

Documentație

Autor,
Andrei **BULIGA**

Rezumat

Acest studiu prezintă o abordare în planificarea și monitorizarea nutriției cu NutriPlanner, o aplicație construită folosind stiva MERN și TypeScript. NutriPlanner își propune să adreseze nevoia crescândă de a gestiona eficient dieta, oferind o platformă ușor de utilizat. Contribuțiile teoretice ale proiectului includ analiza detaliată a problemei și identificarea nevoilor specifice ale utilizatorilor în domeniul planificării meselor și monitorizării nutriției. Pe partea practică, lucrarea dezvoltă o metodologie bazată pe modelul Waterfall, care integrează aspecte complexe, cum ar fi autentificarea utilizatorilor, accesul la rute protejate și administrarea. În final, studiul prezintă o analiză a implementării și testării NutriPlanner, subliniind rezultatele tangibile și implicațiile lor pentru îmbunătățirea calității vieții utilizatorilor.

Cuprins

Abrevieri	5
Lista tabelelor și figurilor	6
Introducere.....	7
I. Raport de Analiză.....	8
1. Document de Viziune.....	8
2. Identificarea și Definirea Problemei	9
2.1 Motivație	9
2.1.1 Spațiul Problemei	9
2.1.2 Stabilirea și Formularea Obiectivelor	11
2.1.2.1 Diagrama Fishbone	11
2.1.2.2 Diagrama de Descompunere a Obiectivelor.....	13
2.1.3 Analiza Contextului	15
3. Cerințe de Sistem.....	16
3.1 Surse de Cerințe	16
3.2 Elicitația Cerințelor	16
3.2.1 Analiza Comparativă a Aplicațiilor	16
3.2.2 Modelul USE-CASE	17
3.2.2.1 Tabel Actori, Roluri și Obiective	18
3.2.2.2 Sistem de creare de planuri nutriționale online.....	19
3.2.2.3 Proces de creare de planuri nutriționale online	21
3.3 Documentarea Cerințelor	23
3.3.1 Procese și Activități.....	24
3.3.1.1 Diagrama de Activități.....	24
3.3.1.2 Diagrama de Stări.....	25
II. Metodologia de lucru	28
1. Tehnologii de lucru.....	28
1.1 Stiva MERN	28
1.2 TypeScript.....	29
1.3 Librării Folosite.....	29
1.4 VS Code	30
2. Model de dezvoltare	30
3. Model de organizare.....	31
III. Proiectarea Sistemului Informatic.....	34
1. Arhitectura Sistemului	34
2. Diagrama Flux de Date.....	35
3. Diagrama pe Componente	37
4. Diagrama de Deployment	38
5. Mecanisme funcționale	40

5.1	Algoritmi și prelucrări interne.....	40
5.2	Structura Datelor	41
6.	Interfața cu utilizatorul	42
IV.	Implementare și Testare	44
1.	VS Code - Configurarea Mediului de Dezvoltare și Organizarea Codului	44
2.	Implementare	46
2.1	Backend.....	46
2.2	Frontend	60
3.	Testare	71
4.	Evaluare	72
	Concluzii.....	73
	Bibliografie.....	74

Abrevieri

<i>MERN</i> ,	MongoDB, Express.js, React.js și Node.js
<i>TS</i> ,	TypeScript
<i>JWT</i> ,	JSON Web Token
<i>API</i> ,	Application Programming Interface
<i>UI</i> ,	User Interface
<i>VS Code</i> ,	Visual Studio Code
<i>CRUD</i> ,	Create-Retrieve-Update-Delete

Lista tabelelor și figurilor

Tabele:

Tabel 1. Analiza actorilor – Roluri și Obiective	18
---	----

Figuri:

Figura 1. Diagrama Fishbone.....	13
Figura 2. Diagrama de descompunere a obiectivelor.....	14
Figura 3. Pagina Principala din NutriPlanner	16
Figura 4. Pagina Detalii Reteta	16
Figura 5. Ecrane din MyFitnessPal	17
Figura 6. Ecrane din Yazio	17
Figura 7. Modelul USE-CASE – Caz general de utilizare.....	21
Figura 8. Diagrama USE-CASE – Caz specific de utilizare.....	22
Figura 9. Diagrama de activități – Proces de creare plan	25
Figura 10. Diagrama de stări – Proces de creare a planului.....	27
Figura 11. Diagrama Gantt - Reprezentarea temporală a etapelor în desfășurare pentru dezvoltarea NutriPlanner	32
Figura 12. Arhitectura sistemului informatic - Model-View-Controller	34
Figura 13. Diagrama Flux de Date.....	36
Figura 14. Diagrama pe componente	37
Figura 15. Diagrama de Deployment.....	39
Figura 16. Structura bazei de date.....	41
Figura 17. Pagina Home pentru Administrator	43
Figura 18. Structura Folderelor	45
Figura 19. Importarea modulelor și bibliotecilor necesare	47
Figura 20. Configurarea variabilelor de mediu și conectarea la baza de date	48
Figura 21. Inițializarea aplicației Express și configurarea middleware-ului	48
Figura 22. Definirea rutelor API.....	48
Figura 23. Configurarea permisiunilor CORS	49
Figura 24. Servirea fișierelor statice și rute pentru producție sau dezvoltare	49
Figura 25. Types.ts.....	50
Figura 26. db.ts	50
Figura 27. userModel.ts	51
Figura 28. recipeModel.ts	53
Figura 29. mealPlanModel.ts	54
Figura 30. userController.ts – exemplificare autentificare reușită și nereușită.....	55
Figura 31. recipeController.ts	57
Figura 32. mealPlanController.ts	59
Figura 33. authService.js.....	62
Figura 34. authService.js.....	63
Figura 35. Login & Register	64
Figura 36. Header.jsx	65
Figura 37. Pagina Home din 3 perspective	67
Figura 38. Pagina RecipeDetails.....	68
Figura 39. Pagina CreateMealPlanPage.....	69
Figura 40. Pagina SavedMealPlanPage & Pop-up cu MealPlanList	70

Introducere

Lucrarea de față abordează dezvoltarea NutriPlanner, o aplicație web modernă și scalabilă pentru planificarea nutrițională, concepută cu ajutorul tehnologiilor MERN și TS. Având la bază principiile programării reactive și scalabile, NutriPlanner reprezintă un exemplu de utilizare eficientă a acestor tehnologii pentru a crea o soluție complexă și eficace în domeniul planificării nutriționale. Contextul actual subliniază importanța unei diete sănătoase și echilibrate, o necesitate tot mai recunoscută și valorizată în societatea contemporană. În acest context, NutriPlanner vine ca răspuns la această cerere tot mai acută, oferind utilizatorilor un instrument intuitiv și personalizabil pentru gestionarea dietei lor zilnice.

Capitolul 1 - Analiza va discuta cerințele specifice și necesitatea aplicației NutriPlanner, analizând în profunzime publicul țintă și cerințele funcționale și nefuncționale ale aplicației. Vom lua în considerare și posibilele constrângeri tehnice care pot apărea în timpul dezvoltării.

Capitolul 2 - Metodologia de Lucru va prezenta procesul de dezvoltare adoptat pentru NutriPlanner, subliniind în mod deosebit valoarea adăugată a tehnologiilor MERN și TS în crearea unei aplicații web scalabile și eficiente. De asemenea, vom discuta utilizarea modelului Waterfall, ca metodologie de gestionare a proiectului, evidențiind beneficiile acestuia în coordonarea eforturilor de dezvoltare.

Capitolul 3 - Proiectare se va concentra pe arhitectura și structura NutriPlanner, cu o prezentare detaliată a designului interfeței cu utilizatorul, a arhitecturii backend și a structurii bazei de date. Vom explora modul în care acestea susțin funcționalitățile unice ale aplicației, ilustrate prin diagrame UML și prin arhitectura Model-View-Controller (MVC).

Capitolul 4 - Implementare și Testare va descrie procesul de implementare a funcționalităților NutriPlanner, precum și metodologia de testare adoptată pentru a asigura calitatea și performanța. Se vor prezenta provocările întâmpinate în procesul de implementare și soluțiile adoptate pentru a le depăși.

În concluzii, vom face o recapitulare a proiectului, evidențiind realizările și impactul pe care NutriPlanner îl poate avea asupra comunității utilizatorilor. Vom discuta, de asemenea, posibilele direcții de dezvoltare pentru viitor, având în vedere cerințele și nevoile în continuă schimbare ale utilizatorilor.

I. Raport de Analiză

Acest capitol abordează elementele esențiale pentru dezvoltarea NutriPlanner, analizând o serie de factori relevanți și dinamici specifice domeniului nutrițional. Vom efectua o evaluare riguroasă a domeniului, identificând clar problema centrală pe care aplicația o adresează, și ne vom aprofunda înțelegerea aspectelor specifice legate de aceasta.

Vom extrage cerințele esențiale pentru implementarea aplicației, respectând specificațiile și obiectivele acesteia.

1. Document de Viziune

În dinamica secolului XXI, suntem martorii unei schimbări semnificative a populației globale către mediul online (United Nations, 2020), un aspect cu impact direct în numeroase domenii, inclusiv cel nutrițional. Analizând tendințele și progresul acestei industrii, constatăm o cerere în creștere pentru soluții digitale care să satisfacă nevoile punctuale ale consumatorilor moderni.

Criza provocată de pandemia COVID-19 a remodelat modul în care interacționăm cu tehnologia, accelerând progresul către digitalizare (McKinsey & Company, 2020). În acest context, aplicația NutriPlanner își propune să devină un aliat al stării de bine și sănătății într-o lume unde tehnologia devine esențială. Ne propunem să furnizăm soluții eficiente și personalizate în domeniul nutrițional, adaptate la nevoile fiecărui utilizator.

Este evident că mediul digital a devenit un instrument esențial pentru eficiență, atât în sfera profesională, cât și în cea personală. În domeniul nutriției, avansul tehnologic a permis crearea unor platforme care facilitează accesul la informații relevante și personalizate, precum și la sfaturi de specialitate (Zarnowiecki et al., 2020). Aplicația NutriPlanner se înscrie în această direcție, oferind un serviciu integrat pentru îmbunătățirea calității vieții utilizatorilor.

Astfel, importanța dezvoltării unei aplicații precum NutriPlanner, care să ofere o platformă digitală intuitivă și accesibilă dedicată nutriției, este mai evidentă ca niciodată. Un astfel de instrument poate avea un impact semnificativ în promovarea unei alimentații sănătoase și a unui stil de viață echilibrat, devenind un aliat de nădejde în gestionarea eficientă a sănătății.

Prin adaptarea la tendințele actuale și viitoare din domeniu și la cerințele specifice ale consumatorilor, NutriPlanner este concepută pentru a întări rolul tehnologiei ca suport în menținerea sănătății și a bunăstării. Într-o lume în care digitalizarea este o prezență constantă

în viața noastră de zi cu zi, această viziune ne va ghida în dezvoltarea aplicației, asigurându-ne că aceasta rămâne relevantă și utilă în contextul actual.

2. Identificarea și Definirea Problemei

2.1 Motivație

2.1.1 Spațiul Problemei

În societatea contemporană, importanța unui stil de viață sănătos și a unei alimentații echilibrate este tot mai recunoscută (Health, 2019, Olabanji, 2022). Cu toate acestea, gestionarea nutriției și a regimului alimentar poate fi o provocare pentru mulți indivizi. Tendința de a se baza pe alimente procesate rapid și comod, ignorând necesitățile nutriționale, este o problemă majoră care contribuie la creșterea ratei obezității și a bolilor legate de dietă (World Health Organisation, 2020). Observăm nevoia unei aplicații web interactive, numită NutriPlanner, care să ajute utilizatorii să își planifice mesele și să își monitorizeze aportul nutrițional.

Sistemul informatic își propune să ajute utilizatorii să mențină un stil de viață sănătos prin punerea la dispoziție a unor funcționalități care facilitează planificarea mesei, oferirea de sugestii personalizate pentru mese și monitorizarea aportului de macro și micronutrienți. Scopul principal este de a face procesul de planificare a meselor și de monitorizare a nutriției mai eficient și mai accesibil.

Taxonomia sistemului informatic propusă pentru NutriPlanner promovează un program general cu o aplicabilitate specifică (planificare și gestionare a mesei), care este destinat tuturor persoanelor care dețin un dispozitiv conectat la internet la momentul utilizării aplicației. Vârsta nu reprezintă un criteriu de filtrare a utilizatorilor. Oricine poate utiliza aplicația, având ocazia de a îmbunătăți starea sa fizică și psihică printr-un plan alimentar corect, indiferent de experiența sau de interesele culinare.

Folosirea sistemului informatic este simplă și accesibilă pentru toți utilizatorii, indiferent de nivelul de cunoștințe tehnice. Utilizatorii pot accesa o gamă largă de rețete, descoperind gusturi noi și creând un echilibru nutrițional adecvat pentru îmbunătățirea sănătății și stării lor de bine. În plus, utilizatorii pot crea un plan de masă săptămânal personalizat, care este împărțit pe zile și conține un număr maxim de cinci rețete pe zi: mic dejun, prânz, cină și două gustări.

Pentru a maximiza eficiența și flexibilitatea aplicației, utilizatorii au posibilitatea de a adăuga sau de a șterge rețete din planul aferent zilei selectate. De asemenea, pentru a oferi o

mai mare flexibilitate, utilizatorii pot șterge complet planul de masa pentru o anumita zi, in cazul in care doresc sa înceapă din nou.

Privind din perspectiva administratorului, acesta are avantajul de a gestiona utilizatorii platformei, validând sau devalizând utilizatorii pentru a activa sau dezactiva accesul la funcționalitățile aplicației. Administratorul poate, de asemenea, adăuga sau șterge rețete, asigurându-se ca tot conținutul disponibil este întotdeauna la zi si relevant pentru utilizatori.

În era digitala de astăzi, aplicațiile web precum NutriPlanner reprezintă un avantaj considerabil pentru utilizatori, deoarece aceștia pot accesa si gestiona rețetele si planurile lor de masa de oriunde, fără a fi nevoie de deplasare sau de o anumita locație fizica. Avantajele lucrului cu o astfel de aplicație includ economisirea timpului si a energiei, precum si posibilitatea de a găsi un echilibru între viața personala si cea profesionala.

Pe piața aplicațiilor web de gestionare a nutriției, NutriPlanner se confrunta cu o concurenta diversa. Cu toate acestea, aceasta aplicație se distinge prin funcționalitățile sale intuitive si personalizate, care sunt proiectate pentru a încuraja o alimentație sănătoasă si pentru a facilita planificarea meselor.

Într-o lume in care ritmul de viață a devenit tot mai accelerat, NutriPlanner oferă o soluție digitala convenabila care elimina stresul asociat cu găsirea timpului pentru a găti si manca sănătos. Mai mult decât atât, prin oferirea unui plan de masa personalizat si a unor rețete diverse, utilizatorii pot evita monotonia in dieta lor si pot descoperi noi modalități de a se bucura de mâncare, păstrându-se in același timp sănătoși.

Spre deosebire de alte aplicații populare, precum MyFitnessPal și Yazio, NutriPlanner adoptă o abordare diferită în asistarea utilizatorilor săi privind nutriția.

MyFitnessPal permite utilizatorilor să își creeze propriile rețete, să înregistreze alimentele consumate și să urmărească activitățile fizice. Aplicația are o bază de date vastă de alimente și oferă sugestii și îndrumări pentru a ajuta utilizatorii să își atingă obiectivele nutriționale.

Yazio, similar cu MyFitnessPal, oferă posibilitatea de a înregistra alimente și activități fizice, și permite de asemenea crearea de rețete proprii. Yazio se concentrează și pe echilibrarea macro și micronutrienților, oferind sugestii de rețete pentru a ghida utilizatorii în alegerea unor opțiuni sănătoase.

NutriPlanner diferă de MyFitnessPal și Yazio prin faptul că oferă utilizatorilor planuri de masă complete cu rețete deja create, făcând procesul de planificare a meselor mult mai eficient și ușor. Utilizatorii nu trebuie să își creeze propriile rețete, ci pot folosi cele oferite de aplicație, care sunt adaptate pentru a îndeplini diferite obiective de sănătate și preferințe

alimentare. Aceasta usurează munca utilizatorilor, deoarece pentru a își atinge obiectivele, acestia trebuie doar să urmărească planul de masă creat cu rețetele deja pregătite de NutriPlanner.

În concluzie, în timp ce MyFitnessPal și Yazio oferă funcționalități pentru a îndruma și a permite personalizarea alimentației, NutriPlanner merge un pas mai departe prin oferirea de planuri de masă cu rețete gata făcute, optimizând procesul și făcând planificarea meselor mai accesibilă și eficientă pentru utilizatori.

2.1.2 Stabilirea si Formularea Obiectivelor

2.1.2.1 Diagrama Fishbone

Pentru a analiza necesitatea și oportunitatea dezvoltării aplicației NutriPlanner, vom utiliza diagrama Fishbone. Am ales această diagramă deoarece ne permite să structurăm și să vizualizăm în mod clar cauzele diferite care au contribuit la necesitatea unei soluții de planificare nutrițională, și cum aceste cauze interacționează pentru a crea un context favorabil dezvoltării NutriPlanner.

Pornind de la necesitatea unei soluții eficiente de planificare și urmărire a nutriției individuale, vom evidenția situațiile care au dus la această nevoie și care justifică crearea unei aplicații web precum NutriPlanner.

În cadrul diagramei Fishbone (Figura 1) putem identifica patru cauze principale care relevă necesitatea acestui produs: Sănătate și Fitness, Accesibilitate, Economie și Marketing.

Primul factor, Sănătate și Fitness, se referă la cererea crescândă pentru o nutriție adecvată și sănătoasă (World Health Organization, 2023) și implicit la necesitatea unui instrument care să permită urmărirea și planificarea eficientă a acesteia. Acest lucru vine ca răspuns la dorința tot mai mare a oamenilor de a adopta un stil de viață sănătos și de a avea un control mai bun asupra alimentației lor (World Health Organization, 2023). În plus, oamenii sunt tot mai interesați de a-și diversifica dieta și de a încerca noi rețete, fără a fi nevoiți să petreacă mult timp căutând și experimentând diferite opțiuni. În acest context, NutriPlanner se dorește a fi un asistent virtual, oferind acces la o diversitate de rețete și planuri nutriționale adaptate nevoilor fiecărei persoane.

Accesibilitatea este a doua cauză principală ce justifică nevoia unei astfel de aplicații. În era digitală, consumatorii se așteaptă la soluții rapide, eficiente și la îndemână pentru a-și gestiona diferite aspecte ale vieții lor, inclusiv alimentația. De asemenea, utilizatorii au nevoie de o platformă care să funcționeze la fel de bine pe dispozitivele mobile, fapt ce subliniază importanța unei aplicații web responsabile ca NutriPlanner.

Sub aspect economic, NutriPlanner vine în întâmpinarea unui segment de piață care caută soluții accesibile și eficiente pentru a-și gestiona nutriția. Prin oferirea unei varietăți de rețete și planuri nutriționale, NutriPlanner economisește timpul utilizatorilor și reduce cheltuielile legate de cumpărarea de produse alimentare neadecvate sau neutilizate.

În sfârșit, avem factorul Marketing. Așa cum ne învață cercetarea de piață (Collins, 2019), pentru a avea succes, un produs nou trebuie promovat eficient. Într-o lume în care informațiile sunt la doar un clic distanță, NutriPlanner trebuie să fie vizibil și atractiv pentru a capta interesul oamenilor.

Strategia de marketing ar trebui să includă o combinație de marketing digital și tradițional (McCormick, 2023). Pentru marketingul digital, optimizarea pentru motoarele de căutare (SEO) și marketingul prin conținut sunt esențiale pentru a ajunge la o audiență mai largă online. Un site web bine proiectat, bloguri informative, video-uri utile și prezența activă pe rețelele sociale pot ajuta în acest sens.

Rețelele sociale, în special, pot fi un instrument puternic de marketing pentru NutriPlanner (McCormick, 2023). Întrucât produsul se adresează unui public care își dorește un stil de viață sănătos, platforme precum Instagram, Pinterest și Facebook pot fi utilizate pentru a posta conținut legat de sănătate și nutriție, inclusiv rețete și sfaturi de sănătate. Aceste postări pot fi însoțite de linkuri către NutriPlanner, încurajând utilizatorii să încerce aplicația.

În ceea ce privește marketingul tradițional, NutriPlanner ar putea colabora cu nutriționiști și personalități din domeniul sănătății pentru a promova produsul (Nacach, 2018). De asemenea, poate fi util să se găsească oportunități de sponsorizare sau parteneriate cu evenimente sportive sau de sănătate pentru a crește vizibilitatea brandului.

Este de asemenea important ca NutriPlanner să aibă o strategie eficientă de marketing prin e-mail (kristjan, 2021). Newslettere informative și personalizate pot fi trimise abonaților, împărtășind cu ei sfaturi despre nutriție, ultimele actualizări ale aplicației și oferte speciale.

În încheiere, succesul NutriPlanner va depinde de o combinație de tehnologie sofisticată, conținut valoros și marketing eficient. Într-o piață competitivă, este esențial să se furnizeze un produs care să răspundă nevoilor consumatorilor și să se găsească modalități inovatoare de a ajunge la publicul țintă.

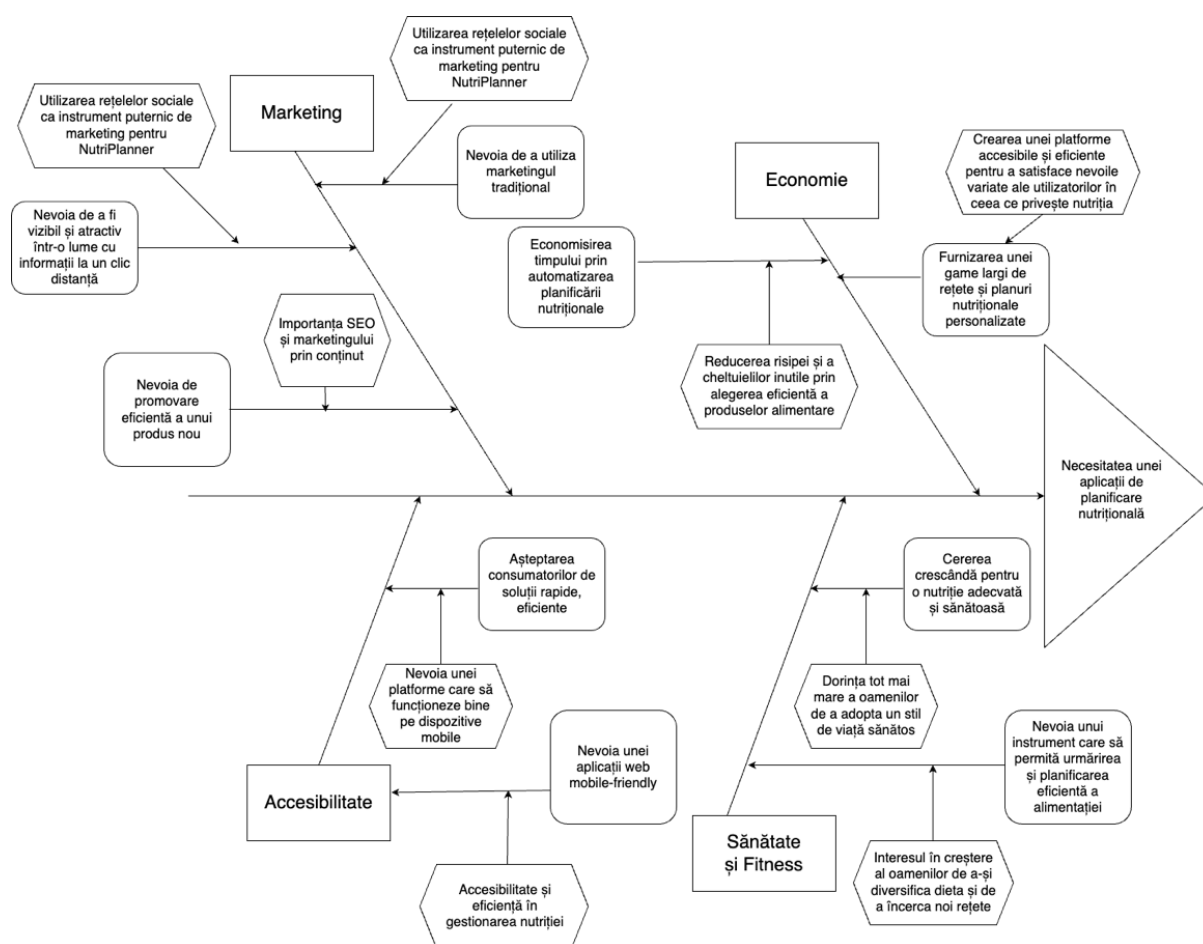


Figura 1. Diagrama Fishbone

2.1.2.2 Diagrama de Descompunere a Obiectivelor

Având la bază detaliile și funcționalitatea aplicației NutriPlanner, ne propunem să dezvoltăm o diagramă de descompunere a obiectivelor care să ilustreze ierarhic obiectivele asumate pentru dezvoltarea acestei aplicații și conexiunile dintre ele.

