Proiect: Chat Client-Server

Documentație

Nume Student: Tcaci Liviu
Grupa: 30221

Facultatea de Automatica şi Calculatoare
4 Ianuarie 2025

Cuprins

1 Enunțul Problemei

Proiectul urmărește dezvoltarea unei aplicații de tip **Chat Client-Server**, care să asigure comunicarea în timp real între utilizatori printr-o interfață web. Din perspectiva cerințelor funcționale, se doresc următoarele aspecte:

1. **Autentificare şi Înregistrare:** Utilizatorii trebuie să aibă posibilitatea de a se înregistra în aplicație, oferind un nume de utilizator şi o parolă. Baza de date internă va stoca aceste informații. De asemenea, se cere implementarea unei funcționalități de "Forgot Password" (resetare parolă) care să permită schimbarea parolei fără a avea acces la cea veche (situație utilă când utilizatorul şi-a uitat parola).
2. **Roluri:** Există două tipuri principale de utilizatori: *user obişnuit* şi *admin*. Un utilizator cu rol de admin are acces la funcții suplimentare: blocarea/deblocarea altor utilizatori, schimbarea rolurilor, moderarea mesajelor, ştergerea istoricelor etc.
3. **Mesaje Publice:** Orice utilizator logat poate trimite mesaje publice care să fie vizibile tuturor celorlalți utilizatori conectați la chat. Aceste mesaje ar trebui stocate într-o bază de date pentru a permite consultarea istoricului.
4. **Camere de Chat:** Utilizatorii pot crea "camere" tematice (e.g., pentru echipe sau subiecte specifice) şi se pot alătura altor camere existente. Mesajele trimise într-o cameră sunt vizibile doar de către participanții acelei camere. De asemenea, se vor salva într-o bază de date, astfel încât să existe un istoric al discuțiilor din fiecare cameră.
5. **Mesaje Private:** Aplicația trebuie să permită trimiterea de mesaje private (unu-la-unu), identificate pe baza numelui de utilizator destinatar. Aceste mesaje rămân confidențiale şi nu sunt vizibile altor persoane.
6. **Funcționalități de Admin:**
   - *Blocare / Deblocare:* Un admin poate bloca un utilizator (care nu va mai putea trimite şi primi mesaje), dar şi debloca ulterior.
   - *Schimbare Rol:* Un admin poate promova un utilizator la rolul de admin sau invers, retrograda un admin la user.
   - *Moderare Mesaje:* Posibilitatea de a şterge/edita anumite mesaje dacă sunt nepotrivite.
   - *Ştergere Istoric:* Adminul poate goli istoricul mesajelor publice sau al unei camere anume, dacă situația o impune.
7. **Istoric Mesaje şi Clear Chat:** Este esențial să existe posibilitatea de a vizualiza istoricul mesajelor (atât public, cât şi al unei camere date) şi, în acelaşi timp, o funcționalitate de "clear chat" (doar pentru admin) care să şteargă din baza de date aceste mesaje.
8. **Notificări:** Sistemul va afişa notificări utilizatorilor atunci când cineva se conectează, se deconectează, intră/iese dintr-o cameră, este blocat etc., pentru a-i menține la curent cu dinamica chatului.

Pe scurt, obiectivul principal este **realizarea unei platforme de discuție online** (tip Slack/Discord minimal), cu un model de permisiuni bazat pe rol, cu stocarea informațiilor în bază de date (*SQLite*) şi cu interfață Web (HTML, CSS, JavaScript) pentru client. Serverul va fi implementat în *Python* folosind *Flask-SocketIO* pentru gestionarea comunicației WebSocket (real-time).

La nivel arhitectural, se vizează separarea clară a logicii server de partea de interfață (client), folosind protocolul *Socket.IO* pentru trimiterea şi recepția de mesaje în timp real. De asemenea, pentru gestionarea autentificării, setarea de roluri, blocarea/deblocarea utilizatorilor etc., se va folosi o bază de date relațională conform diagramei entitate-relație, în care sunt definite tabele

pentru Utilizatori, Mesaje, Camere de Chat, Notificări şi relația many-to-many între Utilizatori şi Camere (UserChatRoom).

2 Instrumente Utilizate

Proiectul a fost dezvoltat folosind un set de tehnologii şi unelte software care acoperă atât partea de *backend* (server), cât şi cea de *frontend* (client), precum şi managementul bazei de date şi partea de documentație.

2.1 Limbaj de Programare şi Framework Server

Python 3.10 (sau versiuni similare) a fost ales drept limbaj principal de programare pentru partea de server datorită ecosistemului său bogat şi a modului simplu de dezvoltare a aplicațiilor web. Pentru realizarea componentelor de server, s-au utilizat:

- **Flask** – Un micro-framework pentru dezvoltare web scris în Python, care oferă uşurință în crearea de rute HTTP şi flexibilitate în arhitectura proiectului.
- **Flask-SocketIO** – O extensie a Flask care gestionează comunicațiile *real-time* folosind *Socket.IO*. Aceasta permite trimiterea/recepția de mesaje instantanee între server şi clienți, facilitând funcționalitățile de chat.

2.2 Bază de Date şi ORM

Pentru stocarea datelor despre utilizatori, mesaje, camere de chat etc., s-au folosit:

- **SQLite** – O bază de date relațională uşor de integrat, care nu necesită un server dedicat, fiind astfel potrivită pentru proiecte de dimensiune mică şi medie. Fişierul `chat.db` conține toate tabelele şi relațiile (Utilizatori, Mesaje, CamereDeChat, Notificari, UserChatRoom).
- **SQLAlchemy** – O bibliotecă ORM (Object-Relational Mapping) pentru Python, care facilitează definirea şi manipularea tabelelor şi interogărilor în mod obiectual, fără a scrie direct cod SQL. Diagrama entitate-relație (ER) a fost transpusă în *model classes*, definind cu claritate relațiile *one-to-many* şi *many-to-many*.

2.3 Frontend (HTML, CSS, JavaScript)

- **HTML5** – Structura principală a interfeței web, incluzând formularele de înregistrare, login, spațiul pentru mesaje publice, liste de camere de chat şi zona de administrare.
- **CSS3** – Definirea stilurilor pentru layout, culori, fonturi şi plasarea componentelor pe ecran. Uneori s-au folosit fişiere CSS externe pentru a organiza codul de stil.
- **JavaScript (ES6+)** – Logică la nivel de client, gestionând evenimente UI (click, submit), comunicarea real-time cu serverul prin Socket.IO, actualizarea dinamica a interfeței în funcție de evenimentele primite.
- **Socket.IO (Client)** – Biblioteca JavaScript care se conectează la serverul Flask-SocketIO, gestionând evenimente precum `connect`, `disconnect`, `message`, `private_message`, etc.

2.4 Instrumente de Dezvoltare şi Management

- **PyCharm** – IDE sau editor de cod utilizat pentru a scrie şi organiza fişierele Python, HTML, CSS, JS.
- **Git** – Sistemul de control al versiunilor folosit pentru a urmări modificările în codul sursă, a colabora şi a menține un istoric clar al evoluției proiectului.

- **GitHub / GitLab** – Spațiu de găzduire a repository-ului, folosind feature-urile de issue tracking, pull requests etc., acolo unde a fost cazul.

3 Diagrame UML

În această secțiune vom prezenta toate diagramele UML relevante pentru proiectul nostru de Chat Client-Server. Aceste diagrame au fost generate folosind cod Mermaid (în timpul dezvoltării), dar vor fi incluse sub formă de imagini în document (capturi .png/.jpg). Fiecare diagramă este explicată mai jos, menționând contextul și scopul său.

