând să afecteze scalabilitatea și performanța. Odată ce o cheie nu mai este necesară, aceasta este eliminată prin metoda *removeItem(key).*

*sessionStorage* se asigură că datele nu pot fi accesate de un atacator limitând accesul la date într-o singură fereastră, legând datele la protocol, domeniu și port și apoi ștergând datele când fereastra este închisă. Următorul scenariu, însă, trebuie luat în considerare. Când utilizatorul se înregistrează în aplicație, datele necesare sunt salvate în sesiunea sessionStorage. Dacă utilizatorul deschide aplicația într-o altă fereastră în care se deconectează, la trecerea în fereastra inițială datele sunt încă prezente în sesiune, prin aceste date aflându-se tokenul care îi va permite utilizatorului să poată realiza toate operațiile, printre care vizualizarea datelor personale, chiar dacă este deconectat. Cel mai periculos este dacă utilizatorul părăsește calculatorul fără a închide browserul și o altă persoană merge în locul său la calculator. Pentru a evita această problemă identitatea utilizatorului trebuie verificată de fiecare dată când datele sunt accesate pentru citire sau scriere. Acest lucru se poate face prin folosirea unui *cookie* specific pentru domeniul aplicației care să fie salvat în *sessionStorage* când se salvează și restul datelor. De fiecare dată când se citește sau se scrie din sessionStorage, se face o verificare pentru a se vedea dacă valoarea curentă a *cookie*-ului este identică cu cea stocată în sesiune. Dacă există o diferență, se vor șterge toate datele din sesiune.

**Validarea adresei de email**

Validarea emailurilor se face utilizând directiva Angular PatternValidator cu un regex. Dacă patternul nu se potrivește se va primi o eroare de validare. În acest mod se poate defini o validare de email personalizată. Verificarea existenței adresei de mail se face prin trimiterea unui mail de confirmare.

Multe elemente UI au fost create folosind componentele oferite de biblioteca Angular Material, un framework UI. Mai departe sunt prezentate componentele Angular Material folosite în acest proiect.

- **Datepicker**

Pentru alegerea datei se folosește Datepicker oferit de Angular Material. Acesta permite utilizatorului să aleagă ușor o dată prin tastarea datei sau prin alegerea ei din calendar. Pentru a evita tastarea greșită a datei de către utilizator, această opțiune s-a dezactivat prin adăugarea proprietății *disabled*. Formatul ales să se afișeze data este cel european, “DD-MM-YYY”, care este diferit de formatul implicit care este oferit de componenta datepicker, “MM/DD/YYYY”. În prezent Angular nu oferă o metodă simplă de a modifica data prin @Input, astfel trebuie scris propriul *DateAdapter*. Acest lucru se face prin crearea unei clase care să extindă *NativeDatreAdapter,* care extinde la rândul ei clasa abstractă DateAdapter<Date>. Această clasă este denumită *CustomDateAdapter* și este definită în fișierul “date-format.ts”, în directorul “app/diary-generic/date-format”. În clasă se implementează metoda format care returnează sub format string data selectată de utilizator sub formatul dorit. Aceasta primește ca

parametrii data care a fost selectată și formatul în care să fie afișată, separă ziua, luna și anul și le pune în formatul dat. În cazul în care formatul dat nu corespunde cu nici un caz, se returnează formatul implicit. Următorul pas este de a crea formatul de dată dorit. Pentru aceasta, în app.module.ts, se declară o constantă MY_DATE_FORMATS unde să poată fi adăugate oricâte formate se dorește. Tot în acest fișier, în providers, se adaugă clasa nou definită *CustomDateAdapter* pentru a fi folosită în loc de *DateAdapter* implicit și constanta MY_DATE_FORMATS pentru a fi folosită în loc de MAT_DATE_FORMATS. MAT_DATE_FORMATS este o colecție de formate pe care datepicker le folosește când parsează și afișează data, asemănător cu MY_DATE_FORMATS care a fost definit.

- **Autocomplete**

Pentru crearea planului în pagina "Meal Plan", introducerea unui nou aliment se face introducând denumirea acestuia într-un câmp cu autocompletare, pentru a ușura introducerea acestuia. Pentru autocompletare se folosește Autocomplete din Angular Material. În modul normal, componenta primește sub formă de listă toate alimentele din lista utilizatorului pe care o afișează în partea de jos a câmpului în mod drop down, utilizatorul putând să selecteze alimentul dorit. În plus, s-a adăugat opțiunea de filtrare care să reducă numărul de rezultate în cazul în care utilizatorul are o listă lungă de alimente. Filtrarea se face prin adăugarea unui pipe care aplică pe fiecare element o metodă de filtrare *_filter(value),* implementat în componenta în care este utilizată. Metoda ia textul introdus de utilizator în câmp și filtrează elementele din listă la alimentele care conțin textul respectiv.

- **Flow chat box**

Pentru chat box s-a folosit componenta "Drag and Drop" din Angular Material. În fereastră se afișează prima dată o listă cu toți colaboratorii și convorbirile cu un colaborator la selectarea acestuia din listă. Deoarece trebuie să se fișeze același lucru ca în paginile normale, s-au introdus componentele corespunzătoare celor două pagini, cu lista și cu pagina de chat, ca și componentă child în componenta "Drag and Drop", iar în fișierul CSS s-a modificat dimensiunea acestora pentru a fi încadrate în fereastră. Fereastra se deschide și se închide printr-o iconiță din meniu și rămâne în poziția pusă pe toată perioada navigației prin aplicație. Pentru ca fereastra să apară deasupra celorlalte elemente și componente din pagină, în CSS, trebuie setate valorile de adâncime z-index și poziția alta decât "static".

## 5.4. Diagrama de deployment

Diagrama de deployment este folosită pentru a prezenta structura hardware necesară pentru a rula întreaga aplicație. Aceasta arată componentele hardware pe care sunt puse să ruleze componentele software ale aplicației și modul de interconectare al componentelor.

În figura 5.6 este prezentată diagrama de deployment a sistemului.

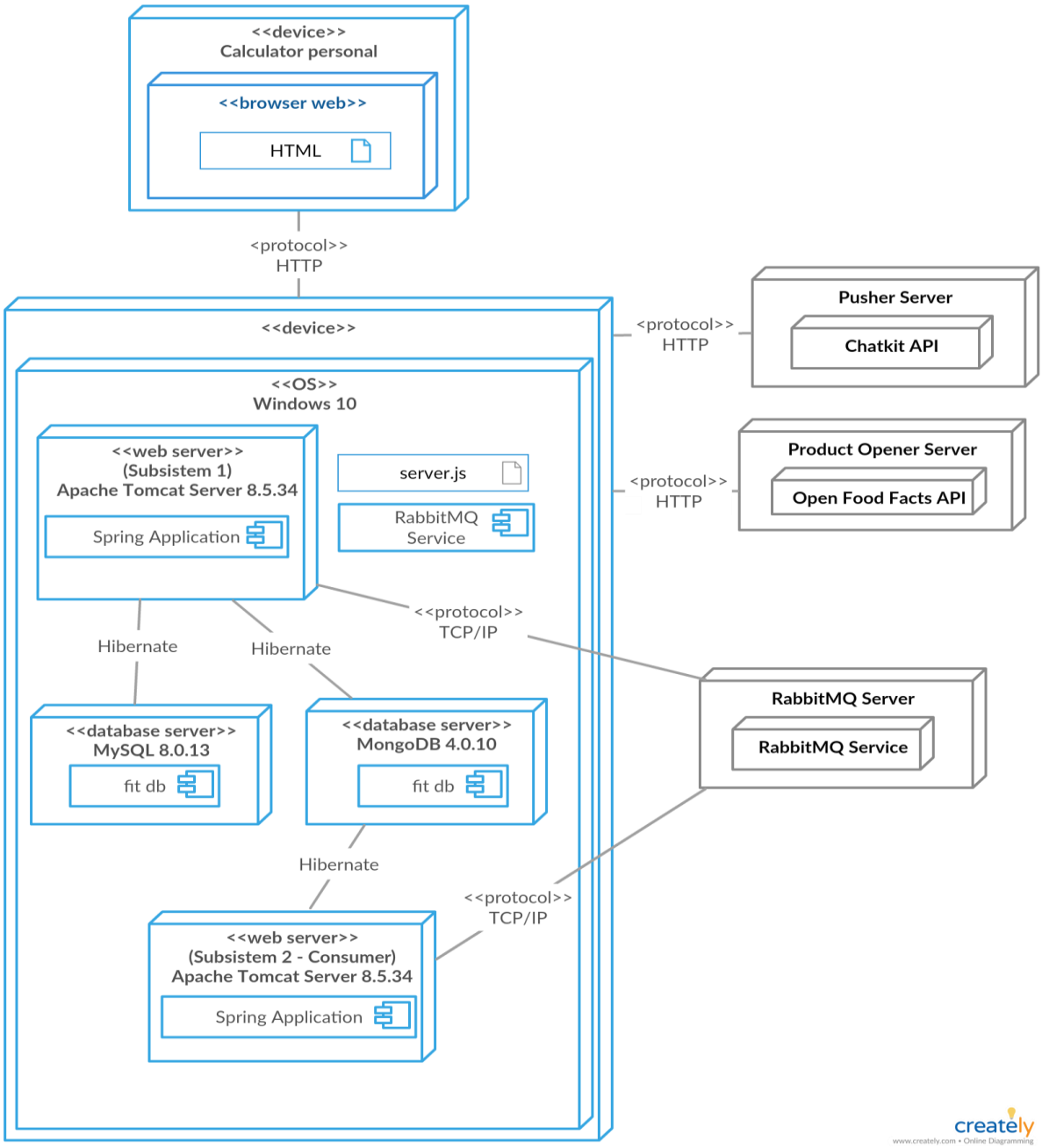



Figura 5.6 Diagrama de deployment

Clienții folosesc un browser de pe calculatorul personal prin care comunică cu serverul folosind protocolul HTTP. Deși în prezent toate componentele sunt puse pe același dispozitiv, proiectul permite scalarea tip UpScaling, prin adăugare de resurse. În cazul unui woarkload crescut care poate apărea în timp, serverele pot fi puse pe calculatoare diferite. Astfel fiecare server va avea propriile resurse.

## 5.5. Elemente de implementare

### *5.5.1. Autentificarea și generarea token-urilor*

Autorizarea se face pe bază de roluri, folosind spring security. Sunt stabilite trei roluri în cadrul aplicației: ADMIN, pentru administrator, CLIENT, reprezentând clientul, și COACH, reprezentând antrenorul și nutriționsitul. Întreaga configurație legată de securitate se află în clasa WebSecurityConfig din pachetul „config”. Aici este configurat ca url-urile „/token” și „/signup” să nu necesite autentificare, restul url-urilor rămânând securizate. Setarea proprietății „prePostEnabled” la true permite utilizarea adnotării @PreAuthorize. Generarea tokenului este realizează de controlerul AuthenticationController folosind TokenProvider. TokenProvider este responsabil cu validarea utilizatorului și generarea tokenului JWT (JSON Web Token). Clasa JwtAuthenticationFilter extinde OncePerRequestFilter care asigură o singură execuție per cerere. Aceasta interceptează toate cererile și caută tokenul din antet, îl extrage și îl analizează pentru a găsi informațiile legate de utilizator, numele și rolul acestuia. Acest lucru va proteja aplicația de cererile care nu au un token de autorizare.

Tokenul este compus din trei părți separate prin puncte: header, payload și semnătura. Payload conține informații despre token, printre care timpul expirării. Timpul expirării este setat la crearea tokenului și este setat la cinci ore de la timpul creării (timpul curent în milisecunde + 60 * 60 * 30). Timpul de 5 ore reprezentat în milisecunde este stocat ca o constantă în clasa Constants. Tot în acest fișier se află cheia pentru semnătură.

În momentul logării cu succes în aplicație, clientul va primi un token de la server pe care îl va salva în sesiune. Pentru fiecare cerere la server, tokenul primit este atașat pentru autentificare. Serverul îl validează și trimite înapoi răspunsul cererii sau o eroare în cazul în care tokenul nu este valid.

În controlere, fiecare metodă este adnotată cu @PreAuthorize, în care se specifică care roluri au autorizația de a accesa metoda respectivă. Se poate folosi și adnotarea @Secured, pentru aceasta fiind nevoie să se seteze proprietatea securedEnabled la true în WebSecurityConfig.

Parolele sunt criptate folosind BCrypt oferit de Spring Security[9]. Acesta implementează hashingul de parole Blowfish în stilul OpenBSD. Spre deosebire de PBKDF2 care este des folosit, BCrypt folosește un algoritm costisitor din punct de vedere al timpului. Acest lucru face ca parola să fie mai sigură împotriva atacurilor de forță brută, deoarece atacatorul are nevoie de mai mult timp pentru a testa fiecare cheie posibilă.

[9] https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html

### *5.5.2. Chart-uri*

Diagramele și cercurile cu progres sunt create folosind frameworkul *ngx-charts* care folosește Angular pentru a randa și anima elementele SVG și folosește biblioteca JavaScript D3[10] pentru funcționalitățile sale.

Cercurile de progres se adaugă în pagina HTML cu <circle-progress></circle-progress>, unde se pot și seta proprietățile, cum ar fi procentul, care este valoarea afișată, raza cercului și afișarea animației. Proprietățile care se aplică peste tot sunt setate în NgModule, pentru a nu fi necesară adăugarea aceluiași lucru de mai multe ori. Valoarea afișată este procentajul valorilor nutriționale ale alimentelor din necesarul zilnic al clientului. Procentajul este calculat, apoi dat pentru a fi afișat. Această valoare este afișată și sub formatul unei bare de progres, care însă a fost creat folosind proprietăți de stil CSS. Pentru acest lucru se adaugă două elemente div, unul cu o dimensiune fixă de 100 pixeli și culoarea de background roșie și celălalt cu dimensiunea procentajul obținut și culoarea albastră. Cele două elemente sunt suprapuse, cel având culoarea albastră aflându-se deasupra. Astfel se obține aspectul de progress bar, în care se poate vedea cât din cantitatea pe care trebuie să o consume în acea zi clientul conține alimentul, pentru cantitatea introdusă.

