

**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 - Health Hub**

**Conducător științific:**

Conf. Dr. Ing. Popescu Vlad

Brașov, 2024

|  |  |
| --- | --- |
| **FIȘA PROIECTULUI DE DIPLOMĂ** | |
| Universitatea Transilvania din Braşov | Proiect de diplomă nr. .......... |
| Facultatea de Inginerie Electrică şi Ştiința Calculatoarelor |
| Departamentul de ............................................................................... | Viza facultății |
| Programul de studii:  **..................................................................** | Anul universitar:  2023 – 2024 |
| Candidat:  **Numele si Prenumele** | Promoția:  2024 |
| Conducător ştiințific:  **Titulatura. Numele și prenumele** |  |
| **PROIECT DE DIPLOMĂ** | |
| Titlul lucrării*: .................................................* | |
| Problemele principale tratate: | |
| Locul şi durata practicii: | |
| Bibliografie:   1. .......................... 2. .......................... 3. ........................... | |
| Aspecte particulare:  - ............................................................;  - ...........................................................  (desene, aplicații practice, metode specifice etc.) | |
| Primit tema la data de: .............................................. | |
| Data predării lucrării: ................................................. | |
| Director departament, Conducător științific,  *Titulatura. Numele și prenumele Titulatura. Numele și prenumele* | |
| Candidat,  *Numele și prenumele* | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PROIECT DE DIPLOMĂ – VIZE** | | | | | |
| Data vizei | Capitole/ problemele analizate | | | | Semnătura conducătorului științific |
|  | ..... | | | |  |
|  | ..... | | | |  |
|  | ..... | | | |  |
|  | ...... | | | |  |
|  | **Verificare procentaj similaritate Turnitin (total ≤25%și o singura sursa ≤10%)** | | | |  |
| **APRECIEREA ŞI AVIZUL CONDUCĂTORULUI ȘTIINȚIFIC** | | | | | |
| (*aprecierea lucrării/ proiectului se face prin raportare la cerințele privind elaborarea şi redactarea stabilite pe PS/ facultate; pot fi utilizate instrumente de evaluare, grile cu criterii unitare de apreciere adoptate de facultate/ departament*) | | | | | |
| Data:  ................................ | | ADMIS pentru susținere/ RESPINS | | Conducător științific  *Titulatura. Numele și prenumele*  *...............................(semnatura)* | |
| **AVIZUL DIRECTORULUI DE DEPARTAMENT** | | | | | |
| Data:  ................................ | | ADMIS pentru susținere/ RESPINS | | Director departament,  *Titulatura. Numele și prenumele*  *..............................(semnatura)* | |
| **SUSȚINEREA PROIECTULUI DE DIPLOMĂ** | | | | | |
| Sesiunea: ................................. | | | | | |
| Rezultatul  susținerii | | | PROMOVAT cu media: | | |
| RESPINS **cu** refacerea lucrării | | |
| RESPINS **fără** refacerea lucrării | | |
| Președinte de comisie,  *Titulatura. Numele și prenumele*  *.................................(semnatura)* | | | | | |

F04-PS7.6-01/ed.2,rev.2

**Cuprins**

Cuprins

[Lista de acronime 6](#_Toc157462202)

[1 Introducere 7](#_Toc157462203)

[1.1 Tema proiectului 7](#_Toc157462204)

[1.2 Scopul aplicației 7](#_Toc157462205)

[1.3 Obiective principale 7](#_Toc157462206)

[1.3.1 Obiective software 7](#_Toc157462207)

[1.3.2 Obiective de documentare 7](#_Toc157462208)

[1.4 Structura lucrării 7](#_Toc157462209)

[2 Tehnologii și metodologie 7](#_Toc157462210)

[2.1 Cadre de lucru 7](#_Toc157462211)

[2.1.1 Angular 7](#_Toc157462212)

[2.1.2 ASP .NET Core Web API 11](#_Toc157462213)

[2.1.3 Microsoft SQL Server 14](#_Toc157462214)

[2.2 Limbaje de programare 16](#_Toc157462215)

[2.2.1 TypeScript 16](#_Toc157462216)

[2.2.2 C# 18](#_Toc157462217)

[2.3 Medii de programare 20](#_Toc157462218)

[2.3.1 Visual Studio Code 20](#_Toc157462219)

[2.3.2 Visual studio 2022 20](#_Toc157462220)

[2.4 Tactici și tehnici de dezvoltare 21](#_Toc157462221)

[2.4.1 Single page application 21](#_Toc157462222)

[2.4.2 Comunicarea HTTP 22](#_Toc157462223)

[2.4.3 JSON Web Token 24](#_Toc157462224)

[2.4.4 Metoda Code First 26](#_Toc157462225)

[2.4.5 Repository Pattern 27](#_Toc157462226)

[3 Structura și dezvoltarea aplicației 28](#_Toc157462227)

[3.1 Arhitectura aplicației – diagrame uml 29](#_Toc157462228)

[3.2 Arhitectura serverului 35](#_Toc157462229)

[3.2.1 Nivelul de prezentare 36](#_Toc157462230)

[3.2.2 Nivelul de domeniu 38](#_Toc157462231)

[3.2.3 Nivelul de infrastructură sau persistență 39](#_Toc157462232)

[3.2.4 Nivelul de interfețe sau adaptoare 40](#_Toc157462233)

[3.3 Arhitectura interfeței cu utilizatorul 41](#_Toc157462234)

[3.3.1 Module 41](#_Toc157462235)

[3.3.2 Directive și componente 42](#_Toc157462236)

[3.3.3 Injectarea dependințelor 43](#_Toc157462237)

[3.3.4 Servicii 44](#_Toc157462238)

[3.4 Elemente de Implementare 45](#_Toc157462239)

[3.4.1 Pagina de autentificare 45](#_Toc157462240)

[3.4.2 Pagina tabloului de bord 48](#_Toc157462241)

[3.4.3 Pagina de monitorizare a caloriilor 52](#_Toc157462242)

[3.4.4 Pagina de monitorizare a exercițiilor și activitățiilor 54](#_Toc157462243)

[3.4.5 Pagina de monitorizare a somnului 56](#_Toc157462244)

[3.4.6 Pagina de rapoarte grafice 59](#_Toc157462245)

[3.5 Testare și validare 60](#_Toc157462246)

[4 Concluzii 60](#_Toc157462247)

[5 Bibliografie 61](#_Toc157462248)

[Lista de figuri, tabele și coduri sursă 62](#_Toc157462249)

[Rezumat 64](#_Toc157462250)

[Abstract 64](#_Toc157462251)

# Lista de acronime

# Introducere

## Tema proiectului

## Scopul aplicației

## Obiective principale

### Obiective software

### Obiective de documentare

## Structura lucrării

# Tehnologii și metodologie

## Cadre de lucru

### 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.

Pentru a profita de capacitățile Angular, o înțelegere de bază a programării web în JavaScript, CSS și HTML este necesară. Fără aceste abilități fundamentale, navigarea în complexitățile dezvoltării aplicațiilor web cu Angular poate prezenta provocări semnificative. În secțiunile următoare, voi prezenta pașii necesari pentru a configura Angular pe computerul personal.

Înainte de a începe instalarea Angular, este important să avem urmatoarele instrumente instalate:

* **Node.js**: Un mediu de execuție introdus în 2009 de Ryan Dahl, folosind JavaScript. Este adoptat pe scară largă de o comunitate vastă de programatori pentru dezvoltarea serverelor de aplicații web.
* **npm (Node Package Manager):** Un instrument folosit de programatori pentru a descoperi, descărca, instala și utiliza module în mod efficient și rapid.

Următorul pas implică deschiderea unei ferestre de terminal și executarea comenzii pentru a instala Angular CLI:



Figura 2.1: Comandă instalare Angular CLI

Odată ce aceste componente fundamentale sunt în loc, inițierea unui nou proiect se realizează foarte ușor cu următoarea comandă:



Figura 2.2: Comandă creare proiect Angular

Un aspect crucial al Angular este accentul pus pe componente, servind drept blocuri fundamentale pentru orice aplicație Angular și fiind instrumentale în crearea interfeței utilizatorului. Componentele gestionează și prezintă datele în mod eficient, fiind responsabile de interacțiunea cu utilizatorului. O componentă Angular constă în trei fișiere esențiale: un fișier TypeScript care reprezintă o clasă ce incapsulează logica și proprietățile specifice acelei clase, un fișier de șablon HTML unde putem folosi aceste proprietăți pentru a le afișa în interfața cu utilizatorul, și un fișier CSS pentru stilizare.

Două avantaje semnificative evidențiază utilizarea răspândită și necesitatea componentelor: modularitatea și reutilizarea. Componentele permit programatorilor să le utilizeze împreună cu alte componente, facilitând crearea unei interfețe mai intricate și estetic plăcută. Componentele au capacitatea de a reacționa la evenimente, cum ar fi apăsarea unei taste pe tastatură sau un clic de mouse, declanșând acțiuni conexe.

Angular simplifică generarea de componente fără conținut cu o singură comandă, traducându-se în economii semnificative de timp. Comanda pentru generarea unei componente în Angular este formulată astfel:

Un aspect crucial al Angular este accentul pus pe componente, servind drept blocuri fundamentale pentru orice aplicație Angular și fiind instrumentale în crearea interfeței utilizatorului. Componentele gestionează eficient și prezintă datele, asumându-și responsabilitatea pentru interacțiunea utilizatorului. O componentă Angular constă în trei fișiere esențiale: un fișier TypeScript care găzduiește logica, un fișier de șablon HTML și un fișier CSS pentru stilizare.

Două avantaje semnificative justifică utilizarea răspândită și necesitatea componentelor: modularitatea și reutilizarea. Componentele permit programatorilor să le utilizeze împreună cu alte componente, facilitând crearea unei interfețe mai complexe. Componentele au capacitatea de a reacționa la evenimente, cum ar fi apăsarea unei taste pe tastatură sau un click de mouse, declanșând acțiuni conexe.

