**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

# GymJournal: Aplicație de gestionat antrenamente pentru Android

**Absolvent**

**Marian Dimofte**

**Coordonator științific**

**Marius Adrian Dumitran**

**București, iunie 2023**

# Rezumat

GymJournal este o aplicație de ultimă oră destinată pentru a eficientiza managementul și analiza sesiunilor de antrenament la sălile de forță. Folosind tehnologia *mobile* GymJournal oferă utilizatorilor o platformă ușor de folosit pentru a înregistra, monitoriza și analiza rutinele lor de antrenament. Cu funcționalități precum contorizarea exercițiilor, planificarea antrenamentelor, urmărirea progresului și vizualizarea informației introduse GymJournal dă posibilitate persoanelor să își optimizeze antrenamentul, să își seteze obiective realistice și să își urmărească progresul de-a lungul unei perioade de timp.

# Abstract

GymJournal is a cutting-edge application designed to streamline the management and analysis of gym workout sessions. By leveraging mobile technology, GymJournal provides users with a user-friendly platform to record, monitor, and analyze their fitness routines. With features like exercise logging, workout planning, progress tracking, and data visualization, GymJournal empowers individuals to optimize their training, set realistic goals, and track their progress over time.

# Cuprins

# Capitolul I – Introducere

În perioada recentă s-a observat o creștere semnificativă a popularității activităților sportive și a exercițiilor fizice. O gamă largă de persoane de toate vârstele și de toate clasele sociale încep să opteze pentru un stil de viață mai activ. Pentru a putea atinge aceste obiective sportive este nevoie de o metoda pentru gestionarea unui plan eficient și observarea progresului pe o perioada lunga de timp. Metodele tradiționale de urmărire a antrenamentelor, precum pixul și hârtia sau alte aplicații generale de luat notițe, s-au dovedit a fi destul ineficiente de folosit în practică de-a lungul timpului. Totuși, cu apariția telefoanelor inteligente și a noilor progrese în tehnologia mobilă a fost deschisă ușa către noi posibilități pentru dezvoltarea de soluții inovative ce au ca scop satisfacerea nevoilor în creștere ale entuziaștilor de fitness.

În răspuns la cererea în creștere, această lucrare de licență prezintă GymJournal, o aplicație inovativă proiectată special pentru a adresa provocările asociate cu stocarea și organizarea informației despre sesiunile de antrenament la sală precedente. GymJournal valorifică capabilitățile dispozitivelor mobile pentru a oferi utilizatorilor o platforma cuprinzătoare și ușor de folosit care eficientizează procesul de a nota, gestiona și analiza sesiunile la sala de sport.

Obiectivul principal al aplicației este de a oferi o soluție completă care întrece limitările metodelor tradiționale de a gestiona și analiza antrenamentele. Folosind cele mai noi tehnologii, GymJournal are ca țintă îmbunătățirea activității de a merge la sală pentru orice persoana. Aceasta ajută utilizatorii să rămână organizați, motivați și concentrați pentru a-și îndeplinii obiectivele în lumea fitness-ului. Prin interfața sa intuitivă și prin gama largă de funcționalități, GymJournal dă posibilitatea utilizatorilor să noteze detalii cu privire la un exercițiu, să planifice antrenamente, să monitorizeze progresul, și în final, să facă decizii informate în legătură cu rutinele de antrenament.

Una dintre funcționalitățile cheie ale aplicației este cea de înregistrare a exercițiilor. Utilizatorii pot, fără efort, sa rețină detalii vitale a oricărei sesiuni de antrenament precum: exercițiul lucrat, numărul de seturi efectuate, numărul de repetări efectuate, greutățile folosite și timpul efectuat în completarea exercițiilor. Această interpretare eficientă elimină nevoia pentru documentare manuală și micșorează șansele de a greși sau omite detalii importante.

Capitolele acestei teze sunt structurate în felul următor:

- Tehnologii – pe partea de server: Acest capitol se concentrează pe partea de tehnologii folosite pentru implementarea server-ului GymJournal. De asemenea include arhitectura *back-end* precum baza de date, limbajele de programare, și *framework*-urile folosite pentru dezvoltarea ei.
- Tehnologii – pe partea utilizatorului: Acest capitol se concentrează pe partea de tehnologii folosite pentru implementarea părții de client a GymJournal. Include detalii despre platforma Android[1], despre .NET[2], *framework*-ul MAUI[3] și alte tehnologii relevante utilizate pentru crearea interfeței prezentate utilizatorului.
- Implementare – pe partea de server: Acest capitol se concentrează pe modul în care tehnologiile alese au fost folosite pentru implementarea părții de server a aplicației. De asemenea acoperă aspecte legate de stocarea și gestionarea datelor, dezvoltarea unui Application Programming Interface (API), autentificarea și autorizarea mecanismelor și alte funcționalități pe parte de server cruciale pentru operare a aplicației.
- Implementare – pe partea utilizatorului: Acest capitol este concentrat pe partea de implementare a părții de client a aplicației. El acoperă aspecte legate de proiectarea și dezvoltarea interfeței, metode prin care utilizatorul poate interacționa cu datele, integrarea unui API și alte funcționalități care contribuie la eficiența aplicației și la crearea unei interfețe ușor de înțeles și de folosit.
- Manualul Utilizatorului: Acest capitol reprezintă un ghid cuprinzător despre folosirea aplicației GymJournal. Oferă instrucțiuni pas cu pas pentru utilizarea funcționalităților aplicației precum crearea unui contului de utilizator, înregistrarea exercițiilor, planificarea antrenamentelor, urmărirea progresului și vizualizarea datelor. Acest capitol se asigură că utilizatorul poate, în mod eficient, să navigheze și să utilizeze funcționalitățile aplicației pentru a-și atinge obiectivele sportive.
- Concluzie: Capitolul final discută eficiența și ușurința utilizării aplicației, evidențiind limitările și provocările întâlnite pe parcursul dezvoltării sale și sugerează potențiale îmbunătățirii care pot fi aduse aplicației pe viitor.

# Capitolul II – Tehnologii

În acest capitol din lucrare sunt acoperite tehnologiile folosite pentru dezvoltarea GymJournal, o aplicație Android .NET MAUI proiectată pentru a eficientiza urmărirea și analiza sesiunilor de antrenament. Acest capitol cuprinde tehnologiile pe partea de client și pe partea de server ce contribuie la funcționarea, eficiența si performanța aplicației.

Progresele rapide în materie de tehnologie mobilă si disponibilitatea de *framework*-uri robuste au pavat calea pentru soluții inovative care satisfac nevoile în continuă creștere ale entuziaștilor de fitness. GymJournal capitulează pe aceste progrese tehnologice pentru a oferi utilizatorilor o platformă cuprinzătoare și ușor de folosit pentru a gestiona eficient antrenamentele la sală.

Capitolul începe prin a explora tehnologiile și *framework*-urile folosite pe partea de server care alimentează funcționalitatea din spatele aplicației. Tehnologiile din partea de server cuprind *framework*-ul folosit, în cazul nostru ASP.NET Core, cunoscut pentru scalabilitatea, performanța și ecosistemul lui de librării și unelte. Acest capitol de asemenea discută selecția unei baza de date, cum ar fi Microsoft SQL Server, pentru a stoca eficient și pentru a gestiona datele utilizatorilor și alte informații ale aplicației, precum alte exerciții.

În plus, acest capitol conține informații despre dezvoltarea unui API folosind ASP.NET Web API, ceea ce facilitează comunicarea fără dificultăți dintre partea de client și partea de server. Incorporarea de autentificare bazată pe *token* ce asigură acces securizat la funcționalitățile aplicației și protejează datele utilizatorilor.

Capitolul continuă cu prezentarea tehnologiilor folosite pe partea de client folosite în dezvoltarea GymJournal. Acesta evidențiază utilizarea platformei Android care cuprinde un număr vast de utilizatori și un set de unelte și resurse diversificat pentru crearea de aplicații mobile ușor de folosit și performante. În plus, capitolul discută implementarea *framework*-ului .NET MAUI, care oferă posibilitatea dezvoltării de aplicații *cross-platform* cu un singur *code-base*, ceea ce asigură accesibilitate largă de-a lungul a mai multor dispozitive și a mai multor sisteme de operare.

Astfel, acest capitol prevede o vedere de ansamblu a tehnologiilor pe partea de server și pe partea utilizatorului care alimentează GymJournal. Folosind aceste unelte avansate și *framework*-uri, aplicația are ca țintă să îmbunătățească experiența mersului la sală pentru orice persoană prin oferirea unei platforme intuitive, eficientă și de încredere pentru a își analiza sesiunile de antrenament.

# Secțiunea II.1 - Server

Capitolul de tehnologii pe partea de server al acestei lucrări de licență explorează tehnologiile cheie și *framework*-urile folosite în dezvoltarea componentelor serverului GymJournal. Acesta este dezvoltat cu ajutorul *framework*-ului .NET și *Entity Framework* [5] (EF). Acest capitol cuprinde tehnologiile de pe partea de server care alimentează infrastructura părții de *back-end* a aplicației, ce oferă posibilitate aplicației să stocheze, proceseze și să gestioneze într-un mod cât mai eficient datele utilizatorilor și datele antrenamentelor.

În epoca digitalizării, tehnologiile pe partea de server joacă un rol crucial în prevederea funcționalității și a infrastructurii pentru aplicațiile web și mobile. GymJournal se folosește o gamă de tehnologii pe partea de server pentru a asigura comunicarea fără probleme dintre partea de client și server, stocare și procurare de date securizată și procesare eficientă de cereri din partea utilizatorului.

Capitolul începe prin a discuta alegerea *framework*-ului pentru partea din spate a GymJournal, ce este ASP.NET Core. ASP.NET Core este un *framework* modern și puternic pentru dezvoltare web produs de către Microsoft. Robustețea, scalabilitatea și capabilitatea *cross-platform* ale *framework*-ului îl fac o alegere ideală pentru implementarea pe partea de server a aplicției. *Framework*-ul ASP.NET Core oferă o fundație solidă pentru a construi aplicații web și ajută aplicația GymJournal să livreze o experiență de încredere și receptivă utilizatorilor.

Apoi, capitolul explorează mai departe aspectul tehnologic legat de gestionarea bazei de date a părții de server. Microsoft SQL Server[6], un popular și robust sistem relațional de gestionare a bazelor de date (DBMS), este utilizat pentru a stoca și gestiona datele utilizatorilor, informațiile legate de exerciții, și alte date relevante pentru aplicație. SQL Server oferă capabilități avansate de interogare, integritate a datelor și funcționalități de securitate, asigurând reliabilitatea și securitatea bazei de date.

Un alt aspect crucial acoperit în acest capitol este dezvoltarea API-ului. GymJournal folosește ASP.NET Web API să expună un set de porturi bine definite care facilitează comunicarea dintre aplicație și server. Acest API manevrează autentificarea utilizatorilor, procurarea datelor legate de exerciții, stocarea datelor și alte operații care sunt necesare pentru a asigura o comunicare lină și o integrare fără efort dintre componentele client și server ale GymJournal.

Securitatea este de importanță maximă când te ocupi cu datele utilizatorilor. Acest capitol de asemenea evidențiază implementarea autentificării pe bază de *token* în cadrul aplicației. Această abordare îmbunătățește securitatea aplicației asigurând faptul că doar utilizatorii autorizați pot să acceseze funcționalitățile aplicației și faptul că datele utilizatorilor sunt protejate de accesul neautorizat al altor persoane.

Tehnologiile pentru partea de server folosite de GymJournal din coloana vertebrală a funcționalității ei, asigură gestionarea eficientă a datelor, comunicare securizată, și performanță de încredere. Folosind aceste tehnologii, GymJournal oferă utilizatorilor o platformă robustă și securizată pentru managementul și analizarea sesiunilor de antrenament.

**Framework-ul din spate**

GymJournal se bazează pe *framework*-ul ASP.NET Core drept schelet pentru implementarea părții de server a aplicației. ASP.NET este un *framework* modern făcut pentru dezvoltare web, produs de către Microsoft. El oferă o infrastructură flexibilă și scalabilă pentru construirea de aplicații web și servicii de mare performanță.