Principalul obiectiv este dezvoltarea unei aplicații web interactive pentru planificare nutrițională, iar obiectivele de nivelul 1 sunt constituite din elementele oferite de aplicație utilizatorilor săi. Astfel, un prim obiectiv este garantarea securității prin crearea unui sistem de autentificare bazat pe adresa de e-mail și parola. De asemenea, ne propunem să includem două categorii de utilizatori: utilizatorul obișnuit și administratorul.

Pentru utilizator, acesta poate explora și selecta rețete, în timp ce administratorul are rolul de a gestiona rețetele și utilizatorii platformei (nivel 2). Fiecare utilizator are un profil care cuprinde informații importante, ca de exemplu accesul la rețetele selectate și structurate. În ceea ce privește administratorul, acesta are posibilitatea de a adăuga sau elimina rețete și de

a valida utilizatorii (nivel 3). Mai mult, utilizatorii pot crea și șterge planuri de masă, în timp ce administratorul poate analiza și valida aceste planuri (nivel 4).

Un alt scop la nivelul 2 este implementarea unei platforme interactive în cadrul aplicației, care să cuprindă o listă de rețete din care utilizatorii pot afla informații și pe care le pot selecta. Având în vedere nivelul 3, planificăm dezvoltarea de funcționalități care să permită utilizatorilor să selecteze rețete în funcție de diferite criterii (de exemplu, tipul de masă).

Ultimul obiectiv de nivel 2 se referă la accesul global la informațiile deținute de toți utilizatorii, care poate fi realizat prin intermediul unei baze de date globale online (MongoDB), menită să sprijine interactivitatea aplicației. În ceea ce privește nivelul 3, plănuim modalități de structurare și stocare a datelor, în funcție de cerințele aplicației. Astfel, ne propunem separarea datelor referitoare la utilizatori, planuri de masă și rețete, pentru a asigura o mai bună gestionare a acestora.

În concluzie, stabilirea ierarhică a obiectivelor facilitează înțelegerea și urmărirea acestora, evidențiind într-un mod gradat, de la general la specific, ce anume se urmărește prin dezvoltarea aplicației web NutriPlanner.

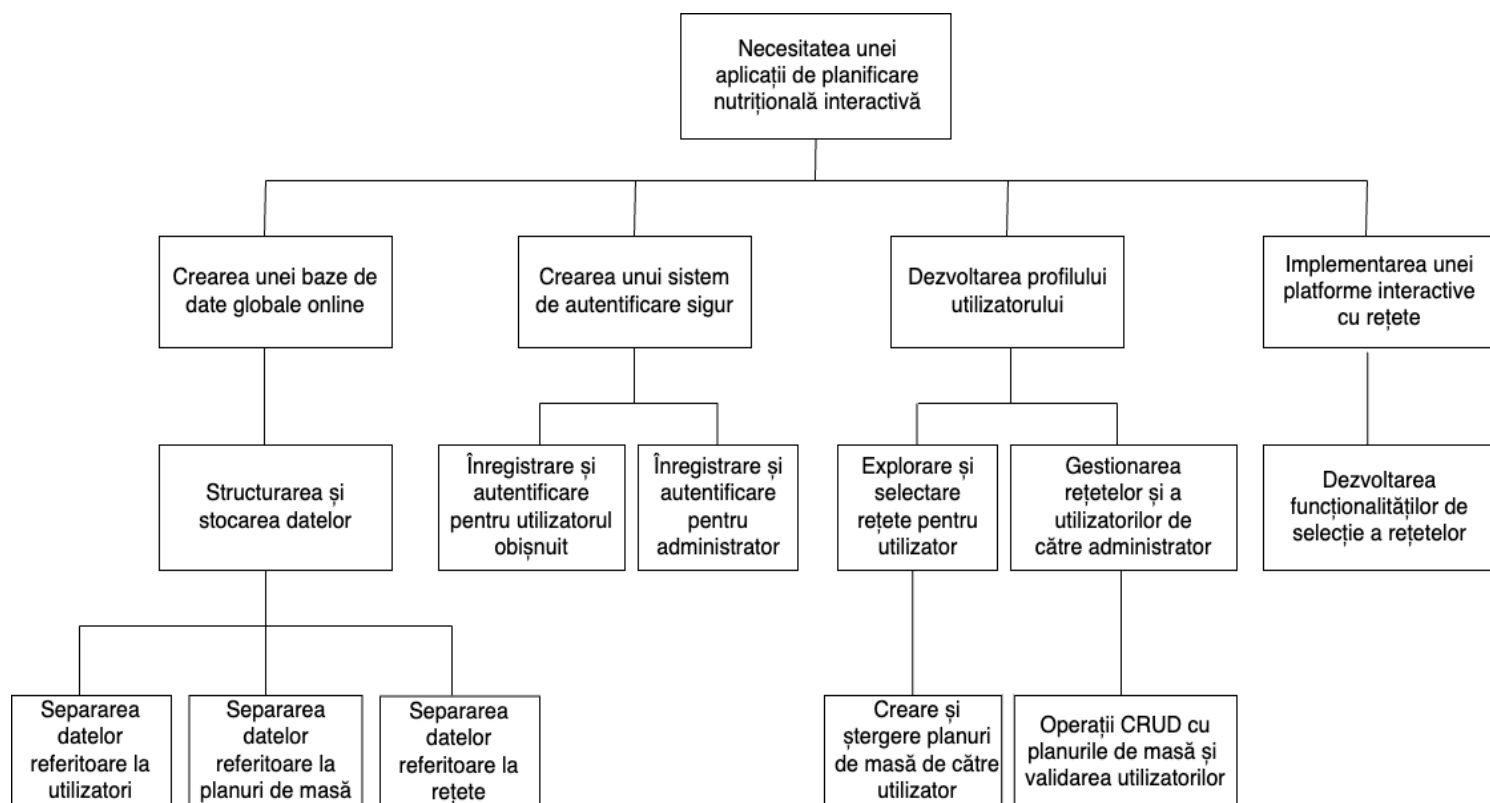


Figura 2. Diagrama de descompunere a obiectivelor

2.1.3 Analiza Contextului

Analiza Fațetelor, un element central în cadrul procesului de analiză a contextului aplicației noastre, joacă un rol vital în a oferi o perspectivă cuprinzătoare și complexă asupra funcționalității sale. Scopul acestei aplicații este de a furniza utilizatorilor o platformă interactivă, în care aceștia pot să creeze și să gestioneze un plan alimentar personalizat, care se bazează pe o diversitate largă de rețete existente în baza de date a sistemului.

Fațeta Subiect se concentrează asupra analizării bazei informaționale a aplicației, jucând un rol crucial în identificarea și gestionarea cerințelor specifice ale sistemului. Înainte de a putea interacționa cu aplicația, utilizatorii trebuie să-și creeze un cont, proces care necesită introducerea unor informații de bază, cum ar fi numele, adresa de email și o parolă. Odată autentificați în sistem, utilizatorii au acces la baza de date cu rețete, unde pot să creeze un plan de masă personalizat pentru fiecare zi a săptămânii, cu opțiunea de a adăuga sau de a șterge rețete în funcție de preferințele personale.

Fațeta Utilizare se axează pe modul în care informațiile sunt utilizate în cadrul aplicației noastre. Utilizatorii au capacitatea de a naviga prin lista vastă de rețete disponibile, de a adăuga rețete la planul lor de masă, de a elimina întregul plan pentru o zi specifică sau de a alege să se delogheze. În plus, administratorii sistemului pot interveni pentru a valida sau a invalida utilizatorii și pot adăuga sau elimina rețete din baza de date.

Fațeta IT se referă la structura tehnologică ce susține aplicația. Sistemul nostru de operare se bazează pe tehnologii avansate de dezvoltare web, cum ar fi bcryptjs pentru criptarea parolilor și JWT pentru autentificare. De asemenea, utilizăm mongoose, un instrument versatil pentru gestionarea bazelor de date, care ne permite să stocăm și să gestionăm informații relevante despre utilizatori, rețete și planurile de masă.

Fațeta Dezvoltare este dedicată procesului de dezvoltare a aplicației, care urmează metodologia Waterfall. Procesul implică o serie de faze consecutive - analiză, proiectare, implementare, testare, lansare și mentenanță - toate parcurse într-o ordine strictă pentru a garanta finalizarea cu succes a proiectului. Această abordare ne permite să tratăm fiecare etapă într-o manieră sistematică și să ne asigurăm că obiectivele stabilite sunt atinse într-un mod eficient și ordonat.

În concluzie, Analiza Fațetelor constituie un instrument valoros care ne permite să înțelegem în profunzime contextul de funcționare a aplicației noastre. Aceasta ne ajută să identificăm și să abordăm cerințele sistemului într-un mod eficient, oferind în același timp utilizatorilor o experiență optimă de utilizare.

3. Cerințe de Sistem

3.1 Surse de Cerințe

Pentru dezvoltarea unei aplicații de planificare a alimentației și nutriție, este esențial să înțelegem nevoile viitorilor utilizatori și să satisfacem aceste așteptări printr-o funcționalitate eficientă și intuitivă. Consultarea stakeholderilor – inclusiv nutriționiștii, medicii dieteticieni și utilizatorii finali – joacă un rol crucial în definirea cerințelor de sistem, permițându-ne să înțelegem complexitatea cerințelor din acest domeniu.

3.2 Elicitația Cerințelor

Elicitația cerințelor este un proces iterativ care implică organizarea, comunicarea și negocierea cu stakeholderii. Vom folosi două metode pentru elicitarea cerințelor: Analiza Comparativă a Aplicațiilor și modelul USE-CASE.

3.2.1 Analiza Comparativă a Aplicațiilor

Analiza comparativă a aplicațiilor este o metodă instrumentală în elicitarea cerințelor, deoarece permite evaluarea concurenței și identificarea unor caracteristici care pot fi valoroase pentru utilizatori. Prin compararea NutriPlanner cu alte aplicații populare din domeniu, putem înțelege mai bine modul în care produsul nostru se poate diferenția și cum poate răspunde mai eficient nevoilor specifice ale utilizatorilor.

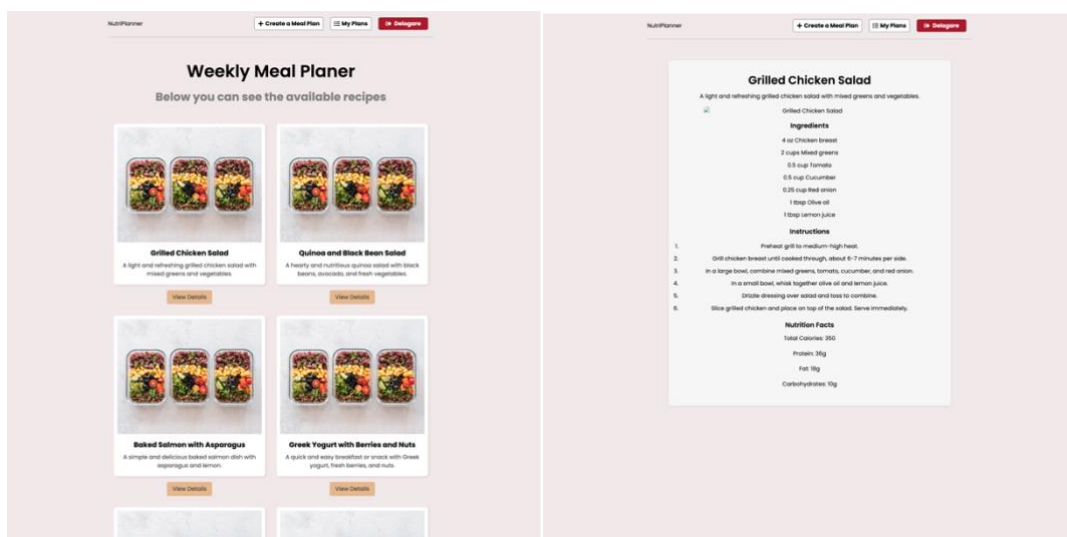


Figura 3. Pagina Principala din NutriPlanner

Figura 4. Pagina Detalii Reteta

NutriPlanner permite utilizatorilor autentificați și validați de către administrator să vizualizeze o varietate de rețete încorporate în aplicație. Aceste rețete sunt create de un expert în nutriție și includ informații esențiale, cum ar fi titlul, descrierea, instrucțiunile de preparare,

ingredientele, conținutul caloric și cantitățile de macronutrienți. Utilizatorii pot crea planuri de masă pentru fiecare zi a săptămânii, selectând din rețetele disponibile. Acestea pot include până la cinci rețete pe zi, structurate ca mic dejun, prânz, cină și două gustări.

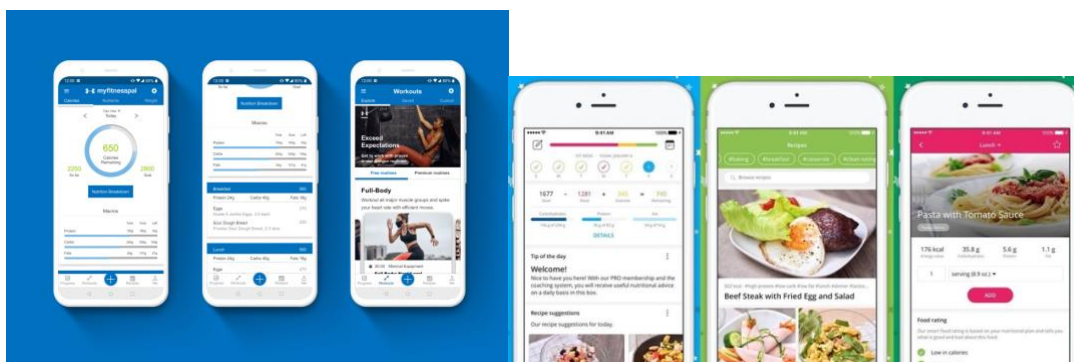


Figura 5. Ecrane din MyFitnessPal

Figura 6. Ecrane din Yazio

Pe de altă parte, MyFitnessPal oferă o gamă mai largă de funcționalități, inclusiv urmărirea activității fizice, posibilitatea de a crea propriile rețete și o bază de date vastă de alimente. Însă, poate fi mai complex și mai puțin focalizat pe planificarea meselor în comparație cu NutriPlanner.

Yazio, similar cu MyFitnessPal, oferă posibilitatea de a înregistra alimente și activități fizice, și de asemenea, permite crearea de rețete proprii. Se concentrează pe echilibrarea macro și micronutrienților, oferind sugestii de rețete pentru a ghida utilizatorii în alegerea unor opțiuni sănătoase.

Un avantaj distinct al NutriPlanner este simplitatea și focalizarea sa strictă asupra planificării meselor cu rețete pregătite de un expert în nutriție. Utilizatorii care caută o abordare simplă și directă vor găsi NutriPlanner foarte eficient. De asemenea, interfața NutriPlanner este ușor de înțeles, mobilă și compatibilă cu toate browserele moderne.

În contrast, MyFitnessPal și Yazio pot fi considerate soluții mai complexe, care oferă o gamă mai largă de funcționalități, dar pot fi mai puțin intuitive și focalizate pentru unii utilizatori care caută strict planificare de mese.

În plus, NutriPlanner este momentan gratuit, în timp ce Yazio și MyFitnessPal oferă anumite funcționalități în planuri gratuite, dar au și planuri premium cu costuri asociate.

3.2.2 Modelul USE-CASE

A doua metodă de elicitare folosită în procesul de înțelegere a funcționalităților aplicației NutriPlanner este modelul Use-Case. Acest model este folosit pentru a oferi o reprezentare vizuală a relațiilor dintre utilizatorii aplicației și sistemul în sine, sub forma unor secvențe de interacțiuni a căror finalitate este atingerea unui rezultat specific.

Astfel, printr-o analiză detaliată a scenariilor de utilizare posibile ale aplicației NutriPlanner, modelul Use-Case subliniază rolul fiecărui actor în generarea unor rezultate observabile și utile pentru utilizatori. Acest model este esențial în dezvoltarea și îmbunătățirea funcționalităților sistemului informatic, având ca scop satisfacerea nevoilor utilizatorilor.

3.2.2.1 Tabel Actori, Roluri și Obiective

Actori	Roluri	Obiective
Utilizator Neautentificat	- Se autentifică/înregistrează	- Încercarea de a accesa aplicația îl redirecționează către pagina de login. Nu are acces la conținutul aplicației până nu se autentifică.
Utilizator Înregistrat	- În așteptare pentru validare	- După autentificare, acesta nu are încă acces la funcționalitățile aplicației până când contul său nu este validat de către administrator.
Utilizator Înregistrat și Validat	- Explorare rețete - Creare/Ștergere planuri alimentare	- Explorarea tuturor rețetelor disponibile - Citirea valorilor nutritive a fiecărei rețete - Crearea și ștergerea de planuri alimentare
Administrator	- Gestionare aplicație - Validare utilizatori - Operații CRUD cu rețete	- Validarea utilizatorilor - Gestionarea bazelor de date - Rezolvarea eventualelor probleme tehnice. - Elaborarea, încărcarea, ștergerea și editarea în aplicație a rețetelor nutritive

Tabel 1. Analiza actorilor – Roluri și Obiective

În vederea clarificării funcționalităților și interacțiunilor din cadrul aplicației NutriPlanner, vom împărți analiza pe patru actori principali: Utilizatorul Neautentificat, Utilizatorul Înregistrat dar Nevalidat, Utilizatorul Înregistrat și Validat, și Administratorul.

Primul actor este Utilizatorul Neautentificat, care la accesarea aplicației este redirecționat către pagina de login. Acesta nu are acces la conținutul aplicației până nu își creează un cont și se autentifică.

După ce a trecut de pagina de login, devine Utilizatorul Înregistrat, dar nu are încă acces la funcționalitățile aplicației până când contul său nu este validat de către administrator. Odată validat, acesta devine Utilizator Înregistrat și Validat.

Utilizatorul Înregistrat și Validat are acces la toate funcționalitățile aplicației NutriPlanner. El poate explora toate rețetele disponibile, poate vedea valoarea nutritivă a fiecărei rețete, poate crea planuri alimentare personalizate și le poate șterge. Scopul principal al acestui actor este de a profita la maximum de capabilitățile aplicației pentru a-și atinge obiectivele nutriționale și de sănătate.

Al patrulea actor principal este Administratorul. Acesta are un rol dublu: pe de o parte, ca administrator, el este responsabil pentru funcționarea corectă și eficientă a aplicației, pentru gestionarea bazelor de date, pentru validarea utilizatorilor și pentru rezolvarea eventualelor probleme tehnice. Pe de altă parte, ca dietician-nutriționist, el are rolul de a elabora, de a încărca, șterge și edita în aplicație rețetele nutritive. Acest actor reprezintă pilonul central al sistemului, având o influență directă asupra calității serviciilor oferite de aplicație.

În ceea ce privește **Modelul USE-CASE**, acesta se bazează pe o reprezentare vizuală a interacțiunilor dintre acești trei actori și întregul sistem. Pentru o înțelegere mai bună, vom prezenta două scenarii: unul general, care ilustrează modul în care Utilizatorul Înregistrat și Validat interacționează cu aplicația pentru a-și crea un plan alimentar, și unul specific, care evidențiază interacțiunea dintre Utilizatorul Înregistrat și Validat, respectiv Utilizatorul Administrator/Dietician pentru a obține informații suplimentare despre o anumită rețetă sau pentru a solicita o rețetă specifică. Aceste scenarii vor pune în lumină complexitatea și interconectivitatea sistemului și ne vor ajuta să identificăm posibilele îmbunătățiri necesare.

3.2.2.2 Sistem de creare de planuri nutriționale online

Arhitectura modelului este dezvoltată accentuând instanțele care conturează cazul de utilizare selectat, subliniind relațiile dintre utilizatori, acțiunile disponibile și rezultatele obținute. Astfel, cazul de utilizare principal abordează o prezentare generală a aplicației NutriPlanner,

evidențiind etapele fundamentale pe care fiecare actor le urmează pentru a asigura asistența nutrițională online.

Prima componentă a modelului, ce reprezintă punctul de start în cadrul sistemului informatic, este mecanismul de autentificare, comun celor doi actori principali, Utilizator Înregistrat și Administrator/Dietician.

- *Autentificare Utilizator Înregistrat*

Utilizatorul Înregistrat, prin completarea datelor personale, are posibilitatea de a-și configura un cont în aplicație, ceea ce oferă accesul la un profil personalizabil, precum și la posibilitatea de modificare a parolei. Orice câmp nou completat este supus unui proces de validare, care poate identifica eventuale încălcări ale regulilor stabilite. Informațiile adăugate sau modificate în aplicație sunt administrate de serverul bazei de date prin operațiuni de tip CRUD.

- *Verificare Utilizator Înregistrat*

Ulterior înregistrării, contul utilizatorului necesită verificarea de către Administrator/Dietician pentru a se asigura autenticitatea utilizatorului. După validare, utilizatorul obține statutul de Utilizator Verificat și poate accesa funcționalitățile aplicației.

- *Configurare Plan Alimentar*

Odată autentificat, Utilizatorul Verificat are posibilitatea de a configura planuri alimentare personalizate selectând rețetele disponibile, considerând valoarea nutrițională și preferințele proprii.

- *Gestionare planuri alimentare*

După configurarea planului alimentar, acesta poate fi gestionat de către utilizator prin diverse acțiuni specifice, precum editarea, eliminarea sau adăugarea de rețete noi.

- *Accesare detalii rețetă*

Un aspect particular de gestionare a rețetelor este accesul la detalii pentru fiecare rețetă, putând fi coordonat printr-o serie de acțiuni specifice.

- *Filtrare, Căutare rețete*

Utilizatorul Verificat poate naviga în listele de rețete aplicând diverse metode care să faciliteze identificarea celor mai adecvate în funcție de nevoile specifice.

- *Creare și Încărcare rețete de către Administrator/Dietician*

Odată autentificat, Administratorul/Dieticianul poate crea și încărca noi rețete în aplicație. Aceste rețete pot fi apoi accesate de către Utilizatorii Înregistrați și Verificați.

