

Universitatea
Transilvania
din Brașov

FACULTATEA DE INGINERIE ELECTRICĂ
ȘI ȘTIINȚA CALCULATOARELOR

PROIECT DE DIPLOMĂ

Conducător științific:

Conf. Dr. Ing. Popescu Vlad

Absolvent:

Cursaru David

BRAȘOV, 2024

Departamentul: Electronică și calculatoare
Programul de studii: Calculatoare

Cursaru David

Proiectarea și Implementarea unei Aplicații Web de Monitorizare a Sănătății Fizice și a Nutriției

Conducător științific:
Conf. Dr. Ing. Popescu Vlad

Brașov, 2024

Cuprins

Cuprins

Lista de acronime	4
1 Introducere	5
1.1 Tema proiectului	5
1.2 Scopul proiectului	5
1.3 Obiective principale	6
1.3.1 Obiective software	6
1.3.2 Obiective de documentare	6
1.4 Aplicații similare	7
1.4.1 Google Fit	7
1.4.2 Zepp	7
1.4.3 MyFitnessPal	8
1.5 Structura lucrării	8
2 Tehnologii și metodologie	9
2.1 Cadre de lucru	9
2.1.1 Angular	9
2.1.2 ASP .NET Core Web API	9
2.1.3 Microsoft SQL Server	11
2.2 Limbaje de programare	12
2.2.1 TypeScript	12
2.2.2 C#	12
2.3 Medii de programare	13
2.3.1 Visual Studio Code	13
2.3.2 Visual studio 2022	13
2.4 Tactici și tehnici de dezvoltare	14
2.4.1 Single page application	14
2.4.2 Comunicarea HTTP	15
2.4.3 JSON Web Token	16
2.4.4 Metoda Code First	17
2.4.5 Repository Pattern	18
3 Structura și dezvoltarea aplicației	19
3.1 Arhitectura aplicației – diagrame uml	20

3.2	Arhitectura serverului	25
3.2.1	Nivelul de prezentare	26
3.2.2	Nivelul de domeniu	28
3.2.3	Nivelul de infrastructură sau persistență	29
3.2.4	Nivelul de interfețe sau adaptoare	30
3.3	Arhitectura interfeței cu utilizatorul	31
3.3.1	Module	31
3.3.2	Directive și componente	32
3.3.3	Injectarea dependențelor	33
3.3.4	Servicii	34
3.4	Elemente de Implementare	35
3.4.1	Pagina de autentificare	35
3.4.2	Pagina tabloului de bord	38
3.4.3	Pagina de monitorizare a caloriilor	42
3.4.4	Pagina de monitorizare a exercițiilor și activităților	45
3.4.5	Pagina de monitorizare a somnului	46
3.4.6	Pagina de rapoarte grafice	49
3.5	Testare și validare	51
4	Concluzii	55
4.1	Realizarea obiectivelor propuse	55
4.2	Dezvoltări ulterioare	56
5	Bibliografie	57
	Lista de figuri, tabele și coduri sursă	57
	Rezumat	59
	Abstract	60

LISTA DE ACRONIME

API - Application Programming Interface
DI - Dependency Injection
DOM - Document Object Model
DTO - Data Transfer Object
HTTP - Hypertext Transfer Protocol
JSON - JavaScript Object Notation
JWT - JSON Web Token
MPA - Multi-Page Application
Oauth 2.0 - Open Authorization 2.0
RDBMS - Relational Database Management System
REST - Representational State Transfer
SPA - Single Page Application
SQL - Structured Query Language
UML - Unified Modeling Language
URL - Uniform Resource Identifier
XML - eXtensible Markup Language

1 INTRODUCERE

1.1 Tema proiectului

Tema acestui proiect de licență este proiectarea și implementarea unei aplicații web complete, de monitorizare a nutriției și a unor aspecte generale de fitness și sănătate pentru facilitarea întreținerii sănătății fizice ale utilizatorilor, utilizând cadre de lucru și tehnologii moderne cum ar fi ASP .NET Web Core API, Angular, SQL Server etc.

Această lucrare constă în primul rând în crearea unei arhitecturi scalabile și eficiente pentru aplicația dezvoltată, mai apoi în integrarea și implementarea funcționalităților specifice pentru o aplicație de acest fel. O altă temă fundamentală în această lucrare a fost crearea unei platforme ușor de folosit care să centralizeze date relevante din alte aplicații sau dispozitive care au posibilitatea de a înregistra parametrii de fitness sau sănătate ce necesită o monitorizare activă prin senzori și de a integra noi funcționalități utile folosind aceste date.

1.2 SCOPUL PROIECTULUI

Pentru că trăim într-o lume în care sedentarismul și stilurile de viață haotice și nesănătoase au ajuns o normalitate, menținerea unui stil de viață activ și sănătos a devenit mai dificil de realizat. Lucratul de acasă, munca la birou sau pur și simplu comoditatea și obiceiurile proaste au ajuns să afecteze sănătatea fizică dar și mentală a multor oameni, mai ales a tinerilor. Prin această aplicație am căutat să aduc o soluție simplă și utilă pentru a facilita abordarea unui stil de viață mai sănătos și de a oferi o unealtă prin care utilizatorii să își monitorizeze și gestioneze nutriția și activitatea fizică, cât și să își seteze obiective pe care să le poată îndeplini. Scopul principal a fost de a crea o aplicație web în care să integrez mai multe componente și funcționalități, inspirându-mă din aplicații cunoscute din această zonă de fitness și nutriție. Am urmărit ca aplicația să aibă o interfață atrăgătoare și ușor de vizualizat,

ușor de folosit de la locul de muncă, de acasă sau din orice alt loc, dorind să implementez un design responsive care să poată fi utilizat de pe dispozitive de orice mărime.

1.3 OBIECTIVE PRINCIPALE

1.3.1 Obiective software

- Dezvoltarea unei aplicații web funcționale
- Dezvoltarea unei pagini de autentificare și înregistrare în cadrul acestei aplicații
- Crearea unei interfețe cu utilizatorul simplă, plăcută și ușor de folosit, evitând un design aglomerat și haotic.
- Implementarea unui design responsive pentru utilizarea pe diferite platforme de diferite mărimi.
- Crearea unui tablou de bord care are rolul de pagina principală, unde se vor centraliza toate datele relevante pentru utilizator.
- Crearea unei pagini de monitorizare și de urmărire a caloriilor și a datelor de nutriție.
- Integrarea și folosirea a cel puțin un API extern.
- Crearea unei pagini de monitorizare și urmărire a exercițiilor și activităților fizice
- Calcularea caloriilor arse din activități exerciții
- Calcularea caloriilor și nutrienților din alimente și mâncăruri
- Monitorizarea nivelului de hidratare prin urmărirea cantității de apă consumate
- Dezvoltarea unei componente prin care utilizatorul să își poată seta obiective
- Dezvoltarea unei componente prin care utilizatoru să își facă programul pe ziua respectivă
- Crearea unei pagini de monitorizare a somnului și a parametrilor acestuia.
- Crearea unei pagini de rapoarte grafice pentru vizualizarea datelor pe intervale de zile/săptămâni/luni/ani.
- Crearea unei pagini de profil și de editare a datelor de profil
- Integrarea unui API prin care să pot prelua date relevante de la dispozitive mobile sau smartwatch-uri
- Encapsularea întregii aplicații web într-o aplicație de mobil

1.3.2 Obiective de documentare

- Documentarea asupra importanței nutriției și a activității fizice

- Analiza stadiului actual al aplicațiilor sau sistemelor din această zonă de nutriție și sănătate
- Documentarea asupra cadrelor de lucru și a tehnologiilor folosite pentru dezvoltarea aplicației (.NET, Angular, TypeScript, C#, Postman, SQL Server, Entity framework etc.)
- Documentarea asupra metodelor de comunicare cu API-uri externe, protocoale HTTP, OAuth 2.0, Google Fit API, APINinjas.

1.4 APLICAȚII SIMILARE

Aplicațiile pe care le voi menționa mai jos au servit ca o sursă de inspirație pentru crearea designului, precum și în abordarea nevoilor utilizatorului prin implementarea funcționalităților aplicației pe care am dezvoltat-o.

1.4.1 Google Fit

Google Fit este o aplicație dezvoltată de Google, disponibilă atât pentru dispozitivele mobile, cât și pentru platforma web. Această aplicație are rolul de a colecta și integra datele privind activitatea fizică și starea de sănătate ale utilizatorilor. Prin conectarea cu diverse dispozitive wearable și smartphone-uri, Google Fit oferă o privire holistică asupra nivelului de activitate, permițând utilizatorilor să își stabilească și să își monitorizeze obiectivele de fitness. Cu funcționalități precum monitorizarea pașilor, a distanței și a caloriilor arse, Google Fit se distinge prin adaptabilitatea sa la diferite surse de date și prin interfața sa prietenoasă, facilitând o gestionare eficientă a sănătății și fitnessului.

1.4.2 Zepp

Zepp, anterior cunoscută sub numele de Amazfit, este o aplicație mobilă și web care însoțește dispozitivele wearables ale brandului. Această platformă oferă utilizatorilor o gamă variată de funcționalități pentru monitorizarea sănătății și fitnessului. Zepp integrează date precum ritmul cardiac, analize de stres și date detaliate despre somn, oferind utilizatorilor o imagine cuprinzătoare a stării lor de sănătate. Cu o interfață intuitivă, Zepp permite utilizatorilor să urmărească și să-și optimizeze activitățile fizice, promovând un stil de viață sănătos și activ.

1.4.3 MyFitnessPal

MyFitnessPal este o aplicație mobilă și web pentru monitorizarea dietei și exercițiilor. Această platformă oferă utilizatorilor instrumente avansate pentru determinarea aportului caloric optim și gestionarea nutrienților în funcție de obiectivele individuale. MyFitnessPal oferă și o variantă premium, care permite ajustarea precisă a cantității de macronutrienți necesară zilnic și oferă o perspectivă detaliată asupra nivelului de colesterol. Aplicația prezintă patru tipuri distincte de grafice și profiluri, inclusiv unul dedicat macronutrienților (grăsimi, carbohidrați, proteine și calorii), un profil de sănătate a inimii (grăsimi, sodiu, colesterol și calorii), un graf pentru carbohidrați, zaharuri, fibre și calorii, precum și un rezumat particularizat ce permite utilizatorilor să selecteze trei elemente nutritive relevante pentru monitorizarea personalizată a dietei lor.

1.5 STRUCTURA LUCRĂRII

Introducere - Acest prim capitol prezintă contextul general al proiectului, obiectivele principale de dezvoltare software și de documentare, scopul și motivația lucrării cât și sisteme similare existente folosite ca sursă de inspirație.

Tehnologii și metodologie - Acest capitol conține fundamentarea teoretică și prezentarea conceptelor necesare și folosite mai departe în capitolul 3, unde se prezintă modul în care a fost implementată și proiectată aplicația. Se prezintă informații teoretice despre tehnologii, cadre de lucru, tactici și tehnici de dezvoltare utilizate în dezvoltarea aplicației.

Structura și dezvoltarea aplicației – În acest capitol se prezintă în detaliu arhitectura aplicației împărțită în serverul de backend și interfața cu utilizatorul, diagrame UML, elementele de implementare unde se detaliază cum s-a dezvoltat fiecare pagină și se prezintă funcționalitățile de bază ale aplicației.

Concluzie – În acest capitol se prezintă concluziile și observațiile finale cu privire la acest proiect și la procesul de dezvoltare și îndeplinirea obiectivelor setate la început. Totodată se prezintă și dezvoltările ulterioare care pot fi adăugate sistemului în viitor.

2 TEHNOLOGII ȘI METODOLOGIE

2.1 CADRE DE LUCRU

2.1.1 Angular

Angular este un cadru de dezvoltare cu sursă deschisă, creat de echipa Angular de la Google și lansat pe 14 septembrie 2016. A reprezentat o evoluție semnificativă față de predecesorul său, AngularJS, fiind o rescriere completă și introducând îmbunătățiri esențiale, cum ar fi limbajul TypeScript, sintaxa și directivele, precum și concepte avansate precum încărcarea amânată (lazy loading).

Angular se utilizează în principal pentru dezvoltarea aplicațiilor web single-page, furnizând o interacțiune cu utilizatorul fluidă, fără a necesita reîncărcarea întregii pagini. Este apreciat la nivel global pentru versatilitatea sa și este utilizat într-o gamă variată de proiecte, de la cele academice și educaționale până la aplicații complexe cu impact global.

Angular Material este o bibliotecă în care se află componente și stiluri predefinite, create special pentru a se integra ușor și rapid în aplicații Angular. Inițial dezvoltată de Google în 2014, această bibliotecă beneficiază constant de îmbunătățiri, actualizări și noi funcționalități din partea echipei Angular. Aplicațiile construite cu Angular Material se pot adapta cu ușurință pe diverse platforme, inclusiv Android, iOS, Windows și MacOS, datorită versatilității tehnologiei.

2.1.2 ASP .NET Core Web API

Platforma Web .NET Core face parte din cadrul de lucru ASP.NET Core, și este folosit pentru a dezvolta servicii web API.

API este prescurtarea de la Application Programming Interface, care s-ar traduce Interfață de Programare a Aplicațiilor. O aplicație Web API este o aplicație care oferă servicii și funcționalități accesibile prin intermediul protocoalelor și standardelor web. Într-o aplicație de acest tip, funcționalitățile sunt expuse prin intermediul unor puncte finale de acces numite

endpoint-uri care pot fi apelate de alte aplicații sau servicii. Aplicația primește cereri HTTP, cum ar fi GET, POST, PUT, DELETE, de la clienți și returnează răspunsuri HTTP care conțin datele solicitate.

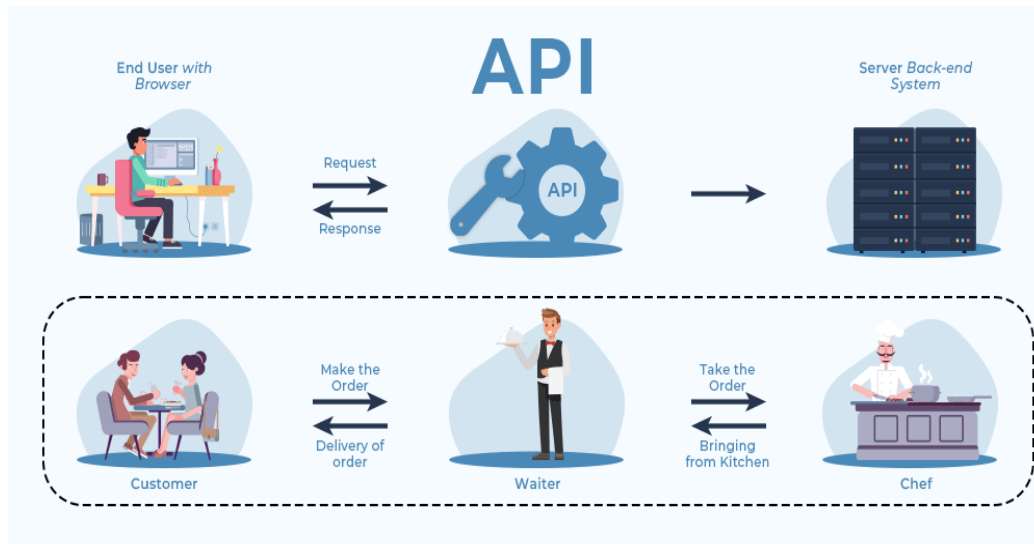


Figura 2.1: Interfața de programare a aplicațiilor [1]

ASP.NET Core Web API este un cadru de lucru puternic și flexibil dezvoltat de Microsoft, care permite crearea de servicii web RESTful scalabile și interoperabile. Această tehnologie face parte din ecosistemul ASP.NET și oferă dezvoltatorilor un set robust de instrumente și biblioteci pentru construirea și implementarea aplicațiilor web bazate pe arhitectura API.

Acest cadru de lucru se bazează pe fundamentul ASP.NET și permite dezvoltatorilor să creeze servicii web care pot fi consumate de diverse clienți, inclusiv:

- aplicații web,
- aplicații mobile,
- aplicații de tip SPA (Single Page Application),
- dispozitive IoT (Internet of Things) și multe altele.

2.1.3 Microsoft SQL Server

Microsoft SQL Server reprezintă un sistem de administrare a bazelor de date relaționale (RDBMS) cu o utilizare extensivă în industrie, datorită versatilității și documentației sale detaliate și cuprinzătoare. Este recunoscut ca unul dintre cele mai valoroase sisteme de gestionare a bazelor de date existente.

SQL Server susține securitatea și conformitatea în mediul enterprise prin funcționalități de securitate precum Transparent Data Encryption, Auditing, Row-Level Security, Dynamic Data Masking și Always Encrypted.” [2]

Transparent Data Encryption este o caracteristică puternică a SQL Server care criptează automat datele în timpul stocării pe disc, asigurând protecția acestora chiar și în situația în care dispozitivul de stocare este compromis. Aceasta asigură confidențialitatea datelor și respectarea cerințelor de conformitate cu privire la securitatea informațiilor.

Auditing este o funcționalitate esențială pentru monitorizarea și urmărirea activităților desfășurate în baza de date. SQL Server permite configurarea auditării la nivel de bază de date și de tabele, oferind informații detaliate despre acțiunile efectuate asupra datelor.

Row-Level Security permite definirea politicilor de securitate la nivel de rând, permițând accesul și vizualizarea datelor doar pentru utilizatorii autorizați. Aceasta oferă un control fin asupra accesului la informații și protejează datele sensibile împotriva accesului neautorizat.

Dynamic Data Masking permite ascunderea sau mascarea datelor sensibile în timp real, în funcție de drepturile de acces ale utilizatorului. Aceasta previne expunerea accidentală a informațiilor confidențiale în aplicații sau rapoarte și asigură protecția datelor sensibile.

Always Encrypted este o funcționalitate avansată care permite criptarea datelor sensibile, inclusiv cheile de criptare, astfel încât acestea să fie păstrate în mod sigur chiar și atunci când sunt în tranzit sau stocate în medii nesigure. Aceasta oferă un nivel înalt de confidențialitate și protecție a datelor.

