**Documentație Backend**

|  |  |
| --- | --- |
| Autori: | **Sandu Alexandru**  **Vîju Tudor-Alexandru** |

**Conținut**

[**1. Controlul versiunilor** 1](#_Toc156390121)

[**1.1. Git** 1](#_Toc156390122)

[**1.2. Repository** 2](#_Toc156390123)

[**1.4. GitHub Desktop** 3](#_Toc156390124)

[**2. Tehnologii și framework-uri utilizate** 4](#_Toc156390125)

[**2.1. Visual Studio 2022** 4](#_Toc156390126)

[**2.2. C# și .NET Framework** 5](#_Toc156390127)

[**2.3. NuGet** 7](#_Toc156390128)

[**2.4. Entity Framework Core** 8](#_Toc156390129)

[**2.5. LINQ** 10](#_Toc156390130)

[**2.6. SignalR** 12](#_Toc156390131)

[**2.7. SQL Server** 13](#_Toc156390132)

[**2.8. Postman** 14](#_Toc156390133)

[**3. Arhitecturi utilizate în aplicație** 15](#_Toc156390134)

[**3.1. Arhitectura Model-View-Controller** 15](#_Toc156390135)

[**3.2. Arhitectura Client-Server** 16](#_Toc156390136)

[**4. Baza de date a aplicației** 17](#_Toc156390137)

[**4.1. Structurarea bazei de date** 18](#_Toc156390138)

[**4.2. Detalierea tabelei User** 18](#_Toc156390139)

[**4.3. Detalierea tabelei Room** 19](#_Toc156390140)

[**4.4. Detalierea tabelei Member** 20](#_Toc156390141)

[**4.5. Detalierea tabelei RoomMessage** 20](#_Toc156390142)

[**5. Cazuri de utilizare a aplicației** 21](#_Toc156390143)

[**5.1. Detalierea cazurilor de utilizare** 22](#_Toc156390144)

# **1. Controlul versiunilor**

## **1.1. Git**



Unul dintre cele mai folosite sisteme de gestionare a versiunilor unei aplicații, în ultimii ani, se numește Git. Git reprezintă un sistem de gestionare a versiunilor de tip open-source ce și-a arătat performanțele și facilitățile oferite în detrimentul competitorilor de pe această piață, rămânând cel mai folosit începând încă de la lansarea acestuia, în anul 2005, pe sistemul de operare Linux, de către Linus Torvalds, creatorul acestui sistem de operare.

Figura 1.1 – Logo Git

Unii competitori au implementat un sistem de control al versiunilor de tip CVCS

(Centralized Version Control System), unde un server acționeaza drept un repository principal centralizat pe care se află stocate toate versiunile de cod sursă. Creatorii Git au implementat un sistem de control al versiunilor de tip DVCS (Distributed Version Control System). Spre deosebire de competitori, avantajul unui sistem de tip DVCS, în comparație cu sistemele tip CVCS unde versiunile sunt salvate într-un repository la distanță, este acela că versiunile de cod ale aplicației pot fi salvate într-un repository la distanță, dar în același timp si fiecare copie a repository-ului principal stocată local acționează ca un repository local ce conține istoricul complet al schimbărilor per versiune.

Git reprezintă un sistem de control al versiunilor esențial pentru dezvoltarea de produse software, majoritatea produselor software ce sunt implementate cu ajutorul Git fiind realizate de către echipe de programatori.

În cadrul unui proiect de tip Git există 3 zone principale:

* Directorul de lucru (Working Directory): Acesta reprezintă o copie individuală a repository-ului principal. Fișierele din repository-ul principal sunt preluate si inserate într-un repository local pe computer-ul personal pentru a putea fi ulterior modificate și utilizate.
* Zona de etapizare (Staging Area): Aceasta este reprezentată de către un fișier care reține informații despre modificările sau adăugările ce ulterior vor putea fi comise către repository-ul principal. Acest fișier, de cele mai multe ori, este localizat în directorul de Git. Acest fișier mai este cunoscut si drept “index”, tocmai datorită faptului că ține evidența modificărilor în comparație cu ce se află pe repository-ul principal.
* Directorul Git (.git directory): În cadrul acestuia sunt memorate baza de date de obiecte și metadatele proiectului. Directorul Git este cea mai importantă zonă din cadrul Git și este cea cu ajutorul căreia este realizată sincronizarea repository-ului local cu cel principal, clonându-se cel din urmă.



Figura 1.2 – Zonele Git

Principalele etape prin care fișierele din fiecare dintre proiectele ce au implementat Git în cadrul acestora sunt:

* Modificat (Modified): Înseamnă că există fișiere în cadrul copiei de repository principal de pe repository-ul local (directorul de lucru) pe care le-ai modificat sau pe care le-ai adăugat. Acestea nu au fost încă comise către repository-ul principal pentru a îl actualiza.
* Etapizat (Staged): Din cadrul acestei etape pot face parte doar fișierele pe care le-ai adăugat sau pe care le-ai modificat. Fiecare fișier din versiunea curentă de pe repository-ul local ce se află în această etapă înseamnă că este selectat pentru a fi comis către repository-ul principal.
* Comis (Comitted): Din cadrul acestei etape pot face parte doar fișierele ce s-au aflat în starea de etapizare și li s-a terminat procesul de comitere către repository-ul principal. În urma acestei etape fișierele aflate pe repository-ul local au fost mutate în repository-ul principal.

## **1.2. Repository**

Un depozit (repository) este o formă de stocare centrală în care se pot salva toate documentele și resursele proiectului. Codul (resource) arhivei proiectului este accesibil tuturor participanților și dezvoltatorilor la proiect, care îl pot folosi pentru a furniza funcții noi sau pentru a rezolva erori din cadrul software.

Conceptele cheie ale unui repository sunt:

* Ramură (branch): O linie independentă de dezvoltare cunoscută sub numele de ramură (branch) are inițial aceeași bază de cod ca și repository-ul principal. Poate exista un set separat de fișiere și commit-uri în cadrul acestui noi branch.
* Commit (folosind comanda git commit): Efectuarea modificărilor fișierelor de proiect și salvarea lor în repository-ul principal este cunoscută sub numele de commit.
* Pull (folosind comanda git pull): Obținerea codului sursă al proiectului de la un repository remote (aflat deobicei pe un server în cloud) la repository-ul aflat pe computerul local este cunoscută sub numele de „extragere dintr-un repository”. Modificările fișierelor (fișierelor care sunt adăugate, editate sau șterse) care trebuie să fie integrate cu repository-ul principal sunt conținute într-o cerere de extragere.
* Push (folosind comanda git push): Push-ul se referă la încărcarea codului sursă creat pe repository-ul local într-un repository de la distanță (cel principal).

Principalii furnizori de servicii de găzduire în mediul on-line de repository-uri de tip Git sunt: Bitbucket, GitHub, respectiv GitLab.

## **1.4. GitHub Desktop**

GitHub Desktop este o aplicație care simplifică gestionarea versiunilor codului sursă folosind Git. Cu o interfață grafică intuitivă, dezvoltatorii pot vizualiza, face commit-uri și sincroniza proiectele direct de pe desktop. Acest instrument permite explorarea istoricului modificărilor, compunerea și revizuirea branch-urilor, facilitând colaborarea în echipă. Integrează strâns cu platforma GitHub, oferind posibilitatea de a urmări și gestiona contribuțiile colaboratorilor. GitHub Desktop oferă un mediu intuitiv pentru gestionarea fluxurilor de lucru, rezolvarea conflictelor și menținerea coerenței în proiectele software. Prin intermediul său, programatorii pot obține o experiență eficientă în dezvoltarea și administrarea proiectelor, îmbunătățind colaborarea și gestionarea codului sursă.