- *Creare Plan de Masă*

Utilizatorul selectează opțiunea de creare a unui plan de masă nou, o acțiune care generează un răspuns din partea Serverului, adăugând planul de masă nou în secțiunea corespunzătoare a bazei de date. Un plan de masă poate conține până la 5 mese - mic dejun, prânz, cină și două gustări.

- *Adăugare Rețete în Planul de Masă*

După crearea planului de masă, utilizatorul poate adăuga rețete pentru fiecare masă, selectând din rețetele disponibile pentru toți utilizatorii validați. Aceste rețete sunt apoi adăugate la planul de masă.

- *Creare Plan de Mese Săptămânal*

Pentru un control mai bun al dietei, utilizatorul poate crea un plan de mese pentru fiecare zi a săptămânii, adăugând diferite planuri de masă pentru fiecare zi.

- *Salvare și Actualizare Plan de Masă*

Utilizatorul poate salva sau actualiza planul de masă oricând dorește. Aceasta va genera o notificare în sistem, menită să îl informeze pe utilizator despre acțiunea efectuată.

Aceste scenarii și cazuri de utilizare facilitează interacțiunea utilizatorilor cu sistemul, oferindu-le flexibilitate și personalizare în gestionarea planurilor lor de dietă.

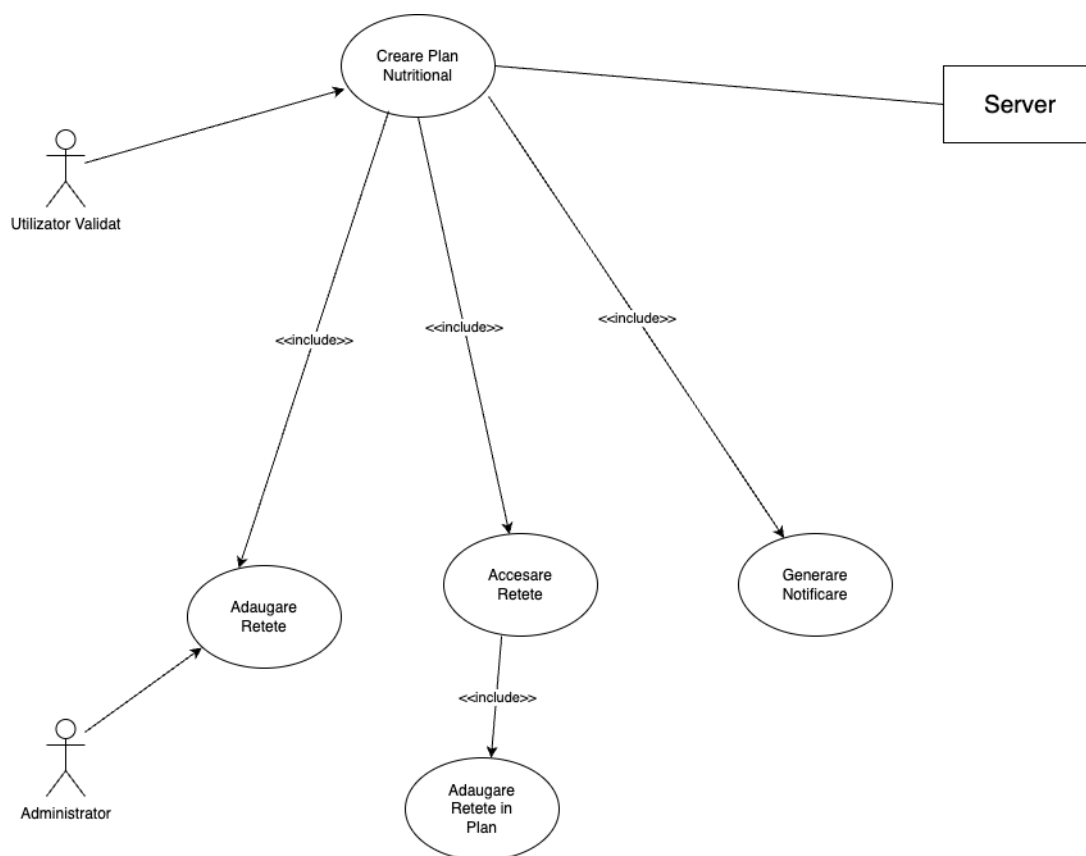


Figura 8. Diagrama USE-CASE – Caz specific de utilizare

3.3 Documentarea Cerințelor

Elementul cheie în definirea capacităților sistemului informatic NutriPlanner se regăsește în ansamblul de cerințe extrase în urma aplicării metodelor de elicitare descrise anterior. Acestea redau într-un mod autentic perspectiva stakeholderilor asupra modului în care ar trebui să arate aplicația și ce funcționalități trebuie să ofere. Conform clasificării cerințelor, condițiile necesare pentru ca platforma NutriPlanner să satisfacă nevoile în domeniul alimentației sănătoase se împart în trei categorii: cerințe funcționale, cerințe calitative și constrângeri.

Cerințele funcționale se referă la funcționalitățile pe care sistemul informatic trebuie să le ofere pentru a răspunde eficient nevoilor utilizatorilor. Printre cele mai relevante, amintim:

- **Sistem de Autentificare:** NutriPlanner include o funcționalitate de înregistrare, autentificare și gestionare a parolei, atât la accesul în aplicație, cât și din profilul personal. Parolele sunt criptate cu bcryptjs, iar autentificarea se bazează pe JWT, asigurând o securitate înaltă a datelor.
- **Gestionare Profil Utilizator:** Utilizatorii au posibilitatea de a-și modifica informațiile personale sau de a-și șterge conturile. Toate aceste acțiuni sunt validate prin token-uri de autentificare, asigurându-se astfel că modificările sunt efectuate de către utilizatorul corect.
- **Planuri de Masă Personalizate:** Utilizatorii își pot crea propriile planuri de masă, fiecare conținând până la 5 rețete pentru mic dejun, prânz, cină și două gustări. Aceste planuri pot fi personalizate pentru fiecare zi a săptămânii, oferindu-le utilizatorilor o flexibilitate maximă în gestionarea alimentației lor.
- **Acces la Rețete:** Toți utilizatorii validați au acces la o bază largă de rețete, permițându-le să exploreze noi idei și să își diversifice dieta.

Cerințele calitative definesc atributele de calitate ale întregului sistem sau ale unor elemente componente, care au valoare atât pentru utilizatori cât și pentru dezvoltatori. Ne concentrăm asupra următoarelor criterii principale de evaluare a calității sistemului informatic:

- **Eficiență și Performanță:** NutriPlanner este construit pentru a fi cât mai rapid posibil, cu imagini optimizate și cele mai bune practici utilizate pentru a asigura o experiență fluidă și rapidă pentru utilizatori.
- **Securitate:** NutriPlanner este protejat cu un certificat SSL după lansarea în producție, care se realizează pe Heroku, asigurând securitatea datelor utilizatorilor.

- **Compatibilitate:** Aplicația este compatibilă cu toate browserele moderne, asigurându-se astfel că oricine poate accesa și utiliza platforma, indiferent de dispozitivul sau sistemul de operare pe care îl folosesc.

La fel ca orice proiect, dezvoltarea NutriPlanner a implicat anumite constrângeri. Acestea includ:

- **Constrângerile de Timp:** Există o limită de timp pentru dezvoltarea și implementarea funcționalităților, precum și pentru rezolvarea eventualelor probleme tehnice.
- **Constrângeri Financiare:** Există anumite costuri asociate cu utilizarea platformelor, serviciilor sau tehnologiilor folosite în dezvoltarea NutriPlanner.
- **Constrângeri Legale:** Dezvoltarea NutriPlanner implică respectarea legilor și reglementărilor privind protecția datelor și securitatea informațiilor, printre altele.

3.3.1 Procese si Activități

3.3.1.1 Diagrama de Activități

Diagrama de activități este un instrument important pentru reprezentarea fluxurilor de activități, evidențiind interacțiunile între actori și sistemul informatic într-un proces anume și diferențiind clar între tipurile de activități întreprinse de fiecare.

Vom analiza procesul de creare a unui plan de masă în cadrul sistemului informatic NutriPlanner (Figura 9). Utilizatorul, responsabil pentru acțiunile care rezultă în generarea unui plan de masă personalizat, inițiază o serie de activități esențiale prin deschiderea aplicației NutriPlanner.

Primul nod decizional ilustrează cele două stări posibile ale utilizatorului. Pe de o parte, acesta ar fi putut fi deja conectat dintr-o sesiune precedentă, caz în care datele sale de autentificare rămân stocate în cache-ul aplicației și este direcționat direct către pagina principală de planificare a meselor. Pe de altă parte, un utilizator care nu s-a conectat anterior sau care s-a deconectat în sesiunea precedentă este redirecționat către pagina de autentificare, unde se solicită introducerea datelor de autentificare.

Acest proces de autentificare ține cont de validitatea datelor introduse. Dacă acestea nu respectă criteriile stabilite, aplicația generează un mesaj de eroare și utilizatorul trebuie să reintroducă datele de autentificare, intrând într-un ciclu care se încheie numai când se introduc datele în formatul corect. În caz contrar, dacă autentificarea este validată, aplicația încarcă pagina principală cu funcționalitatea de planificare a meselor.

Pe pagina de planificare, dacă utilizatorul nu a creat deja un plan de masă, acesta poate apăsa pe butonul de creare a unui nou plan. Dacă există deja un plan, acesta poate fi modificat

sau păstrat așa cum este. După ce se alege sau se creează un plan, utilizatorul selectează rețetele pentru fiecare masă a zilei.

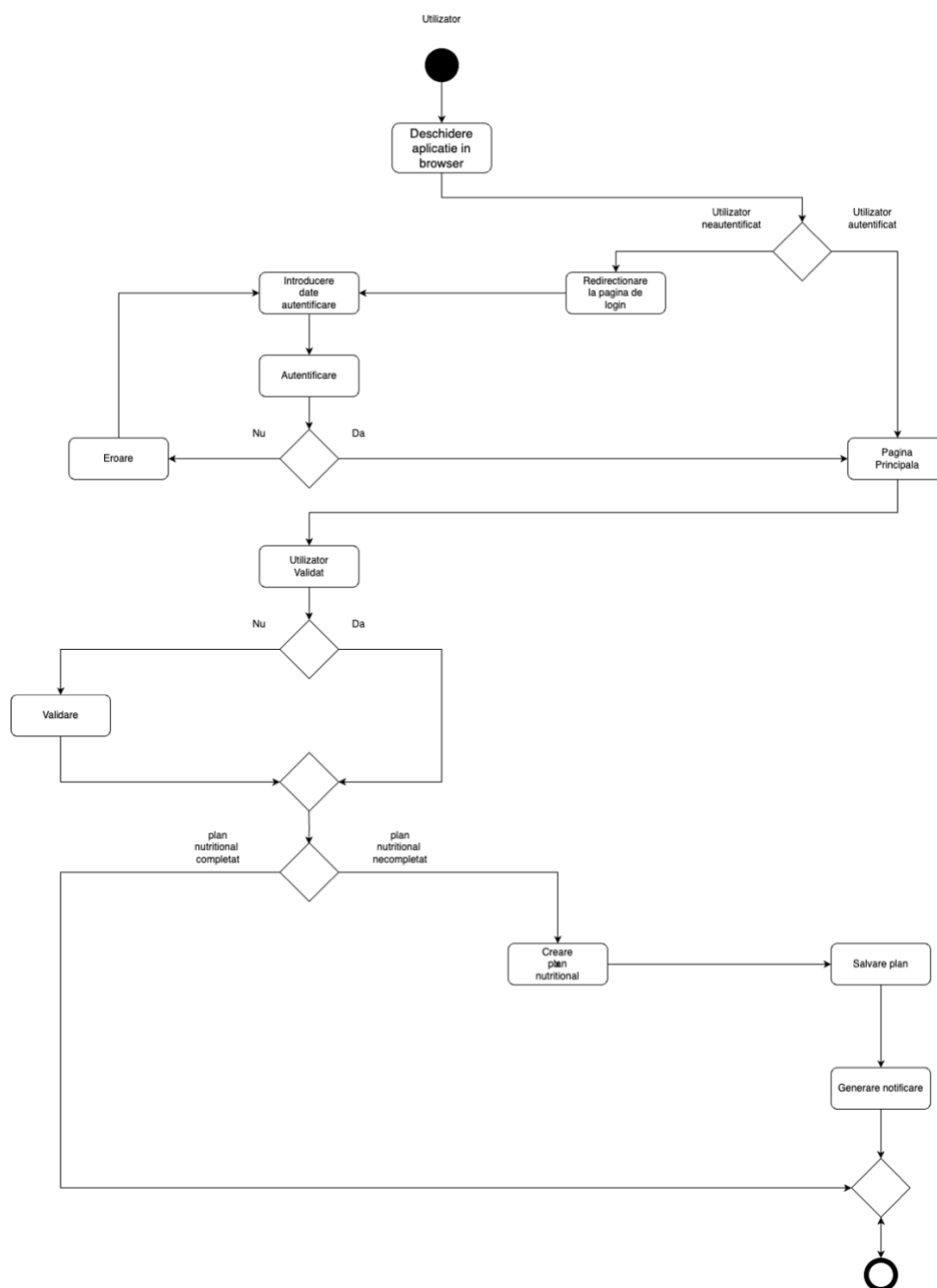


Figura 9. Diagrama de activități – Proces de creare plan

Odată ce planul este complet, sistemul îl salvează și generează o notificare pentru a anunța utilizatorul că planul de masă a fost creat cu succes. Acest proces se încheie cu nodul final de activitate, care indică îndeplinirea obiectivelor utilizatorului și închiderea fluxului de activități.

3.3.1.2 Diagrama de Stări

Diagrama de stări este un instrument grafic de reprezentare a tranzițiilor ce au loc în cadrul sistemului informatic, atât la nivel de subsisteme cât și componente. Aceste stări sunt

legate prin tranzițiile care le coordonează, legitimându-le existența într-un cadru global. Astfel, evoluția de la o stare la alta se indică prin săgeți curbe, ilustrând procesul dinamic ce inițiază punctele statice evidențiate. În acest caz, vom discuta despre stările care caracterizează procedura de creare a unui plan de masă în NutriPlanner, proces efectuat exclusiv de către utilizator (Figura 10).

Starea inițială, semnalată printr-un punct de culoare neagră, denotă momentul în care aplicația se află într-un stadiu inactiv. În momentul în care aplicația este deschisă, utilizatorul devine activ, însă neautentificat. Pentru a începe utilizarea efectivă a sistemului informatic, utilizatorul trebuie să se autentifice cu contul creat, evoluând spre starea de utilizator autentificat. În paralel, procesul de deconectare de la contul accesat se realizează prin delogare, ceea ce duce la revenirea la starea inițială. În direcția opusă a traseului stărilor, sistemul informatic devine inactiv prin închiderea aplicației, ajungând înapoi la punctul de pornire.

Ca utilizator autentificat, acesta poate aștepta pentru a fi validat.

După această etapă, cererea de acces a utilizatorului este supusă unui proces de validare de către administrator. În acest punct, dacă validarea este confirmată, utilizatorul avansează către starea de utilizator autentificat și validat, având acces la o varietate de rețete și posibilitatea de a crea un plan de masă. Dacă validarea este respinsă, utilizatorul revine la starea anterioară de utilizator nevalidat.

În momentul în care utilizatorul este validat, acesta are acces la o serie de rețete, a cărui stare este inițiată prin vizualizarea meniului cu opțiuni. Procesul de selecție a rețetelor și, implicit, de trimitere a acestora, se încheie cu succes prin salvarea datelor furnizate în baza de date. Starea de completare a planului de masă este asociată cu cea de creare a planului de masă prin apariția imediată a unei notificări care indică succesul în crearea planului, după calcularea necesarului de nutrienți bazat pe selecția de rețete.

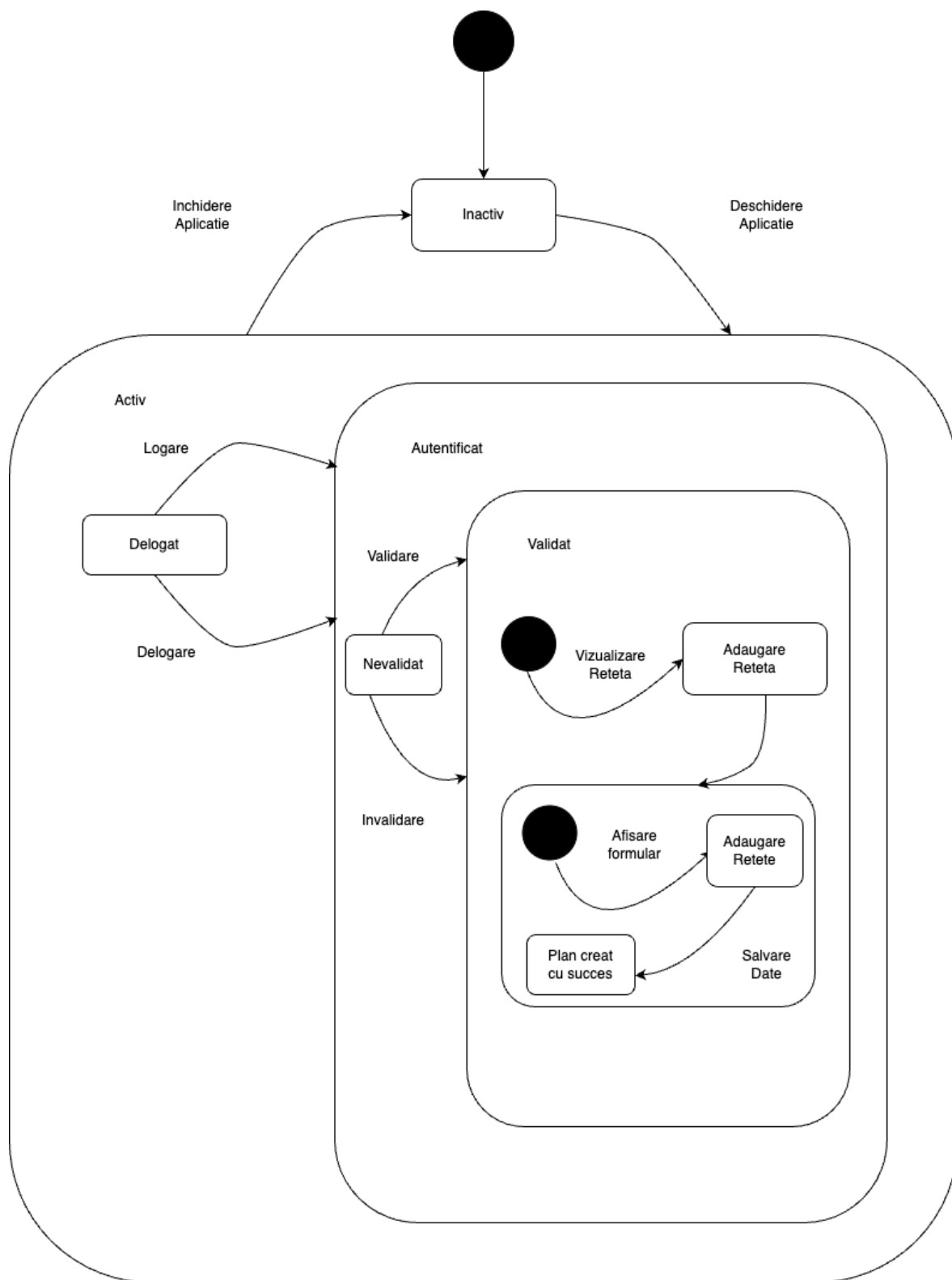


Figura 10. Diagrama de stări – Proces de creare a planului

II. Metodologia de lucru

În capitolul precedent, am analizat nevoia și motivația din spatele creării NutriPlanner, o aplicație dedicată planificării alimentare și nutriționale, precum și cerințele esențiale pe care aceasta trebuie să le îndeplinească pentru a veni în beneficiul utilizatorilor. Cunoscând clar obiectivele noastre, în acest capitol ne vom concentra asupra tehnologiilor care formează fundamentul tehnic al NutriPlanner.

Vom argumenta selecția acestor tehnologii pentru proiectul nostru și vom descrie modul în care am aplicat aceste instrumente pentru dezvoltarea și îmbunătățirea procesului de dezvoltare a aplicației. În final, vom ilustra metodologiile de lucru și practicile pe care le-am adoptat pentru a eficientiza și optimiza etapele de dezvoltare și implementare a aplicației.

1. Tehnologii de lucru

1.1 Stiva MERN

Stiva MERN a fost aleasă ca fundament tehnologic în elaborarea aplicației web contemporane datorită unui amalgam de motive strategice, în pofida altor alternative precum Angular, Vue pentru frontend, C# pentru backend, sau Next.js pentru full stack. În primul rând, React.js, ca parte a stivei MERN, prezintă o adaptabilitate superioară și un ecosistem în continuă creștere, comparativ cu Angular și Vue. Aceasta permite o scalabilitate și modularitate deosebită, esențială pentru dezvoltarea sustenabilă a aplicației. În al doilea rând, alegerea Node.js și Express.js pentru backend, în comparație cu C#, se datorează coerenței lingvistice; utilizând JavaScript atât pe frontend cât și pe backend, se asigură o unificare în dezvoltare și o mai bună comunicare între părțile componente ale aplicației. În plus, Node.js prezintă un avantaj în performanță datorită naturii sale non-blocante, și oferă o suită extinsă de biblioteci prin intermediul NPM. În ceea ce privește Next.js, deși este o alegere solidă pentru dezvoltare full stack, optarea pentru stiva MERN a fost determinată de flexibilitatea pe care o oferă componentele individuale în customizare și integrare.

MongoDB: Această bază de date a fost selectată datorită aptitudinii sale de a arhiva informații într-un format agil și maleabil, denumit JSON. Aceasta oferă posibilitatea de a structura date de natură complexă și de a le manipula cu destindere. De asemenea, MongoDB se caracterizează printr-o scalabilitate înaltă, asigurându-se că aplicația în discuție poate evolua fără a întâmpina probleme de performanță atribuite bazei de date.

Express.js: Reprezintă o opțiune judicioasă pentru construcția serverului aplicației. Este ușor de asimilat, în același timp fiind robust și adaptabil. Express facilitează demersul dezvoltării unui server, permițând o concentrare mai intensă asupra funcționalităților aplicației.

React.js: Consistă într-o bibliotecă cu mare recunoaștere în domeniu, destinată edificării interfețelor de utilizator. Contribuie la crearea unei interfețe de utilizator receptivă și accesibilă. React permite realizarea componentelor reutilizabile, ceea ce conduce la o sporire a eficacității în procesul de dezvoltare și întreținere a codului.

Node.js: Reprezintă platforma pe care se desfășoară integral aplicația noastră. Se impune ca o selecție consecventă pentru aplicațiile web full-stack în JavaScript, având în vedere că dă posibilitatea dezvoltatorilor de a recurge la același limbaj atât în sfera client, cât și în cea a serverului. Aceasta diminuează complexitatea proiectului și amplifică eficacitatea procesului de dezvoltare.

1.2 TypeScript

Pe lângă utilizarea stivei MERN, pentru partea de backend a aplicației am ales să utilizăm TypeScript. TypeScript este un limbaj de programare open-source dezvoltat de Microsoft, care se bazează pe JavaScript, adăugând tipizare statică și alte caracteristici care îmbunătățesc securitatea și eficiența codului (Microsoft, 2022). Prin utilizarea TypeScript, am reușit să îmbunătățim siguranța și ușurința de utilizare a codului nostru de backend.

