Messzendzser témalaboratórium

Készítette: Kovács Noémi, Hankóczi Gábor, Gyóni Ákos

Az alkalmazás végleges funkciói

- A felhasználók tudnak regisztrálni és utána be tudnak lépni a megadott felhasználónévvel/e-maillel és jelszóval.
- Bejelentkezéskor megnyílik az első beszélgetés.
- Ha rákattintanak az éppen megnyitott beszélgetés nevére, akkor lenyílik egy menü, ahol lehet váltani, hogy kivel beszél a felhasználó.
- A felhasználók tudnak egymásnak privát szöveges és hang üzeneteket küldeni, amik megmaradnak kilépés után is és visszanézhetőek/hallgathatóak.
- A felhasználók tudnak képeket megosztani egymással, amik megmaradnak kilépés után is és visszanézhetőek(a képek gomb megnyomásával, aztán a kép kiválasztásával a fájl választóból küldhetőek).
- A felhasználók fel tudják hívni egymást, a telefon gomb megnyomásával. Ezután el tudják fogadni a hívást a felugró ablakon belül a felvétel, vagy le tudják rakni a lerakás gombbal.
- Folyamatban lévő hívást be tudják fejezni a lerakás gomb megnyomásával.
- A hívást egy hang is jelzi.
- A felhasználók tudnak egy megosztott whiteboardra több színnel rajzolni, ahol megnyomják a bal egérgombot, ott megjelenik a kiválasztott színből egy pont(színt lehet váltani a színes négyzetekre kattintással), a tábla akkor is frissül a változtatásokkal, ha más rajzol rá.
- A whiteboardot le lehet törölni a törlés gombbal.

Különböző egységek leírása

Frontend

Modul célja

Az alkalmazás minden elemének megjelenítésére szolgáló modul, mely dinamikusan reagál a különböző felhasználói interakciókra.

Változások az eredeti tervhez képest:

A frontend megalkotása során nem történt nagyobb változás az eredeti tervekhez képest. Az alapkoncepció alapján React-et használtunk a UI megjelenítéséhez. A JavaScriptet is alkalmaztuk egyes esemény kezelést igénylő helyen. A Bootstrap alkalmazására viszont nem került sor, mivel úgy gondoltuk, hogy többet fejlődhetünk, ha a projektet csak CSS segítségével valósítjuk meg. Kezdetleges terveink szerint lehetőség lett volna e-mail cím - jelszó kombinációval is belépni, de a sok módosítás következtében végül ez a funkció nem került megvalósításra. A fejlesztés során elérhető volt egy navigációs sáv az alkalmazás tetején, ahol tulajdonképpen bármelyik megjelenítési oldalról el lehetett jutni egy másik tetszőlegesre. Ez a funkció ki lett szedve a kihasználatlanság indokával.

Frontend felépítése:

Az alkalmazásunk Fronted része két fontosabb mappából áll:

- a public mappa, amely az alkalmazáshoz szükséges hang és kép anyagokat tartalmazza. Az audio mappában található csengőhang a VoIP kommunikációhoz szükséges. Az images mappa tartalma a UI felületen megjelenítendő ikonokból áll.
- az src mappa melyben a Frontend forrásfájljai találhatóak:
    - a components mappa tartalmazza az összes React és JavaScript komponenst.
    - a css mappa eleme a megfelelő stílusban való megjelenítéséért felelős.

Osztályok rövid leírása

- **Chat:** a beszélgetési felület megjelenítésére és a felületről indítható interakciók kezdeményezésére használható. A szükséges API lekérdezéseket is implementálja.

- **DecideCall:** valójában egy felugró ablak, amely a Chat komponensben hívódik meg, amennyiben a jelenlegi felhasználót egy másik felhasználó próbálja elérni VoIP kommunikáció segítségével.
- **Home:** az alkalmazás főoldala, ahonnan lehetőség van tovább navigálni a bejelentkezés vagy a regisztrációs oldalra.
- **InCall:** egy felugró ablak, amely a Chat komponensen jelenik meg, amint a felhasználó hívást kezdeményez illetve, ha a hívott felhasználó elfogadta a bejövő hívást.
- **LogForm:** a bejelentkezési felület, ahol a felhasználó be tud lépni a már korábban létrehozott fiókjába.
- **RegForm:** a regisztráció funkció megvalósítására szolgáló osztály.
- **VoipComponent:** a VoIP kommunikáció lebonyolításáért felelős.
- **WhiteBoard:** a rajztábla megjelenítéséért és a rajta végrehajtott interakciókért felelős osztály.
- **WhiteBoardComponent:** a rajztáblán történő események kezelésére szolgál, emellett az API-val való kommunikációt is megvalósítja.

Kiindulás

A program indulásakor egy főoldal jelenik meg, ahol a felhasználó a bejelentkezés vagy a regisztráció gomb megnyomásával átirányításra kerül a megfelelő oldalra.

Regisztráció

A regisztrációért a RegForm osztály felelős, amely felületen a felhasználónak meg kell adnia egy form segítségével a későbbiekben majd a hitelesítéshez szükséges adatait. Ezek az információk pontosabban a nevet, az e-mail címet és egy tetszőleges jelszót jelentik. A funkcióban useSate hook segítségével tárolásra kerülnek a fent említett személyes információk (username, email, password), valamint a majd esetlegesen felmerülő hibaüzenet (message). A felhasználó mindhárom adatot köteles megadni, enélkül nem jön létre a regisztráció, erre a felület is figyelmeztet azzal, hogy hiányosság esetén nem enged továbblépni. A Regisztráció gomb megnyomása következtében form elküldése kerül és létrejön egy aszinkron API hívás, amelyben átadásra kerülnek a beírt adatok az API-nak. Amennyiben az API hívás sikeres volt a useState-ben tárolt adatok törlésre kerülnek és az oldalról automatikus átnavigálás történik a bejelentkezési felületre. Amennyiben bármilyen

probléma felmerült az API-val való kommunikáció során az a megfelelő useState-ben tárolásra kerül, majd a form alján pirosan láthatóvá válik a hiba.