Angular simplifică generarea de componente fără conținut cu o singură comandă, arătând încă odată avantajele oferite de un cadru de lucru în termeni de eficaciatate și fluiditate a procesului de dezvoltare. Comanda pentru generarea unei componente în Angular este formulată astfel:



Figura 2.3: Comandă generare componentă Angular

Mai departe, putem vedea mai jos imagini cu conținutul unei componente în Angular:

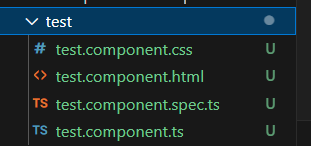


Figura 2.4: Structura unei componente Angular



Figura 2.5: Fișier HTML dintr-o componentă Angular



Figura 2.6: Fișier TypeScript dintr-o componentă Angular

Până în acest moment, am explorat posibilitățile oferite de Angular și beneficiile asociate utilizării acestei tehnologii în dezvoltarea unei aplicații. Cu toate acestea, detaliile anterioare referitoare la componente și comenzi reprezintă doar punctul de plecare. Deși acestea constituie un fundament esențial pentru crearea unei aplicații web, Angular oferă mai multe resurse pentru a eficientiza și accelera procesul de dezvoltare, iar printre acestea se numără Angular Material. 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. Similar cu Angular, Angular Material este supus unor actualizări regulate, la intervale de câteva luni, impunând necesitatea de a fi la curent cu versiunile disponibile. Aplicațiile construite cu Angular Material se pot adapta cu ușurință pe diverse platforme, inclusiv Android, iOS, Windows și MacOS, datorită versatilității tehnologiei. Pentru a beneficia de Angular Material în cadrul proiectului meu, a fost suficient să execut următoarea comandă în terminalul corespunzător proiectului și să selectez opțiunile oferite:



Figura 2.7: Comandă adaugare Angular Material în proiect

### ASP .NET Core Web API

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

Înainte de a merge mai departe și să intru în detalii legate de acest cadru de lucru ar fi mai bine să explic ce este o aplicație de tip Web API mai întâi. 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 primeste cereri HTTP, cum ar fi GET, POST, PUT, DELETE, de la clienti și returnează răspunsuri HTTP care conțin datele solicitate.

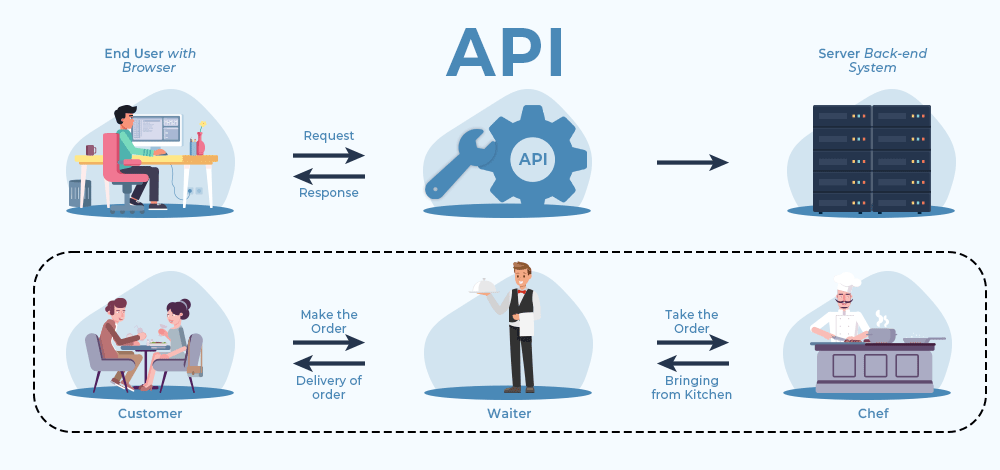


Figura 2.8: Interfața de programare a aplicațiilor

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.

Unul dintre punctele forte ale acestui cadru de lucru este abilitatea sa de a returna date într-un format standardizat, cum ar fi JSON sau XML , oferind astfel o interacțiune eficientă între server și client. Caracteristica centrală a framework-ul de dezvoltare ASP.NET este reprezentată de suportul său pentru arhitectura REST . Aceasta implică utilizarea standardului HTTP și a metodelor sale (GET, POST, PUT, DELETE) pentru a crea, citi, actualiza și șterge resursele expuse prin intermediul API-ului. Prin adoptarea unei abordări bazate pe resurse și folosind conceptele de identificator al resursei (URI ) și de reprezentare a resursei, pot proiecta și implementa servicii web scalabile și ușor de utilizat.

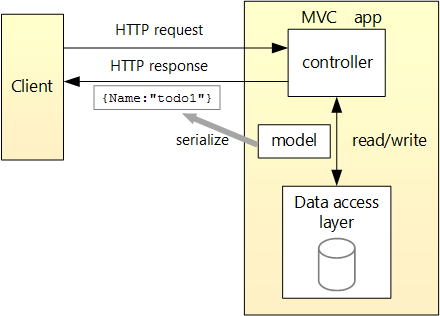


Figura 2.9: Modul de funcționare al ASP .NET Core Web API

Cadrul Web ASP.NET Core facilitează, de asemenea, crearea de servicii web securizate, prin integrarea cu mecanisme precum autentificarea și autorizarea. Prin utilizarea sistemului de autentificare și autorizare bazat pe token-uri, pot proteja resursele expuse prin API și pot controla accesul utilizatorilor sau aplicațiilor la acestea.

Un alt aspect important al acestui cadru de lucru este flexibilitatea sa. Acesta poate fi utilizat împreună cu diverse tehnologii și biblioteci, cum ar fi Entity Framework pentru accesul la baze de date, biblioteca Newtonsoft.Json pentru serializarea și deserializarea datelor în format JSON, și multe altele. De asemenea, este compatibil cu diferite formate de date, cum ar fi XML și JSON, permițând astfel integrarea cu o varietate de clienți și servicii.

Platforma Web API .NET Core pune la dispoziție, de asemenea, un set bogat de instrumente pentru testare și depanare. Prin intermediul unor instrumente precum Postman sau Swagger, pot testa și interoga serviciile API, pot crea documentație automată și pot monitoriza traficul și performanța acestora.

### 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.

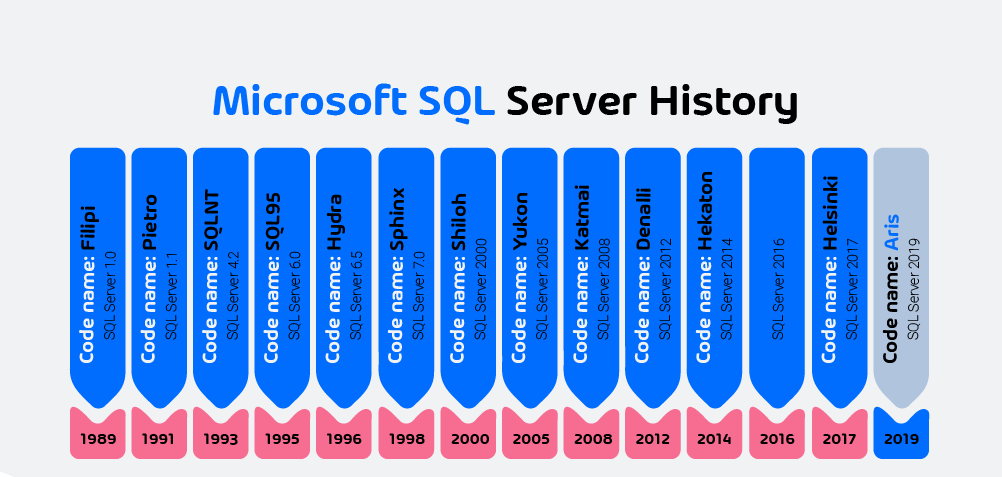


Figura 2.10: Istoria versiunilor Microsoft SQL Server

În etapa inițială, SQL Server a fost conceput în decursul anilor '80 de către Sybase Inc., iar în prezent, este sub administrarea și dezvoltarea continuă a Microsoft. De-a lungul decadelor, SQL Server a suferit un proces de evoluție și îmbunătățire constantă, transformându-se într-un produs solid și de încredere pentru gestionarea datelor în diferite contexte de afaceri. „Securitatea este esențială în orice sistem de stocare și procesare a datelor, iar SQL Server s-a mândrit că este cea mai securizată bază de date în ultimii opt ani, conform bazei de date a Institutului Național de Standarde și Tehnologie(NIST). 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.” [1]

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.

Prin intermediul acestor funcționalități de securitate, SQL Server asigură protecția datelor și respectarea standardelor și cerințelor de securitate în medii enterprise. Aceasta face din SQL Server o alegere potrivită pentru aplicații și sisteme care necesită un nivel înalt de securitate și conformitate.

În lumea actuală, gestionarea eficientă a datelor este esențială pentru succesul organizațiilor de orice dimensiune. O platformă robustă și cu un set extins de funcționalități reprezintă elementul esențial pentru a garanta stocarea, manipularea și administrarea eficientă a datelor. Unul dintre aceste sisteme este Microsoft SQL Server.

SQL Server, produsul dezvoltat de către Microsoft, furnizează o platformă solidă pentru eficientizarea și securizarea gestionării datelor. Este recunoscut pentru caracteristicile sale avansate de securitate, opțiunile de scalabilitate și suportul pentru disponibilitatea înaltă și recuperarea în caz de dezastru. Aceste calități fac din SQL Server o alegere populară pentru organizații de diferite dimensiuni, de la întreprinderi mici până la corporații mari.

Cartea „Microsoft SQL Server 2019: A Beginner's Guide” de Dusan Petkovic [2], 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.” [2]

## Limbaje de programare

### 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.

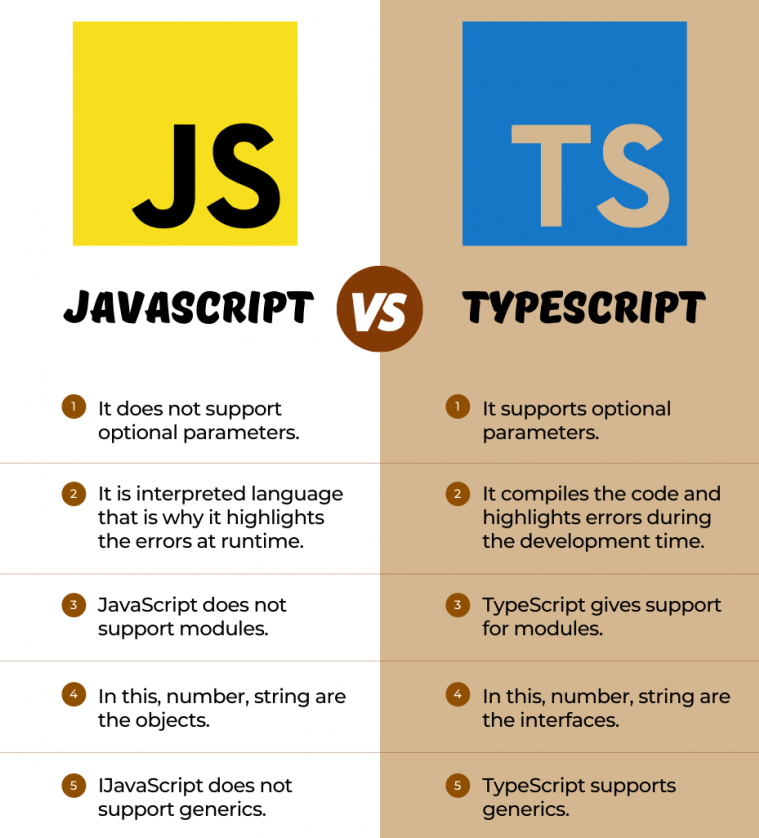


Figura 2.11: JavaScript vs TypeScript

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.