1.3 Librării Folosite

Am utilizat o serie de librării adiționale în dezvoltarea NutriPlanner, atât pentru backend, cât și pentru frontend:

- Backend: Axios pentru gestionarea cererilor HTTP, bcryptjs pentru criptarea datelor sensibile, mongoose pentru modelarea datelor în MongoDB, jsonwebtoken pentru gestionarea autentificării și a securității, și uuid pentru generarea identificatorilor unici.
- Frontend: Material-UI în alcătuirea unei interfețe de utilizator atractive și interactive, Material-UI Data Grid ca vehicul pentru administrarea datelor într-o structură de grilă, redux ca unelte pentru supervizarea stării aplicației, react-router-dom pentru dirijarea navigării în aplicație, axios pentru efectuarea și gestionarea cererilor HTTP și react-modal pentru crearea de ferestre modale responsabile în cadrul interfeței.

1.4 VS Code

Toată codificarea pentru acest proiect a fost realizată în Visual Studio Code (VS Code). Acest editor de cod gratuit și open-source de la Microsoft este extrem de versatil și personalizabil, suportând o gamă largă de limbaje de programare și marcatori, inclusiv cele folosite în acest proiect.

VS Code oferă o serie de caracteristici cheie care l-au făcut ideal pentru acest proiect. Acestea includ integrare Git, evidențierea sintaxei, finalizarea inteligentă a codului, fragmente de cod pentru reutilizarea rapidă a codului, debugging în timp real și o varietate de extensii care îmbunătățesc și personalizează experiența de dezvoltare.

De asemenea, VS Code a permis dezvoltarea în TypeScript, oferind suport nativ pentru limbaj și îmbunătățind fluxul de lucru printr-o varietate de caracteristici, cum ar fi evidențierea erorilor în timp real și sugestii de cod inteligente.

VS Code a fost, de asemenea, un instrument esențial în gestionarea și organizarea codului proiectului, facilitând navigarea prin fișiere și directoare, precum și utilizarea Git pentru versionare și colaborare.

2. Model de dezvoltare

În construcția platformei NutriPlanner, am implementat modelul de dezvoltare Waterfall. Acest model, cunoscut și sub numele de modelul Cascadă, implică secvențialitatea strictă a etapelor de dezvoltare, fiecare etapă fiind dependentă de finalizarea celei anterioare.

Acest model permite o organizare clară și metodică a proiectului, cu etape bine definite și cu o structură ce facilitează o mai bună înțelegere a rezultatului final dorit. Principalele etape ale modelului Waterfall sunt:

Analiza cerințelor: În această fază, am definit clar cerințele platformei NutriPlanner. Am identificat nevoile utilizatorilor țintă, am evaluat resursele necesare și am definit obiectivele și funcționalitățile dorite ale aplicației.

Proiectare: După ce cerințele au fost stabilite, am trecut la faza de proiectare, unde am definit structura și designul platformei NutriPlanner. Am elaborat schițe, diagrame și alte materiale vizuale pentru a ilustra modul în care diversele elemente ale platformei interacționează între ele.

Implementare: Această etapă a implicat dezvoltarea efectivă a platformei, în conformitate cu cerințele și planurile definite în etapele anterioare. Am utilizat stiva MERN, TypeScript și diverse alte librării pentru a crea platforma.

Testare: În această etapă, am testat sistemul pentru a ne asigura că funcționează corect și că îndeplinește toate cerințele stabilite. Am identificat și remediat diverse probleme și erori care au apărut în timpul testării.

Lansare și Mentenanță: După ce toate problemele și erorile au fost rezolvate, am lansat platforma NutriPlanner. În continuare, vom monitoriza platforma pentru a identifica și remedia eventualele probleme care pot apărea și vom adăuga funcționalități noi în funcție de feedback-ul utilizatorilor.

Unul dintre avantajele majore ale modelului Waterfall este capacitatea sa de a permite estimări precise ale costurilor și timpului necesar pentru dezvoltare. În plus, deoarece fiecare etapă trebuie să fie completată înainte de a trece la următoarea, acest model facilitează o documentare completă a întregului proces de dezvoltare.

În alegerea modelului Waterfall, am apreciat structura sa clară și ordonată și capacitatea sa de a ne permite să ne concentrăm pe dezvoltarea platformei NutriPlanner fără a fi distrași de cerințe în schimbare. De asemenea, având în vedere natura proiectului nostru, care este concentrat exclusiv pe dezvoltarea unei platforme software, modelul Waterfall a fost cea mai potrivită alegere.

3. Model de organizare

Diagrama Gantt reprezintă un instrument grafic ce se folosește în gestionarea proiectelor pentru a urmări și organiza evenimentele pe o scală de timp. Se prezintă sub forma unui grafic cu bare orizontale, care reprezintă durata fiecărei activități în cadrul proiectului. Aceasta oferă o perspectivă clară și vizuală asupra fluxului de muncă, ajutând la identificarea duratelor, suprapunerilor și dependențelor dintre diferite activități.

Fiecare etapă este reprezentată printr-o bară orizontală, cu culori diferite pentru a facilita distincția între etape: Analiză – roșu, Proiectare – galben, Implementare și Testare – albastru, Documentare - verde.

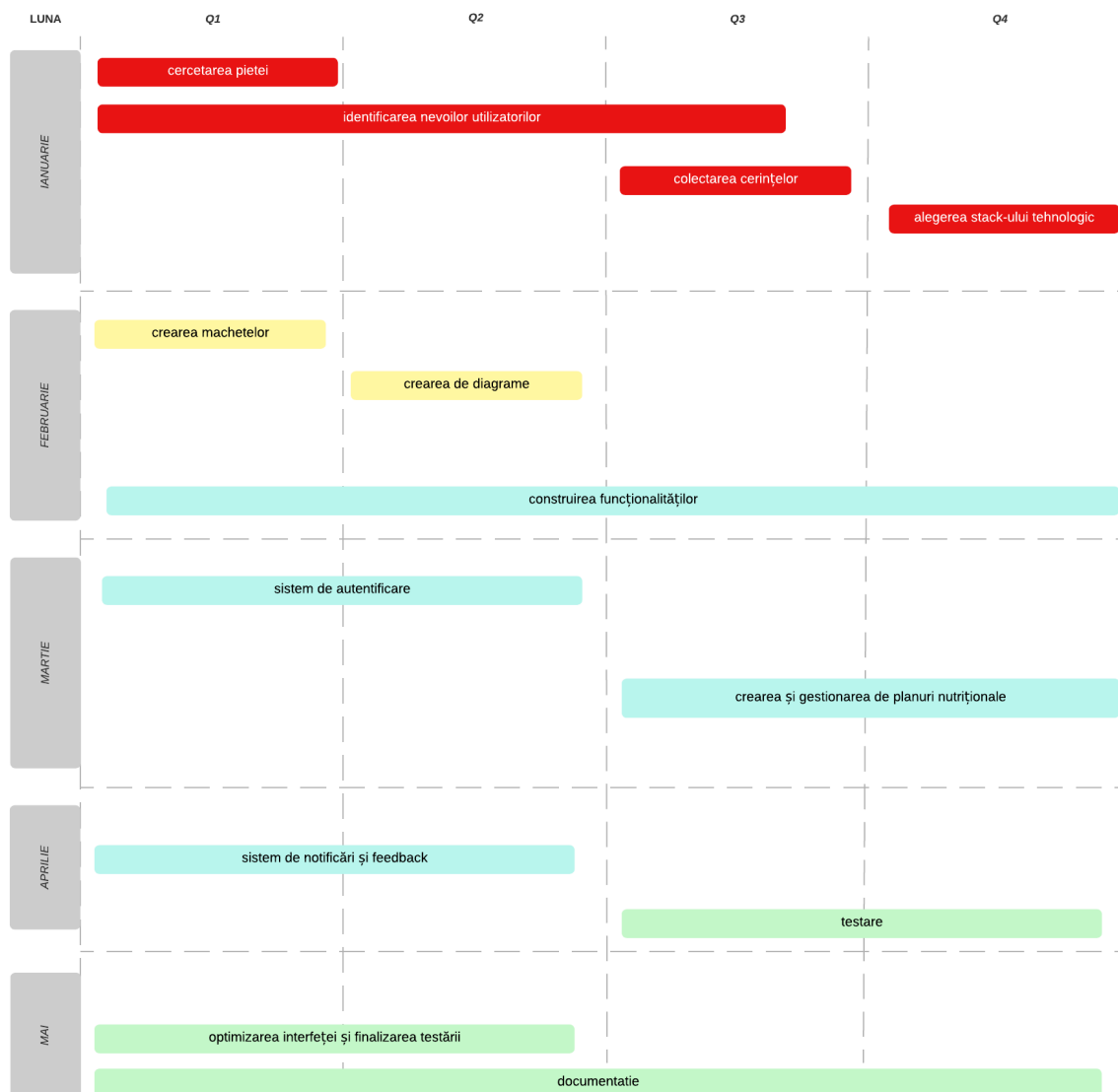


Figura 11. Diagrama Gantt - Reprezentarea temporală a etapelor în desfășurare pentru dezvoltarea NutriPlanner

În ianuarie, proiectul NutriPlanner începe cu faza de Analiză. Această etapă implică cercetarea pieței și identificarea nevoilor utilizatorilor potențiali. Acest lucru este esențial pentru a asigura că aplicația abordează probleme reale și aduce valoare. În acest timp, echipa colectează cerințe și decide asupra funcționalităților cheie pe care aplicația le va avea. Se alege stack-ul tehnologic MERN datorită versatilității și eficienței sale în dezvoltarea aplicațiilor web.

În februarie, începe faza de Proiectare. Se creează machete pentru interfața utilizatorului, asigurându-se că aceasta va fi intuitivă și atrăgătoare. În același timp, echipa de

dezvoltare lucrează la crearea de diagrame arhitecturale care detaliază modul în care diferitele componente ale aplicației vor comunica și interacționa între ele.

La sfârșitul lunii februarie, începe faza de Implementare. În această etapă, echipa de dezvoltare construiește funcționalitățile aplicației. Aceasta include configurarea bazei de date folosind MongoDB pentru a stoca și gestiona date, crearea serverului cu Express.js pentru a gestiona solicitările și răspunsurile HTTP, și dezvoltarea logicii de backend folosind Node.js. Paralel, se creează interfața de utilizator cu React.js, un framework popular pentru construirea de interfețe reactive.

În martie, se concentrează pe dezvoltarea funcționalităților cheie, inclusiv un sistem de autentificare pentru utilizatori, posibilitatea de a crea și gestiona planuri nutriționale și integrarea cu surse externe de informații despre alimente și nutriție.

În aprilie, implementarea continuă cu accent pe funcționalități care îmbunătățesc experiența utilizatorului, cum ar fi notificări și feedback. În paralel, începe faza de Testare pentru a identifica și corecta erori sau probleme de performanță în aplicație.

În mai, echipa se concentrează pe optimizarea interfeței și finalizarea testării. În plus, sunt stabilite conexiuni cu baze de date externe și alte servicii pentru a îmbogăți conținutul aplicației. În acest moment, începe și elaborarea documentației, care include manuale pentru utilizatori și documentație tehnică pentru dezvoltatori.

Astfel, prin folosirea diagramei Gantt și o planificare atentă, proiectul NutriPlanner asigură o abordare structurată în dezvoltarea aplicației, ținând cont de toate aspectele cruciale, de la analiza inițială până la implementare și testare.

III. Proiectarea Sistemului Informatic

Acest capitol va explora structura și mecanismele care guvernează aplicația web NutriPlanner. Vom examina în detaliu arhitectura sa, punând în evidență modul în care diferitele componente se îmbină pentru a oferi funcționalitățile dorite, și ne vom aprofunda în modul în care datele și evenimentele sunt gestionate și transferate în cadrul aplicației.

1. Arhitectura Sistemului

Configurația sistemului pentru aplicația NutriPlanner este alcătuită în concordanță cu principiile arhitecturii Model-View-Controller (MVC), o metodologie apreciată în sfera dezvoltării aplicațiilor web. MVC separă structura logică a aplicației în trei entități corelate, având ca rezultat o scalabilitate și o gestionare a întreținerii mult mai eficiente. Acest cadru oferă dezvoltatorilor ocazia de a optimiza anumite secțiuni ale aplicației fără a se preocupa de eventualele consecințe asupra celorlalte segmente ale sistemului.

În cadrul aplicației NutriPlanner, structura MVC este personalizată într-o manieră distinctă pentru a se plia pe cerințele unui sistem de coordonare nutrițională. Sistemul se divide în trei piloni fundamentali: Model, View și Controller.

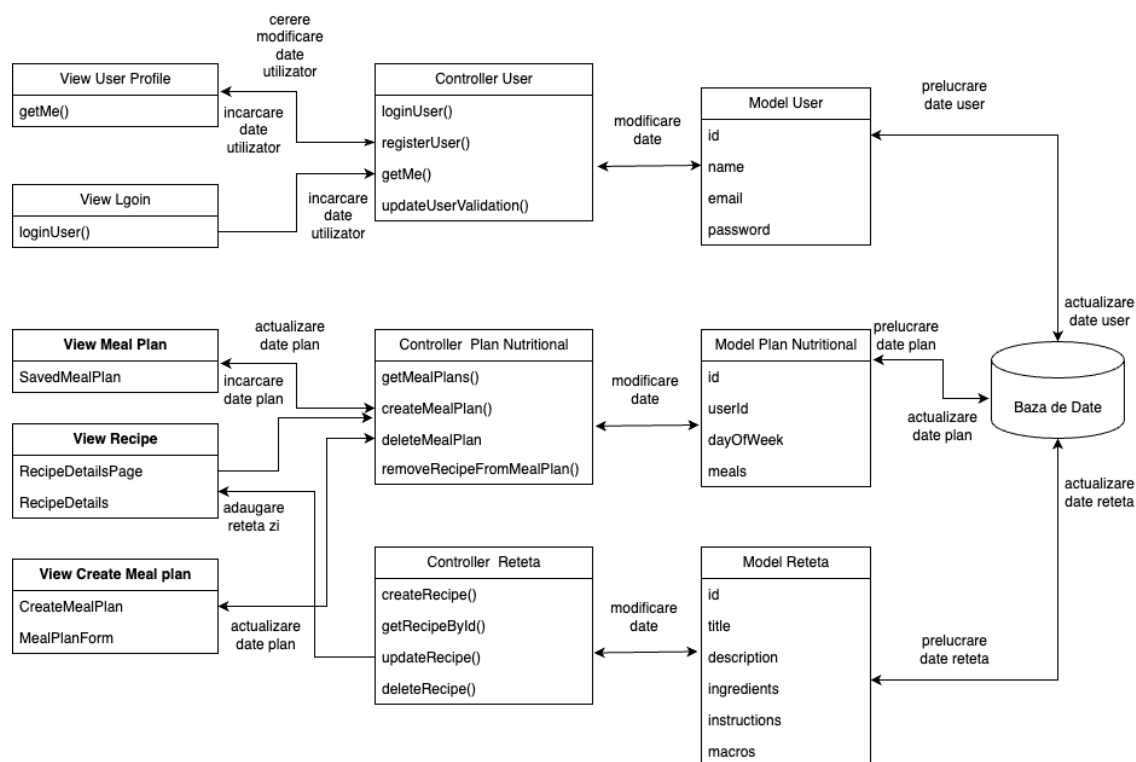


Figura 12. Arhitectura sistemului informatic - Model-View-Controller

Model: Această entitate simbolizează organizarea datelor și interacțiunea acestora cu baza de date. În cadrul NutriPlanner, avem diverse modele ce se potrivesc cu structurile de date distincte, cum ar fi Utilizator, Rețetă și Plan de Mese. De pildă, modelul Utilizator poate îngloba câmpuri precum numele, adresa de e-mail și opțiunile dietetice. Modelele cooperează cu baza de date MongoDB pentru a depozita și extrage informații.

View: Reprezintă nivelul de prezentare și este însărcinat cu afișarea datelor către utilizatori într-un format accesibil și intuitiv. NutriPlanner se bazează pe React pentru a elabora interfețe de utilizator interactive. React favorizează crearea de componente reutilizabile, implicând că anumite secțiuni ale interfeței, precum meniuri sau antet, pot fi refolosite consecvent pe întreg spectrul aplicației.

Controller: Entitatea de Controller operează ca o punte între Model și View și este mandată cu administrarea logicii de afaceri. În contextul NutriPlanner, Express, un cadru de aplicații web pentru Node.js, este folosit pentru a dezvolta controllere. Acestea supervisează solicitările HTTP venite de la clienți, procesează datele și livrează răspunsuri adecvate.

Pentru a avea o perspectivă mai nuanțată, să examinăm un exemplu concret, și anume procedura de adăugare a unei rețete într-un plan de mese:

- Utilizatorul selectează o rețetă printr-o interfață grafică (View) și alege să o adauge în planul de masă pentru o zi specifică.
- View-ul trimite o solicitare HTTP POST către Controller cu detaliile rețetei și planului de masă.
- Controllerul validează informațiile primite și, dacă sunt valide, face o cerere către Model pentru a adăuga rețeta la planul de masă în baza de date MongoDB.
- După adăugarea cu succes a rețetei, Controllerul returnează un răspuns pozitiv înapoi la View.
- View-ul actualizează interfața, confirmând utilizatorului că rețeta a fost adăugată cu succes.

2. Diagrama Flux de Date

Diagrama Flux de Date în aplicația NutriPlanner oferă o reprezentare vizuală a modului în care informațiile circulă și se modifică în cadrul sistemului. Fiecare etapă din diagramă marchează o transformare sau un proces asupra datelor care sunt transmise între diferitele componente ale aplicației. În Figura 13, prezentăm diagrama fluxului de date pentru NutriPlanner, având în vedere actorii și funcționalitățile specificate.

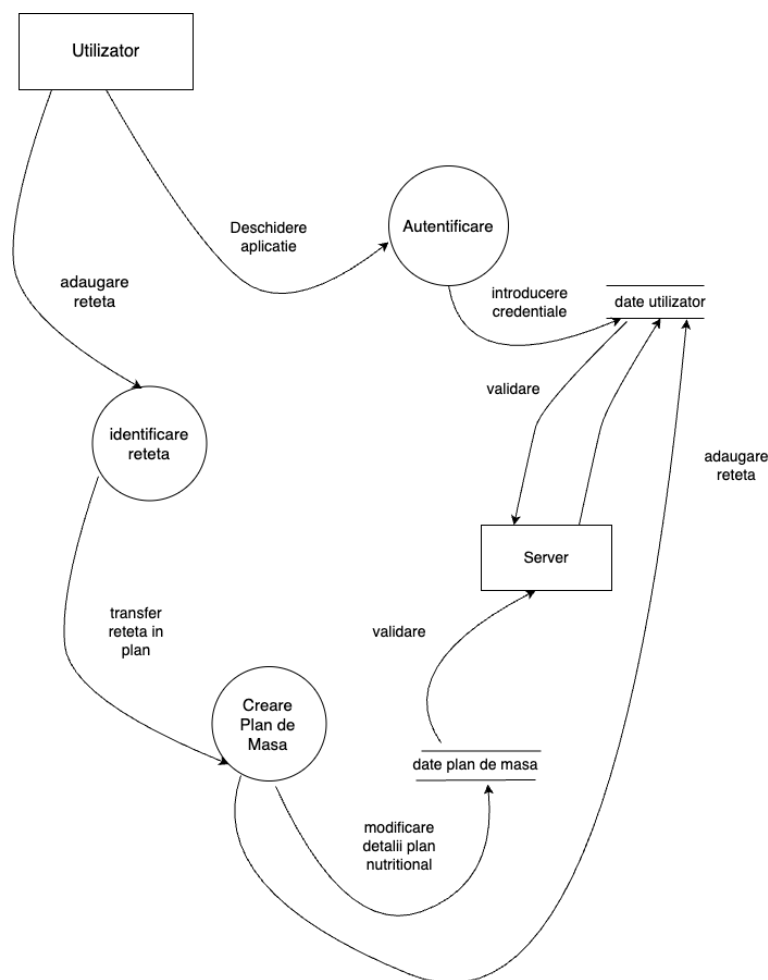


Figura 13. Diagrama Flux de Date

În diagrama ilustrată în Figura 13, observăm că principalii actori sunt Utilizatorul Administrator, Utilizatorul și Serverul. Utilizatorul Administrator inițiază procesul de autentificare prin introducerea credențialelor proprii și, odată autentificat, poate accesa funcționalitățile specifice rolului său, cum ar fi efectuarea de operații CRUD (Create, Read, Update, Delete) asupra rețetelor. De asemenea, Utilizatorul Administrator are capacitatea de a valida sau devalida conturile utilizatorilor. Dacă un cont de utilizator nu este validat, acesta nu poate realiza acțiuni în aplicație, ci este redirecționat către o pagină care afișează un mesaj de așteptare pentru validare.

În ceea ce privește Utilizatorul, acesta poate efectua acțiuni de înregistrare, autentificare și închidere a sesiunii. Odată validat de către Administrator, utilizatorul poate vizualiza toate rețetele disponibile pe pagina principală, poate selecta o rețetă specifică pentru a vedea detaliile acesteia, accesa o pagină unde poate crea planuri de masă selectând până la 5 rețete pe zi pentru maximum 7 zile consecutive, vizualiza planurile de masă create într-o pagină separată, și șterge o rețetă din planul de masă sau poate șterge întregul plan de masă pentru o anumită zi.

Serverul joacă un rol esențial în gestionarea autentificării și a sesiunilor pentru Utilizator și Administrator, procesează și validează datele trimise de actori, gestionează stocarea și accesul la baza de date asigurându-se că informațiile sunt corecte și actualizate, și răspunde la solicitările de date ale Utilizatorilor și Administratorului.

Pentru a exemplifica fluxul de date în NutriPlanner, să considerăm scenariul în care un utilizator adaugă o rețetă la planul său de masă. Utilizatorul navighează prin interfața grafică și alege să adauge o rețetă la planul de masă pentru o anumită zi. O solicitare este trimisă către server, care include informații despre rețeta selectată și ziua în care va fi inclusă în planul de masă. Serverul procesează solicitarea, validând datele și actualizând baza de date cu noul plan de masă. O confirmare este trimisă înapoi la utilizator, iar interfața grafică este actualizată pentru a reflecta noile schimbări în planul de masă.

Astfel, prin diagrama fluxului de date, putem vizualiza și înțelege modul în care informațiile sunt gestionate și transformate în cadrul sistemului NutriPlanner.