Bejelentkezés

A bejelentkezés megvalósítása a LogForm feladata. Működése nagyon hasonló a regisztráció folyamatához. Tárolás itt csak a felhasználónév, a jelszó és az esetleges hiba során történik. Hasonlóan itt is meg kell adni adatokat, a felhasználónevet és a jelszót, amellyel a felhasználó regisztrálta magát. A form értelemszerűen itt sem engedi üresen hagyni az egyes mezőket a hitelesítés végett, erre hibaüzenet figyelmeztet az üresen hagyott bemeneti mező alatt. A Bejelentkezés gomb megnyomása válta ki az API hívást, ahol elküldésre kerülnek az adatok. A sikeres hívás következtében az adatmezők törlődnek és egy cookie tárolásra kerül, – ami az adott felhasználó session-jéhez tartozik – automatikus navigáció valósul meg a felhasználó beszélgetési felületére. Ha létezik visszakapott hibaüzenet az a form alján kerül kiírásra.

Chat ablak felépítése

A beszélgetés ablak alap felépítése három részre tagolható. A felső sávban van megjelenítve a jelenleg kiválasztott beszélgetéshez tartozó ember neve, a rajztábla megjelenítésére és a hívás kezdeményezésére szolgáló gombok. A középső részben a kiválasztott beszélgetéshez kapcsolódó üzenetek látszanak, amelyek minden üzenetküldés hatására frissülnek SignalR segítségével. Az oldal betöltődésekor alapvetően betöltődnek a felhasználóhoz tartozó chatroom-ok egy API Get hívással. A chatroom-ok közül alapból kiválasztásra kerül a legelső. Minden egyes szoba váltáskor (és az alap beállításakor) becsatlakozik egy új csoportba, ahol megkapja az üzeneteket, amiket rögtön megjelenít ezután. A régebbi üzenetek lekérdezésekor a loadOlderMessages hívódik, ami a már régebb óta ott lévő üzeneteket képes lekérdezni. Itt alapvetően be van állítva hány üzenet legyen lekérdezve. Új üzenetek után a loadNewerMessages hívódik az aktuális üzenetek megjelenítése érdekében. Az üzenetek között lehet görgetni, ha a betöltött üzenetek tetejére értünk, API hívódik a még korábbi üzenetek lekérdezésére, ezt a handleScroll függvény kezeli. A megjelenítésnél típus alapján másféleképp vannak megjelenítve az üzenetek. Minden üzenet felett megjelenik a küldés dátuma. A küldő személytől függően két oldalt jelennek meg az üzenetek.

Szöveges üzenet küldése

A bemeneti mezőben megadott üzenetet egy állapotban eltárolja a funkció majd a küldés gomb hatására API hívás keletkezik, amiben elküldésre kerül az üzenet és a chatroom amiben az üzenet létrejött. Az elküldött üzenet szinte rögtön látszik ezután a felületen.

Hangüzenet küldése

Hangüzenet felvétele egy gomb megnyomásával kezdeményezhető, megnyomva elindul a hang rögzítése. Leállításkor (ugyanaz a gomb) leállítódik a felvétel és eltárolódik a hangüzenet. Majd egy API hívás keretei között elküldjük a hangüzenetet a megfelelő szobához rendelve. Az elküldött üzenet szinte rögtön látszik ezután a felületen.

Kép küldése

A gombot megnyomva egy láthatatlan input mezőt triggerelünk, amely egy fájlt vár bemenő paraméternek. A fájlt kiválasztva API hívódik, amiben küldésre kerül a kép. Az elküldött üzenet szinte rögtön látszik ezután a felületen.

VoIP kommunikáció

Kimenő hívás

A VoIP kommunikáció vizuális megjelenítésére a DecideCall és InCall funkciók lettek kialakítva. A kommunikáció lebonyolításáért a VoipComponent felelős. A Chat komponensből a hívás gomb megnyomása következtében az állapot, ami azt jelzi, hogy a felhasználó hívásban van-e, átállítódik. A hívott fél neve beállításra kerül és meghívódik a VoipComponent példányon a hívás funkció. Itt a komponens létrehozásakor létrejött és kapcsolódott user-agent a SIP szerver felé jelzi a hívási kérelmet, majd vár, hogy a szerver, illetve a hívott fél választ adjon a kérésre. Ha pozitív választ kapott akkor kiépíti az RTP kapcsolatot, és elkezdi lejátszani a kapott hangot. Ezzel párhuzamosan láthatóvá válik az InCall felület, amin látható a hívott fél neve és egy gomb, ami a telefon lerakására szolgál. A hívás lerakása gomb megnyomásával bezáródik az ablak és visszaállítódik a láthatóság hamisra. A Chat funkcióban elutasítás hatására a hangUp függvény hívódik, ami tovább hív a VoipComponentre.Ekkor a VoipComponentben eltárol RTP adatfolyam megszakításra kerül, a hívás másik fele illetve a SIP proxy szerver értesülnek a hívás végének tényéről.