Cartea „Microsoft SQL Server 2019: A Beginner's Guide” de Dusan Petkovic [3], publicată în 2020, servește drept ghid pentru începători în lumea SQL Server. Această lucrare cuprinde conceptele fundamentale ale SQL Server, interogările SQL, proiectarea bazelor de

date și sarcinile de administrare. Prin intermediul acestui ghid, pot învăța să utilizez SQL Server în mod eficient și să mă familiarizez cu principiile și practicile esențiale ale gestionării bazelor de date.

„SQL Server este un sistem puternic și bogat în funcționalități pentru gestionarea bazelor de date relaționale, dezvoltat de Microsoft. Oferă o platformă robustă pentru stocarea, gestionarea și manipularea datelor, fiind o opțiune populară pentru organizații de toate dimensiunile. Cu funcționalitățile sale avansate de securitate, opțiunile de scalabilitate și suportul pentru disponibilitate înaltă și recuperare în caz de dezastre, SQL Server oferă o soluție cuprinzătoare pentru gestionarea datelor. Această carte servește ca ghid pentru începători în utilizarea SQL Server, acoperind conceptele fundamentale, interogările SQL, proiectarea bazelor de date și sarcinile de administrare.” [3]

2.2 LIMBAJE DE PROGRAMARE

2.2.1 TypeScript

TypeScript este un limbaj de programare deschis pentru folosire dezvoltat de compania Microsoft, bazat pe JavaScript. Acesta a fost creat pentru a extinde JavaScript prin adăugarea de tipuri statice și funcționalități specifice dezvoltării de aplicații de mari dimensiuni și complexitate.

Unul dintre principalele avantaje ale TypeScript este posibilitatea de a adăuga tipuri de date la variabile, funcții și obiecte. Aceasta aduce beneficii semnificative în dezvoltarea software, deoarece oferă un sistem puternic de verificare a tipurilor în timpul compilării și elimină o serie de erori comune de tipuri care pot apărea în JavaScript.

Cu ajutorul TypeScript, pot scrie cod mai sigur, mai ușor de întreținut și mai scalabil, beneficiind totodată de avantajele ecosistemului JavaScript.

2.2.2 C#

C# este un limbaj de programare de nivel înalt, orientat pe obiecte, creat în anul 2000 de Microsoft. A fost proiectat cu scopul de a oferi putere, versatilitate și claritate, adaptându-se pentru dezvoltarea diverselor aplicații, inclusiv cele enterprise, web și mobile.

C# se încadrează în categoria limbajelor de programare C și are o puternică legătură cu platforma de dezvoltare .NET. Acesta dispune de un sistem de tipuri solid, administrarea automată a memoriei și o sintaxă clară și expresivă, ceea ce îl plasează alegerea în preferințele dezvoltatorilor pentru crearea de aplicații fiabile și scalabile.

Una dintre caracteristicile-cheie ale limbajului C# este orientarea sa pe obiecte. Acesta îmi permite să modelez structuri de date și să creez obiecte care să interacționeze între ele prin intermediul conceptelor de clasă, moștenire, încapsulare și polimorfism. Aceasta facilitează dezvoltarea de cod modular și extensibil, care poate fi mai ușor de întreținut pe parcursul timpului.

2.3 MEDII DE PROGRAMARE

2.3.1 Visual Studio Code

Editorul de cod sursă cunoscut sub numele de Visual Studio Code sau VS Code este un produs dezvoltat de Microsoft și a câștigat rapid popularitate în rândul comunității dezvoltatorilor. Am ales să utilizez Visual Studio Code pentru dezvoltarea aplicației mele datorită numeroaselor beneficii pe care le oferă. Unul dintre avantajele majore ale utilizării Visual Studio Code este experiența de lucru eficientă și rapidă. Cu ajutorul completării automate a codului și a evidențierii sintaxei, am putut scrie codul mai repede și mai precis. Aceasta mi-a permis să creez aplicația într-un timp mai scurt, fără a face compromisuri în ceea ce privește calitatea.

2.3.2 Visual studio 2022

Visual Studio 2022 este cea mai nouă versiune a mediului de dezvoltare integrat (IDE) oferit de Microsoft, având un set bogat de funcționalități și îmbunătățiri care îi susțin utilizatorii în dezvoltarea software. Această versiune aduce o serie de avantaje și facilități pentru programatori, în special pentru proiectele bazate pe platforma ASP.NET Core Web API.

Unul dintre aspectele cheie ale Visual Studio 2022 este suportul avansat pentru cadrul de lucru Entity Framework (EF). EF este o tehnologie dezvoltată de Microsoft care facilitează lucrul cu bazele de date relaționale pentru a face interacțiunea cu acestea cât mai simplă și eficientă. Cu ajutorul Visual Studio 2022, dezvoltatorii au posibilitatea de a folosi instrumentele

EF pentru a defini modelele de date, a efectua operații de interogare și manipulare a datelor și a crea migrări ale structurii bazei de date. Aceasta aduce un nivel ridicat de productivitate și ușurință în dezvoltarea și gestionarea bazei de date.

2.4 TACTICI ȘI TEHNICI DE DEZVOLTARE

2.4.1 Single page application

Single Page Application (SPA) este o abordare modernă în dezvoltarea aplicațiilor web, care se diferențiază de modelul tradițional de navigare între pagini multiple. Într-o SPA, întreaga aplicație este încărcată o singură dată în browser, iar toate interacțiunile ulterioare cu serverul sunt gestionate prin intermediul comunicării asincrone cu servicii web.

Mai jos putem vedea o imagine sugestivă pentru comparația dintre SPA și MPA. În această figura putem observa diferențele la nivel de performanță, mentenanță, securitate, cost, scalabilitate etc.

	SPAs	MPAs
PERFORMANCE	Faster loading time	Slower loading time
DEBUGGING	More difficult	Well supported by debugging tools
DEVELOPMENT	Fast	Slower, more complex
MAINTENANCE	Fast & easy	Slower
SECURITY	simplified	More challenging
SEO	Limited	Easier & more effective
COST	More expensive	Less expensive
SCALABILITY	Not scalable	Scalable

Figura 2.2: Single Page Application vs Multiple Page Application [4]

Principala caracteristică a unei SPA este că toate schimbările de conținut și de stare sunt realizate dinamic, fără a se reîncărca întreaga pagină. În schimb, se utilizează tehnici de manipulare a DOM pentru a actualiza și reafirma secțiunile specifice ale aplicației, oferind astfel o experiență fluidă și interactivă utilizatorului.

Una dintre principalele avantaje ale unei SPA este viteza și reactivitatea, deoarece paginile nu trebuie reîncărcate în mod constant. Acest lucru conduce la o interacțiune mai fluidă și la o utilizare îmbunătățită. De asemenea, o SPA permite încărcarea diferită a resurselor, ceea ce duce la o utilizare mai eficientă a băndei de internet.

2.4.2 Comunicarea HTTP

Protocolul HTTP (Hypertext Transfer Protocol) este un standard folosit pentru a facilita schimbul de informații între client și server pe World Wide Web. În cadrul arhitecturii REST (Representational State Transfer), comunicarea prin HTTP ocupă o poziție centrală în interacțiunea dintre utilizator și serviciul web.

Principala caracteristică a comunicării HTTP în cadrul unei arhitecturi RESTful este reprezentată de utilizarea metodelor HTTP pentru a accesa și manipula resursele. Cele mai comune metode utilizate în acest context sunt GET, POST, PUT, DELETE.

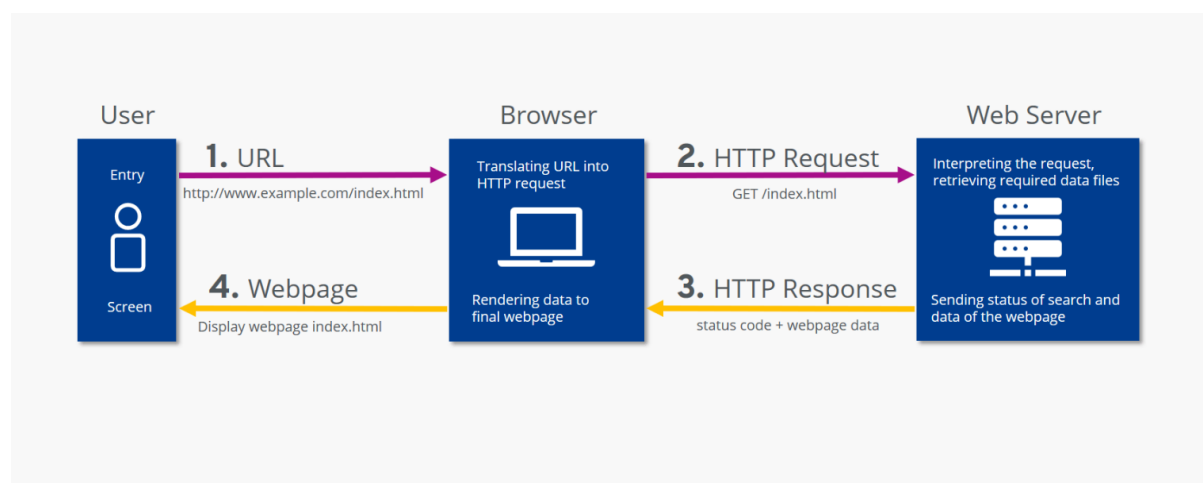


Figura 2.3: Comunicarea HTTP [5]

Comunicarea HTTP în arhitectura RESTful facilitează interoperabilitatea și integrarea între sisteme diferite. Folosind metodele HTTP standard și respectând principiile REST, pot crea servicii web scalabile, ușor de utilizat și independente de platformă.

2.4.3 JSON Web Token

JSON Web Token (JWT) este o tehnologie de securitate care permite transmiterea sigură a datelor între diferite entități într-un format compact și autentificat. Este utilizat în mod obișnuit în aplicațiile web și mobile pentru autentificare și autorizare.

JWT constă în trei părți de bază: header, payload și semnătură. Header-ul sau antetul furnizează informații despre algoritmul de criptare folosit și tipul de token. Payload-ul are datele pe care doresc să le transmit, cum ar fi informații despre utilizator sau detalii de autorizare. Semnătura este formată ca și rezultat al aplicării unui algoritm de criptare asupra header-ului, payload-ului și unei chei secrete, asigurând astfel autenticitatea și integritatea tokenului.

JWT este folosit în principal pentru autentificare și autorizare. După ce un utilizator se autentifică cu succes, un token JWT este generat și trimis către client. Acesta poate fi inclus mai departe în cererile către server pentru a valida identitatea utilizatorului. Serverul poate verifica și valida token-ul primit pentru a autoriza accesul la resursele protejate.

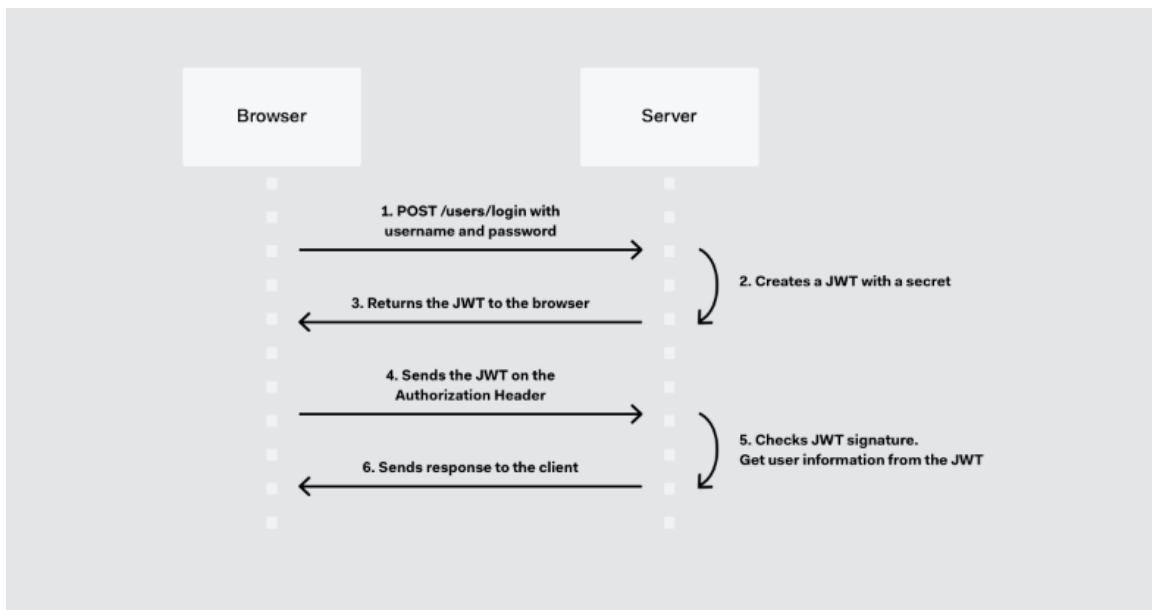


Figura 2.4: Modul de funcționare al JWT [6]

2.4.4 Metoda Code First

Metoda Code First este o abordare de dezvoltare utilizată în cadrul de lucru Entity Framework (EF) pentru gestionarea bazei de date în aplicații .NET. Ea pune accent pe definirea și configurarea modelelor de date în codul sursă al aplicației, urmând ca baza de date să fie generată automat pe baza acestor modele.

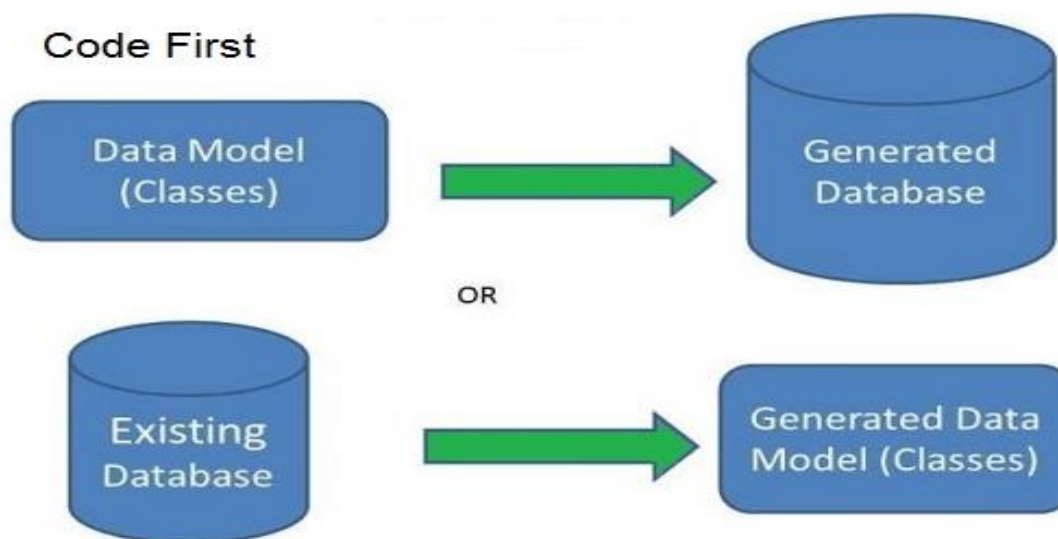


Figura 2.5: Code First vs Database First [7]

Procesul de utilizare a metodei Code First implică următorii pași:

- 1) Definirea modelelor de date: definesc clase în codul sursă al aplicației pentru a reprezenta entitățile și relațiile din domeniul de lucru. Aceste clase pot fi decorate cu attribute specifice pentru a configura aspecte precum cheile primare, relațiile între entități și restricțiile.
- 2) Crearea contextului de bază de date: creez o clasă care extinde clasa DbContext din Entity Framework. Această clasă reprezintă contextul de bază de date și conține seturile de date (DbSet) pentru fiecare entitate. De asemenea, se poate configura conexiunea la baza de date și alte opțiuni specifice.
- 3) Generarea și aplicarea migrărilor: După ce modelele de date au fost definite, Entity Framework permite generarea automată a migrărilor. Acestea reprezintă schimbările

necesare în schema bazei de date pentru a reflecta modelele de date. Migrările pot fi apoi aplicate pentru a crea sau actualiza baza de date.

- 4) Utilizarea bazei de date: După generarea bazei de date, pot utiliza contextul de bază de date pentru a interacționa cu datele. Acest lucru include operații precum adăugare, actualizare, ștergere și interogare a entităților.

Metoda "Code First" oferă un nivel ridicat de control și flexibilitate în gestionarea bazei de date în aplicații .NET. Dezvoltatorii pot defini modelele de date în mod explicit și personalizat, iar Entity Framework se ocupă de generarea schemelor și migrărilor necesare. Acest lucru facilitează dezvoltarea rapidă și iterativă, oferind totodată posibilitatea de a adăuga modificări și funcționalități suplimentare pe măsură ce aplicația evoluează.

2.4.5 Repository Pattern

Repository Pattern este o strategie de proiectare utilizată în dezvoltarea software pentru a organiza și gestiona interacțiunea cu baza de date într-o aplicație. În loc să scriu cod specific pentru fiecare operațiune de acces la date, acest șablon imi permite să definesc o interfață comună pentru accesul la date și să implementez această interfață în clase separate numite repository-uri.

Interfața repository definește metodele generice pentru a crea, citi, actualiza și șterge datele din baza de date. Fiecare repository concret implementează aceste metode în funcție de logica specifică a entităților și a mecanismului de stocare. Astfel, repository pattern imi oferă un nivel de abstractizare care separă logica de afaceri de detalii specifice ale bazei de date.

Unul dintre principalele motive pentru care am optat pentru Repository Pattern este reutilizabilitatea și flexibilitatea pe care o oferă. Prin intermediul acestui pattern, pot defini interfețe clare și abstracte pentru operațiile de bază în lucrul cu date (CRUD), cum ar fi citirea, actualizarea, crearea și ștergerea. Astfel, pot implementa diferite repository-uri care să utilizeze diferite tehnologii de stocare a datelor, precum bazele de date (relaționale sau non-relaționale), fără a afecta logica de afaceri a aplicației.

3 STRUCTURA ȘI DEZVOLTAREA APLICAȚIEI

În acest capitol voi examina arhitectura generală a aplicației, evidențiind componentele și fluxul de date între ele, mă voi concentra pe detaliile de implementare ale aplicației și voi explora diferite aspecte ale funcționalităților principale.