Pentru graficele line chart, datele sunt date sub format JSON (JavaScript Object Notation) și sunt organizate în grupuri de name, value, name conținând data care este afișată pe axa x, iar value valoarea pe acea dată care este afișată pe axa y. Datele pot fi date doar ca și string și trebuie să fie puse în ordinea în care se dorește să fie afișate pe axă, deoarece ngx-charts le vede doar ca și stringuri și nu face o ordonare a lor.

### *5.5.3. Pagina de chat*

Pentru crearea paginii de chat se folosește API-ul Chatkit[11], care permite crearea acesteia într-un timp scurt, spre deosebire de crearea cu WebSockets. Pentru a comunica cu serverul, Chatkit oferă un fișier server.js care trebuie adăugat la proiect. Fișierul poate fi găsit pe pagina lor[12]. În fișier se află două endpointuri, „/users” și „/authenticate”. „/users” crează un nou utilizator în Chatkit cu un id „userId”, dacă utilizatorul nu există. „/authenticate” autentifică utilizatorul când încearcă să se conecteze la Chatkit.

Chatkit permite crearea de camere private. Membrii acestor camere pot trimite și vedea mesajele între ei. Un utilizator care nu este inclus în această cameră nu poate vedea mesajele sau trimite mesaje membrilor camerei, decât dacă este adăugat. Nu există o limită pentru numărul de camere private care pot fi create. Prin urmare, cum în pagina de chat a aplicației trebuie să comunice doar doi utilizatori, s-a decis crearea a câte o cameră privată pentru fiecare colaborare creată. La crearea unei noi colaborări, cei doi utilizatori care au stabilit colaborarea sunt adăugați în Chatkit și o nouă cameră privată este creată, unde cei doi utilizatori sunt adăugați. Fiecare cameră privată are asociată un *id* pe baza căruia se caută mesajele din aceasta.

[10] D3.js: https://d3js.org/
[11] Chatkit: https://pusher.com/chatkit
[12] Chatkit, fișier server.js cu exemplu de folosire: https://pusher.com/tutorials/chatroom-angular-chatkit

Datele utilizatorului care se salvează în Chatkit sunt numele de utilizator și un id. Imaginea de profil nu este stocată cum aceasta este deja obținută și salvată în sesiune, de une poate fi obținută pentru a o afișa și în pagina de chat.

### *5.5.4. Prelucrarea separată a datelor*

Subsistemul Consumer, pe lângă crearea evaluărilor, determină și 100 cele mai consumate alimente pe care le salvează în baza de date, în documentul „top_food”. O instanță conține numele alimentului, numărul total de utilizatori care l-au consumat și data ultimei actualizări. Informațiile acestea pot fi utilizate pentru statistici și în dezvoltări ulterioare. De fiecare dată când un utilizator adaugă un nou aliment în jurnal, datele sunt trimise subsistemului Consumer. Mesajul primit include numele alimentului, cantitatea consumată și numărul de calorii și de macronutrienți. Deși în acest moment se folosește doar numele alimentului s-a ales trimiterea mai multor date care vor permite mai departe crearea unui algoritm mai complex, pentru a realiza mai multe statistici.

La primirea unui mesaj, subsistemul se uită în baza de date pentru a vedea dacă alimentul din mesaj există în documentul „top_food”. Dacă da, se incrementează numărul și se setează data la data curentă, după care obiectul este actualizat în baza de date. Dacă nu se găsește acest aliment, prima dată se verifică numărul de alimente din documente. În cazul în care nu depășește dimensiunea maximă de 100 de înregistrări se crează un nouă instanță cu numărul total setat la 1 care este salvată în document. Dacă este atinsă limita, se parcurge lista pentru a se vedea dacă există alimente care au numărul total mai mare de unu însă nu a mai fost actualizat de peste trei luni, sau numărul total este egal cu unu și nu a fost actualizat de o lună, deci alimentul nu a fost consumat de mult timp sau este consumat rar de puține persoane. La găsirea unui astfel de aliment, acesta este șters. După această parcurgere se verifică dacă de această dată există spațiu pentru a se adăuga noul aliment, caz în care este adăugat.

### *5.5.5. Punctarea alimentelor și sugerarea de alimente clientului*

Pe baza planului de nutriție creat, clientul poate vizualiza cât de bine a urmat planul în ultimele șapte zile și în cazul în care nu a consumat suficiente calorii poate primi o recomandare cu alimente ce ar fi putut să mai consume pentru a învăța în timp să își atingă scopul.

Fiecare aliment salvat în baza de date are asignate punctaje pentru carbohidrați, proteine și grăsimi. Punctajele posibile sunt 1, 2, 3, 5, 8, 13. Un punctaj mic înseamnă că alimentul respectiv are o cantitate mică de macro respectiv, iar un punctaj mare o cantitate mare. S-a ales folosirea numerelor din șirul lui Fibonacci deoarece este mai ușor de determinat care valoare să se asigneze. Ceea ce se realizează este o mapare a valorilor macro a alimentului la șase valori pentru a putea mai ușor determina alimentele care ar putea să le consume clientul pentru a atinge necesarul zilnic.

Pentru a crea o listă cu alimente propuse, se iau din baza de date evaluările realizate de subsistemul Consumer pentru ultimele șapte zile. Valorile acestora sunt

însumate, apoi scăzute din totalul care trebuia consumat pe acele zile, deoarece se dorește să se determine cât ar mai fi trebuit să consume clientul pe durata unei săptămâni. Valorile obținute sunt distribuite pe cele șapte zile, obținându-se cât ar mai fi trebuit să consume clientul zilnic pentru a-și fi atins scopul. La fel ca și la punctarea alimentelor, valorile obținute sunt mapate la cele șase valori. Pentru fiecare macro în parte, se verifică dacă valoarea este mare, ceea ce indică că clientul este cu mult sub recomandare pentru acel macro. De exemplu, pentru proteine, dacă valoarea obținută este 8 înseamnă că clientul trebuia să mai consume cel puțin 20 grame de proteine. În cazul unei valori mari se caută în baza de date alimente care să aibă punctajul identic cu cel al valorii sau apropiate, având grijă să nu se depășească numărul de calorii necesare prin consumarea alimentului. În căutare se mai ia în considerare și gramajul care trebuie consumat, acesta fiind afișat în sugerare.

Pentru ca clientul să vadă vizual cât de mult a respectat sau nu planul, se afișează două diagrame, una pentru calorii și cealaltă pentru macro. Valorile puse în diagrame sunt diferențele obținute din scăderea evaluărilor însumate din totalul necesar pe cele șapte zile. Zero înseamnă că s-a consumat exact cât era recomandarea, o valoare negativă înseamnă că nu s-a consumat suficient, iar o valoare pozitivă faptul că s-a depășit valoarea recomandată.

Pe lângă diagrame se afișează un mesaj personalizat care specifică textual clientului care macro au fost consumați în cantitatea recomandată și care nu.

### *5.5.6. Adăugarea unui aliment nou în lista de alimente*

Pașii de adăugare a unui aliment în listă de către un antrenor/nutriționist sunt asemănători cu cei pentru cazul clientului. În ambele cazuri se folosește aceeași componentă, FoodGenericComponent. Ambele tipuri de utilizatori sunt autorizate să apeleze cele două metode POST din cadrul acestei operații. Pentru a se ști pentru care client să se salveze alimentul, în cazul antrenorului/nutriționistului se va da clientul pe care l-a selectat pentru a adăuga un aliment, iar în cazul clientului se vor da datele sale salvate în serviciu.

După verificarea dimensiunii numelui produsului, se face un apel pentru a fi salvat în baza de date în cazul în care el a fost luat din API, nefiind găsit în baza de date a aplicației. S-a ales crearea unei metode separate de verificare a existenței produsului în baza de date și de adăugare al acestuia la ea deoarece pe viitor, dacă se dorește eliminarea căutării în API și folosirea doar a bazei de date a aplicației, va fi necesară modificarea codului doar în fron-end prin ștergerea liniei de cod care face apelul de adăugare. În back-end va fi necesară eliminarea metodelor de adăugare, acestea nemaifiind folosite, dar nu va trebui să se modifice o metodă. Dacă decizia de adăugare s-ar realiza la căutare ar fi necesară modificarea metodei de adăugare și retestarea pentru a se asigura că modificarea nu a afectat funcționalitatea. Un alt motiv este faptul că nu se știe exact ce produs este căutat, utilizatorul putând să folosească un grup de cuvinte care are ca rezultat mai multe produse

Primul pas este de a se verifica dimensiunea numelui produsului. În cazul în care aceasta depășește limita permisă, execuția se va opri și se va afișa un mesaj de avertizare

utilizatorului, care îi va transmite problema. Dacă nu se specifică nici un nume, deci intrarea este nulă sau „”, butonul de adăugare este dezactivat.

Următorul pas este de a salva produsul în lista personală a clientului. Pentru aceasta se face un apel POST în care se dă alimentul care trebuie adăugat. Controlerul apelează metoda „create” din PersonalFoodService. Aceasta se asigură că datele numerice în virgulă mobilă au o singură precizie, prin rotunjirea lor. Datele primite sunt convertite la modelul corespunzător. Tabelul personal_food este în relație many to one cu tabelul user_tabel, astfel modelul corespunzător tabelului trebuie să aibă setat utilizatorul. Astfel, se caută și se setează utilizatorul. Mai departe se salvează alimentul în baza de date prin apelul metodei *save()* din PersonalFoodRepository.

În caz de succes se returnează alimentul salvat, iar în front-end utilizatorul este redirecționat în pagina cu lista de alimente personale pentru a putea vizualiza noul aliment adăugat. Redirecționarea se face pe baza tipului de utilizator.

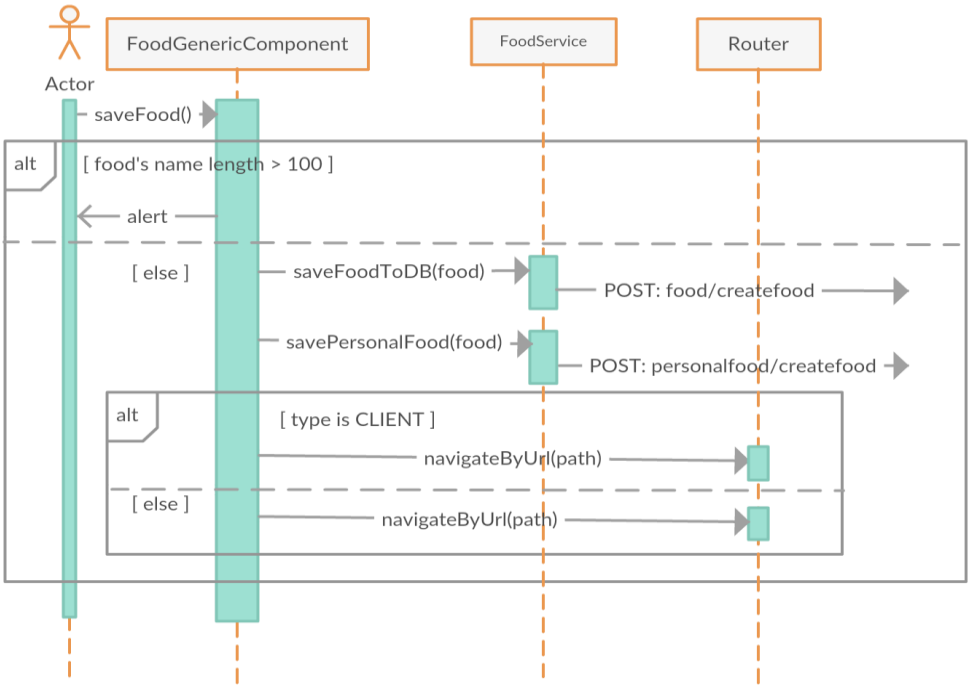



Figura 5.7 Diagrama secvențială pentru adăugarea unui aliment în listă, front-end

În figurile 5.7. și 5.8 sunt perzentate diagramele secvențiale pentru adăugarea alimentului în lista personală a clientului pe partea de fron-end respectiv back-end. Actorul poate fi clientul sau antrenorul/nutriționistul.

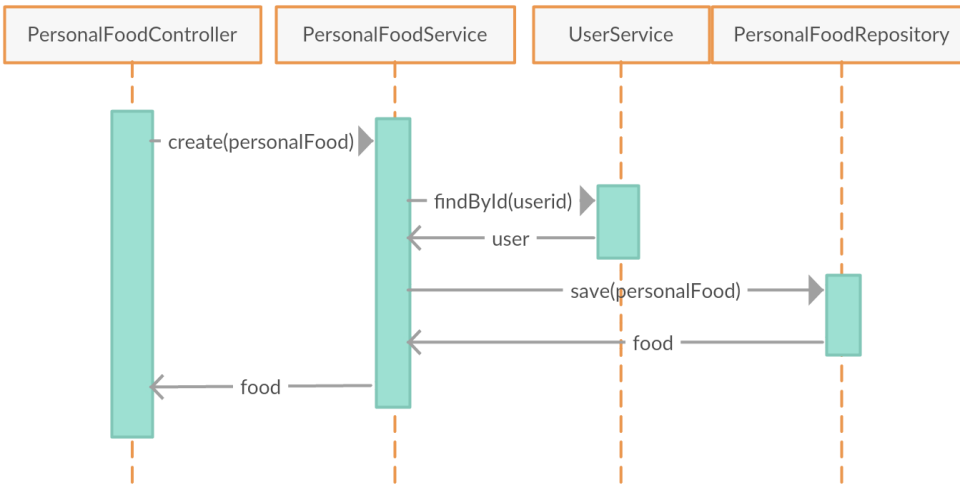



Figura 5.8 Diagrama secvențială pentru adăugarea unui aliment în listă, back-end

### *5.5.7. Evaluarea consumului*

Când clientul apasă butonul de adăugare în jurnal, se apelează metoda „add()” din componenta generică MealPlanGenericComponent. Se folosește o componentă generică deoarece și antrenorul/nutriționistul pot vizualiza, adăuga și șterge alimente din plan și deci apare cod duplicat.