3. Diagrama pe Componente

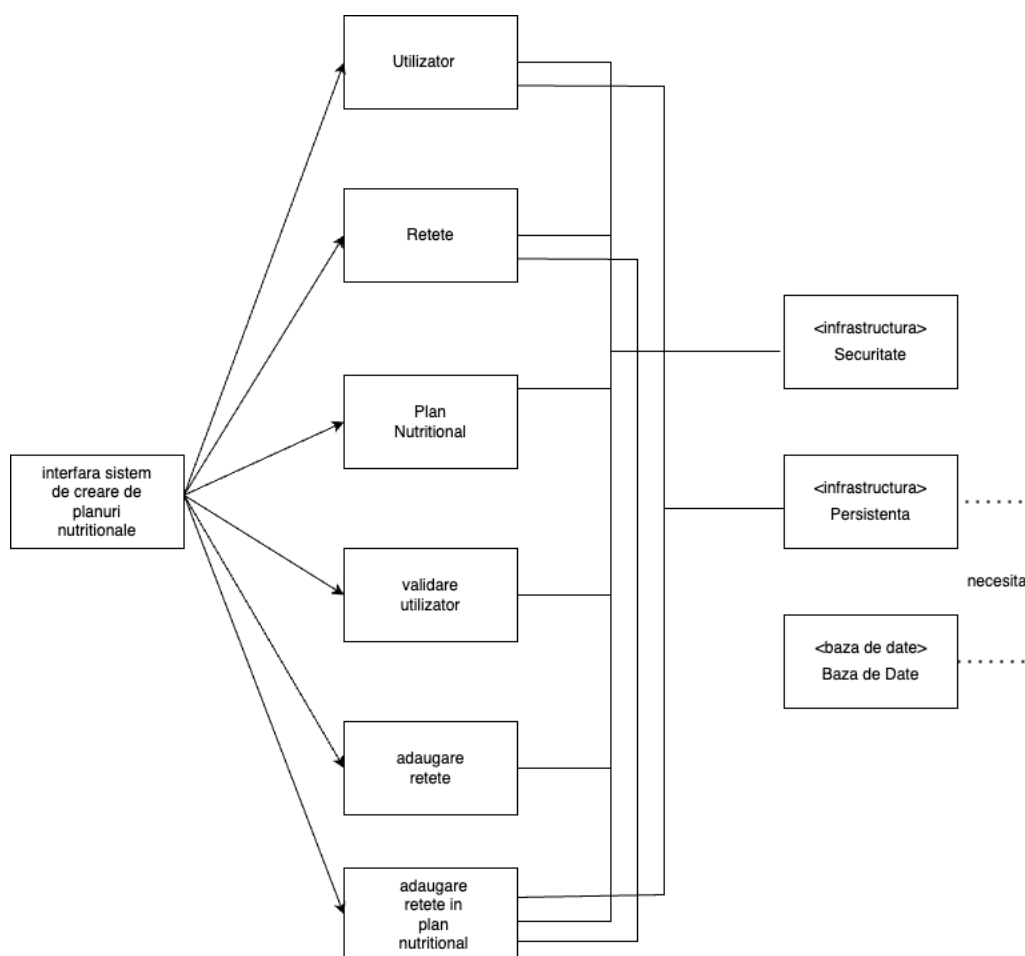


Figura 14. Diagrama pe componente

În scopul de a dezvălui componentele software și relațiile dintre acestea care formează sistemul NutriPlanner, vom dirija atenția asupra detaliilor Diagramei pe Componente. Pachetul pe care îl analizăm, reprezentând aplicația web NutriPlanner în întregime, include componente interconectate ce asigură eficacitatea funcționalităților implementate.

Nucleul acestui sistem, intitulat NutriPlanner, furnizează interfața principală în care sunt încorporate diverse componente. Una dintre componentele cruciale de care depinde NutriPlanner este AdminDashboard, care include interfața și modulele aferente pentru validarea utilizatorilor și gestionarea rețetelor. Într-o poziție similară se află și componenta CreateMealPlanPage, care permite utilizatorilor să compună planuri de mese și să selecteze rețete pentru integrare în aceste planuri, având o conexiune directă cu baza de date pentru salvarea planurilor.

De asemenea, avem componenta Home, care servește ca punct de intrare în aplicație și unde utilizatorii validați pot răsfoi rețetele disponibile, în timp ce cei nevalidați sunt întâmpinați cu un mesaj de așteptare. De asemenea, componentele Login și Register sunt configurate astfel încât să permită o navigare facilă între ele pentru utilizatori, deși funcționează independent. În plus, avem componenta RecipeDetailsPage, care oferă informații detaliate despre o rețetă selectată.

O altă componentă importantă, în corelare cu NutriPlanner, este SavedMealPlansPage, care permite utilizatorilor să gestioneze și să revizuiască planurile de mese pe care le-au creat și salvat înainte. Aceasta necesită autentificarea utilizatorilor și accesul la baza de date.

Toate componentele menționate sunt interconectate prin intermediul NutriPlanner, care facilitează accesul la date. În lumina responsabilităților și funcționalităților fiecărei componente, securitatea în accesul la date este imperativă. Astfel, există o componentă de Securitate și Acces la Date, care se asigură că autentificarea se face în mod sigur cu token-uri JWT și că parolele sunt criptate înainte de stocare.

De asemenea, accesul la date implică și menținerea persistenței acestora, o componentă care este gestionată printr-o conexiune specifică și necesită accesul la baza de date MongoDB prin intermediul librăriei mongoose.

4. Diagrama de Deployment

Pentru a avea o înțelegere mai clară asupra arhitecturii NutriPlanner și modul în care componentele sale interacționează, este necesară examinarea dispoziției fizice a acestora.

Figura 15 redă o reprezentare vizuală a arhitecturii client-server a NutriPlanner, evidențiind modul în care componentele sunt desfășurate și comunică între ele.

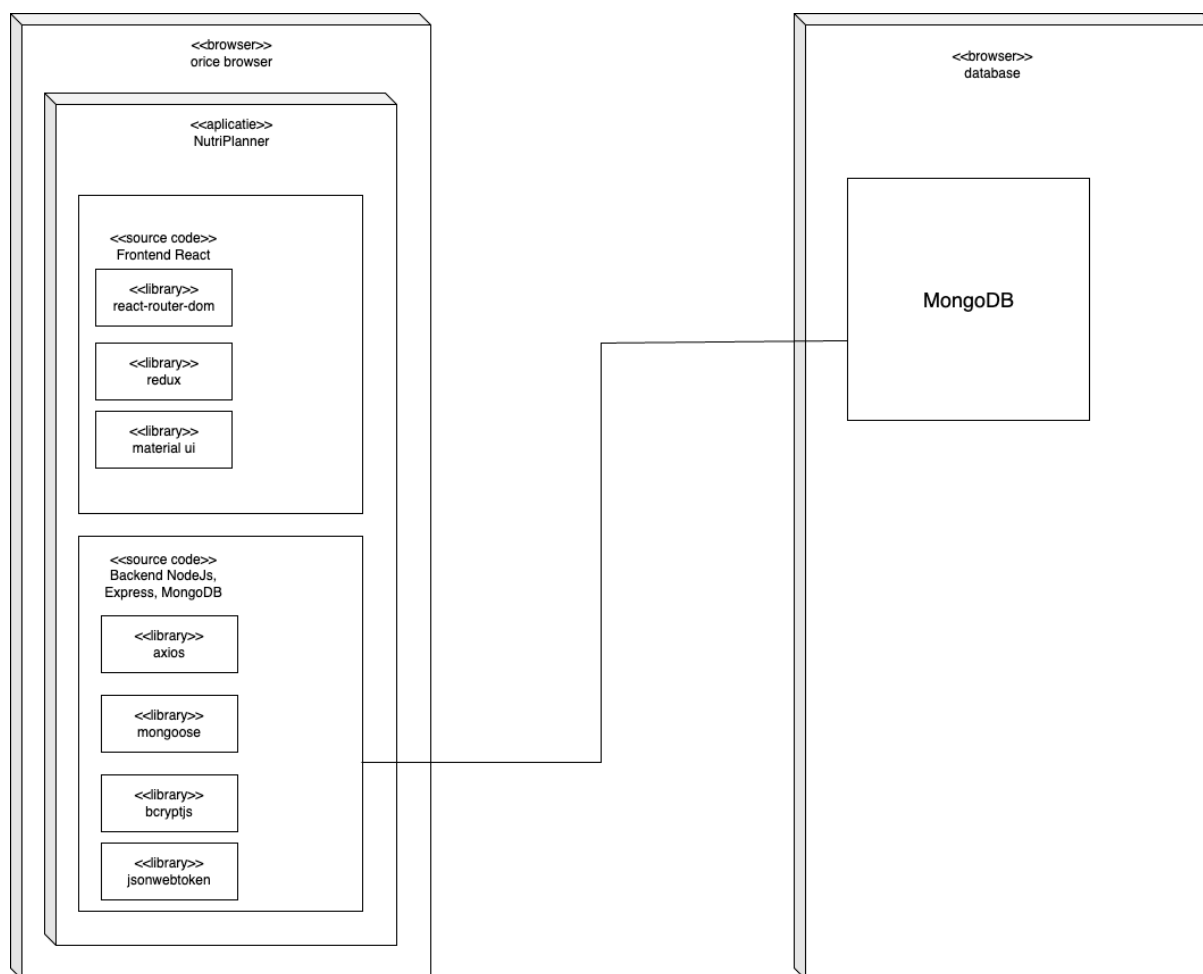


Figura 15. Diagrama de Deployment

În partea de client, NutriPlanner utilizează React, o bibliotecă JavaScript, pentru a construi interfața cu utilizatorul. React permite crearea de componente reutilizabile și performante prin utilizarea JSX, o sintaxă de tip template care facilitează scrierea codului de interfață într-un mod mai expresiv și lizibil. Fișierele JSX sunt însoțite de fișiere JavaScript (.js) pentru gestionarea logicii.

În ceea ce privește testarea în partea de client, NutriPlanner folosește Jest, un framework de testare popular pentru aplicații JavaScript și React. Jest ajută la asigurarea calității și stabilității codului de frontend prin realizarea de teste automate.

În ceea ce privește partea de server, NutriPlanner este construită folosind TypeScript împreună cu Node.js. TypeScript, care este un superset al JavaScript, permite o dezvoltare mai

structurată și sigură prin adăugarea de tipuri statice. Pentru gestionarea cererilor și răspunsurilor HTTP, se utilizează Express.js, un framework web minimalist pentru Node.js. MongoDB, o bază de date NoSQL, este utilizată pentru stocarea și gestionarea datelor. Cu ajutorul bibliotecii Mongoose, NutriPlanner poate modela și manipula eficient datele în format JSON.

În ceea ce privește testarea în partea de server, NutriPlanner utilizează Mocha, un framework de testare pentru aplicații JavaScript rulate în Node.js. Acesta ajută la verificarea corectitudinii și fiabilității logicii de backend.

Comunicarea între client și server este facilitată prin WebSocket, o tehnologie care permite schimbul de date în timp real printr-o conexiune bidirecțională.

În sinteză, diagrama de deployment pentru NutriPlanner dezvăluie o structură client-server robustă, cu o selecție bine gândită de tehnologii care contribuie la performanța, interactivitatea, și securitatea aplicației.

5. Mecanisme funcționale

5.1 Algoritmi și prelucrări interne

În sistemul NutriPlanner, procesarea datelor se face cu ajutorul unui set divers de algoritmi și structuri de date care sunt adaptate la cerințele specifice ale aplicației. De exemplu, algoritmul de autentificare este esențial pentru a asigura că numai utilizatorii înregistrați au acces la anumite funcționalități ale aplicației, iar algoritmul de creare a unui plan de masă permite utilizatorilor să creeze și să salveze planuri de masă personalizate.

Unul dintre elementele cheie în prelucrarea datelor în NutriPlanner este folosirea de obiecte JavaScript pentru stocarea și manipularea datelor într-un format structurat. Acest lucru este în concordanță cu formatul JSON, care este utilizat în mod extensiv pentru comunicarea datelor între client și server, precum și pentru stocarea datelor în MongoDB.

Pe partea de backend, structura este împărțită în trei entități principale: Users, MealPlans și Recipes. Fiecare dintre aceste entități are propriul model, rute și controller, ceea ce facilitează o abordare modulară și o mai bună organizare a codului. Bibliotecile utilizate în backend includ axios pentru cererile HTTP, bcryptjs pentru hash-ingul parolilor, express ca framework web, express-async-handler pentru manipularea erorilor asincrone, jsonwebtoken pentru autentificare și mongoose pentru modelarea obiectelor.

În partea de frontend, NutriPlanner beneficiază de un design bine structurat, împărțit în pagini, cu componente reutilizabile stocate într-un folder de componente. Se utilizează Material

UI pentru a oferi o interfață de utilizator modernă și atrăgătoare. Mai mult, Material UI Data Grid este folosit pentru gestionarea utilizatorilor de către administrator, oferind o vedere tabulară detaliată și opțiuni de sortare și filtrare.

React Hooks, în special useEffect și useState, sunt utilizate pentru a gestiona și actualiza starea componentelor. Acestea sunt cruciale pentru a reflecta schimbările în timp real în interfața cu utilizatorul, cum ar fi actualizarea informațiilor într-un grid când un utilizator este validat sau invalidat.

În ceea ce privește bibliotecile utilizate pe partea de frontend, NutriPlanner se bazează pe un set de biblioteci, inclusiv @material-ui/core pentru componente UI, @material-ui/icons pentru iconițe, react-redux pentru gestionarea stării, react-router-dom pentru rutare, și axios pentru cereri HTTP, printre altele.

NutriPlanner implementează, de asemenea, diverse metode de optimizare a performanței și scalabilității, cum ar fi indexarea în MongoDB și paginarea datelor pentru a reduce volumul de date transferate la fiecare cerere și pentru a îmbunătăți timpul de răspuns.

5.2 Structura Datelor

În faza de proiectare a aplicației NutriPlanner, un aspect crucial îl reprezintă modelarea și organizarea datelor. Diagrama Entitate-Relație este un instrument eficient în acest sens, contribuind la documentarea și vizualizarea cerințelor legate de structura datelor. Aceasta ajută la identificarea elementelor fundamentale cum ar fi entitățile, relațiile între entități și atributele semnificative, asigurând o bază solidă pentru dezvoltarea sistemului informatic.

NutriPlanner utilizează MongoDB ca sistem de gestiune a bazelor de date, care este cunoscut pentru flexibilitatea sa în stocarea datelor sub formă de documente. În Figura 16 este prezentată structura datelor, ilustrând modul în care sunt organizate și interconectate entitățile principale ale aplicației: Users, MealPlans și Recipes.

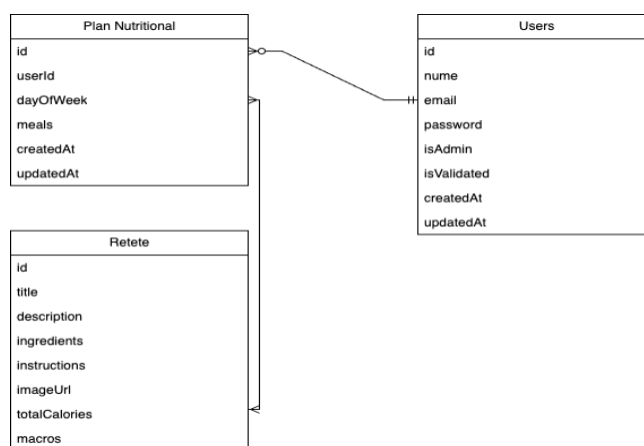


Figura 16. Structura bazei de date

Prima entitate pe care o vom discuta este Users. Aceasta servește drept punct central pentru gestionarea conturilor de utilizator în cadrul aplicației NutriPlanner. Fiecare utilizator are attribute precum nume, email, parolă, indicator dacă este administrator (isAdmin) și un indicator de validare (isValidated) care verifică dacă contul unui utilizator este activ și autentificat. Această entitate este într-o relație de tip One-To-Many cu entitatea MealPlans, semnificând faptul că un utilizator poate crea mai multe planuri alimentare, însă fiecare plan este asociat cu un singur utilizator.

În continuare, entitatea MealPlans este o componentă esențială în NutriPlanner, având rolul de a gestiona și organiza planurile alimentare create de utilizatori. Entitatea conține un ID al utilizatorului asociat (userId), ziua săptămânii (dayOfWeek) și mesele, care sunt structurate în categorii: mic dejun (breakfast), prânz (lunch), cină (dinner), gustare1 (snack1) și gustare2 (snack2). Fiecare masă poate conține mai multe rețete prin intermediul unui array de ID-uri de rețete (recipeId). Această structură permite flexibilitate în alocarea rețetelor la diferite mese și zile.

În final, entitatea Recipes gestionează informațiile referitoare la rețetele culinare. Aceasta conține titlul rețetei, o descriere, o listă de ingrediente cu nume, cantitate și unitate de măsură, instrucțiuni de preparare sub formă de array de stringuri, un URL către o imagine reprezentativă, numărul total de calorii și o descompunere a macronutrienților în proteine, grăsimi și carbohidrați.

6. Interfața cu utilizatorul

În această secțiune, vom explora componentele esențiale ale interfeței cu utilizatorul în cadrul aplicației NutriPlanner, concentrându-ne pe schema de culori, elementele de design și structura paginilor, care împreună contribuie la realizarea unei experiențe utilizator coerente și eficiente.

Primul aspect, schema de culori, a fost meticolos ales pentru a reflecta o estetică agreabilă, fără a compromite funcționalitatea. Astfel, fundalul general al aplicației este într-o nuanță caldă de roz deschis (hex #f1e8ea), în timp ce pentru antet, casetele de conținut și butoane se utilizează un alb pur (ffffff), care contrastează armonios cu textul negru (#000000). Bordura casetelor de conținut este conturată de un gri discret (#e6e6e6). În plus, pentru a conferi un aspect dinamic, butoanele folosesc culorile roșu închis (#ad192a) sau negru (#000000), în funcție de importanța acțiunii pe care o reprezintă.

La nivelul elementelor de design, aplicația NutriPlanner implementează caracteristici vizuale moderne. Fontul Poppins, în diferite greutăți și stiluri, oferă o textură variată în

conținut. Meniul orizontal, localizat în antet, facilitează navigarea printr-un set de link-uri negre, care schimbă culoarea în gri închis când sunt selectate cu cursorul, semnalând astfel interacțiunea. Casetele de conținut servesc ca instrumente de structurare, împărțind informația într-un mod clar și ordonat. Butoanele cu aspect reliefat sunt configurate să se redimensioneze ușor atunci când sunt selectate, furnizând feedback vizual. Formularele pentru introducerea datelor sunt concepute pentru eficiență, având etichete explicite și câmpuri de introducere bine definite. Ilustrațiile și vectorii sunt utilizați în mod judicios pentru a oferi indicii vizuale și a îmbogăți estetica generală, fără a suprasatura spațiul vizual.

În ceea ce privește structura paginilor, aceasta se adaptează la specificul conținutului și funcționalității. Elementele omniprezente în toate paginile sunt antetul, care găzduiește meniul orizontal, și footer-ul care oferă informații suplimentare. Anumite pagini, cum ar fi cea de înregistrare sau de planificare alimentară, adoptă o estetică Light, beneficiind de un fundal roz.

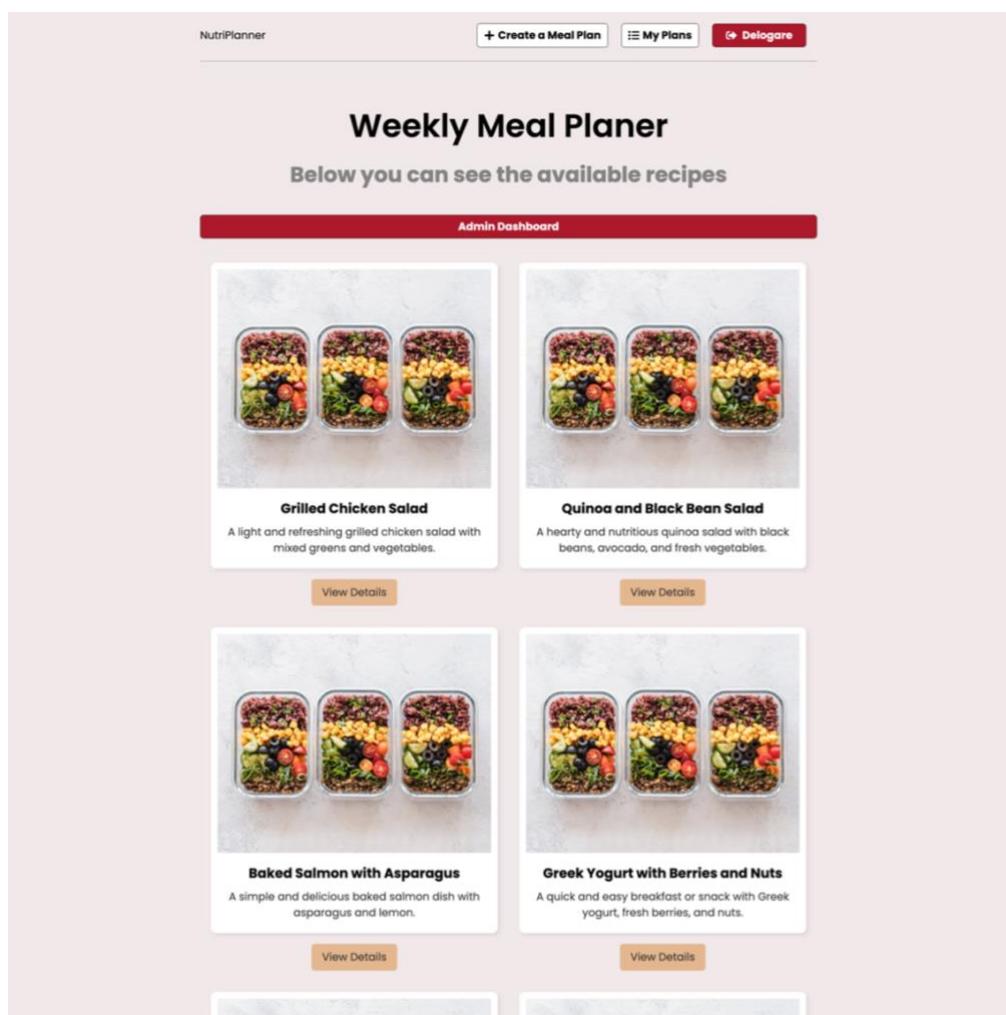


Figura 17. Pagina Home pentru Administrator

IV. Implementare și Testare

În cadrul acestui capitol, vom examina elementele fundamentale ale aplicației NutriPlanner și vom detalia procesele implicate în dezvoltarea și testarea acestora. Vom începe cu o prezentare concisă a mediului de dezvoltare și modul în care este organizat codul sursă. Mai departe, vom aprofunda particularitățile implementării în sine, urmând apoi să ne concentrăm asupra procesului de testare pentru a asigura calitatea și fiabilitatea aplicației.

1. VS Code - Configurarea Mediului de Dezvoltare și Organizarea Codului

VS Code a fost ales ca editor de cod în proiectul NutriPlanner, datorită performanței sale superioare și eficienței în gestionarea resurselor comparativ cu alte editoare, cum ar fi Atom, care poate fi mai lent și mai consumator de resurse. În plus, integrarea Git din VS Code este mai profundă și mai fluidă, fără necesitatea extensiilor suplimentare pe care Sublime Text și Atom le-ar necesita pentru a atinge funcționalitatea similară.

Un alt avantaj distinctiv al editorului VS Code este suportul extins pentru extensii și nivelul înalt de personalizare, care se dovedește a fi mai variat și mai bine integrat decât în cazul celorlalte editoare. Deși WebStorm vine cu multe caracteristici încorporate, nu permite același grad de personalizare pe care îl oferă VS Code.

Instrumentele inteligente de debugging și sugestiile de codificare în VS Code reprezintă un alt avantaj, îmbunătățind productivitatea prin oferirea unor unelte de calitate într-un pachet accesibil. Deși WebStorm are funcționalități similare, VS Code îmbină aceste unelte într-un pachet mai ușor.

Comunitatea activă și suportul continuu din partea Microsoft sunt, de asemenea, factori determinanți în alegerea VS Code. Popularitatea în creștere a acestui editor asigură actualizări regulate, îmbunătățiri și o bază solidă de cunoștințe pentru dezvoltatori.

În plus, interfața modernă și intuitivă a VS Code creează un mediu plăcut pentru dezvoltare, permitând configurări personalizate ale interfeței. Aceasta este în contrast cu interfețele mai rigide ale unor alte editoare.