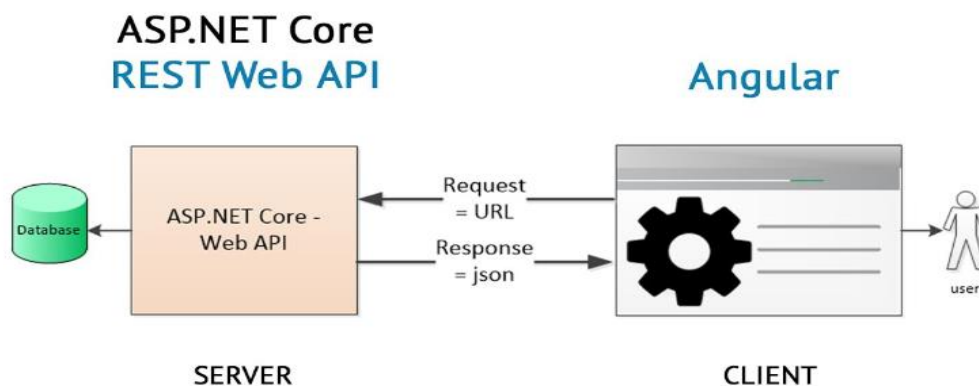


Figura 3.1: Arhitectura aplicației [8]

Arhitectura aplicației implică două componente principale: clientul (Angular) și serverul (ASP.NET Core Web API). Aceste componente lucrează împreună pentru a asigura funcționalitatea aplicației și pentru a permite interacțiunea cu utilizatorii.

Comunicația între client și server se realizează prin intermediul cererilor HTTP. Clientul Angular trimite cereri HTTP către server pentru a solicita date sau pentru a trimite informații. Serverul primește aceste cereri, le procesează și returnează răspunsurile corespunzătoare. De obicei, aceste răspunsuri sunt în format JSON și conțin datele solicitate sau rezultatele operațiilor efectuate.

Angular se concentrează pe partea de prezentare și interacțiune cu utilizatorul, în timp ce ASP.NET Core Web API se ocupă de gestionarea logicii de afaceri și interacțiunea cu baza de date. Această separare facilitează dezvoltarea, testarea și mentenanța aplicației, și permite scalabilitatea și extensibilitatea ulterioară.

3.1 ARHITECTURA APLICAȚIEI – DIAGrame UML

Diagramele UML reprezintă o modalitate eficientă de a vizualiza și de a comunica aspecte cheie ale proiectului, precum componentele, relațiile și interacțiunile dintre acestea.

Prin utilizarea diagramelor UML, pot prezenta concepte complexe într-un mod clar și coerent, facilitând înțelegerea și transmiterea informațiilor. Printre tipurile de diagrame UML frecvent utilizate în documentația software se numără diagramele de cazuri, care descriu cazurile de utilizare ale aplicației, diagramele de clase, care descriu structura entităților și relațiile dintre acestea, diagramele de secvență, care evidențiază interacțiunile între diferite componente ale sistemului într-un mod temporal, și diagramele de activitate, care ilustrează fluxurile de lucru și procesele.

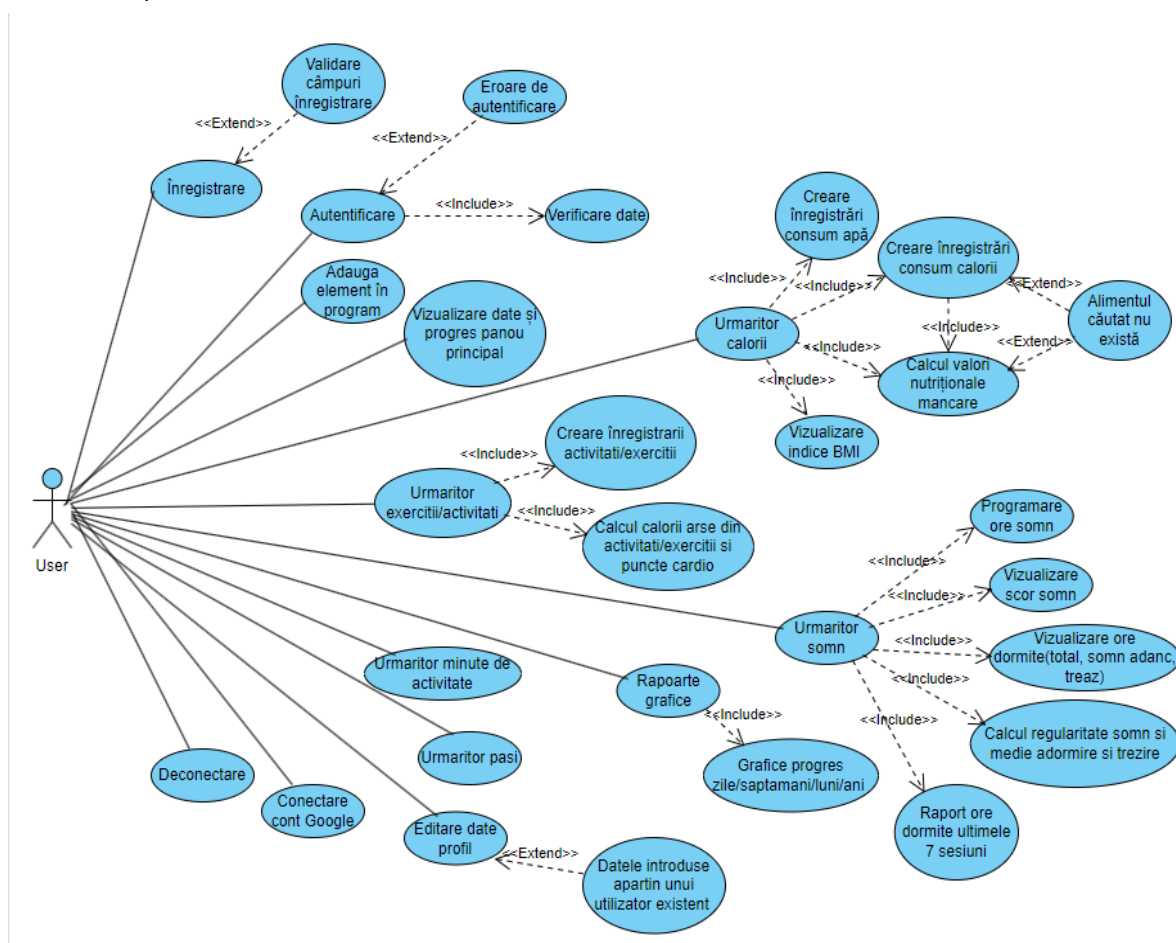


Figura 3.2: Diagrama de cazuri

Așa cum este ilustrat în diagrama de cazuri de mai sus, există un singur actor principal, utilizatorul, care poate face următoarele operațiuni:

- Autentificare/Deconectare/Editare date profil/Conectare la contul Google
- Vizualizare/monitorizare parametrii fitness: consum calorii, calorii arse, valori nutriționale ale alimentelor, număr de pași, minute de activitate, durata exercițiilor, puncte cardio, parametrii de somn.
- Operații de setare program: ore de somn, adăugare elemente/activități/programări calendar.
- Operații de creare înregistrări: hidratare, calorii consumate din alimente, calorii arse din exerciții/activități.
- Rapoarte: vizualizarea înregistrărilor pe intervale de: zile, săptămâni, luni, ani.

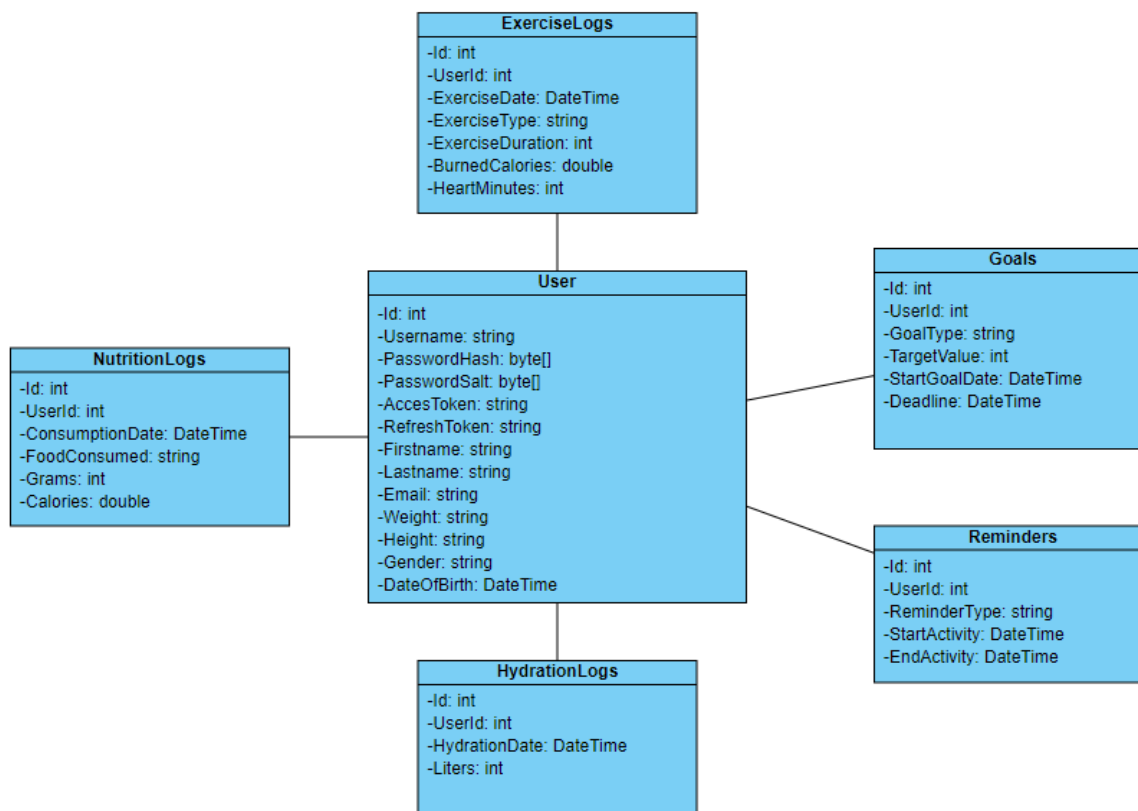


Figura 3.3: Diagrama de clase

Diagrama de clase ilustrată prezintă structura și relațiile dintre clasele cheie ale aplicației. În diagrama dată, există șase clase principale: User, NutritionLogs, ExerciseLogs,

HydrationLogs, Goals și Reminders. Fiecare dintre aceste clase este asociată cu clasele DTO și Repository corespunzătoare.

- **Clasa User:** Reprezintă un utilizator al aplicației și conține informațiile personale precum numele, prenumele, adresa de email, greutate, înălțime, gen, data nașterii, cât și informațiile sensibile precum: parola după aplicarea unei funcții de hash și valoarea unică adăugată numită "Salt" pentru a asigura securitatea parolei utilizatorilor. Tot aici avem și token-urile de acces pentru a accesa datele din aplicația mobilă "Google Fit".
- **Clasele _Logs:** Acestea reprezintă datele de nutriție, hidratare și exerciții/activități înregistrate de utilizator. Acestea sunt legate de clasa User prin proprietatea comună UserId. Fiecare clasă conține un set de proprietăți de bază cum ar fi: Id-ul înregistrării, Id-ul utilizatorului care creează înregistrarea, proprietăți de tip DateTime care reflectă timpul la care au fost create aceste înregistrări. Apoi mai există un set de proprietăți specifice pentru fiecare tip de înregistrare în parte. De exemplu, pentru clasa ExerciseLogs avem tipul exercițiului, durata acestuia, câte calorii au fost arse pe parcursul acestui exercițiu dar și punctele cardio acumulate, acestea fiind calculate în funcție de intensitatea activității aleasă la momentul adăugării unei înregistrări.
- **Clasa Goals:** Această clasă reprezintă obiectivele puse de utilizator. Acesta are posibilitatea de a-și seta obiective pentru fiecare din clasele de înregistrări, cât și pentru datele provenite dintr-o aplicație terță parte, în cazul nostru fiind vorba despre date din aplicația Google Fit.
- **Clasa Reminders:** Reprezintă elementele adăugate în programul propriu al utilizatorului din secțiunea "My schedule" din pagina principală a aplicației. Aceste elemente sau obiecte conțin un text reprezentat de proprietatea ReminderType și un interval de timp în care sunt programate să se întâmple aceste evenimente sau activități.

Fiecare dintre aceste clase este asociată cu o clasă DTO, și anume UserDTO, ExerciseLogsDTO, NutritionLogsDTO și HydrationLogsDTO, GoalsDTO, RemindersDTO și încă alte câteva clase pentru anumite funcționalități specifice cum ar fi conectarea, înregistrarea unui cont nou sau actualizarea parolei.

Aceste clase DTO reprezintă elemente esențiale pentru gestionarea transferului eficient și sigur al datelor între diferite componente ale sistemului. Aceste obiecte oferă un mod structurat și eficient de a împacheta și transmite date între nivelurile aplicației, contribuind la optimizarea comunicării între straturile de server și client. Avantajele utilizării DTO-urilor includ reducerea traficului de rețea prin transmiterea doar a informațiilor esențiale, izolarea modificărilor structurale între straturi și o gestionare mai ușoară a schimbărilor în modelele de date.

Mai departe voi prezenta diagrama claselor de control, care gestionează și coordonează logica aplicației. Acestea acționează ca un intermediar între straturile inferioare ale aplicației, cum ar fi serviciile și obiectele de acces la date, și straturile superioare, cum ar fi interfața utilizatorului sau alte sisteme externe. Clasa de control primește cereri (requests) de la frontend sau alte surse, apoi dirijează aceste cereri către serviciile adecvate pentru procesare. Rezultatele sunt apoi împachetate și returnate către frontend sau către componentele relevante ale aplicației. Această abordare ajută la organizarea și separarea logică a responsabilităților în cadrul serverului backend.

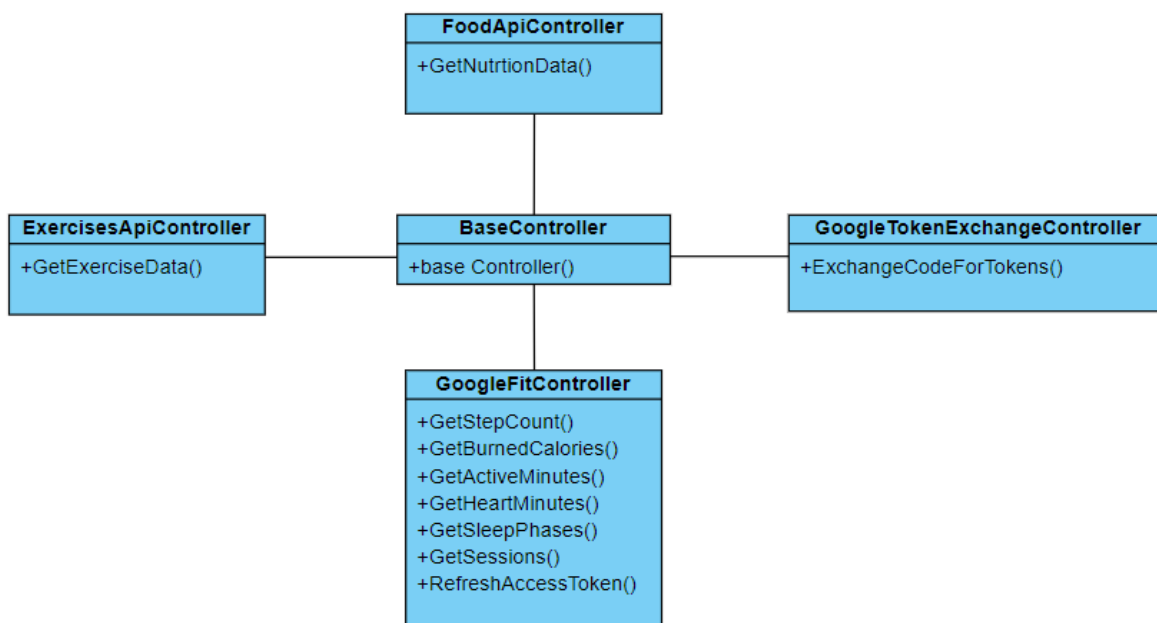


Figura 3.4: Diagrama claselor de control responsabile de comunicarea cu servere externe prin REST API.

Celorlalte clase de control le-am asociat și câte o clasă adițională Repository, practic reprezentând un nivel în plus în structura serverului care oferă metode de interacționare cu baza de date pentru accesul și manipularea datelor relevante fiecărei clase, precum crearea, citirea, actualizarea și ștergerea înregistrărilor.