Bejövő hívás

Bejövő hívást a csengőhang jelzi. A VoipComponent egy callback függvény segítségével jelzi a bejövő hívás tényét. Várja, hogy a decline függvény meghívására megtagadja a hívást és erről tájékoztassa a többi résztvevő, vagy az accept függvény meghívására kiépítse az RTP kapcsolatot és megkezdje a hang küldését és fogadását. Az incomingCallHandler meghívásra kerül, itt tárolásra kerül a hívó neve egy állapotban, és beállítódik a bejövő hívást jelző érték. Ekkor láthatóvá válik a DecideCall funkció felülete, ahol ki van írva a hívó neve és két gomb van megjelenítve. Hívás elutasítása következtében ugyancsak a hangUp függvény hívódik és megszakításra kerül a kapcsolat a fentebbiek szerint. Amennyiben a hívott fél elfogadja a bejövő hívást az eddigi ablak láthatatlanná válik és az InCall ablak lesz látható. A hívás lerakása esetén történő események fentebb leírásra kerültek.

Whiteboard elérése, megjelenítése

A rajztábla megjelenítése a WhiteBoard funkcióban van tárolva, az eseménykezelő része pedig a WhiteBoardComponentben. A Chat funkcióból lehet elérni a rajztáblát, a hitelesítésen való túljutás után megjelenik az adott chatroom-hoz tartozó rajztábla, amin már be van töltve a korábbi rajzok eredménye. A rajztáblán lehetőség van különböző előre definiált színekkel rajzolni, amiket a gombok segítségével lehet átállítani. A táblát lehet teljesen törölni, ilyenkor mindenhonnan törlődik az eddig meglévő rajz.

API

Az api végpontot biztosít, amin keresztül a felhasználók hozzáférhetnek a szerveren tárolt adatokhoz és szolgáltatásokhoz. Ennek megoldására az asp.net core által szolgált controllereket használtuk.

- **GetChatroomsController:** Ezen a végponton keresztül lehet lekérni azokat a chat szobákat, aminek a bejelentkezett felhasználó a része. A lekérdezés formája: HTTP GET a szerveren a /api/GetChatrooms címre, illetve a felhasználót identity-vel azonosító sütit is el kell küldeni.
- **GetImageController:** Ezen a végponton keresztül lehet lekérni azokat a korábban feltöltött képeket, amikhez a bejelentkezett felhasználónak van hozzáférése. A lekérdezés formája: HTTP GET a szerveren a /api/GetImage címre, query formájában meg kell adni a lekérdezni kívánt kép tokenjét, illetve a felhasználót identity-vel azonosító sütit is el kell küldeni.

- **GetMessagesController:** Ezen a végponton keresztül lehet lekérni azokat a korábban elküldött üzeneteket, amikhez a bejelentkezett felhasználónak van hozzáférése. A lekérdezés formája: HTTP GET a szerveren a /api/GetMessages címre, a fejlécben meg kell adni a chat szoba azonosítóját, amiből az üzeneteket keressük, hogy hány darab üzenetet szeretnénk lekérni, illetve szűrési feltételnek, hogy milyen irányban keressünk a megadott időponthoz képest (pl.: ha az utolsó 20 üzenetet szeretnénk lekérni, akkor a mostani dátumot adnánk meg, és visszafelé keresnénk), illetve a felhasználót identity-vel azonosító sütit is el kell küldeni.
- **GetVoiceController:** Ezen a végponton keresztül lehet lekérni azokat a korábban feltöltött hangokat, amikhez a bejelentkezett felhasználónak van hozzáférése. A lekérdezés formája: HTTP GET a szerveren a /api/GetVoice címre, query formájában meg kell adni a lekérdezni kívánt hang tokenjét, illetve a felhasználót identity-vel azonosító sütit is el kell küldeni.
- **LoginController:** Ezen a végponton keresztül tud bejelentkezni a felhasználó. A lekérdezés formája: HTTP POST a szerveren a /api/Login címre, a fejlécben meg kell adni a felhasználó felhasználónevét és jelszavát. Sikeres azonosítás esetén a válasz fejlécében beállítja az alapvető azonosításhoz szükséges sütit, illetve json formában elküldi a whiteboard azonosításhoz használt felhasználó tokent is.
- **MessageSenderHub:** Ezen a végponton tud a SignalR csatlakozni a szerver szolgáltatásaihoz.
- **RegisterController:** Ezen a végponton keresztül tud regisztrálni a felhasználó. A lekérdezés formája: HTTP POST a szerveren a /api/Register címre, a fejlécben meg kell adni a felhasználó felhasználónevét, email címét és jelszavát. Sikeres regisztrációt egy OK üzenettel jelzi json formában, hiba esetén a hiba okát adja vissza json formában.
- **SendImageController:** Ezen a végponton keresztül lehet kép üzenetet küldeni. A lekérdezés formája: HTTP POST a szerveren a /api/SendImage címre, a fejlécben meg kell adni a chat szoba azonosítóját, el kell küldeni a felhasználót azonosító sütit, illetve a kérés testében meg kell adni a képadatot. Sikeres feltöltést egy OK üzenettel jelzi json formában, hiba esetén a hiba okát adja vissza json formában.
- **SendVoiceController:** Ezen a végponton keresztül lehet hang üzenetet küldeni. A lekérdezés formája: HTTP POST a szerveren a /api/SendVoice címre, a fejlécben meg kell adni a chat szoba azonosítóját, a hangadat formátumát, a hangüzenet hosszát másodpercben, el kell küldeni a felhasználót azonosító sütit, illetve a kérés testében

meg kell adni a képadatot. Sikeres feltöltést egy OK üzenettel jelzi json formában, hiba esetén a hiba okát adja vissza json formában.

- **SendMessageController:** Ezen a végponton keresztül lehet szöveges üzenetet küldeni. A lekérdezés formája: HTTP POST a szerveren a /api/SendMessage címre, a fejlécben meg kell adni a chat szoba azonosítóját, az üzenetet URL kódolva (https://www.w3schools.com/tags/ref_urlencode.ASP), illetve el kell küldeni a felhasználót azonosító sütit. Sikeres feltöltést egy OK üzenettel jelzi json formában, hiba esetén a hiba okát adja vissza json formában.
- **WebSocketController:** Ezen a WebSocket végponton keresztül tud a whiteboard csatlakozni a szerverhez.