TypeScript aduce o serie de funcționalități suplimentare față de JavaScript, cum ar fi tipurile de date avansate(inclusiv tipuri personalizate), interfețe, clase, moștenire, module și multe altele. Aceste caracteristici îmbunătățesc productivitatea dezvoltatorului, oferind suport pentru dezvoltarea de aplicații scalabile și ușor de întreținut.

Un alt aspect important al TypeScript este că acesta este transpilat în JavaScript standard, ceea ce înseamnă că codul TypeScript poate fi rulat pe orice browser sau platformă care suportă JavaScript.

TypeScript a devenit tot mai popular în comunitatea dezvoltatorilor datorită beneficiilor pe care le aduce în dezvoltarea aplicațiilor web și a aplicațiilor pe partea de server. Este folosit în special în cadrul cadrelor de lucru și bibliotecilor populare, precum Angular, React și Vue.js.

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

### 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.

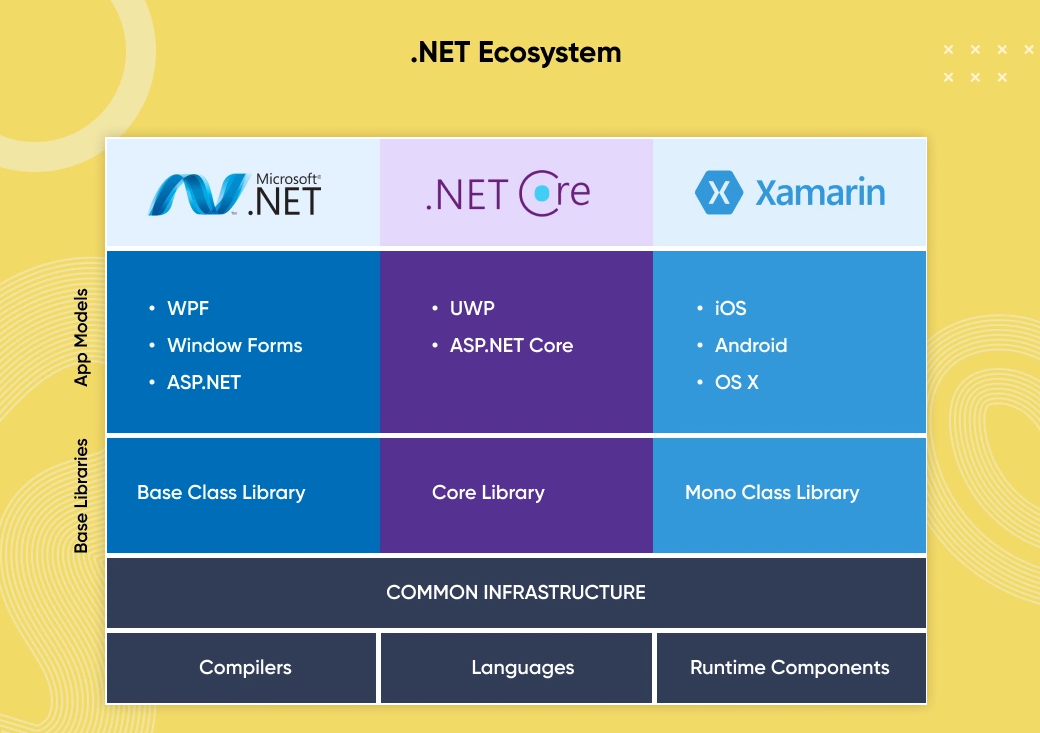


Figura 2.12: Ecosistemul .NET

Una dintre caracteristicile-cheie ale limbajului C# este orientarea sa pe obiecte. Acesta imi 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.

Un alt aspect important al limbajului C# este tipizarea sa statică. Acest lucru înseamnă că toate variabilele și expresiile trebuie să fie declarate cu un tip de date specific înainte de a fi utilizate și că tipurile nu pot fi schimbate ulterior. Utilizarea tipizării statice aduce avantaje, inclusiv identificarea mai eficientă a erorilor de tip în faza de compilare și contribuie la creșterea siguranței și clarității în cod. C# oferă, de asemenea, suport puternic pentru programarea asincronă. Acest lucru este deosebit de util în aplicațiile care trebuie să răspundă la evenimente și să desfășoare operații de lungă durată fără a bloca firul principal de execuție. Prin utilizarea cuvântului cheie "async" și a bibliotecii de tipuri "Task", pot crea operații asincrone și pot beneficia de o performanță sporită în aplicațiile lor.

C# este, de asemenea, strâns integrat cu platforma .NET, ceea ce înseamnă că pot avea acces la o gamă largă de biblioteci și cadre de lucru pentru dezvoltarea aplicațiilor. Aceasta include .NET Framework, care este utilizat pentru dezvoltarea de aplicații Windows, precum și .NET Core, care este utilizat pentru dezvoltarea de aplicații cross-platform. Există și cadre de lucru specifice domeniului, cum ar fi ASP.NET pentru dezvoltarea aplicațiilor web și Xamarin pentru dezvoltarea aplicațiilor mobile.

C# este utilizat într-o varietate de domenii, printre care dezvoltarea de aplicații mobile, aplicații web și desktop, jocuri, sisteme înglobate etc. Comunitatea dezvoltatorilor C# este activă și oferă suport și resurse pentru îmbunătățirea cunoștințelor și abilităților în acest limbaj.

Cu toate acestea, C# este un limbaj în continuă evoluție, iar Microsoft lansează în mod regulat noi versiuni și actualizări pentru a adăuga funcționalități noi și pentru a îmbunătăți performanța și productivitatea dezvoltatorilor. Prin urmare, rămâne un limbaj viu și actual în industria software, oferind posibilități nelimitate pentru crearea de aplicații moderne și inovatoare.

## Medii de programare

### 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.

Un alt instrument care mi-a fost de mare ajutor în dezvoltarea aplicației este GitHub Copilot. Acesta este un sistem de inteligență artificială dezvoltat de GitHub și OpenAI, care oferă sugestii și completări de cod bazate pe contextul și nevoile mele. Cu ajutorul GitHub Copilot, am putut scrie cod mai rapid și mai eficient, beneficiind de recomandări inteligente și adaptate specificului proiectului meu.

### 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.

## Tactici și tehnici de dezvoltare

### 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 sugestiva pentru comparația dintre SPA și MPA. În această figura putem observa diferențele la nivel de performanță, mentenanță, securitate, cost, scalabilitate etc.

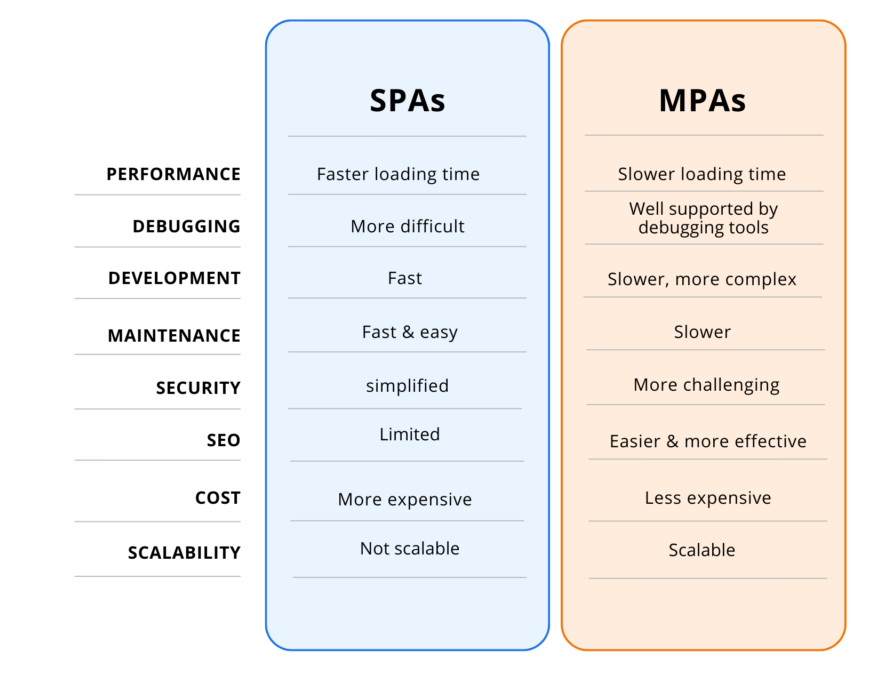


Figura 2.13: Single Page Application vs Multiple Page Application

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.

O altă caracteristică importantă a SPA este reutilizarea codului și separarea clară între frontend și backend. SPA-urile utilizează API-uri pentru a comunica cu serviciile web și pentru a obține sau a trimite date. Aceasta permite dezvoltatorilor să creeze și să mențină logică de afișare și de interacțiune în frontend, în timp ce serverul se ocupă de prelucrarea datelor și de logica de afaceri în backend.

Pentru a realiza o SPA, se utilizează de obicei un cadru de lucru de dezvoltare a frontend-ului, cum ar fi Angular, React sau Vue.js. Aceste cadre oferă un set de instrumente și biblioteci care facilitează gestionarea stării aplicației, rutarea, manipularea DOM-ului și comunicarea cu serviciile web.

Cu toate acestea, există și câteva aspecte de luat în considerare atunci când se dezvoltă o SPA. Deoarece întreaga aplicație este încărcată o singură dată, inițializarea inițială a aplicației poate necesita mai mult timp, iar gestionarea memoriei și a resurselor trebuie abordată cu atenție pentru a evita problemele de performanță.

### 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: Această metodă este folosită pentru a prelua o resursă specifică de la server. De exemplu, un utilizator poate efectua o cerere de GET pentru a obține informații despre un produs particular de pe un site de comerț electronic.
* POST: Metoda POST este utilizată pentru a transmite date către server în scopul creării unei noi resurse. De exemplu, atunci când un vizitator completează un formular de înregistrare pe un site web, informațiile sunt trimise prin intermediul unei cereri de Î pentru a crea un cont nou.
* PUT: Metoda PUT este utilizată pentru a actualiza o resursă existentă sau pentru a crea o resursă nouă într-un anumit loc. De exemplu, un client poate utiliza o cerere PUT pentru a actualiza detaliile unui produs dintr-un magazin online.
* DELETE: Metoda DELETE este utilizată pentru a șterge o resursă specifică de pe server. De exemplu, un client poate utiliza o cerere DELETE pentru a șterge un articol dintr-un blog.

Pe lângă metodele menționate mai sus, comunicarea HTTP în cadrul arhitecturii RESTful se bazează pe alte elemente importante:

* URI (Uniform Resource Identifier): Reprezintă un identificator unic care identifică resursa la care se face referire în cerere. De exemplu, într-o cerere GET, URI-ul poate indica adresa URL a unei pagini web sau a unui serviciu web.
* Header-uri HTTP: Furnizează detalii adiționale legate de cerere sau răspuns, cum ar fi specificarea tipului de conținut acceptat sau transmis, informații de autentificare, gestionarea cache-ului și altele.
* Body: Acesta reprezintă conținutul datelor trimise sau primite în cadrul unei cereri sau a unui răspuns HTTP. De exemplu, într-o cerere POST, corpul poate conține datele formularului trimise de utilizator.