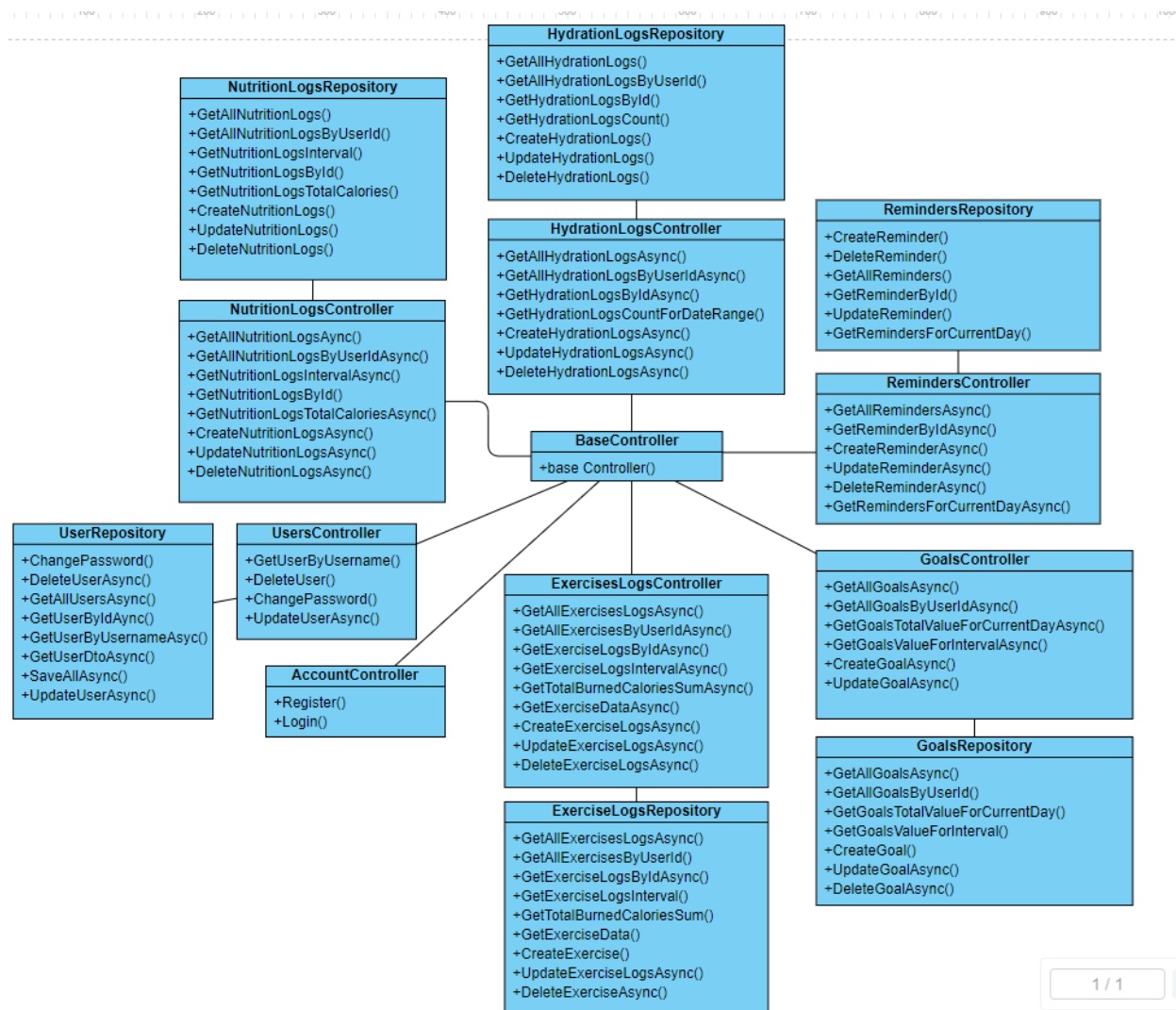


Figura 3.5: Diagrama claselor de control asociate cu clasele Repository

Această diagramă evidențiază relațiile dintre clase, totodată arătând metodele existente și aferente fiecărei clase. Mai mult, putem vedea și stratul adițional adăugat de clasele Repository care sunt asociate claselor de control. Prin urmare putem vedea mai clar

scopul claselor de control și modul în care comunică cu baza de date prin intermediul stratului Repository, folosindu-se de metodele din aceste clase pentru a-și îndeplini cererile venite din frontend și pentru a trimite răspunsul cu informațiile necesare.

3.2 ARHITECTURA SERVERULUI

Arhitectura curată în ASP.NET Core Web API este o abordare arhitecturală care promovează modularitatea, separarea responsabilităților și testabilitatea în dezvoltarea aplicațiilor web. Aceasta se bazează pe principii precum împărțirea proiectului în straturi și dependențe inversate pentru a obține un design flexibil și modular.

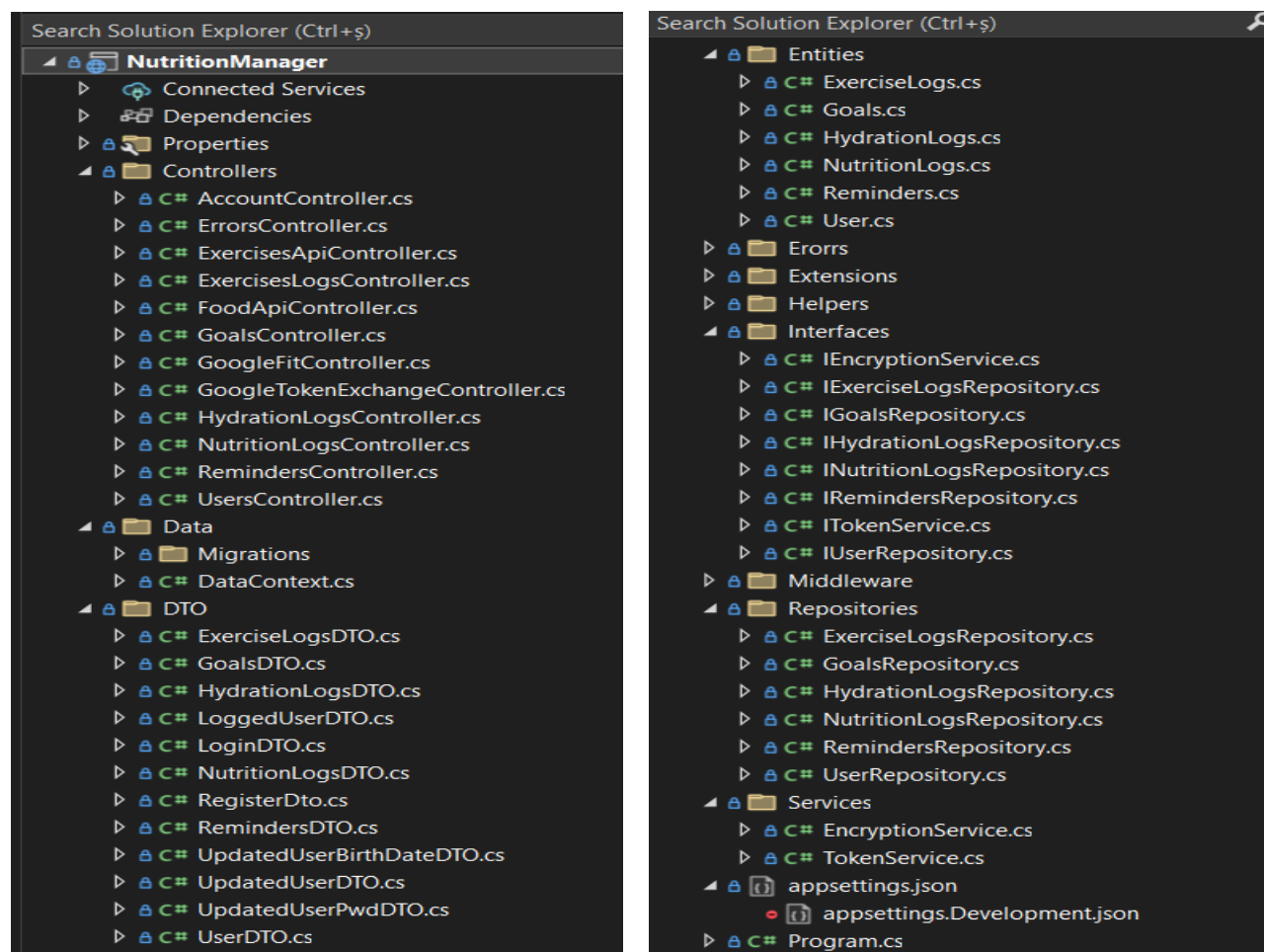


Figura 3.6: Structura fișierelor în server

Aceste foldere reprezintă componente cheie în structura aplicației și joacă un rol crucial în organizarea și modularizarea codului sursă. Ele contribuie la crearea unei arhitecturi robuste și scalabile, care facilitează dezvoltarea, întreținerea și extinderea aplicației pe termen lung. Fiecare folder are un scop bine definit și îndeplinește anumite responsabilități, asigurând separarea logică a funcționalităților și a modulelor.

Aplicația pe care am dezvoltat-o are mai multe funcționalități, putând să le categorizez în două părți:

- Funcționalități de bază ale aplicației, de sine stătătoare, având legătură direct cu baza de date relațională creată special pentru această aplicație.
- Funcționalități bazate pe servere sau servicii externe, implementate prin comunicarea cu API-urile puse la dispoziție de Google sau APINinjas. Această comunicare se desfășoară folosind modelul architectural de principii și convenții pentru dezvoltarea și interacționarea cu servicii web numită REST API.

În aplicația mea, am ales ca arhitectura serverului să urmeze o abordare stratificată, care este un model obișnuit și des folosit în dezvoltarea software.

3.2.1 Nivelul de prezentare

Primul nivel este nivelul de prezentare care se ocupă de logica de prezentare și este responsabil pentru interacțiunea cu clienții, primirea și procesarea cererilor HTTP și trimiterea unor răspunsuri corespunzătoare. Acest strat este reprezentat de clasele de control pe care le-am văzut mai sus în diagramele de clase.

Controlerele sau clasele de control sunt definite ca și clase care utilizează atributul [ApiController] și sunt decorate cu diferite attribute și adnotări pentru a configura comportamentul și rutarea cererilor HTTP.

Clasa NutritionLogsController reprezintă un controler în cadrul aplicației mele responsabil de gestionarea cererilor HTTP referitoare la entitatea NutritionLogs. Acesta este un exemplu de implementare a unui controler API simplu.

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class NutritionLogsController : ControllerBase
{
```

Figura 3.7: Atributul ApiController

Controlerul este atribuit cu atributul `[Route("api/[controller]")]`, ceea ce definește ruta de bază pentru toate acțiunile din acest controler. De exemplu, pentru această clasă, ruta de bază va fi `"api/NutritionLogs"`. Controlerul utilizează și atributul `[ApiController]`, care oferă suport integrat pentru validarea automată a cererilor, serializarea și alte funcționalități comune pentru controlerele API.

```
[HttpGet("interval")]
public async Task<IEnumerable<NutritionLogs>> GetNutritionLogsIntervalAsync(int
userId, DateTime startDate, DateTime endDate)
{
    return await _nutritionLogsRepository.GetNutritionLogsInterval(userId,
startDate, endDate);
}
```

Codul 1: Metodă din cadrul clasei de control NutritionLogsController

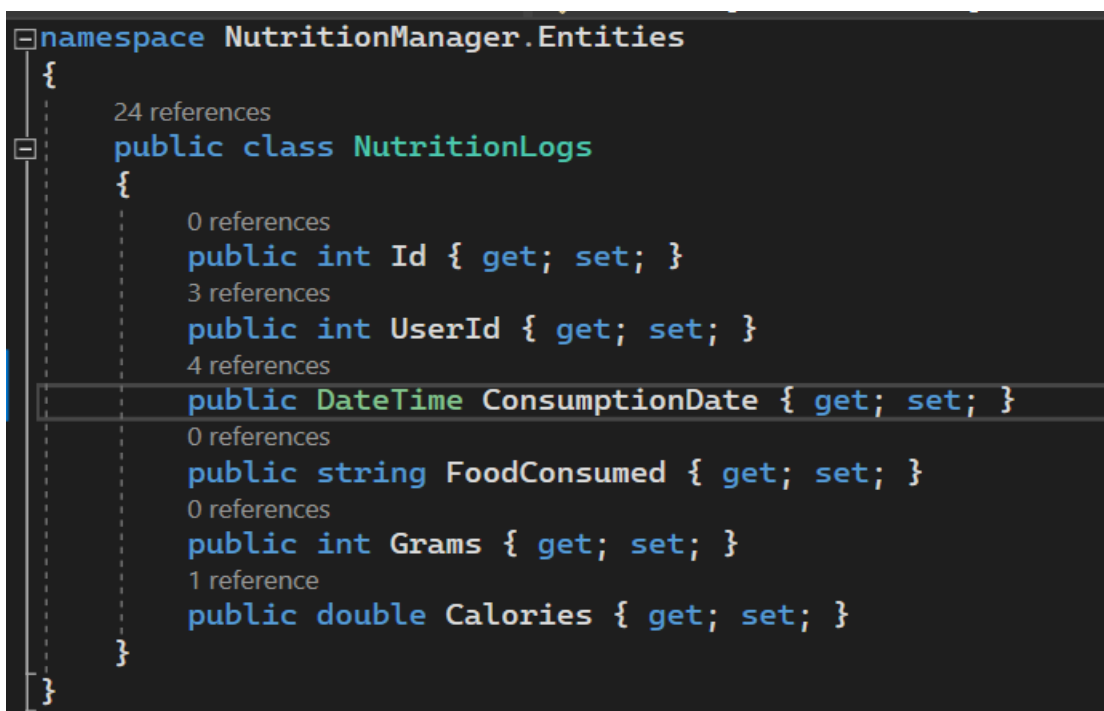
Metoda „GetNutritionLogsIntervalAsync” este o acțiune definită în controlerul NutritionLogsController și este marcată cu atributul `[HttpGet]`. Această acțiune este responsabilă de gestionarea cererilor HTTP de tip GET adresate către ruta `"api/NutritionLogs/interval"`. Aceasta va returna ca și răspuns un string JSON care conține toate înregistrările din categoria nutriție din baza de date, create de utilizatorul cu ID-ul transmis ca și parametru, în intervalul dat de parametrii `startDate` și `endDate` de tip `DateTime`

```
{
  "id": 38,
  "userId": 1,
  "consumptionDate": "2024-01-10T22:11:21.1299601",
  "foodConsumed": "burger",
  "grams": 150,
  "calories": 356.6
},
{
  "id": 45,
  "userId": 1,
  "consumptionDate": "2023-12-10T00:00:00",
  "foodConsumed": "cake",
  "grams": 200,
  "calories": 787.2
}
```

Figura 3.8: Exemplu de răspuns JSON

3.2.2 Nivelul de domeniu

Al doilea nivel al serverului este nivelul de domeniu. Acest strat încapsulează logica centrală a afacerii și regulile de bază. Definește entități, agregate, obiecte de valoare și servicii care reprezintă și gestionează conceptele de afaceri. În acest nivel găsim entitățile, definite încă de la începutul aplicației. Acestea servesc un rol important în logica centrală a afacerii și a reprezentării conceptelor de afaceri, acestea reflectând fidel realitatea domeniului de aplicare al acestui proiect.



```
namespace NutritionManager.Entities
{
    24 references
    public class NutritionLogs
    {
        0 references
        public int Id { get; set; }
        3 references
        public int UserId { get; set; }
        4 references
        public DateTime ConsumptionDate { get; set; }
        0 references
        public string FoodConsumed { get; set; }
        0 references
        public int Grams { get; set; }
        1 reference
        public double Calories { get; set; }
    }
}
```

Figura 3.9: Exemplu de Entitate

Entitățile în ASP.Net Core Web API sunt de obicei implementate ca și clase, care conțin proprietăți care reprezintă caracteristicile și atributele entității respective. Aceste proprietăți pot fi adnotate cu atribute speciale, cum ar fi [Required] sau [MaxLength], pentru a specifica reguli de validare și constrângeri asupra datelor.

Clasa NutritionLogs reprezintă o entitate cheie în cadrul domeniului aplicației mele și este utilizată pentru gestionarea informațiilor despre detaliile consumului de alimente și mâncăruri ale utilizatorului. Aceasta încapsulează datele și momentul în care utilizatorul a

creat o înregistrare, oferind o abstracție a datelor și comportamentului asociate acestora în cadrul domeniului de business al aplicației.

3.2.3 Nivelul de infrastructură sau persistență

Al treilea nivel al serverului este nivelul de infrastructură sau persistență. Acest strat se ocupă de accesul la date și de aspectele legate de stocarea acestora. Are rolul de a interacționa cu baza de date, API-uri externe sau orice altă sursă de date.

În cadrul aplicației, acest nivel este reprezentat de clasele Repository care implementează logica de bază pentru acces la date (operații CRUD).

Un repository este responsabil pentru abstractizarea detaliilor specifice ale accesului la date și oferă o interfață consistentă și simplificată pentru a realiza operații de citire, scriere, actualizare și ștergere a datelor. Astfel, repository-ul oferă un nivel de abstractizare între aplicație și resursele de date, permițând o separare clară a responsabilităților și o gestionare mai ușoară a datelor.

Utilizarea unui repository în cadrul controlerului permite separarea logică dintre gestionarea cererilor HTTP și accesul la date. Astfel, controlerul nu trebuie să cunoască detalii specifice despre modul în care sunt stocate și accesate datele, ci doar să utilizeze interfețele oferite de repository pentru a efectua operațiile dorite.