ASP.NET Core oferă mai multe avantaje care îl fac o alegere potrivită pentru dezvoltarea din spatele GymJournal:

1. Capabilitatea *cross-platform*: ASP.NET Core este proiectat pentru a fi *cross-platform*, oferind GymJournal posibilitatea de a fi folosit pe mai multe sisteme de operare, precum Windows, Linux și macOS. Această versatilitate asigură mare accesibilitate și flexibilitate în calitate de opțiuni de găzduire.
2. Performanță și scalabilitate: ASP.NET Core este optimizat pentru performanță înaltă și scalabilitate, asigurând servirea unui număr mare de utilizatori ai GymJournal simultani și al unui volum mare de lucru. Arhitectura lui modulară și ușoară contribuie la timpi de răspuns rapizi și la utilizarea eficientă a resurselor.

3. Arhitectura de tip MVC[7]: ASP.Net Core urmărește șablonul arhitectural de tip MVC (Model-View-Controller) care oferă o separare clară a rolurilor unui obiect și facilitează menținerea pe termen lung a aplicației. Șablonul MVC promovează organizarea codului, făcând mai ușor de manageriat și de îmbunătățit componente diferite ale aplicației.
4. Ecosistem Vast: ASP.NET Core beneficiază de un ecosistem vast de librării, unelte și suport din partea comunității. Acest ecosistem oferă o gamă largă de componente și integrări gata de utilizat, accelerând dezvoltarea și reducând efortul necesar pentru a implementa funcționalități comune.
5. Integrarea cu .NET și C#[8]: ASP.NET Core este construit pe *framework*-ul .NET Core și utilizează limbajul de programare C#. Această integrare oferă programatorilor posibilitatea de a folosi funcționalitățile puternice ale ecosistemului .NET, incluzând *strong typing, garbage collection*, și librăria de clase cuprinzătoare.
6. *Middleware Pipeline*: ASP.NET Core oferă unelte robuste pentru testare și *debugging*, incluzând suport integrat pentru testarea unitate și integrată automată. Acesta facilitează dezvoltarea de componente de încredere și fără erori pentru GymJournal.

Folosind framework-ul ASP.NET Core, GymJournal beneficiază de performanță, scalabilitate, capabilitate cross-platform și un ecosistem vast. Aceste avantaje contribuie la dezvoltarea infrastructurii robuste și de încredere din spate ce alimentează funcționalitățile care stau la centrul aplicației, asigurând o experiență eficientă și fără probleme pentru utilizator.

**Framework-ul din spate**

În implementarea părții de *back-end* a GymJournal, gestionarea bazei de date într-un mod eficient și de încredere este esențial pentru stocarea, procurarea datelor utilizatorilor și datelor antrenamentelor. GymJournal folosește un sistem relațional de management al bazelor de date (DBMS) ce este robust, în mod specific, Microsoft SQL Server, pentru a îndeplini aceste cereri.

Un DBMS servește ca și fundație pentru stocarea și organizarea datei într-o formă cât mai structurată. Oferă un set de unelte și funcționalități pentru a interacționa cu baza de date, a realiza interogări, a îmbunătăți integritatea datelor și are grijă de securitatea și procurarea datelor în mod eficient. Alegerea de a folosi Microsoft SQL Server drept DBMS pentru

GymJournal este condusă de gama extinsă de funcționalități și de reputația bună pentru performanță și fiablitate.

Microsoft SQL Server oferă câteva avantaje pentru managementul bazei de date al aplicației:

1. Capabilități relaționale ale bazei de date: Microsoft SQL Server este un DBMS relațional bine cunoscut și bogat în funcționalități. Are suport pentru modelul relațional, oferind organizarea datei în tabele cu relații bine definite. Această structură oferă aplicației capabilitatea de a stoca și gestiona diferite entități ale datelor, cum ar fi utilizatorii, exercițiile, sesiunile de antrenament, într-un mod structurat și eficient.
2. Integritatea și securitatea datelor: Microsoft SQL Server oferă mecanisme robuste pentru a facilita integritatea datelor și pentru a menține securitatea informațiilor sensibile. Oferă suport pentru funcționalități precum constrângeri, declanșatori, proceduri stocate pentru a facilita respectarea regulilor aplicației și să mențină consistența datelor. Pot fi definite constrângeri pentru a asigura acuratețea și validitatea datelor stocate, prevenind inconsistența și introducerea de date eronate.
3. Optimizarea interogării și performanței: Microsoft SQL Server oferă capabilități puternice pentru interogare, oferind procurarea de date specifice către API bazate pe criterii variate. Limbajul SQL[9], care are suport pe SQL Server, oferă un set bogat de operatori și funcții pentru interogarea și manipularea datelor. În plus, SQL Server oferă funcționalități de optimizare a indexării și a performanței care îmbunătățesc viteza și eficiența procurării de date.
4. Scalabilitatea și avabilitatea: Microsoft SQL Server oferă opțiuni de scalabilitate pentru a manevra un număr mare de date robuste și trafic de utilizatori intens. Are suport pentru funcționalități precum replicare, clusterizare, partiționare, oferind aplicației capabilitatea de a scala infrastructura bazei de date, pe măsură ce numărul de utilizatori crește.

**Interacțiunea cu baza de date**

Pentru a interacționa cu baza de date SQL Server, GymJournal folosește Entity Framework (EF), un *framework* popular oferit de Microsoft care face cartografiere obiect-relațională. EF oferă integrare lină între obiectul model din aplicație și baza de date din spate, oferind programatorilor oportunitatea de a lucra într-un mod mai intuitiv și orientat-obiect. Cu

EF, se pot defini modelele de date ca și clase, înființând relații între entități și să se facă operații pe baza de date folosind tehnici familiare orientate-obiect.

Entity Framework, de asemenea, manevrează sarcinile esențiale precum generarea de interogări de tip SQL, managerizarea conexiunilor cu baza de date, migrările bazei de date. Abstractizează complexitatea accesării datelor, reducând volumul de cod necesar pentru interacționarea cu baza de date. EF are suport pentru diferite interpretări, incluzând Code-First și Database-First, oferind Flexibilitate și adaptabilitate procesului de devlopare a GymJournal.

Folosind Microsoft SQL Server ca DBMS și folosind Entity Framework pentru gestionarea bazei de date, GymJournal asigură stocarea, procurarea și gestionarea eficiente a datelor utilizatorilor și a informației exercițiilor. Aceste tehnologii contribuie la integritatea datelor, performanța și scalabilitatea aplicației, asigurând o experiență lină și consistentă utilizatorilor.

**Dezvoltarea API-ului**

Dezvoltarea API (Application Programming Interface) joacă un rol crucial în tehnologiile *backend* ale aplicației, oferind posibilitatea de a comunica și interacționa lin între partea de client și server. GymJournal folosește ASP.NET Web API ca *framework* pentru construirea și expunerea unui set bine definit de *endpoints* care facilitează schimbul de date și accesul funcțional.

ASP.NET Web API[10] este un *framework* puternic oferit de Microsoft, proiectat în mod special pentru construirea de API-uri de tip REST. Urmărește principiile transferului reprezentativ de stări (REST), oferind o interpretare standard pentru crearea serviciilor web care pot fi consumate de diferiți clienți, incluzând navigatorul de internet, aplicații de tip mobile și alte sisteme de terță parte.

Utilizarea a ASP.NET Web API oferă mai multe beneficii pentru GymJournal:

1. Arhitectură de tip REST[11]: ASP.Net promovează adoptarea de arhitectură de tip REST, ceea ce evidențiază o abordare fară stări, orientată-resursă a design-ului de API. Urmărind principiile REST, GymJournal reușește să ajungă la un nivel înalt de interoperabilitate, scalabilitate și simplitate în design de API, facându-l mai ușor de consumat de către clienți și de lucrat cu endpoint-urile expuse.

2. HTTP *verbs* și *Status-Codes*: ASP.Net Web API folosește verbele HTTP standard (GET, POST, PUT, DELETE, etc.) pentru a reprezenta diferite operații pe resursele lui. De asemenea, utilizează codurile status HTTP pentru a oferi răspunsuri semnificative către client. Această aderență către convențiile HTTP asigură claritate și consistență în design-ul API-ului, facându-l mai intuitiv de înteles pentru programatori.
3. Serializarea și negocierea conținutului: ASP.NET Web API oferă suport inclus pentru negocierea conținutului, oferind clientului posibilitatea de a cere date în diferite formate (JSON, XML, etc.) bazat pe preferințele lor sau pe necesități. De asemenea, include serializare automată și deserializare automată pentru obiecte, simplificând procesul de conversie a datei între reprezentările din server și din client.
4. Integrarea de alte componente: ASP.NET Web API poate să integreze ușor cu alte componente din pipeline-ul ASP.NET Core. Acestea oferă posibilitatea de a incorpora funcționalități adiționale in GymJournal precum autentificare, logare, tratarea excepțiilor în procesul de developare al API-ului, îmbunătățind securitatea, performanța, și menținerea pe termen lung.
5. *Routing* și *Routing* pe bază de atribute: ASP.NET Web API oferă mecanisme flexibile de *routing* care oferă developerilor posibilitatea de a defini structura URL și rutele cererilor pentru a accesa acțiuni de tip controler. Are suport pentru ambele, *routing* bazat pe convenție, și *routing* pe bază de atribute, oferind GymJournal flexibilitatea de a alege o abordare care se potrivește cât mai bine design-ului API și organizării.
6. Versionare și Documentare: ASP.NET Web API are suport pentru a oferi versiuni API-urilor, oferind posibilitatea de a introduce modificări și îmbunătățiri API-ului menținând suportul pentru versiunile precedente (*backward compatibility*). În plus, variate unelte și *framework*-uri pot fi utilizate pentru generarea de documentație pentru API, făcând mai ușor pentru programatori înțeleagerea structuri API-ului, *endpoint*-urile disponibile, formaturile cerere/răspuns, și alte date relevante.

Folosind ASP.NET Web API, GymJournal asigură developarea unui API robust și bine structurat care asigură comunicarea lină dintre aplicația client și server. Adoptarea principiilor REST, aderarea la convențiile HTTP, suportul pentru negociere al conținutului și integrarea altor componente contribuie la crearea unui API eficient, scalabil și ușor de utilizat care îndeplinește cererile clienților API-ului.

**Autentificarea și autorizarea**

Autentificarea și autorizarea sunt aspecte vitale alte tehnologiilor pe partea de server pentru GymJournal, asigurând acces securizat la funcționalitățile aplicației și protecția datelor utilizatorilor. În această implementare, GymJournal folosește o autentificare bazată pe un sistem de *token*, proiectat *custom*, pentru autorizarea și autentificarea cererilor utilizatorilor.

Autentificarea bazată pe *token* se referă la schimbarea de *token*-uri digital semnate între partea de client și partea de server pentru a stabili și menține identitatea unui utilizator pe parcursul sesiunii lui. Această abordare oferă mai multe avantaje implementării, incluzând scalabilitatea, ușurința implementării și lipsa stărilor.

Autentificarea bazată pe *token* din cadrul aplicației este compusă din următoarele:

1. Logarea și înregistrarea unui utilizator: Sistemul permite utilizatorilor să își înregistreze conturile oferind informația necesară, cum ar fi numele și parola. Odată înregistrați utilizatorii pot să se conecteze în aplicație introducând credețialele de conectare. Serverul verifică credențialele și generează un *token* de acces unic pentru acel user.
2. Generarea de *token* și conectarea: Când un utlizator se conectează cu succes serverul generează un *token* de acces, pe care îl returnează înapoi clientului.
3. Validarea *tokenului*: Pentru cererile ulterioare, clientul va include *token*-ul de acces în cerere. Serverul va valida *token*-ul, verificând dacă *token*-ul din cerere este identic cu cel din baza de date. Dacă *token*-ul este valid, serverul oferă acces către funcționalitatea cerută.
4. Autorizarea bazată pe rol: Pe lânga autentificare, sistemul de *token* implementează autorizarea bazată pe rol. Fiecare utilizator are atribuit câte un rol care determină nivelul de acces la aplicație. De exemplu, un utilizator cu rolul de *Admin* are privilegii superioare unui utilizator cu rolul de *Regular*. Serverul verifică rolul utilizatorului și autorizează sau restricționează accesul la resurse specifice lui.
5. Ștergerea *token*-ului: Pentru a îmbunătăți securitatea și utilizarea, sitemul de *token* generează un token nou la o conectare nouă, *token*-ul vechi devenind neutilizabil. Astfel la o conectare pe un nou dispozitiv, este necesară reconectarea pe cel vechi.