La apelul metodei „add()” se face un apel POST prin serviciul MealPlanService, așteptându-se înapoi un răspuns de succes. În caz de succes (răspuns 200), clientul este redirecționat în pagina cu jurnal, Diay, unde poate vedea alimentele din plan care au fost consumate adăugate. În figura 5.9 se pot observa interacțiunile care au loc între clasele din Angular, în ordine secvențială.

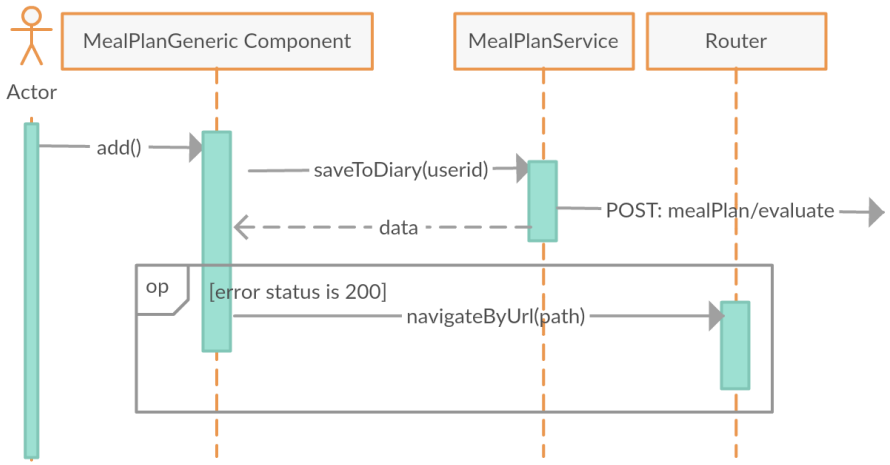



Figura 5.9 Diagrama secvențială trimitere plan nutrițional în jurnal, front-end

Pe partea de back-end, se primește cererea care este mapată la metoda corespunzătoare din MealPlanController, care apelează metoda de evaluare din MealPlanService „evaluate”. Această metodă obține toate intrările din planul de nutriție pe ziua curentă, le trimite subsistemului Consumer, care sunt setate ca și consumate sunt salvate în jurnal. Datele sunt șterge la final din tabelul meal_plan. Comunicarea dintre clasele implicate în acești pași este prezentată în figura 5.10.

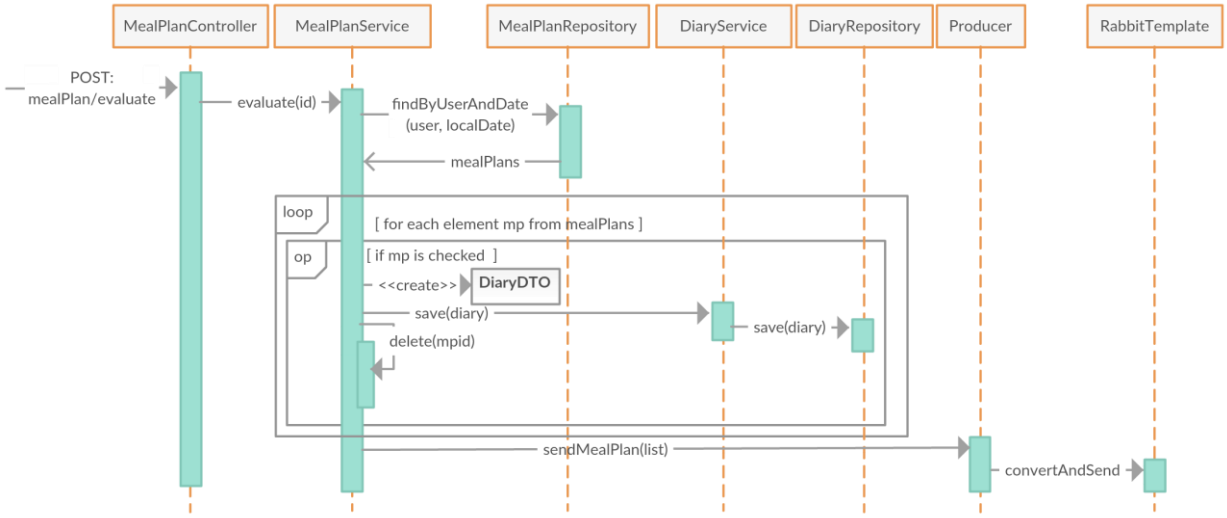



Figura 5.10 Diagrama secvențială trimitere plan nutrițional în jurnal, back-end

Subsistemul Consumer ascultă pentru primirea mesajelor pe coadă. La detectarea un mesaj, acesta ia mesajul respectiv din care extrage lista cu planuri. În primul pas verifică dimensiunea listei pentru a se asigura dacă nu este nulă. Mai departe, dacă există elemente în listă, se calculează suma totală de calorii și macro care trebuia consumată și suma de calorii și macro consumate, acestea fiind a alimentelor care au fost setate ca fiind consumate. Aplicația permite după trimiterea datelor spre jurnal să se mai adauge în plan alte alimente care după aceea să fie trimise de asemenea în jurnal. Astfel se poate ca în baza de date să existe deja o evaluare pe ziua curentă. Subsistemul va căuta în baza de date dacă există o evaluare pe ziua curentă. Dacă există, cele două sume obținute sunt însumate la datele din evaluarea găsită, după care este actualizată. În caz contrar, se crează o nouă evaluare unde se pun sumele, după care aceasta este salvată în baza de date. Interacțiunile dintre clase și metodele apelate în acest proces sunt ilustrate în figura 5.11.

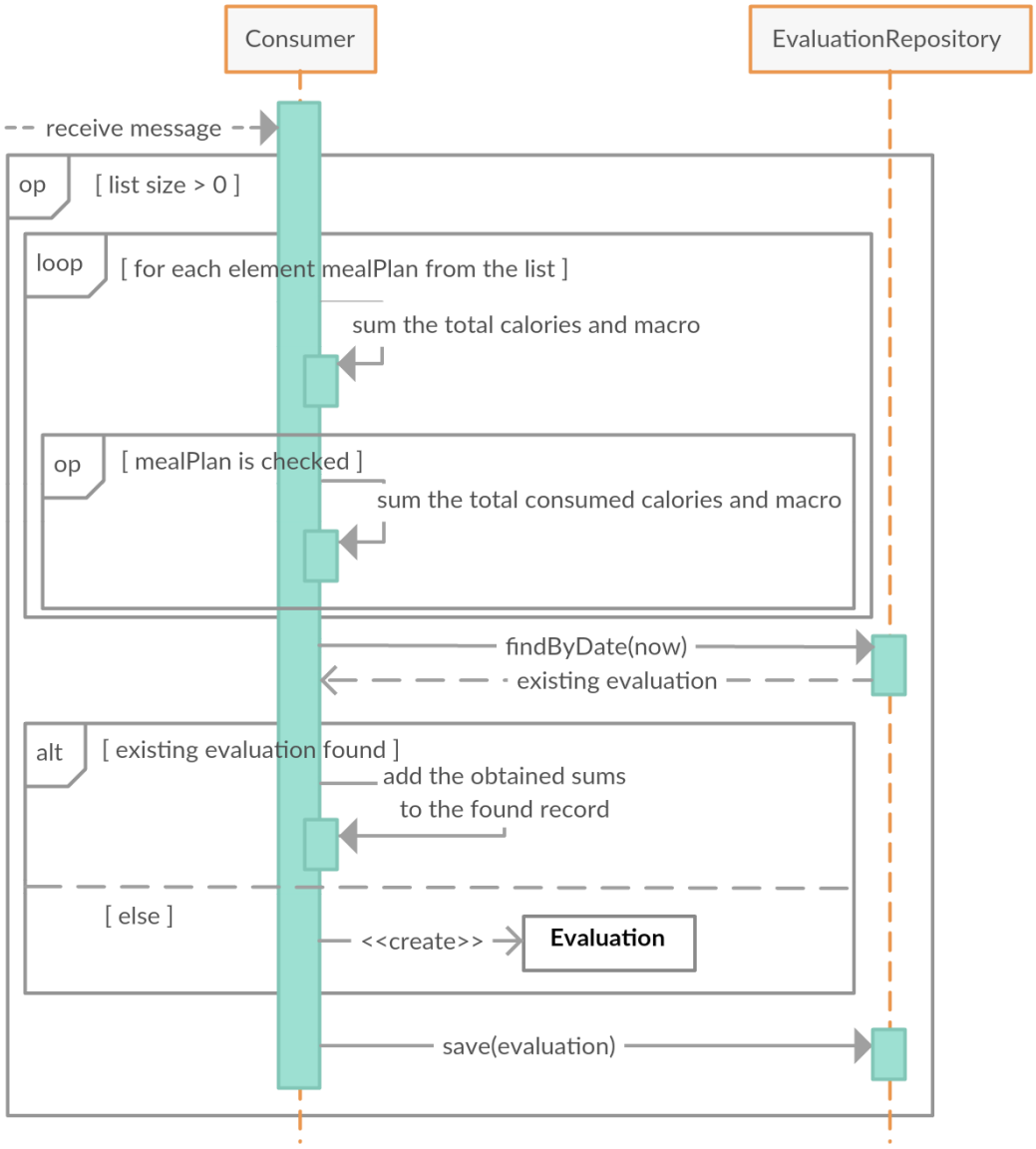



Figura 5.11 Diagrama secvențială procesare evaluare de subsistemul Consumer

## 5.6. Proiectarea datelor

Aplicația utilizează două baze de date, MySQL și MongoDB. Datele nestructurate, care pot fi în număr mare și trebuie să fie accesate rapid sunt puse în MongoDB. Prin dernomalizarea datelor se îmbunătățește performanța de citire, chiar dacă acest lucru duce la scăderea performanței de scriere. Datele care nu vor crește mult în timp, trebuie să respecte proprietățile ACID sau sunt supuse unor interogări mai complexe care necesită joinuri, sunt puse în MySQL.

Pentru configurarea bazelor de date în cadrul Spring s-a folosit un fișier application.properties, unde sunt setate baza de date, credențialele proprietarului bazei de

date și driverul necesar ambelor baze de date. Pentru MySQL s-a făcut o configurare JPA, în clasa JPAConfig.

Maparea entităților și persistența datelor este realizată cu Hibernate, prin intermediul JDBC. Interfața JpaRepository este folosită pentru a implementa nivelul de acces al datelor, oferind metode de acces la baza de date relațională. Pentru baza de date MongoDB se folosește interfața MongoRepository.

O adnotare folosită este @MappedSuperclass, care îi specifică lui JPA să includă proprietățile de persistență a clasei de bază ca și cum ar fi fost declarate de clasa child care extinde superclasa adnotată cu @MappedSuperclass. Cu toate acestea, moștenirea este vizibilă doar în OOP, deoarece, din perspectiva bazei de date, nu există nici o indicație a clasei de bază. Doar entitatea din clasa child va avea o tabelă mapată asociată.

### *5.6.1. MySQL*

Contrângerile folosite pe datele din baza de date relațională sunt:

- NOT NULL – coloana nu poate primi valoarea NULL;
- UNIQUE – nu permite valori duplicate pe coloană;
- DEFAULT – dacă nici o valoare nu este dată coloanei, atunci aceasta primește valoarea specificată la DEFAULT;
- PRIMARY KEY – impune tabelului să accepte date unice pentru coloană și crează un index unic pentru accesarea rapidă a tabelului;
- FOREIGN KEY – crează o legătură între două tabele printr-o coloană specifică a ambelor tabele.

În proiectarea bazei de date a fost urmărită normalizarea acesteia. În continuare sunt descrise formele normale pe care baza de date le respectă.

Forma normală 1 (FN1) impune ca fiecare câmp din tabel să dețină o valoare atomică și fiecare înregistrare să fie definită astfel încât să poată fi identificată unic folosind cheia primară. Constrângerile sunt îndeplinite fiecare atribut primind doar o valoare atomică și fiecare înregistrare fiind unică prin utilizarea cheii primare.

Forma normală 2 (FN2) necesită îndeplinirea FN1 și impune ca atributele să nu fie dependente parțial de cheia primară. În cazul unei dependențe parțiale este necesară separarea în tabele. Cum toate tabelele din baza de date au cheia primară formată dintr-un singur atribut, FN2 este îndeplinită.

Forma normală 3 (FN3) impune îndeplinirea FN2 și să nu existe dependențe tranzitive. Fiecare atribut care nu este cheie nu depinde de un alt antribut care nu este cheie care depinde în întregime de cheia primară.

Forma Normală Boyce-Codd impune îndeplinirea FN3 și ca atributele să depindă de o cheie în întregime.

În figura 5.12 este reprezentată diagrama bazei de date MySQL.