3.1 Diagrama Cazurilor de Utilizare

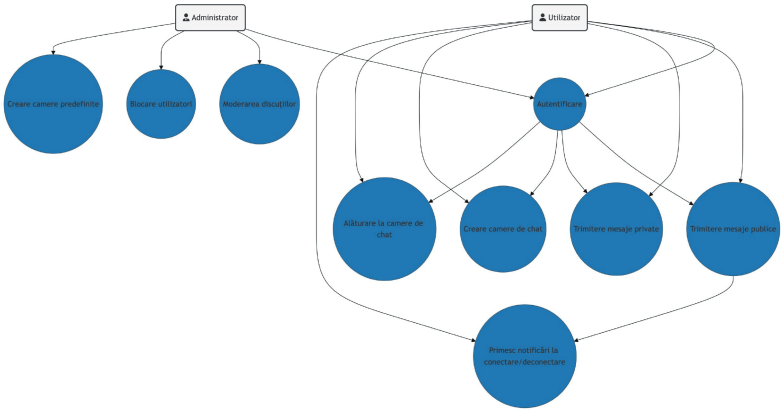

Figura 1: Diagrama Cazurilor de Utilizare (User & Admin)

Descriere Generală: Diagrama cazurilor de utilizare (*Use Case Diagram*) evidențiază interacțiunile de nivel înalt dintre actori (**Utilizator** şi **Administrator**) şi sistem. Diagrama cuprinde următoarele cazuri de utilizare:

- **UC1: Autentificare** (comun User & Admin)
- **UC2: Trimitere Mesaje Publice** (User)
- **UC3: Trimitere Mesaje Private** (User)
- **UC4: Creare Camere de Chat** (User)
- **UC5: Alăturare la Camere** (User)
- **UC6: Notificări la Conectare/Deconectare** (User)
- **UC7: Moderarea Discuțiilor** (Admin)
- **UC8: Blocare Utilizatori** (Admin)
- **UC9: Creare Camere Predefinite** (Admin)

În diagrama de mai sus, săgețile arată relația de "interacțiune" dintre actor şi cazurile de utilizare, iar extinderile UC1 → UC2/UC3/UC4/UC5 denotă că autentificarea este o condiție prealabilă pentru funcționalitățile de mesagerie, camere etc.

Rolurile Actorilor:

- **Utilizator (User):**
  - *Acces* la UC1 (autentificare), fiind necesar să introducă un cont valid.
  - *Drepturi normale* de a trimite mesaje publice (**UC2**) sau private (**UC3**), precum şi de a crea camere (**UC4**) ori de a se alătura camerelor existente (**UC5**).

  - *Primesc notificări* (UC6) la fiecare conectare/deconectare a altor useri, inclusiv la intrarea/ieşirea din camere.
- **Administrator (Admin):**
  - În plus faţă de drepturile de "User", Adminul se poate ocupa de **Moderarea Discuțiilor** (UC7), care implică editarea/ştergerea de mesaje nepotrivite din camere.
  - Poate **bloca** (UC8) utilizatori care încalcă regulile, ceea ce le restricționează accesul la sistem.
  - **Crează camere predefinite** (UC9) — camere speciale la care oricine se poate alătura fără a depinde de un anumit creator (ex.: "anunțuri").

Detalii Fiecare Caz de Utilizare:

UC1: Autentificare
- *Scop:* Permite identificarea utilizatorului în sistem (username + parolă).
- *Flux principal:* Userul introduce credențialele, sistemul le validează consultând baza de date. Dacă e valid, userul primeşte accesul la restul cazurilor de utilizare.
- *Actori implicați:* Utilizator, Administrator.
- *Extensii:* Forgot Password, mesaje de eroare la date invalide.

UC2: Trimitere Mesaje Publice
- *Scop:* Un utilizator să poată trimite un mesaj vizibil tuturor utilizatorilor conectați.
- *Flux principal:* Userul scrie un mesaj, dă "send", serverul îl distribuie prin SocketIO şi îl salvează (tabelul Mesaje cu tip_mesaj = "public").
- *Precondiție:* Utilizatorul trebuie să fie autentificat (UC1).

UC3: Trimitere Mesaje Private
- *Scop:* O discuție confidențială între 2 useri.
- *Flux principal:* Userul alege un destinatar (din lista celor conectați), compune mesajul; serverul îl trimite doar destinatarului.
- *Precondiție:* UC1 (autentificare reușită).

UC4: Creare Camere de Chat
- *Scop:* Permite unui user să creeze o cameră nouă (ex.: "proiectX").
- *Flux principal:* Se introduc nume, descriere, datele sunt trimise serverului, serverul adaugă un nou rând în `CamereDeChat`, userul devine creator (creator_id).

UC5: Alăturare la Camere de Chat
- *Scop:* Utilizatorul alege să intre într-o cameră deja existentă.
- *Flux principal:* Serverul verifică dacă accesul este permis, dacă da, îl adaugă în `UserChatRoom`. Ulterior, userul vede/trimite mesaje doar în acea cameră.

UC6: Notificări la Conectare / Deconectare
- *Scop:* Sistemul să anunțe userii despre cine intră/iese din chat, cine se alătură camerelor etc.
- *Flux principal:* La fiecare connect/disconnect, serverul emite un eveniment "usernotification" cu datele necesare.

UC7: Moderarea Discuțiilor (Admin)

- *Scop:* Administratorul să poată edita/şterge mesaje nepotrivite, menținând un conținut adecvat.
- *Flux principal:* Adminul se alătură camerei, identifică mesajul problema, îl modifică/şterge din DB, iar serverul actualizează conținutul pentru toți userii camerei.

UC8: Blocare Utilizatori (Admin)

- *Scop:* Adminul restricționează accesul unui user la platformă.
- *Flux principal:* Adminul selectează userul, serverul setează userul ca "blocked" şi îl deconectează imediat din chat. Dacă se doreşte ulterior, se poate "unblock".

UC9: Creare Camere Predefinite (Admin)

- *Scop:* Adminul poate crea camere cu statut special (ex. "anunțuri globale").
- *Flux principal:* Se introduc datele, serverul înregistrează camera ca "predefinită" (poate fi un flag `isPredefined`), toți userii o pot vedea automat.

3.2 Diagrame de Activități

În continuare, vom prezenta principalele diagrame de activități (*activity diagrams*) pentru fiecare caz de utilizare. Fiecare diagramă este salvată în format imagine (ex. `mermaid_UC1.png`), dar provine din codul mermaid de mai jos.

3.2.1 Activitatea: Autentificare

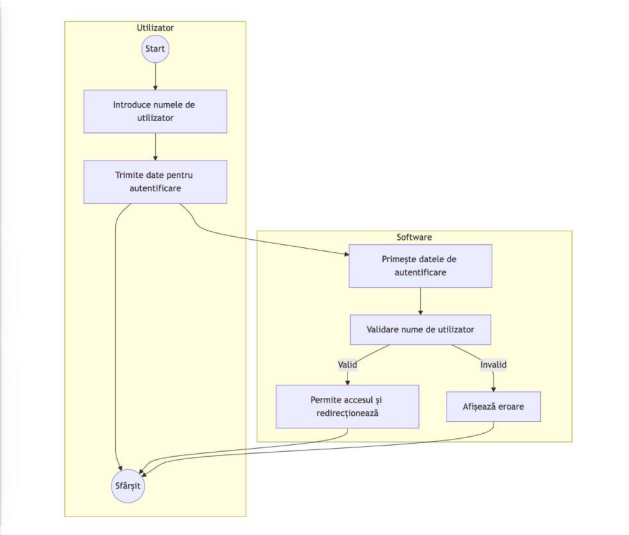

Figura 2: Diagrama de Activități - Autentificare

Explicație:

- **Utilizatorul** introduce credențialele (nume/parolă).
- **Software-ul** primește datele, le validează, iar dacă sunt corecte, permite accesul, altfel afișează eroare.
- Se încheie procesul după ce userul este logat cu succes sau i se cere să reîncerce.