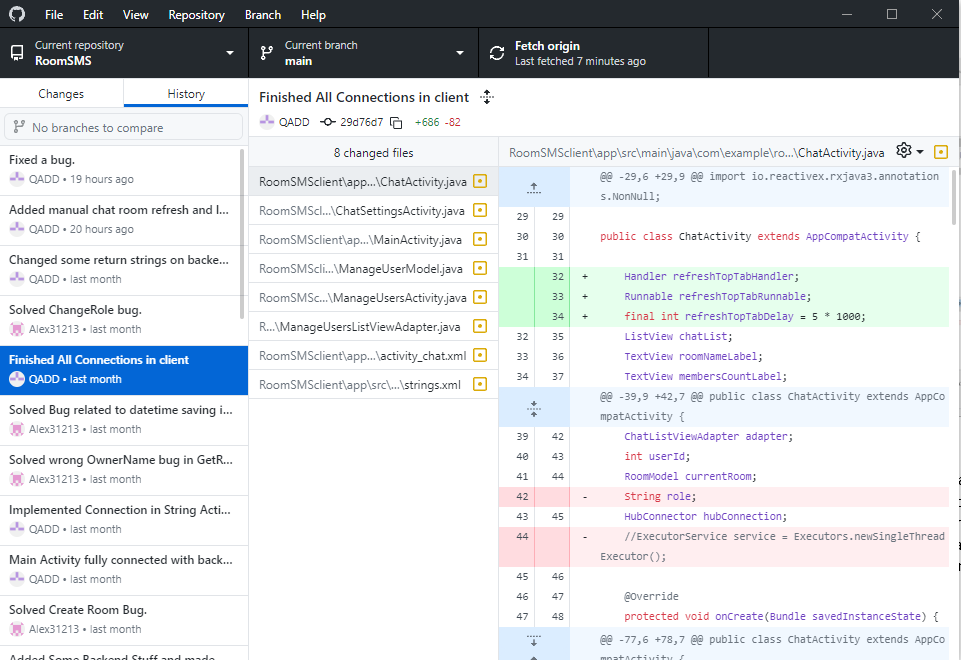
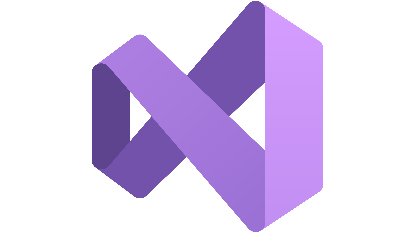


Figura 1.3 - Interfață GitHub Desktop Proiect

În cadrul imaginii din Figura 1.3 se poate observa proiectul de recreare a unei aplicații de mesagerie implementat cu ajutorul Git, în cadrul interfaței GitHub Desktop. Aici se pot observa modificările aduse fișierelor din proiect față de ce se află pe repository-ul principal, respectiv numărul fișierelor modificate sau adăugate. De asemenea se poate observa și istoricul de commit-uri, cu datele când acestea au fost executate. Am folosit GitHub Desktop pentru a îndeplini necesitatea de versionare a proiectului datorită accesibilității si ușurinței de operare în cadrul ei.

# **2. Tehnologii și framework-uri utilizate**

## **2.1. Visual Studio 2022**



Microsoft Visual Studio 2022 este un mediu de dezvoltare integrat (IDE), el fiind unul dintre IDE-urile ce dezvoltatorii îl folosesc frecvent pentru a colabora la proiecte de codare și depanare. Această platformă a fost dezvoltată de Microsoft și este adesea folosită pentru a crea software care rulează pe Windows, Android și iOS. Alte exemple de IDE-uri folosite de catre programatori sunt: Eclipse, IntelliJ IDEA, PyCharm.

Figura 2.1 – Logo Visual Studio 2022

De la lansarea sa inițială în 1997, Visual Studio a crescut pentru a se clasa printre cele mai utilizate IDE-uri disponibile. Utilizatorii pot construi site-uri web, aplicații web, servicii online și aplicații mobile folosind acest mediu de dezvoltare.

Visual Studio 2022 este scris în limbajele de programare C++ și C# și include un editor de cod care se folosește IntelliSense. De asemenea, are un depanator integrat care funcționează atât ca depanator la nivel de cod sursă, cât și ca depanator la nivel de cod mașină. Mediul de dezvoltare integrat creat de către Microsoft, permite, în special, programatorilor să scrie și să schimbe codul din cadrul aceluiași proiect, în regim de colaborare între ei. Visual Studio 2022 include, de asemenea, instrumente de asistență pentru a scrie cod, cum ar fi evidențierea sintaxei, modele de codare AI (Artificial Intelligence) și instrumente de analiză la depanare.

Cea mai recentă versiune a Visual Studio, Visual Studio 2022 17.8.1, care acceptă conectivitate atât cu GitHub, cât și cu serviciul Microsoft Azure Cloud computing, a fost publicată pe 9 ianuarie 2024.

Creatorul Visual Studio 2022, Microsoft, a anunțat faptul că acest IDE poate fi folosit pentru crearea de software pentru Windows, iOS si Android, fără vreun cost, cu ajutorul versiunii “Community”.

Acest proiect utilizează Microsoft Visual Studio 2022 pentru partea de back-end a aplicației.

## **2.2. C# și .NET Framework**

C# este un limbaj de programare de nivel înalt, multi-paradigmă, de uz general.

O echipă de la Microsoft condusă de Anders Hejlsberg a început să lucreze la C# la sfârșitul anilor 1990. Cunoscută inițial sub numele de “Cool”, limbajul a fost redenumit C# când proiectul .NET a fost făcut public pentru prima dată în vara anului 2002. Sufixul „#” a fost folosit pentru a indica faptul că limbajul este un avans al C++, care a servit drept inspirație pentru design-ul CLI (Command-line Interface) și limbajului C#.

Când au fost lansate .NET Framework 1.0 și Visual Studio.NET 2002 în 2002, prima versiune de C#, denumită 1.0, a fost de asemenea făcută accesibilă. De atunci, atât modificările semnificative, cât și cele minore ale limbajului de programare au fost publicate.