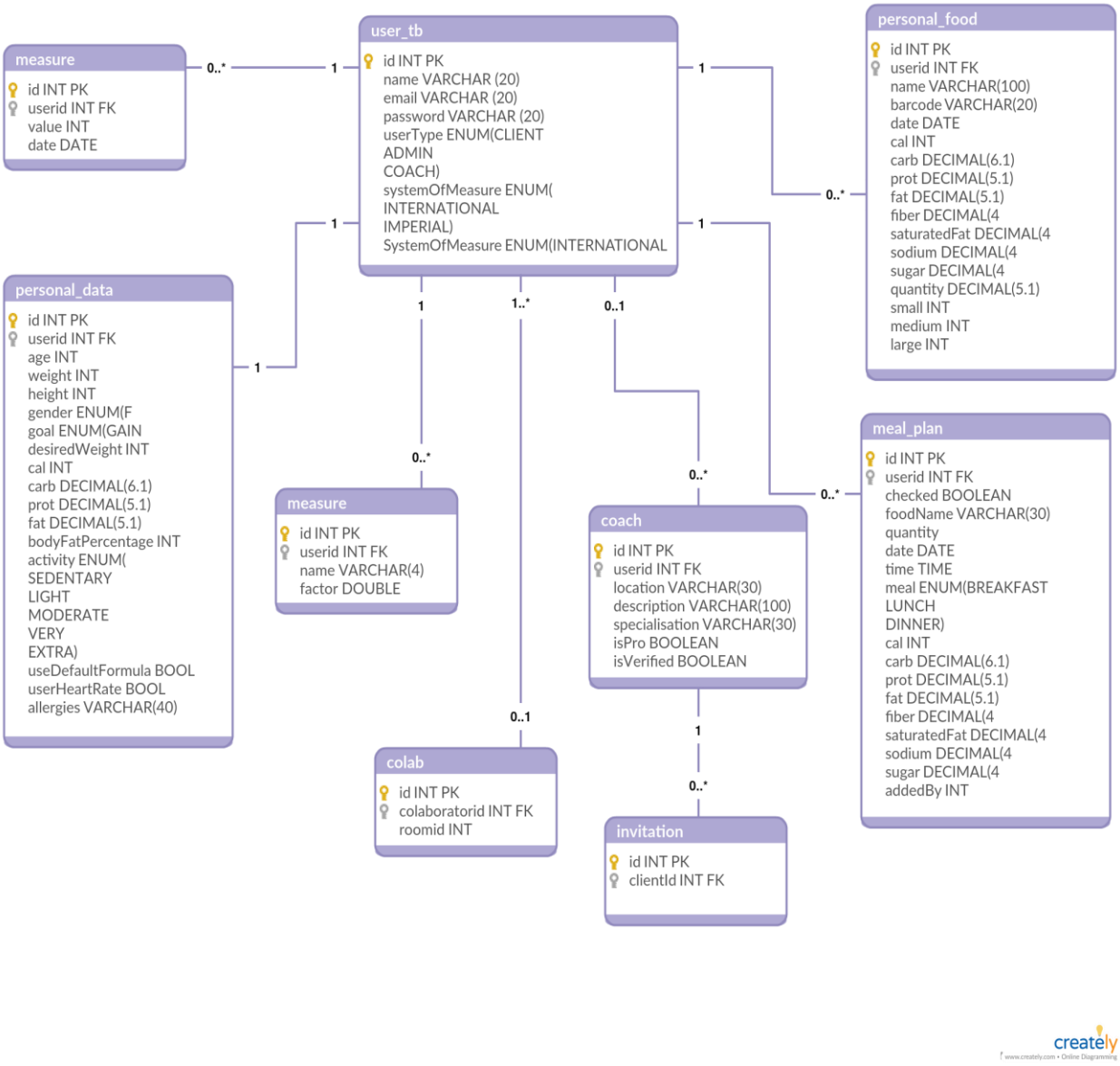



Figura 5.12 Diagrama bazei de date MySQL

În continuare sunt descrise tabelele. Fiecare tabel este identificat în mod unic printr-o cheie primară numerică, numită “id”. Data este de tipul LocalDate pentru toate atributele care stochează o dată. Cantitatea ("quantity") și valorile calorice („cal”), macro („carb”, „prot”, „fat”) și nutriționale („fiber”, „sugar”, „sodium”, „saturatedFat”) au tipul Double. Caloriile au precizia 5, iar restul 4.

Tabelul *“coach_data”* conține informațiile antrenorilor/nutriționiștilor, acestea fiind locația în care locuiește antrenorul/nutriționistul, descrierea acestuia și specializarea pe care o are. Pe lângă acestea se mai țin două valori booleane “isPro” și “isVerified” care specifică dacă antrenorul/nutriționistul este verificat ca fiind o sursă de încredere, respectiv dacă este un antrenor/nutriționist cu credențiale care îi permit evaluarea celorlalți antrenori/nutriționiști pentru a putea fi setați ca fiind verificați.

Tabelul *"colab"* conține colaborările nutriționistului/antrenorului pe care le are cu clienții. Aceasta conține id-ul colaboratorului, adică clientul cu care colaborează și id-ul camerei private din Chatkit.

Tabelul *„invitation"* conține invitațiile de a colabora trimise de clienți antrenorilor/nutriționiștilor, conținând un mesaj text scris de client.

Tabelul *"meal_plan"* conține planurile de alimentație ale clientului, adăugate de client sau antrenor/nutriționist. Pentru a ști cine a adăugat înregistrarea, se introduce atributul "addedBy". Restul informațiile includ numele alimentului ("foodName"), pentru care dată din plan este adăugat alimentul ("date"), timpul la care alimentul este setat ca fiind consumat de către client ("time"), cantitatea care trebuie consumată ("quantity"), masa la care este consumat ("meal") și valorile calorice („cal"), macro („carb", „prot", „fat") și nutriționale („fiber", „sugar", „sodium", „saturatedFat") ale alimentului pentru cantitatea introdusă. Atributul boolean „checked" este folosit pentru a indica dacă clientul a consumat alimentul, informație folosită în evaluarea clientului și pentru a se ști dacă alimentul trebuie sau nu să fie adăugat în jurnalul alimentelor consumate.

Tabelul *„measure"* conține factorii de convertire a unităților de măsură în sistemul metric. Stringul „name" reprezintă numele unității de măsură din care se convertește și are dimensiunea maximă de 10 caractere. Numărul „factor" reprezintă factorul cu care trebuie înmulțită valoarea pentru a se converti în sistemul metric și este de tipul Double.

Tabelul *„personal_data"* conține datele personale ale clienților. Aici sunt stocate informațiile necesare pentru a calcula cantitatea de calorii și macro pe care clientul trebuie să le consume zilnic, acestea fiind vârsta, înălțimea, genul, scopul, greutatea pe care dorește să o atingă clientul, procentajul de grăsime și nivelul de activitatea al clientului. Pentru a se ști care dintre cele două formule folosite de aplicație să se utilizeze în calcul, se folosește atributul boolean „useDefaultFormula". Setarea la „true" indică utilizarea formulei implicite, altfel se utilizează formula Katch-McArdle. Pentru a se ști dacă să se folosească ritmul cardiac în calcularea caloriilor arse, se folosește atributul boolean "useHeartRate". Setarea la "true" indică utilizarea ritmului cardiac. Alergiile clientului sunt salvate în atributul "allergies" sub format String. Mesele pe care le are clientul sunt salvate sub formă de listă în Java, fiind mapate în baza de date ca obiecte incorporabile, adică obiecte care nu sunt entități, ci tipuri simple, în acest caz fiind String. Nivelul de activitate ale clientului, scopul și genul acestuia pot lua doar valori predefinite, salvate în clasa Enum. Celelalte atribute ale entității reprezintă valorile obținute în urma aplicării formulei pentru determinarea cantității de calorii și a macro necesari zilnic. Tabelul se află în relație One-toOne cu tabelul "user", conținând id-ul din tabelul "user" ca și cheie străină.

Tabelul *„personal_food"* conține alimentele salvate în lista personală de alimente ale clienților. Informațiile stocate sunt numele și codul de bare al alimentului, valorile calorice, macro și nutriționale ale alimentului. Pentru alimentele care pot fi încadrate în dimensiunile mic, mediu sau mare, sunt introduse atributele numerice opționale, „small",

„medium” și „large”, care reprezintă cantitatea alimentului pentru dimensiunea respectivă.

Tabelul *„user_app”* conține datele utilizatorilor aplicației, acestea fiind numele de utilizator, emailul, parola, tipul clientului și sistemul de măsură. Sistemul de măsură și tipul de utilizator pot lua doar valori predefinite, din clasa Enum. Tabelul are patru relații One-to-Many cu „foods”, „mealPlans”, „colabs” și „invitations”. Deci un utilizator poate avea asociate zero, una sau mai multe alimente, planuri de alimentație, colaborări și invitații.

Adnotările folosite pentru a mapa entitățile sunt:

- @Entity – marchează clasa ca fiind o entitate
- @Table – specifică numele tabelului din baza de date la care este mapată entitatea; fără această adnotare se mapează numele clasei;
- @Id – marchează câmpul ca fiind cheie primară;
- @Column – marchează câmpul ca fiind un atribut din tabel;
- @Enumerated – este folosită pentru a converti datele enumerate din Java în reprezentarea bazei de date și invers;
- @ElementCollection - mapează obiectele care nu sunt entități (în acest proiect, mesele pe care le iau clienții);
- @OneToMany – este folosită pentru a reprezenta o relație One-to-Many; proprietatea “mappedBy” este folosită pentru a specifica care variabilă este folosită pentru a reprezenta clasa părinte în clasa child;
- @ManyToOne – este folosită pentru a reprezenta o relație Many-to-One.
- @JoinColumn – referențiază coloana mapată;
- @OneToOne – este folosită pentru a reprezenta o relație One-to-One;
- @MapsId – este utilizat în relațiile @ManyToOne și @OneToOne pentru a spune lui Hibernate să folosească cheia străină unei entități asociate ca cheie primară.

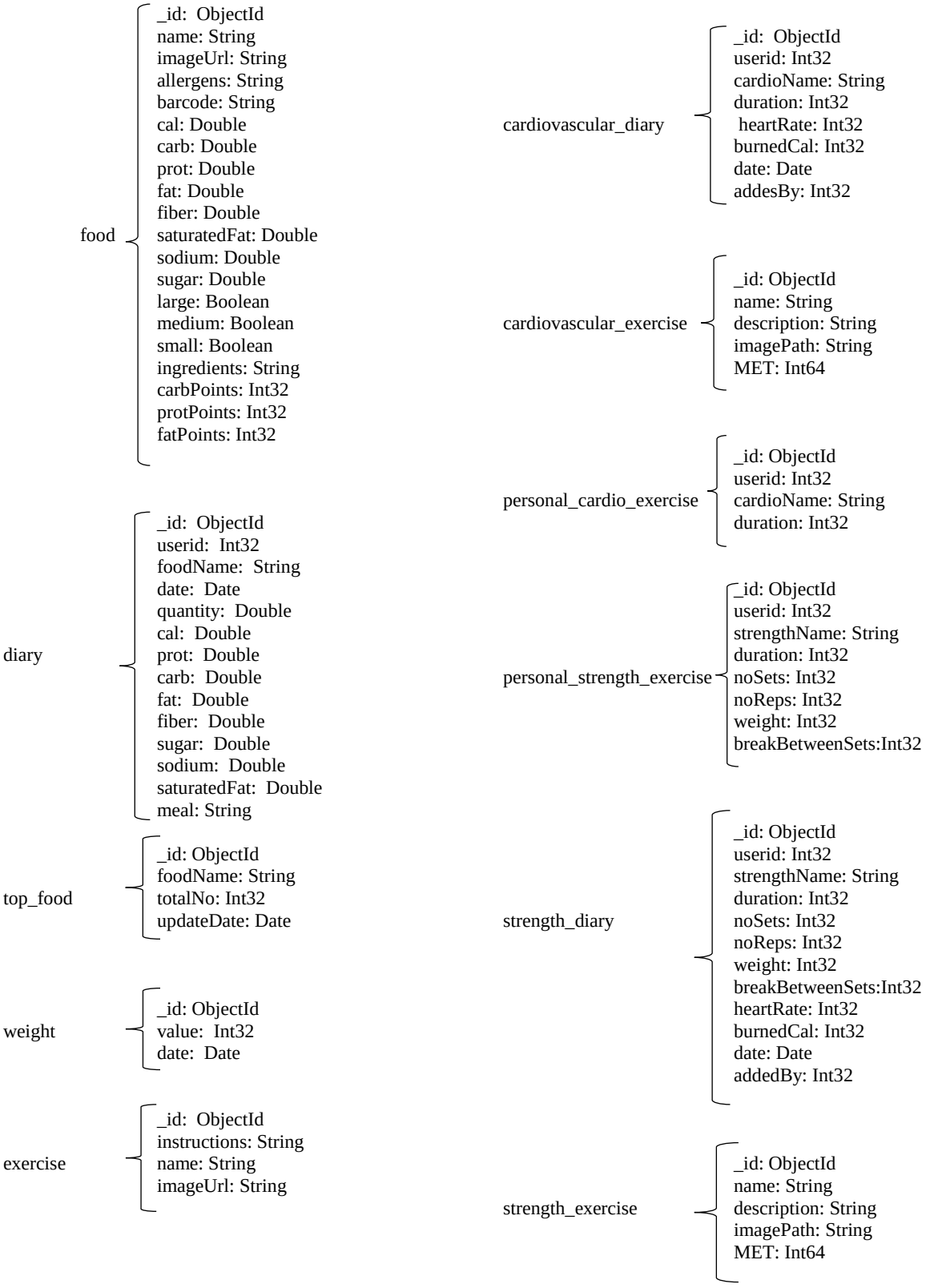
### *5.6.2. MongoDB*

În MongoDB datele sunt stocate sub format JSON în documente. Structurarea bazei de date din MongoDB este flat, fără nici o imbricare. S-a ales această organizare deoarece nici un document nu are nevoie de alte documente la extragerea conținutului de date.

Adnotările folosite pentru a mapa documentele sunt:

- @Document – specifică numele documentului din baza de date la care este mapată clasa;
- @Id – marchează câmpul ca fiind cheie primară.

Documente din baza de date MongoDB:



Documentul *„cardiovascular_diary”* conține date despre exercițiile cardiovasculare ale clienților, afișate în jurnalul cu exerciții fizice. Acesta conține durata exercițiului, ritmul cardiac al clientul pe parcursul efectuării exercițiului, numărul de calorii care au fost arse realizând exercițiul, durata și data în care trebuie realizat acest exercițiu. Pentru exercițiile fizice, jurnalul este combinat cu planul de antrenament, unde clienții își adaugă exercițiile, urmând ca după ce le-au realizat să completeze ritmul cardiac pentru a obține caloriile arse. Exercițiile pot fi adăugate de antrenori/nutriționiști, astfel, pentru a se ști cine l-a adăugat s-a introdus atributul „addedBy”.

Documentul *„cardiovascular_exercise”* conține date despre exercițiile cardiovasculare înregistrate în aplicație, pe care clienții pot să le adauge în lista proprie de exerciții cardiovasculare sau pe care antrenorii/nutriționiștii le pot adăuga în lista unui client. Acesta conține numele exercițiului, o descriere a acestuia, calea spre imaginea care reprezintă exercițiul și valoarea MET.

Documentul *„diary”* reprezintă jurnalul de alimentație al clientului, conținând alimentele pe care clientul le-a consumat. Datele stocate sunt data în care a fost consumat alimentul („date”), numele alimentului („foodName”), cantitatea consumată („quantity”), masa în cadrul căreia a fost consumat alimentul („meal”) și valorile calorice („cal”), macro („carb”, „prot”, „fat”) și nutriționale („fiber”, „sugar”, „sodium”, „saturatedFat”) ale alimentului pe cantitatea consumată.