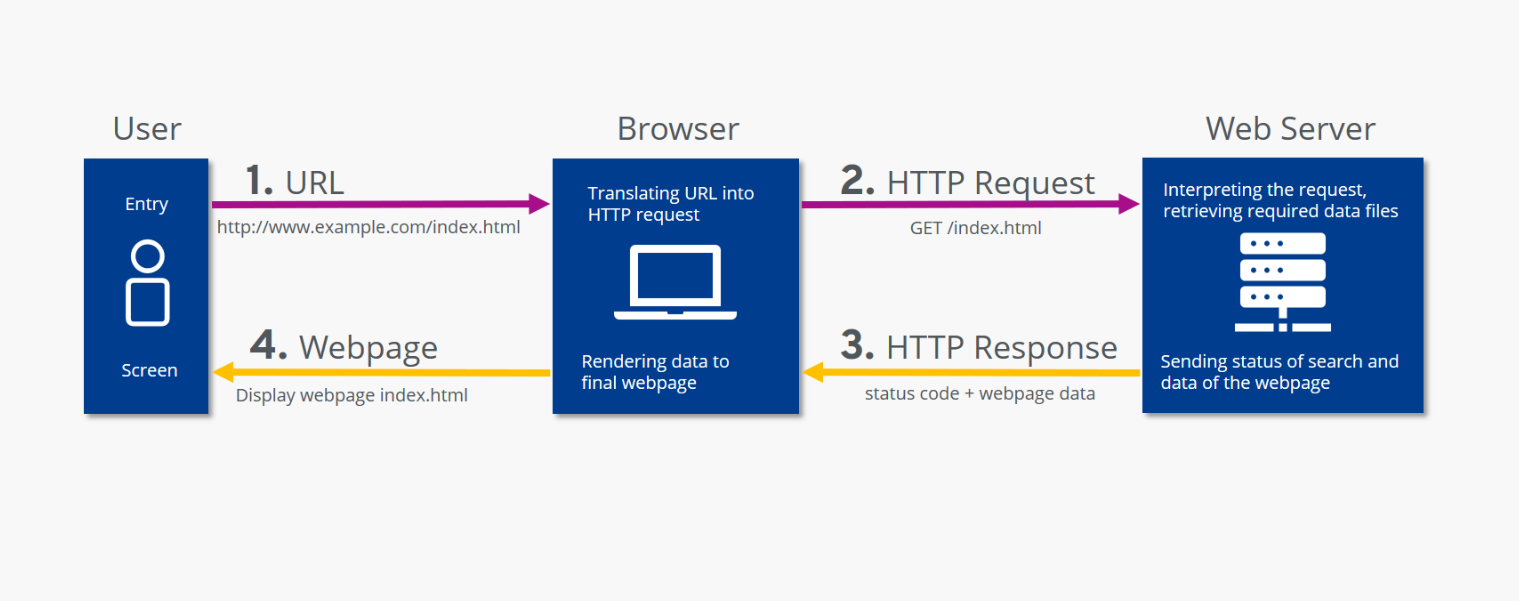


Figura 2.14: Comunicarea HTTP

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.

De asemenea, comunicarea HTTP oferă suport pentru diferite formate de date, cum ar fi JSON și XML, permițând astfel transmiterea și recepționarea datelor într-un mod flexibil și eficient.

### 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 autenticat. 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.

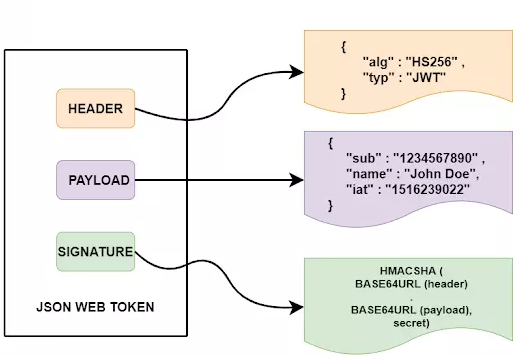


Figura 2.15: Structura JWT

Unul dintre principalele avantaje ale JWT este faptul că poate fi verificat rapid și eficient de către părțile implicate, fără a necesita interogarea unei baze de date sau un server de autentificare. Acest aspect face ca JWT să fie o soluție scalabilă și eficientă în aplicațiile distribuite.

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.

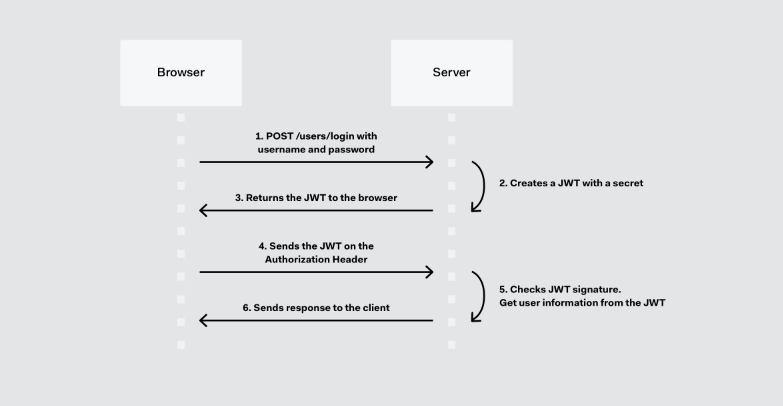


Figura 2.16: Modul de funcționare al JWT

Un alt aspect interesant al JWT este că poate transporta informații suplimentare despre utilizator, numite și claim-uri. Aceste claim-uri pot conține privilegii speciale, roluri, date de profil sau alte informații relevante pentru aplicație.

Este important de menționat că securitatea JWT depinde de păstrarea în siguranță a cheii secrete utilizate pentru semnătură. Aceasta trebuie protejată și accesibilă doar entităților autorizate. De asemenea, dimensiunea și complexitatea token-ului trebuie luate în considerare, deoarece acesta este trimis cu fiecare cerere, ceea ce poate afecta performanța aplicației.

### 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.

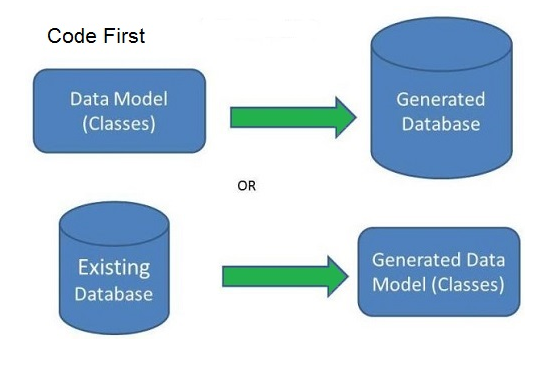


Figura 2.17: Code First vs Database First

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 atribute 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.

### 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. Am ales să utilizez Repository Pattern în cadrul aplicației mele pentru a obține o structură eficientă și modulară a accesului la 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.

# 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.

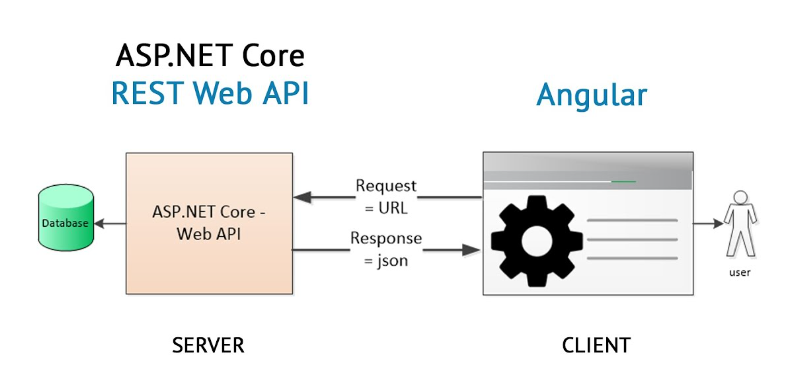


Figura 3.1: Arhitectura aplicației

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.

## 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.

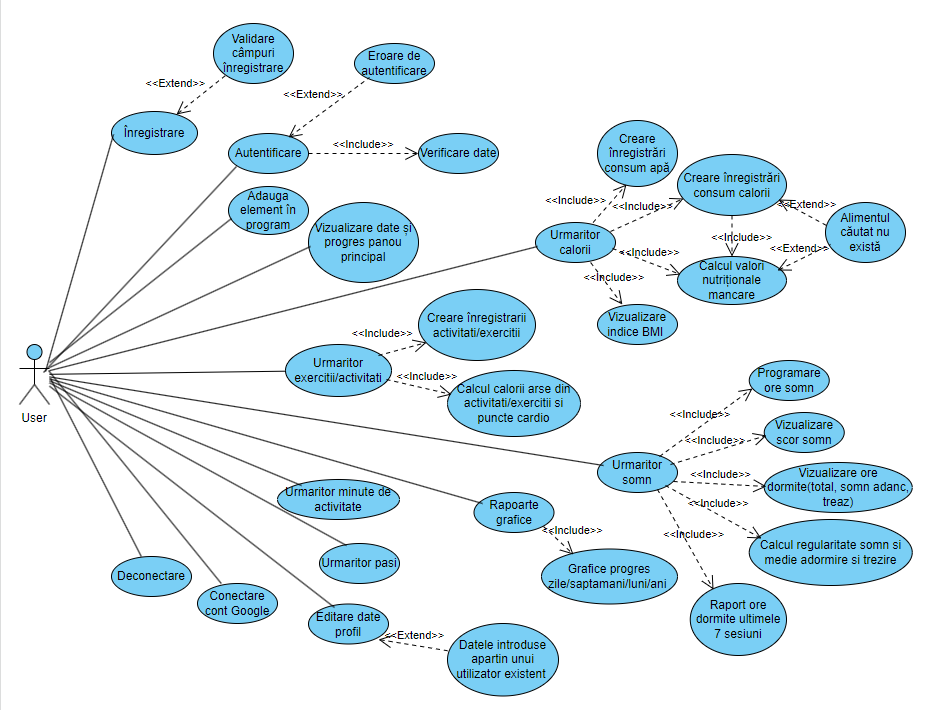


Figura 3.2: Diagrama de cazuri

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/activitați/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.