Implementând la comandă sistemul de autentificare bazat pe *token* se oferă aplicației control reglat fin si bine definit asupra procesului de autentificare și autorizare. Oferă flexibilitate în definirea rolurilor utilizatorilor și în nivelurile de acces, asigurând că există măsurile de securitate potrivite pentru protejarea datelor utilizatorilor.

Dezvoltând și implementând *custom* sistemul de autentificare bazat pe *token*, GymJournal asigră acces la funcționalitățile lui și protejează datele utilizatorilor. Acestă abordare oferă flexibilitate, scalabilitate și control asupra procesului de autentificare și autorizare, aplicația putând oferi o experiență sigură și personalizată pentru clienții lui.

**Alegerea limbajului de programare pe partea de server**

Alegerea limbajului pe partea de server este o decize critică în dezvoltarea infrastructurii din spatelele aplicației. Limbajul de pe partea de server servește ca și unealta principală pentru procesarea cererilor, executarea logicii de business, generarea răspunsurilor. Pentru GymJournal alegerea limbajului pe partea de server este C#.

C# ( # fiind citit *sharp*) este un limbaj de programare modern, orientat-obiect, proiectat de către Microsoft. Este parte din frameworkul .NET, care oferă un set cuprinzător de librării și unelte pentru construirea de aplicații robuste și scalabile. C# oferă o multitudine de funcționalități care îl fac o alegere ideală pentru developarea părții de server a GymJournal:

1. Limbaj de programare orientat strict obiect: C# este un limbaj de programare static, ceea ce înseamnă că variabilele trebuie să aibe tipuri predefinite. Acestă funcționalitate asigură fiabilitatea codului și detectarea rapidă de erori de scriere. În plus, C# urmărește paradigma programării orientate-obiect, oferind programatorilor capabilitatea de a crea cod modular și reutilizabil prin incapsularea datelor și comportamentelor în clase și obiecte.
2. Integrarea cu *framework*-ul .NET: C# integrează în mod lin cu capabilitățile extensive oferite de *framework*-ul .NET. Acest *framework* oferă o gamă largă de librării, API-uri, și unelte care simplifică developarea sarcinilor comune, cum ar fi manevrarea fișierelor, manipularea datelor, networking-ul, securitatea. Folosind aceste funcționalități ale *framework*-ului .NET, GymJournal poate să accelereze dezvoltarea, să îmbunătățească productivitatea, și să beneficieze de un ecosistem robust.
3. Performanță și eficiență: C# este cunoscut pentru performață și eficiență. Limbajul de programare este compilat într-un limbaj intermediar care poate fi executat de către .NET *runtime*. *Runtime*-ul folosește compilare *just-in-time* pentru a converti codul din limbajul de programare intermediar în cod mașină în momentul rulării, îmbunătățind performanța. Procesul de compilare, împreună cu variatele tehinici de optimizare în

momentul rulării .NET, asigură executarea eficientă și performanța înaltă a codului de pe partea de server a GymJournal.

4. Funcționalități ale limbajului și productivitate: C# include o gamă largă de funcționalități ale limbajului care îmbunătățesc productivitatea programatorilor. Aceste funcționalități includ gestionare de memorie automată prin *garbage collection*, suport pentru programarea asincronă cu cuvintele *async*/*await*, LINQ (cereri pe limbaj integrat) pentru cereri și manipulare efieciente, are suport pentru paradigme de programare moderne, precum programarea funcțională.
5. Unelte și suportul din partea comunității: C# are o comunitate de developeri activă și un ecosistem de unelte, librării, *framework*-uri extins. Acestă comunitate activă oferă suport, resurse, și oportunități de colaborare, făcând foarte ușor a se găsi soluții, a accesa documentația, a fi la curent cu cele mai bune practici, în proiectarea aplicației. În plus, medii de programare populare precum Visual Studio, Visual Studio Code oferă IDE-uri puternice cu funcționalități precum completarea de cod, *debugging*, testarea, care îmbunătățesc experiența de dezvoltare.

Selectând C# ca limbajul de programare pentru partea de server a GymJournal, dezvoltarea aplicației beneficiază de robustețea limbajului, de productivitate, și de integrarea cu *framework*-ul .NET. C# oferă posibilitatea de a implementa în mod eficient componente pentru aplicația GymJournal precum logica de business, procesarea de date, *endpoint*-urile API-ului. Folosind capabilitățile extinse ale C#, și *framework*-ul .NET, GymJournal poate avea o infrastructură scalabilă și performantă pe partea de server pentru a îndeplini sarcinile utilizatorilor.

**Testare automată**

Testarea și asigurarea calității joacă un rol esențial în procesul de dezvoltare al oricărei aplicații comerciale, asigurând că aplicația funcționează în modul dorit și că îndeplinește așteptările utilizatorilor ei. În această implementare, GymJournal adoptă o suită de teste care sunt concentrate pe testarea la nivel de unitate folosind framework-ul de testare NUnit[14], împreună cu utilizarea framework-ului Moq[15] pentru crearea de obiecte false.

1. Testarea la nivel de unitate cu Nunit: GymJournal folosește NUnit ca *framework* de testare pentru a testa unități individuale de cod, precum metode și funcții, în mod izolat. Testele la nivel de unitate sunt scrise pentru a verifica comportamentul și corectitudinea

unor unități specifice, asigurându-se că ele produc rezultatul așteptat pentru diferite date de intrare și scenarii. NUnit oferă posibilitatea de a lucra cu un set bogat de afirmări, rulări de teste, fixuri de teste, oferind capabilitatea de a fi realizate teste la nivel de unitate expresive și care pot fi menținute.

2. Falsificarea obiectelor folosind Moq: Pentru a izola unități de cod pe durata testării la nivel de unitate și pentru a scoate dependințele pe resurse și sisteme externe, GymJournal folosește Moq, un *framework* specializat în falsificarea de obiecte pentru testele la nivel de unitate. Moq oferă posibilitatea de a fi create obiecte false care simulează un anumit comportament al unor dependițe din codul de testat, oferind control fin asupra scenariilor testate. Cu Moq, se poate defini acest comportament al dependințelor, modifica și verifica interacțiuni, facilitând eficacitatea testelor la nivel de unitate.

Testarea la nivel de unitate oferă mai multe beneficii pentru GymJournal:

- Detectarea rapidă a *bug*-urilor: Testele la nivel de unitate oferă GymJournal capabilitatea de a găsi erori și *bug*-uri devreme în decursul procesului de dezvoltare. Testând unități individuale în mod izolat, se pot identifica rapid și adresa probleme înainte de a se propaga în alte părți ale sistemului.
- Încrederea în cod și în schimbările din cod: Testele la nivel de unitate oferă o siguranță pentru schimbările din cod. Cu o suită de teste robustă, codul poate fi schimbat cu încredere, știind că picarea testelor va fi un semnal că au fost introduse defecte în cod.
- Documentare și întelegerea codului: Testele la nivel de unitate bine scrise servesc ca și o documentație pentru cod. Oferă exemple de cum unități de cod ar trebui folosite și pot ajuta în înțelegerea logicii complexe și cazurilor limită.
- Integrarea continuă și developarea: Testele la nivel de unitate se integrează ușor cu alte integrări sau cu alte componenete. Ele pot fi rulate automat cu fiecare schimbare de cod, oferind un *feedback* pentru integritatea codului și asigurând că doar codul de calitate ajunge în mediul de producție.

Folosind testarea la nivel de unitate cu NUnit și folosind Moq pentru falsificarea dependințelor, GymJournal reușește să îndeplinească un nivel înalt de calitate a codului, fiabilitate și menținere pe termen lung. O abordare riguroasă a testării ajută la identificarea și la rezolvarea problemelor din timp, îmbunătățește stabilitatea aplicației, inspiră încredere în cod.

# Secțiunea II.2 - Client

Partea de client a aplicației GymJournal joacă un rol crucial în livrarea unei experiențe line și intuitive a utilizatorilor. Acst capitol explorează tehnologiile folosite în developarea părții de client a aplicației, care este reprezentată de o aplicație mobile dezvoltată folosind *framework*-ul .NET MAUI (Multi-platform App UI) și care urmărește tiparul din Community Toolkit MVVM (Model-View-ViewModel)[12][13].

Partea de client a GymJournal se concentrează pe oferirea unei aplicații mobile care are multe funcționalități, care este plăcută din punct de vedere vizual, care rulează pe Android, cu posibilitatea de extindere și pe iOS, Android, Windows și macOS. Folosind capabilitățile *framework*-ului .NET MAUI și ale *framework*-ului Community Toolkit MVVM, GymJournal reușește să obțină un grad mare de capacitate de refolosire a codului, mentenanță pe termen lung, și compatibilitate cu alte platforme.

În acest capitol, vor fi acoperite variatele tehologii și abordări folosite în developarea aplcației pe partea de client. Vor fi enumerate și elaborate beneficiile folosirii .NET MAUI ca *framework* de bază, ceea ce oferă aplcației posibilitatea de a fi scrisă cu un singur cod și de a fi folosită pe mai multe platforme, eliminând nevoie dezvoltare într-un mediu specific. În plus, vor fi enunțate și avantajele adoptării tiparului MVVM împreună cu framework-ul Community Toolkit, care promovează separarea de responsabilități și facilitează dezvoltarea de cod simplu și testabil.

Mai departe, vor fi acoperite funcționalități și componente oferite de către .NET MAUI și de către Community Toolkit, precum legarea datei, navigarea, controalele din interfață, care dau posibilitate aplicației să devină o interfață cu utilizatorul plăcută la vedere și interactivă. Aceste tehnologii ajută aplicația să îmbunătățească interacțiunea cu utilizatorul, să eficientizeze sincronizarea datelor, și să ofere o experiență conistentă pe diferite platforme.

Folosind puterea *framework*-ului .NET MAUI și a *framework*-ului Community Toolkit MVVM, este asigurat că aplicația mobile livrează o experiență lină și plină de funcționalități utilizatorilor. Tehnologiile pe partea de client folosite în proiectarea a GymJournal pavează calea pentru o aplicație plăcută la vedere, scalabilă, ușor de menținut care servește toate nevoile în materie de notare a antrenamentelor pe Android.

**.NET MAUI**

.NET MAUI (Multi-platform App UI) servește ca *framework* de bază pentru partea de client a aplicației. El reprezintă versiunea evoluată a *framework*-ului Xamarin.Forms[16], proiectat pentru a simplifica dezvoltarea de aplicații mobile *cross-platform*. Folosind .NET MAUI, aplicația are capabilitatea semnificativă de a transfera cod pe un număr multiplu de platforme, incluzând Android, iOS, Windows și macOS, fără compromiterea elementelor native pentru fiecare platformă.

Există mai multe motive pentru care .NET MAUI reprezintă o alegere foarte bună în realizarea părții de client a aplicației:

1. Developarea *cross-platform*: .NET MAUI asigură capabilitatea aplicației de a construi o singură bază de cod care poate fi instalată pe mai multe platforme. Aceasta elimină nevoia de a depune eforturi separate de dezvoltare pentru fiecare platformă în parte, salvând timp și efort și în același timp asigurând consistență în funcționalitate și în experiența utilizatorilor pe toate platformele.
2. Interfață care arată nativ: Cu .NET MAUI, se pot crea aplicații mobile care livrează o interfață care seamănă cu cea nativă. *Framework*-ul oferă acces la un set bogat de controale și componente de tipul interfață-utilizator, oferind aplicației capabilitatea de a avea o interfață care să fie plăcută vizual, care să răspundă într-un mod receptiv la cererea utilizatorului și care să se integreze ușor cu alte elemente native din platformă. Utilizatorii aplicației se vor bucura de o imagine familiară pe dispozitivele lor individuale, îmbunătățind investirea utilizatorilor și utilitatea aplicației.
3. O singură bază de cod: Unul dintre avantajele majore ale *framework*-ului este abilitatea de a transfera o parte semnificativă de cod pe toate platformele. Aplicația are logică de bază, modele de date, reguli de business într-o singură bază de cod, reducând eforturile necesare pentru mentenanță și reducând redundanța. Această abordare, eficientizează procesul de developare, asigură consistență pe mai multe platforme, simplifică rezolvarea de erori și adăugarea de funcționalități noi.
4. Performanță îmbunătățită: .NET MAUI este proiectat să livreze cu performanță mare pe platforme variate. Folosește capacitățile native și optimizările existente pe fiecare platformă, rezultând în aplicații mobile rapinde. GymJournal beneficiază de un motor de *rendering* optimizat, asigurând tranziții, animații și interacții line, care contribuie la fluiditatea și la plăcerea experienței utilizatorului.