Documentul *„evaluation”* conține evaluările clientului realizate de al doilea subsistem, Consumer. Informațiile stocate sunt data pe care este evaluarea („date”), totalul caloric și macro, al alimetelor atât consumate cât și neconsumate din planul nutriționat al clientului („calTotal”, „carbTotal”, „protTotal”, „fatTotal”) și totalul caloric și macro, al alimetelor consumate din planul nutriționat al clientului („calConsumed”, „carbConsumed”, „protConsumed”, „fatConsumed”).

Documentul *„food”* conține alimentele salvate ale aplicației. Informațiile stocate sunt numele, calea spre imagine și codul de bare al alimentului, valorile calorice, macro și nutriționale ale alimentului, ingredientele, alergenii conținuți în aliment. Unele alimente pot fi încadrate în trei categorii de dimensiuni (mic, mediu, mare), cum ar fi mărul sau banana, pe baza cărora li se poate estima cantitatea. Pentru aceste alimente, sunt introduse atributele numerice opționale, „small”, „medium” și „large”, care reprezintă cantitatea alimentului pentru dimensiunea respectivă.

Documentul *„personal_cardio_exercise”* conține exercițiile cardio personale ale clientului. Informațiile ținute în document sunt numele exercițiului („cardioName”) și durata exercițiului care are salvată ultima valoare introdusă de client.

Documentul „personal_strength_exercise” conține exercițiile de forță personale ale clientului. Informațiile ținute în document sunt numele exercițiului („strengthName”), durata exercițiului care are salvată ultima valoare introdusă de client („duration”), pauza dintre seturi („breakBetweenSets”), numărul de seturi („noSets”), numărul de repetiții („reps”) și greutatea greutăților folosite.

Documentul *„profile”* conține imaginile de profil ale clienților.

Documentul *„strength_diary”* conține date despre exercițiile de forță ale clienținor, afișate în jurnalul cu exerciții fizice. Acesta conține durata exercițiului, numărul de seturi și de repetiții, greutatea cu care se lucrează, pauza dintre seturi, ritmul cardiac, numărul de calorii care sunt arse realizând exercițiul și data în care trebuie realizat acest exercițiu. Exercițiile pot fi adăugate de antrenori/nutriționiști, atributul „addedBy” indicând cine le-a adăugat.

Documentul *„strength_exercise”* conține date despre exercițiile de forță înregistrate în aplicație, pe care clienții pot să le adauge în lista proprie de exerciții de forță sau pe care antrenorii/nutriționiștii le pot adăuga în lista unui client. Acesta conține numele exercițiului, o descriere a acestuia, calea spre imaginea care reprezintă exercițiul și valoarea MET.

Documentul *„weight”* conține grutățile clienților înregistrate zilnic de aceștia. Aici sunt stocate data înregistrării și greutatea măsurată de client.

# Capitolul 6. Testare şi Validare

Există două metode de testare, manuală și automată. În testarea manuală testele sunt executate manual. În timpul procesului se execută cazurile de test și se generează rapoarte de test fără ajutorul unui software de testare. În testarea automată se scriu coduri de test pentru a automatiza execuția testelor. Testele sunt scrise cu ajutorul unui software care îl și execută pentru a compara rezultatul cu cel așteptat.

**Pentru front-end** s-a folosit testarea manuală, care permite observarea cu ușurință a defectelor care pot apărea, prin analizarea cererilor și a răspunsurilor primite în consola din browser.

În tabelul 6.1 sunt prezentate cazurile de test pentru logarea unui utilizator în sistem. S-a utilizat notarea prescurtată pentru scrierea și interpretarea mai rapidă a cazurilor de test.

Tabel 6.1 Cazuri de test petntru operația de logare

| **Cazuri de test** | **Pași** | **Rezultate așteptate** | **Status (not exec /blocked/fail/pass)** |
|---|---|---|---|
| | | | |
| **Caz de test:** | | | |
| **1** | Verifică următoarele combinații de email și parolă.<br>Notă: V înseamnă valid, I înseamnă invalid, iar blank înseamnă lăsarea intrării liberă. | Notă: E înseamnă mesaj de eroare, BE înseamnă deblocarea butonului de logare, BB înseamnă dezactivarea butonului de logare, iar P înseamnă redirecționare în pagina de profile, „Profile”. | |
| 1.1 | blank, blank | E | pass |
| 1.2 | blank, I | E | pass |
| 1.3 | blank, V | E | pass |
| 1.4 | I, blank | E | pass |
| 1.5 | V, blank | E | pass |
| 1.6 | I, I | E | pass |
| 1.7 | I, V | E | pass |
| 1.8 | V, I | E | pass |
| 1.9 | V, V | H | pass |

În figurile 6.1 și 6.2 se poate observa rezultatul testării a două din cazurile de testare din tabelul 6.1.

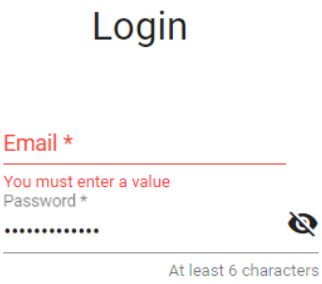



Figura 6.2 Screenshot pentru cazul de test 1.3

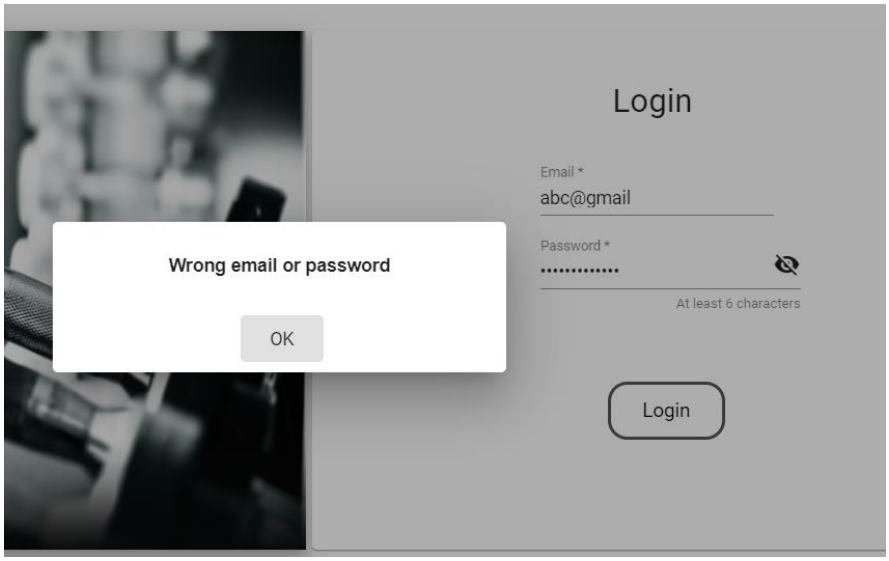



Figura 6.1 Screenshot pentru cazul de test 1.7

În tabelul 6.2 sunt prezentate cazurile de test pentru afișarea datelor nutriționale ale alimentelor din lista personală, din pagina „Food List”. În figurile 6.3 și 6.4 se pot observa efectele testării cazurilor 2.2 și 2.3.

Tabel 6.2Cazuri de test petntru afișarea datelor alimentelor din lista personală

| **Cazuri de test** | **Pași** | **Rezultate așteptate** | **Status (not exec /blocked/fail/pass)** |
|---|---|---|---|
| | | | |
| **Caz de test:** | | | |
| **2** | Verifică corectitudinea valorilor afișate în lista cu alimente personale | | |
| 2.1 | Selectează un aliment din listă. | Se afișează valorile pentru cantitafea implicită de 100g, care sunt identice cu valorile din baza de date. Procentele sunt corecte. | pass |
| 2.2 | Modifică cantitatea implicită la un număr cu două zecimale. | Valorile se modifică automat, reprezentând valorile pentru cantitatea introdusă. Numere sunt afișate cu o singură zecimală. | pass |
| 2.3 | Lasă intrarea pentru cantitate goală. | Valorile afișate sunt setate la zero. | pass |
| 2.4 | Modifică cantitatea la un număr negativ. | Valorile afișate sunt setate la zero. | pass |

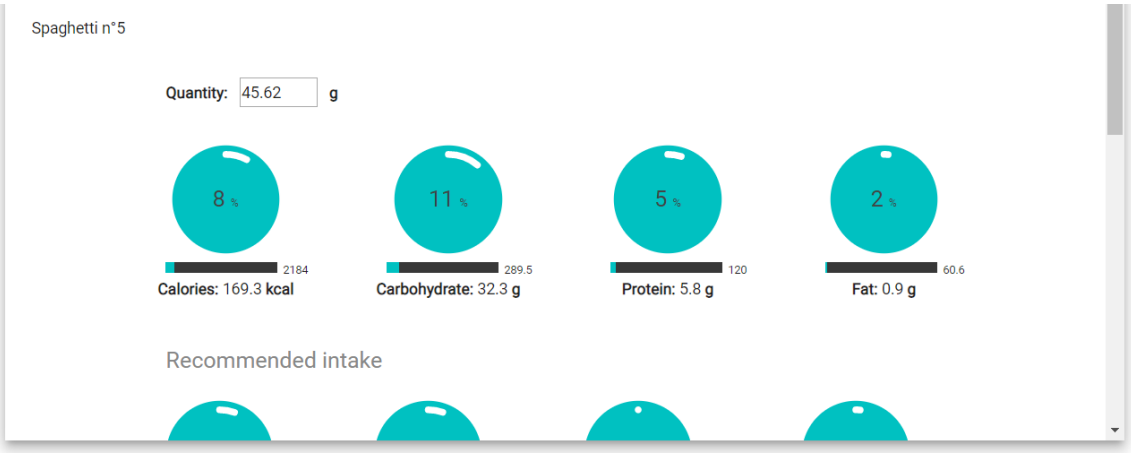



Figura 6.3 Screenshot pentru cazul de test 2.2

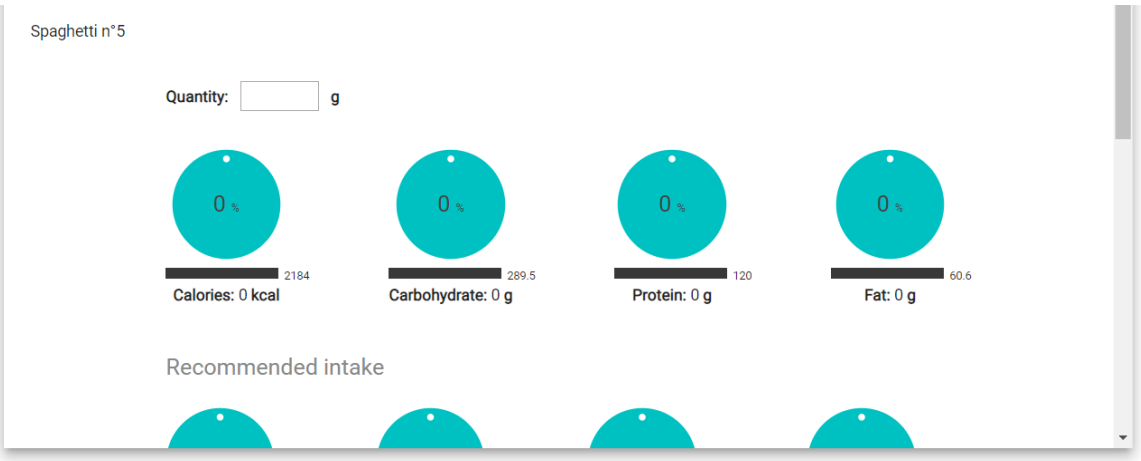



Figura 6.4 Screenshot pentru cazul de test 2.3

**Pentru back-end** s-a folosit testarea automată. Există mai multe tehnici de testare, în acest caz alegându-se testarea de integrare de tip White Box. În această tehnică se crează scenarii de test pentru a se vedea dacă componentele transferă datele între ele corect, în care trebuie să se cunoască starea internă a componentei care este testată. Principalul avantaj al acestei tehnici este posibilitatea testării codului pentru a detecta bug-uri. Scopul acestei testări este de a verifica dacă datele sunt transferate corect între nivelele sistemului.

Testarea s-a realizat folosind suportul oferit de Spring Boot și s-a realizat pentru fiecare nivel: repository, service și controller. Toate testele se află în pachetul „/src/test/java/com.finalproject.userportal”. Pentru a putea folosi suportul oferit de Spring Boot în testele Junit, este necesară utilizarea adnotării @RunWith(SpringRunner.class), care va oferi o punte între testarea oferită de Spring Boot și testele JUnit.

Pentru testarea claselor Repository este necesară utilizarea adnotării @DataJpaTest, care va înlocui orice sursă de date cu baza de date încorporată în memorie, H2, setează Hibernate și Spring Data, scanează entitățile și pornește logarea SQL pentru a se înregistra toate tranzacțiile și modificările bazei de date efectuate de tranzacții. Un alt lucru de care este nevoie este salvarea unor date în baza de date, care se poate realiza folosind TestEntityManager.

Un exemplu de test este testarea metodei *findByName(String)* a clasei FoodRepository. Testul se afă în clasa FoodRepositoryImplTest, unde metoda *whenFindByName_thenReturnFood()* crează un obiect, food, pe care îl persistă în baza de date folosind TestEntityManager, după care apelează metoda testată și verifică dacă numele alimentului din rezultatul obținut este identic cu cel al alimentului creat pentru a fi persistat, „food”.

Pentru testarea codului din clasele Service, deși acestea sunt dependente de Repository, nu trebuie să se știe cum este implementat nivelul de persistență. Acest lucru se poate realiza prin folosirea datelor mocking.