3.2.2 Activitatea: Trimitere Mesaje Publice

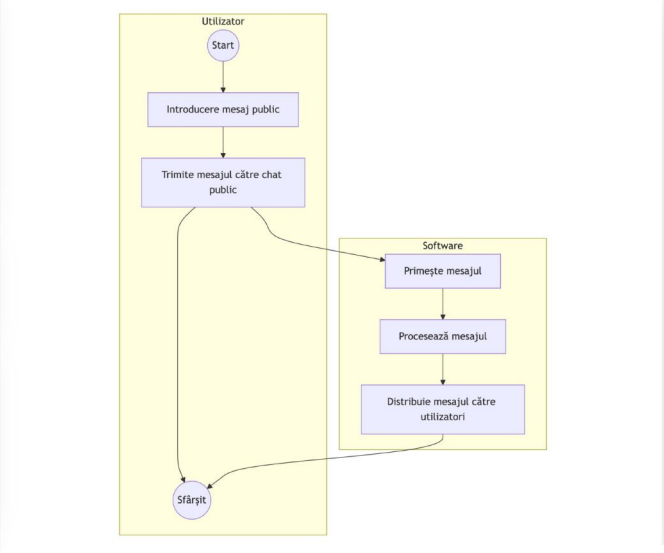

Figura 3: Diagrama de Activități - Mesaje Publice

Explicație:

- Utilizatorul introduce mesajul într-un formular/textarea.
- Serverul primește mesajul, îl procesează (opțional persistă în DB) şi îl transmite tuturor clienților conectați.
- Diagrama subliniază împărțirea între partea de **Utilizator** şi **Software**.

3.2.3 Activitatea: Trimitere Mesaje Private

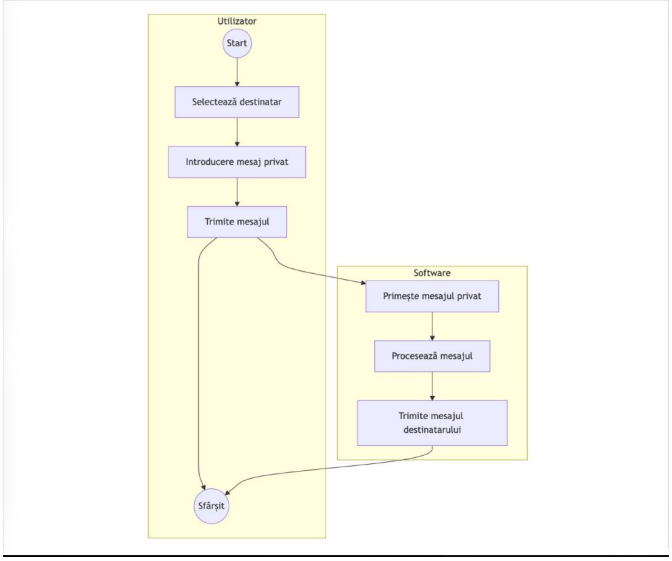

Figura 4: Diagrama de Activități - Mesaje Private

Explicație:

- Utilizatorul îşi alege destinatarul (listă de useri online, de ex.) şi compune mesajul.
- Mesajul este preluat de server, marcat ca *privat*, şi livrat doar destinatarului.

- Diagrama evidențiază faptul că restul userilor nu primesc acel conținut.

3.2.4 Activitatea: Creare Camere de Chat

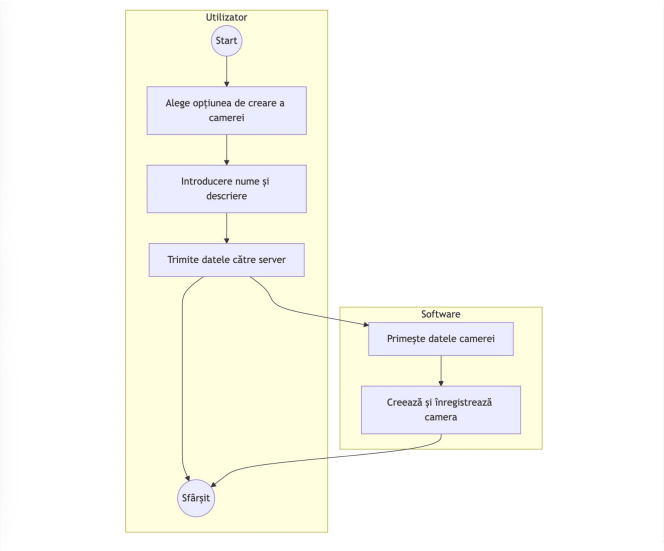

Figura 5: Diagrama de Activități - Creare Cameră

Explicație:

- Userul alege opțiunea de "Create Room" şi completează numele/descrierea camerei.
- Serverul înregistrează camera (ex. creează un rând în tabelul `CamereDeChat`) şi confirmă succesul.

3.2.5 Activitatea: Alăturare la Camere de Chat

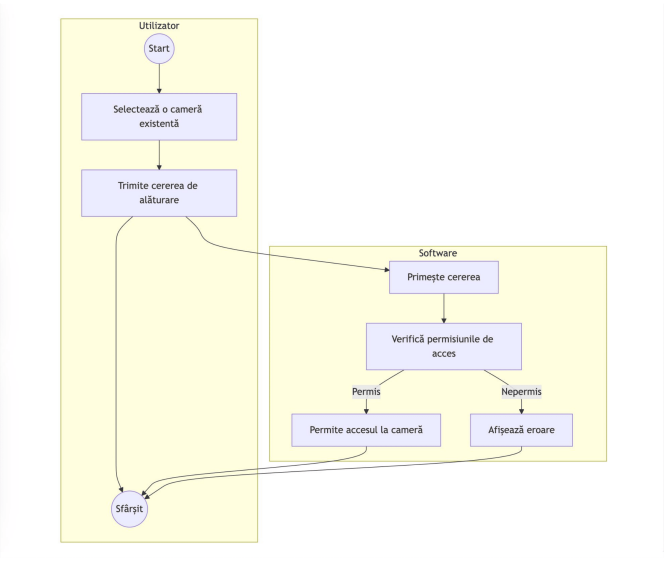

Figura 6: Diagrama de Activități - Alăturare la Cameră

Explicație:

- Utilizatorul selectează o cameră din lista celor existente.

- Serverul verifică permisiunile (sau dacă e nevoie de parolă la cameră) şi, dacă e permis, îl adaugă în `UserChatRoom` şi îi permite accesul la discuții.

3.2.6 Activitatea: Notificări la Conectare/Deconectare

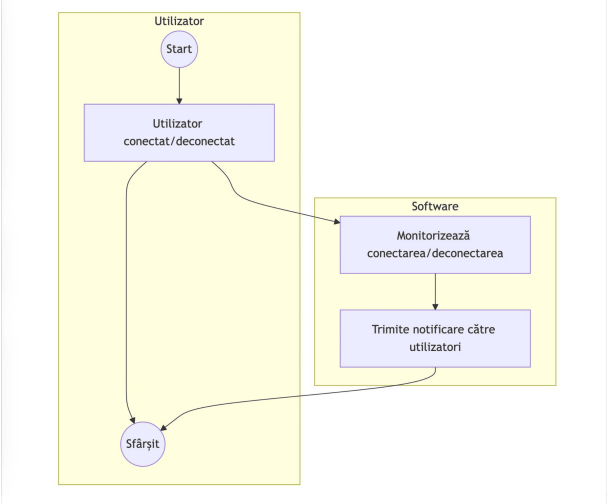

Figura 7: Diagrama de Activități - Notificări

Explicație:

- De fiecare dată când un utilizator intră/iese din sistem, serverul detectează evenimentul.
- Trimite o notificare către toți userii prin SocketIO, astfel încât aceştia să vadă "X s-a conectat/deconectat".

3.2.7 Activitatea: Moderarea Discuțiilor (Admin)

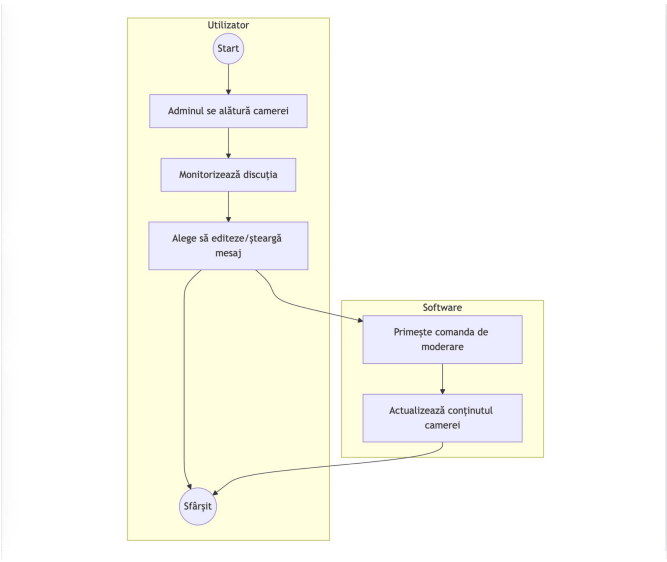

Figura 8: Diagrama de Activități - Moderare

Explicație:

- Adminul se alătură unei camere pentru a observa mesajele.