5. Ecosistem și unelte: .NET MAUI are suport într-un ecosistem extins și suport de unelte robuste. Aplicația poate să folosească un număr larg de pachete, componente și librării de terță parte care să îmbunătățească funcționalitatea și capacitățile aplicației. În plus, procesul de developare este eficientizat cu ajutorul IDE-urilor (medii de programare integrate) precum Visual Studio și Visual Studio Code, care oferă testare avansată, verificare avansată de erori, unelte de profilare.
6. Asigură viitorul aplicației: Adoptând .NET MAUI, GymJournal asigură faptul că aplicația va putea fi folosită o perioadă îndelungată. *Framework*-ul beneficiază de îmbunătățiri continue și de suport din partea celor de la Microsoft și a comunității *open-source*. GymJournal poate să profite de avantajul funcționalităților noi, optimizărilor de performanță, și îmbunătățirilor de platformă, în momentul în care vor fi disponibile, pe măsură ce avansează tehnologia mobile.

Folosind *framework*-ul .NET MAUI, GymJournal reușește să obțină un nivel ridicat de refolosință a codului, compatibilitate cross-platform, o interfață care seamănă cu cea nativă. Acestea fac ca aplicația să devină robustă, plăcută la vedere și să poată fi rulată ușor pe dispozitive variate. Ecosistemul cuprinzător în care trăiește *framework*-ul .NET MAUI facilitează și mai mult procesul de developare, facilitând aplicația să livreze o experiență de calitate înaltă utilizatorilor.

**Framework-ul Community Toolkit MVVM**

Framework-ul Community Toolkit joacă un rol în îmbunătățirea funcționalității și a experienței utilizatorului. Fiind o colecție *open-source* de controale, extensii, utilități, Community Toolkit oferă aplicației o abundență de resurse pentru a eficientiza dezvoltarea, a îmbunătăți experiența utilizatorului, și să accelereze viteza de producție.

Community Toolkit are mai multe componente care sunt folositoare:

1. Controale gata de utilizare: Community Toolkit oferă o gamă largă de controale gata de utilizat care pot fi folosite în cadrul aplicației pentru a îmbogăți interfața. Aceste controale includ: butoane, câmpuri pentru introducere de text, dialoguri, grafice etc. Acestea pot fi încorporate cu ușurință în cadrul aplicației, scăzând timpul și efortul de dezvoltare, în același timp asigurând consistența și calitatea vizuală a elementelor din interfață.

2. Naviagare și *routing*: Community Toolkit oferă componente de navigare și de *routing* care simplifică implementarea navigării din cadrul aplicației. Cu ajutorul lor pot fi definite căi de navigare, poate fi implementat ușor navigarea înapoi, și pot fi manevrate tranzițiile dintre diferite pagini și vederi fără efort. Acestea rezultă într-o experiență a utilizatorului lină și intuitivă, oferindu-le capabilitatea de a naviga prin funcționalitățile aplicației fără dificultăți.
3. Legarea datelor și validare: Aplicația profită de capabilitățile oferite de Community Toolkit de a lega datele. Legarea datelor oferă aplicației posibilitatea de e stabili conexiuni între modelele de date și interfața utilizatorului, asigurând că schimbările în date sunt văzute automat în interfață.
4. Comportamente și convertoare: *Framework*-ul Community Toolkit include comportamente și convertoare care extind funcționalitea elementelor din intergfață a aplicației. Comportamentele oferă aplicației capabilitatea de a atașa acțiuni adiționale sau interacțiuni de elementele din interfață, creând un aspect dinamic și interactiv al aplicației. Convetoarele facilitează transformarea datelor și formatarea, oferind aplicației flexibilitatea de a adapta prezentarea datelor în funcție de o anumită cerere.
5. Inovație condusă de comunitate: *Framework*-ul Community Toolkit este un produs al colaborării programatorilor și contribuitorilor din comunitatea .NET. Acestă abordare colaborativă asigură că aplicația beneficiază de îmbunătățiri frecvente și de funcționalități noi conduse de cunoștințele colective și experiența colectivă a comunității.

Folosind *Framework*-ul Community Toolkit, aplicația însușește un set bogat de controale, capacități de navigare, legare de date, funcții de validare, comportamente și convertoare. Acestea îmbunătățesc interfața utilizatorului și accelerează procesul de dezvoltare. În plus, colaborarea și invoarea adusă de natura unor componente ale unei comunități, asigură că aplicația rămâne la zi cu cele mai bune practici în materie de aplicații mobile și poate să se adapteze la așteptările utilizatorilor și la moda din industrie.

**Arhitectura MVVM**

Aplicația GymJournal urmărește tiparul arhitectural MVVM (Model-View-ViewModel) pe partea de client. MVVM este un șablon arhitectural folosit pe scală largă în developarea de aplicații mobile moderne care promovează separarea responsabilităților,

menținerea pe o durată îndelungată de timp și abilitatea de a fi testată. Folosind MVVM, aplicația GymJournal reușește să obțină o separare clară dintre date și model, interfață și logica de business, oferind o arhitectură modulară și scalabilă pe partea de client.

Câteva detalii și avantaje în folosirea tiparului MVVM:

1. Model: Modelele reprezintă datele și logica de business a aplicației. Ele incapsulează entitățile de date, serviciile și operațiile necesare pentru a manevra și procura date. Modelele oferă fundația pentru gestionarea și organizarea datelor aplicației, asigurând consistență și integritate. Aplicația conține pe partea de model servicii, cum ar fi cele care comunică cu serverul, dar și servicii care conțin logica de business.
2. View (vedere): Reprezintă interfața cu utilizatorul. Cuprinde elementele vizuale, diagramele și controalele cu care utilizatorii interacționează pentru a accesa funcționalitățile aplicației. În aplicație, stratul de vedere conține mai multe pagini, ecrane și componente cu care se poate interacționa, care afișează informația și înregistrează datele de intrare. Partea de vedere este responsabilă pentru încărcarea elementelor de natură grafică și pentru a răspunde la acțiunile utilizatorului, cum ar fi butoane sau gesturi.
3. ViewModel (vedere-model): Reprezintă stratul intermediar dintre cele două straturi prezentate mai devreme. Acționează ca și un pod care conectează modelele de date și interfața cu utilizatorul. Partea de vdere-model incapsulează logica de prezentare, transformarea de date, și gestionarea stărilor necesare pentru a popula vederea cu datele care trebuie prezentate și pentru a răspunde comunicării utilizatorului. Expune propietăți și comenzi de care o vedere se leagă, stabilind comunicarea lină și sincronizarea dintre vedere și data de dedesubt.
4. Legarea datelor: Unul dintre principiile de bază ale tiparului MVVM este legarea datelor, care facilitează sincronizarea automată a datei dintre vdere-model și vedere. Aplicația folosește capacitățile de legare a datelor oferite de *framework*-ul .NET MAUI pentru a stabili conexiuni dintre proprietățile din vedere-model și elementele de interfață din vedere. Aceasta asigură că orice schimbare din vedere-model este în mod automat reflectată în vedere și vice-versa, reducând nevoia de sincronizare manuală a datelor și de reîncarcări manuale ale interfeței.
5. Comenzi: Aplicația folosește comenzi în vedere-model care manevrează acțiunile utilizatorului declașate de elemente din interfață. Comenzile incapsulează logica și operațiile necesare când o anumită acțiune apare, cum ar fi o apăsare pe buton sau un

gest. Folosind comenzi, aplicația reușește să obțină separarea responsabilităților și menține o distincție clară dintre interfața cu utilizatorul și logica de business, rezultând în cod mai clar și care poate fi menținut mai mult timp.

6. Grad ridicat de testare și de menținere pe termen lung: Arhitectura MVVM promovează testarea și menținerea codului pe o durată mare de timp. Aplicația poate avea teste la nivel de unitate pentru un strat, independent ce celelalte, pentru a verifica corectitudinea logicii de business și transformările de date. În plus, separarea clară a responsabilităților și modularitatea din MVVM facilitează menținerea codului și îmbunătățirile pe viitor, deoarece modificările dintr-un start nu solicită schimbări extinse în celelate.

Adoptând tiparul arhitectural MVVM aplicația reușește să obțină o arhitectură pe partea de client modulară, ușor de menținut și de testat. Separarea responsabilităților dintre straturile Model, Vedere și Vedere-model promovează scalabilitate, flexibilitate și ușurință în refolosirea codului. GymJournal beneficiază de capacitățile de comenzi și legare a datei oferite de *framework*-ul .NET MAUI și de *framework*-ul Community Toolkit, oferind sincronizare a datei și interacțiuni line. Tiparul MVVM eficientizează eforturile de îmbunătățire și testare a codului și contribuie la calitatea și robustețea generală a aplicației.

# Capitolul III – Implementare

În acest capitol se vorbește despre detaliile de implementare pentru aplicația GymJournal, concentrându-se pe ambele, pe partea de server, cât și pe partea de client. Detaliile de implementare oferă lămuriri despre cum diverse tehnologii și *framework*-uri discutate în capitolul precedent sunt folosite să aducă viață aplicației. Examinând implementările pe partea de client și pe partea de server, se poate obține o înțelegere cuprinzătoare despre cum funcționalitățile aplicației sunt construite și integrate.

Detaliile de implementare sunt împărțite în două mari părți, implementarea pe partea de server și implementarea pe partea de client. Fiecare parte explorează modul în care au fost folosite tehnologii relevante, unelte, tehinici pentru a dezvolta aplicația GymJournal. Vor fi discutate detalii precum implementarea gestionării bazei de date, developarea API-ului, autentificarea și autorizarea.

Mai departe, la implementarea părții de client, vor fi prezentate părți de cod, exemple și explicări care ilustrează cum aplicația folosește aceste tehnologii să livreze funcționalitățile de bază. Analizând detaliile de implementare cititorii vor obține informații importante despre procesul de developare al aplicației.

Acest capitol reprezintă ca o punte dintre conceptele teoretice discutate mai devreme și realizarea practică a aplicației. Arată cum tehnologiile alese mai devreme funcționează într-un context practic, și le demosntrează eficiența în crearea unei aplicații reușite de notat antrenamentele.

## Secțiunea III.1 – Server

Capitolul de detalii de implementare pe partea de server al acestei lucrări de licență explorează cum tehnologiile cheie și *framework*-urile au fost folosite în dezvoltarea componentelor de pe partea de server a aplicației. Acest capitol cuprinde detalii de implementare pe partea de server care explică cum tehnologiile alimentează infrastructura părții din spate a aplicației, cum oferă capabilitatea aplicației să stocheze, proceseze și să managerizeze într-un mod cât mai eficient datele utilizatorilor și datele antrenamentelor.

Capitolul începe prin a discuta hostarea serverului pe un container de Docker[17]. Docker oferă o platformă ușor de utilizat și eficientă pentru împachetarea și instalarea aplicațiilor, oferind instalare lină pe mai multe medii. Vor fi analizate containerele de docker utilizate și dependințele dintre ele.

În continuare, vom vorbi despre gestionarea bazei de date, despre cum este folosit Entity Framework în accesarea, manevrarea și stocarea datelor în baza de date. Se va mai vorbi și despre migrări și despre configurarea și inițializarea bazei de date.