Particularități ale limbajului C#:

* Programare Orientată-Obiect (POO/OOP): Programarea orientată-obiect este un concept fundamental în cadrul programării. C# este un limbaj de programare de tip orientat-obiect. Spre deosebire de limbajele de programare procedurale, unde este dificil de dezvoltat și îmbunătățit codul pe măsură ce codul se dezvoltă și pe măsură ce dimensiunea proiectelor crește,în cadrul limbajelor OOP sunt facilitate dezvoltarea și îmbunătățirea codului proiectelor. Conceptele specifice OOP, precum încapsularea datelor, moștenirea, polimorfismul și interfețele sunt prezente în cadrul limbajului C#.
* Actualizabil și scalabil: Pentru a putea scala o aplicație se îndepartează fișierele învechite și le înlocuim cu altele noi. Ar putea fi dificil să se actualizeze componentele software fără ca programatorul ce se ocupă de acest lucru sa provoace greșeli în codul scris de el. Modificările aduse codului pot afecta aplicația în cadrul rulării acesteia, iar C# pentru a ajuta și preveni problemele întâmpinate la apariția erorilor în aplicație, acceptă versionarea proiectului.
* Type-safe: Codul C# are permisiunea de a accesa doar zonele din memorie pentru care are accesul de utilizare. În C# listele au index-ul de start egal cu 0, variabilele de referință, precum clasele si obiectele, sunt inițializate implicit cu null în caz de este omisă inițializarea acestora de către programator, respectiv variabilele ce stocheaza valori inițializate cu 0. De asemenea, în limbajul C# conversiile implicite de tipuri de valori, ca de exemplu dintr-o variabila de tip float intr-o variabila de tip boolean. În consecință, securitatea programului este crescută.
* Garbage Collector (GC): Garbage Collector-ul (GC) îndeplinește funcția unui manager automat de memorie în cadrul Common Language Runtime (CLR). Gestionarea alocării și eliberării de memorie pentru o aplicație este gestionată de către garbage collector. Drept beneficiu, programatorii care lucrează cu programe gestionate de un Garbage Collector nu trebuie să scrie cod pentru a gestiona funcțiile de gestionare a memoriei. Gestionarea automată a memoriei poate rezolva probleme comune, cum ar fi uitarea de a elibera memoria alocată unui obiect, ceea ce ar putea duce la o scurgere de memorie sau încercarea de a utiliza memoria unui obiect eliberat fără permisiunea acestuia, lucru ce poate provoca erori grave în aplicație.

Arhitectura .NET poate fi vazută în imaginea de mai jos. Aceasta este afișată în mod ierarhizat, Common Language Runtime-ul (CLR) reprezentând nivelul cel mai putin abstractizat, fiind cel mai apropiat de limbajul cod-mașină.

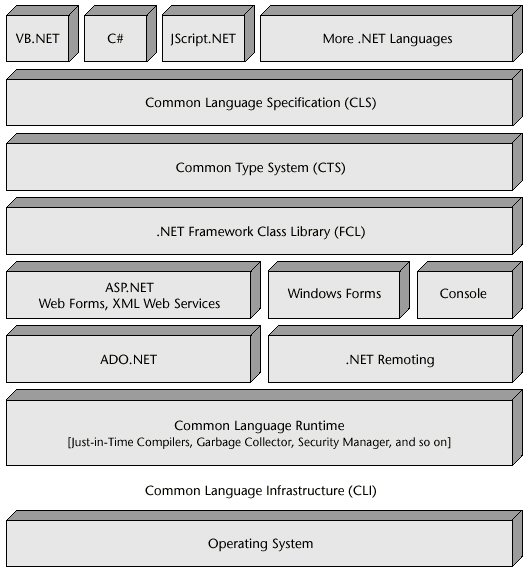


Figura 2.2 – Arhitectura .NET

Acest proiect folosește versiunea de 6.0 de .NET.

## **2.3. NuGet**

Metoda de partajare a codului acceptată de Microsoft pentru .NET (inclusiv .NET Core) este NuGet, care stipulează procedurile pentru producerea, găzduirea și consumarea pachetelor .NET și oferă instrumentele necesare în fiecare caz.

NuGet este un sistem care permite dezvoltatorilor să scrie, să schimbe și să utilizeze cod utilizabil, aceasta fiind o componentă crucială a oricărei platforme moderne de dezvoltare. Un astfel de cod este adesea inclus în pachete care includ și alte materiale necesare proiectelor ce folosesc aceste pachete, cum ar fi codul de tip compilat (sub formă de DLL-uri).

Folosind NuGet, care vine preinstalat cu Visual Studio 2022, se pot căuta, obține și instala pachete pentru proiectul curent. În plus, le permite dezvoltatorilor să actualizeze pachetele la versiuni mai noi și să gestioneze dependențele de pachete pentru proiecte. Acesta poate fi folosit de asemenea pentru a instala pachete realizate de către companii sau dezvoltatori individuali, precum și pentru a instala pachete de tip open-source.

NuGet poate fi folosit pentru orice tip de proiecte realizate în cadrul IDE-ului Visual Studio, precum proiecte de tip desktop, web, sau chiar și pentru cele de tip mobile.

În cadrul proiectului este folosit NuGet pentru a adauga urmatoarele pachete:

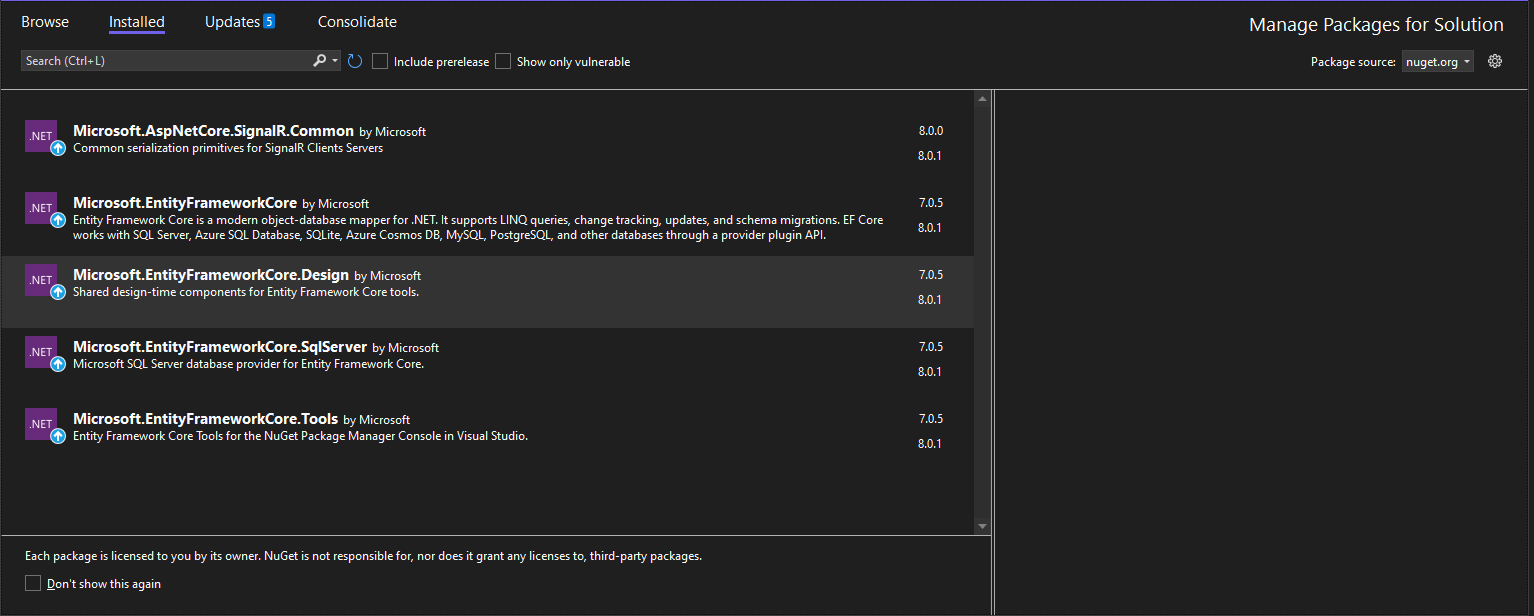


Figura 2.3 - Pachetele NuGet ale aplicației

Figura 4.3 – Pachetele NuGet ale aplicației

## **2.4. Entity Framework Core**

Un framework ORM (Object-Relational Mapper) pentru accesul la date în .NET se numește Entity Framework (EF) Core. Este o versiune multiplatformă, flexibilă, open-source a tehnologiei de acces la date Entity Framework, care a fost publicată în același timp cu .NET Core. Acest framework este compatibil cu mai multe sisteme de operare, inclusiv Windows, Linux și Mac OS.