Backend

Változások az eredeti tervhez képest

Eleve az üzeneteket saját megvalósítású szerializálóval akartuk átküldeni a frontend és backend közti kapcsolaton, de aztán rájöttünk, hogy van beépített szerializálás és ezért inkább azt használtuk.
Az új üzeneteket eleinte időközönként pollolta a frontend, de konzulensi javaslatra és mivel az eredeti nem jól skálázható megoldás és leterheli a szervert, ezért inkább áttértünk SignalR használatára, ami szerver oldalról értesíti a szobában lévő felhasználókat, ha jön új üzenet. Ez a megoldás sokkal szebb és hatékonyabb mint amit eleve terveztünk csinálni.
A hitelesítést eleve más módon valósítottuk meg de konzulensi javaslatra átálltunk Entity Framework-re, mert ez hatékonyabb.

Felhasznált könyvtárak

Az kliensek értesítéséhez a Microsoft SignalR könyvtárát használtuk, a hitelesítéshez pedig az Identity Framework-öt.

Osztályok rövid leírása

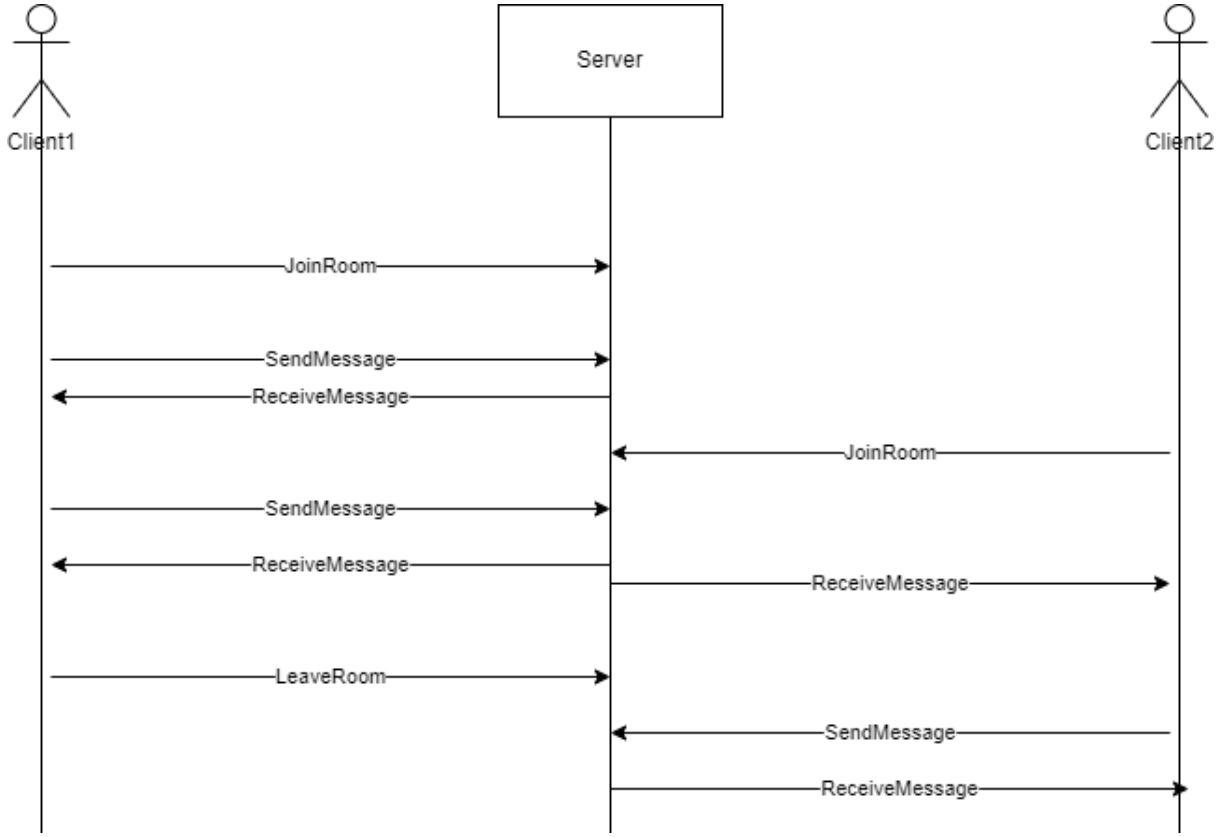
- **MessageManager:** ez az osztály tárol el és kérdez le üzeneteket az adatbázisból. A hang és kép üzeneteket a MediaManager-rel menti fájlba és az adatbázisba csak tokeneket tárol, amivel később el lehet érni. A lekérdezés megadható számú üzenetet ad vissza, ami bármelyik fajtájú lehet (szöveg, hang vagy kép).

- **MediaManager:** ez az osztály tárol el vagy olvas ki képeket, hangokat és Whiteboard-okat fájlba és visszaad róluk tokent, amivel később ki is tudja olvasni őket. Kép- és hang tároláshoz meg lehet adni formátumot is.
- **Message:** absztrakt ősosztálya az üzeneteknek, amiket el kell tárolni. Leszármazottjai: TextMessage, ImageMessage és VoiceMessage. Ezek segítik az adatbázis leképezését.
- **MessageSenderHub:** ez az osztály valósítja meg a SignalR-es Hub-ot, amin keresztül üzeneteket lehet küldeni. Lehet csatlakozni vagy kilépni szobákból és adott szobába üzenetet küldeni.