```
public class NutritionLogsRepository : INutritionLogsRepository
{
    private readonly DataContext _context;

    public NutritionLogsRepository(DataContext context)
    {
        _context = context;
    }

    public async Task<NutritionLogs> CreateNutritionLogs(NutritionLogs
nutritionLogs)
    {
        _context.Add(nutritionLogs);
        await _context.SaveChangesAsync();
        return nutritionLogs;
    }
}
```

Codul 2: Exemplu de clasă Repository

NutritionLogsRepository implementează interfața INutritionLogsRepository și oferă metode pentru a realiza operații asupra entităților NutritionLogs. Aceste metode includ obținerea tuturor înregistrărilor de tip NutritionLogs, obținerea unei înregistrări după ID, crearea înregistrărilor, actualizarea sau ștergerea acestora, obținerea înregistrărilor pe un interval de timp dat sau obținerea totalului de calorii consumate din toate înregistrările într-un interval dat.

INutritionLogsRepository este un contract care definește operațiile permise asupra datelor legate de jurnalele nutriționale, contribuind la separarea logicii de acces la date de detaliile specifice ale implementării.

NutritionLogsRepository este injectat în controlerul NutritionLogsController prin intermediul constructorului, utilizând mecanismul de injecție de dependențe al cadrului de lucru ASP.NET Core. Prin utilizarea metodei ToListAsync din Entity Framework Core, operația de acces la date este efectuată în mod asincron, evitând blocarea firului de execuție și îmbunătățind performanța generală a aplicației.

3.2.4 Nivelul de interfețe sau adaptoare

Al patrulea nivel este nivelul de interfețe sau adaptoare. Acesta, în contextul arhitecturii software, este un nivel care oferă interfețe sau adaptori pentru comunicarea cu alte sisteme, module sau servicii externe. Acest strat îndeplinește mai multe roluri esențiale într-un sistem software, contribuind la separarea componentelor, gestionarea dependențelor externe și facilitarea integrării.

În cadrul aplicației mele, acest nivel este reprezentat de clasele DTO. Obiectele de transfer de date reprezintă o clasă utilizată pentru a transfera date între straturi sau componente diferite ale aplicației. Scopul principal al DTO-urilor este de a facilita transferul eficient și sigur al datelor între client și server.

DTO-urile sunt adesea utilizate pentru a defini structura și conținutul datelor care sunt trimise sau primite prin intermediul serviciilor API. Acestea pot fi utilizate pentru a izola entitățile de baza de date sau alte modele de date interne și pentru a expune doar informațiile relevante către client.

3.3 ARHITECTURA INTERFEȚEI CU UTILIZATORUL

Interfața utilizator reprezintă componenta principală a aplicației prezentate, și este responsabilă de interacțiunea cu utilizatorul. Pentru implementarea interfeței am ales Angular, cadru de lucru pe care l-am descris mai în detaliu și cu mai multe amănunte în subcapitolul Cadre de lucru. Am ales Angular deoarece oferă un set bogat de funcționalități și abstracții care simplifică dezvoltarea unei aplicații web complexe, precum și o structură modulară.

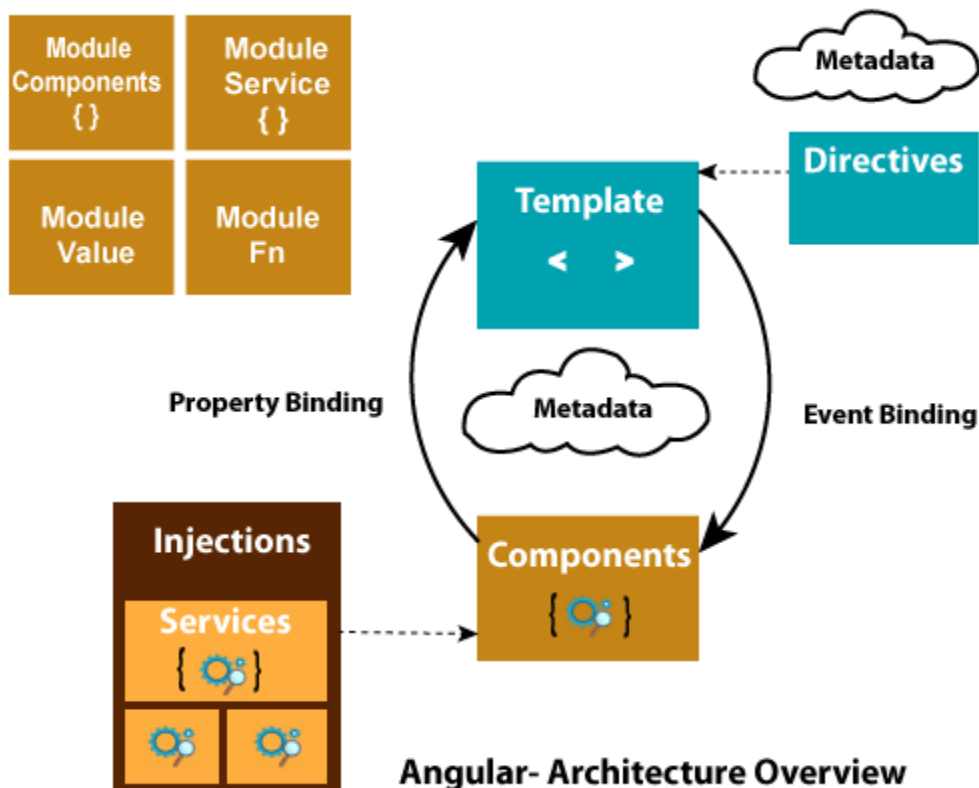


Figura 3.10: Arhitectura Angular [9]

3.3.1 Module

În Angular, aplicațiile sunt modulare și utilizează conceptul de NgModule. Fiecare aplicație are cel puțin un NgModule, numit AppModule, care este modulul rădăcină și este utilizat pentru pornirea aplicației. NgModule descrie modul în care părțile aplicației se încadrează împreună și are câteva proprietăți cheie în configurarea sa.

Un modul Angular este o colecție logică de componente, directive, servicii și alte resurse legate, care sunt grupate împreună pentru a oferi o funcționalitate specifică. Modulul

reprezintă o unitate modulară independentă, care poate fi încărcată și utilizată în cadrul aplicației.

3.3.2 Directive și componente

Directivele sunt clase decorate cu decoratorul `@Directive` și pot fi aplicate ca atribute pe elemente HTML existente sau pot crea elemente HTML personalizate. Există două tipuri principale de directive: directivele structurale și directivele de atribut.

```
<div class="auth-wrapper">
  <form #authForm="ngForm" (ngSubmit)="onSubmit(authForm)" class="form-
  wrapper">
```

Codul 3: Formular de autentificare și înregistrare în HTML

Această secțiune de cod reprezintă formularul de autentificare și înregistrare. Utilizând directiva `ngForm`, formularul este legat de componenta corespunzătoare și este gestionat prin intermediul metodei `onSubmit()`.

```
<mat-form-field appearance="fill" class="example-full-width" *ngIf="!isLoginMode">
  <mat-label>First Name</mat-label>
  <input matInput placeholder="Enter your firstname" type="name"
name="firstname" ngModel [required]="true">
</mat-form-field>
<mat-form-field appearance="fill" class="example-full-width"
*ngIf="!isLoginMode">
  <mat-label>Last Name</mat-label>
  <input matInput placeholder="Enter your lastname" type="name" name="lastName"
ngModel [required]="true">
</mat-form-field>
```

Codul 4: Secțiune de cod cu câmpuri care conțin directiva `mat-form-field`

În funcție de valoarea variabilei `isLoginMode`, sunt afișate sau ascunse câmpurile pentru nume, prenume și email. Câmpurile sunt definite folosind directiva `mat-form-field` și sunt validate utilizând atributul `[required]`.

Componentele sunt definite prin intermediul decoratorului `@Component` și pot fi considerate ca și clase cu metode și proprietăți specifice. Componentele sunt definite prin intermediul acestui decorator, care primește un obiect de metadata ce descrie comportamentul și aspectul componentei.

```
@Component({
  selector: 'app-live-clock',
  templateUrl: './live-clock.component.html',
  styleUrls: ['./live-clock.component.css']
})
export class LiveClockComponent implements OnInit {

  dateTime!: Date;

  ngOnInit() {
    timer(0, 1000).subscribe(() => {
      this.dateTime = new Date;
    })
  }
}
```

Codul 5: Cod sursă pentru Componenta LiveClock

Componenta este definită prin intermediul acestui decorator, care specifică selectorul (selector: 'app-live-clock') utilizat pentru a include componenta în template-urile altei componente, șablonul HTML (templateUrl: './live-clock.component.html') care definește aspectul componentei și elementele vizibile în interfața cu utilizatorul, și fișierele CSS (styleUrls: ['./live-clock.component.css']) asociate componentei pentru stilizare.

În cadrul acestei componente, inițializările sunt plasate în metoda **ngOnInit**. Aceasta este chemată o singură dată, după ce legăturile clasei au fost stabilite. În contextul acesta, nu este nevoie de constructor pentru că nu avem dependențe de injectat. Constructorul este destinat exclusiv inițializării membrilor clasei și injectării dependențelor. Acest lucru se datorează faptului că constructorul este apelat înainte ca componenta să fie creată, în timp ce **ngOnInit** va fi apelată doar după crearea componentei. Această metodă face parte din interfața **OnInit**, astfel încât clasa trebuie să implementeze această interfață.

3.3.3 Injectarea dependențelor

Un aspect arhitectural important al interfeței în Angular este injectarea dependențelor (DI). Acesta este un mecanism prin care furnizez o instanță nouă a unei clase cu dependențele sale complete. În Angular, DI este gestionat de Injector, un serviciu de gestionare a dependențelor care se ocupă de înregistrarea dependențelor și de crearea lor.

```

constructor(
  private fb: FormBuilder,
  private snackBar: MatSnackBar,
  private googleAPIService: GoogleAPIService,
  private sleepRegularityService: SleepRegularityService
) {}

```

Codul 6: Exemplificarea unui constructor de componentă în Angular

Injectarea dependențelor se bazează pe constructorii claselor. Atunci când Angular creează o componentă, acesta solicită injectorului serviciile de care componenta are nevoie. Injectorul menține o colecție de instanțe de servicii create anterior. Atunci când o instanță de serviciu este solicitată, injectorul verifică dacă aceasta există deja în colecție. Dacă serviciul nu se află în colecție, injectorul creează o nouă instanță și o adaugă în colecție înainte de a o returna către Angular. Odată ce toate serviciile solicitate au fost rezolvate și returnate, Angular poate apela constructorul componentei și poate furniza acele servicii ca argumente.

3.3.4 Servicii

Serviciile reprezintă unul dintre cele mai importante concepte în arhitectura Angular. Ele sunt utilizate pentru a împărți și gestiona logica de afaceri, funcționalitățile comune și accesul la resurse externe în întreaga aplicație. Serviciile furnizează o modalitate eficientă de a separa preocupările și de a încapsula funcționalitățile specifice într-un mod reutilizabil și modular.

```

@Injectable({
  providedIn: 'root'
})
export class GoogleAPIService {
  constructor(private http: HttpClient) { }
  getStepCount(userId: number, startTimeMillis: number, endTimeMillis: number):
  Observable<any> {
    const endpoint = environment.userManagement.baseUrl + 'GoogleFit/StepsCount/' +
    userId;
    const body = {
      StartTimeMillis: startTimeMillis,
      EndTimeMillis: endTimeMillis
    };
    return this.http.post<any>(endpoint, body);
  }
}

```

Codul 7: Cod sursă pentru exemplificarea unui serviciu din cadrul aplicației

Serviciul `GoogleAPIService` utilizează injectarea de dependențe pentru a obține o instanță a serviciului `HttpClient`, care este folosit pentru a efectua cereri HTTP către serverul API. Astfel, serviciul se bazează pe `HttpClient` pentru comunicarea cu serverul pentru a trimite cereri și a primi răspunsuri cu date relevante pentru utilizator. Prin intermediul metodelor sale (`getStepCount()`, `getActiveMinutes()`, `getSleepPhases()`, `getSessions()`, `getBMRCalories()`, `getHeartMinutes()`), `GoogleAPIService` oferă funcționalități precum trimiterea cererilor pentru date precum: număr de pași efectuați în ziua respectivă, minutele de activitate, sesiuni de somn și detaliile despre acestea, kaloriile arse de metabolism, puncte cardio. Aceste servicii trimit cereri HTTP de tip GET sau POST către serverul aplicației, iar serverul se ocupă mai departe de trimiterea cererilor de HTTP către serverul Google cu informațiile necesare pentru a primi răspunsurile cu datele cerute.

Prin separarea funcționalităților într-un serviciu dedicat, aplicația devine mai modulară, ușor de întreținut și extensibilă. Alte componente pot utiliza serviciul `GoogleAPIService` prin injectarea sa în constructorul lor, permițându-le să beneficieze de funcționalitățile acestui serviciu fără a duplica codul și logica asociată.

3.4 ELEMENTE DE IMPLEMENTARE

3.4.1 Pagina de autentificare

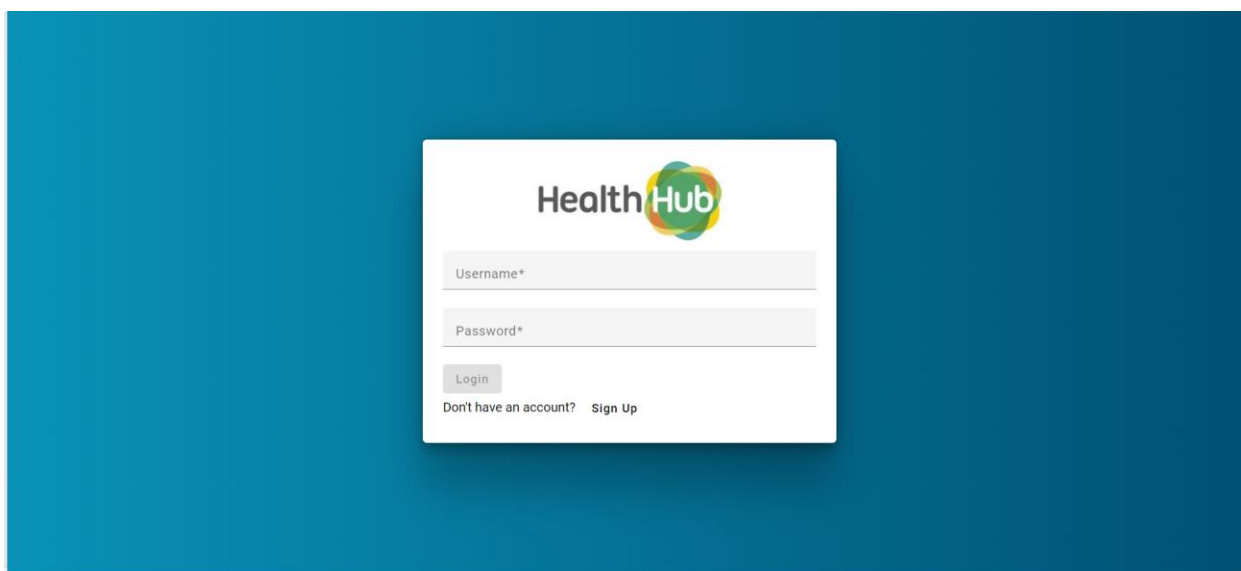


Figura 3.11: Pagina de autentificare

Ca și funcționalitate, pagina de autentificare este obișnuită, având trei părți principale: autentificarea, înregistrarea și validarea datelor, care este inclusă în autentificare și înregistrare.

Clasa AccountController este o clasă de controler în cadrul serverului, responsabilă de gestionarea operațiilor legate de autentificare și autorizare. În constructorul clasei AccountController, sunt injectate trei dependențe necesare pentru funcționarea controlerului: DataContext, ITokenService, IMapper.

```
[HttpPost("register")]
public async Task<ActionResult<LoggedUserDTO>> Register(RegisterDto registerDto)
{
    if (await UserExists(registerDto.Username))
        return BadRequest("Username is taken");

    using var hmac = new HMACSHA512();
    var user = new User
    {
        Username = registerDto.Username.ToLower(),
        PasswordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(registerDto.Password)),
        PasswordSalt = hmac.Key,
        RefreshToken = "",
        Firstname = registerDto.Firstname,
        Lastname = registerDto.Lastname,
        Email = registerDto.Email,
        Weight = registerDto.Weight,
        Height = registerDto.Height,
        Gender = registerDto.Gender,
        DateOfBirth = registerDto.DateOfBirth
    };

    _context.Users.Add(user);
    await _context.SaveChangesAsync();
}
```

Codul 8: Metoda Register din clasa AccountController

Pentru a crea un nou utilizator, se folosește clasa HMACSHA512 pentru generarea hash-ului parolei. Parola este convertită la litere mici folosind ToLower() pentru a asigura coerența. Hash-ul rezultat și cheia generată sunt asiguate proprietăților PasswordHash și

PasswordSalt ale obiectului User. Aceste valori vor fi stocate în baza de date pentru verificarea ulterioară a parolei.

Apoi, obiectul User este adăugat în setul de utilizatori (Users) al contextului de date _context și se apelează metoda SaveChangesAsync() pentru a salva modificările în baza de date.

```
[HttpPost("login")]
public async Task<ActionResult<LoggedUserDTO>> Login(LoginDTO loginDTO)
{
    var user = await _context.Users.SingleOrDefaultAsync(x => x.Username == loginDTO.Username);

    if (user == null)
    {
        return Unauthorized("invalid username");
    }

    using var hmac = new HMACSHA512(user.PasswordSalt);
    var computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(loginDTO.Password));