Un exemplu de test este testarea metodei *findByName* a serviciului FoodService, pentru a se vedea dacă primește și structurează corect datele în JSON. Testul se află în clasa FoodServiceImplTest. Adnotarea @MokBean crează un mock pentru

FoodRepository care este folosit pentru a ocoli apelul spre FoodRepository. Metoda *setUp()* este executată înainte de a se executa testul și are rolul de a crea și returna o listă cu un obiect Food la apelul metodei *findAll()* din FoodRepository. Metoda de test, adnotată cu @Test, apelează metoda de căutare a unui aliment după nume, *searchByName(),* extrage numele alimentului din rezultat și verifică dacă acesta este identic cu rezultatul așteptat, în acest caz „apple”.

Pentru testarea claselor Controller se folosește adnotarea @WebMvcTest pentru a configura automat infrastructura Spring MVC pentru test. Tot @WebMvcTest va oferi și clasa MockMvc pentru a putea crea cereri fără a fi necesară pornirea unui server HTTP. Adnotarea @MockBean este folosită pentru a mocui serviciile folosite de clasele Controller.

Un exemplu de test este testarea metodei *searchByName* a clasei FoodController, pentru a vedea dacă primește datele corect de la serviciu. Testul se află în clasa FoodRestControllerTest, unde este implementată metoda de testare *givenFoods_whenGetFoodsByName_thenReturnJsonObject().* Metoda mocuiește serviciul FoodService și specifică ca la apelul metodei *searchByName(String, Integer)* să se returneze un JSONObject creat de aceasta. În ultimul pas, folosind MockMvc, se face o cerere GET pentru metoda testată și se verifică corectitudinea rezultatului.

# Capitolul 7. Manual de Instalare si Utilizare

## 7.1. Instalare și rulare

### *7.1.1. Instalare resurse necesare*

Pentru rularea corectă a aplicației trebuie instalate și configurate următoarele instrumente software:

- **JDK** (Kit de dezvoltare Java) versiunea 1.8.0

  JDK este disponibil pe site-ul “https://www.oracle.com/technetwork/java/javase/downloads/jdk8downloads-2133151.html”. După descărcare, trebuie să se stabilească variabilele de environment JAVA_HOME și JRE_HOME. Pentru mașinile care rulează pe sistemul de operare Windows10, acest lucru se face prin accesarea “Start/Sistem Properties/Environment Variables/Sistem Variables”, unde se crează cele două variabile, specificând calea spre executabilul descărcat.
- **IntelliJ IDEA**

  Acesta a fost utilizat pentru a dezvolta aplicația și poate fi găsit pe site-ul “https://www.jetbrains.com/idea/download/#section=windows”.
- **Tomcat Server**

  Acesta este configurat automat folosind framework-ul Spring Boot.
- **MySQL**

  Este disponibil pe “https://dev.mysql.com/downloads/workbench/”. După descărcare se execută executabilul pentru a se instala. Următorul pas este de a se configura baza de date. Codul necesar care trebuie rulat în MySQL Workbench se află în aplicație, în pachetul resources.
- **MongoDB**

  Este disponibil pe “https://www.mongodb.com/download-center/community”. Acesta trebuie instalat prin rularea executabilului, după care trebuie configurată baza de date.
- **NodeJS**

  Este disponibil pe "https://nodejs.org/en/download/". După descărcare, acesta trebuie instalat prin rularea executabilului.
- **Angular 6**

  Se instalează rulând în terminalul NodeJs comanda “npm install -g @angular/cli”.
- **RabbitMQ**

Pentru a putea folosi serviciul oferit de RabbitMQ este necesară instalarea acestuia. RabbitMQ necesită și Erlang pentru Windows pentru a fi instalat. Pentru acest proiect s-a ales versiunea recomandată, 20.3.x. Instrucțiuni de instalare, precum și executabilele se pot găsi pe site-ul “https://www.rabbitmq.com/install-windows.html”. Servicul RabbitMQ se va porni automat după instalare.

### *7.1.2. Importarea proiectului*

Proiectul poate fi importat din IntelliJ. Din meniu se selectează “VCS | Checkout fron Version Control | Git”. În “Clone Repository”, se specifică URL-ul spre repository, acesta fiind “https://github.com/ProdanAndreea/Fit”. În căsuța “Directory” se specifică folderul unde să se salveze proiectul clonat. Se apasă “Clone” pentru crearea proiectului, după care “Yes” pentru confirmare.

### *7.1.3. Rularea proiectului*

Rularea se face, pe back-end, prin rularea calsei “UserPortalApplication”, a ambelelor subsisteme. Pe front-end, IntelliJ va detecta serverul Angular, iar în proiect este configurată comanda care să se execute la rulare, astfel rularea se poate face simplu prin selectarea din meniu a “Run | Run ‘Angular CLI Server’".

Pentru a putea folosi API-ul Chatkit este necesară rularea fișierului server.js, aflat în documentul proiectului.

## 7.2. Manual de utilizare

### 1. Înregistrare

Utilizatorul poate să își creeze un cont nou în aplicație prin apăsarea link-ului “Register” din pagina principală de logare. Utilizatorul va fi redirecționat în prima pagină de înregistrare, unde trebuie să introducă numele de utilizator, adresa de mail, parola care dorește să o folosească pentru logare și tipul de utilizator care dorește să fie în aplicație, CLIENT pentru client și COACH pentru antrenor/nutriționist. Mai departe, în funcție de tipul de utilizator ales, utilizatorul va fi redirecționat în altă pagină unde să completeze datele adiționale corespunzătoare tipului de utilizator ales. în figura 7.1 se pot observa cele două pagini de înregistrare care trebuie completate în cazul clientului. După completarea tuturor pașilor, se va afișa un mesaj în care utilizatorul este informat faptul că un mail de confirmare a fost trimis la adresa introdusă de acesta. Mail-ul conține un link de confirmare. La accesarea linkului specific, utilizatorul va fi înregistrat cu succes în aplicație.

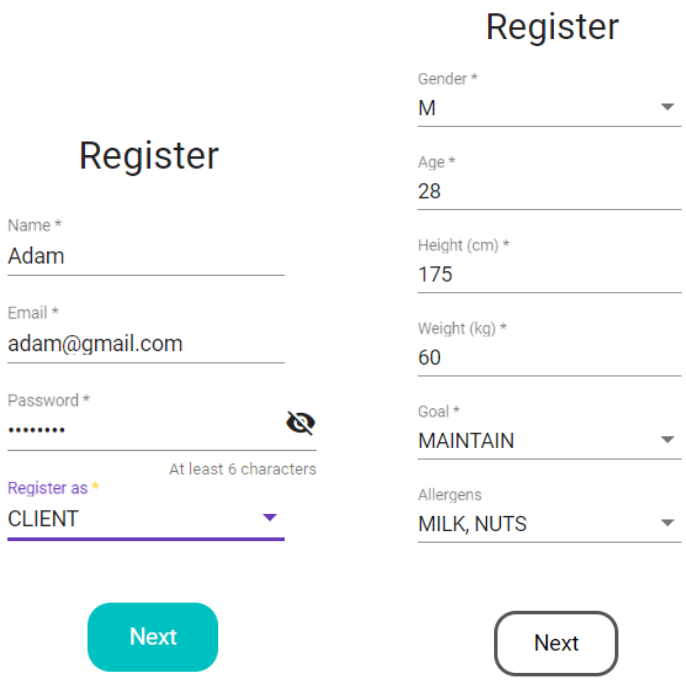



Figura 7.1 Screenshot-uri înregistrare în aplicație ca și client

## 2. Login

Pentru înregistrare, utilizatorul trebuie să introducă adresa de mail și parola cu care s-a înregistrat în aplicație. În momentul introducerii ambelor date sub format corect, butonul pentru logare "Login" este afișat, ca și în exemplul din figura 7.2. La apăsarea butonului de logare, dacă numele de utilizator și parola sunt ale unui utilizator existent, aplicația va redirecționa utilizatorul în pagina de profil a utilizatorului, care este pagina principală a aplicației.

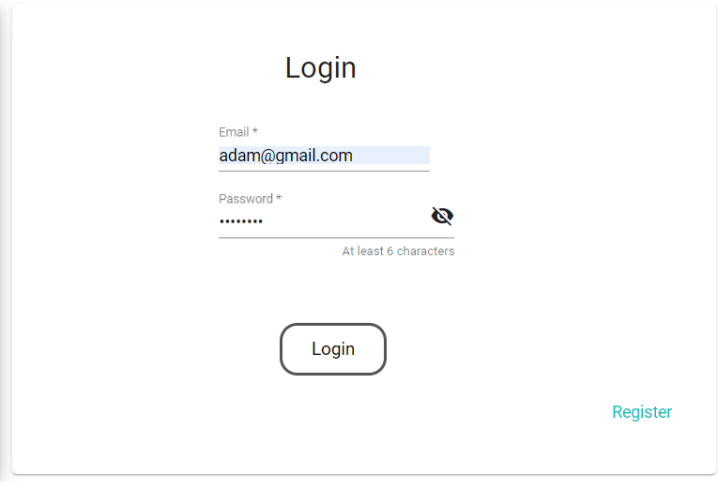



Figura 7.2 Screenshot logare

### 3. Editare profil

Atât clientul cât și antrenorul/nutriționistul pot să își editeze datele personale în pagina “Profile”. În cazul clientului, acesta poate selecta care dintre cele două formule pentru calcularea caloriilor și a macronutrienților dorește să o folosească. În cazul formulei Katch-McArdle sunt afișate informațiile adiționale care trebuie introduse pentru a se putea folosi formula, procentajul de grăsime și nivelul de activitate, așa cum se poate vedea în figura 7.3. Alăturat se află un buton “Calculate” care la apăsare afișează o pagină în care se poate calcula procentajul de grăsime folosind formula oferită de aplicație.

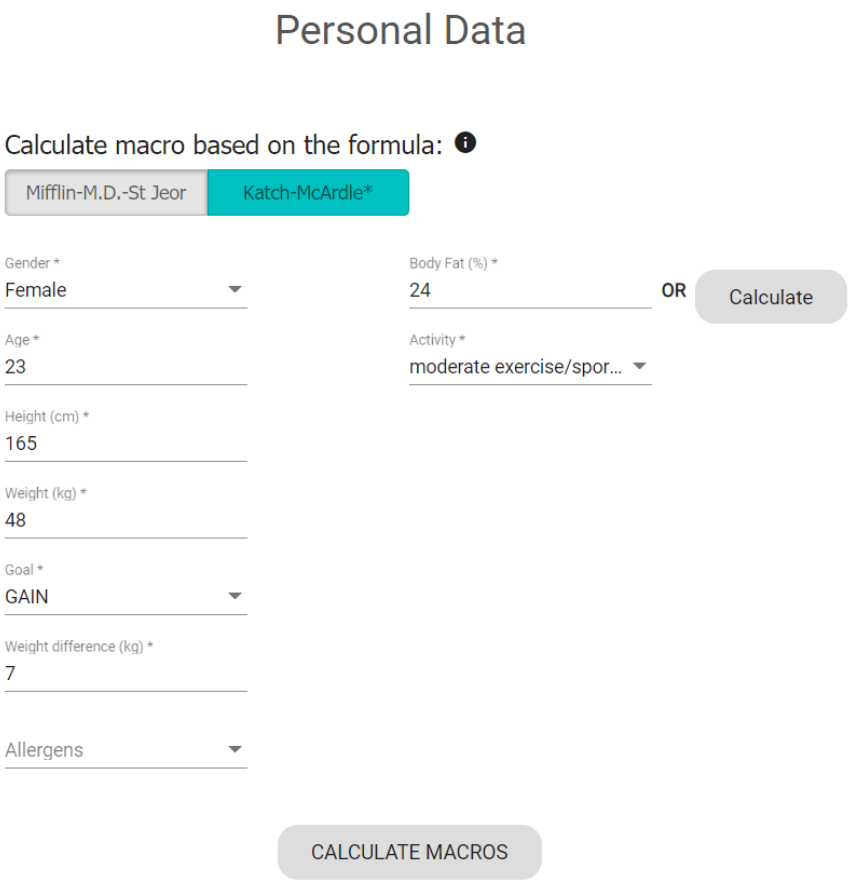



Figura 7.3 Screenshot calculare calorii și macro

După completarea datelor, la apăsarea butonului “Calculate Macros” datele introduse sunt salvate și se calculează caloriile și macronutrienții necesari pe o zi. Aceste rezultate sunt afișate tot în pagina “Profile”, la secțiunea “Macros”, ca în figura 7.4. Aici, în partea stângă sunt afișate rezultatele, care nu pot fi modificate, pentru ca clientul să le poată avea tot timpul ca o referință. În partea dreaptă sunt afișate tot rezultatele, care sunt editabile și care sunt folosite în aplicație.

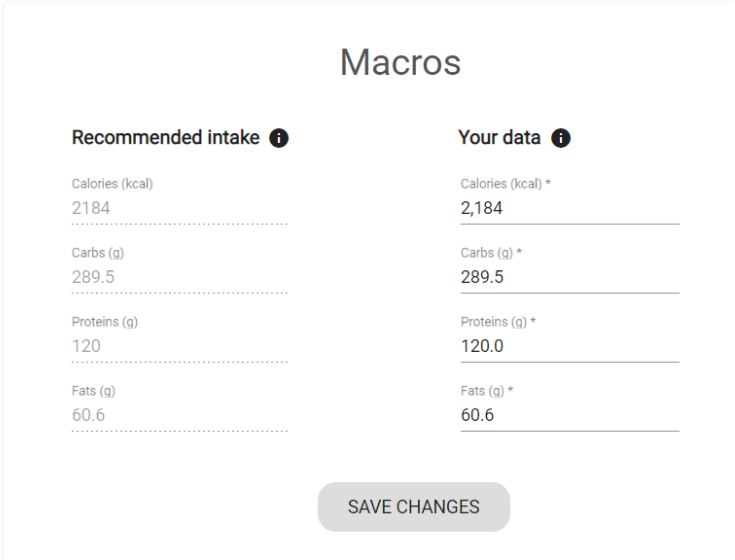



Figura 7.4 Screenshot valori recomandate pentru o zi

Sistemul metric este sistemul folosit implicit în aplicație. Acesta poate fi modificat la cel imperial tot din această pagină. Tot aici se poate seta dacă să se folosească ritmul cardiac în calcularea caloriilor arse, prin selectarea „Yes” sau „No”. Mesele dintr-o zi sunt setate implicit la breakfast, lunch și dinner, însă pot fi modificate, prin ștergerea, modificarea ordinii lor sau adăugarea unei noi mese. Ștergerea se face prin apăsarea iconiței „x” din dreptul mesei.