SignalR működése

A SignalR megvalósítása a MessageSenderHub-ból generálódik a Program.cs-ben ahol hozzáadjuk a szolgáltatásokhoz. A frontend hívásokat “átalakítja” a backenden létező függvényekké és ezáltal tudnak kommunikálni.

SignalR diagram



Entity Framework működése

Adatelérési módszernek az Entity Framework-öt választottuk. Ehez a modelleket legeneráltattuk a korábban létrehozott adatbázis alapján. Ezek a modellek a Messzendzser.Model.DB.Models névtér alatt találhatóak. Az adatbázis kontextus két file-ban található: MySqlDbConnection.cs és MySqlDbConnection.public.cs. Azért választottuk szét a két részt, hogyha esetleg újra kell generálni az adatbázis kontextust, akkor ne kelljen újra megírni az adatlekérdezéshez használt függvényeket. Így egyszerűbbé vált egy újabb absztrakció bevezetése: IDataSource. Az alkalmazásban történő összes adatbáziselérés ezen az interfészen keresztül történik, így ha az adatbázis hozzáférést, vagy magát az adatbázist le kéne cserélni, akkor elég lenne csak itt módosítani a projektet. Az adatbázis felépítését leíró migrációk a Messzendzser.Model.Migrations névtérben találhatóak.

Adatbázis

Az adatbázis MySQL alapú, és a MariaDB Szerver valósítja meg. Az adatséma két fő részből áll:

1. Az identity által generált felhasználói adatokat tároló táblák
2. Az általunk készített táblák melyek az alkalmazás működéséhez tartozó adatokat tárolják

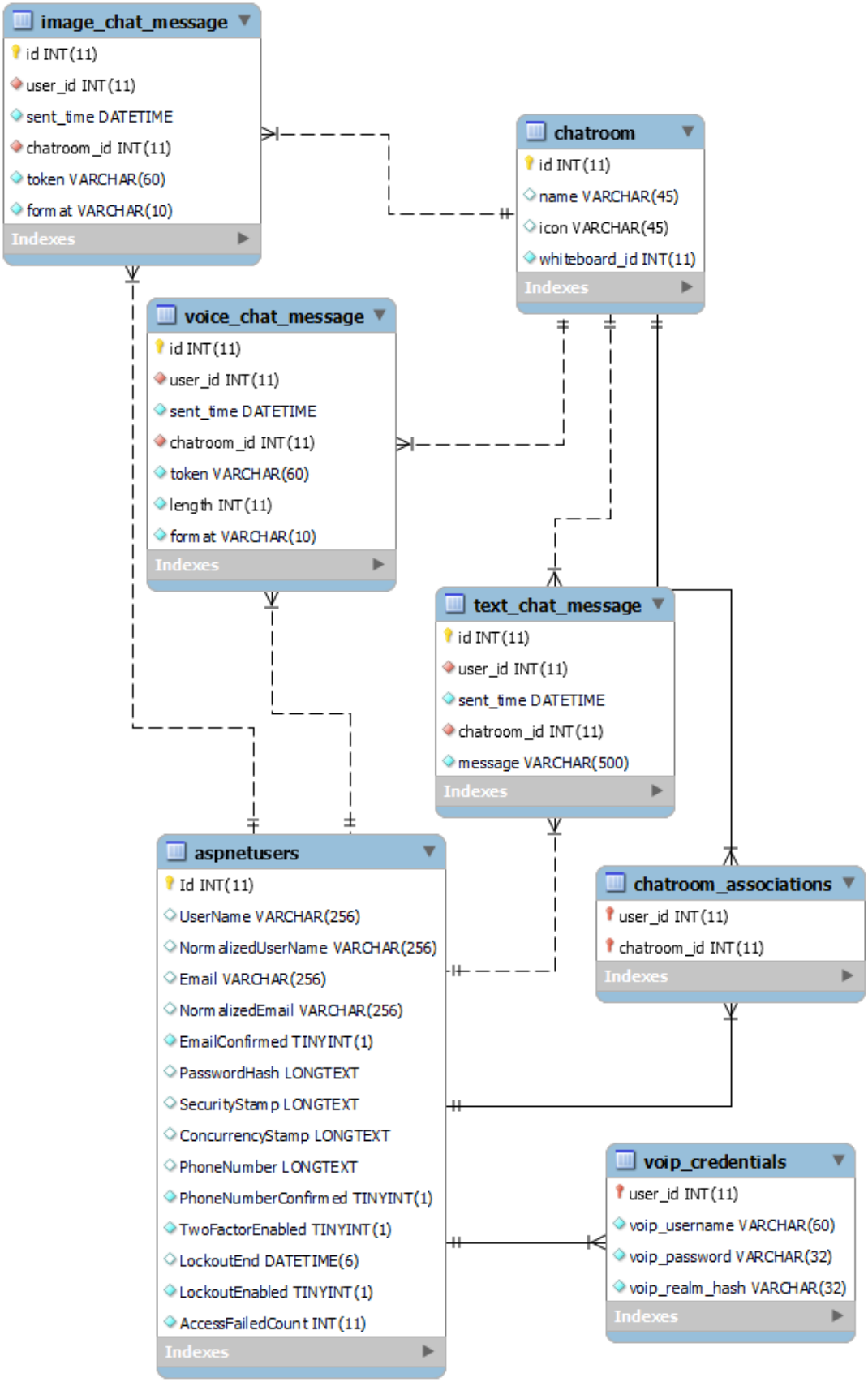
Az általunk létrehozott táblák:

- **chatroom:** A beszélgetési szobákat reprezentálja. Jelen megvalósítás szerint csak ketten kerülnek egy szobába, azonban ennek a táblánk a létezése könnyen megvalósíthatóbá teszi, hogy többen is jelen lehessenek egy szobában.
- **chatroom_associations:** Kapcsolótábla, ami számon tarta, hogy mely felhasználók mely szobákban vannak benne. Több-több kapcsolatot tesz lehetővé a felhasználók és a chat szobák között.
- **text_chat_message:** Szöveges üzeneteket ír le.
- **voice_chat_message:** Hang üzeneteket ír le. Eltárolja a tokent, ami alapján a hang fájl lekérdezhető az api-n keresztül.
- **image_chat_message:** Kép üzeneteket ír le. Eltárolja a tokent, ami alapján a kép fájl lekérdezhető az api-n keresztül.
- **voip_credentials:** A SIP regisztrációhoz szükséges adatokat tárolja.