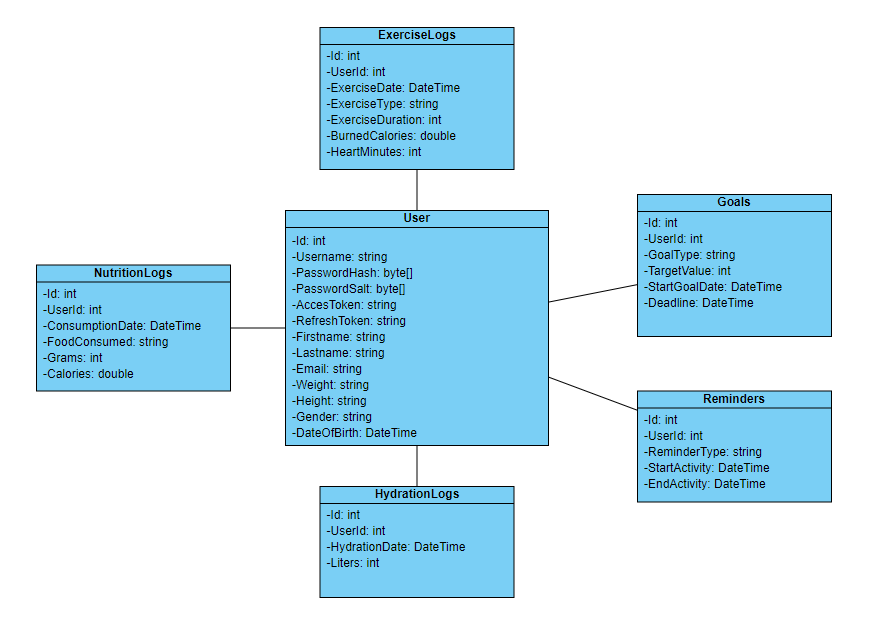


Figura 3.3: Diagrama de clase

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 adaugată numită “Salt” pentru a asigura securitatea parolei utilizatorilor. Tot aici avem si token-urile de acces pentru a accesa datele din aplicația mobile “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 înregistării, 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, cate 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-si 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 secțuinea “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.

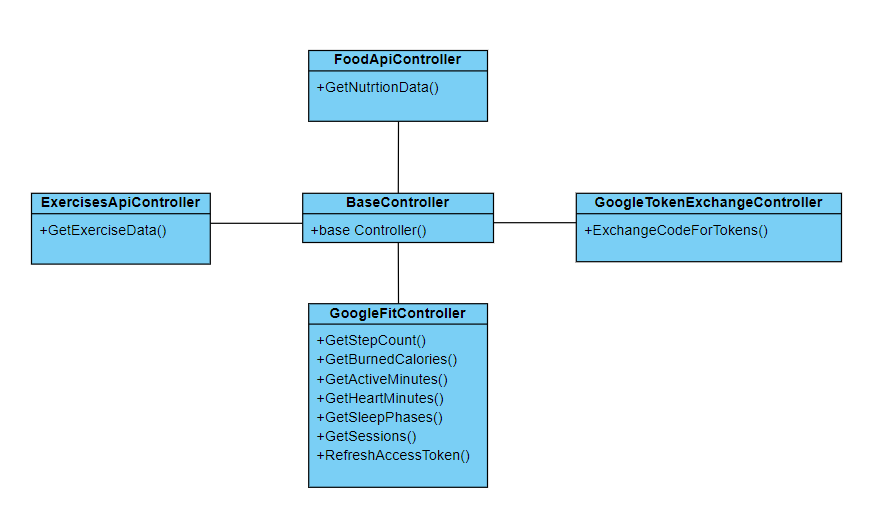


Figura 3.4: Diagrama claselor de control responsabile de comunicarea cu servere externe prin REST API.

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.

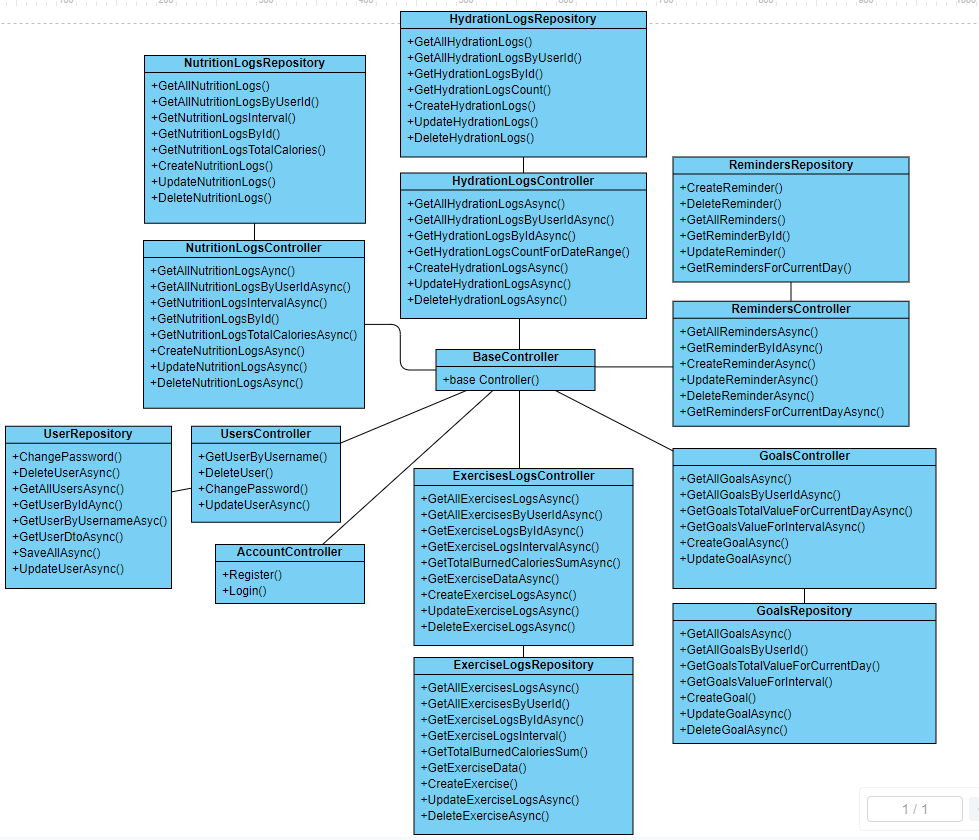


Figura 3.5: Diagrama claselor de control asociate cu clasele Repository

Această diagramă evidențiază relațiile dintre clase, totodată aratând metodele existente și aferente fiecarei 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 raspunsul cu informațiile necesare.

## Arhitectura serverului

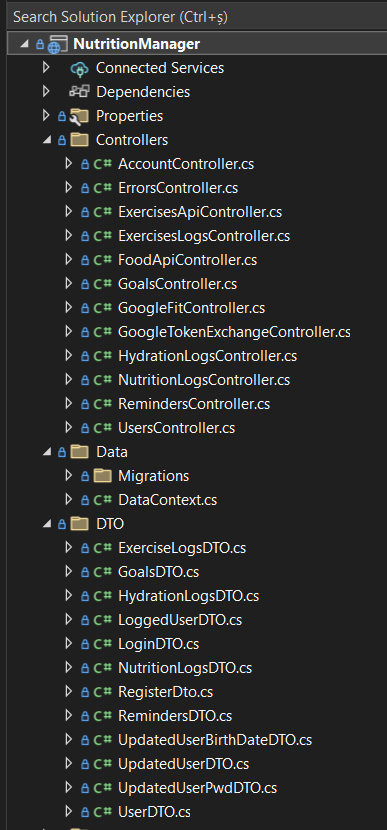
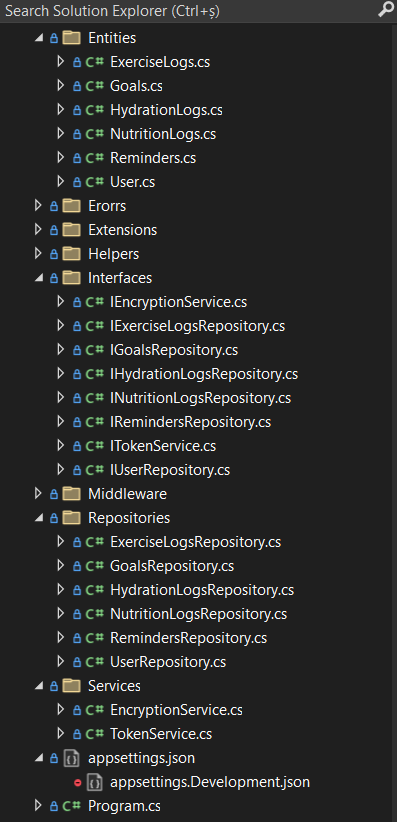
Arhitectura curată în ASP.NET Core Web API este o abordare arhitecturală care promovează modularitatea, separarea responsabilităților și testabilitatea în dezvoltarea aplicațiilor web. Aceasta se bazează pe principii precum împărțirea proiectului în straturi și dependențe inversate pentru a obține un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

### 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 atribute 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.

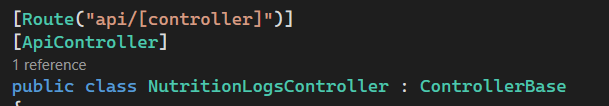


Figura 3.7: Atributul ApiController

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 raspuns 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.

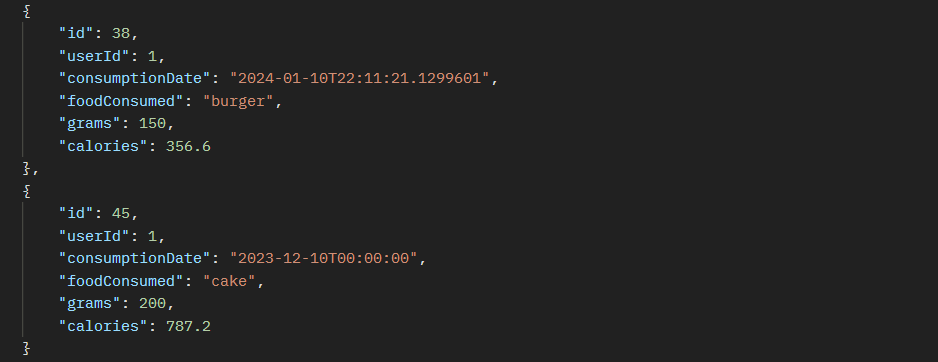


Figura 3.8: Exemplu de răspuns JSON

### 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.

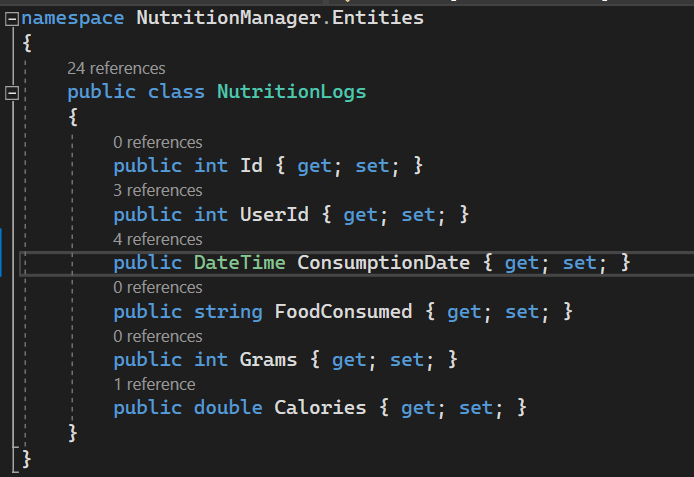


Figura 3.9: Exemplu de Entitate

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 encapsulează 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.