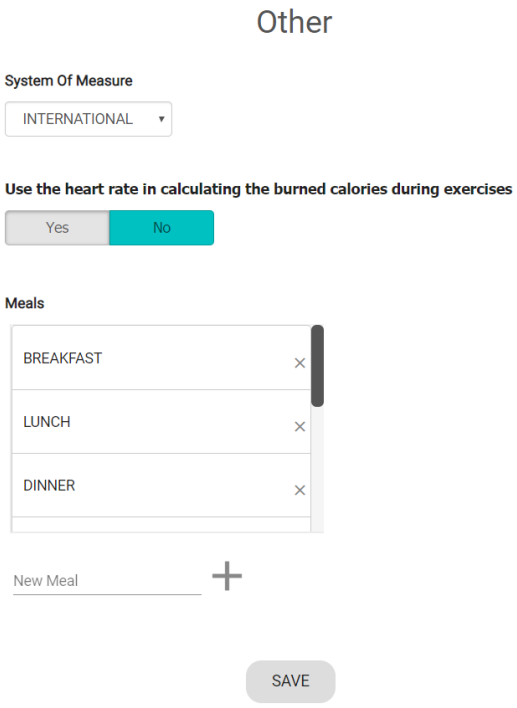



Figura 7.5 Screenshot alte opțiuni

Adăugarea unei noi mese se realizează prin completarea numelui mesei în câmpul de sub tabelul cu mese și apăsarea iconiței „+”. Ordonarea lor se face prin „drag and drop”. Totate date opționale se găsesc în secțiunea „Other”, cum se poate observa în figura 7.5.

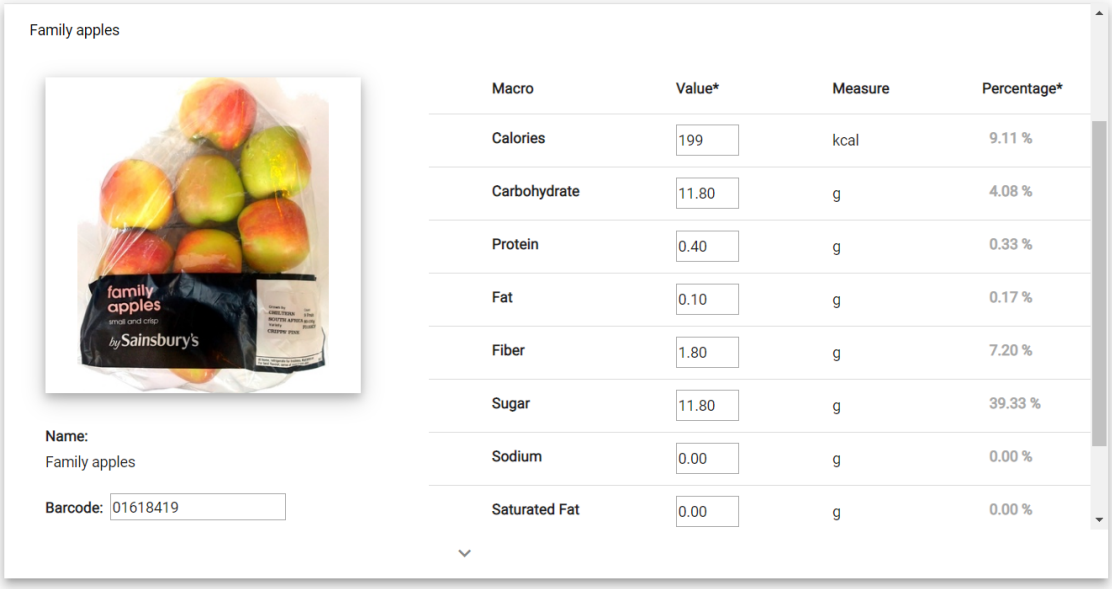



Figura 7.6 Screenshot vizualizare datele unui aliment căutat

### 4. Adăugare aliment în lista personală de alimente

Pentru a adăuga un nou aliment în lista sa personală, clientul trebuie să îl caute în pagina “Add Food”. Pentru a-l căuta după nume, trebuie să selecteze opțiunea “Name”, iar pentru a-l căuta după codul de bare trebuie să se selecteze opțiunea “Barcode”. După completarea câmpului cu numele sau codul de bare, trebuie să se apase butonul “Search”. La apăsarea acestuia, se vor afișa într-o listă toate rezultatele care au fost găsite. La selectarea unui produs din listă, acesta se va extinde, ca și în figura 7.6. Aici sunt afișate toate datele alimentului, valorile nutriționale și codul de bare putând fi editat. În câmpul “Percentage”, pentru fiecare valoare, clientul poate vedea cât la sută reprezintă din necesarul său zilnic. Valorile sunt pentru 100 de grame. După verificarea datelor și eventual modificarea lor, alimentul poate fi adăugat în listă prin apăsarea butonului “Add To Diary”. La apăsarea butonului, clientul va fi redirecționat în pagina “Food List” unde poate vedea alimentul adăugat.

Antrenorul/nutriționistul poate căuta și adăuga un nou aliment în lista clientului urmând aceiași pași ca și clientul, după selectarea clientului și selectarea opțiunii “Add Food”.

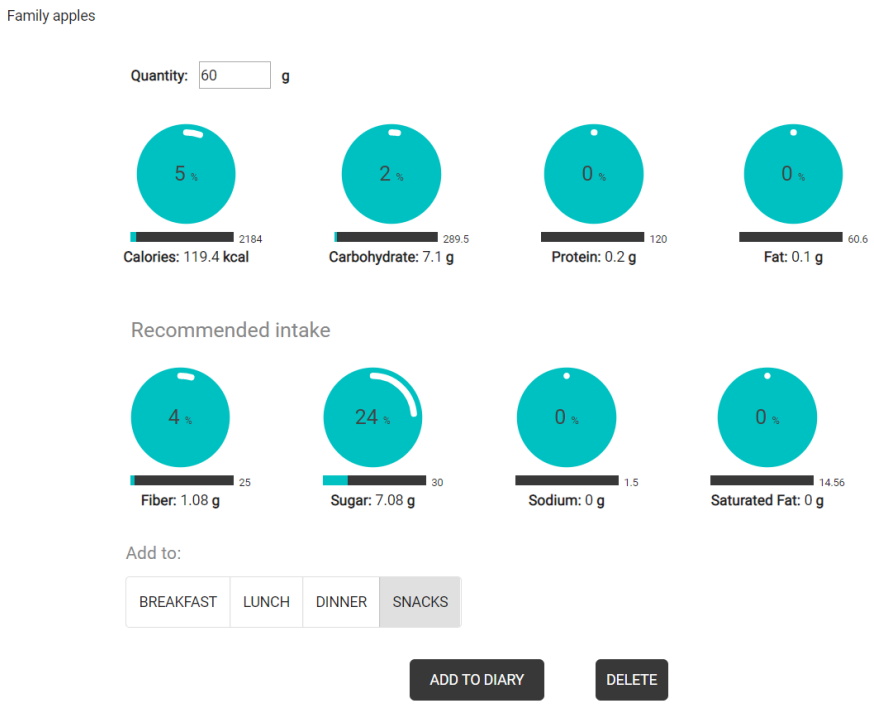



Figura 7.7 Vizualizare date aliment

### 5. Adăugare aliment în jurnal

Pentru a adăuga un aliment în jurnal, prima dată acesta trebuie selectat din lista din pagina "Food List". La selectarea acestuia, rândul este extins, afișându-se datele alimentului. În cercuri sunt afișate procentajele din necesarul zilnic al clientului, în barele de progres se poate vedea cât din ceea ce trebuie să consume clientul acoperă alimentul, iar sub bara de progres valoarea cantitativă. Modificarea cantității se realizează prin modificarea câmpului "Quantity", implicit fiind afișat pentru 100 de grame. După setarea cantității trebuie să se aleagă masa la care să se adauge alimentul ca fiind consumat. Ultimul pas este de a apăsa butonul "Add To Diary". La apăsarea butonului, clientul va fi redirecționat în pagina de jurnal "Diary" unde poate vedea alimentul adăugat.

În figura 7.8 se poate observa un exemplu pentru vizualizarea datelor unui aliment din listă pentru a-l adăuga în jurnal.

În figura 7.8 se poate observa pagina de jurnal, cu calendarul deschis pentru a selecta data pe care să se afișeze jurnalul.


Figura 7.8 Screenshot jurnal

### 6. Creare plan de masă

Planul de masă este creat în pagina “Meal Plan”. Aici clientul poate selecta data pentru care să creeze planul, implicit afișându-se pentru ziua curentă. Adăugarea unui nou aliment în plan se realizează prin completarea în partea de jos a câmpurilor. Datele care trebuie completate sunt masa la care să se adauge și numele alimentului, celelalte autocompletându-se automat. După completare, pentru a-l adăuga, trebuie apăsată iconița “+”.

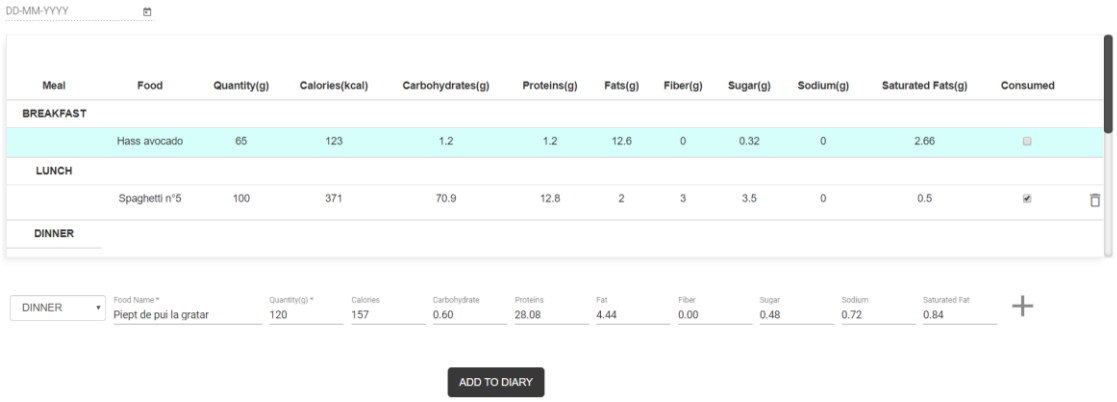



Figura 7.9 Screenshot plan de masă

Setarea unui aliment ca fiind consumat se face prin apăsarea pe căsuța din coloana “Consumed”. Ștergerea se face prin apăsarea iconiței care reprezintă un recycle bin, din ultima coloană. Alimentele introduse de antronor/nutriționist au în fundal culoarea albastră și nu pot fi șterse de client.

Adăugarea planului în jurnal se face prin apăsarea butonului “Add To Diary”.

Antrenorul/nutriționistul poate crea planuri de masă pentru un client în același mod ca și clientul, după selectarea clientului.

### 7. Adăugare exercițiu cardio și de forță în lista personală

Pentru a adăuga un nou exercițiu, clientul trebuie să meargă în pagina "Exercises", să selecteze ce tip de exercițiu dorește să adauge, cardio sau de forță, și să apase pe iconița "+" din colțul din dreapta de deasupra listei cu exerciții. Ca rezultat, o listă cu exerciții este afișată, unde se poate selecta orice rând pentru a se afișa date adiționale, care sunt imaginea care reprezintă exercițiul, o descriere a exercițiului și valoarea MET al acestuia. Pentru a-l adăuga în listă trebuie să se apase butonul "Add To List". Aceiași pași trebuie urmați și de antrenor/nutriționist, după ce s-a selectat clientul.

### 8. Adăugare exercițiu în jurnal

Adăugarea unui nou exercițiu se face din pagina "Exercices", prin selectarea tipului de exercițiu, selectarea din listă a exercițiului pe care clientul dorește să-l adauge și apăsarea butonului "Add To Diary". La selectarea unui exercițiu se pot seta date precum durata, în cazul exercițiilor cardio, și numărul de seturi și de repetiții, în cazul exercițiilor de forță.

Antrenorul/nutriționistul poate adăuga exerciții pentru un client, pentru a le crea un plan de antrenament, în același mod ca și clientul, după selectarea clientului.

În figurile 7.10 și 7.11 se pot vedea exemple de exercițiu cardio și de forță salvate în lista personală, care au fost selectate și cărora li s-au completat câmpurile pentru a se adăuga în jurnal.

În figura 7.12 se poate observa jurnalul după adăugarea exercițiilor cardio, unde exercițiile adăugate de antrenor/nutriționist au un fundal albastru.