În contextul specific al aplicației NutriPlanner, unde TypeScript este folosit pentru backend, suportul nativ pentru TypeScript în VS Code reprezintă un avantaj semnificativ, deoarece permite o integrare fără probleme și eficientă cu limbajul de programare ales.



Figura 18. Structura Folderelor

În Figura 18 se poate observa organizarea generală a fișierelor într-un proiect dezvoltat în Visual Studio Code. Proiectul exemplificat, denumit NutriPlanner, este organizat în două componente principale: backend și frontend.

În cadrul folderului backend, sunt prezente trei compartimente principale: controllers, models și routes. Fișierele din controllers se ocupă de manipularea și gestionarea cererilor primite de la client, controlând logica aplicației și procesarea datelor. Folderul models definește structura datelor utilizate în aplicație și este responsabil pentru interacțiunea cu baza de date. Dosarul routes gestionează rutarea cererilor către controlerele corespunzătoare, definind căile API și modul în care aplicația răspunde la diferite tipuri de cereri HTTP.

La nivelul codului sursă în frontend, interacțiunea cu utilizatorul se realizează prin componente, pagini și caracteristici (features). Componentele sunt blocurile de construcție ale interfeței cu utilizatorul și pot fi reutilizate în diverse părți ale aplicației. Paginile reprezintă secțiuni complete ale aplicației și sunt compuse din mai multe componente. Caracteristicile gestionează serviciile și starea aplicației, inclusiv funcționalități precum autentificarea utilizatorului și comunicarea cu serverul.

Conform principiilor ingineriei software, structurarea adecvată a codului și organizarea fișierelor într-o manieră logică și coerentă sunt cruciale pentru menținerea și scalarea aplicației într-un mod eficient. VS Code oferă instrumentele necesare pentru a realiza acest lucru într-un mediu modern și intuitiv.

2. Implementare

2.1 Backend

Secțiunea curentă se va concentra asupra examinării amănunțite a structurii și funcționalității segmentului backend al aplicației, având în vedere că aceasta constituie fundația sistemului. Vom pune accentul pe modul în care sunt organizate și interconectate directoarele și fișierele, de asemenea vom discuta contribuția diverselor librării în asigurarea funcționalității coezive a aplicației. În plus, vom furniza explicații cuprinzătoare privind rolul fiecărui fișier, oferind contextul necesar pentru înțelegerea capturilor de ecran ce vor însoți textul.

În ceea ce privește structura directorului backend, fișiere esențiale precum `server.ts` și `types.ts` sunt proeminente. Fișierul `server.ts` joacă un rol central în inițierea aplicației, ocupându-se de configurarea serverului și orchestrarea interacțiunii dintre rute, middleware și servirea fișierelor statice. Între timp, fișierul `types.ts` se angajează în stabilirea tipurilor personalizate folosite în aplicație.

Mai departe, directorul de configurare, denumit `config`, adăpostește fișiere ce contribuie la specificația parametrilor sistemului, precum fișierul `db.ts`, esențial în realizarea unei conexiuni adecvate cu baza de date MongoDB.

În ceea ce privește controlerele, acestea sunt așezate în directorul numit `controllers`. Aici, fiecare fișier este însărcinat cu gestionarea logicii de afaceri pentru diverse componente ale aplicației, fie că se referă la utilizatori, rețete sau planuri de mese.

Directorul intitulat `models` găzduiește schemele de date ce reprezintă structurile informaționale pentru diverse entități din sistem, cum ar fi utilizatorii și rețetele.

Avansând în înțelegerea arhitecturii, directorul routes include fișierele de rutare, care sunt responsabile pentru definirea punctelor de terminație ale API-ului, esențiale în gestionarea datelor.

În ceea ce privește directorul middleware, acesta cuprinde fișiere care servesc drept straturi intermediare, capabile să execute diverse operații înainte de prelucrarea logică din controlere. Exemple includ validarea autentificării și gestionarea erorilor.

Este important să menționăm că backend-ul integrează o serie de biblioteci și cadre de dezvoltare, precum Express pentru a facilita dezvoltarea API-ului web, Mongoose pentru a îmbunătăți interacțiunea cu MongoDB, jsonwebtoken pentru a asigura autentificarea și bcryptjs pentru securizarea parolelor.

Această arhitectură, împreună cu utilizarea adecvată a bibliotecilor și a cadrului de dezvoltare, contribuie semnificativ la crearea unui backend eficient și securizat.

Focalizându-ne pe server.ts, acest fișier joacă rolul de punct de intrare în sistem. Configurând serverul, stabilind conexiunea cu baza de date și specificând rutele API, acesta este esențial în orchestrarea aplicației. Să examinăm fiecare segment al acestui fișier în detaliu:



```
1 import path from 'path'
2 import express, { Request, Response, NextFunction } from 'express'
3 import 'colors'
4 import dotenv from 'dotenv'
5 import { errorHandler } from './middleware/errorMiddleware'
6 import connectDB from './config/db'
7 import generateSitemap from './sitemap'
8
9 import userRoutes from './routes/userRoutes'
10 import recipeRoutes from './routes/recipeRoutes'
11 import mealPlanRoutes from './routes/mealPlanRoutes'
```

Figura 19. Importarea modulelor și bibliotecilor necesare

Inițial, importăm modulele și bibliotecile esențiale prin intermediul unui bloc de cod. De exemplu, "express" este importat pentru crearea serverului, în timp ce "dotenv" este utilizat pentru configurarea variabilelor de mediu. În plus, rutele definite în alte fișiere sunt aduse în context.



```
1 dotenv.config()
2 const PORT = process.env.PORT || 3000
3
4 generateSitemap()
5 connectDB()
```

Figura 20. Configurarea variabilelor de mediu și conectarea la baza de date

În continuare, avem un segment de cod responsabil cu configurarea variabilelor de mediu folosind biblioteca "dotenv". Este, de asemenea, esențial în acest punct să stabilim portul pe care serverul va funcționa și să inițiem procesul de conectare la baza de date, apelând funcția "connectDB()".



```
1 const app = express()
2
3 app.use(express.json())
4 app.use(express.urlencoded({ extended: false })))
```

Figura 21. Inițializarea aplicației Express și configurarea middleware-ului

Pe măsură ce avansăm în fișier, ne întâlnim cu un fragment care se ocupă de inițializarea aplicației Express și configurarea middleware-ului. Această secțiune de cod este crucială pentru configurarea aplicației astfel încât să poată analiza cererile JSON și URL-encoded în mod eficient.



```
1 app.use('/api/users', userRoutes)
2 app.use('/api/recipes', recipeRoutes)
3 app.use('/api/mealplans', mealPlanRoutes)
```

Figura 22. Definirea rutelor API

Următoarea secțiune definește rutele API, care sunt esențiale pentru specificarea punctelor de acces la diferite resurse, cum ar fi utilizatorii, rețetele și planurile de mese.


```

1 app.use((req: Request, res: Response, next: NextFunction) => {
2   res.setHeader('Access-Control-Allow-Origin', '*')
3   res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE')
4   res.setHeader('Access-Control-Allow-Headers', 'Content-Type')
5   next()
6 })

```

Figura 23. Configurarea permisiunilor CORS

De asemenea, este necesară o secțiune dedicată configurării permisiunilor CORS. Prin intermediul acestui middleware, sunt setate anumite headere care permit cererilor provenite din diferite origini să acceseze API-ul, asigurându-se astfel o comunicare adecvată între frontend și backend.

```

1 if (process.env.NODE_ENV === 'production') {
2   app.use(express.static(path.join(__dirname, '../frontend/build')))
3
4   app.get('*', (_, res) => {
5     res.sendFile(path.join(__dirname, '../frontend/build/index.html'))
6   })
7 } else {
8   app.get('/', (_, res) => {
9     res.status(200).json({ message: 'Platforma Rezervari Initializata' })
10  })
11 }

```

Figura 24. Servirea fișierelor statice și rute pentru producție sau dezvoltare

```

1 import { Request } from 'express'
2 import mongoose, { Document } from 'mongoose'
3 import { IUser } from '../models/userModel'
4
5 export type UserDocument = IUser & Document
6
7 declare global {
8   namespace Express {
9     interface Request {
10       user?: UserDocument
11       isAdmin?: boolean
12     }
13   }
14 }
15
16 export type Recipe = {
17   recipeId: mongoose.Schema.Types.ObjectId
18 }
19
20 export type Meals = {
21   breakfast: Recipe[]
22   lunch: Recipe[]
23   dinner: Recipe[]
24   snack1: Recipe[]
25   snack2: Recipe[]
26 }
27

```

Figura 25. Types.ts

În cadrul fișierului types.ts, un aspect semnificativ este utilizarea documentului din biblioteca Mongoose pentru a reprezenta un document de utilizator în baza de date. De asemenea, se remarcă o secțiune denumită "declare global", care extinde interfața globală Request din Express prin adăugarea proprietăților "user" și "isAdmin". Această extindere este esențială, deoarece facilitează accesul la obiectul "user" și valoarea "isAdmin" în toate cererile care sunt transmise de la client la server.

De asemenea, se observă utilizarea unui câmp denumit "user", de tip "UserDocument", care reprezintă obiectul utilizator asociat cu cererea după autentificare. Prezența unui semn de întrebare sugerează că acest câmp este opțional, indicând faptul că nu toate cererile vor conține acest câmp – un exemplu fiind cererile care nu implică autentificare.

În mod similar, câmpul "isAdmin", de tip boolean, joacă un rol important în indicarea faptului că utilizatorul curent are privilegii de administrator. Similar cu câmpul "user", acest câmp este de asemenea opțional.

Mai departe, fișierul definește un tip denumit "Recipe", care reprezintă o rețetă în aplicație. Este esențial de remarcat că acest tip este compus exclusiv dintr-un ID de rețetă, care este reprezentat printr-un ObjectId furnizat de biblioteca Mongoose.

În încheiere, tipul "Meals" este definitiv pentru un plan de mese. Acesta conține un array de rețete pentru fiecare masă a zilei, incluzând micul dejun, prânz, cină, și două gustări. Fiecare masă este reprezentată printr-un array de "Recipe", facilitând astfel asocierea a mai multor rețete cu o singură masă. Această structură contribuie la flexibilitatea și funcționalitatea aplicației în gestionarea și organizarea planurilor de mese.



```
1 import mongoose from 'mongoose'
2
3 mongoose.set('strictQuery', true)
4
5 const connectDB = async (): Promise<void> => {
6   try {
7     const conn = await mongoose.connect(process.env.MONGO_URI as string)
8     console.log(`MongoDB Connected: ${conn.connection.host}`.cyan.underline)
9   } catch (error) {
10    console.log(`Error: ${(error as Error).message}`.red.underline.bold)
11    process.exit(1)
12   }
13 }
14
15 export default connectDB
16
```

Figura 26. db.ts

Fișierul db.ts deține responsabilitatea esențială de a stabili conexiunea cu baza de date MongoDB, fiind instrumental în acest proces prin intermediul bibliotecii Mongoose.

La începutul fișierului, biblioteca Mongoose este importată, evidențiind faptul că este o componentă centrală în interacțiunea cu baza de date MongoDB în cadrul aplicațiilor Node.js. Este demn de remarcat faptul că fișierul configurează Mongoose prin setarea opțiunii 'strictQuery' pentru a asigura generarea de erori atunci când sunt utilizate câmpuri nedefinite în interogări, un aspect care contribuie la integritatea și siguranța datelor.

Pe lângă aceasta, fișierul definește o funcție asincronă denumită connectDB, care se angajează în stabilirea conexiunii cu baza de date MongoDB. Acest lucru este realizat folosind adresa de conexiune recuperată din variabilele de mediu. În situația în care conexiunea este stabilită cu succes, un mesaj informativ este afișat în consolă. În contrast, în cazul în care apar erori, acestea sunt înregistrate în consolă și procesul este încheiat cu un cod de eroare. Funcția connectDB este apoi exportată, oferind posibilitatea de a fi invocată din alte fișiere pentru a inițializa conexiunea la baza de date.

```
1 import mongoose from 'mongoose'
2
3 export interface IUser extends mongoose.Document {
4   name: string
5   email: string
6   password: string
7   isAdmin: boolean
8   isValidated: boolean
9 }
10
11 const userSchema = new mongoose.Schema<IUser>({
12   {
13     name: {
14       type: String,
15       required: [true, 'Va rugam adaugati un nume'],
16     },
17     email: {
18       type: String,
19       required: [true, 'Va rugam adaugati un email'],
20       unique: true,
21     },
22     password: {
23       type: String,
24       required: [true, 'Va rugam adaugati o parola'],
25     },
26     isAdmin: {
27       type: Boolean,
28       required: true,
29       default: false,
30     },
31     isValidated: {
32       type: Boolean,
33       required: true,
34       default: false,
35     },
36   },
37   {
38     timestamps: true,
39   }
40 })
41
42 const User = mongoose.model<IUser>('User', userSchema)
43
44 export default User
45
```

Figura 27. userModel.ts

Fișierul următor în analiză se concentrează pe definirea unui model pentru utilizatori, și face uz de biblioteca Mongoose, reiterându-i rolul în comunicarea cu baza de date MongoDB în contextul aplicațiilor Node.js. Modelul este crucial pentru a impune structura documentelor de utilizatori în colecția MongoDB, asigurând astfel o organizare coerentă și uniformă a datelor.

Fișierul în discuție se concentrează pe importarea bibliotecii Mongoose pentru a permite definirea atât a schemei cât și a modelului de utilizator. Într-un pas suplimentar, se definește interfața TypeScript IUser, care extinde interfața Document oferită de Mongoose. Aceasta adaugă proprietăți suplimentare la obiectele de tip IUser, extinzând proprietățile implicite ale unui document Mongoose. Aceste proprietăți suplimentare include: nume, email, parolă, statutul de administrator și validarea contului de utilizator.

Urmează crearea schemei Mongoose, numită userSchema, care dictează structura documentelor din colecția de utilizatori. Pentru fiecare câmp, sunt specificate criterii cum ar fi tipul de date, obligativitatea, unicitatea și valoarea implicită. În plus, opțiunea de a adăuga timbre de timp este setată ca fiind activă, oferind astfel fiecărui document câmpurile createdAt și updatedAt.

Una din funcționalitățile principale al acestui fișier este crearea modelului User, asociat cu schema userSchema. Acest model permite crearea și interacțiunea cu documentele din colecția de utilizatori. În cele din urmă, modelul User este exportat ca export implicit, astfel încât să poată fi importat și utilizat în alte fișiere pentru a interacționa cu colecția de utilizatori din baza de date.

```

1 import mongoose from 'mongoose'
2
3 interface IRecipe {
4   title: string
5   description: string
6   ingredients: [
7     {
8       name: string
9       quantity: number
10      unit: string
11    }
12  ]
13   instructions: string[]
14   imageUrl: string
15   totalCalories: number
16   macros: {
17     protein: number
18     fat: number
19     carbohydrates: number
20   }
21 }
22
23 const recipeSchema = new mongoose.Schema<IRecipe>({
24   title: {
25     type: String,
26     required: true,
27   },
28   description: {
29     type: String,
30     required: true,
31   },
32   ingredients: [
33     {
34       name: { type: String, required: true },
35       quantity: { type: Number, required: true },
36       unit: { type: String, required: true },
37     },
38   ],
39   instructions: {
40     type: [String],
41     required: true,
42   },
43   imageUrl: {
44     type: String,
45   },
46   totalCalories: {
47     type: Number,
48     required: true,
49   },
50   macros: {
51     protein: { type: Number, required: true },
52     fat: { type: Number, required: true },
53     carbohydrates: { type: Number, required: true },
54   },
55 })
56
57 const Recipe = mongoose.model<IRecipe>('Recipe', recipeSchema)
58
59 export default Recipe
60

```

Figura 28. recipeModel.ts

Fișierul recipeModel.ts urmează un traseu similar, definind un model pentru rețete cu ajutorul bibliotecii Mongoose. Procesul este analog cu cel al modelului de utilizator, cu scopul de a stabili structura documentelor pentru rețete în colecția MongoDB.

Importăm biblioteca Mongoose pentru a defini schema și modelul.

Definim interfața TypeScript IRecipe, care specifică structura unei rețete. Aceasta include proprietățile: title (șir de caractere pentru titlul rețetei), description (șir de caractere pentru descrierea rețetei), ingredients (o listă de obiecte care conțin numele ingredientului, cantitatea și unitatea de măsură), instructions (o listă de șiruri de caractere pentru instrucțiunile de preparare a rețetei), imageUrl (șir de caractere pentru URL-ul imaginii rețetei), totalCalories

(număr pentru totalul de calorii al rețetei) și macros (un obiect care conține cantitățile de proteine, grăsimi și carbohidrați în rețetă).

Cream o schemă Mongoose recipeSchema care definește structura documentelor în colecția de rețete. Utilizăm aceleași opțiuni ca și schema de utilizator pentru a valida și structura datele (tip, necesitate etc.).

Cream modelul Recipe asociat cu schema recipeSchema pentru a crea și interacționa cu documentele în colecția de rețete.

Exportăm modelul Recipe ca export implicit, permițându-i să fie importat și utilizat în alte fișiere pentru a interacționa cu colecția de rețete în baza de date.

```
1 import mongoose from 'mongoose'
2
3 interface IMealPlan {
4   userId: mongoose.Schema.Types.ObjectId
5   dayOfWeek: number
6   meals: {
7     breakfast: [{ recipeId: mongoose.Schema.Types.ObjectId }]
8     lunch: [{ recipeId: mongoose.Schema.Types.ObjectId }]
9     dinner: [{ recipeId: mongoose.Schema.Types.ObjectId }]
10    snack1: [{ recipeId: mongoose.Schema.Types.ObjectId }]
11    snack2: [{ recipeId: mongoose.Schema.Types.ObjectId }]
12  }
13 }
14
15 const mealPlanSchema = new mongoose.Schema<IMealPlan>({
16   {
17     userId: {
18       type: mongoose.Schema.Types.ObjectId,
19       ref: 'User',
20       required: true,
21     },
22     dayOfWeek: {
23       type: Number,
24       required: true,
25     },
26     meals: {
27       breakfast: [
28         {
29           recipeId: {
30             type: mongoose.Schema.Types.ObjectId,
31             ref: 'Recipe',
32           },
33         },
34       ],
35       lunch: [
36         {
37           recipeId: {
38             type: mongoose.Schema.Types.ObjectId,
39             ref: 'Recipe',
40           },
41         },
42       ],
43       dinner: [
44         {
45           recipeId: {
46             type: mongoose.Schema.Types.ObjectId,
47             ref: 'Recipe',
48           },
49         },
50       ],
51       snack1: [
52         {
53           recipeId: {
54             type: mongoose.Schema.Types.ObjectId,
55             ref: 'Recipe',
56           },
57         },
58       ],
59       snack2: [
60         {
61           recipeId: {
62             type: mongoose.Schema.Types.ObjectId,
63             ref: 'Recipe',
64           },
65         },
66       ],
67     ],
68   },
69   {
70     timestamps: true,
71   }
72 })
73
74 const MealPlan = mongoose.model<IMealPlan>('MealPlan', mealPlanSchema)
75
76 export default MealPlan
77
```

Figura 29. mealPlanModel.ts

În fișierul numit `mealPlanModel.ts`, se începe prin importarea bibliotecii `Mongoose`, necesară pentru definirea schemei și modelului unei rețete. Se definește interfața `TypeScript IRecipe`, care stipulează structura unei rețete și include proprietăți variate, precum titlu, descriere, ingrediente, instrucțiuni de preparare, URL-ul imaginii, totalul de calorii și macro-nutrienți.

Apoi, este creată o schemă `Mongoose` denumită `recipeSchema`, care stabilește structura documentelor din colecția de rețete. Aceasta folosește opțiuni similare cu cele utilizate în schema de utilizator pentru a valida și a structura datele, precum tipul de date, necesitatea și altele.

Se continuă cu crearea modelului `Recipe`, asociat cu schema `recipeSchema`. Acest model va permite crearea și manipularea documentelor în colecția de rețete.

În final, modelul `Recipe` este exportat ca export implicit, astfel încât să poată fi importat și folosit în alte părți ale aplicației, facilitând interacțiunea cu colecția de rețete din baza de date.

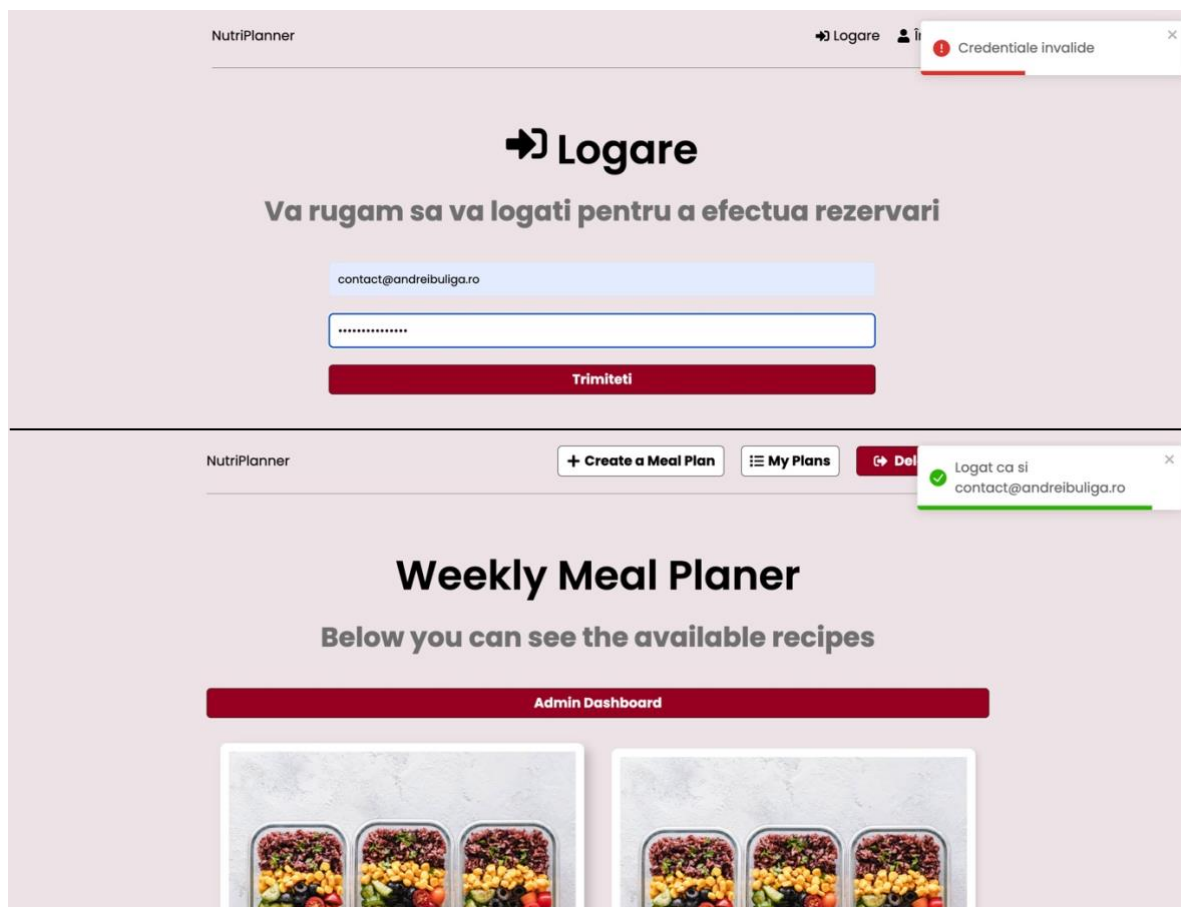


Figura 30. `UserController.ts` – exemplificare autentificare reușită și nereușită

Funcția denumită `registerUser` se încadrează în procesul de înregistrare al unui utilizator nou, prin extragerea datelor necesare din corpul cererii și validarea acestora. Ulterior, se utilizează algoritmul `bcrypt` pentru criptarea parolei, după care se inițiază un nou document de utilizator în baza de date. Ca răspuns, se trimite codul de stare, datele utilizatorului și un token JWT.