- Dacă detectează conținut nepotrivit, poate edita/şterge acel mesaj (serverul actualizează conținutul stocat).

3.2.8 Activitatea: Blocare Utilizatori (Admin)

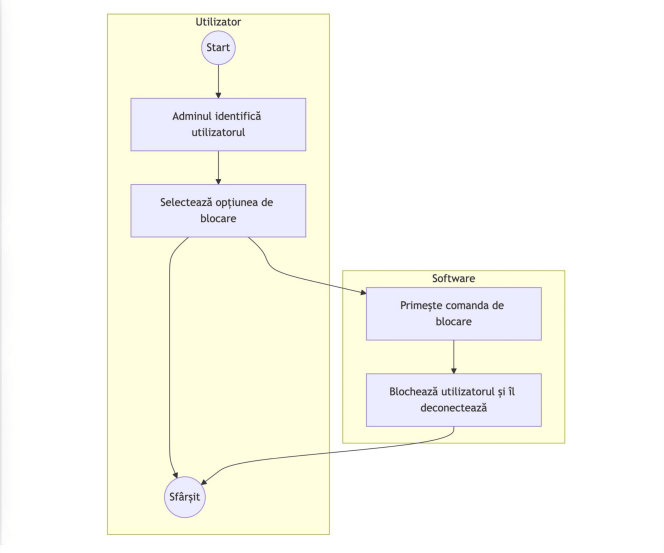

Figura 9: Diagrama de Activități - Blocare Utilizator

Explicație:

- Un Admin identifică userul pe care doreşte să îl blocheze.
- Serverul marchează acel user ca `blocked` în bază de date şi îl deconectează de la chat.

3.2.9 Activitatea: Creare Camere Predefinite (Admin)

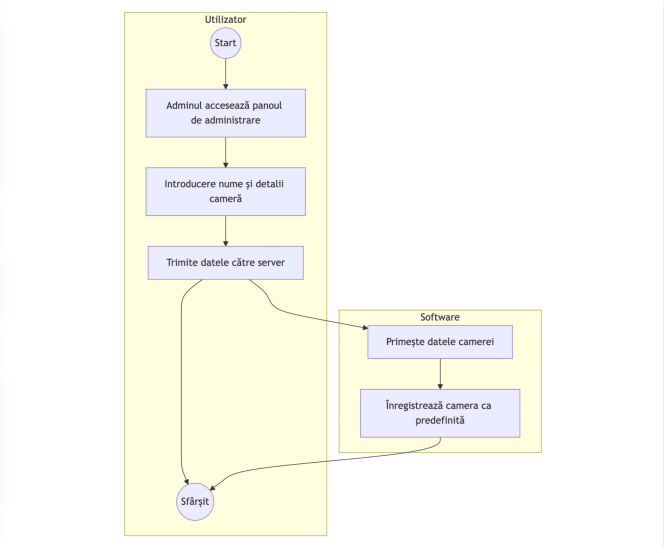

Figura 10: Diagrama de Activități - Camere Predefinite (Admin)

Explicație:

- Adminul poate crea camere speciale cu un statut predefinit (ex. “anunțuri globale”).
- Datele sunt transmise la server şi se stochează la fel ca la orice altă cameră, dar cu un flag *isPredefined = true*.

3.3 Diagrama ER (Entitate-Relatie)

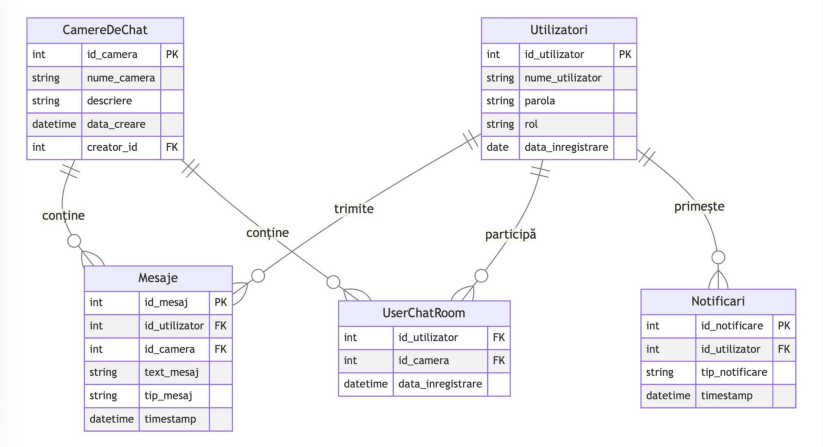

Figura 11: Diagrama ER a aplicației (Utilizatori, Mesaje, CamereDeChat, Notificari, UserChatRoom)

Explicație:

- **Utilizatori** – conține id, nume, parolă, rol, dată înregistrare, etc.
- **Mesaje** – fiecare mesaj stochează userul emitent, camera (dacă e public/room) şi timestamp.
- **CamereDeChat** – tabele cu nume, descriere, date despre creator.
- **Notificari** – pot păstra evenimentele (ex. user conectat, blocat, etc.).
- **UserChatRoom** – tablă de legătură many-to-many între useri şi camere.

3.4 Diagrama de Componente

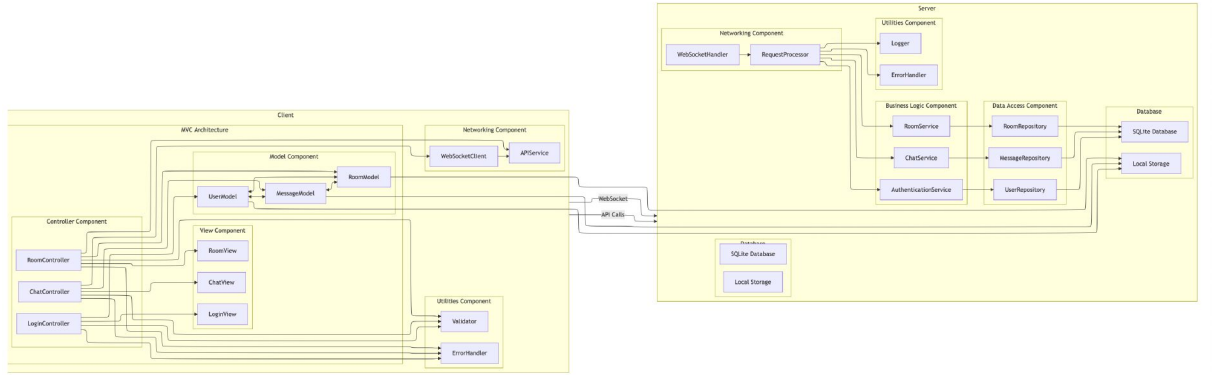

Figura 12: Diagrama de Componente (Client/Server)

Explicație:

- **Server** este împărțit în componente de Networking (WebSocketHandler), Business Logic (ex. ChatService) şi Data Access (repo-uri), plus o bază de date SQLite.
- **Client** este structurat conform arhitecturii MVC, folosind modele, view-uri, controllere, plus un modul Networking (WebSocketClient) şi un modul de stocare locală (LS).