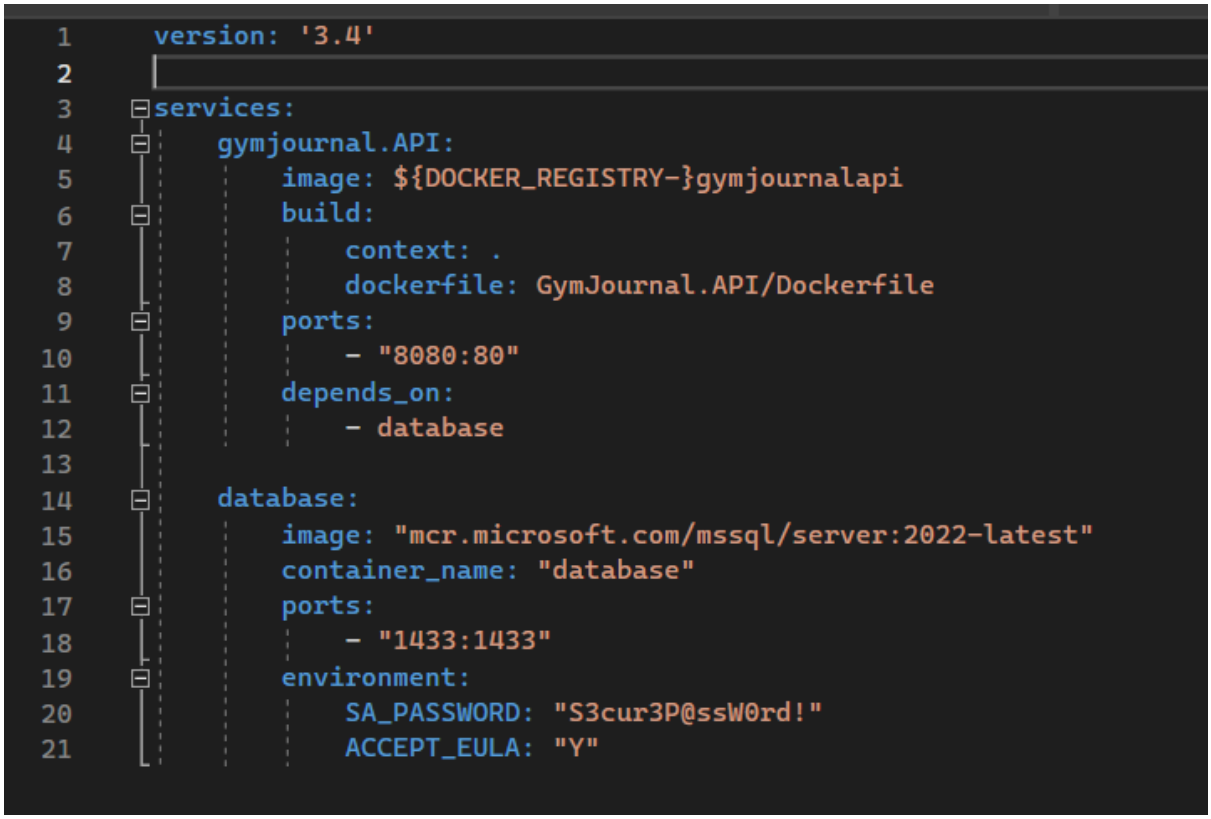
După care capitolul explorează mai departe developarea API-ului. Va fi explicat cum GymJournal folosește ASP.NET Web API ca să expună un set de bine definit de *endpoint*-uri care facilitează comunicarea dintre aplicație și server. Va fi discutat cum acest API manevrează autentificarea utilizatorilor, procurarea datelor legate de exerciții, stocarea datelor și alte operații care sunt necesare pentru a asigra o comunicare lină și integrare fără efort dintre componentele client și server ale GymJournal.

Mai departe vor fi discutate servicii care sunt utilizate în procesarea si manipularea datelor. Vor fi discutate cum au fost declarate, apelate și cum funcționează serviciul de procesare al erorilor, serviciul care comunică cu baza de date.

În final vor fi discutate cum partea de server a aplicației realizează autentificarea și autorizarea utilizatorului. Va fi urmărit cum este realizat generarea *token*-ului, cum sunt verificate cererile de la clienți.

**Hostarea folosind docker**

Aplicația pe partea de server poate fi rulată folosind Docker compus fie cu o comandă în Windows Powershell (*docker-compose up* la care se adaugă numele și calea către fișierul *docker-compose.yml* din interiorul proiectului), fie folosind Visual Studio, selectând proiectul de Docker ca fiind proiectul de start și rulând aplicația. Proiectul de Docker creează două containere, unul folosind o imagine de Microsoft SQL Server, pe care este hostată baza de date la portul 1433 și un container care reprezintă API-ul pe partea de server care este hostat la portul 8080 (Vezi *figura 1.a*).

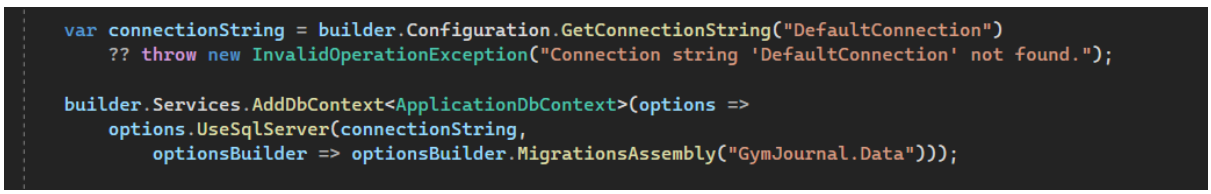


```yaml
version: '3.4'

services:
    gymjournal.API:
        image: ${DOCKER_REGISTRY-}gymjournalapi
        build:
            context: .
            dockerfile: GymJournal.API/Dockerfile
        ports:
            - "8080:80"
        depends_on:
            - database

    database:
        image: "mcr.microsoft.com/mssql/server:2022-latest"
        container_name: "database"
        ports:
            - "1433:1433"
        environment:
            SA_PASSWORD: "S3cur3P@ssW0rd!"
            ACCEPT_EULA: "Y"
```

*Figura 1.a*

**Gestionarea lucrului cu baza de date**

Pentru a se face legătura dintre API și baza de date sunt necesare următoarele linii de cod din Program.cs, care creează șirul de caractere care conține detaliile conexiunii cu baza de date (Vezi *figura 2.a*). Șirul de caractere este declarat în fișierul AppSettings.json. (Vezi *figura 2.b*)



```
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection")
    ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString,
        optionsBuilder => optionsBuilder.MigrationsAssembly("GymJournal.Data")));
```

*Figura 2.a*


*Figura 2.b*

În urma liniilor de cod din imaginea de mai sus (Vezi *figura 2.a*) este creat în cadrul serviciului aplicației contextul bazei de date, folisindu-se de migrările respective (Vezi *figura 3.a*), de asemenea, la crearea bazei de date sunt aplicate și configurări pe entități, cum ar fi următoarea care adaugă intrări de tipul *muschi* în tabela cu același nume (Vezi *figura 3.b*).

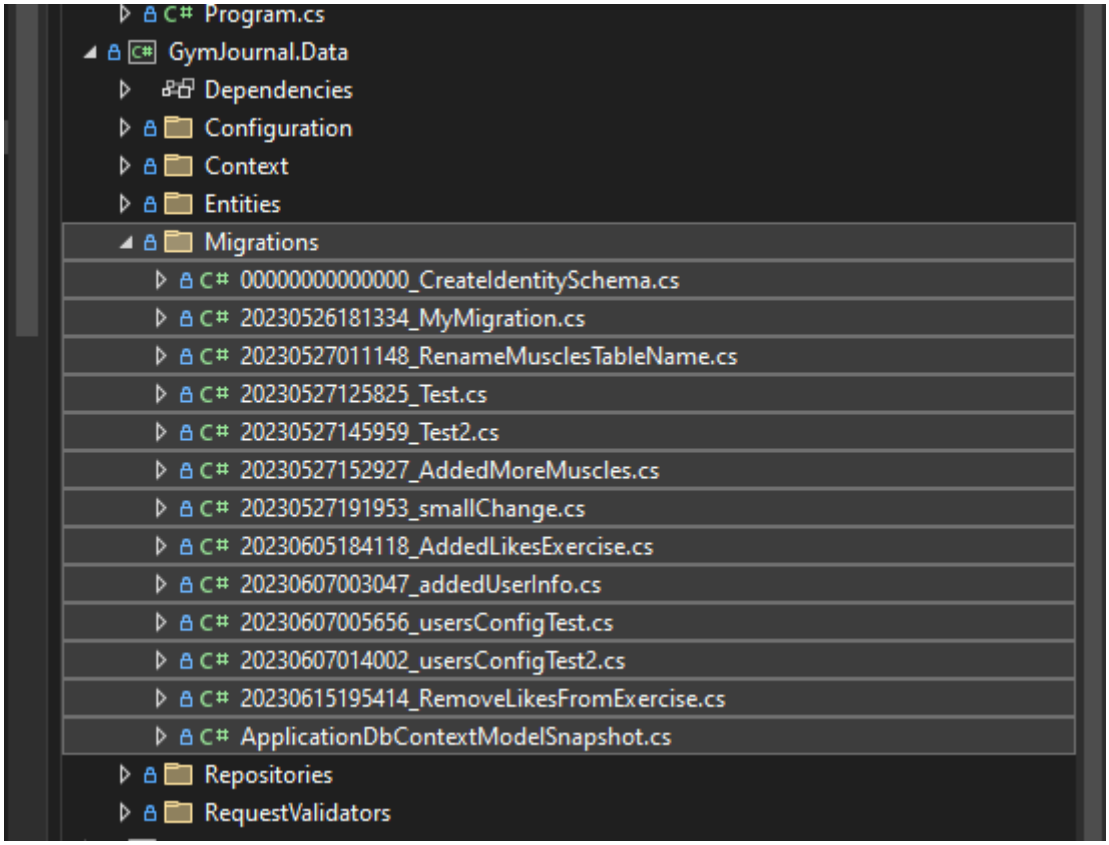
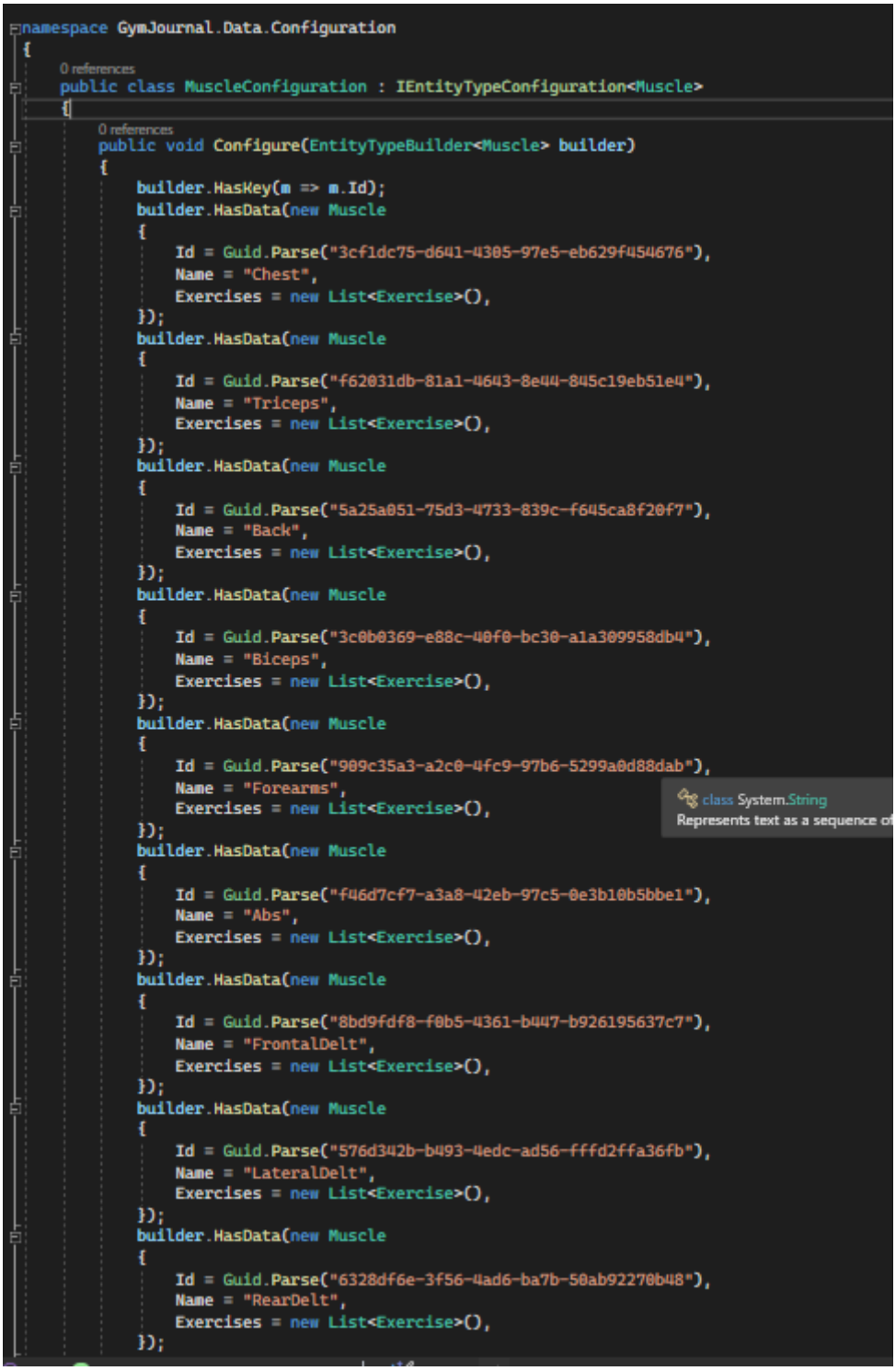