### 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.

### 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.

## 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 si 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.

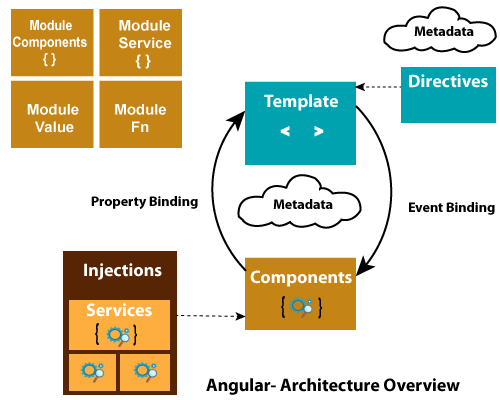


Figura 3.10: Arhitectura Angular

### 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.

### 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 metadate 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.

### Injectarea dependințelor

Un aspect arhitectural important al interfeței în Angular este injectarea dependințelor(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 dependințelor 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.

### 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 raspunsuri 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, caloriile 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.

## Elemente de Implementare

### Pagina de autentificare

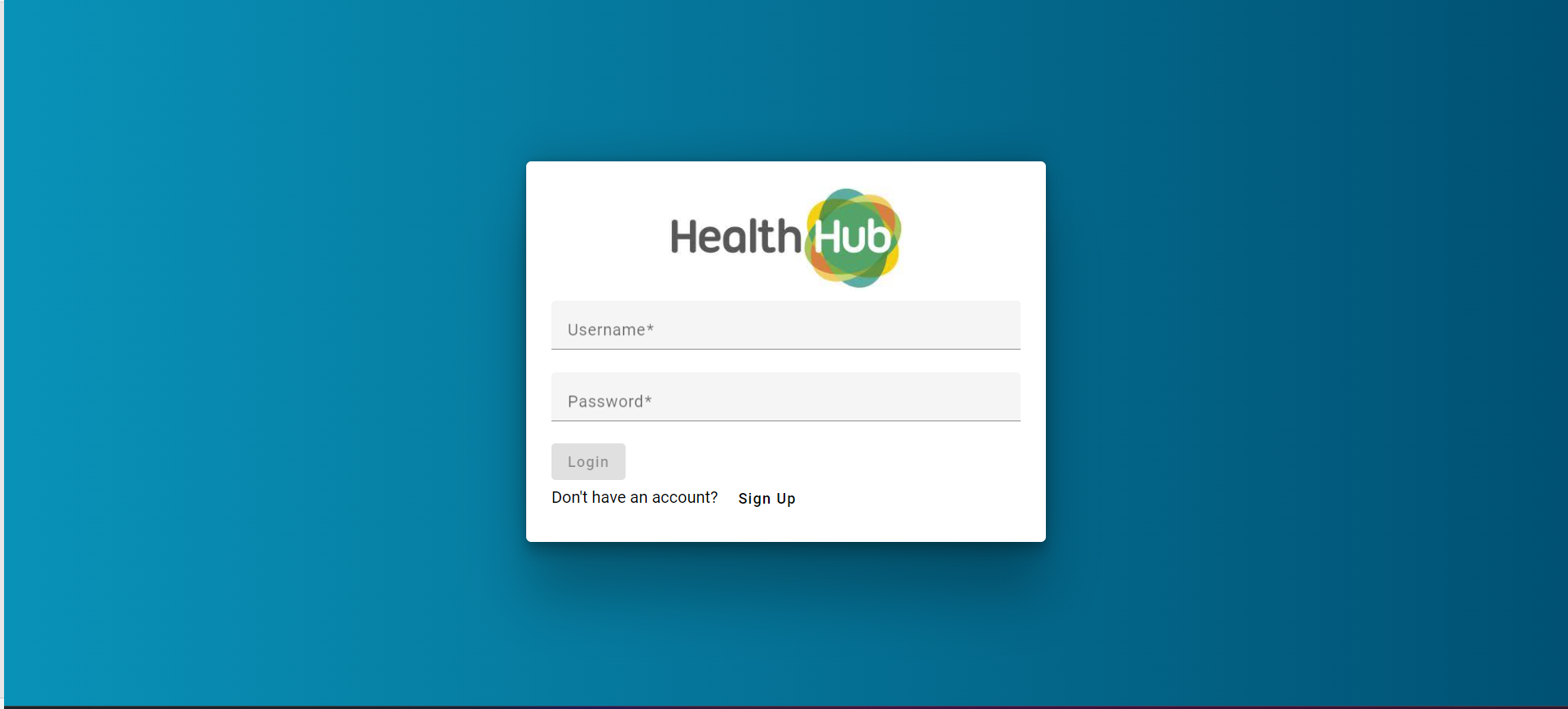


Figura 3.11: Pagina de autentificare

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 asignate 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.

### Pagina tabloului de bord

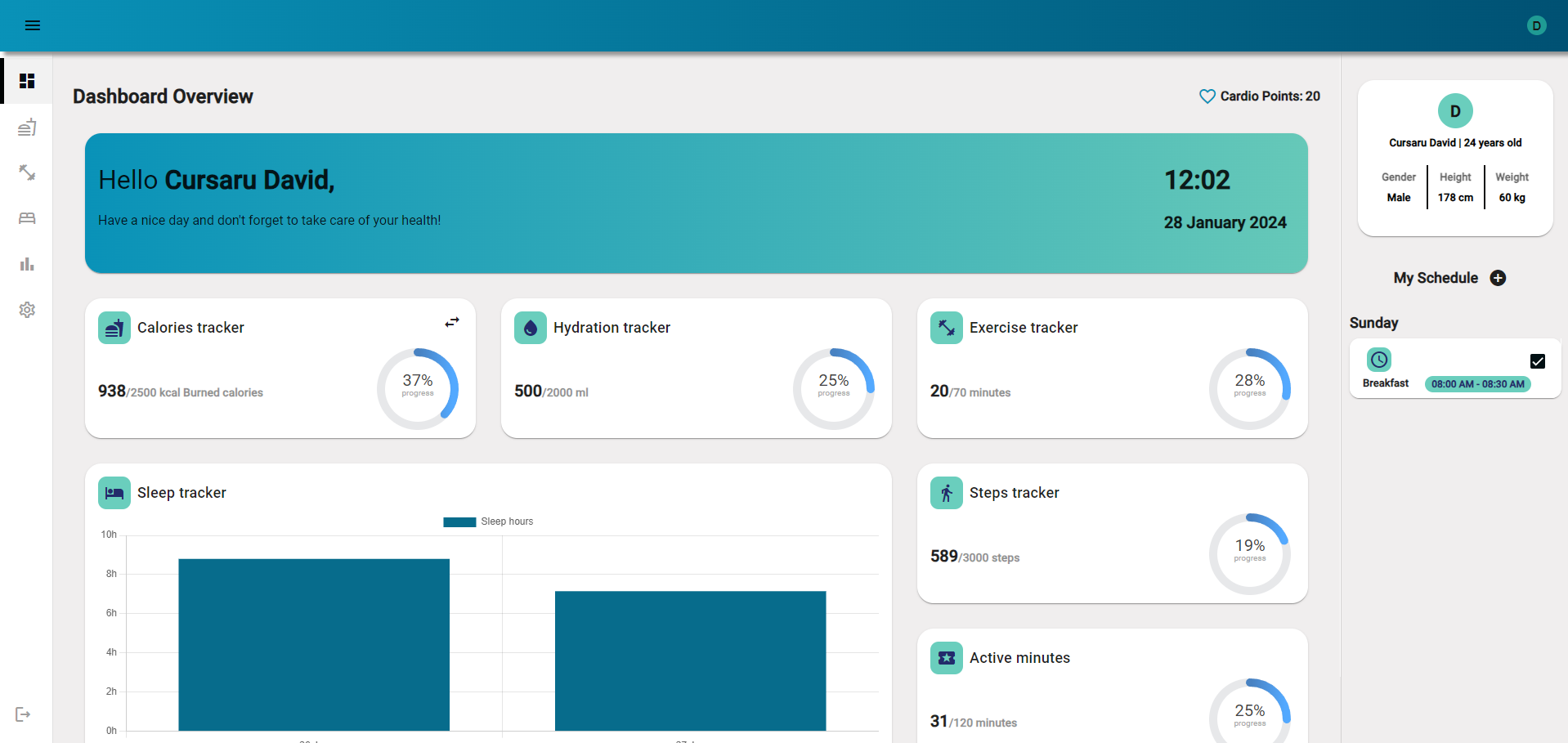


Figura 3.12: Pagina Dashboard overview

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 sănatate 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 bara extensibilă în partea dreaptă. Mai este și o bara extensibilă de navigare în partea stângă, prin care utlizatorul poate naviga prin toate paginilie 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 fiecarui card în funcție de indicele 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 parametrii sunt monitorizați doar pe ziua curentă, la începutul fiecarei zile pornindu-se de la 0. Totodată, cardurile conțin și obiectivele setate pe ziua respecetivă 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 întru-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 informațile 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 iși poate monitoriza numărul de pași, minutele de activitate, caloriile 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 cosumată de utilizator întru-un intervat 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 in 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 caloriilor arse de metabolism din Google Fit

Accesarea datelor din Google Fit implică folosirea protocolului de securitate OATH 2.O. 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 cand 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 OATH 2.0, cerându-se permisiunile de citire sau scriere a datelor pe care le prelucrează aplicația.