3.5 Diagrame de Pachete

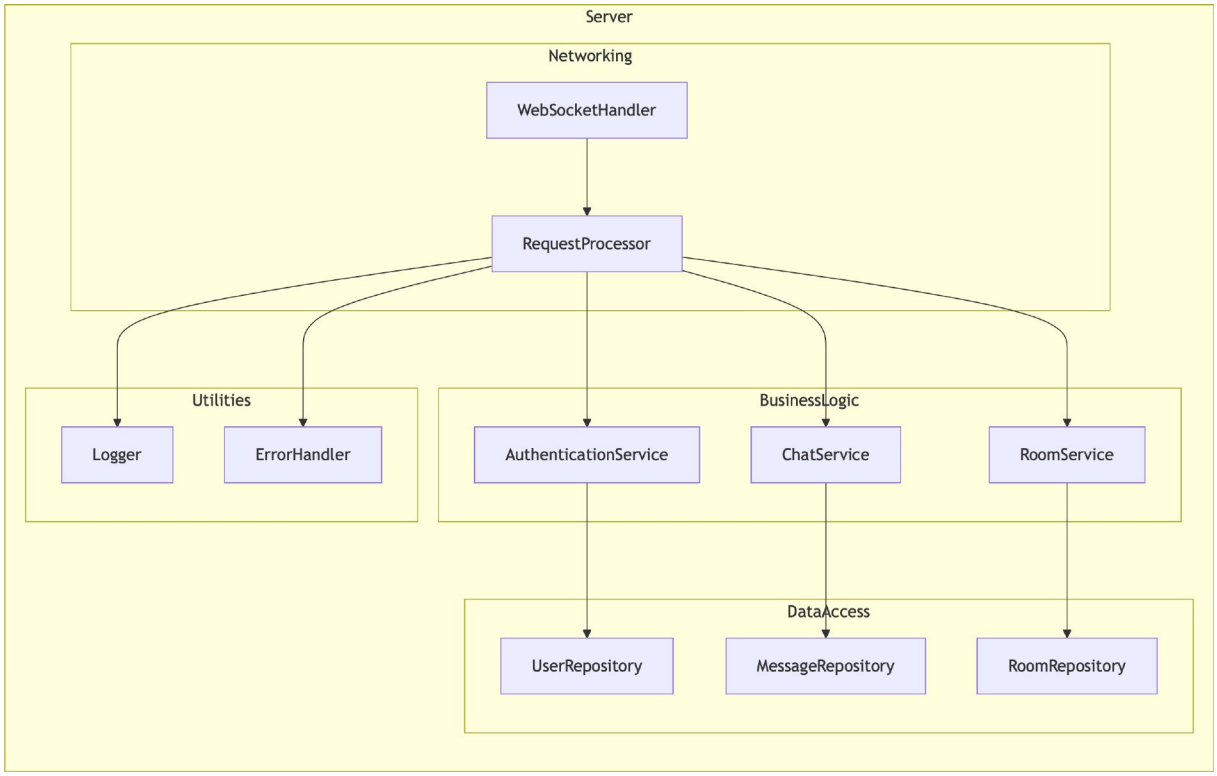

Figura 13: Diagrama de Pachete - Server

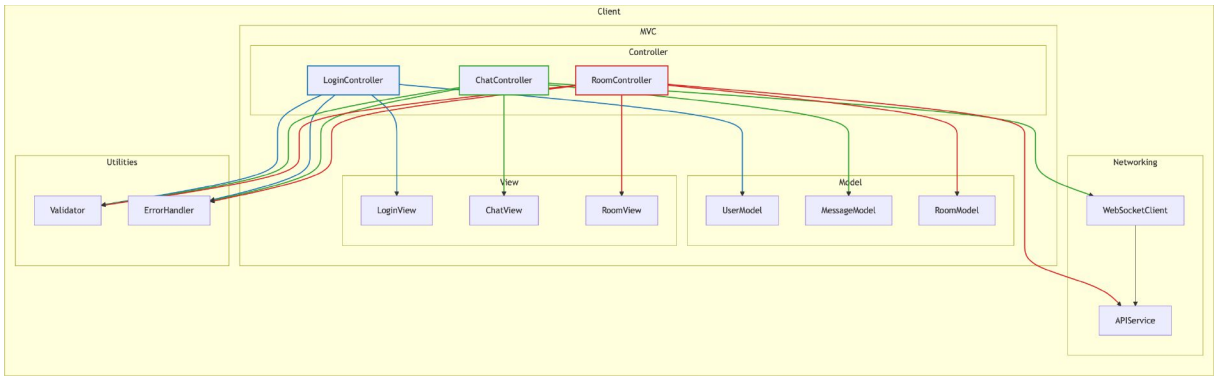

Figura 14: Diagrama de Pachete - Client

Explicație:

- **Server** e separat în pachetele: Networking, BusinessLogic, DataAccess, Utilities.
- **Client** are pachetele MVC (Model, View, Controller), Networking (WebSocket, APIService) şi Utilities.
- Săgeţile de dependență arată modul în care pachetele interacționează (ex. ChatController –¿ WebSocketClient).

3.6 Diagrame de Clase

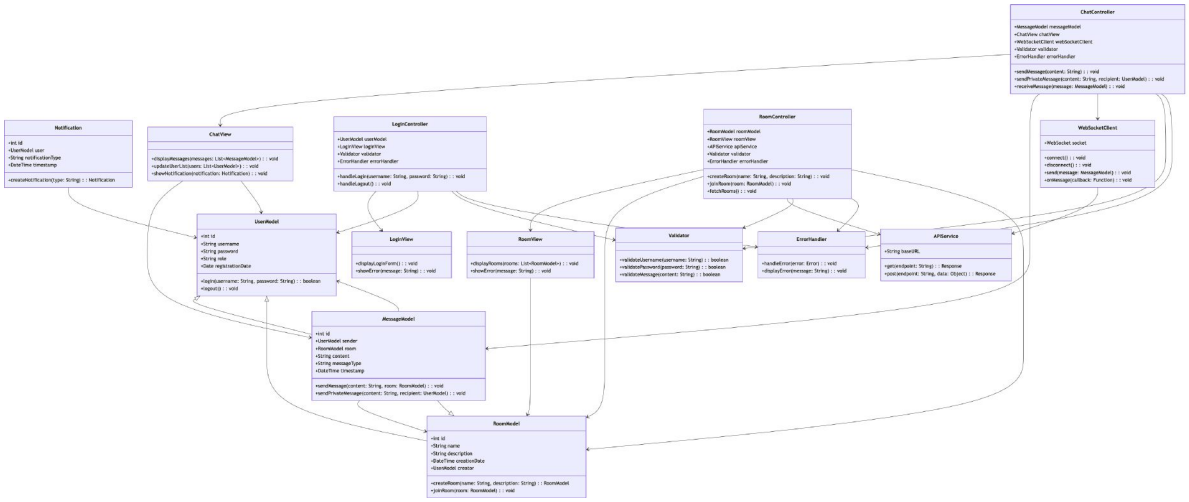

Figura 15: Diagrama de Clase (Server)