Entity Framework Core a început să fie utilizat și în afara framework-ului de .NET de la versiunea 6.0.

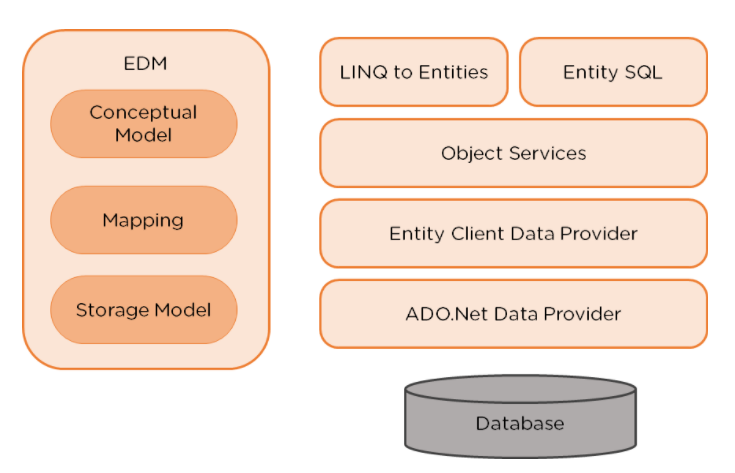


Figura 2.4.1 – Arhitectura Entity Framework

Object-Relational Mapper, sau ORM, este un instrument care produce automat clase în cod bazate pe tabelele din baza de date. De asemenea, acesta cunoaște și să creeze tabele în baza de date folosind clasele create în codul proiectului. Cu alte cuvinte, poate furniza și codul SQL-ul necesar pentru a construi baza de date pe baza claselor din cod.

Când se efectuează operațiuni CRUD (Create-Read-Update-Delete), framework-ul ORM creează codul SQL-ul adecvat pe care baza de date îl poate înțelege. Dezvoltatorii de aplicații interacționează adesea cu obiectele de tip business ale aplicației. Concluzionând, se poate afirma faptul că framework-ul ORM elimină necesitatea de a scrie cod de acces la date din baza de date, cod pe care dezvoltatorii îl scriu de cele mai multe ori.

Entity Framework Core, prin pachetele sale, oferă suport de compatibilitate atât pentru baze de date relaționale precum:

* Oracle
* PostgreSQL
* SQL Server
* SQLite

, dar de asemenea și pentru baze de date non-relaționale precum:

* Firebird
* Azure Cosmos DB

Pentru a putea utiliza Entity Framework Core în cadrul aplicației a fost necesar a se instala pachetele corespunzătoare, ce sunt compatibile cu tipul bazei de date care este folosită.

În cadrul acestui proiect au fost instalate următoarele pachete Entity Framework Core, versiunea 7.0.5, pentru SQL Server:

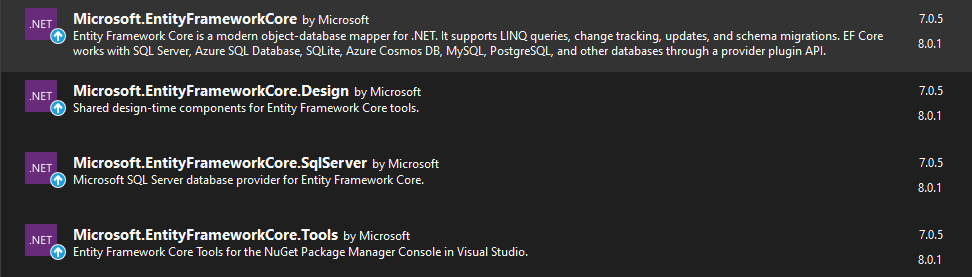


Figura 2.4.2 – Pachetele EntityFrameworkCore instalate

## **2.5. LINQ**

Language Integrated Query, sau LINQ, a fost făcut disponibil pentru prima dată în versiunea de .NET 3.5 și în Visual Studio 2008. LINQ oferă limbajelor de tip .NET (cum ar fi C#, VB.NET etc.) capacitatea de a crea interogări pentru a prelua date din sursa de date. Programatorii îl folosesc pentru a îndeplini sarcini precum preluarea datelor salvate despre un client al unui magazin. Astfel de date erau stocate într-o bază de date separată de program în anii precedenți, iar accesarea acesteia necesita cunoașterea diferitelor limbaje de interogare precum SQL, XML etc.

În trecut, niciun limbaj de tip .NET nu accepta crearea de interogări către o bază de date. Singura modalitate de a interoga o bază de date până la apariția LINQ era scrierea de interogări în cod SQL, sub formă de șiruri de caractere, pe care baza de date le primea de la programator si le procesa ulterior pentru a fi oferit rezultatul.

Microsoft a creat LINQ pentru a rezolva acest tip de probleme. Acesta adaugă încă o capacitate limbajelor de programare de tip .NET, permițându-le să creeze interogări adresate oricărei surse de date compatibilă cu LINQ. LINQ exclude necesitatea de cunoaștere ale altor limbaje de programare precum SQL sau de alte mecanisme de limbaj non-.NET, fiind necesară doar cunoașterea limbajului C#, deoarece sintaxa pentru crearea unei interogări este aceeași, indiferent de tipul de sursă de date utilizată. De exemplu, codul scris de către programator pentru a crea sau a actualiza, folosind interogări, datele stocate într-o bază de date de tip relațional folosește aceeași sintaxă ca și crearea sau actualizarea unei interogări pentru datele stocate într-o listă din codul C#. În plus, LINQ poate fi utilizat cu orice tip de bază de date, inclusiv cu cele care utilizează SQL, XML, ADO.NET, servicii web și alte limbaje.

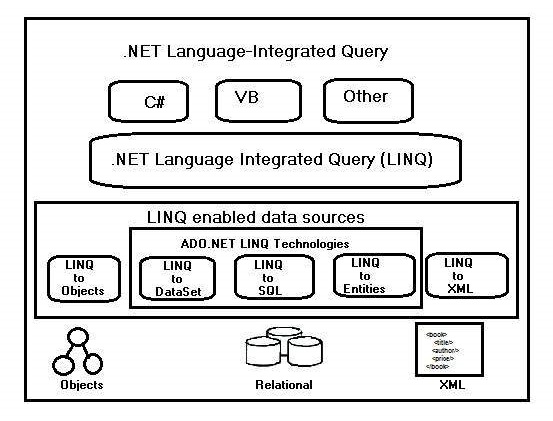


Figura 2.5.1 – Arhitectura LINQ

Pentru a include LINQ în cadrul proiectului scris în limbajul de programare C#, trebuie inclus în proiect namespace-ul “System.Linq”. LINQ se folosește de următoarele interfețe pentru a putea fi asigurată funcționalitatea dorită:

* IQueryable<T>
* IEnumerable<T>

LINQ este capabil să realizeze asupra unui set de date operații uzuale precum ordonarea acestora în funcție de un criteriu stabilit folosind funcții precum “OrderBy”, filtrarea datelor prin operații precum “Select” sau “FirstOrDefault”, respectiv gruparea acestora folosind funcții precum “GroupBy”.