*Figura 3.a*



```
namespace GymJournal.Data.Configuration
{
    0 references
    public class MuscleConfiguration : IEntityTypeConfiguration<Muscle>
    {
        0 references
        public void Configure(EntityTypeBuilder<Muscle> builder)
        {
            builder.HasKey(m => m.Id);
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("3cf1dc75-d641-4305-97e5-eb629f454676"),
                Name = "Chest",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("f62031db-81a1-4643-8e44-845c19eb51e4"),
                Name = "Triceps",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("5a25a051-75d3-4733-839c-f645ca8f20f7"),
                Name = "Back",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("3c0b0369-e88c-40f0-bc30-a1a309958db4"),
                Name = "Biceps",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("909c35a3-a2c0-4fc9-97b6-5299a0d88dab"),
                Name = "Forearms",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("f46d7cf7-a3a8-42eb-97c5-0e3b10b5bbe1"),
                Name = "Abs",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("8bd9fdf8-f0b5-4361-b447-b926195637c7"),
                Name = "FrontalDelt",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("576d342b-b493-4edc-ad56-fffd2ffa36fb"),
                Name = "LateralDelt",
                Exercises = new List<Exercise>(),
            });
            builder.HasData(new Muscle
            {
                Id = Guid.Parse("6328df6e-3f56-4ad6-ba7b-50ab92270b48"),
                Name = "RearDelt",
                Exercises = new List<Exercise>(),
            });
```

class System.String
Represents text as a sequence of

*Figura 3.b*

După ce baza de date a fost *host*-ată, declarată, inițializată și configurată, aplicația poate să lucreze direct cu entitățile declarate în contextul bazei de date, deși nu este recomandat. Prin urmare, aplicația are mai multe servicii care se ocupă cu manipularea datelor, fără a expune modul în care acestea sunt stocate către exterior.

**Servicii care lucrează cu baza de date**

Serviciile care lucrează cu baza de date acționează ca un fel de intermediar. În loc ca serverul să expună entitățile, așa cum sunt ele declarate în contextul bazei de date, serverul expune un set de cereri și un set de răspunsuri care sunt folosite la comunicare (Vezi *figura 4.c* – cerere de ștergere a unui exercițiu) care joacă rol de abstractizare a datelor.

Serviciile primesc o astfel de cerere, execută ce au de executat (de exemplu ștergere de exercițiu – Vezi *figura 4.b*) și returnează un răspuns.

Pe lângă serviciile care comunică cu contextul bazei de date și modifică date, mai sunt și servicii care au alte scopuri, de exemplu serviciile care se ocupă cu validarea unei cereri, și serviciul care se ocupă cu răspunsul în cazul în care cererea nu este validă. Aceste servicii, și celelate de mai devreme, sunt adăugate în *Program.cs* folosind *Dependency Injection*[18] (Vezi *figura 4.a*).

```
builder.Services.AddScoped<IExerciseRepository, ExerciseRepository>();
builder.Services.AddScoped<IWorkoutRepository, WorkoutRepository>();
builder.Services.AddScoped<IWorkoutPlanRepository, WorkoutPlanRepository>();
builder.Services.AddScoped<IMuscleRepository, MuscleRepository>();
builder.Services.AddScoped<IUserInfoRepository, UserInfoRepository>();

builder.Services.AddScoped<IApplicationDbContext, ApplicationDbContext>();
builder.Services.AddScoped<ApplicationDbContextInitializer>();


builder.Services.AddScoped<IValidationAuthorization, ValidationAuthorization>();
builder.Services.AddScoped<ExceptionHandler>();


builder.Services.AddScoped<IMuscleValidators, MuscleValidators>();
builder.Services.AddScoped<IExerciseValidators, ExerciseValidators>();
builder.Services.AddScoped<IWorkoutValidators, WorkoutValidators>();
builder.Services.AddScoped<IWorkoutPlanValidators, WorkoutPlanValidators>();
builder.Services.AddScoped<IUserInfoValidators, UserInfoValidators>();
```

*Figura 4.a*

```
2 references
public async Task Delete(DeleteExerciseCommand command, CancellationToken cancellationToken = default)
{
    var entity = await _dbContext.Exercises
        .Include(e => e.Muscles)
        .Include(e => e.Workouts)
        .FirstOrDefaultAsync(e => e.Id == command.ExerciseId, cancellationToken);

    if (entity == null)
    {
        throw new BadRequestException("The exercise you want to delete does not exist.");
    }

    _dbContext.Exercises.Remove(entity);
    await SaveChanges(cancellationToken);
}
```

*Figura 4.b*

```
namespace GymJournal.Domain.Commands.ExerciseCommands
{
    5 references
    public class DeleteExerciseCommand
    {
        1 reference
        public Guid UserId { get; set; }
        1 reference
        public Guid UserToken { get; set; }
        1 reference
        public Guid ExerciseId { get; set;}
    }
}
```

*Figura 4.c*

În imaginea de mai sus (Vezi *figura 4.b*) este prezentată o porțiune de cod care se ocupă cu ștergerea unui exercițiu. Este primit ca parametru de intrare o comandă de tipul *delete exercise*. Entitatea este căutată în baza de date și apoi este ștearsă. La final sunt salvate modificările.

**Servicii care validează cererile**

Pe lângă serviciile care se ocupă cu completarea unei cereri sunt și servicii care se ocupă cu validarea lor. În exemplul următor (Vezi *figura 5.a*) este expusă validarea cererii de ștergere a unui exercițiu, care verifică validitatea cererii și apoi identitatea utilizatorului (Vezi *figura 5.b*) și aruncă excepții *custom*.

```
2 references
public async Task Validate(DeleteExerciseCommand command)
{
    if (command == null)
    {
        throw new InvalidRequestException("null command");
    }

    await _validationAuthorization.ValidateRegularUser(command.UserId, command.UserToken);
}
```

*Figura 5.a*

```
22 references
public async Task ValidateRegularUser(Guid UserId, Guid UserToken)
{
    var user = await _dbContext.UserInfos.FirstOrDefaultAsync(u => u.Id == UserId);

    if (user == null)
    {
        throw new UnauthorizedAccessException("invalid UserId");
    }

    if (user.Token != UserToken)
    {
        throw new UnauthorizedAccessException("invalid UserToken");
    }
}
```

*Figura 5.b*

**Construirea API-ului**

În arhitectura .NET API-urile sunt construite pe bază de *controller*-e. Folosind următoarea linie de cod (Vezi *figura 6.a*) sunt adăugate *controller*-e în aplicație. După aceea *routing*-ul se face, decorând *controller*-ul în felul următor (Vezi *figura 6.b*) și metoda (Vezi *figura 6.c*). După aceea sunt deschise în mod automat *endpoint*-urile cu rutele respective, și la solicitarea unuia, este apelată metoda care corespunde rutei.

```
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
```

*Figura 6.a*

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

*Figura 6.b*

```
[HttpPost("Delete")]
0 references
public async Task<IActionResult> Delete([FromBody] DeleteExerciseCommand command)
{
    try
    {
        await _exerciseValidators.Validate(command);

        await _exerciseRepository.Delete(command);

        return StatusCode(StatusCodes.Status204NoContent);
    }
    catch (Exception ex)
    {
        return _exceptionHandler.HandleException(ex);
    }
}
```

*Figura 6.c*

Tot aici (Vezi *figura 6.c*) putem observa *flow*-ul general pe care îl are o cerere: primire, validare, executare și apoi returnare de răspuns. De cele mai multe ori răspunsul conține codul de status 200 (pozitiv), dar în funcție de caz, poate fi și un status negativ.

### Serviciul care returnează răspunsuri negative

Serviciul care returnează răspunsuri negative se ocupă cu răspunsurile în cazurile în care cererea nu poate fi îndeplinită. Fie din cauza bazei de date, acces neautorizat sau acțiuni fără sens, serviciul returnează un răspuns negativ, în oricare dintre eceste excepții (Vezi *figura 8.a*).

```
11 references
public class ExceptionHandler
{
    23 references
    public IActionResult HandleException(Exception ex)
    {
        if (ex is UnauthorizedAccessException)
        {
            var errorResponse = new ErrorResponse
            {
                Message = ex.Message,
            };

            return new ObjectResult(errorResponse)
            {
                StatusCode = StatusCodes.Status401Unauthorized
            };
        }
        else if (ex is BadRequestException)
        {
            var errorResponse = new ErrorResponse
            {
                Message = ex.Message,
            };

            return new ObjectResult(errorResponse)
            {
                StatusCode = StatusCodes.Status400BadRequest
            };
        }
        else if (ex is InvalidRequestException)
        {
            var errorResponse = new ErrorResponse
            {
                Message = ex.Message,
            };

            return new ObjectResult(errorResponse)
            {
                StatusCode = StatusCodes.Status405MethodNotAllowed
            };
        }
```

*Figura 8.a*

**Sistemul de logare cu *token***

În momentul în care serverul primește o comandă de conectare (Vezi *figura 9.a*), el generează un *token* random, care este returnat ca răspuns. Același lucru se întâmplă și la înregistrare și la editarea contului. În momentul editării și în momentul înregistrării serverul stochează în baza de date un *hash* al parolei, nu parola în clar (Vezi *figura 9.b*).

```
2 references
public async Task<LoginUserInfoResponse> Login(LoginUserInfoQuery query, CancellationToken cancellationToken = default)
{
    var entity = await _dbContext.UserInfos
        .FirstOrDefaultAsync(e => e.Name == query.UserName, cancellationToken);

    if (entity == null)
    {
        throw new BadRequestException("The userInfo you want to Login does not exist.");
    }

    if (!BCryptNet.Verify(query.UserPassword, entity.Password))
    {
        throw new UnauthorizedAccessException("Invalid password for login.");
    }

    entity.Token = Guid.NewGuid();
    await SaveChanges(cancellationToken);

    return new LoginUserInfoResponse
    {
        UserId = entity.Id,
        UserToken = entity.Token,
        UserRole = entity.Role,
    };
}
```

*Figura 9.a*

```
2 references
public async Task<LoginUserInfoResponse> Login(LoginUserInfoQuery query, CancellationToken cancellationToken = default)
{
    var entity = await _dbContext.UserInfos
        .FirstOrDefaultAsync(e => e.Name == query.UserName, cancellationToken);

    if (entity == null)
    {
        throw new BadRequestException("The userInfo you want to Login does not exist.");
    }

    if (!BCryptNet.Verify(query.UserPassword, entity.Password))
    {
        throw new UnauthorizedAccessException("Invalid password for login.");
    }

    entity.Token = Guid.NewGuid();
    await SaveChanges(cancellationToken);

    return new LoginUserInfoResponse
    {
        UserId = entity.Id,
        UserToken = entity.Token,
        UserRole = entity.Role,
    };
}
```

*Figura 9.b*

# Secțiunea III.3 – Client

Partea de client a aplicației GymJournal joacă un rol crucial în livrarea unei experiențe line și intuitive a utilizatorilor. Acest capitol explorează în detaliu cum sunt folosite tehnologiile în dezvoltarea părții de client a aplicației, care reprezintă o aplicație mobile folosind *framework*-ul .NET MAUI (Multi-platform App UI) și care urmărește tiparul din Community Toolkit MVVM (Model-View-ViewModel).

În acest capitol, vor fi acoperite variatele detalii de implementare a tehologiilor și abordările folosite în developarea aplcației pe partea de client. Vor fi prezentate elemente ale tiparului MVVM folosite în cadrul aplicației. De asemenea for fi expuse părți de cod care vor prezenta diverse comenzi pentru apăsarea de butoane, introducere de text.

Tot aici, vor fi prezentate diferite servicii din cadrul aplicației, care se ocupă cu diverse sarcini, cum ar fi comunicarea cu partea de sever, păstrarea profilului utilizatorului curent.