### Pagina de monitorizare a caloriilor

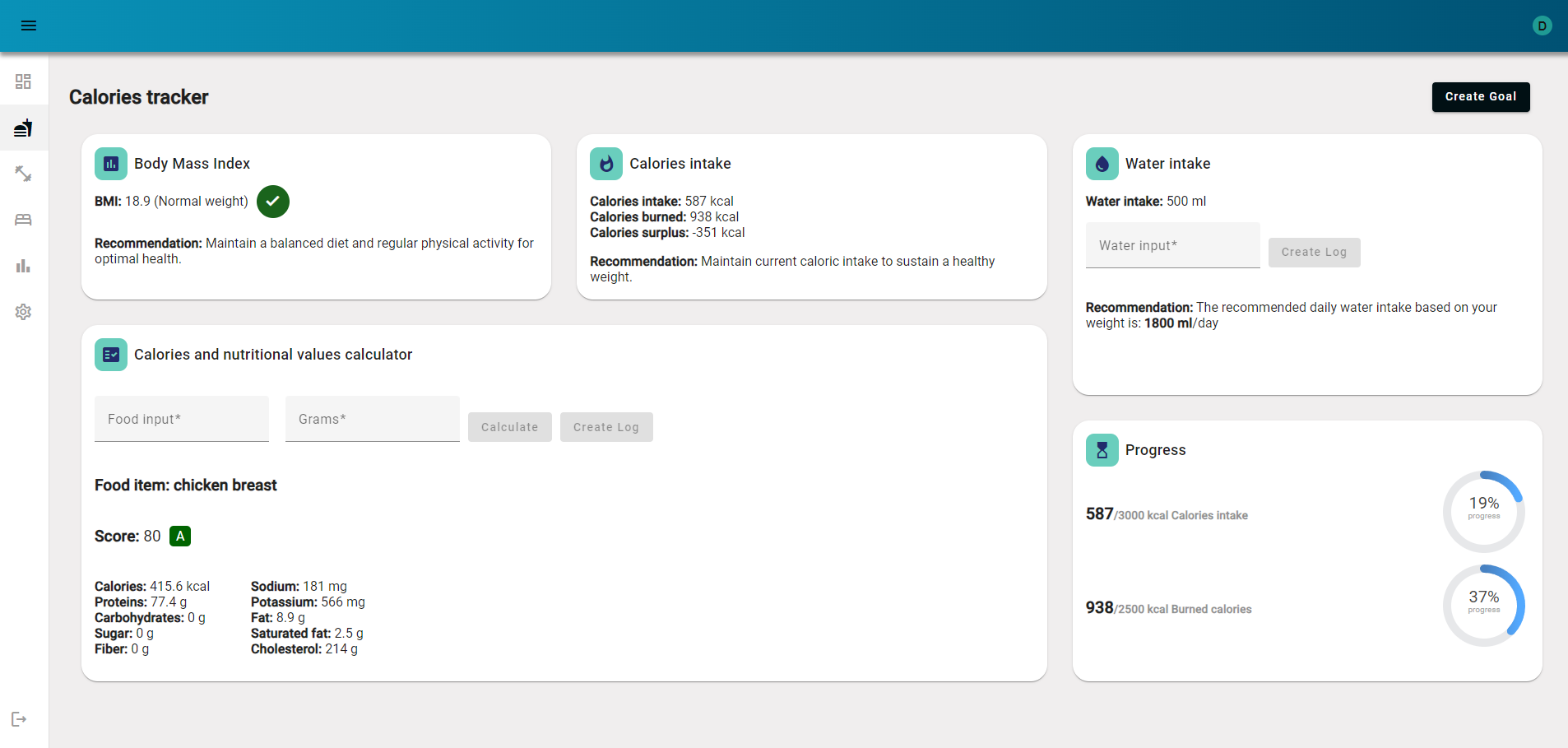


Figura 3.13: Pagina Calories tracker

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. Daca această diferență depașeste 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 depașesc o anumită valoare precum sarea, zahărul, colesterolul, punctajul se va reduce mai mult decât pentru proteine de exemplu. În cazul în care nu se depașeste 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 prelcura 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 caloriile 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

### Pagina de monitorizare a exercițiilor și activitățiilor

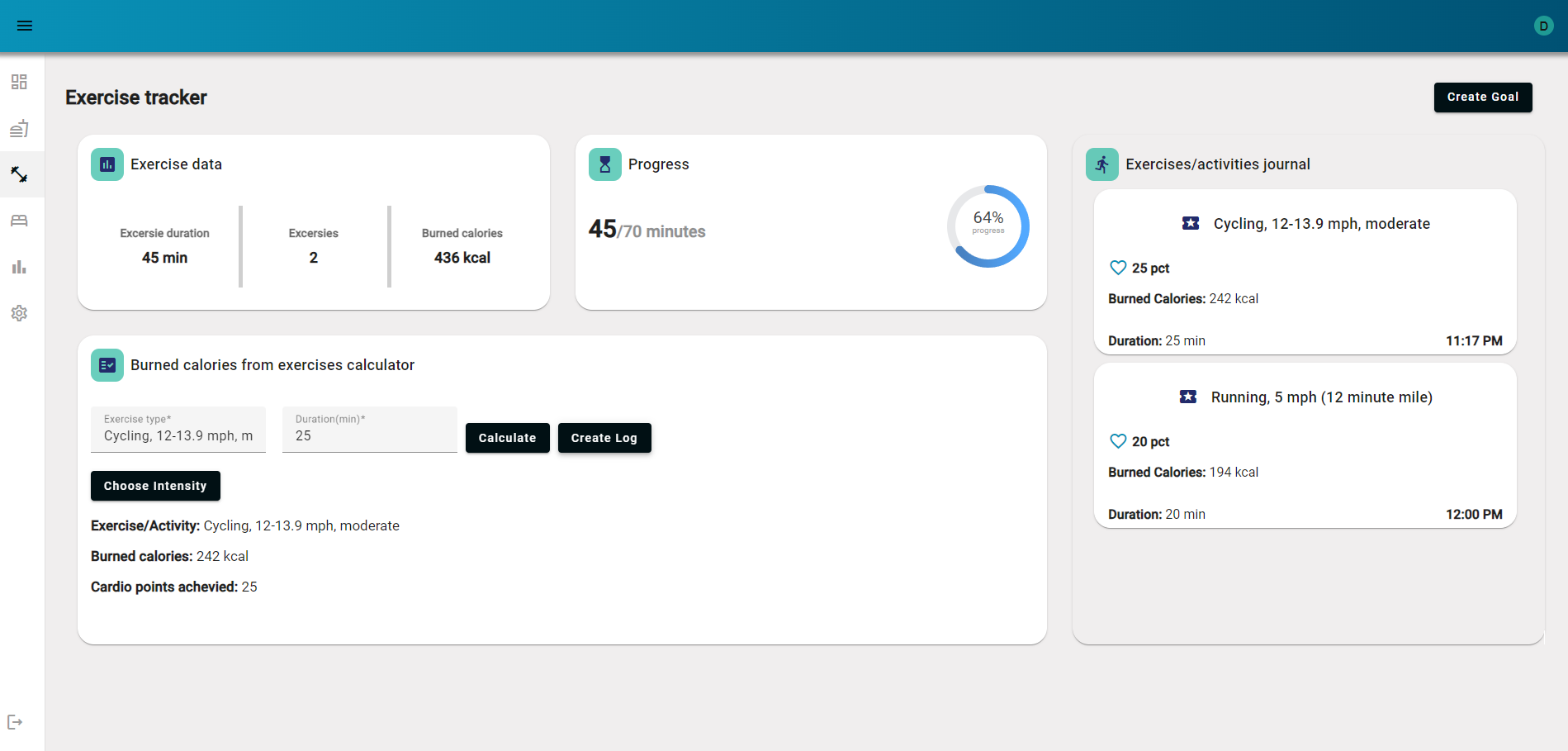


Figura 3.14: Pagina Exercise tracker

Pagina *Exercise tracker*  este similară cu pagina *Calories tracker*  deoarece pentru a calcula caloriile 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 sa primim un raspuns JSON cu caloriile arse pe ora pentru acest tip de exercițiu. Totuși, fiindcă în cazul exerciților 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 sa 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.

### Pagina de monitorizare a somnului

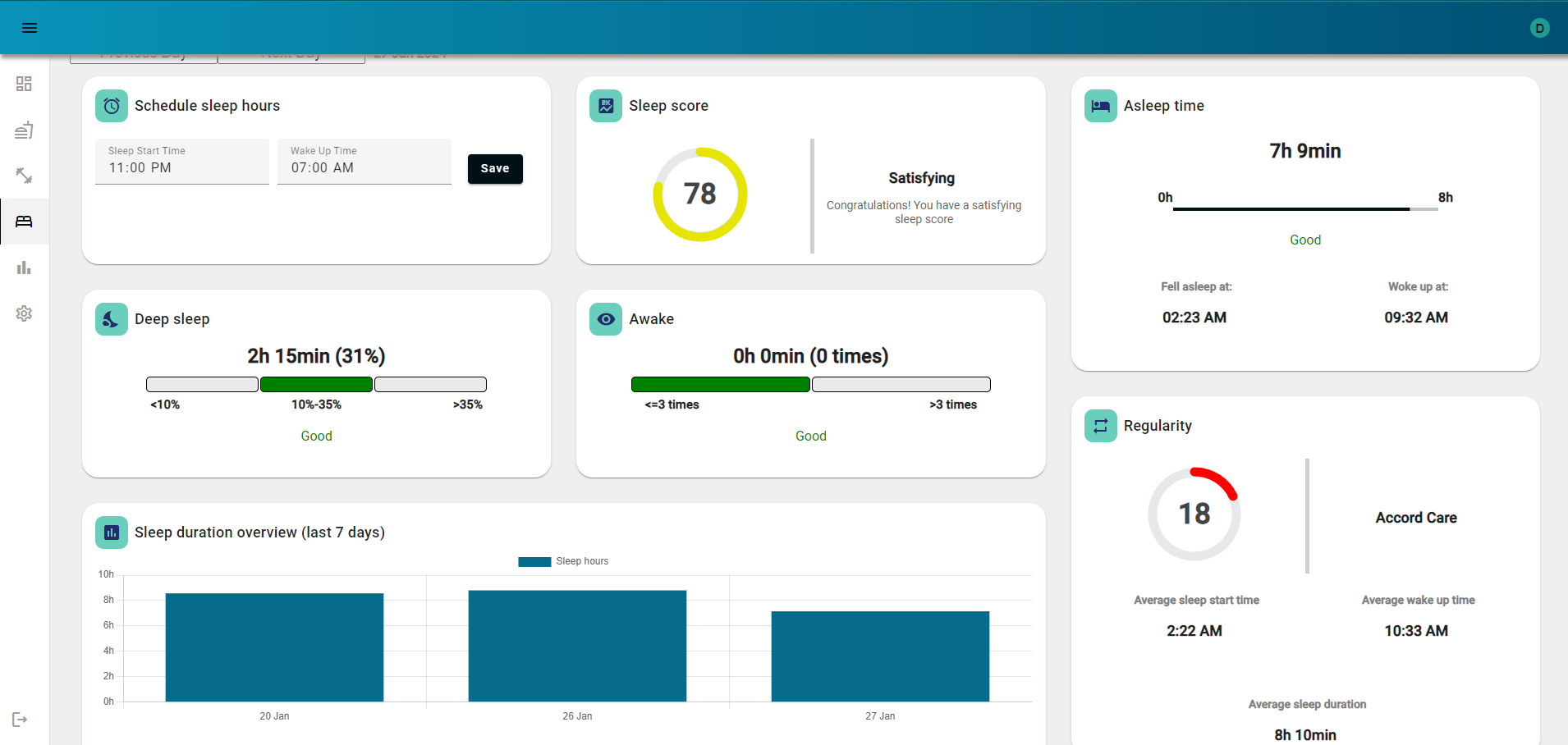


Figura 3.15: Pagina Sleep tracker

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 trezirii 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 terța 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.GetUserByIdAync(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 atributele starTimeNanos 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 componenetelor 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 atributele 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 înmultește cu 100 și se rotunjește la un număr întreg.