    for (int i = 0; i < computedHash.Length; i++)
    {
        if (computedHash[i] != user.PasswordHash[i])
            return Unauthorized("invalid password");
    }
    return new LoggedUserDTO
    {
        Username = user.Username,
        Token = _tokenService.CreateToken(user)
    };
}
```

Codul 9: Metoda Login din clasa AccountController

Metoda primește un obiect LoginDTO ca parametru, care conține numele de utilizator și parola introduse de utilizator în momentul autentificării.

Pentru verificarea parolei, se folosește cheia (PasswordSalt) stocată în obiectul User găsit în baza de date. Se utilizează clasa HMACSHA512 cu această cheie pentru a calcula hash-ul parolei introduse în formularul de autentificare (loginDTO.Password). Se compară hash-ul obținut cu hash-ul stocat în obiectul User pentru a valida parola. Dacă hash-urile nu se potrivesc, se returnează un răspuns Unauthorized cu mesajul "invalid password".

Dacă autentificarea este reușită, se realizează o mapare a obiectului User într-un obiect LoggedUserDTO. Token-ul JWT este generat pentru noul utilizator utilizând serviciul _tokenService și este asignat proprietății Token a obiectului LoggedUserDTO împreună cu numele de utilizator.

3.4.2 Pagina tabloului de bord

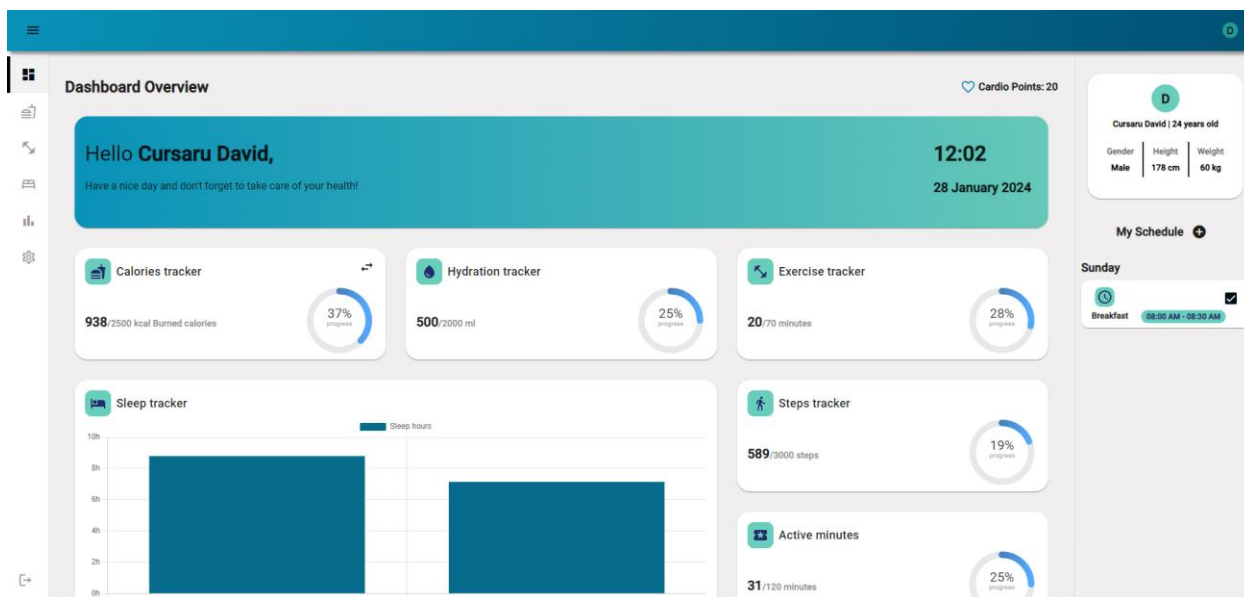


Figura 3.12: Pagina Dashboard overview

Dupa autentificare, prin intermediul fișierului de typescript app-routing.module din Angular, direcționez utilizatorul în pagina principală a aplicației numită *Dashboard overview*. Aici utilizatorul poate să monitorizeze și să vizualizeze toți parametrii de fitness, nutriție sau sănătate din cadrul aplicației, având o privire de ansamblu asupra progresului și al programului acestuia.

Interfața cu utilizatorul este formată din mai multe carduri care conțin informații specifice și o bară extensibilă în partea dreaptă. Mai este și o bară extensibilă de navigare în partea stângă, prin care utilizatorul poate naviga prin toate paginile aplicației.

Pentru carduri am folosit componenta din Angular Material *mat-card*, folosindu-mă și de *mat-grid-list* și *mat-grid-tile* pentru a plasa cardurile în pozițiile dorite, astfel formând așezarea pe care o puteți vedea și în figura de mai sus. M-am folosit de directiva *ngFor* pentru a genera toate cardurile dinamic, afișând conținutul fiecărui card în funcție de indicii acestora.

Această așezare cu toate elementele ei este și adaptabilă la mărimi de ecran diferite. În acest sens am folosit *BreakpointObserver* care este o clasă din Angular Material care permite monitorizarea schimbărilor în dimensiunile ecranului sau în alți factori care definesc punctele de rupere (breakpoints). În funcție de aceste puncte definesc aranjamente specifice ale cardurilor folosindu-mă de proprietățile ale componentelor *mat-grid-list* și *mat-grid-tile* pentru numărul de coloane și de linii.

Fiecare card conține date specifice cum ar fi numărul de calorii arse, număr de pași, cantitatea de apă consumată etc. Toți acești parametri sunt monitorizați doar pe ziua curentă, la începutul fiecărei zile pornindu-se de la 0. Totodată, cardurile conțin și obiectivele setate pe ziua respectivă pentru fiecare parametru în parte.

Pentru cercul de progres care reprezintă procentajul de progres din totalul setat ca și obiectiv, am folosit componenta *ng-circle-progress* pe care am adăugat-o prin instrumentul *npm* folosind comanda: *npm i ng-circle-progress*. Această componentă vine cu mai multe proprietăți de design, elemente vizuale, unități, titlu, subtitlu, efecte și valorile procentajelor. Prin proprietatea *percent* am afișat procentajele calculate pentru fiecare parametru monitorizat.

Pentru graficul din interiorul cardului *Sleep tracker* am folosit libraria *ng2-charts* pe care am instalat-o folosind comanda: *npm i ng2-charts*. Aici avem afișate durata sesiunilor de somn din ultimele 7 zile (sesiunile înregistrate cu un smartwatch prin aplicația GoogleFit). Este de menționat că vor fi afișate doar zilele în care au fost înregistrate sesiuni de somn, deci dacă în ultimele 7 zile au fost înregistrate 2 sesiuni de somn, vom avea doar zilele respective cu orele de somn aferente zilelor acelea în grafic.

Toată logica acestei componente constă într-un număr mare de funcții pe care le apelez în metoda *ngOnInit()*. Aceasta este o metodă specifică Angular care face parte din ciclul de viață al unei componente. Această metodă este apelată de către Angular după ce componenta a fost creată și toate datele de intrare (input) au fost inițializate, dar înainte de a fi afișată pe ecran.

Datele de nutriție, hidratare și activitate fizică afișate în aceste carduri provin din două surse:

- Baza de date relațională SQL, care conține informațiile și înregistrările create de utilizator pentru calorii consumate, calorii arse din exerciții, apă consumată, durata totală a exercițiilor.
- Date provenite din aplicația de mobil Google Fit, unde utilizatorul își poate monitoriza numărul de pași, minutele de activitate, kaloriile arse de metabolism cât și date despre somn

Pentru a prelua datele necesare din baza de date, am creat servicii care să interacționeze cu serverul, și să facă cereri HTTP cu detaliile necesare cum ar fi ID-ul utilizatorului autentificat, cât și intervalul de timp pentru care vrem să luăm date.

```
getWaterQuantity(loggedUserId: any, startDate: string, endDate: string) {  
    const endpoint = environment.userManagement.baseUrl +  
'hydrationLogs/count?userId=' + loggedUserId + '&startDate=' + startDate +  
'&endDate=' + endDate;  
    return this.http.get(endpoint);  
}
```

Codul 10: Serviciu pentru luarea cantității de apă consumate într-un interval de timp

Acesta este un exemplu de serviciu simplu prin care comunicăm cu serverul pentru a prelua informații despre cantitatea de apă consumată de utilizator într-un interval de timp dat.

Pentru a prelua date din Google Fit, am folosit aceeași procedură, doar că formatul pentru parametrii startDate și endDate trebuie să fie în milisecunde.

```
getBMRCalories(userId: number, startTimeMillis: number, endTimeMillis: number):  
Observable<any> {  
    const endpoint = environment.userManagement.baseUrl +  
'GoogleFit/BMRCalories/' + userId;  
    const body = {  
        StartTimeMillis: startTimeMillis,  
        EndTimeMillis: endTimeMillis  
    };  
  
    return this.http.post<any>(endpoint, body);  
}
```

Codul 11: Serviciu pentru preluarea kaloriilor arse de metabolism din Google Fit

Accesarea datelor din Google Fit implică folosirea protocolului de securitate OAuth 2.0. Acesta este un protocol de autorizare deschis și standardizat, utilizat pentru acordarea securizată a accesului la resursele unui utilizator fără a dezvălui detaliile de autentificare. A fost proiectat pentru a oferi un mecanism simplu și sigur pentru autentificarea și autorizarea aplicațiilor la serviciile web, fără a dezvălui parolele utilizatorilor.

Fluxul de autorizare folosit în cadrul aplicației mele este *Authorization Code Grant*. În acest flux, aplicația client primește un cod de autorizare după autentificarea utilizatorului cu credențialele contului Google, iar apoi schimbă acest cod pentru un token de acces și un token de actualizare. În momentul autentificării utilizatorului cu credențialele Google, aplicația mea cere permisiunea utilizatorului de a-i citi datele de somn și de activitate. După acceptarea permisiunilor, utilizatorul este redirecționat la pagina principală a aplicației mele cu un cod de autorizare ca și parametru al URL-ului de redirecționare. Acest cod este mai apoi trimis către server, unde se face o cerere HTTP POST cu un corp care conține codul de autorizare, ID-ul clientului, URL-ul de redirecționare și metoda de autorizare folosită. Dacă cererea este reușită, vom primi ca și răspuns un JSON cu token-ul de acces care este mai departe folosit pentru a putea face cereri pentru date din Google Fit, cât și un token de actualizare. Pentru motive de securitate, token-ul de acces are o durată de validitate de aproximativ 60 de minute, iar pentru a putea efectua cereri către server în continuare, fără a cere utilizatorului să se conecteze din nou cu credențialele Google, folosim token-ul de actualizare pentru a genera un nou token de acces de fiecare dată când acesta expiră.

```
try
{
    string clientId = _configuration["GoogleAuth:ClientId"]; ;
    string clientSecret = _configuration["GoogleAuth:ClientSecret"];
    string redirectUri = "http://localhost:4200/layout/dashboard";
    var requestData = new Dictionary<string, string>
    {
        { "code", authorizationCode },
        { "client_id", clientId },
        { "client_secret", clientSecret },
        { "redirect_uri", redirectUri },
        { "grant_type", "authorization_code" }
    };
};
```

```
var client = _httpClientFactory.CreateClient();
var response = await client.PostAsync("https://oauth2.googleapis.com/token",
new FormUrlEncodedContent(requestData));
```

Codul 12: Obținerea token-urilor de acces de la Google

Pentru a se conecta la contul său Google și pentru a porni practic fluxul de generare a codului de autorizare și a token-urilor de acces necesare, utilizatorul poate să navigheze în pagina *Profile settings* a aplicației, unde este butonul *Sing in with Google*. Apăsând acest buton, se începe fluxul OAuth 2.0, cerându-se permisiunile de citire sau scriere a datelor pe care le prelucrează aplicația.

3.4.3 Pagina de monitorizare a caloriilor

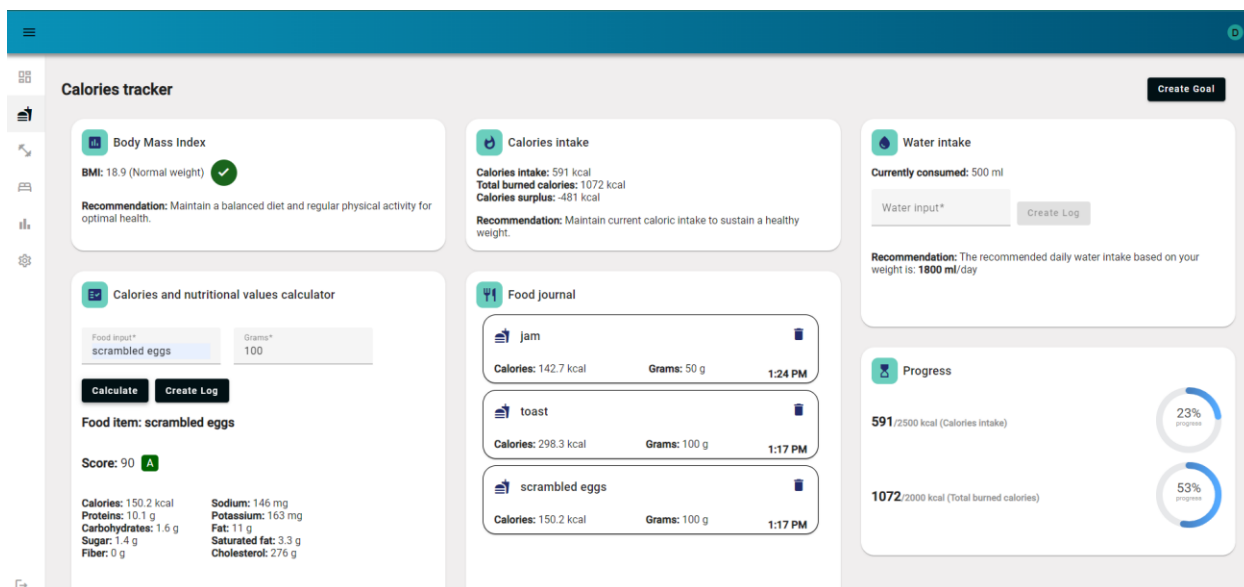


Figura 3.13: Pagina Calories tracker

Pagina *Calories tracker* urmează aceeași structură vizuală ca și pagina principală, toate datele și informațiile relevante fiind afișate ca și conținut în componentele *mat-card* din Angular Material.

În această pagină, utilizatorul își poate vedea indicele de masă al corpului calculat în funcție de înălțimea lui și greutatea lui, totodată fiind disponibile și niște recomandări generale pentru fiecare categorie de greutate. Totodata, în urmatorul card se află date despre caloriile consumate, caloriile arse și diferența dintre acestea două fiind afișată ca și surplusul caloriilor.

Un alt card deține funcționalitatea de a crea înregistrări de hidratare, în care utilizatorul își poate pune cantitatea de apă consumată în mililitri. Totodată este vizibilă și cantitatea totală de apă consumată în ziua respectivă, cât și o recomandare generală privind cantitatea de apă pe care utilizatorul ar trebui să o consume în funcție de greutatea lui.

Funcționalitatea principală a acestei pagini este calculatorul nutrițional de alimente și mâncăruri. Utilizatorul poate introduce alimente sau mâncăruri și cantitatea acestora ca să afle valorile nutriționale pe care le oferă. Am implementat și un sistem de punctare a acestor alimente în funcție de cantitățile de nutrienți pe care le oferă și cantitățile recomandate pentru consum pentru un stil de viață sănătos. Pentru a calcula punctajul unui aliment, am definit niște praguri de consum pentru fiecare nutrient în parte, acestea reprezentând valoarea maximă necesară unui adult de sex masculin sau feminin pentru o viață sănătoasă. Mai departe iau valorile actuale pentru alimentul inserat, și calculez diferența dintre valori și praguri. Dacă această diferență depășește 50% din pragul total pe o zi, atunci scorul scade gradual în funcție de cât de mult depășește acest procent. Totodată am definit niște parametrii de greutate pentru fiecare nutrient, astfel încât pentru nutrienții care afectează mai mult organismul dacă depășesc o anumită valoare precum sarea, zahărul, colesterolul, punctajul se va reduce mai mult decât pentru proteine de exemplu. În cazul în care nu se depășește acel procent, punctajul va crește.

Pentru a prelua aceste detalii nutriționale pentru alimentele introduse, am folosit un API RESTful pus la dispoziție de APINinjas. Astfel, în serverul aplicației mele am implementat metode care să comunice cu acest server extern, și prin cereri de tip HTTP GET către un endpoint, la care specificăm cheia API unică în antetul cererii, și un parametru de interogare ce reprezintă alimentul introdus pentru care vrem să primim detaliile nutriționale. Mai departe vom primi un string JSON cu toate aceste valori și le vom trimite mai departe către frontend, unde vom prelucra acest raspuns și îl vom afișa în interfața cu utilizatorul. Dacă alimentul introdus nu există în baza de date a celor de la APINinjas, voi afișa un mesaj că alimentul nu există.

Dacă alimentul există, utilizatorul poate să creeze o înregistrare pentru a-și adăuga kaloriile consumate împreună cu alimentul consumat în baza de date pentru actualizarea parametrilor de nutriție monitorizați și în celelalte componente.

```
[HttpGet("nutrition")]
public async Task<ActionResult<string>> GetNutritionData(string query)
{
    try
    {
        string apiKey = _configuration["NinjasAPIKey:APIKey"];
        _httpClient.DefaultRequestHeaders.Add("X-API-Key", apiKey);
        HttpResponseMessage response = await
        _httpClient.GetAsync($"nutrition?query={query}");
        if (response.IsSuccessStatusCode)
        {
            string result = await response.Content.ReadAsStringAsync();
            return Ok(result);
        }
        else
        {
            return StatusCode((int)response.StatusCode, $"Error:
{response.ReasonPhrase}");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Exception: {ex.Message}");
    }
}
```

Codul 13: Metodă din server pentru returnarea datelor de nutriție pentru un aliment ca și răspuns

3.4.4 Pagina de monitorizare a exercițiilor și activităților

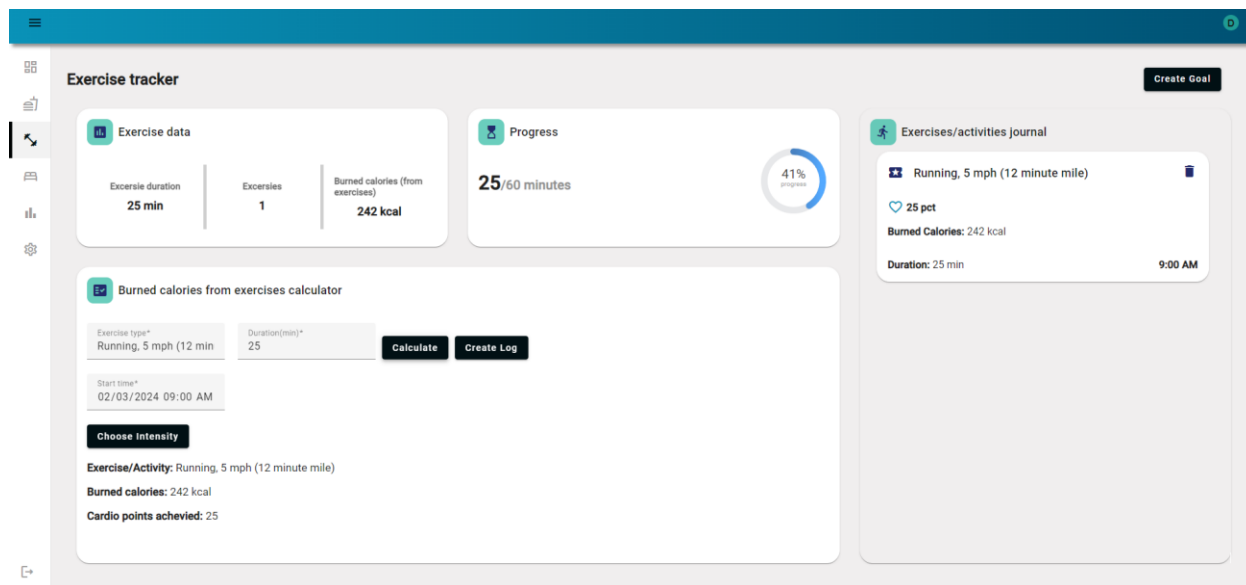


Figura 3.14: Pagina Exercise tracker

Pagina *Exercise tracker* este similară cu pagina *Calories tracker* deoarece pentru a calcula kaloriile arse pentru un exercițiu sau activitate introdusă, am folosit un API tot de la APINinjas. Acest API funcționează asemănător, serverul trimite o cerere tip HTTP GET către endpointul celor de la APINinjas cu cheia și cu un parametru de interogare care conține tipul exercițiului, ca mai apoi să primim un răspuns JSON cu kaloriile arse pe ora pentru acest tip de exercițiu. Totuși, fiindcă în cazul exercițiilor pentru o interogare introdusă puteam primi un răspuns cu mai multe tipuri de exerciții, am implementat o listă de auto-completare dinamică. În momentul în care utilizatorul începe să scrie ceva în câmp, un observabil ascultă schimbările de valoare din acest câmp, preia valoarea actuală și o folosește ca să trimită o cerere către server. Atunci când acest apel HTTP este rezolvat cu succes, datele din răspuns sunt preluate sub forma unui array de obiecte *Exercise*. Apoi, este creat un nou array *exerciseSuggestions*, în care sunt stocate numele exercițiilor, folosind metoda *map* pentru a extrage numele din obiectele *Exercise*. Acest nou array este mai departe folosit pentru lista de auto-completare, lăsând utilizatorului posibilitatea de a alege din variantele existente de exerciții.