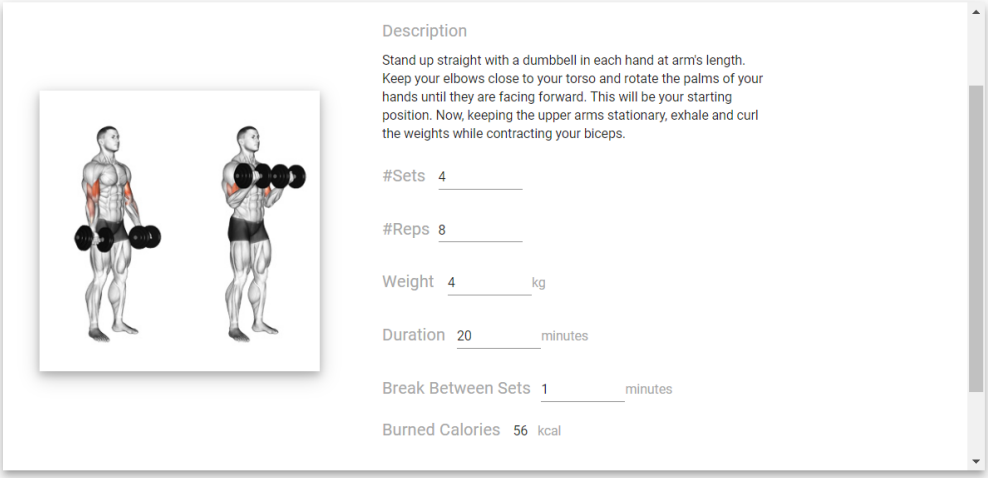



Figura 7.10 Screenshot adăugare exercițiu de forță în jurnal

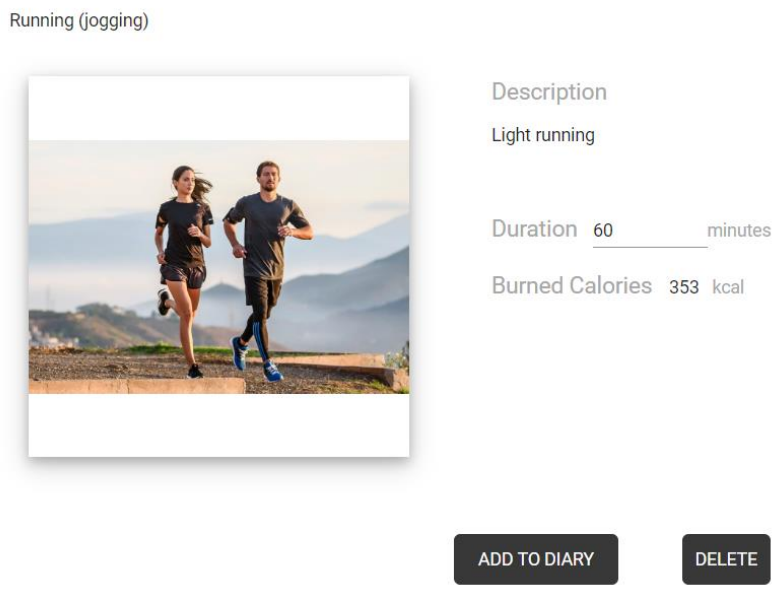



Figura 7.12 Screenshot adăugare cardio în jurnal


Figura 7.11 Screenshot jurnal antrenament

### 9. Trimitere invitație

Pentru a trimite o invitație, clientul trebuie să meargă în pagina “Collabs”, unde poate căuta antrenorii și nutriționiștii din aplicație. La selectarea unuia din listă, se afișează o fereastră cu datele despre acesta, un exemplu fiind prezentat în figura 7.13. Prezența iconiței din colțul din dreapta sus indică faptul că persoana este o sursă de încredere. Opțional, clientul poate scrie un mesaj text care să îl trimită cu invitația. Pentru a finaliza trimiterea, se apasă butonul “Send”.

### 10. Acceptare invitație

Pentru a accepta o invitație, antrenorul/nutriționistul trebuie să meargă în pagina “Collab” unde, în tabelul din partea stângă poate vedea toate invitațiile. Prin selectarea unei invitații, se va deschide o fereastră în care se va afișa și mesajul scris de client, cu

opțiunea de a accepta invitația. Prin apăsarea butonului “Accept”, se va crea colaborarea, putând să se vadă cum clientul a fost adăugat în tabelul din dreapta, unde sunt afișați clienții cu care colaborează.

### 11. Comunicare pagina de chat

Comunicarea se poate face prin intrarea în pagina de chat “Chat” sau prin selectarea iconiței de lângă această opțiune. La oricare dintre aceste opțiuni se va afișa lista cu toate persoanele cu care clientul colaborează, pentru a selecta cu cine dorește să comunice. La selectarea acestuia se va afișa pagina cu mesaje. în cazul antrenorului/nutriționistului, pagina este accesată prin selectarea clientului după aceea a opțiunii “Chat” sau prin selectarea iconiței pentru chat care va deschide o mică fereastră “flow box” de unde poate selecta și comunica prin mesaje cu clientul cu care dorește.

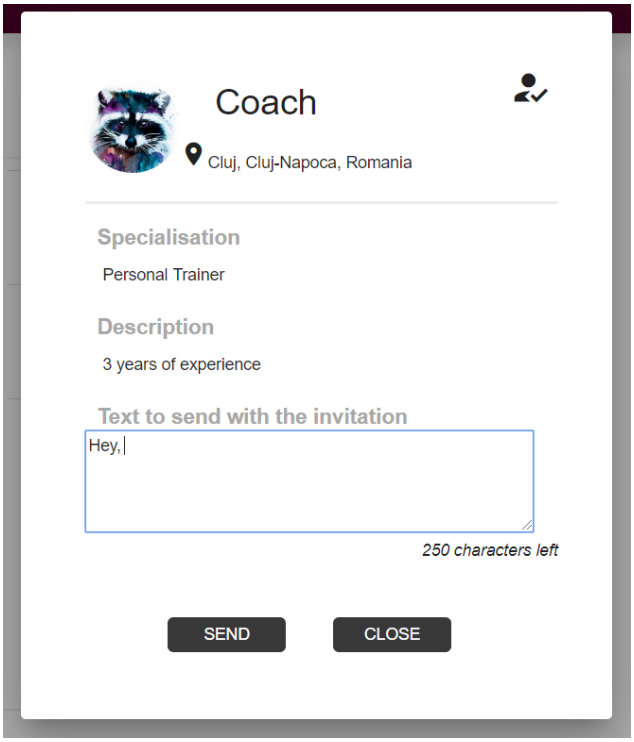



Figura 7.13 Screenshot invitație

# Capitolul 8. Concluzii

În cadrul acestui capitol sunt prezentate realizările și obiectivele care au fost atinse și dezvoltările ulterioare.

## 8.1. Realizarea obiectivelor propuse

Obiectivul principal propus a fost de a se realiza o aplicație în nutriție și fitness care să aducă ceva nou față de aplicațiile actuale prin introducerea antrenorilor și a nutriționiștilor ca un nou tip de utilizator, care nu sunt strict introduși de aplicație. Acest obiectiv a fost atins cu succes. Orice antrenor sau nutriționist poate să își creeze un cont în aplicație și orice client poate astfel să îi găsească și să le ceară să colaboreze pentru a fi ajutați.

Al doilea obiectiv a fost de a i se oferi utilizatorului mai multe metode de a calcula caloriile și macronutrienții necesari într-o zi, ceva ce alte aplicații nu oferă. Acest obiectiv a fost atins, aplicația oferind două metode. De asemenea, s-au introdus două metode și pentru calcularea caloriilor arse, oferindu-i utilizatorului mai multă flexibilitate.

Al treilea obiectiv a fost de a crea un design simplu, care să nu conțină multe elemente aglomerate într-o pagină, ceea ce se întâmplă în cele mai multe aplicații de acest gen în momentul în care au multe funcționalități. Acest obiectiv a fost atins odată datorită faptului că s-a decis crearea unei aplicații web în loc de android, ceea ce a oferit mai mult spațiu care poate fi utilizat pe o pagină și în al doilea rând prin folosirea unui grup restrâns de culori. De asemenea, pentru utilizatorii care poate nu sunt obișnuiți cu anumiți termeni s-au adăugat notări care să le explice semnificația lor, ceea ce îndeplinește și cerința nonfuncțională de utilizabilitate a aplicației.

Cerința pentru performanță a fost ca timpul de răspuns să fie 1.0 secunde. Această cerință nu este îndeplinită tot timpul, uneori anumite operații luând mai mult. O soluție pentru această problemă este de a se pune aplicația pe Cloud, care poate să furnizeze mai multe servere și să folosească un load balancer pentru a împărți traficul peste servere.

Cerințele de securitate sunt îndeplinite parțial. Motivul este acela că aplicația folosește protocolul HTTP care trimite informația necriptată, deci aceasta poate fi ușor interceptată și citită de un atacator. În ciuda acestui fapt, se folosesc tokenuri și roluri pentru autorizare, email și parolă pentru autentificare, iar parolele sunt salvate în baza de date criptat. â

Cerințele de openess, eterogenitate sunt și ele respectate.

Realizarea aplicației prin separarea front-endului și a back-endului în două proiecte separate și separarea back-endului la rândul său în două subsisteme puse în două proiecte separate permite scalarea aplicației pe viitor prin punerea serverelor pe calculatoare separate. Organizarea pe nivele va permite pe viitor adăugarea simplă de noi funcționalități și modificarea funcționalităților existente. Separarea între front-end și back-end permite extinderea aplciației pe dispozitive mobile mult mai ușor, fiind necesară implementarea doar a părții de front-end.

Funcționalitățile propuse au fost și ele implementate cu succes, conform cerințelor, pentru fiecare actor în parte.

## 8.2. Dezvoltări ulterioare

O primă dezvoltare care poate fi adusă este extinderea aplicației pe dispozitive mobile. Acest lucru ar permite introducerea unor noi funcționalități care sunt greu de implementat într-o aplicație web. Câteva funcționalități care s-ar putea implementa sunt:

- Conectarea cu dispozitive, cum ar fi ceasurile fitness. Aplicația ar putea extrage date cum sunt numărul de pași efectuați, exercițiile realizate, caloriile arse și orele de somn, care sunt importante pentru a obține un rezultat bun în urma antrenamentului.
- Scanarea codului de bare. În prezent aplicația permite căutarea alimentului după codul de bare, dar acesta trebuie introdus manual. Prin scanarea lui, alimentul ar fi mai ușor de găsit și s-ar putea folosi și pentru introducerea lui în jurnal, pentru a nu-l mai căuta în listă.
- Primirea de notificări, de exemplu pentru a-i aminti clientului că nu a consumat toate alimentele din planul creat.
- Posibilitatea de a introduce pe lângă greutate și imagini pentru a vedea în timp mai bine progresul.

Aplicația poate fi îmbunătățită și prin implementarea unor noi funcționalități. Funcționalități noi care pot fi aduse și sunt:

- Creara unui algoritm care să îi sugereze clientului exerciții pe care să le realizeze dacă a depășit numărul de calorii permise pe perioada unei săptămâni. Pentru a determina cât mai corect câte calorii ar trebui arse este necesară clasificarea alimentelor în alimente care au carbohidrați buni, care se digeră lent, și carbohidrați răi, care se digeră lent.
- Oferirea unei posibilități de plată prin care clientul să poată plăti antrenorul sau nutriționistul, prin integrarea unui API pentru care gestionează tranzacții online, cum sunt Google Pay sau PayPal.
- Posibilitatea de a crea planuri în xml care să fie mapate după aceea în aplicație, în plan.
- Creare rețete sau căutarea lor pe site-uri de unde să fie importate.
- Folosirea AI pentru a aproxima numărul de zile necesare până la atingerea scopului, pe baza greutății introduse în timp și a evaluărilor clientului și a celorlalți clienți.

# Bibliografie

[1] Marie Ng, Tom Fleming, Margaret Robinson, Blake Thomson, Nicholas Graetz, Christopher Margono et al.: *Global, regional, and national prevalence of overweight and obesity in children and adults during 1980–2013: A systematic analysis for the Global Burden of Disease Study*, THE LANCET JOURNALS, 2014. Disponibil pe: https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(14)60460-8/fulltext

[2] Alamgir Khan1, Sami Ullah Khan1, Salahuddin Khan1, Syed zia-ul-islam1, Naimatullah Khan Baber2, Manzoor Khan3: *Nutritional complications and its effects on human health*. Journal of Food Science and Nutrition, 2018; pag.18. Disponibil pe: https://www.alliedacademies.org/articles/nutritional-complications-and-its-effects-on-human-health.pdf

[3] Christopher Wharton, Carol Johnston, Barbara K. Cunningham, Danielle Sterne: *Dietary Self-Monitoring, But Not Dietary Quality, Improves With Use of Smartphone App Technology in an 8-Week Weight Loss Trial. Journal of Nutrition Education and Behavior*, 2014. Disponibil pe: https://asu.pure.elsevier.com/en/publications/dietary-self-monitoring-but-not-dietary-quality-improves-with-use

[4] Lora E. Burke, et.al. Self-Monitoring in Weight Loss: A Systematic Review of the Literature. Disponibil pe: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3268700/#R12

[5] Gabrielle M Turner-McGrievy et. al.: *Comparison of traditional versus mobile app self-monitoring of physical activity and dietary intake among overweight adults participating in an mHealth weight loss program*. The Journal of the American Medical Informatics Association. Disponibil pe: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3628067/

[6] Frankenfield D1, Roth-Yousey L, Compher C.: *Comparison of predictive equations for resting metabolic rate in healthy nonobese and obese adults.* Journal of the American Dietetic Associacion, 2005. Disponibil pe: https://www.ncbi.nlm.nih.gov/pubmed/15883556

[7] Ted Kallmyer: *How to Calculate Your Macros to Transform Your Body*. Disponibil pe: https://healthyeater.com/how-to-calculate-your-macros

[8] Marta Lonnie, Emma Hooker, Jeffrey M. Brunstrom, Bernard M. Corfe, Mark A. Green, Anthony W. Watson, Elizabeth A. Williams, Emma J. Stevenson, Simon Penson, and Alexandra M. Johnstone: *Protein for Life: Review of Optimal Protein Intake, Sustainable Dietary Sources and the Effect on Appetite in Ageing Adults*, 2018 [online]. Disponibil pe: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5872778/

[9] Jakob Nielsen: *Usability Engineering*, 1993. Capitolul 5.

[10] Yang Xiaojie: Analysis of DBMS: MySQL Vs PostgreSQL. Kemi-Tornio University of Applied Sciences Technology. Disponibil pe: https://www.theseus.fi/bitstream/handle/10024/27471/Final_Thesis_Xiaojie_Yang.pdf

[11] Datastax: *NoSQL Performance Benchmarks, Cassandra vs MongoDB vs Couchbase vs Hbase*. Disponibil pe: https://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-Hbase

# Lista figurilor din lucrare

# Lista tabelelor din lucrare

# Glosar de Termeni

| | |
|---|---|
| API | Application Programming Interface |
| API | Application Programming Interface |
| ACID | Atomicity, Consistency, Isolation, Durability |
| CRUD | Create, Read, Update, Delete |
| CSS | Cascading Style Sheet |
| HTTP | HyperText Transfer Protocol |
| HTML | Hypertext Markup Language |
| JSON | JavaScript Object Notation |
| MVC | Model View Controller |
| SQL | Structured Query Language |
| SVG | Scalable Vector Graphics |
| UI | User Interface |
| URL | Uniform Resource Locator |

# Diagramă de clase