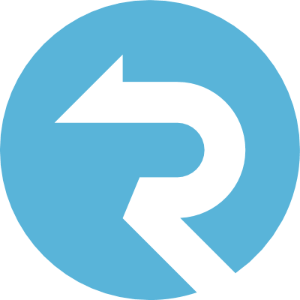


Figură 2.5.2 - Exemplu utilizare LINQ din proiect

Avantajele utilizării LINQ în cadrul unui proiect scris în limbajul de programare C# sunt următoarele:

* Reuzabilitatea interogărilor scrise în LINQ
* Cod mai ușor de citit și descifrat
* Conversie facilă dintr-un tip de date în alt tip de date, ca de exemplu din XML în SQL si invers
* Programatorul nu mai e nevoit a cunoaște limbaje de programare specifice fiecărui limbaj de interogare în parte, LINQ realizând conversia din cod C# în limbajul de interogare în mod automat
* Include suport acordat pentru depanarea codului și pentru IntelliSense
* Strongly Typed: O eroare de compilare va fi produsă când se încearcă atribuirea unei valori unei variabile care nu este implicit convertibilă sau dacă se încearcă a se trimite un tip incorect de parametru ca argument.

## **2.6. SignalR**



Adăugarea de capabilități de tip online în timp real, pentru aplicații, este simplificată de biblioteca ASP.NET SignalR, bibliotecă utilizată de programatorii .NET. Spre deosebire de funcționalitatea web tradițională, care necesită ca server-ul să aștepte ca un client să solicite date înnoite, conexiunea web în timp real permite serverului să trimită informații către clienții conectați, instantaneu, pe măsură ce aceste informații oferite de către server devin disponibile.

Figura 2.6 – Logo SignalR

SignalR a fost lansat de către firma Microsoft în data de 18 februarie 2013, autorii acestei librării fiind David Fowler and Damian Edwards. Ultima versiune a acestei librării este versiunea 2.4.3 , lansată în luna ianuarie, 2022.

Comunicarea online bidirecțională în timp real între server și clienții acestuia este posibilă folosind librăria SignalR de tip open-source. Este posibilă crearea de cod în partea de server a proiectului, care poate comunica instantaneu cu clienții acestuia folosind SignalR.

În plus, SignalR oferă un API ușor de utilizat, de nivel înalt, pentru a realiza RPC-uri (Remote Procedure Call) de la server la client, în aplicațiile de tip ASP.NET, permițând codului .NET de pe partea de server să acceseze funcțiile JavaScript din browser-ul unui client. SignalR include, de asemenea, și metode utile de gestionare, cum ar fi evenimentele de conectare/deconectare, gruparea conexiunilor și parte de autorizare a accesului clientului la codul SignalR de pe server.

Dacă se dorește comunicarea de date în format JSON comprimat, trebuie ca dezvoltatorul să-și creeze propria logică pe partea de server și pe partea de client, deoarece librăria SignalR transmite datele numai în text simplu sau în format JSON necomprimat. SignalR se bazează în mare măsură pe metode asincrone pentru a oferi performanțe instantanee.

SignalR este compatibil cu o varietate de clienți precum:

* Aplicații desktop
* Aplicații mobile
* Browser
* Dispozitive IoT (Internet of Things)
* Console de jocuri

Pentru comunicarea între clienți și servere, SignalR oferă următoarele tipuri:

* Hub-uri: Pentru ca clientul și serverul să apeleze unul altuia metode, hub-urile oferă un API de nivel înalt. Este recomandat să fie folosite Hub-uri dacă se dorește să se evite gestionarea propriilor dispecerizări în timp ce trimiteți mai multe tipuri de mesaje între un server și un client, deoarece Hub-urile nu au nevoie de acestea. Principalul avantaj este că dacă se alege crearea unei aplicații folosind Hub-uri va fi mai simplă de construit decât dacă se alege utilizarea unei conexiuni persistente.
* Conexiuni persistente: Accesul direct la un protocol de comunicație de nivel scăzut oferit de SignalR este disponibil prin conexiuni persistente. Un id de conexiune identifică în mod unic fiecare conexiune client-server. Prin urmare, se poate utiliza acest model în SignalR dacă aplicația necesită control suplimentar asupra unei conexiuni.

Pentru a gestiona comunicarea în timp real de la server la client, SignalR folosește o varietate de tehnologii, precum:

* Long Polling
* WebSockets
* Evenimente trimise de server(Server-Sent Events)

## **2.7. SQL Server**

Limbajul de programare SQL (Structured Query Language) este folosit pentru stocarea și procesarea datelor în cadrul bazelor de date relaționale. O bază de date relațională stochează date le sub formă tabelară, cu rânduri și coloane care indică diferite proprietăți ale datelor și relațiile dintre valorile acelor atribute. Cu ajutorul SQL se pot stoca, actualiza, elimina, căuta și prelua date din baza de date, folosind instrucțiuni SQL. SQL poate fi, de asemenea, utilizat pentru a îmbunătăți și menține performanța bazei de date.

Un sistem de management al bazelor de date relaționale (RDBMS) este SQL Server, creat de către firma Microsoft. Acesta folosește drept limbaj de interogare limbajul SQL, iar ca extensie procedurala utilizează T-SQL. Stocarea și preluarea datelor pentru diverse aplicații este scopul său principal.

În SQL Server, secțiunea de „Managment System” e responsabilă de faptul că bazele de date sunt stocate eficient și că datele pot fi căutate sau preluate cât mai rapid posibil. În plus, se asigură că numeroase conexiuni la aceleași date sunt gestionate corect pentru a preveni apariția problemelor dacă două persoane fac aceeași actualizare în același timp.

În cadrul acestei aplicații s-a folosit Microsoft SQL Server 2022, versiunea 16.0.1000.6 (x64).

## **2.8. Postman**



Figura 2.8.1 – Logo Postman

Pentru a avea un programator web posibilitatea de a testa API-urile create în partea de back-end a aplicației, unul dintre cele mai utilizate instrumente de testare a API-urilor se numește Postman.

În anul 2012, după ce a lucrat la Yahoo Bangalore, dezvoltatorul de software Abhinav Asthana a creat Postman. Scopul propus pentru acest proiect era acela de a simplifica testarea API-urilor.

Cu ajutorul unei interfețe grafice integrate în cadrul acestei platforme, incorporată pentru a face testarea API-urilor mai facilă, Postman este un client HTTP care evaluează cererile HTTP trimise către un furnizor de API. Ca urmare, un programator poate obține diverse forme de răspunsuri ce ulterior trebuiesc a fi validate.

În cadrul Postman pot fi testate diverse endpoint-uri, precum cele de tip:

* GET
* DELETE
* PUT
* POST

Pentru a prezenta cum este executată partea de testare cu ajutorul Postman, am realizat testarea endpoint-ului responsabil cu acțiunea de autentificare în cadrul aplicației. Acesta se află în partea de back-end a aplicației, în hub-ul numit “AppHub”, funcția numindu-se “Login”. Aceasta este o funcție ce suportă websockets. Ea a fost accesată folosind URL-ul “[http://localhost:5190](http://localhost:5190/login)”, body-ul fiind trimis în format JSON (JavaScript Object Notation).

Putem observa faptul ca în corpul cererii a fost trimis în format JSON email-ul “alex312@gmail.com” si parola “parola”. Ulterior, după ce server-ul a primit cererea și a procesat-o, ni s-a oferit un răspuns.