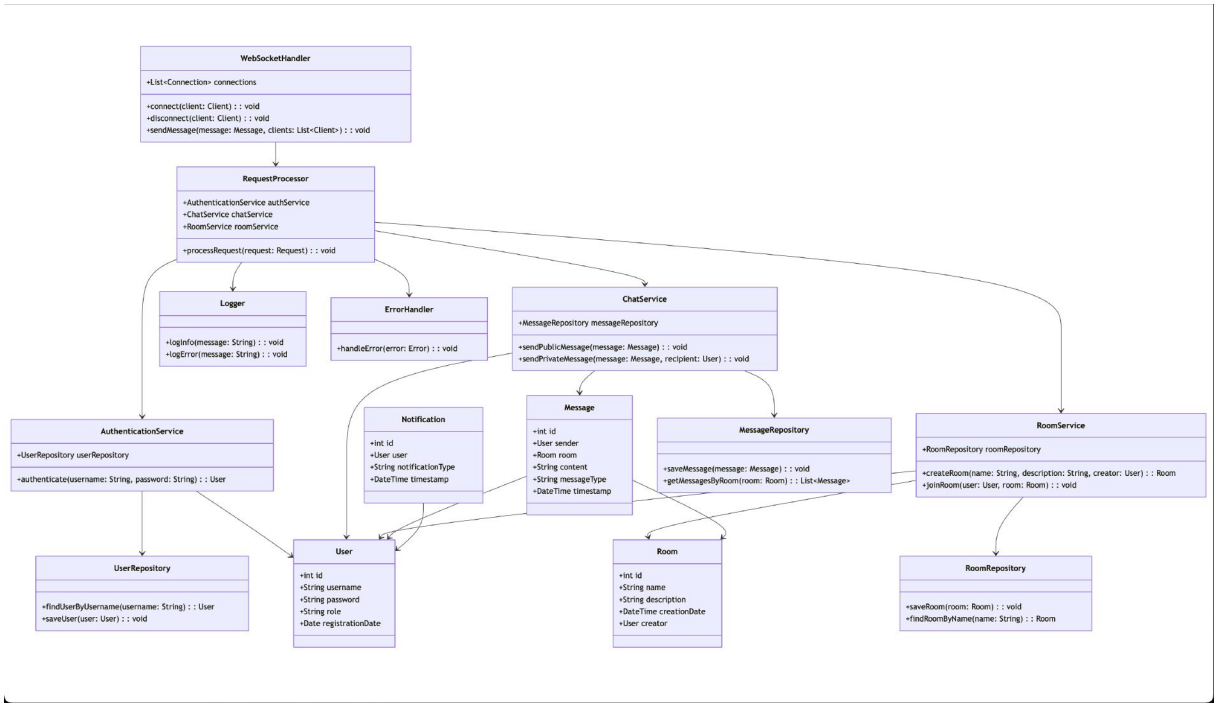

Figura 16: Diagrama de Clase (Client - MVC)

Explicație:

- Diagrama de clase pentru **Server** scoate în evidență entități precum *User, Message, Room*, dar şi clasele de infrastructură (*AuthenticationService, ChatService, RoomService*) şi de gestionare a cererilor (*RequestProcessor*).
- Diagrama de clase pentru **Client** arată arhitectura MVC, unde Modelele (*UserModel, MessageModel, RoomModel*) conțin logica şi datele, View-urile (*LoginView, ChatView, RoomView*) afişează elementele, iar Controllerele (*LoginController, ChatController, RoomController*) leagă totul şi se ocupă de comunicarea cu serverul.

4 Diagrame de Secvență

Diagramele de secvență (*Sequence Diagrams*) oferă o perspectivă dinamică asupra modului în care entitățile (actori, componente software) interacționează în timp. Pentru proiectul nostru de Chat Client-Server, am extras principalele fluxuri de lucru: **autentificare**, **trimitere mesaje publice**, **mesaje private**, **creare/alăturare camere**, **moderare**, **blocare utilizatori** etc.

Diagrama evidențiază ordinea apelurilor (mesajelor) între obiecte, indicând clar "cine apelează pe cine" și momentul la care are loc fiecare apel.

4.1 Secvență: Autentificare

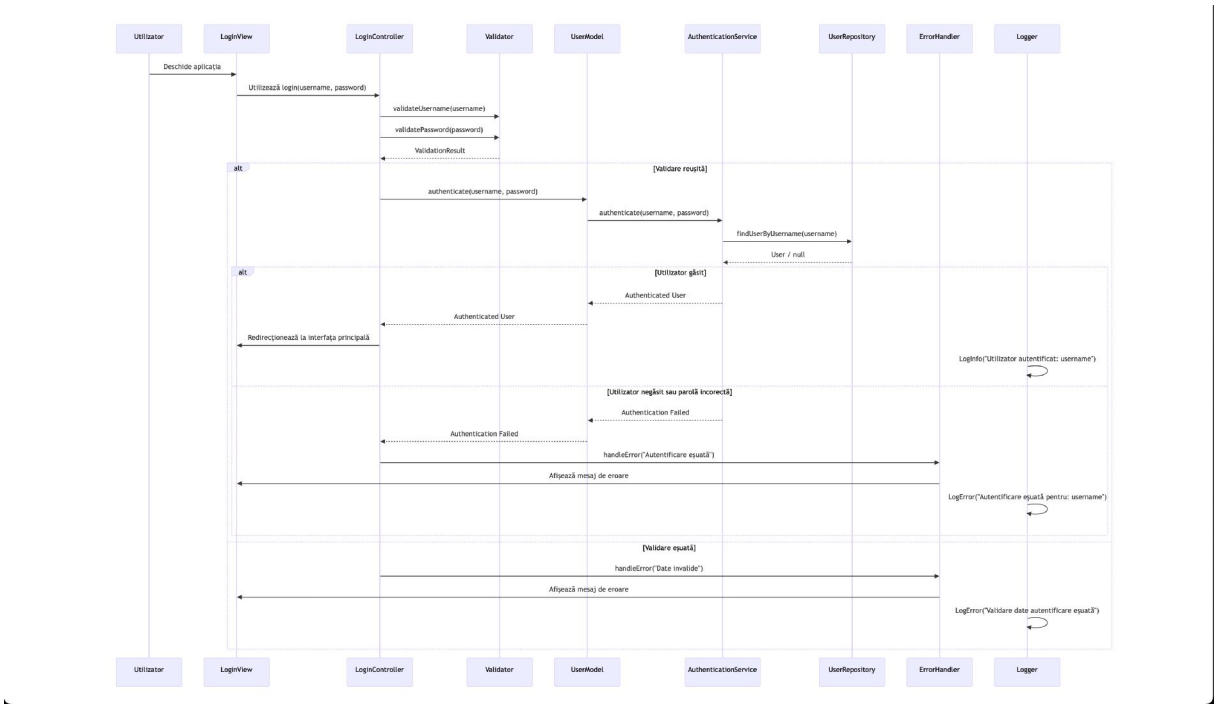

Figura 17: Diagrama de Secvență: Proces de Autentificare

Descriere:

- **Actorul Utilizator** deschide aplicația (clientul web) și apelează **LoginView** pentru a introduce credențiale (username, parolă).
- **LoginView** transferă datele la **LoginController**, care le validează prin **Validator** (username/password).
- Dacă validarea reușește, **LoginController** apelează **UserModel** → **AuthenticationService** → **UserRepository** pentru a găsi userul în baza de date.
- Dacă userul este găsit și parola corespunde (a se vedea `check_password_hash`), atunci **AuthenticationService** îi confirmă identitatea lui **UserModel**, care returnează un rezultat de tip "Autentificare reușită".
- **LoginController** anunță **LoginView** să redirecționeze userul spre interfața principală (chat).
- În caz de eșec (user inexistent, parolă greșită, date invalide), **ErrorHandler** afișează un mesaj de eroare, iar **Logger** înregistrează incidentul.

4.2 Secvență: Trimitere Mesaje Publice

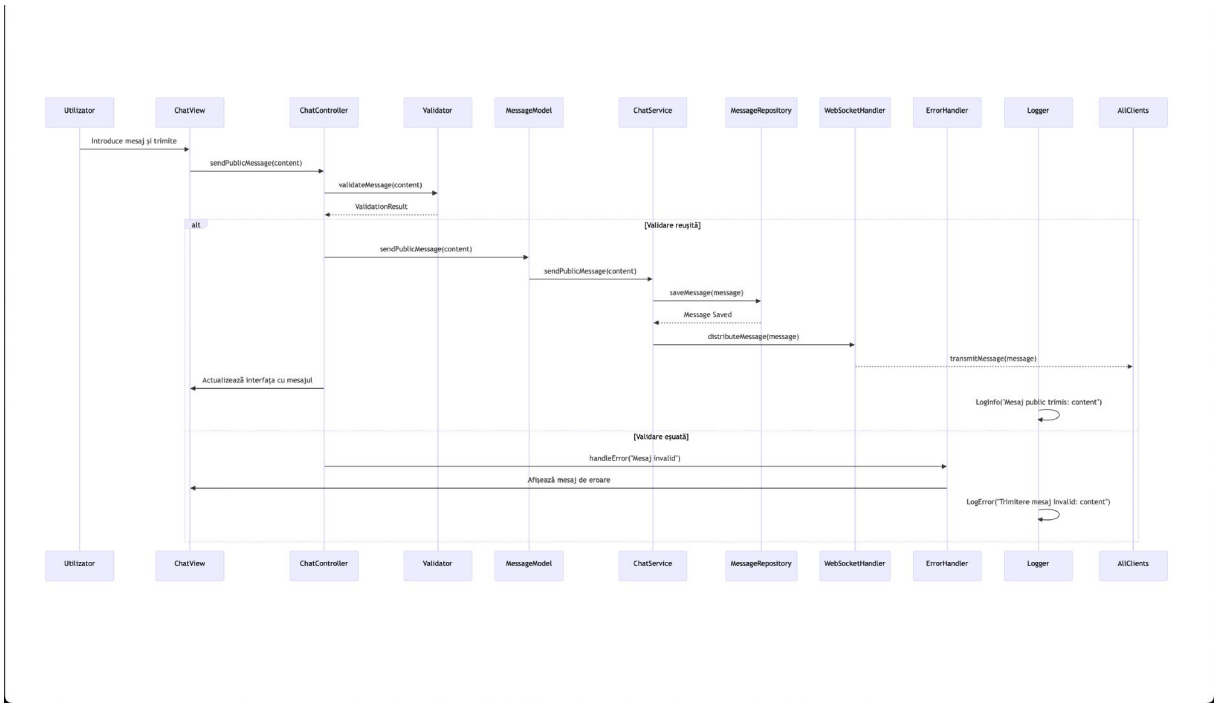

Figura 18: Diagrama de Secvență: Mesaje Publice

Descriere:

- **Actorul (Utilizatorul)** introduce conținutul mesajului în **ChatView**, apoi apasă "Send".
- **ChatView** apelează **ChatController**, care (înainte de a trimite), validează textul prin **Validator** (e.g., interzice mesaje goale).
- Dacă validarea e ok, **ChatController** cheamă metoda `sendPublicMessage` din **MessageModel**, care transmite datele către **ChatService**.
- **ChatService** salvează mesajul via **MessageRepository** și distribuie apoi către toți utilizatorii prin **WebSocketHandler**.
- Toate clienții (*AllClients*) primesc evenimentul "public_message" și își actualizează UI-ul.
- În cazul în care validatorul respinge conținutul (p. ex. "Mesaj invalid"), **ErrorHandler** indică problema.

4.3 Secvență: Trimitere Mesaje Private

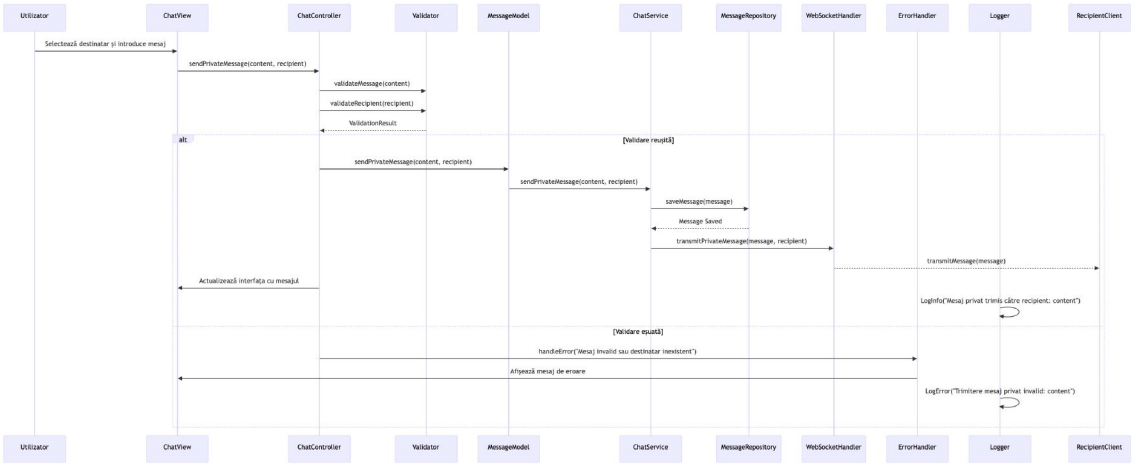

Figura 19: Diagrama de Secvență: Mesaje Private

Descriere:

- Userul alege **destinatarul** în **ChatView** (ex.: meniu drop-down cu userii online) și scrie textul mesajului.
- **ChatController** verifică, prin **Validator**, că *destinatarul* e valid/ online și că textul e corect (nu e gol).
- Dacă validare e reușită, **MessageModel** apelează **ChatService** → **MessageRepository** pentru a salva mesajul cu `tip_mesaj = "private"`.
- **ChatService** folosește **WebSocketHandler** să trimită direct doar *destinatarului.*
- Expeditorul primește un "echo" local, destinatarul primește mesajul în UI-ul său, restul userilor nu-l văd.

4.4 Secvență: Creare Cameră de Chat

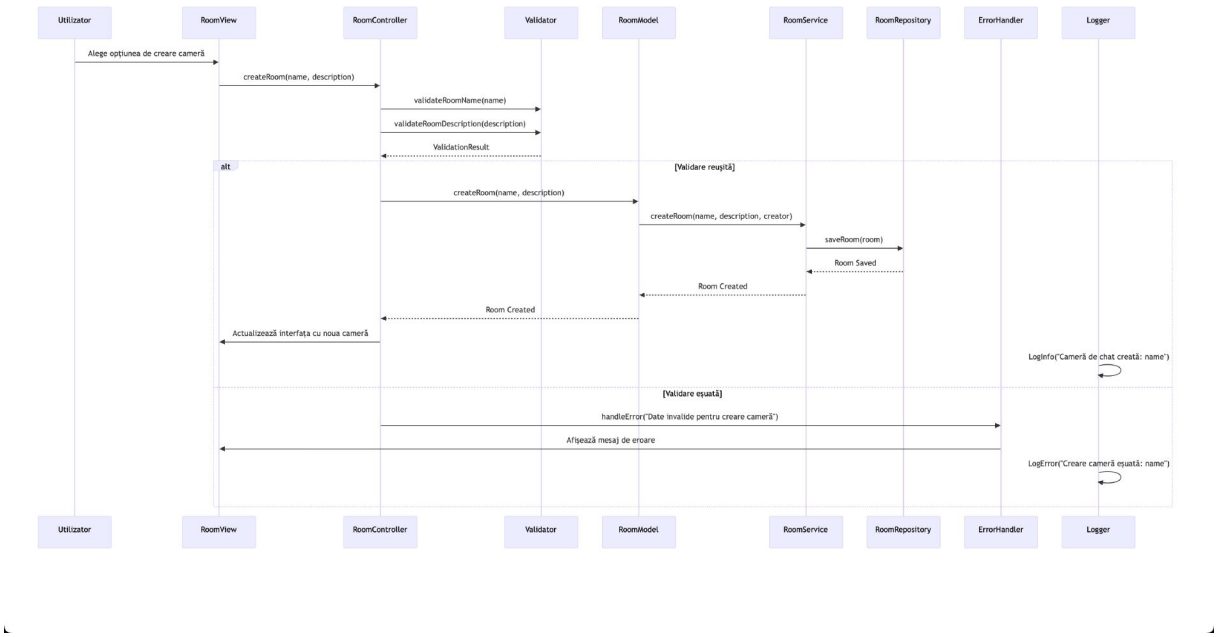

Figura 20: Diagrama de Secvență: Creare Cameră

Descriere:

- În **RoomView**, userul (Utilizator) alege "Create Room", introduce *nume* și *descriere*.
- **RoomController** rulează validări (nume cameră unic?), apoi apelează **RoomModel**, care cheamă **RoomService** → **RoomRepository**.
- Se creează un nou *row* în tabelul `CamereDeChat`, userul devine `creator_id`.
- **RoomView** e actualizată, afișând camera nouă.