Kapcsolatok:

- chatroom—aspnetuser: Több-Több kapcsolat
- text_chat_message— chatroom: Egy-Több kapcsolat
- image_chat_message— chatroom: Egy-Több kapcsolat
- voice_chat_message— chatroom: Egy-Több kapcsolat

– Voip_credentials—aspnetuser: Egy-Egy kapcsolat



Whiteboard

Változások az eredeti tervhez képest

Eleve pollinggal akartuk lekérdezni az adatokat a szervertől, de erre elég gyorsan rájöttünk, hogy nem jó megoldás, mert nem jól skálázható és leterheli a szervert. Utána TCP-n keresztül üzenetekkel akartuk értesíteni a klienseket a változtatásokról és a kliensek is TCP üzenetekben küldték volna az új rajzolások adatait. De a böngészők nem engednek nyers TCP kapcsolatot nyitni, ezért áttértünk a WebSocket protocolra és azon keresztül küldtük át az üzeneteket.

Felhasznált könyvtárak

Az üzenetek továbbításához a WebSocket-et, a táblák képeinek elmentéséhez és rajzolásához pedig a SkiaSharp külső könyvtárat.

Osztályok rövid leírása

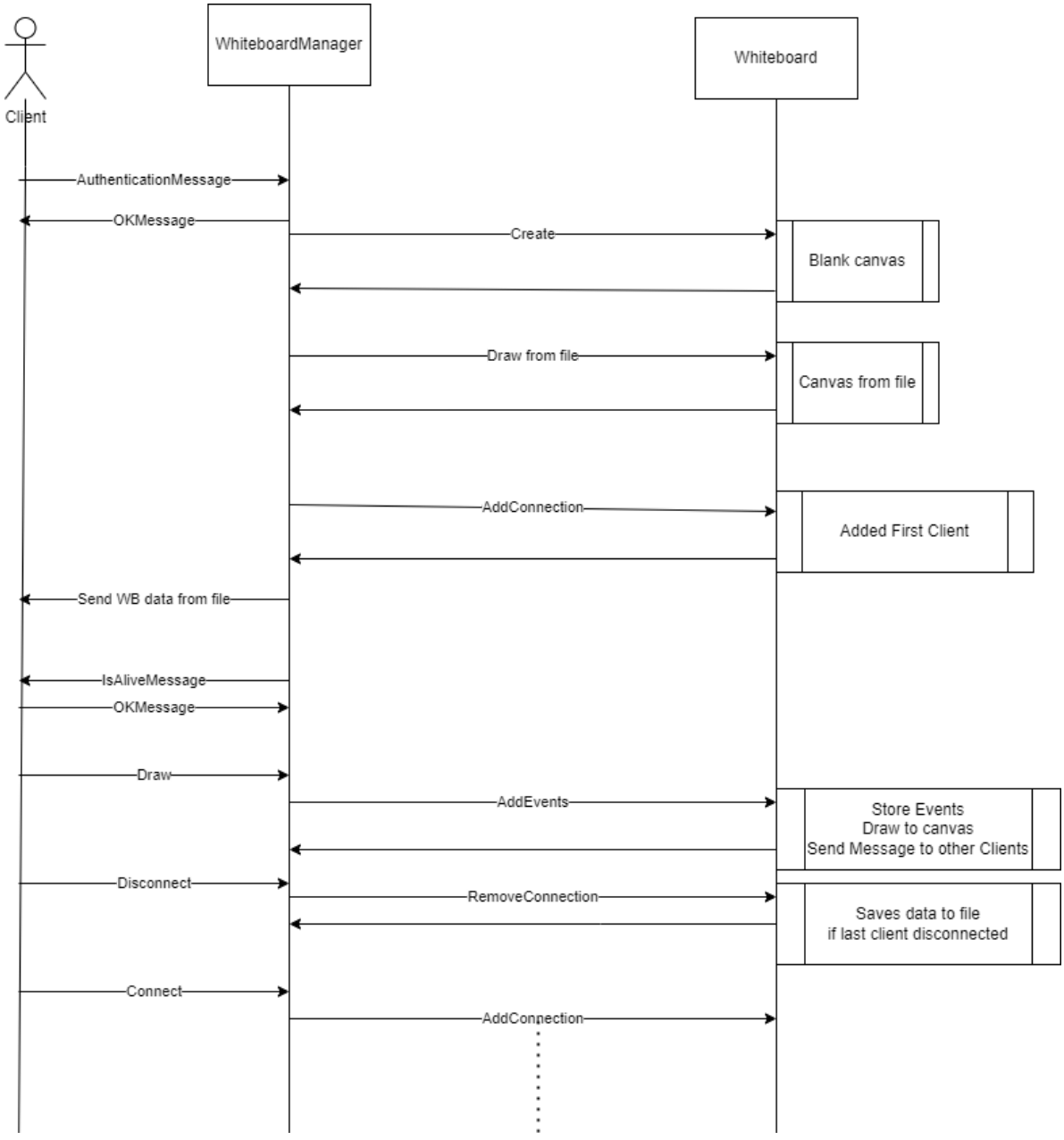
- **WhiteboardManager:** ez az osztály indítja a WebSocket kapcsolatokat és kezeli az üzeneteket, ezen felül chat szobánként is eltárolja a Whiteboard-okat.
- **Whiteboard:** ez az osztály eltárolja a hozzá csatlakozott klienseket és a jelenlegi táblára rajzolt képet, valamint az utolsó mentés óta kapott eseményeket. A WhiteboardManageren keresztül értesíti a klienseket a változásokról.
- **WhiteboardMessage:** absztrakt ősosztálya a Whiteboard üzeneteknek, amiket WebSocketen keresztül szerializálva küldünk át. Több fajtája is van: Authentication, Denied, OK, Event és IsAlive. Ezek később lesznek részletezve.
- **WhiteboardEvent:** absztrakt ősosztálya a Whiteboard eseményeknek, amik a WhiteboardEventMessage üzenetekben vannak eltárolva. Több fajtája is van: Image, Clear, Line és Dot. Ezek később lesznek részletezve.