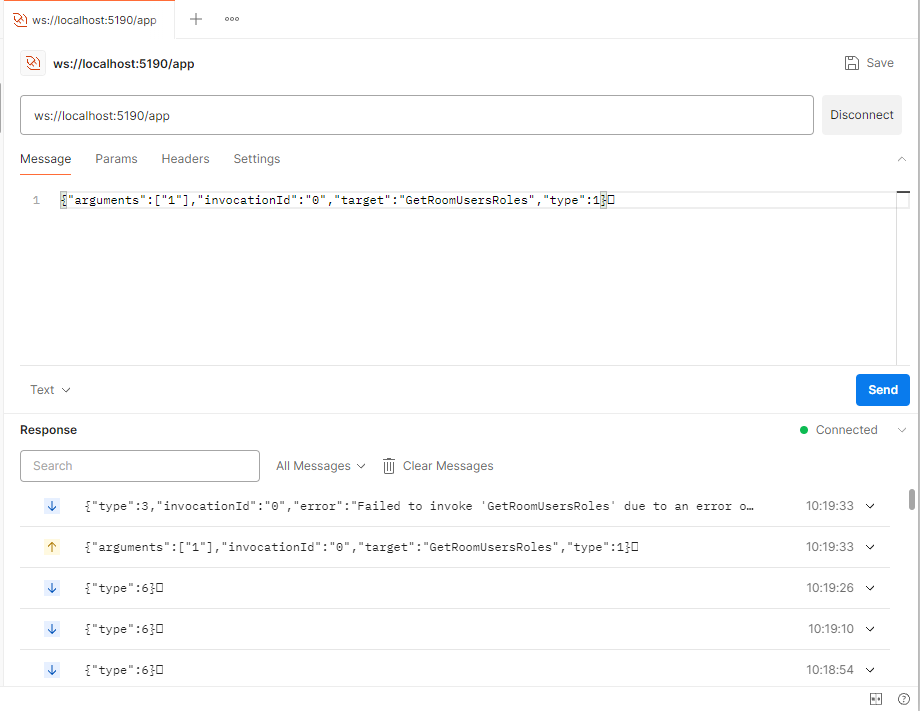


Figura 2.8 - Testare in Postman

# **3. Arhitecturi utilizate în aplicație**

## **3.1. Arhitectura Model-View-Controller**

Modelul de design arhitectural cunoscut sub numele de Model-View-Controller (MVC) împarte o aplicație în trei părți logice principale: modelul(Model), vizualizarea(View) și controlerul(Controller), așa cum sugerează și numele. Fiecare dintre aceste părți a fost creată pentru a face față unui anumit task în dezvoltarea aplicațiilor. MVC este unul dintre cele mai populare framework web pentru realizarea de proiecte care sunt scalabile și extensibile.

Explicarea celor 3 părți:

* Model(Model): Întreaga parte de logică a utilizatorului, legat de date, este corelată cu partea de model. Acesta poate fi folosit pentru a reprezenta orice date legate de datele care sunt transmise între componentele view și controller. Datele din baza de date pot fi adăugate sau preluate de către partea de model. Controlerul nu se poate conecta singur la baza de date, prin urmare modelul răspunde la cererea sa atunci când controlerul are nevoie de date din baza de date. Controlerul primește datele necesare de la model odată ce acesta a interacționat cu baza de date.
* Vizualizare(View): Întreaga funcționalitate UI(User Interface) a aplicației, reprezentată de partea de Android.
* Controler(Controller): Este partea care face posibilă comunicarea dintre model și view, el fiind un intermediar între cele 2. Controlerul trebuie doar să instruiască modelul, acesta nefiind necesar să gestioneze partea de logică a datelor. Pentru a reda produsul finit, procesează solicitările primite, lucrând cu componenta de view și cea de Model pentru a schimba datele primite.

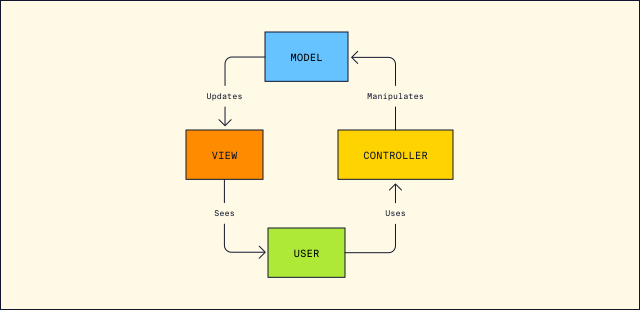


Figura 3.1 – Arhitectura Model-View-Controller

## **3.2. Arhitectura Client-Server**

Un sistem distribuit care cuprinde atât clienți, cât și servere este cunoscut drept fiind o arhitectură client-server. Găzduirea, administrarea și furnizarea de servicii clienților sunt realizate de către server. Clienții se conectează la server și folosesc un protocol pentru a comunica cu acesta. Ca rezultat, ori de câte ori un client solicită un serviciu, serverul evaluează cererea și îi răspunde clientului.

În concluzie, în cadrul unei arhitecturi de tip client-server se disting 2 clase:

* Client: Orice calculator care întocmește o cerere către server este clasificat drept un client.
* Server: Calculatorul care este destinat să îndeplinească cererile clientului este cunoscut sub numele de server.

Avantajele incorporării unei arhitecturi de tip client-server:

* Securitate
* Control centralizat al informațiilor
* Scalabilitate
* Acces facil al clienților

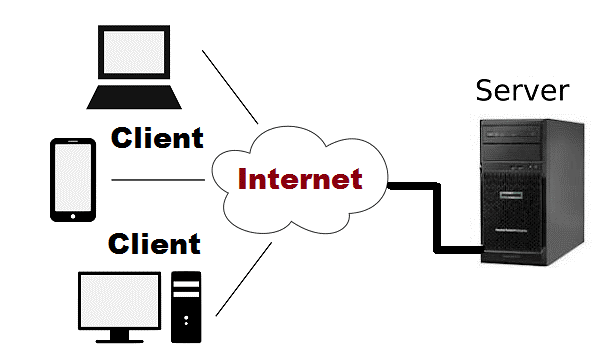


Figura 3.2 – Arhitectura Client-Server

În contextul aplicației noastre, clientul este reprezentat de către orice utilizator care acceseaza URL-ul destinat paginii de front-end din browser, iar server-ul este reprezentat de către aplicațtia de back-end construită în limbajul C#. Comunicarea dintre acestea 2 este realizată folosindu-se websockets.

# **4. Baza de date a aplicației**

Pentru a înmagazina informațiile din cadrul aplicației ce necesită stocare pe un termen îndelungat de timp a fost utilizat sistemul de management al bazelor de date relaționale denumit Microsoft SQL Server 2022, pe versiunea 16.0.1000.6 (x64).

Pentru a putea integra baza de date în cadrul aplicației, s-a instalat în partea de back-end a aplicației pachetul de tip NuGet denumit “Microsoft.EntityFrameworkCore” pentru SqlServer.

Pentru această aplicație s-a folosit abordarea de tip “Database First Approach”, cu care au fost create codurile necesare în cod modelelor corespondente tabelelor din baza de date. Aceasta s-a realizat folosind metoda “Scaffold” oferită de către “Microsoft.EntityFramework.Core.Design”, string-ul de conexiune către baza de date fiind localizat în fișierul “appsettings.json”.