**Comunicarea cu serverul**

Comunicarea cu serverul se face prin intermediul unor servicii care sunt declarate în *MauiProgram.cs* folosind *Dependency Injection* (Vezi *figura 10.a*). După aceea ele sunt accesibile de peste tot din aplicație și pot fi folosite pentru a cere date de la server (Vezi *figura 10.b*).

```
builder.Services.AddScoped<IExerciseService, ExerciseService>();
builder.Services.AddScoped<IMuscleService, MuscleService>();
builder.Services.AddScoped<IUserInfoService, UserInfoService>();
builder.Services.AddScoped<IWorkoutPlanService, WorkoutPlanService>();
builder.Services.AddScoped<IWorkoutService, WorkoutService>();
```

*Figura 10.a*

```
2 references
public async Task Delete(Guid id)
{
    HttpClient httpClient = new HttpClient();

    UriBuilder builder = new UriBuilder(_constantsService.HostAddress)
    {
        Path = "/Exercise/Delete"
    };
    var url = builder.Uri.ToString();

    var command = new DeleteExerciseCommand
    {
        UserId = _identityService.UserId,
        UserToken = _identityService.UserToken,
        ExerciseId = id,
    };

    HttpContent content = new StringContent(JsonSerializer.Serialize(command), Encoding.UTF8, "application/json");

    var response = await httpClient.PostAsync(url, content);

    if (!response.IsSuccessStatusCode)
    {
        throw new ServerRequestException(await response.Content.ReadAsStringAsync());
    }
}
```

*Figura 10.b*

Tot în imaginea de mai sus (Vezi *figura 10.b*) putem observa modul în care este realizată o cerere către server. Întâi se declară un obiect de tipul *HttpClient,* după care se creează adresa la care este făcută cererea, după care se creează corpul cererii și se trimite către server. După aceea răspunsul este manevrat și returnat, dacă e cazul.

**MVVM**

În continuare este prezentat ca exemplu cazul unei apăsări pe buton pentru a demonstra folosirea tiparului MVVM. În imagine (Vezi *figura 11.a*) putem observa elementele de pe pagina de *login*, și felul în care sunt poziționate. Sunt câmpuri pentru introdus numele de utilizator și parola, etichete care identifică câmpurile, și locuri pentru a afișa mesajele de validare. La final avem un puton pentru a trmite formularul mai departe.

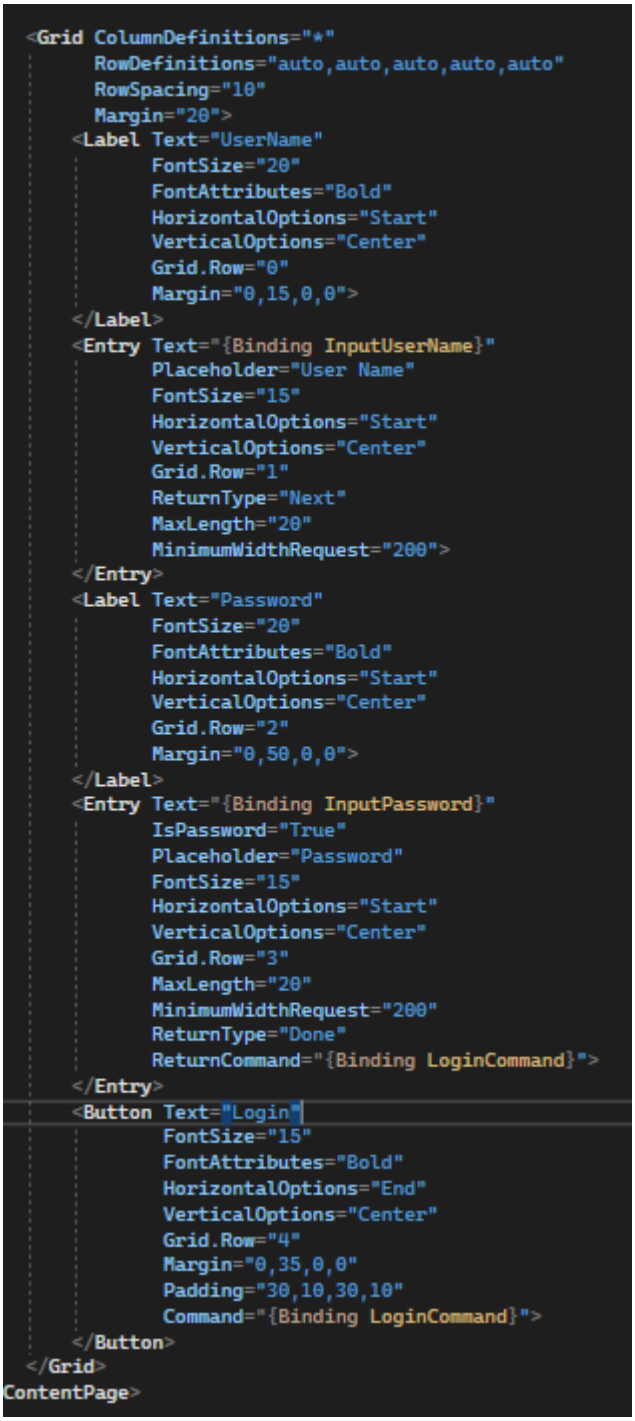
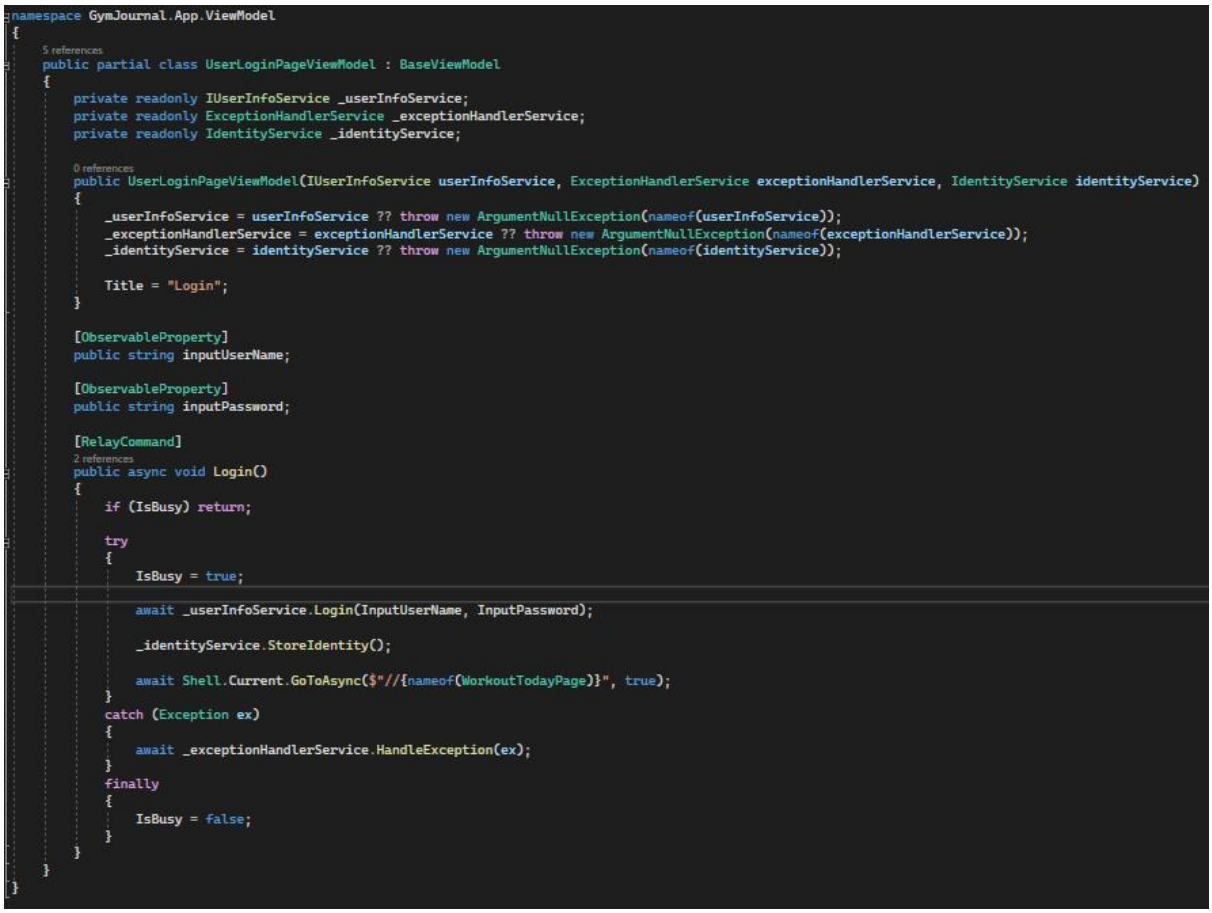



*Figura 11.a*

Formularul este întâi trimis către obiectul de care este legat acest *ContentPage* și anume modelul de vedere al paginii de *login* (Vezi *figura 11.b*), unde avem câmpuri pentru fiecare intrare marcate ca fiind tip obiect observabil, tip care va fi detaliat în următoarea secțiune. Dacă

pagina nu se ocupă cu nici o altă cerere în acest moment, atunci facem o cerere de login, salvăm informația legată de identitate local și navigăm la o altă parte a aplicației.



```
namespace GymJournal.App.ViewModel
{
    5 references
    public partial class UserLoginPageViewModel : BaseViewModel
    {
        private readonly IUserInfoService _userInfoService;
        private readonly ExceptionHandlerService _exceptionHandlerService;
        private readonly IdentityService _identityService;

        0 references
        public UserLoginPageViewModel(IUserInfoService userInfoService, ExceptionHandlerService exceptionHandlerService, IdentityService identityService)
        {
            _userInfoService = userInfoService ?? throw new ArgumentNullException(nameof(userInfoService));
            _exceptionHandlerService = exceptionHandlerService ?? throw new ArgumentNullException(nameof(exceptionHandlerService));
            _identityService = identityService ?? throw new ArgumentNullException(nameof(identityService));

            Title = "Login";
        }

        [ObservableProperty]
        public string inputUserName;

        [ObservableProperty]
        public string inputPassword;

        [RelayCommand]
        2 references
        public async void Login()
        {
            if (IsBusy) return;

            try
            {
                IsBusy = true;

                await _userInfoService.Login(InputUserName, InputPassword);

                _identityService.StoreIdentity();

                await Shell.Current.GoToAsync($"//{nameof(WorkoutTodayPage)}", true);
            }
            catch (Exception ex)
            {
                await _exceptionHandlerService.HandleException(ex);
            }
            finally
            {
                IsBusy = false;
            }
        }
    }
}
```

*Figura 11.b*

## Community Toolkit MVVM

Decorările din funcțiile de mai devreme (de exmplu obiect observabil) au legătură cu integrarea *framework*-ului Community Toolkit MVVM. Acest *framework* generează automat cod pentru variabilele respective care să se ocupe cu diverse sarcini generice, cum ar fi *update*-urile din interfață a unei valori schimbate în memorie. Codul generat poate fi găsit la categoria dependințe (Vezi *figurile 12.a și 12.b*).