```
this.exerciseFormGroup.valueChanges.subscribe((value: string) => {
  if (value) {
    this.getExerciseList(this.exerciseFormGroup.value);
  }
})
```

```

});
getExerciseList(formValue: any) {
  const exerciseType = formValue.exerciseType;
  this.userService.getCaloriesBurned(exerciseType).subscribe(
    (res: Exercise[]) => {
      this.exerciseSuggestions = res.map((exercise: Exercise) =>
exercise.name);
    },
    (error) => {
      console.error("Error fetching data:", error);
    }
  );
}
}

```

Codul 14: Generare listă auto-completare pentru tipuri de exerciții

După alegerea tipului de exercițiu/activitate și durata, utilizatorul poate să aleagă intensitatea. În funcție de această intensitate, se vor calcula punctele cardio obținute. Pentru fiecare minut de activitate care pune inima în mișcare, la o intensitate medie, se obține un punct cardio. După calcularea acestor parametrii, utilizatorul poate crea o înregistrare, generându-se automat și un card care conține detaliile acelei sesiuni de activitate, cât și ora la care s-a efectuat acest antrenament.

3.4.5 Pagina de monitorizare a somnului

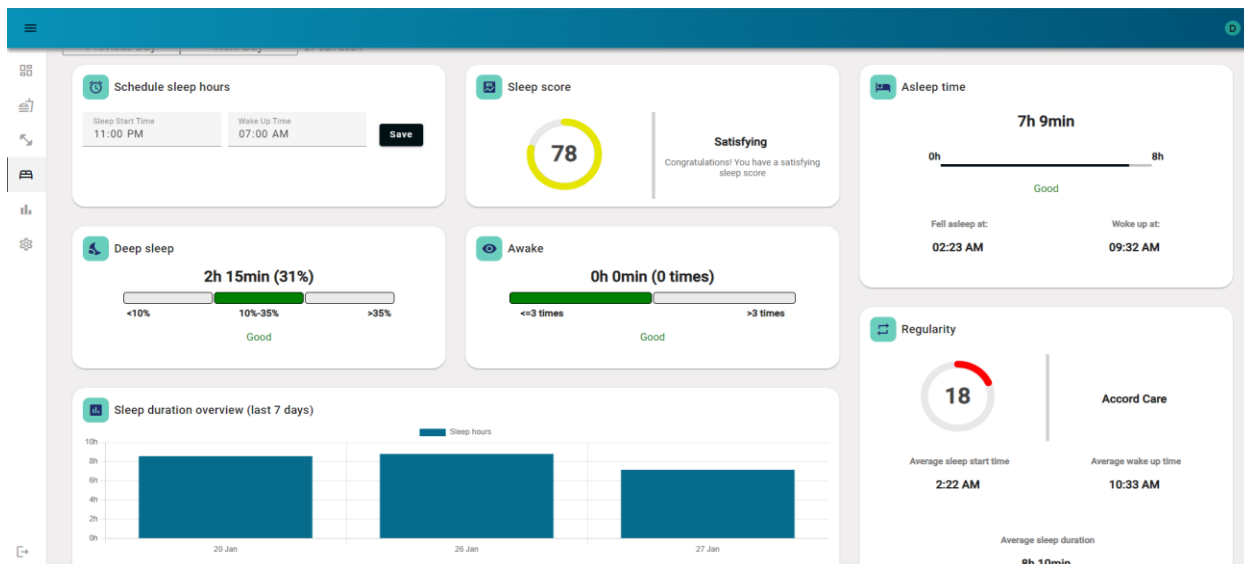


Figura 3.15: Pagina Sleep tracker

Pagina *Sleep tracker* este destinată monitorizării aspectelor principale ale unui somn sănătos cum ar fi durata totală a somnului, durata somnului adânc și procentajul acesta din totalul de somn, durata și numărul de treziri pe care utilizatorul le-a avut în timpul sesiunii de somn, regularitatea orelor de începere a somnului și de trezire.

Pentru a avea acces la aceste date, utilizatorul are nevoie de un smartwatch pe care să îl poarte pe mână în timpul sesiunilor de somn. Totodată acesta trebuie să fie conectat la aplicația mobilă dezvoltată special pentru ceasul lui pentru a salva datele de somn. Mai de parte acesta trebuie să se conecteze cu această aplicație terță parte la Google Fit, care va prelua datele de somn. Mai departe, HealthHub va prelua aceste date de somn prin apelurile HTTP către serverul Google sub forma unor JSON-uri care conțin sesiunile de somn.

```
try
{
    // Constuirea URL-ului pentru apel
    string requestUrl =
    $"https://www.googleapis.com/fitness/v1/users/me/sessions?startTime={startTime}&endTime={endTime}";
    User user = await _userRepository.GetUserByIdAsync(userId);

    if (user == null)
    {
        return NotFound("User not found");
    }
    string accessToken = user.AccessToken;
    string refreshToken = user.RefreshToken;
    var client = _httpClientFactory.CreateClient();
    client.DefaultRequestHeaders.Add("Authorization", $"Bearer {accessToken}");
    client.DefaultRequestHeaders.Add("Accept", "application/json");
    var response = await client.GetAsync(requestUrl);
}
```

Codul 15: Apelul HTTP GET pentru a primi sesiunile de somn dintr-un interval de timp

În variabila *response* vom avea un string JSON care va conține toate sesiunile de somn împachetate ca și obiecte ale unui array *session*. Aceste obiecte conțin atributul *name*, prin care putem să vedem că este vorba despre o sesiune de somn, aceste sesiuni putând să conțină și alte activități. Alte atribute relevante sunt *startTimeMillis* și *endTimeMillis* care reprezintă marca temporală UNIX în milisecunde, fiind data și ora de începere și de încheiere a sesiuni de somn.

Pentru a accesa mai multe detalii despre fiecare sesiune de somn în parte, avem nevoie de un apel HTTP POST către un alt endpoint care ne va întoarce un set de date sub un format asemănător ce reprezintă fazele somnului. Fiecare fază a somnului este reprezentată de o valoare de tip `int`, și este conținută tot într-un obiect ce face parte dintr-un array numit *bucket*. Fiecare obiect are această valoare de tip `int` dar și attributele `startTimeNanos` și `endTimeNanos` care reprezintă intervalul de timp în care acea fază a somnului a avut loc.

Pentru a prelua aceste răspunsuri în frontend pentru a le procesa și a le afișa în interfață am creat servicii speciale pentru fiecare din aceste apeluri, pe care le-am integrat în logica componentelor din interfața cu utilizatorul.

O funcționalitate importantă pe care am implementat-o folosind aceste date a fost sistemul de scor al somnului și sistemul de regularitate și al calculării orelor medii de somn și de trezire pentru ultimele șapte zile. Voi vorbi puțin despre calculul scorului de regularitate și al orelor medii de somn și de trezire.

Pentru a calcula regularitatea, vom avea nevoie de minim trei sesiuni de somn înregistrate în ultimele 7 zile, acesta fiind intervalul pe care îl folosesc în apelul către server pentru a prelua datele despre sesiunile de somn. Dacă avem aceste 3 sesiuni înregistrate vom forma un array numit `SleepData` care va conține obiecte ce reprezintă fiecare sesiune de somn înregistrat cu attributele `startTimeMillis` și `endTimeMillis`. Vom apela mai departe o metodă a serviciului `SleepRegularityService` numită `calculateConsistencyScore`, cu array-ul `SleepData` ca și parametru. În această metodă se fac următorii pași:

- Se sortează datele despre perioadele de somn în funcție de timpul de început al somnului.
- Se calculează diferențele de timp între perioadele consecutive de somn.
- Se calculează deviația standard a diferențelor de timp pentru a evalua gradul de regularitate.
- Se calculează scorul de consistență utilizând o formulă care ține cont de deviația standard și media diferențelor de timp.
- Se aplică un factor de pondere acestui scor de consistență, se înmulțește cu 100 și se rotunjește la un număr întreg.

3.4.6 Pagina de rapoarte grafice

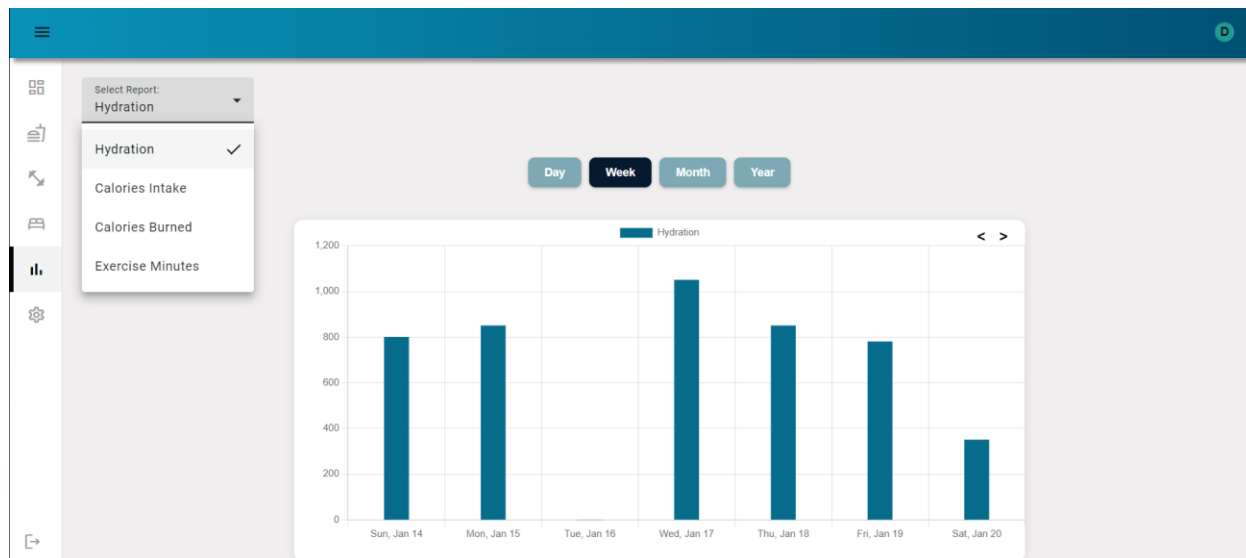


Figura 3.16: Pagina Reports

Pagina de rapoarte conține un grafic care poate afișa date pe diferite intervale de timp precum zile, săptămâni, luni și ani, utilizatorul putând să navigheze prin aceste intervale cu ajutorul butoanelor de deasupra graficului. Tot odată există și o listă în partea stângă de unde se poate alege tipul de informație care va fi afișat pe grafic. Pentru zile, săptămâni și luni există și niște cursoare prin care putem naviga înainte sau înapoi cu o zi pentru intervalul Day, o lună pentru intervalul Week și respectiv un an pentru intervalul Month.

Acest grafic a fost implementat cu ajutorul *ng2-charts* care facilitează integrarea bibliotecii *Chart.js* în aplicațiile Angular, care mi-a permis să creez și să personalizez graficul într-un mod simplu. Pentru a integra aceste biblioteci ajutătoare în aplicația mea am folosit comenzile *npm i ng2-charts* și *npm i chart.js* în terminal, și a trebuit să includ modulele aferente în fișierul *app.module.ts* din proiectul meu.

Pentru a procura înregistrările de care avem nevoie pentru alimentarea graficelor cu date, am folosit serviciul *UserService*. Aici avem definite următoarele metode: *getAllExerciseLogs()*, *getAllNutritionLogs()* și *getAllHydrationLogs()*. Aceste metode din serviciul Angular trimit niște apeluri de tip HTTP GET cu ID-ul utilizatorului ca și parametru de interogare către serverul de backend. Aceste apeluri sunt preluate de către clasele de

control care apelează mai departe metodele Repository aferente, prin care se preiau toate înregistrările făcute de utilizatorul cu ID-ul dat. Aceste date sunt trimise înapoi către frontend sub forma JSON și prelucrate mai departe pentru a le afișa în grafice.

În primul rând setez toate intervalele de timp de care avem nevoie folosind Moment.js. Astfel definesc niște proprietăți ale componentei Reports care conțin datele în diferite formate în funcție de nevoie. Printre proprietățile definite se numără începutul și sfârșitul zilei curente, a săptămânii curente, a lunii și a anului curent.

În metoda ngOnInit() fac un apel către funcția getHydrationLogs() care va returna toate înregistrările de hidratare a utilizatorului, acestea fiind datele default care vor fi afișate în grafic la fiecare inițializare a componentei. Pentru a vedea celelalte tipuri de date cum ar fi calorii arse sau durata exercițiilor, utilizatorul poate să aleagă din meniul din stânga o altă categorie. În momentul alegerii unei alte categorii din acea listă, vom apela celelalte funcții de preluare a datelor din server, în funcție de alegerea făcută. Această schimbare se face prin funcția changeReportType() pe care o apelăm prin directiva (selectionChange) din Angular în fișierul HTML.