A whiteboard üzenetek kezelése

A WhiteboardManager osztály WebSocket-en keresztül kap új kapcsolatokat. Ezektől vár egy hitelesítés üzenetet (WhiteboardAuthenticationMessage). Ha az adatok amiket kap egy olyan felhasználót igazolnak, aki hozzáfér a kért szoba Whiteboard-jához, akkor küld egy elfogadó üzenetet (WhiteboardOKMessage) és átlép hitelesített állapotba, ha nem helyesek a kapott adatok akkor elutasítja a kapcsolatot és erről üzenetet is küld (WhiteboardDeniedMessage). Hitelesített állapotban időnként megkérdezi a felhasználót, hogy még csatlakozva van-e, egy

üzenet küldésével (WhiteboardIsAliveMessage), amire vár egy megerősítő üzenetet (WhiteboardOKMessage), ha ilyet nem kap, akkor felbontja a kapcsolatot és ha nincs több felhasználó akkor kimenti a tábla tartalmát fájlba (a MediaManager osztály segítségével). Amikor sikeresen csatlakozik a felhasználó, akkor megkapja a tábla jelenlegi képét egy üzenetben (WhiteboardEventMessage, amiben van egy WhiteboardImageEvent) és a további változtatásokról is küld üzeneteket. Egy üzenetben általában több esemény is lehet(főleg WhiteboardLineEvent a jelenlegi megvalósításban). Hibás üzenet típusok esetén jelez a küldőnek (WhiteboardDeniedMessage).

Whiteboard kapcsolat létrejövésének diagramja



VoIP

Változások az eredeti tervhez képest

Modul célja: Egy SIP proxy szerver megvalósítása volt a cél, amire a kliensek websocket alapú kapcsolattal rá tudnak csatlakozni.
Kezdetben a SIPSorcery (https://github.com/sipsorcery-org/sipsorcery) osztálykönyvtárral próbálkoztunk, mivel ez támogatta az összes SIP lekérdezés típus amire szükségünk volt, illetve a websocket interfészt is, így ideális eszköznek tűnt a feladatra. Miután sikeresen elértük, hogy csatlakozni lehessen (nem web alapú) kliensel, rájöttünk, hogy a könyvtár ugyan megvalósítja a SIP által definiált protokollt, és volt "Proxy" nevű mintaprojektje, de semmilyen mögöttes logikát nem valósít meg.
Ezek után a Lumisoft.net (https://github.com/ststeiger/Lumisoft.Net.Core) könyvtárral kezdtünk el ismerkedni, mivel ez a proxy szerver minden funkcióját megvalósította, azonban nem tudott websocketen keresztül kommunikálni. Ennek a bővítése mellett döntöttünk.
A könyvtárt főleg a LumiSoft.Net.WebSocket névtér megírásával egészítettük ki, de emellett pár már meglévő osztályt is módosítanunk kellett.

LumiSoft.Net.WebSocket névtér osztályai:

- WebSocket_Server
  - ListeningPoint
  - WebSocket_Acceptor
- WebSocket_ServerSession
- WebSocket_ServerSessionEventArgs
- WebSocket_Session
- WebSocket_SessionCollection
- WebSocket_Stream

Az eredeti könyvtárban módosított osztályok:

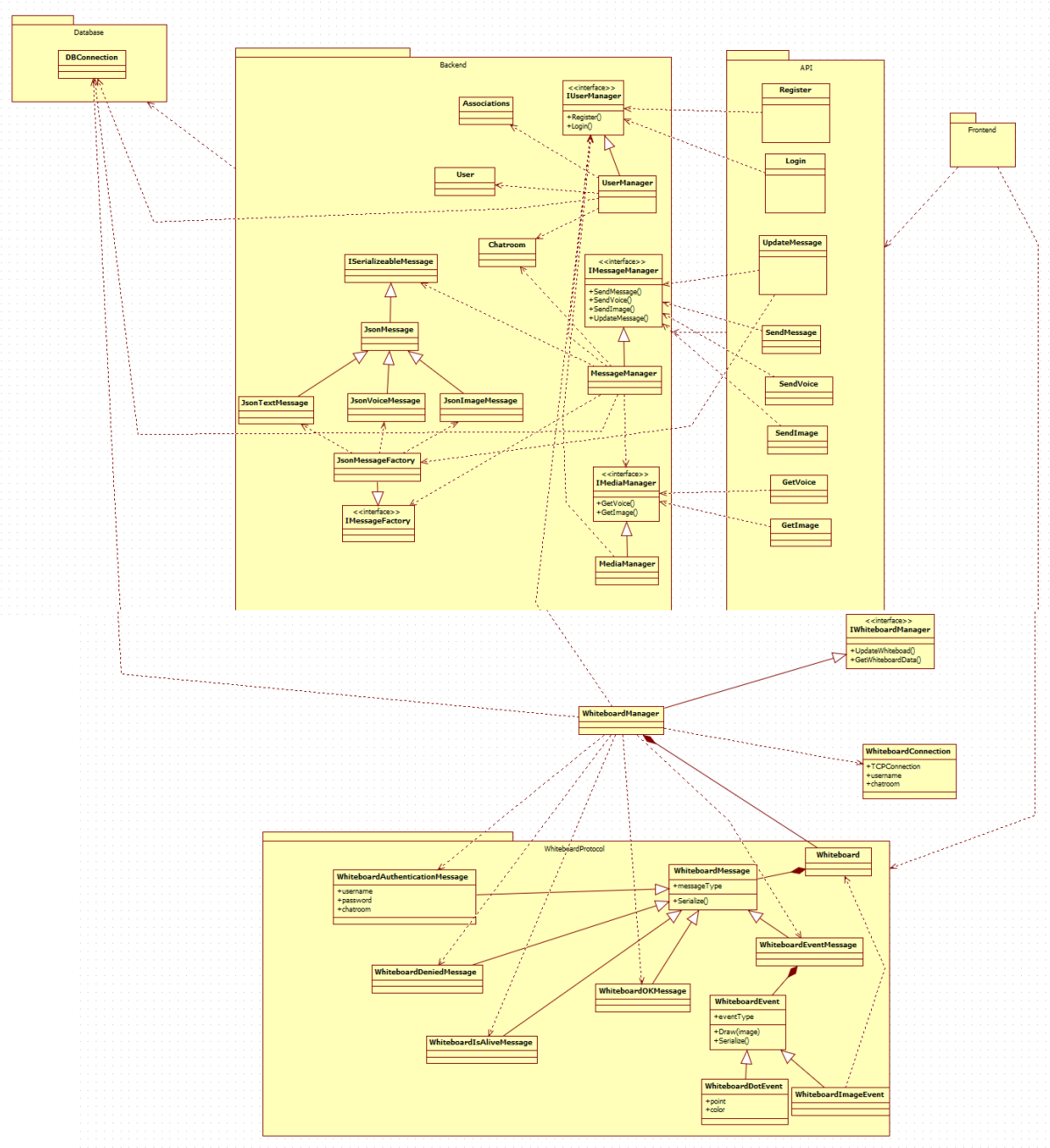
- SIP_Flow
- SIP_TransportLayer

Módosított osztályok:

- **SIP_Flow**: Az osztályt összesen annyira módosítottuk, hogy a knostruktora létre tudja hozni az osztályt úgy, hogy az WebSocket_Server osztályra hivatkozzon a korábbi TCP_Server helyett.
- **SIP_TransportLayer:** A módosítás itt is hasonló, felvettük a CreateFromWebSocketSession nevű függvényt, hogy létre tudjon hozni SIP_Flow-t és SIP_TransportLayert úgy, hogy WebSocket_Server-re hivatkozzon a korábbi TCP_Server helyett.
- Hozzáadott osztályok:
- **WebSocket_Server:** Az osztály lényege, hogy egy vagy több végponton hallgatózzon bejövő TCP kapcsolatokra, és ha érkezik kapcsolódási kérelem, akkor kezelje azt a kapcsolatot, illetve tartsa számon a már beérkezett kapcsolatokat.
- **ListeningPoint:** (a WebSocket_Server egy belső osztálya) Egy végpontot reprezentál, illetve számon tartja a végponthoz tartozó socketet.
- **WebSocket_Acceptor:** (a WebSocket_Server egy belső osztálya) A kapcsolatokat fogadó folyamatot reprezentálja, aszonkron módon kezeli, fogadja el, illetve tárolja a kapcsolatokat, így biztosítva a nagy áteresztőképességet.
- **WebSocket_ServerSession:** Egy websocket kapcsolatot testesít meg. Elárolja a kapcsolathoz tartozó általános adatokat (pl.: kapcsolat kiépülésének ideje, szerver objektum, titkosított-e a kapcsolat, lokális kapcsolat azonosító…) Ez az osztály indítja el a TCP fölötti kommunikációt a kliens és a szerver között. Először a kapcsolatot egy SslStream-be csomagolja, így egy biztonságos, titkosított adatfolyamot kapunk. Ezek után az ssl folyamot becsomagolja egy WebSocket_Stream-be, majd elindítja annak kézfogását, így a végleges adatfolyam már csak a WebSocket protokollon belüli SIP kommunikáció lesz.
- **WebSocket_ServerSessionEventArgs:** Szerver eseményeket leíró adatok leírására szolgál.
- **WebSocket_SessionCollection:** A WebSocket kapcsolatokat leíró osztályok gyűjteménye, ami lehetőséget nyújt a kapcsolatok lekérdezésére akár IP cím alapján.
- **WebSocket_Stream:** Stream-ből leszármazó osztály, ami a WebSocket protokollt valósítja meg. Eltárolja a beágyazott Stream-et, illetve véghez tudja vinni a kézfogást, amit az RFC 6455 ír le (https://www.rfc-editor.org/rfc/rfc6455). Felelős az üzenetek maszkolásáért.

- **DataFrame:** (A WebSocket_Stream egy belső osztálya) Az RFC 6455 5. Szekciója által leírt adatkeretet valósítja meg. Felelős továbbá az adatok validációjáért.

Kezdeti UML Diagram



A diagramon látható hierarchia alapján terveztünk eleve elindulni a fejlesztéssel, de időközben néhány helyen ettől eltértünk, ami a végleges verzióban is látszik.