Funcția intitulată `loginUser` se ocupă de autentificarea unui utilizator preexistent, prin identificarea acestuia în baza de date folosind adresa de email și ulterior prin verificarea corectitudinii parolei criptate. Răspunsul include codul de stare, datele utilizatorului și un token JWT.

Funcția `getUsers` accesează baza de date pentru a extrage o listă cu utilizatorii înregistrați, pe care o transmite înapoi sub formă de răspuns JSON.

Funcția `updateUserValidation` se angajează în modificarea stării de validare a unui utilizator. Aceasta își bazează funcționarea pe ID-ul utilizatorului, preluat din parametrii cererii și pe starea de validare, preluată din corpul cererii. Se efectuează o căutare în baza de date și se actualizează starea de validare. Răspunsul include datele actualizate ale utilizatorului.

Funcția `generateToken` este specializată în producerea unui token JWT pentru un ID de utilizator specific, facilitând procesul de autentificare pentru cererile ulterioare.

Aceste funcții sunt exportate, astfel încât să fie disponibile pentru integrare în diverse segmente ale aplicației.


```

1 import { Request, Response } from 'express'
2 import asyncHandler from 'express-async-handler'
3 import Recipe from '../models/recipeModel'
4
5 export const getRecipes = asyncHandler(async (req: Request, res: Response) => {
6   const recipes = await Recipe.find({})
7   res.json(recipes)
8 })
9
10 export const createRecipe = asyncHandler(
11   async (req: Request, res: Response) => {
12     const {
13       title,
14       description,
15       ingredients,
16       instructions,
17       imageUrl,
18       totalCalories,
19       macros,
20     } = req.body
21
22     const recipe = new Recipe({
23       title,
24       description,
25       ingredients,
26       instructions,
27       imageUrl,
28       totalCalories,
29       macros,
30     })
31
32     const createdRecipe = await recipe.save()
33     res.status(201).json(createdRecipe)
34   }
35 )
36
37 export const getRecipeById = asyncHandler(
38   async (req: Request, res: Response) => {
39     const recipe = await Recipe.findById(req.params.id)
40
41     if (recipe) {
42       res.json(recipe)
43     } else {
44       res.status(404)
45       throw new Error('Recipe not found')
46     }
47   }
48 )
49
50 export const updateRecipe = asyncHandler(
51   async (req: Request, res: Response) => {
52     const {
53       title,
54       description,
55       ingredients,
56       instructions,
57       imageUrl,
58       totalCalories,
59       macros,
60     } = req.body
61
62     const recipe = await Recipe.findById(req.params.id)
63
64     if (recipe) {
65       recipe.title = title
66       recipe.description = description
67       recipe.ingredients = ingredients
68       recipe.instructions = instructions
69       recipe.imageUrl = imageUrl
70       recipe.totalCalories = totalCalories
71       recipe.macros = macros
72
73       const updatedRecipe = await recipe.save()
74       res.json(updatedRecipe)
75     } else {
76       res.status(404)
77       throw new Error('Recipe not found')
78     }
79   }
80 )
81
82 export const deleteRecipe = asyncHandler(
83   async (req: Request, res: Response) => {
84     const recipe = await Recipe.findById(req.params.id)
85
86     if (recipe) {
87       await recipe.remove()
88       res.json({ message: 'Recipe removed' })
89     } else {
90       res.status(404)
91       throw new Error('Recipe not found')
92     }
93   }
94 )
95

```

Figura 31. recipeController.ts

Fișierul `recipeController.ts` conține o serie de funcții controler care gestionează logica de backend pentru diferite operațiuni legate de rețete, cum ar fi obținerea tuturor rețetelor, crearea unei noi rețete, obținerea unei rețete după ID, actualizarea și ștergerea unei rețete.

Funcția intitulată `getRecipes` se încadrează în domeniul de interogare a bazei de date pentru a extrage toate rețetele existente. Aceasta nu necesită parametri de intrare și se angajează în execuția unei interogări asincrone prin utilizarea modelului `Recipe`. Ca răspuns, se furnizează un obiect JSON ce conține lista de rețete.

Funcția denumită `createRecipe` participă activ în procesul de creare a unei noi rețete. Aceasta extrage datele necesare din corpul cererii, incluzând attribute precum titlul, descrierea, ingredientele, instrucțiunile, adresa URL a imaginii, totalul de calorii și macro-nutrienții. Se inițiază ulterior un nou document în baza de date folosind modelul `Recipe`. Ca răspuns, funcția returnează codul de stare HTTP 201, indicând crearea cu succes, împreună cu detaliile rețetei în format JSON.

Funcția `getRecipeById` are ca obiectiv principal extragerea unei rețete specifice, identificate printr-un ID unic. Aceasta preia ID-ul din parametrii cererii și efectuează o interogare asincronă în baza de date. În cazul în care rețeta este identificată cu succes, se returnează rețeta sub formă de obiect JSON. În caz contrar, se răspunde cu un cod de stare HTTP 404, indicând că resursa nu a fost găsită, împreună cu un mesaj de eroare.

Funcția numită `updateRecipe` este implicată în procesul de actualizare a unei rețete existente. Funcția extrage ID-ul rețetei din parametrii cererii și informațiile actualizate din corpul cererii. Prin intermediul unei interogări asincrone, se caută rețeta corespunzătoare în baza de date. Dacă rețeta este identificată, attributele sunt actualizate și documentul este salvat înapoi în baza de date. Ca răspuns, funcția oferă rețeta actualizată în format JSON. Dacă rețeta nu este identificată, răspunsul include un cod de stare HTTP 404 și un mesaj de eroare.

Funcția denumită `deleteRecipe` se angajează în eliminarea unei rețete din baza de date. Aceasta preia ID-ul rețetei din parametrii cererii și inițiază o interogare asincronă pentru a identifica rețeta. Dacă rețeta este găsită, aceasta este eliminată, iar ca răspuns se furnizează un mesaj de confirmare. În cazul în care rețeta nu este identificată, se returnează un cod de stare HTTP 404 și un mesaj de eroare.

Aceste funcții sunt exportate pentru a permite integrarea lor în diferite componente ale aplicației, asigurând astfel manipularea eficientă a rețetelor în cadrul bazei de date.

```

1 import { Request, Response } from 'express'
2 import asyncHandler from 'express-async-handler'
3 import MealPlan from '../models/mealPlanModel'
4 import { Recipe, Meals } from '../types'
5 import { ObjectId } from 'mongodb'
6 import { Schema } from 'mongoose'
7
8 export const getMealPlans = asyncHandler(
9   async (req: Request, res: Response) => {
10     const mealPlans = await MealPlan.find({
11       userId: (req.user as any)._id,
12     }).populate(
13       'meals.breakfast.recipeId meals.lunch.recipeId meals.dinner.recipeId meals.snack1.recipeId meals.snack2.recipeId'
14     )
15     res.json(mealPlans)
16   }
17 )
18
19 export const createMealPlan = asyncHandler(
20   async (req: Request, res: Response) => {
21     const { dayOfWeek, meals } = req.body
22
23     const existingMealPlan = await MealPlan.findOne({
24       userId: (req.user as any)._id,
25       dayOfWeek,
26     })
27
28     if (existingMealPlan) {
29       res.status(400)
30       throw new Error('A meal plan already exists for this day')
31     } else {
32       const mealPlan = new MealPlan({
33         userId: (req.user as any)._id,
34         dayOfWeek,
35         meals,
36       })
37
38       const createdMealPlan = await mealPlan.save()
39       res.status(201).json(createdMealPlan)
40     }
41   }
42 )
43
44 export const getMealPlanById = asyncHandler(
45   async (req: Request, res: Response) => {
46     const mealPlan = await MealPlan.findById(req.params.id).populate(
47       'meals.breakfast.recipeId meals.lunch.recipeId meals.dinner.recipeId meals.snack1.recipeId meals.snack2.recipeId'
48     )
49
50     if (mealPlan) {
51       res.json(mealPlan)
52     } else {
53       res.status(404)
54       throw new Error('Meal plan not found')
55     }
56   }
57 )
58
59 export const deleteMealPlan = asyncHandler(
60   async (req: Request, res: Response) => {
61     const mealPlan = await MealPlan.findById(req.params.id)
62
63     if (mealPlan) {
64       await mealPlan.remove()
65       res.json({ message: 'Meal plan removed' })
66     } else {
67       res.status(404)
68       throw new Error('Meal plan not found')
69     }
70   }
71 )
72
73 export const updateMealPlan = asyncHandler(
74   async (req: Request, res: Response) => {
75     const { dayOfWeek, meals } = req.body
76
77     const mealPlan = await MealPlan.findById(req.params.id)
78
79     if (mealPlan) {
80       mealPlan.dayOfWeek = dayOfWeek
81       mealPlan.meals = meals
82
83       const updatedMealPlan = await mealPlan.save()
84       res.json(updatedMealPlan)
85     } else {
86       res.status(404)
87       throw new Error('Meal plan not found')
88     }
89   }
90 )
91
92 export const removeRecipeFromMealPlan = asyncHandler(
93   async (req: Request, res: Response) => {
94     const mealPlan = await MealPlan.findById(req.params.id)
95
96     if (mealPlan) {
97       const recipeId = req.params.recipeId
98
99       for (let mealType in mealPlan.meals) {
100         mealPlan.meals[mealType as keyof Meals] = (mealPlan.meals as Meals)[
101           mealType as keyof Meals
102         ].filter(
103           (recipe: Recipe) => recipe.recipeId.toString() !== recipeId
104         ) as [{ recipeId: Schema.Types.ObjectId }]
105       }
106
107       const updatedMealPlan = await mealPlan.save()
108       res.json(updatedMealPlan)
109     } else {
110       res.status(404)
111       throw new Error('Meal plan not found')
112     }
113   }
114 )
115

```

Figure 32. mealPlanController.ts

Fișierul numit `mealPlanController.ts` cuprinde o suită de funcții controler esențiale pentru administrarea planurilor de mese. Acestea sunt integrate ca middleware în rutele Express și coordonează operațiuni CRUD referitoare la planurile de mese.

O componentă importantă a acestui fișier, `getMealPlans`, extrage integralitatea planurilor de mese atribuite unui utilizator, recurgând la metoda `find` a modelului `MealPlan`. Folosind metoda `populate`, informațiile legate de rețete sunt accesate, iar output-ul este livrat sub forma unui răspuns în format JSON.

Cu un rol distinct, `createMealPlan` supervizează inițierea unui plan de mese. Se identifică informații referitoare la ziua săptămânii și rețete din corpul cererii și se evaluează prezența unui plan de mese alocat zilei în cauză. Dacă nu există, un nou plan este generat, și se salvează în baza de date. Răspunsul comunicat evidențiază planul de mese nou și codul de stare 201.

În contrast, `getMealPlanById` își concentrează eforturile pe accesarea unui plan de mese singular, utilizând identificatorul asociat, și pe încărcarea detaliilor rețetelor prin metoda `populate`. În cazul în care planul este identificat, el este remis sub forma unui răspuns JSON. Altfel, se comunică codul de stare 404, împreună cu un mesaj de eroare.

Având un scop clar, `deleteMealPlan` conduce operațiunea de ștergere a unui plan de mese. Planul este căutat în baza de date folosind identificatorul său și, dacă este localizat, este suprimat. Un mesaj de confirmare este încorporat în răspunsul JSON trimis. Alternativ, se oferă codul de stare 404 și un mesaj de eroare.

Funcția `updateMealPlan` coordonează procesul de modificare a unui plan de mese. Informații relevante și identificatorul planului sunt extrase, iar în cazul în care planul este localizat, acesta este modificat și salvat. Răspunsul trimis prezintă planul de mese actualizat în format JSON.

O altă funcție cu un rol important, `removeRecipeFromMealPlan`, se angajează în eliminarea unei rețete anume dintr-un plan de mese. Identificatoarele sunt extrase din parametrii cererii, iar dacă planul de mese este identificat în baza de date, rețeta specificată este eliminată. Planul de mese revizuit este salvat și prezentat ca răspuns JSON.

2.2 Frontend

În acest subcapitol, ne vom dedica atenției asupra evaluării riguroase a arhitecturii și capacității de operare a segmentului frontend al aplicației. Aprofundarea modului de organizare și finalității fiecărui director și fișier din sfera frontend-ului este de o importanță capitală, dat fiind că acesta constituie punctul de interacțiune cu utilizatorul și deliniează modalitatea în care

acesta comunică cu aplicația. Vom investiga, în adiție, suitele de biblioteci implementate și modalitatea prin care acestea contribuie la îmbunătățirea experienței utilizatorului. Vom furniza ilustrații și explicații pentru fiecare fișier, facilitând astfel atașarea de capturi de ecran relevante.

Arhitectura directorului frontend cuprinde fișiere cardinale precum App.js, ce funcționează ca portal de acces în aplicație, activând elementele centrale și coordonând rutarea. index.js și index.css sunt, similar, vitale, având rolul de a activa și desfășura aplicația React.

Directorul intitulat app adăpostește setările de stocare Redux, cu fișierul store.js având menirea de a iniția magazinul de stări Redux.

În cadrul directorului numit components, se regăsesc componente React modulare, ce constituie elementele fundamentale ale interfeței cu utilizatorul, inclusiv Header.jsx, RecipeCard.jsx, și MealPlanList.jsx.

Directorul intitulat features este gazda logicilor dedicate caracteristicilor specifice, cuprinzând starea Redux și serviciile pentru o gamă diversificată de funcționalități ale aplicației, cu accent pe autentificare.

Pages constituie un director ce găzduiește componentele care reflectă paginile integrale ale aplicației, inclusiv pagina de start, pagina de autentificare și pagina de înregistrare.

Directorul denumit tests adăpostește fișierele de testare pentru aplicație, acestea fiind cruciale în asigurarea calității și funcționării conforme a codului.

Sub umbrela bibliotecilor, segmentul frontend capitalizează React.js în vederea edificării interfeței cu utilizatorul, având Material-UI ca un pilon cardinal în realizarea designului UI. Redux și Redux Toolkit sunt aplicate în administrarea stării aplicației, în timp ce axios joacă un rol preponderent în facilitarea comunicării cu backend-ul prin intermediul solicitărilor HTTP. Moment este utilizat în manipularea datelor temporale, iar react-toastify contribuie cu notificări atrăgătoare în cadrul aplicației.



```

1  import axios from 'axios'
2
3  const API_URL = '/api/users/'
4
5  const getToken = () => {
6    const user = JSON.parse(localStorage.getItem('user'))
7    if (user && user.token) {
8      return user.token
9    }
10   return null
11 }
12
13 const register = async (userData) => {
14   const response = await axios.post(API_URL, userData)
15
16   if (response.data) {
17     localStorage.setItem('user', JSON.stringify(response.data))
18   }
19   return response.data
20 }
21
22 const login = async (userData) => {
23   const response = await axios.post(API_URL + 'login', userData)
24
25   if (response.data) {
26     localStorage.setItem('user', JSON.stringify(response.data))
27   }
28   return response.data
29 }
30
31 const logout = () => localStorage.removeItem('user')
32
33 const authService = {
34   register,
35   logout,
36   login,
37   getToken,
38 }
39
40 export default authService
41

```

Figura 33. authService.js

În domeniul autentificării utilizatorilor, există o diversitate de componente esențiale care participă activ la procesul de înregistrare, validare și coordonare a sesiunilor utilizatorilor. Un exemplu semnificativ este reprezentat de modulul authService.js, acesta având funcționalitatea de a oferi metode pentru interfațarea cu API-ul dedicat autentificării. Acest modul aduce în prim-plan importul bibliotecii axios cu scopul de a iniția cereri HTTP, definirea

constantei API_URL care reflectă URL-ul de bază pentru punctul de terminare al utilizatorilor, alături de funcții precum getToken, register, login și logout. Acestea dețin responsabilități specifice în cadrul procesului de autentificare și în administrarea sesiunilor utilizatorilor.

```
1 import { createSlice, createAsyncThunk, createAction } from '@reduxjs/toolkit'
2 import authService from '../authService'
3 import { extractErrorMessage } from '../utils'
4
5 const user = JSON.parse(localStorage.getItem('user'))
6
7 const initialState = {
8   user: user ? user : null,
9   isAdmin: user ? user.isAdmin : false,
10  isValidated: user ? user.isValidated : false,
11  isLoading: false,
12 }
13
14 export const register = createAsyncThunk(
15   'auth/register',
16   async (user, thunkAPI) => {
17     try {
18       return await authService.register(user)
19     } catch (error) {
20       return thunkAPI.rejectWithValue(extractErrorMessage(error))
21     }
22   }
23 )
24
25 export const login = createAsyncThunk('auth/login', async (user, thunkAPI) => {
26   try {
27     return await authService.login(user)
28   } catch (error) {
29     return thunkAPI.rejectWithValue(extractErrorMessage(error))
30   }
31 })
32
33 export const logout = createAction('auth/logout', () => {
34   authService.logout()
35   return {}
36 })
37
38 export const authSlice = createSlice({
39   name: 'auth',
40   initialState,
41   reducers: {
42     logout: (state) => {
43       state.user = null
44       state.isAdmin = false
45       state.isValidated = false
46     },
47   },
48   extraReducers: (builder) => {
49     builder
50       .addCase(register.pending, (state) => {
51         state.isLoading = true
52       })
53       .addCase(register.fulfilled, (state, action) => {
54         state.user = action.payload
55         state.isAdmin = action.payload.isAdmin
56         state.isValidated = action.payload.isValidated
57         state.isLoading = false
58       })
59       .addCase(register.rejected, (state) => {
60         state.isLoading = false
61       })
62       .addCase(login.pending, (state) => {
63         state.isLoading = false
64       })
65       .addCase(login.fulfilled, (state, action) => {
66         state.user = action.payload
67         state.isAdmin = action.payload.isAdmin
68         state.isValidated = action.payload.isValidated
69         state.isLoading = false
70       })
71       .addCase(login.rejected, (state) => {
72         state.isLoading = false
73       })
74   },
75 })
76
77 export default authSlice.reducer
78
```

Figura 34. authService.js

O altă componentă crucială este `authSlice.js`, atribuit cu sarcina de a administra starea de autentificare în aplicație. Aceasta recurge la utilizarea `createSlice` și `createAsyncThunk` din `@reduxjs/toolkit` pentru a configura reduceri și acțiuni asincrone. Starea inițială a reducerului se stabilește pe baza datelor prelevate din `localStorage`, iar acțiunile `register` și `login` cooperează cu `authService` pentru înregistrarea sau validarea utilizatorilor și actualizarea stării reducerului. Acțiunea de `logout` invocă metoda `logout` din `authService` și reinitializează starea reducerului. Reducerul gestionează starea autentificării reflectând rezultatele acțiunilor.



Figura 35. Login & Register

De asemenea, avem componente React care se dedică prezentării vizuale a procesului de autentificare. Un exemplu în acest sens este `Login.jsx`, care propune o interfață pentru validare, păstrând o stare locală pentru administrarea datelor formularului. Prin utilizarea lui `useDispatch` și `useSelector`, aceasta inițiază acțiuni și accesează starea autentificării. La expedierea formularului, se apelează acțiunea de `login`, iar în caz de succes, se redirecționează utilizatorul către pagina principală. În mod similar cu `Login.jsx`, `Register.jsx` este responsabil pentru înregistrarea noilor utilizatori și include verificări pentru confirmarea coincidenței parolelor și un acord GDPR.


```

1  import {
2    FaSignInAlt,
3    FaSignOutAlt,
4    FaUser,
5    FaPlus,
6    FaList,
7  } from 'react-icons/fa'
8  import { Link, useNavigate } from 'react-router-dom'
9  import { useSelector, useDispatch } from 'react-redux'
10 import { logout } from '../features/auth/authSlice'
11 import '../styles/Header.css'
12
13 function Header() {
14   const navigate = useNavigate()
15   const dispatch = useDispatch()
16   const { user } = useSelector((state) => state.auth)
17
18   const onLogout = () => {
19     dispatch(logout())
20     navigate('/')
21   }
22
23   return (
24     <header className='header'>
25       <div className='logo'>
26         <Link to='/'>NutriPlanner</Link>
27       </div>
28       <ul>
29         {user ? (
30           <div>
31             {user.isValidated && (
32               <li>
33                 <Link
34                   className='btn-header btn-reverse'
35                   to='/create-meal-plan'
36                 >
37                   <FaPlus /> Create a Meal Plan
38                 </Link>
39               </li>
40               <li>
41                 <Link
42                   className='btn-header btn-reverse'
43                   to='/saved-meal-plans'
44                 >
45                   <FaList /> My Plans
46                 </Link>
47               </li>
48             )}
49           </div>
50         ) : (
51           <li style={{ display: 'flex', alignItems: 'center' }}>
52             <button
53               className='btn'
54               onClick={onLogout}
55               style={{ marginRight: '1rem' }}
56             >
57               <FaSignOutAlt /> Delogare
58             </button>
59           </li>
60         ) : (
61           <li>
62             <Link to='/login'>
63               <FaSignInAlt /> Logare
64             </Link>
65           </li>
66           <li>
67             <Link to='/register'>
68               <FaUser /> Inregistrare
69             </Link>
70           </li>
71         )}
72       </ul>
73     </header>
74   )
75 }
76
77 export default Header
78
79
80
81

```

Figura 36. Header.jsx

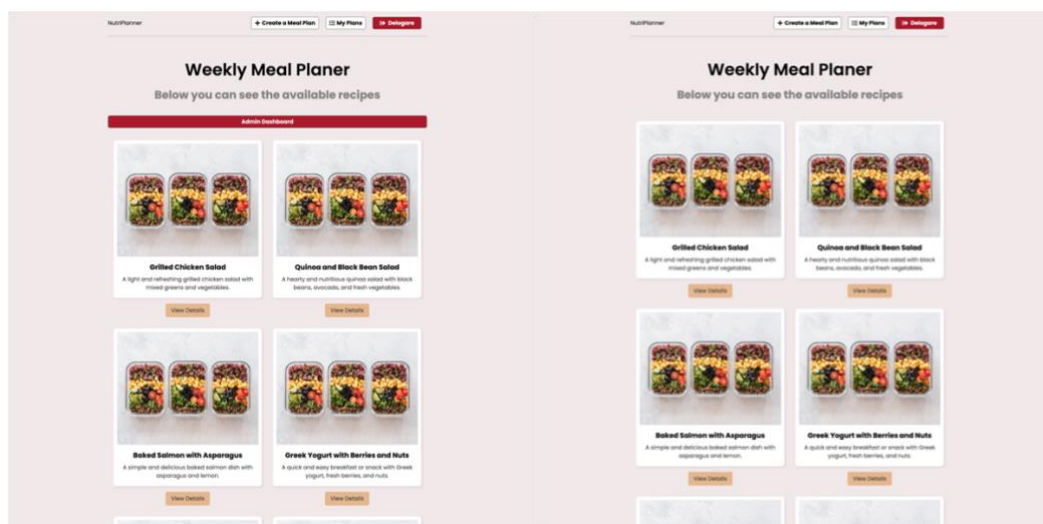
Rolul componentei Header este de a proiecta bara de navigație în partea superioară a aplicației. Aceasta importă pictogramele necesare din react-icons/fa, se folosește de Link și useNavigate din react-router-dom pentru navigație, useSelector și useDispatch din react-redux pentru administrarea stării, importă acțiunea de logout din slice-ul de autentificare și stilurile aferente.