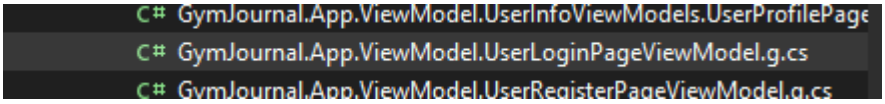



*Figura 12.a*

```
// <auto-generated/>
#pragma warning disable
#nullable enable
namespace GymJournal.App.ViewModel
{
    /// <inheritdoc/>
    5 references
    partial class UserLoginPageViewModel
    {
        /// <inheritdoc cref="inputUserName"/>
        [global::System.CodeDom.Compiler.GeneratedCode("CommunityToolkit.Mvvm.SourceGenerators.ObservablePropertyGenerator", "8.2.0.0")]
        [global::System.Diagnostics.CodeAnalysis.ExcludeFromCodeCoverage]
        9 references
        public string InputUserName
        {
            get => inputUserName;
            [global::System.Diagnostics.CodeAnalysis.MemberNotNull("inputUserName")]
            set
            {
                if (!global::System.Collections.Generic.EqualityComparer<string>.Default.Equals(inputUserName, value))
                {
                    OnInputUserNameChanging(value);
                    OnInputUserNameChanging(default, value);
                    OnPropertyChanging(global::CommunityToolkit.Mvvm.ComponentModel.__Internals.__KnownINotifyPropertyChangingArgs.InputUserName);
                    inputUserName = value;
                    OnInputUserNameChanged(value);
                    OnInputUserNameChanged(default, value);
                    OnPropertyChanged(global::CommunityToolkit.Mvvm.ComponentModel.__Internals.__KnownINotifyPropertyChangedArgs.InputUserName);
                }
            }
        }

        /// <inheritdoc cref="inputPassword"/>
        [global::System.CodeDom.Compiler.GeneratedCode("CommunityToolkit.Mvvm.SourceGenerators.ObservablePropertyGenerator", "8.2.0.0")]
        [global::System.Diagnostics.CodeAnalysis.ExcludeFromCodeCoverage]
        9 references
        public string InputPassword
        {
            get => inputPassword;
            [global::System.Diagnostics.CodeAnalysis.MemberNotNull("inputPassword")]
            set
            {
                if (!global::System.Collections.Generic.EqualityComparer<string>.Default.Equals(inputPassword, value))
                {
                    OnInputPasswordChanging(value);
                    OnInputPasswordChanging(default, value);
                    OnPropertyChanging(global::CommunityToolkit.Mvvm.ComponentModel.__Internals.__KnownINotifyPropertyChangingArgs.InputPassword);
                    inputPassword = value;
                    OnInputPasswordChanged(value);
                    OnInputPasswordChanged(default, value);
                    OnPropertyChanged(global::CommunityToolkit.Mvvm.ComponentModel.__Internals.__KnownINotifyPropertyChangedArgs.InputPassword);
                }
            }
        }

        /// <summary>Executes the logic for when <see cref="InputUserName"/> is changing.</summary>
```

*Figura 12.b*

# Capitolul IV – Manual

În acest capitol vor fi prezentate pe scurt diverse funcționalități ale aplicației evidențiind pașii necesari pentru a utiliza funcțiile din aplicație. Vor fi prezentați în primul rând pașii necesari pentru crearea unui cont în aplicație și după aceea vor fi prezentați pașii necesari pentru adăugarea unui exercițiu nou.

### Înregistrarea unui cont nou

Pentru a se crea un cont nou în aplicație trebuie ca utilizatorul să nu se mai fi conectat, sau să se fi deconectat, folosind butonul cel mai de jos din meniul principal al aplicației. În imagine (Vezi *figura 13.a*) este meniul de identificare al aplicației. Pentru a te înregistra în aplicație, trebuie apăsat butonul de înregistrare și apoi introduse datele necesare în formularul de înregistrare (Vezi *figura 13.b*) și apoi, după înregistrare, utilizatorul este redirecționat la pagina principală a aplicației.

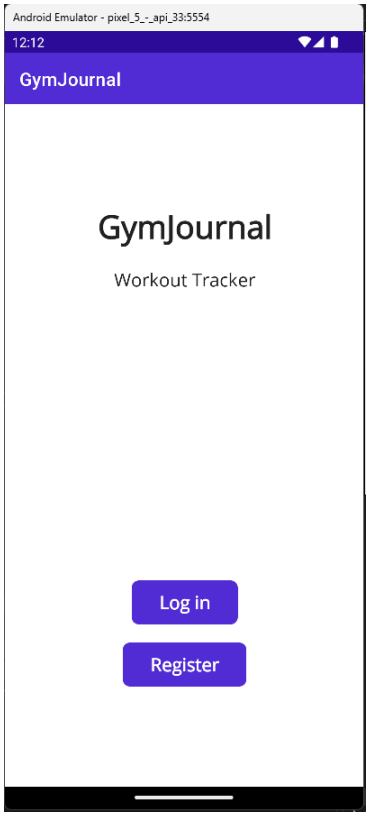



*Figura 13.a*

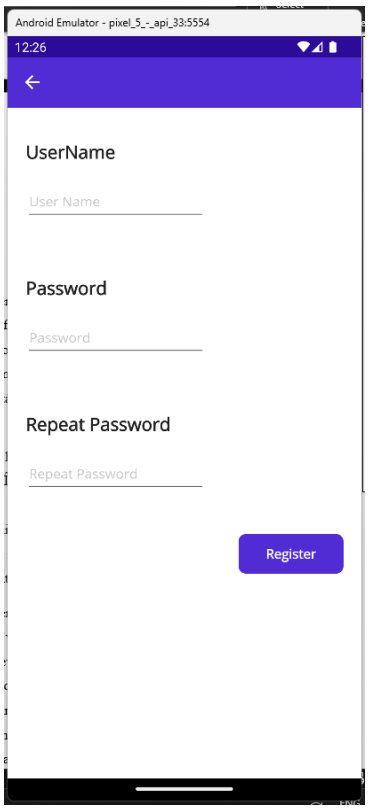



*Figura 13.b*

**Adăugarea unui exercițiu nou**

Pentru adăugarea unui exercițiu nou, din pagina principală se navighează către pagina de *Explore* la secțiunea *Exercise* (Vezi *figura 14.a*), de acolo se apasă butonul de adăugare exercițiu, care deschide un formular (Vezi *figura 14.b*). Acolo se introduc datele necesare pentru exercițiu și se aleg muschii lucrați de exercițiu din listă, după care se trimite formularul către server și utilizatorul este redirecționat înapoi la pagina de explorare exerciții.

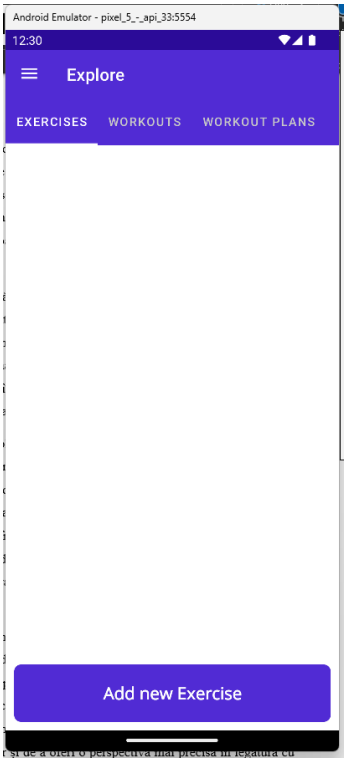



*Figura 14.a*

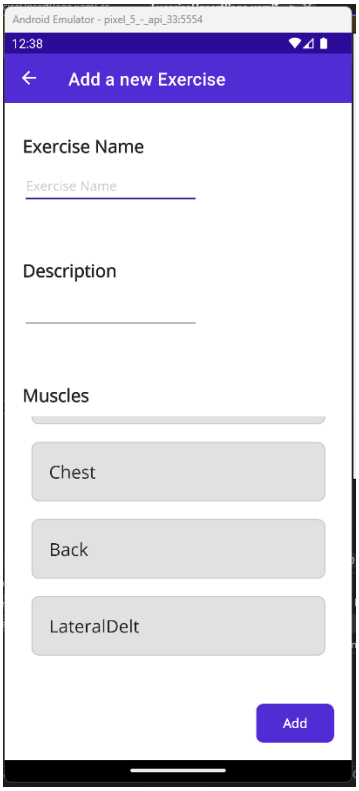



*Figura 14.b*

# Capitolul V – Concluzie

De-a lungul acestei lucrări de licență, au fost prezentate proiectarea și implementarea aplicației GymJournal, o aplicație de fitness pentru urmăriea antrenamentelor. Au fost examinate aspecte variate, incluzând scopul aplicației și obiectivele, cererile utilizatorilor, arhitectura sistemului, detalii de implementare. Folosind o gamă de tehnologii, *framework*-uri, și bune practici, aplicația a fost dezvoltată cu succes, reprezentând o soluție pentru utilizatori să își noteze antrenamentele.

Cercetarea prezentată în această lucrare de licență face contribuții semnificative în domeniul aplicațiilor pentru antrenament. În primul rând, aplicația adresează nevoia unei soluții cuprinzătoare și ușor de folosit de oricine pasionat de fitness pentru a monitoriza și a gestiona antrenamentele. Incorporând funcționalități precum notarea exercițiilor, aplicația ajută utilizatorii să își urmărească progresul în materie de sport, să își seteze obiective realistice și să facă decizii informate în legătură cu antrenamentul.

În al doilea rând, detaliile de implementare prezentate de-a lungul lucrării, arată cum au fost utilizate tehnologii si *framework*-uri variate. Pe parte de server, am explorat *framework*-ul din spate, gestionarea bazei de date, developarea API-ului, mecanismele de autentificare și de autorizare, limbajul folosit pe partea de server, scalabilitatea. Pe partea de client au fost examinate *framework*-ul .NET MAUI, *framework*-ul Community Toolkit, arhitectura MVVM, tiparuri de navigare, și componente din interfața cu utilizatorul. Aceste informații oferă cititorilor înțelegere cuprinzătoare asupra aspectelor care au avut parte în developarea aplicației GymJournal.

Deși GymJournal reprezintă o inovație semnificativă, este important de realizat că are limitări care trebuie identificate pentru developarea pe viitor. Una dintre limitări ar fi faptul că aplicația nu are integrare cu dispozitive populare de fitness, ceea ce ar putea îmbunătăți acuratețea și ar putea realiza în mod automat colectarea datelor legate de exerciții. Integrarea cu API-uri de la dispozitive populare de fitness cum ar fi *smartwatch, fitness tracker,* ar oferi utilizatorilor posibilitatea de importa data foarte ușor și de a oferi o perspectivă mai precisă în legătură cu obiectivele lor în materie de fitness.

În plus, scalabilitatea și optimizările pentru performanță implementate în cadrul aplicației sunt făcute special pentru un număr relativ redus de utilizatori. Pe măsură ce numărul de utilizatori crește, vor trebui luate măsuri cum ar fi balansarea greutății, scalarea pe orizontală, și altele pentru a asigura o performanță optimă.

În plus, extinzând capabilitățile aplicației dincolo de sesiuni de antrenament individuale, pentru a include sesiuni de grup, programarea de antrenamente individuale, sau funcționalități de interacțiune socială ar ajuta la crearea ideii de comunitate și la motivarea interpersonală. Încorporarea acestor elemente, ar oferi utilizatorilor o experiență mai colaborativă și mai atractivă.

În concluzie, aplicația reprezintă o contribuție valoroasă în domeniul aplicațiilor de contorizat antrenamente. Prin dezvoltarea de succes a ei, am demonstrat eficiența tehnologiei în sprijinirea utilizatorilor în atingerea obiectivelor lor în materie de fitness. Folosind tehnologii, *framework*-uri, aplicația oferă utilizatorilor o platformă de încredere, plină de funcționalități pentru urmărirea și gestionarea antrenamentelor.

Detaliile de implementare prezentate în acestă lucrare oferă informații valoroase în legătură cu procesul de dezvoltare, arhitectura sistemului, deciziile în materie de tehnologii din cadrul aplicației. De la implementarea pe partea de server la cea de pe parte de client, am demonstrat tehnici și metodologii folosite la nivel de industrie care asigură o soluție de calitate și eficientă.

În continuare, aplicația GymJournal are potențialul de a fi îmbunătățită și extinsă. Integrând dispozitive fitness, implementând funcționalități de scalabilitate și altele pentru comunitate, aplicația poate să își continue evoluția pentru a îndeplini nevoile în continuă schimbare ale pasionaților de fitness.

# Bibliografie


[1] Android - https://developer.android.com/docs/

[2] .NET - https://learn.microsoft.com/en-us/dotnet/

[3] .NET MAUI - https://learn.microsoft.com/en-us/dotnet/maui/

[4] ASP.NET Core - https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0

[5] Entity Framework - https://learn.microsoft.com/en-us/ef/

[6] Microsoft SQL Server https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16

[7] MVC - https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started

[8] C# - https://learn.microsoft.com/en-us/dotnet/csharp/

[9] SQL Language - https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/

[10] ASP.NET Web API - https://learn.microsoft.com/en-us/aspnet/web-api/

[11] REST Architecture - https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

[12] Community Toolkit MVVM - https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/

[13] Model-View-ViewModel (MVVM) - https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

[14] NUnit - https://docs.nunit.org/

[15] Moq - https://documentation.help/Moq/

[16] Xamarin.Forms - https://learn.microsoft.com/en-us/xamarin/xamarin-forms/

[17] Docker - https://docs.docker.com/

[18] Depedency Injection - https://en.wikipedia.org/wiki/Dependency_injection