```
<div class="report-options">
  <mat-form-field appearance="fill">
    <mat-label>Select Report:</mat-label>
    <mat-select [(value)]="selectedReport"
(selectionChange)="changeReportType()">
      <mat-option value="Hydration">Hydration</mat-option>
      <mat-option value="Calories Intake">Calories Intake</mat-option>
      <mat-option value="Calories Burned">Calories Burned</mat-option>
      <mat-option value="Exercise Minutes">Exercise Minutes</mat-option>
    </mat-select>
  </mat-form-field>
</div>
```

Codul 16: Cod sursă pentru lista de categorii de date pentru afișarea în grafic

Pentru a crea graficul folosesc instrucțiunea new Chart("MyChart" , {}) din cadrul bibliotecii Chart.js, care creează o nouă instanță a obiectului Chart. În interiorul acoladelor se află toate proprietățile și opțiunile de inițializare a graficului, precum tipul, culoarea, mărimea, axele x și y, etichetele cât și setul de date de afișat.

Pentru a formata setul de date în funcție de intervalul de timp selectat de utilizator am implementat funcții de filtrare a datelor în funcție de aceste intervale, funcții de filtrare a etichetelor pentru a reflecta și vizual setul de date și intervalele de timp aferente acestora, și multe alte funcții responsabile de logica, prelucrarea și gruparea setului de date pentru afișarea în grafic.

Metoda `updateChartData`:

- Această metodă actualizează datele pentru grafic în funcție de intervalul selectat (zi, săptămână, lună, an).

Metode pentru navigarea înainte și înapoi (`moveToPreviousWeek`, `moveToNextWeek`):

- Aceste metode actualizează datele pentru afișarea graficului în perioadele anterioare sau ulterioare, în funcție de intervalul selectat.

Metode pentru filtrarea datelor (`filterByDay`, `filterByWeek`, `filterByMonth`, `filterByYear`):

- Aceste metode filtrează datele în funcție de intervalul selectat și grupează apoi datele în funcție de zi, săptămână, lună sau an.

Metoda `groupByDate`:

- Această metodă grupează datele în funcție de intervalul specificat și întoarce un obiect de tip hartă cu chei și valori corespunzătoare.

Alte metode utilizate (`sortChartLabels`, `getISOWeek`, `getWeekNumber`, `aggregateData`, `changeChartInterval`):

- Aceste metode îndeplinesc diverse funcționalități, cum ar fi sortarea etichetelor de pe grafic, obținerea numărului săptămânii, agregarea datelor și schimbarea intervalului graficului.

3.5 TESTARE ȘI VALIDARE

Testarea este un proces prin care se evaluează și se verifică funcționalitățile unei aplicații pentru a identifica eventualele erori, bug-uri sau disfuncționalități. Este un efort meticulos și sistematic de a asigura că software-ul respectă cerințele stabilite și oferă

rezultatele dorite. Testarea acoperă diverse aspecte ale aplicației, inclusiv funcționalitatea, performanța, securitatea și compatibilitatea.

Pentru testarea frontend am folosit testarea manuală, care mi-a permis să observ cu ușurință defectele ce pot apărea, prin analizarea consolei de dezvoltare oferită de browser, prin aplicarea diferitelor scenarii pe care un utilizator le va încerca, prin punerea mea în situația unui utilizator care va folosi aplicația pentru prima oară sau chiar prin efectuarea unor acțiuni repetate pe anumite componente pentru a asigura stabilitatea și viteza de afișare a datelor la inițializare sau consistența datelor afișate în interfața cu utilizatorul. Acest tip de testare a fost continuă pe tot parcursul dezvoltării aplicației. De fiecare dată când adăugam o componentă sau o funcționalitate nouă, o testam intensiv prin aplicarea unor cazuri de uz în repetate rânduri pentru a vedea comportamentul. Am încercat să mă gândesc mereu la cazurile speciale care pot apărea, astfel aplicând scenarii prin care să obțin o acoperire cât mai largă a cazurilor testate.

Tabel 1: Scenarii de test pentru autentificare

Scenarii de test	Combinații	Rezultate așteptate	Evaluare(trecut/picat)
Autentificare	Enunț: Verifică următoarele combinații de username și parolă.		
1.1	blank, blank	Buton Login blocat	trecut
1.2	blank, invalid	Buton Login blocat + Câmpul blank culoare roșie	trecut
1.3	blank, valid	Buton Login blocat + Câmpul blank culoare roșie	trecut
1.4	invalid, blank	Buton Login blocat + Câmpul blank culoare roșie	trecut
1.5	valid, blank	Buton Login blocat + Câmpul blank culoare roșie	trecut

1.6	invalid, invalid	Buton Login deblocat, mesaj de eroare la autentificare	trecut
1.7	invalid, valid	Buton Login deblocat, mesaj de eroare la autentificare	trecut
1.8	valid, invalid	Buton Login deblocat, mesaj de eroare la autentificare	trecut
1.9	valid, valid	Buton Login deblocat, autentificare reușită -> redirectionare către Dashboard	trecut

În acest tabel sunt prezentate cazurile de test pentru autentificarea unui utilizator în aplicație. Acesta este un exemplu de abordare sistematică și concisă de a testa manual funcționalități simple sau complexe, utilizând tabele de cazuri în care putem vedea mult mai clar ce vrem să testăm, combinațiile posibile și care sunt rezultatele acestor înșirui de evenimente în cadrul unei funcționalități ale aplicației.

Pentru testarea backend am folosit tot testarea manuală, dar de data aceasta utilizând un tool ajutor specializat, numit Postman. Postman este un instrument puternic și versatil dedicat dezvoltatorilor de software și echipelor de testare, utilizat pentru testarea și gestionarea interacțiunilor cu API-uri. Acest tool oferă un mediu prietenos și intuitiv pentru a crea, testa și documenta solicitări HTTP, facilitând dezvoltarea și debugging-ul API-urilor.

Aceeași abordare de testare continuă am folosit-o și în cadrul dezvoltării serverului de backend, testând metodele, API-urile, apelurile și comunicarea cu alte servicii în mod sistematic și constant. Cu Postman am putut verifica apelurile HTTP pe care le făceam către endpointurile create în server într-un mod simplu și rapid. Am creat apeluri predefinite de diferite tipuri (GET, POST, DELETE etc.), am adăugat URL-urile endpointurilor furnizate de serverul meu, detaliile apelurilor și informațiile cerute de acestea, și prin o singură apăsare de buton am putut vedea răspunsul furnizat de server, viteza de răspuns, mesajele de eroare, mărimea răspunsurilor și multe alte informații utile.

Astfel am putut testa la fiecare pas, metodele și funcționalitățile dezvoltate în server, având parte de un feedback continuu, rapid și eficient pentru a îmbunătăți și valida funcționalitățile serverului încă din fazele incipiente ale dezvoltării. Totodată, am putut crea utilizatori, înregistrări și am putut comunica cu API-uri fără a avea încă început proiectul de

frontend. Acest lucru a făcut ca dezvoltarea serverului să fie independentă de alți factori cum ar fi stadiul de dezvoltare al interfeței cu utilizatorul.

Un alt aspect pe care l-am testat cu ajutorul acestui instrument a fost viteza de răspuns a apelurilor către server. Pentru că un utilizator poate ajunge să aibă foarte multe înregistrări create, și pentru că în pagini precum tabloul de bord avem multiple apeluri făcute în inițializarea componentei, am căutat să optimizez cât mai mult metodele de a prelua astfel de pachete de date mari pentru a avea un timp de răspuns cât mai rapid.

Conform unui studiu condus de Jakob Nielsen [10], se recomandă ca timpul de răspuns al unei aplicații să fie în jurul valorii de 1.0 secunde pentru a asigura un flux neîntrerupt în gândirea utilizatorului. În vederea menținerii atenției utilizatorului asupra sarcinii, limitele acceptate pentru timpul de răspuns sunt în general sub 10 secunde. Astfel, se sugerează ca operațiile să se desfășoare într-un interval de 1.0 secunde pentru majoritatea acțiunilor, iar pentru operațiile CRUD (Create, Read, Update, Delete) care implică aducerea unei cantități mari de date, limita să fie de 3 secunde. Această optimizare a timpului de răspuns contribuie la îmbunătățirea experienței utilizatorului în cadrul aplicațiilor software.

Pentru a optimiza acești timpi de răspuns, am folosit DTO-uri, prin care am putut aduce în frontend doar datele necesare. Totodată pentru a evita repetarea apelurilor către servere externe cum ar fi APINinjas sau Google API și a nu încărca serverul cu foarte multe apeluri în bucle, am adăugat anumite date preluate din aceste servere în baza de date a aplicației pentru a putea face calcule sau a aplica mult mai rapid interogări pe aceste date. Astfel am reușit să ajung la un timp de răspuns sub 1.0 secunde pentru paginile care conțin cele mai multe apeluri cum ar fi *Dashboard overview* și *Sleep tracker*.

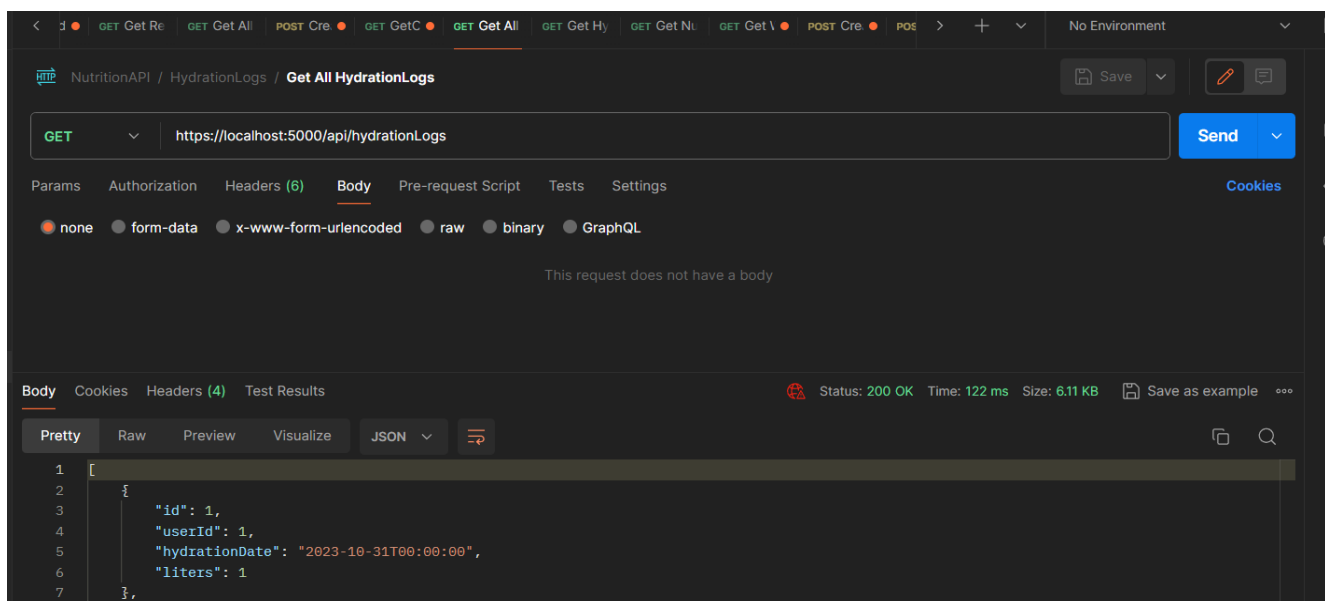


Figura 3.17: Exemplu de testare a unui apel HTTP GET cu Postman

4 CONCLUZII

4.1 REALIZAREA OBIECTIVELOR PROPUSE

Obiectivul principal al acestei lucrări a fost de a crea o aplicație completă și funcțională care să servească utilizatorilor ca o unealtă de monitorizare și centralizare a datelor de nutriție și a parametrilor relevanți pentru abordarea unui stil de viață sănătos. Acest obiectiv a fost îndeplinit, aplicația ajungând într-un stadiu final, stabil și complet funcțional.

Obiectivul dezvoltării unei pagini de autentificare și înregistrare a fost și el atins, aplicația dispunând de aceste pagini care au fost testate și validate ca fiind funcționale.

Obiectivul dezvoltării unei interfețe cu utilizatorul simplă și ușor de folosit a fost îndeplinit din punctul meu de vedere, aplicația având o interfață cu o așezare structurată, uniformă și aerisită.

Obiectivul implementării unui design responsive pentru utilizarea pe diferite dispozitive de diferite mărimi a fost îndeplinit și el, aplicația având un design care se poate adapta la diferite mărimi pentru dispozitive precum tablete, telefoane, laptop-uri, PC-uri.

Obiectivele creării unui tablou de bord, unei pagini de monitorizare a caloriilor, exercițiilor, a somnului, unei pagini de rapoarte grafice și a unei pagini de profil au fost toate îndeplinite, aplicația dispunând de astfel de pagini și de toate funcționalitățile propuse de acestea.

Obiectivele de a integra cel puțin un API extern în aplicația mea a fost și el îndeplinit, având nu unul, ci trei API-uri utilizate în serverul aplicației. Este vorba de două API-uri folosite pentru a calcula caloriile arse de anumite exerciții sau activități pe care utilizatorul le introduce, și pentru a calcula numărul de calorii și de nutrienți conținute de alimentele introduse de utilizator. Cel de al treilea API este API pus la dispoziție de Google pentru a comunica și a prelua date din aplicația mobilă Google Fit. Prin acest API am îndeplinit și obiectivul de a integra un API prin care să pot prelua date de fitness relevante de la dispozitive mobile sau smartwatch-uri.

Obiectivul de encapsulare a întregii aplicații într-o aplicație de mobil nu a fost îndeplinit. Deși am urmărit dezvoltarea unui design responsive pentru a putea face această tranziție mai

ușoară. Timpul nu mi-a permis să fac și acest pas, pe care îl voi considera ca o dezvoltare ulterioară necesară pentru viitor.

4.2 DEZVOLTĂRI ULTERIOARE

O primă dezvoltare care poate fi făcută, este integrarea sau migrarea aplicației într-o aplicație nativă mobile, care ar oferi o performanță și calitate ridicată pe dispozitivele mobile. Acest lucru este important deoarece aplicațiile de fitness și nutriție au o relevanță mare pentru acest tip de dispozitive.

O altă dezvoltare ulterioară ce poate fi adusă este de a implementa niște algoritmi de recomandare, pentru a recomanda utilizatorului mâncăruri sănătoase în funcție de preferințele și alegerile lor, exerciții pentru a arde caloriile consumate provenite din anumite alimente. Tot odată o dezvoltare ce poate fi adusă este implementarea notificărilor pentru diferite funcționalități cum ar fi hidratare, ora de culcare, ora de mișcare sau activitate fizică.

Pentru a îmbunătății și extinde aplicația și mai mult se mai pot adăuga parametrii pentru monitorizare cum ar fi pulsul sau alți parametrii relevanți pentru sănătatea utilizatorului.

Crearea de planuri personalizate de slăbire sau îngrășare pe care mai apoi utilizatorul să le poată urma, notificări în acest sens și poate utilizarea AI-ului pentru implementarea unor algoritmi de aproximare a timpului necesar pentru a atinge anumite scopuri sau obiective setate de utilizator.

5 BIBLIOGRAFIE

- [1] „Geek for geeks,” [Interactiv]. Available: <https://www.geeksforgeeks.org/what-is-an-api/>.
- [2] A. H. D. N. J. R.-J. A. S. D. R. B. W. Kellyn Gorman, *Introducing Microsoft SQL Server 2019*, Packt Publishing, 2019.
- [3] D. Petkovik, *Microsoft SQL Server 2019: A Beginner's Guide*, 2020.
- [4] „CDN Blog,” [Interactiv]. Available: <https://cdn-blog.scalablepath.com/uploads/2022/12/spa-va-mpa-key-features-1024x800.png>.
- [5] „Ionos,” [Interactiv]. Available: <https://www.ionos.ca/digitalguide/hosting/technical-matters/what-is-http/>.
- [6] „Auth0,” [Interactiv]. Available: <https://auth0.com/learn/json-web-tokens>.
- [7] [Interactiv]. Available: <https://uthpalasandeepa.medium.com/database-first-to-code-first-ef-core-2-0-40e74e980cbf>.
- [8] [Interactiv]. Available: https://i.ytimg.com/vi/_fxrIdX1GIQ/maxresdefault.jpg.
- [9] „Javatpoint,” [Interactiv]. Available: <https://www.javatpoint.com/angular-8-architecture>.
- [10] J. Nielsen, *Usability Engineering*, San Francisco: AP Professional, 1993.

LISTA DE FIGURI, TABELE ȘI CODURI SURSĂ

FIGURI

Figura 2.8: Interfața de programare a aplicațiilor [1]	10
Figura 2.13: Single Page Application vs Multiple Page Application [4]	14
Figura 2.14: Comunicarea HTTP [5]	15
Figura 2.16: Modul de funcționare al JWT [6]	16
Figura 2.17: Code First vs Database First [7]	17
Figura 3.1: Arhitectura aplicației [8]	19
Figura 3.2: Diagrama de cazuri	20
Figura 3.3: Diagrama de clase	21
Figura 3.4: Diagrama claselor de control responsabile de comunicarea cu servere externe prin REST API.	23
Figura 3.5: Diagrama claselor de control asociate cu clasele Repository	24
Figura 3.6: Structura fișierelor în server	25
Figura 3.7: Atributul ApiController	27
Figura 3.8: Exemplu de răspuns JSON	27
Figura 3.9: Exemplu de Entitate	28
Figura 3.10: Arhitectura Angular [9]	31
Figura 3.11: Pagina de autentificare	35
Figura 3.12: Pagina Dashboard overview	38
Figura 3.13: Pagina Calories tracker	42

Figura 3.14: Pagina Exercise tracker	45
Figura 3.15: Pagina Sleep tracker	46
Figura 3.16: Pagina Reports.....	49
Figura 3.17: Exemplu de testare a unui apel HTTP GET cu Postman	54

TABELE

Tabel 1: Scenarii de test pentru autentificare.....	52
-----------------------------------------------------	----

CODURI SURSĂ

Codul 1: Metodă din cadrul clasei de control NutritionLogsController	27
Codul 2: Exemplu de clasă Repository	29
Codul 3: Formular de autentificare și înregistrare în HTML.....	32
Codul 4: Secțiune de cod cu câmpuri care conțin directiva mat-form-field	32
Codul 5: Cod sursă pentru Componenta LiveClock	33
Codul 6: Exemplificarea unui constructor de componentă în Angular.....	34
Codul 7: Cod sursă pentru exemplificarea unui serviciu din cadrul aplicației	34
Codul 8: Metoda Register din clasa AccountController	36
Codul 9: Metoda Login din clasa AccountController.....	37
Codul 10: Serviciu pentru luarea cantității de apă consumate într-un interval de timp.....	40
Codul 11: Serviciu pentru preluarea caloriilor arse de metabolism din Google Fit	40
Codul 12: Obținerea token-urilor de acces de la Google.....	42
Codul 13: Metodă din server pentru returnarea datelor de nutriție pentru un aliment ca și răspuns	44
Codul 14: Generare listă auto-completare pentru tipuri de exerciții.....	46
Codul 15: Apelul HTTP GET pentru a primi sesiunile de somn dintr-un interval de timp.....	47
Codul 16: Cod sursă pentru lista de categorii de date pentru afișarea în grafic.....	50

REZUMAT

În această lucrare de licență am prezentat în detaliu dezvoltarea unei aplicații de monitorizare a nutriției și a sănătății fizice, inspirându-mă și documentându-mă din aplicațiile și sistemele deja existente din această arie. Am integrat astfel funcționalități relevante pentru a ajuta utilizatorii de a aborda un stil de viață mai sănătos într-un mod mai simplu și mai plăcut.

Aplicația a fost dezvoltată folosind cadre de lucru și unelte care ajută procesul de dezvoltare și îl face mai eficient. Pentru partea de server s-a folosit ASP .NET Web Core API împreună cu Postman pentru testarea apelurilor HTTP și SQL Server pentru a crea baza de date. Pentru frontend am folosit Angular versiunea 16.2.9.

Aplicația conține următoarele pagini: Pagina de autentificare și înregistrare, Pagina tabloului de bord, Pagina de monitorizare a caloriilor, Pagina de monitorizare a exercițiilor și activităților, Pagina de monitorizare a somnului, Pagina de rapoarte grafice și Pagina de profil. Aceste pagini oferă utilizatorului o interfață intuitivă și ușor de folosit, care conține funcționalități utile pentru a ajuta și a motiva utilizatorul în a urma un stil de viață mai sănătos, prin monitorizarea datelor precum caloriile și nutrienți consumați din alimente, caloriile arse din exerciții, parametrii de somn, minutele de activitate și numărul de pași efectuați. Pentru toate aceste date poate fi vizualizat și progresul în funcție de obiectivele setate de utilizator. Aceste date pot fi observate și într-un raport grafic, care poate afișa numerele datelor înregistrate pe intervale de zile, săptămâni, luni și chiar ani.

Realizarea acestui proiect a fost motivată în primul rând de dorința mea de a urma un stil de viață mai sănătos, simțind nevoia de o unealtă care să mă ajute să monitorizez aspectele importante ale unui asemenea stil de viață. Acest aspect este foarte important mai ales în vremurile pe care le trăim deoarece obezitatea, bolile de inimă și alte complicații de sănătate sunt tot mai des întâlnite datorită sedentarismului și a unui stil de viață nesănătos și haotic.

ABSTRACT

In this thesis I have presented in detail the development of a nutrition and physical health monitoring application, drawing inspiration and documentation from existing applications and systems in this area. I have thus integrated relevant functionalities to help users to approach a healthier lifestyle in a simpler and more enjoyable way.

The app was developed using frameworks and tools that help the development process and make it more efficient. For the server side, ASP.NET Web Core API was used together with Postman for testing HTTP calls and SQL Server to create the database. For the frontend we used Angular version 16.2.9.

The application contains the following pages: Login and registration page, Dashboard page, Calorie monitoring page, Exercise and activity monitoring page, Sleep monitoring page, Graph reports page and Profile page.

These pages provide the user with an intuitive and easy-to-use interface that contains useful functionality to help and motivate the user to follow a healthier lifestyle by tracking data such as calories and nutrients consumed from food, calories burned from exercise, sleep parameters, minutes of activity and number of steps taken. For all these data, progress can also be viewed according to the goals set by the user. This data can also be seen in a graphical report, which can display the numbers of data recorded over days, weeks, months and even years.

The project was motivated primarily by my desire to follow a healthier lifestyle, and I felt the need for a tool to help me monitor the important aspects of such a lifestyle. This is especially important in these times because obesity, heart disease and other health complications are increasingly common due to sedentarism and unhealthy and chaotic lifestyles.