Pentru a calcula media orelor de începere a somnului și de trezire a fost puțin mai dificil la început datorita formatului orelor AM și PM al acestora. Eu am dorit să fac o medie a orelor folosind obiecte de tip Date, dar totodată am dorit ca aceste calcule să fie independente de dată, și acestea să reflecte efectiv media orelor de începere a somnului și a trezirii, nu un punct de mijloc între doua date și ore. În primă fază, am implementat acest calcul în mod intuitiv, ca și pentru o listă de numere, dar folosind variabile de tip Date. Făcând asta, am dat de o problemă care apare în momentul în care utilizatorul doarme la începutul unei zile, spre exemplu ora 02:00 dimineața, iar în aceeași zi seara, va dormi de la ora 22:00 sau orice altă oră din seara aceleiași zile. Acest caz va strica calculele de medie, și va da ca rezultat ora 12:00 prânz, eu dorind într-un astfel de caz să primesc 00:00, reprezentând miezul nopții.

Pentru a rezolva această problemă, am creat o abordare care folosește și urmărește următorul curs de acțuni:

1. **Convertirea orelor de începere a somnului în format de 24 de ore:**
   * Se utilizează Moment.js pentru a formata timpurile de început a somnului în formatul 'HH:mm' (24 de ore).
   * Rezultatul este un array de string-uri care reprezintă orele de început ale somnului convertite.
2. **Calcularea mediei în format float:**
   * String-urile formatate anterior sunt apoi parsate folosind Moment.js pentru a obține obiecte de timp Moment.
   * Se calculează suma totală a valorilor float reprezentând timpul în ore (folosind funcția **timeToFloat** pentru conversia din 'HH:mm' în valoare float).
   * Se calculează valoarea medie a acestor float-uri.
3. **Conversia în formatul orar 12 ore:**
   * Valoarea medie float este divizată în parte întreagă și zecimală.
   * Se ajustează orele în funcție de valoarea întreagă și se calculează minutele în funcție de partea zecimală.
   * Se determină dacă este dimineață sau după-amiază (AM/PM).
   * Se convertește înapoi la formatul 'hh:mm AM/PM'.
   * Rezultatul este ora medie a începutului somnului în format orar de 12 ore.

### Pagina de rapoarte grafice

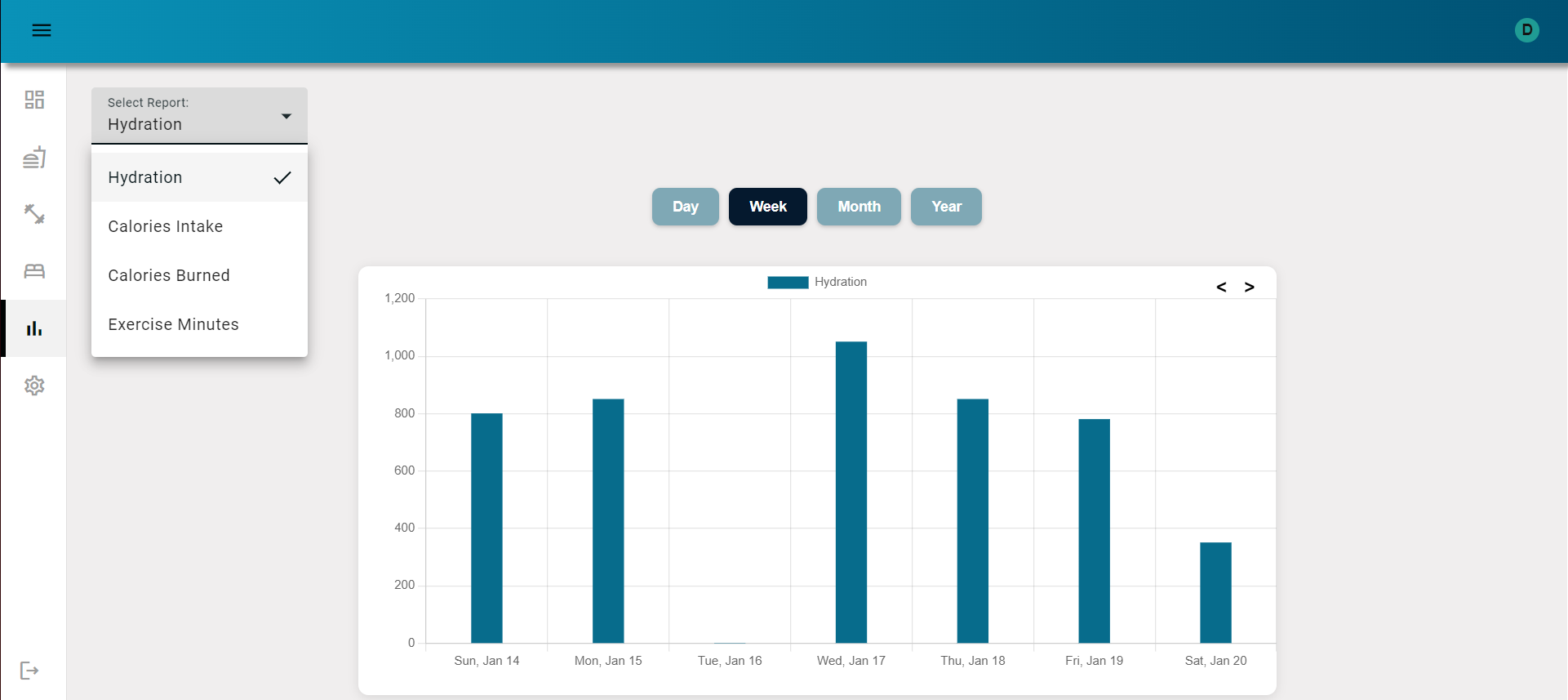


Figura 3.16: Pagina Reports

Pagina de rapoarte conține un grafic care poate afișa date pe diferite intervale de timp precum zile, saptamâni, 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 întru-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, marimea, 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 grupază 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.

## Testare și validare

# Concluzii

# Bibliografie

|  |  |
| --- | --- |
| [1] | A. H. D. N. J. R.-J. A. S. D. R. B. W. Kellyn Gorman, Introducing Microsoft SQL Server 2019, Packt Publishing, 2019. |
| [2] | D. Petkovik, Microsoft SQL Server 2019: A Beginner's Guide, 2020. |

# Lista de figuri, tabele și coduri sursă

FIGURI

[Figura 2.1: Comandă instalare Angular CLI 8](#_Toc157462154)

[Figura 2.2: Comandă creare proiect Angular 8](#_Toc157462155)

[Figura 2.3: Comandă generare componentă Angular 10](#_Toc157462156)

[Figura 2.4: Structura unei componente Angular 10](#_Toc157462157)

[Figura 2.5: Fișier HTML dintr-o componentă Angular 10](#_Toc157462158)

[Figura 2.6: Fișier TypeScript dintr-o componentă Angular 10](#_Toc157462159)

[Figura 2.7: Comandă adaugare Angular Material în proiect 11](#_Toc157462160)

[Figura 2.8: Interfața de programare a aplicațiilor 12](#_Toc157462161)

[Figura 2.9: Modul de funcționare al ASP .NET Core Web API 13](#_Toc157462162)

[Figura 2.10: Istoria versiunilor Microsoft SQL Server 14](#_Toc157462163)

[Figura 2.11: JavaScript vs TypeScript 17](#_Toc157462164)

[Figura 2.12: Ecosistemul .NET 18](#_Toc157462165)

[Figura 2.13: Single Page Application vs Multiple Page Application 21](#_Toc157462166)

[Figura 2.14: Comunicarea HTTP 24](#_Toc157462167)

[Figura 2.15: Structura JWT 25](#_Toc157462168)

[Figura 2.16: Modul de funcționare al JWT 25](#_Toc157462169)

[Figura 2.17: Code First vs Database First 26](#_Toc157462170)

[Figura 3.1: Arhitectura aplicației 28](#_Toc157462171)

[Figura 3.2: Diagrama de cazuri 30](https://d.docs.live.net/902647bd31baf3bf/Desktop/Proiect%20licenta%20Cursaru%20David.docx#_Toc157462172)

[Figura 3.3: Diagrama de clase 31](https://d.docs.live.net/902647bd31baf3bf/Desktop/Proiect%20licenta%20Cursaru%20David.docx#_Toc157462173)

[Figura 3.4: Diagrama claselor de control responsabile de comunicarea cu servere externe prin REST API. 33](#_Toc157462174)

[Figura 3.5: Diagrama claselor de control asociate cu clasele Repository 34](#_Toc157462175)

[Figura 3.6: Structura fișierelor în server 35](https://d.docs.live.net/902647bd31baf3bf/Desktop/Proiect%20licenta%20Cursaru%20David.docx#_Toc157462176)

[Figura 3.7: Atributul ApiController 36](#_Toc157462177)

[Figura 3.8: Exemplu de răspuns JSON 37](#_Toc157462178)

[Figura 3.9: Exemplu de Entitate 38](#_Toc157462179)

[Figura 3.10: Arhitectura Angular 41](#_Toc157462180)

[Figura 3.11: Pagina de autentificare 45](#_Toc157462181)

[Figura 3.12: Pagina Dashboard overview 48](#_Toc157462182)

[Figura 3.13: Pagina Calories tracker 52](#_Toc157462183)

[Figura 3.14: Pagina Exercise tracker 54](#_Toc157462184)

[Figura 3.15: Pagina Sleep tracker 56](#_Toc157462185)

[Figura 3.16: Pagina Reports 59](#_Toc157462186)

CODURI SURSĂ

[Codul 1: Metodă din cadrul clasei de control NutritionLogsController 37](#_Toc157462187)

[Codul 2: Exemplu de clasă Repository 39](#_Toc157462188)

[Codul 3: Formular de autentificare și înregistrare în HTML 42](#_Toc157462189)

[Codul 4: Secțiune de cod cu câmpuri care conțin directiva mat-form-field 42](#_Toc157462190)

[Codul 5: Cod sursă pentru Componenta LiveClock 43](#_Toc157462191)

[Codul 6: Exemplificarea unui constructor de componentă în Angular 44](#_Toc157462192)

[Codul 7: Cod sursă pentru exemplificarea unui serviciu din cadrul aplicației 44](#_Toc157462193)

[Codul 8: Metoda Register din clasa AccountController 46](#_Toc157462194)

[Codul 9: Metoda Login din clasa AccountController 47](#_Toc157462195)

[Codul 10: Serviciu pentru luarea cantității de apă consumate într-un interval de timp 50](#_Toc157462196)

[Codul 11: Serviciu pentru preluarea caloriilor arse de metabolism din Google Fit 50](#_Toc157462197)

[Codul 12: Obținerea token-urilor de acces de la Google 52](#_Toc157462198)

[Codul 13: Metodă din server pentru returnarea datelor de nutriție pentru un aliment ca și răspuns 54](#_Toc157462199)

[Codul 14: Generare listă auto-completare pentru tipuri de exerciții 55](#_Toc157462200)

[Codul 15: Apelul HTTP GET pentru a primi sesiunile de somn dintr-un interval de timp 57](#_Toc157462201)

# Rezumat

# Abstract