La nivelul componentei, useNavigate este utilizat pentru a dirija programatic după deconectare, iar useDispatch și useSelector sunt folosite pentru a accesa metodele de actualizare a stării și pentru a extrage informații despre utilizator. Funcția onLogout se ocupă cu deconectarea utilizatorului. Aceasta expediază o acțiune de logout la store-ul reducerilor și apoi redirecționează utilizatorul către pagina principală.

În secțiunea de returnare, componenta generează un antet care cuprinde un logo și o listă de link-uri. Link-urile vizibile depind de starea de autentificare a utilizatorului:

- Dacă utilizatorul este autentificat, și dacă contul acestuia este validat, se afișează link-uri pentru crearea unui plan de masă și pentru a vizualiza planurile de masă salvate. Indiferent de validarea contului, este afișat un buton de logout.
- Dacă utilizatorul nu este autentificat, se afișează link-uri pentru conectare și înregistrare.

Pictogramele asociate cu fiecare link sunt afișate alături de text, furnizând un indiciu vizual asupra funcționalității fiecărui link. Stilurile sunt aplicate link-urilor și butoanelor, crescând atractivitatea și accesibilitatea acestora.



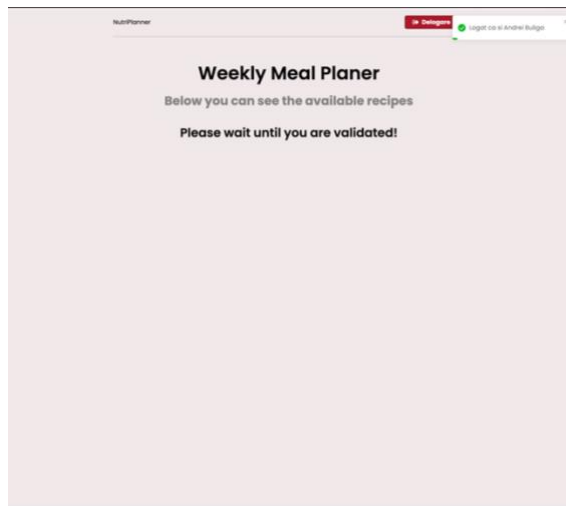


Figura 37. Pagina Home din 3 perspective

Componenta `Home.jsx` constituie pagina principală a aplicației, având rolul de a afișa rețete și alte elemente, în funcție de starea de autentificare și permisiunile utilizatorului. În cadrul acestei componente, sunt importate diferite module, precum `Link` din `react-router-dom` pentru navigare, `useSelector` din `react-redux` pentru accesul la starea globală, `axios` pentru realizarea cererilor HTTP, plus `useEffect` și `useState` din `React` pentru administrarea stării locale și a ciclului de viață al componentei.

Componenta se folosește de `useSelector` pentru a accesa starea de autentificare și informațiile despre utilizatorul actual din starea globală. De asemenea, folosește `useState` pentru a menține o stare locală cu lista de rețete care urmează să fie afișate.

Funcția `useEffect` este utilizată pentru a declanșa funcția `fetchRecipes` când componenta este montată sau când informațiile despre utilizator se modifică. Aceasta încarcă rețetele din API doar în cazul în care utilizatorul este autentificat și validat.

`FetchRecipes` este o funcție asincronă care realizează o cerere GET către `/api/recipes` pentru a extrage rețetele, actualizând apoi starea locală cu datele obținute.

În cadrul secțiunii de returnare a componentei, un titlu și un paragraf introductiv sunt afișate. În funcție de statutul utilizatorului:

- Dacă acesta este autentificat și are rol de administrator, i se afișează un buton pentru a naviga către tabloul de bord al adminului.
- Dacă utilizatorul este autentificat, dar nu este încă validat și nu are rol de administrator, un mesaj îl îndeamnă să aștepte validarea.

- Dacă utilizatorul este validat, se afișează lista de rețete, fiecare rețetă fiind reprezentată printr-o componentă RecipeCard și având un link pentru vizualizarea detaliilor suplimentare.

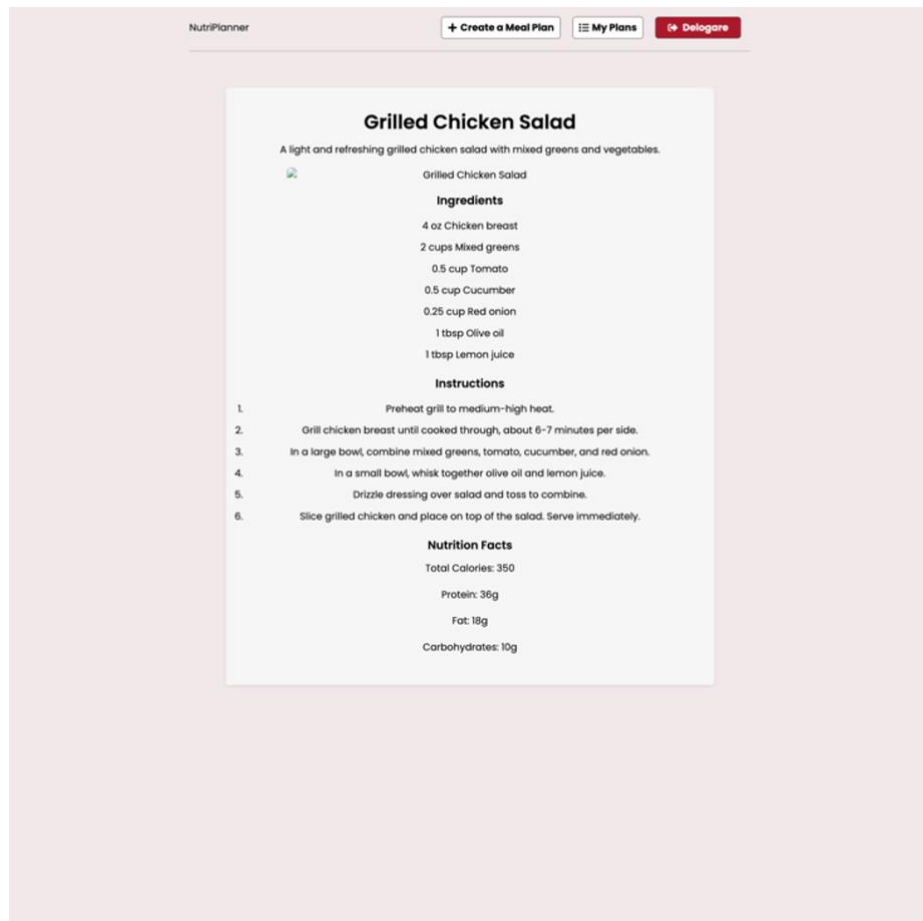


Figura 38. Pagina RecipeDetails

Componenta RecipeDetailsPage importă useParams din react-router-dom pentru a accesa parametrii URL-ului curent, useEffect și useState din React pentru administrarea stării locale și a ciclului de viață al componentei, plus axios pentru realizarea cererilor HTTP.

În cadrul acestei componente, useParams este utilizat pentru a extrage parametrul id din URL, ce reprezintă ID-ul rețetei ce trebuie afișată. În acest context, useState are rolul de a menține rețeta curentă în starea locală.

Funcția useEffect este utilizată pentru a invoca funcția fetchRecipe atunci când componenta este montată sau când id-ul se schimbă. Aceasta efectuează o cerere HTTP către API pentru a prelua detaliile rețetei bazate pe ID și actualizează rețeta în starea locală.

În secțiunea de return, în cazul în care rețeta a fost încărcată, componenta RecipeDetails este afișată, cu rețeta ca proprietate. Dacă rețeta nu a fost încărcată, un mesaj de încărcare este afișat.

Componenta `RecipeDetails` primește o rețetă ca proprietate și o descompune în diverse părți, precum titlu, descriere, imagine, ingrediente, instrucțiuni și așa mai departe.

În cadrul secțiunii de return, componenta afișează detaliile rețetei într-un format structurat, incluzând o imagine, o listă de ingrediente și instrucțiuni de preparare. De asemenea, sunt prezentate informații nutriționale precum numărul total de calorii și macronutrienți (proteine, grăsimi, carbohidrați).

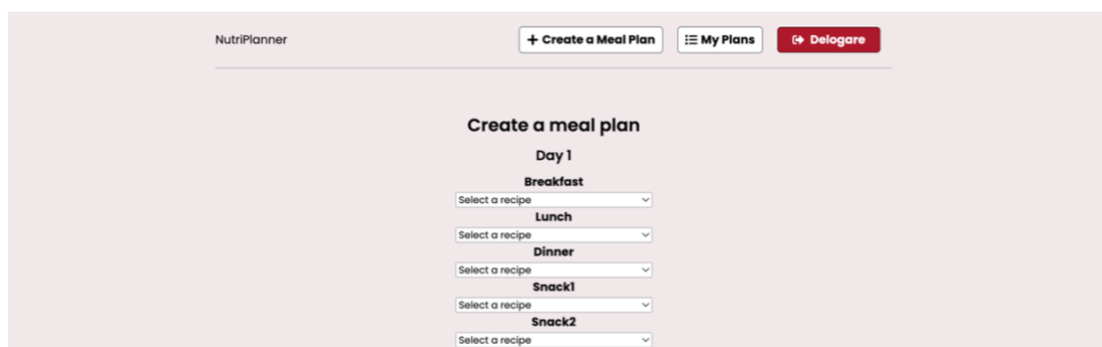


Figura 39. Pagina `CreateMealPlanPage`

Componentele `React CreateMealPlanPage` și `MealPlanForm` lucrează împreună pentru a permite utilizatorilor să creeze un plan de masă.

Componenta `CreateMealPlanPage` importă necesarele hook-uri și componente, inclusiv `useEffect`, `useState`, `axios`, `useSelector`, `MealPlanForm` și `authService`. Aceasta folosește `useSelector` pentru a extrage informații despre utilizator din starea globală și definește starea locală pentru erori și rețete utilizând `useState`.

Componenta utilizează `useEffect` pentru a încărca rețetele disponibile de pe server doar în cazul în care utilizatorul este validat. De asemenea, definește funcția `createMealPlan` care preia planurile de masă și le salvează folosind un serviciu API. Dacă apare o eroare în timpul salvării, aceasta este setată în starea locală.

În secțiunea de return, se afișează un titlu, potențiale erori și formularul pentru planul de masă utilizând componenta `MealPlanForm`.

Componenta `MealPlanForm` primește `onSubmit` și `recipes` ca proprietăți. Aceasta definește `mealTypes` ca un array de tipuri de mese, cum ar fi mic dejun, prânz, cină și altele. În plus, stabilește două stări locale, `selectedRecipes` și `selectedMeals`, pentru a urmări rețetele și mesele selectate.

Componenta definește funcția `handleSelectMeal` pentru a gestiona selecția unei mese, aceasta actualizând stările `selectedMeals` și `selectedRecipes` cu mesele selectate. De asemenea, definește funcția `handleSubmit` pentru a gestiona trimiterea formularului. Aceasta procesează

mesele selectate și le trimite utilizând funcția `onSubmit`, care a fost pasată ca proprietate. În cazul în care planul de masă este creat cu succes, se afișează un mesaj de alertă.

În secțiunea de return, componenta afișează un formular care permite utilizatorului să selecteze rețete pentru diferite tipuri de mese în fiecare zi. După ce selecțiile au fost finalizate, utilizatorul poate salva planul de masă prin apăsarea butonului de trimitere.

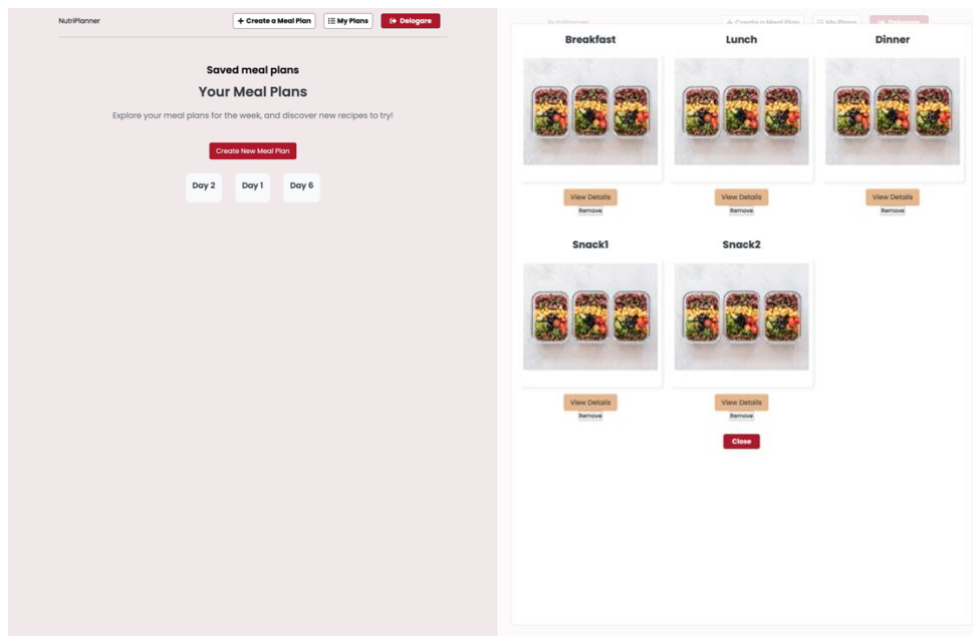


Figura 40. Pagina `SavedMealPlanPage` & Pop-up cu `MealPlanList`

Componentele React `SavedMealPlansPage` și `MealPlanList` colaborează pentru a permite utilizatorilor să vizualizeze planurile de masă salvate.

Componenta `SavedMealPlansPage` importă necesarele hook-uri, `axios` pentru realizarea apelurilor API, componenta `MealPlanList` și `authService`. Aceasta stabilește starea inițială a `mealPlans` ca un array gol și utilizează `useSelector` pentru a obține informații despre utilizator din starea globală.

Se definește funcția `fetchMealPlans`, ce este responsabilă pentru preluarea planurilor de masă de la server. Folosește `useCallback` pentru a menține referința la funcția stabilă. În plus, utilizează `useEffect` pentru a apela funcția `fetchMealPlans` când componenta este montată.

Se definește de asemenea funcția `handleMealPlanUpdate`, care va fi folosită pentru a actualiza starea locală a planurilor de masă.

În secțiunea de return, se afișează un titlu și lista de planuri de masă utilizând componenta `MealPlanList`.

Componenta `MealPlanList` primește `mealPlans` și `onMealPlanUpdate` ca proprietăți. Aceasta stabilește starea locală pentru modal și pentru mesele zilnice curente. Definește funcțiile `openModal` și `closeModal` pentru a controla afișarea modalului.

Se definește funcția `handleRemoveRecipeFromMeal` pentru a șterge o rețetă dintr-un plan de masă și a actualiza planurile de masă.

În secțiunea de return, componenta afișează o listă de planuri de masă. Atunci când un plan este selectat, un modal se deschide, afișând rețetele pentru fiecare tip de masă în ziua respectivă. Utilizatorul poate, de asemenea, șterge o rețetă din planul de masă.

3. Testare

Testarea are un rol esențial în asigurarea calității și eficienței unui sistem informatic. Prin această abordare se identifică și se remediază deficiențele într-un stadiu timpuriu, facilitând optimizarea performanței sistemului. În cadrul acestui proiect, s-au aplicat diverse metode și instrumente pentru a evalua fiabilitatea și funcționalitatea componentelor din backend și frontend.

Backend-ul a fost testat cu o combinație de Mocha și Chai, în conjuncție cu TypeScript. Mocha, un framework de testare popular, permite organizarea suitelor de teste într-un mod clar și concis, facilitând crearea de teste asincrone. Chai, o bibliotecă de aserțiuni, extinde capacitatea de testare a Mocha, oferind o varietate de stiluri de aserțiuni ce permit validarea rezultatelor în diferite moduri. Utilizarea TypeScript adaugă un strat suplimentar de siguranță prin introducerea tipurilor statice, contribuind la prevenirea erorilor la nivel de cod.

Frontend-ul a necesitat un set diferit de abordări și instrumente pentru testare. În acest sens, s-a utilizat Jest ca instrument principal pentru testarea componentelor React și a serviciilor asociate. Jest, un cadru de testare conceput special pentru JavaScript, este renumit pentru rapiditatea sa și pentru capacitatea de a facilita scrierea de teste clare și concise. Acesta este optimizat pentru aplicații construite cu React și vine cu funcționalități care permit o gamă largă de teste, incluzând testarea unitară și de integrare.

Procesul de testare a fost structurat astfel încât să acopere diferite aspecte esențiale, precum funcționalitatea, accesibilitatea, compatibilitatea și performanța. Testarea a avut loc la diferite niveluri, cu accent pe testarea unitară ca punct de plecare, care verifică funcționalitatea unităților individuale de cod, precum metodele și clasele, pentru a se asigura că acestea funcționează corect în izolare.

Pe măsură ce dezvoltarea a progresat, testele s-au extins pentru a include teste de integrare și, în cele din urmă, teste de sistem, pentru a evalua modul în care componentele interacționează între ele și se comportă ca un întreg.

În concluzie, testarea constituie un element esențial în ciclul de dezvoltare al oricărui sistem informatic. Prin utilizarea unor instrumente și metode de testare adecvate, precum Mocha, Chai și Jest, am asigurat că sistemul dezvoltat este fiabil, performant și îndeplinește cerințele stabilite.

4. Evaluare

După implementarea proceselor de testare, s-a efectuat evaluarea și verificarea unui set de parametri care influențează calitatea produsului final, în cazul de față, NutriPlanner. Evaluarea a inclus analiza comportamentului sistemului în timpul interacțiunii cu interfața sa, monitorizarea eficienței comunicației și verificarea compatibilității cu diverse dispozitive.

Testarea funcțională s-a axat pe comportamentul NutriPlanner în timpul interacțiunilor cu interfața utilizatorului. Această metodă de testare, adesea numită "Black Box", se concentrează pe rezultatele obținute în urma anumitor acțiuni sau intrări de date, fără a lua în considerare mecanismele interne ale sistemului. În cadrul NutriPlanner, s-a evaluat eficiența comunicației prin monitorizarea tranzițiilor între diferitele pagini ale aplicației, viteza de transmitere a datelor și gestionarea acestora în diferitele activități ale sistemului.

Pentru a asigura o experiență de utilizare uniformă și naturală pe orice dispozitiv, NutriPlanner a fost testat pe o varietate de dispozitive, simulând diverse medii de funcționare. În timpul acestor teste, s-au identificat și ajustat anumite inconsistente vizuale și de funcționalitate, legate de scalabilitate, aspect vizual și adaptabilitate la diferite formate de ecran.

Testarea performanței pentru NutriPlanner a implicat simularea unor scenarii de utilizare pentru a identifica punctele în care gestionarea datelor sau implementarea algoritmilor nu era optimă. De exemplu, s-au efectuat teste comparative pentru a evalua eficiența stocării și accesării informațiilor nutriționale și a rețetelor. Acest lucru a implicat monitorizarea timpului de încărcare pentru diferite elemente ale aplicației și evaluarea eficienței în prelucrarea și afișarea datelor.

În concluzie, evaluarea a reprezentat o etapă esențială în asigurarea calității și fiabilității NutriPlanner, permițând identificarea și remedierea problemelor de funcționalitate, compatibilitate și performanță.

Concluzii

În cadrul acestei lucrări am abordat dezvoltarea unei soluții tehnologice pentru provocarea accesului constant și eficient la planificarea și gestionarea nutriției, prin intermediul aplicației NutriPlanner. Prin funcționalitățile pe care le-am implementat, am reușit să satisfacem cerințele utilizatorilor, punând accent pe interactivitate, ușurința în utilizare și valorificarea optimă a capacităților oferite de tehnologia actuală.

Pentru a ne asigura că NutriPlanner rămâne relevant și competitiv în fața inovațiilor tehnologice viitoare, avem în plan integrarea cu alte aplicații și platforme (în afară de bazele de date de nutriție și rețete existente), precum servicii de urmărire a activității fizice sau platforme de cumpărături online. De asemenea, explorăm posibilitatea implementării unui sistem de plată securizat, care ar permite adăugarea de funcționalități premium și ar crea oportunități de monetizare pentru dezvoltatori și parteneri.

Pe parcursul realizării acestui sistem informatic, am dobândit competențe valoroase în domeniul managementului de proiect, învățând pașii necesari pentru dezvoltarea unui produs tehnologic. Mai mult, ne-am aprofundat cunoștințele în domeniul IT, explorând un mediu de dezvoltare de aplicații mobile complet nou și interacționând cu tehnologii de ultimă generație precum Firebase. Această experiență ne-a oferit o înțelegere mai profundă a soluțiilor bazate pe cloud și a beneficiilor aduse de stocarea datelor într-o bază de date globală.

În concluzie, NutriPlanner reprezintă un pas important spre facilitarea accesului la o nutriție sănătoasă și echilibrată. Este o unealtă ce împuternicește utilizatorii să își gestioneze eficient dieta, să găsească și să partajeze rețete, și să obțină informații nutriționale de încredere. În același timp, oferă o platformă pentru specialiștii în nutriție pentru a-și împărtăși cunoștințele și a ajuta utilizatorii să facă alegeri alimentare mai informate.

Bibliografie

- Collins, A. (2019, January 7). *20 Highly Effective Ways to Promote a Product*. Shopify.
<https://www.shopify.com/blog/how-to-market-a-product>
- Health, N. (2019, November 22). *Importance of Balanced Diet in a Healthy Lifestyle*.
Narayana Health Care. <https://www.narayanahealth.org/blog/importance-of-balanced-diet-for-a-healthy-lifestyle/>
- kristjan. (2021, April 8). How to Create a Marketing Plan for a New Product (2021). *Adacted*.
<https://adacted.com/new-product-marketing-plan-guide/>
- McCormick, K. (2023, May 31). 12 Ways to Effectively Promote a New Product or Service. *Www.wordstream.com*.
<https://www.wordstream.com/blog/ws/2020/07/29/how-to-promote-a-product>
- McKinsey & Company. (2020, October 5). *COVID-19 digital transformation & technology / McKinsey*. Mckinsey. <https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever>
- Nacach, J. (2018, September 15). What Is a Good Marketing Strategy for New Product Development? *The Meltwater Blog*. <https://www.meltwater.com/en/blog/what-is-a-good-marketing-strategy-for-new-product-development>
- Olabanji, I. (2022, August 22). *Importance Of Balanced Diet In A Healthy Lifestyle*.
Healthsoothe.com. <https://www.healthsoothe.com/importance-of-balanced-diet-in-a-healthy-lifestyle/>
- United Nations. (2020). *The Impact of Digital Technologies / United Nations*. *Www.un.org*;
United Nations. <https://www.un.org/en/un75/impact-digital-technologies>
- World Health Organisation. (2020, April 29). *Healthy diet*. World Health Organisation.
<https://www.who.int/news-room/fact-sheets/detail/healthy-diet>
- World Health Organization. (2023). *Promoting healthy diets*. *Www.who.int*.
<https://www.who.int/westernpacific/activities/promoting-healthy-diets>
- Zarnowiecki, D., Mauch, C. E., Middleton, G., Matwiejczyk, L., Watson, W. L., Dibbs, J., Dessaix, A., & Golley, R. K. (2020). A systematic evaluation of digital nutrition promotion websites and apps for supporting parents to influence children's nutrition. *International Journal of Behavioral Nutrition and Physical Activity*, 17(1).
<https://doi.org/10.1186/s12966-020-0915-1>