## **4.1. Structurarea bazei de date**

Datorită faptului că scopul acestui proiect este acela de a realiza o aplicație ce operează în timp real ce recreează o aplicație de mesagerie, majoritatea operațiunilor executate în cadrul acesteia necesitând o soluție de stocare pe termen lung, această aplicație a avut nevoie de informații stocate într-o bază de date.

Ulterior vizionării schemei bazei de date, se pot observa 4 tabele denumite astfel: “User”, “Room”, “Member”, “RoomMessage”. Aceste tabele reprezintă datele stocate în baza de date referitoare la fiecare utilizator al acestei aplicații, respectiv fiecărei camere create în cadrul acesteia.

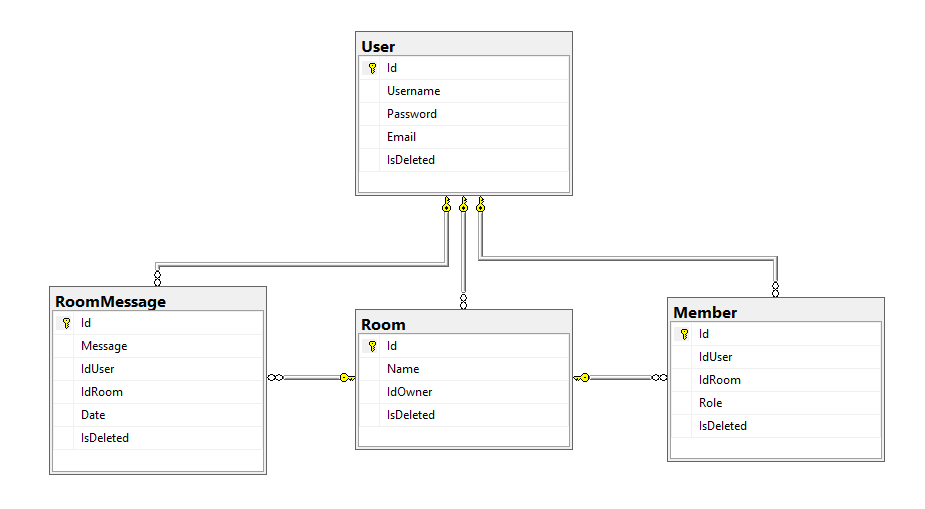


Figura 4.1 - Entity Relationship Diagram-ul (ERD) aplicației

## **4.2. Detalierea tabelei User**

Tabela “User” conține următoarele câmpuri:

* Id: El este autoincrementabil, fiind PK-ul(Primary Key-ul) tabelei “User”.
* Username: El este stocat folosind tipul nvarchar(50). “nvarchar” este un șir de caractere cu lungime variabilă ce poate atinge o lungime maximă specificată între paranteze. În acest caz, lungimea maxima a câmpului “Username” este de 50 de caractere.
* Password: Parola este memorată într-un tip de date nvarchar(100). În aceasta tabela este reținută parola contului după ce a fost hash-uită folosind standardul RFC2898.
* Email: Pentru “Email” s-a folosit tipul de date nvarchar(50). În acest câmp este memorat email-ul utilizatorului afiliat username-ului.
* isDeleted: Acest câmp stochează o valoare de tip bit. Valoarea de tip bit poate stoca un tip întreg, ce poate fi doar “0” sau “1”. Valoarea “1” este echivalentul lui “TRUE” din boolean, aceasta semnificând că utilizatorul este șters. Dacă valoarea este “0”, echivalentul lui “FALSE” din boolean, utilizatorul este unul activ, acesta, acesta nefiind șters. Ștergerea în baza de date a unui utilizator se face prin modificarea valorii câmpului din “0” în “1”.

Exemplu de informație memorată în tabela “User” în baza de date:



Figura 1.2 - Exemplu date memorate în tabela User

## **4.3. Detalierea tabelei Room**

Tabela “Room” conține următoarele câmpuri:

* Id: El este autoincrementabil, fiind PK-ul(Primary Key-ul) tabelei “Room”.
* Name: : El este stocat folosind tipul nvarchar(50). “nvarchar” este un șir de caractere cu lungime variabilă ce poate atinge o lungime maximă specificată între paranteze. În acest caz, lungimea maxima a câmpului “Name” este de 50 de caractere.
* IdOwner: El reprezintă un FK(Foreign Key) catre id-ul din tabela “User”, fiind PK-ul(Primary Key-ul) tabelei “User”.
* isDeleted: Acest câmp stochează o valoare de tip bit. Valoarea de tip bit poate stoca un tip întreg, ce poate fi doar “0” sau “1”. Valoarea “1” este echivalentul lui “TRUE” din boolean, aceasta semnificând că utilizatorul este șters. Dacă valoarea este “0”, echivalentul lui “FALSE” din boolean, utilizatorul este unul activ, acesta, acesta nefiind șters. Ștergerea în baza de date a unui utilizator se face prin modificarea valorii câmpului din “0” în “1”.

Exemplu de informație memorată în tabela “Room” în baza de date:



Figura 2.3 - Exemplu date memorate în tabela Room

## **4.4. Detalierea tabelei Member**

Tabela “Member” conține următoarele câmpuri:

* Id: El este autoincrementabil, fiind PK-ul(Primary Key-ul) tabelei “Member”.
* IdUser: El reprezintă un FK(Foreign Key) catre id-ul din tabela “User”, fiind PK-ul(Primary Key-ul) tabelei “User”.
* IdRoom: El reprezintă un FK(Foreign Key) catre id-ul din tabela “Room”, fiind PK-ul(Primary Key-ul) tabelei “Room”.
* Role: El este stocat folosind tipul nvarchar(20). “nvarchar” este un șir de caractere cu lungime variabilă ce poate atinge o lungime maximă specificată între paranteze. În acest caz, lungimea maxima a câmpului “Role” este de 20 de caractere.
* isDeleted: Acest câmp stochează o valoare de tip bit. Valoarea de tip bit poate stoca un tip întreg, ce poate fi doar “0” sau “1”. Valoarea “1” este echivalentul lui “TRUE” din boolean, aceasta semnificând că utilizatorul este șters. Dacă valoarea este “0”, echivalentul lui “FALSE” din boolean, utilizatorul este unul activ, acesta, acesta nefiind șters. Ștergerea în baza de date a unui utilizator se face prin modificarea valorii câmpului din “0” în “1”.

Exemplu de informație memorată în tabela “Member” în baza de date:



Figura 3.4 - Exemplu date memorate în tabela Member

## **4.5. Detalierea tabelei RoomMessage**

Tabela “RoomMessage” conține următoarele câmpuri:

* Id: El este autoincrementabil, fiind PK-ul(Primary Key-ul) tabelei “RoomMessage”.
* Message: El este stocat folosind tipul nvarchar(500). “nvarchar” este un șir de caractere cu lungime variabilă ce poate atinge o lungime maximă specificată între paranteze. În acest caz, lungimea maxima a câmpului “Message” este de 500 de caractere.
* IdUser: El reprezintă un FK(Foreign Key) catre id-ul din tabela “User”, fiind PK-ul(Primary Key-ul) tabelei “User”.
* IdRoom: El reprezintă un FK(Foreign Key) catre id-ul din tabela “Room”, fiind PK-ul(Primary Key-ul) tabelei “Room”.
* Date: El reprezintă data la care a fost postat mesajul în cadrul camerei, acest câmp fiind stocat folosindu-se tipul de date datetime.
* isDeleted: Acest câmp stochează o valoare de tip bit. Valoarea de tip bit poate stoca un tip întreg, ce poate fi doar “0” sau “1”. Valoarea “1” este echivalentul lui “TRUE” din boolean, aceasta semnificând că utilizatorul este șters. Dacă valoarea este “0”, echivalentul lui “FALSE” din boolean, utilizatorul este unul activ, acesta, acesta nefiind șters. Ștergerea în baza de date a unui utilizator se face prin modificarea valorii câmpului din “0” în “1”.

Exemplu de informație memorată în tabela “RoomMessage” în baza de date:



Figura 4.5 - Exemplu date memorate în tabela RoomMessage

# **5. Cazuri de utilizare a aplicației**

Scopul acestei aplicații este acela de a crea o zonă unde orice utilizator al acesteia, ulterior creării unui cont de utilizator, să poată adera unei camere și să converseze împreună cu alți utilizatori precum el.

În aceasta aplicație poate fi identificat un singur tip de actor:

* Utilizator: Acesta folosește aplicația cu scopul de a putea comunica cu alți utilizatori.

Cazurile de utilizare întâlnite de către utilizator în folosirea aplicației sunt următoarele:

* Crearea unui cont de utilizator
* Autentificarea în aplicație folosind contul de utilizator creat
* Crearea unei camere cu ajutorul căreia poate conversa cu alți utilizatori:
  + Atribuirea unui nume camerei create
  + Schimbarea numelui atribuit camerei
* Vizualizarea camerelor în care acest utilizator a aderat
* Adăugarea unui utilizator în cadrul unei camere create
* Atribuirea unui rol unui alt utilizator în cadrul camerei de care aparține
* Părăsirea unei camere
* Ștergerea unei camere
* Trimiterea unui mesaj în cadrul camerei de care acesta aparține

În imaginea de mai jos este reprezentată diagrama cazurilor de utilizare pentru aceasta aplicație. Diagrama a fost realizată folosind site-ul web <https://app.diagrams.net>.

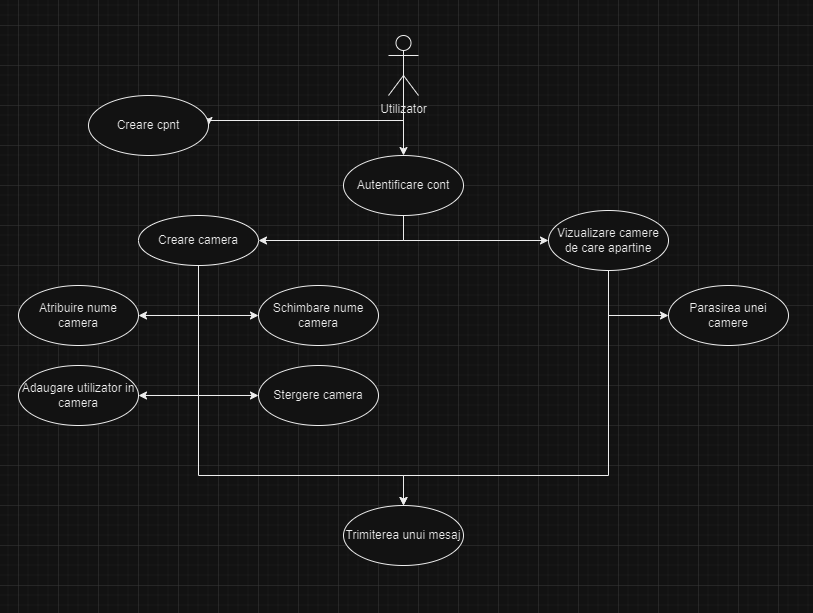


Figura 5 - Diagramă cazuri de utilizare

## **5.1. Detalierea cazurilor de utilizare**

În următorul tabel sunt detaliate pe scurt toate cazurile de utilizare specifice unui utilizator definit în Figura 7.0(de mai sus):

|  |  |
| --- | --- |
| **Caz de utilizare** | **Descriere** |
| Creare cont | Crearea unui cont de utilizator constă în apelarea endpoint-ului numit ”RegisterUser” cu parametrii string username, string email, string password. În cadrul acestuia se verifică dacă parametrii nu sunt null, respectiv dacă există deja un email la fel în baza de date. De trece validarile, se creează un user ce se adaugă în baza de date, cu o parola hash-uită folosind standardul RFC2898. |
| Autentificare cont | Autentificarea contului constă în apelarea endpoint-ului numit “Login” cu parametrii string email, string password. Se verifică daca aceștia sunt diferiți de null, iar de se trec validările, se verifică de există în baza de date și de parola hash-uită corespunde cu cea oferită prin parametru în varianta nehash-uită. |
| Creare cameră | Crearea unei camere se face folosind endpoint-ul ”CreateRoom” cu parametrii int id, string roomName. Id-ul reprezintă id-ul utilizatorului de creează camera. De id-ul există în baza de date se creează o camera de e adaugată în baza de date cu un nume atribuit acesteia. |
| Schimbare nume cameră | Schimbarea numelui unei camere se face apelând endpoint-ul ”ChangeRoomName” cu parametrii int idRoom, string newRoomName. De există deja această cameră, se updatează numele acesteia cu noul nume. |
| Vizualizare camere de care aparține utilizatorul | Vizualizarea camerelor se face prin apelarea endpoint-ului ”GetRoomsWithGivenMember” cu parametrul int memberId. Această metodă returnează o listă cu camerele în care este inclus utilizatorul cu id-ul memberId. |
| Adăugare utilizator | Adăugarea unui utilizator se face prin apelarea endpoint-ului ”AddMember” cu parametrii int idRoom, string email. De există camera cu acel idRoom, respectiv un utilizator cu acel email oferit ca parametru, email unic stocat în baza de date, și de acesta nu aparține deja camerei în care urmeaza a fi adăugat, se creează o noua entitate tip Member ce e adăugată în baza de date. |
| Părăsirea unei camere | Părăsirea unei camere se poate face apelând endpoint-ul ”RemoveMemberFromRoom” cu parametrii int idUser, int idRoom. Se verifică dacă există o cameră cu acel id unic, respectiv un user care l-a rândul lui aparține acestei camere. Daca trece validările, se schimbă valoarea câmpului isDeleted din entitatea de Member asociata user-ului în true, ceea ce reprezintă că acest user nu mai aparține camerei. |
| Ștergerea unei camere | Ștergerea unei camere se face prin accesarea endpoint-ului ”RemoveRoom” cu parametrul idRoom. De această cameră există, se schimbă valoarea câmpului isDeleted din entitatea de Room asociata id-ului camerei în true, ceea ce reprezintă că această cameră nu mai există. |
| Trimiterea unui mesaj într-o cameră | Postarea unui mesaj într-o cameră se face prin utilizarea endpoint-ului ”SendMessage” cu parametrii int idUser, int idRoom, string message. Se verifică de exista un user si o cameră asociate acelor id-uri, respectiv de nu s-a primit un mesaj gol. De validările reușesc, se adaugă o nouă entitate RoomMessage în baza de date, ce reprezintă mesajul postat în acea cameră de către utilizatorul cu id-ul primit ca parametru, la timpul execuției endpoint-ului. De asemenea se trimite o notificare cu ajutorul websocket-ului implementat cu ajutorul SignalR, pentru a se updata lista de mesaje din acea cameră în cadrul aplicației de apelează acest endpoint, în acest caz în cadrul front-end-ului realizat cu ajutorul Android Studio. |