4.5 Secvență: Alăturare la Cameră de Chat

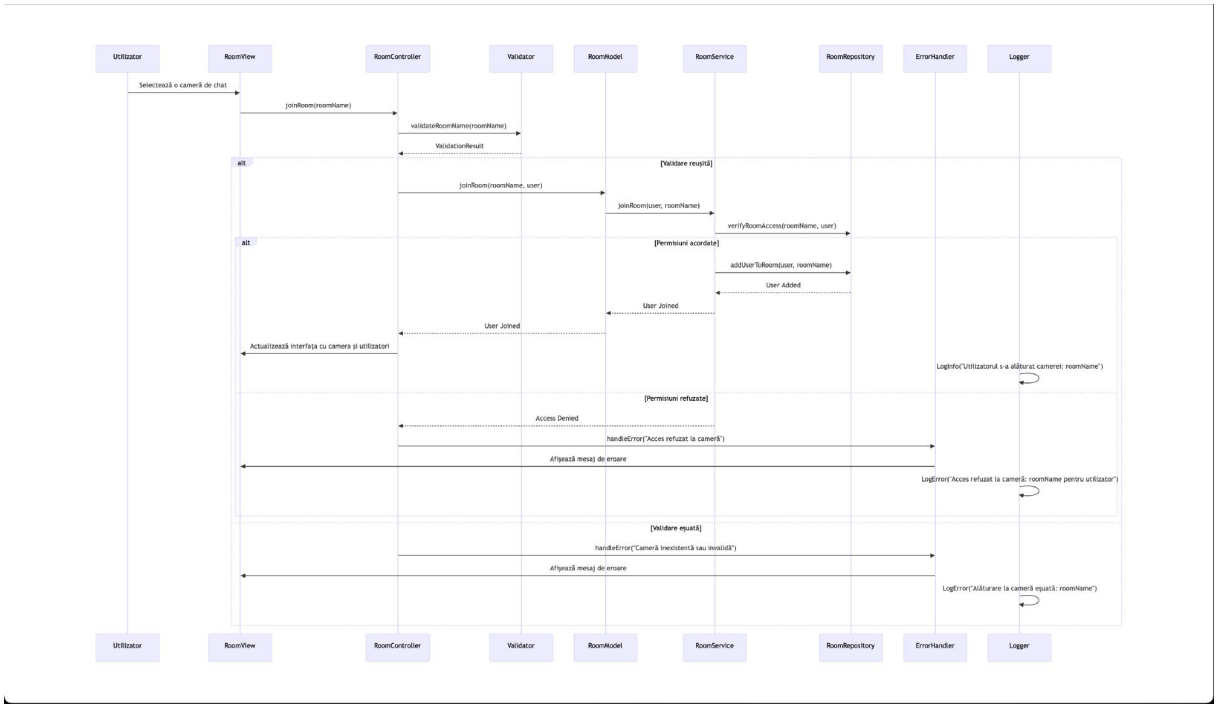

Figura 21: Diagrama de Secvență: Alăturare Cameră

Descriere:

- Utilizatorul selectează o cameră deja disponibilă (e.g., listă camere) din **RoomView** și cere "Join".
- **RoomController** verifică, prin **Validator**, numele camerei. Apoi, **RoomModel** -¿ **RoomService** -¿ **RoomRepository** pentru a vedea dacă userul are acces.
- Dacă e permis, se face un *insert* în `UserChatRoom` (legătura user–cameră).
- **RoomController** îi semnalează **RoomView** să afișeze succesul și să arate conținutul camerei.

4.6 Secvență: Notificări la Conectare/Deconectare

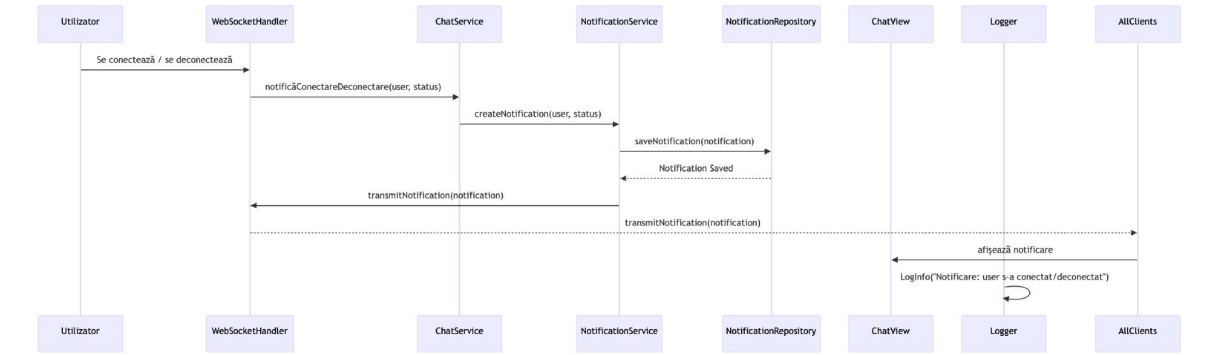

Figura 22: Diagrama de Secvență: Notificări Conectare/Deconectare

Descriere:

- Când un **Utilizator** se conectează sau se deconectează, **WebSocketHandler** anunță **ChatService** (ex.: "user X connected/disconnected").
- **ChatService** poate crea un `Notification` via **NotificationService/Repository** (opțional) și emite evenimente ("user_notification") tuturor clienților.
- Fiecare **ChatView** afișează un banner sau un mesaj scurt ("X s-a conectat!").

4.7 Secvență: Moderarea Discuțiilor (Admin)

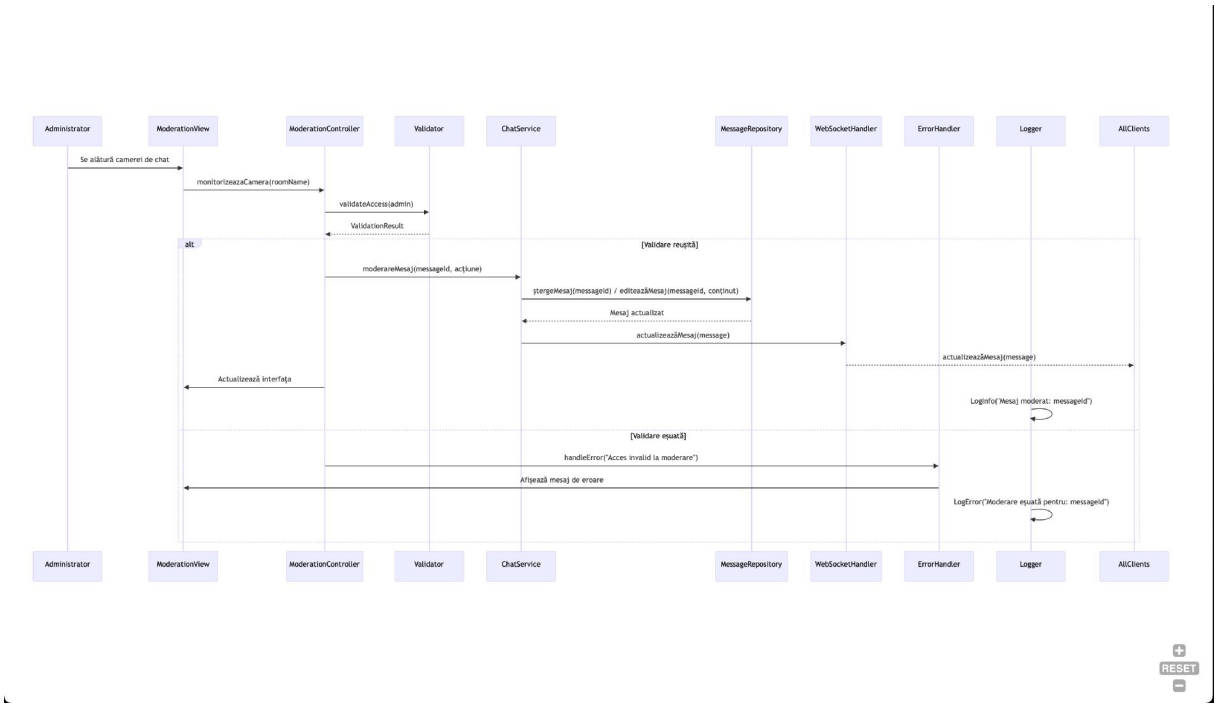

Figura 23: Diagrama de Secvență: Moderarea Discuțiilor

Descriere:

- **Administratorul** se alătură unei camere pentru a monitoriza discuția.
- Dacă vede conținut nepotrivit, **ModerationController** trimite o comandă (`moderareMesaj(messageId, actiune)`).
- **ChatService** actualizează conținutul mesajului în **MessageRepository** (ştergere/editare).
- **WebSocketHandler** notifică toți clienții din cameră cu noua stare a mesajului.

4.8 Secvență: Blocare Utilizatori (Admin)

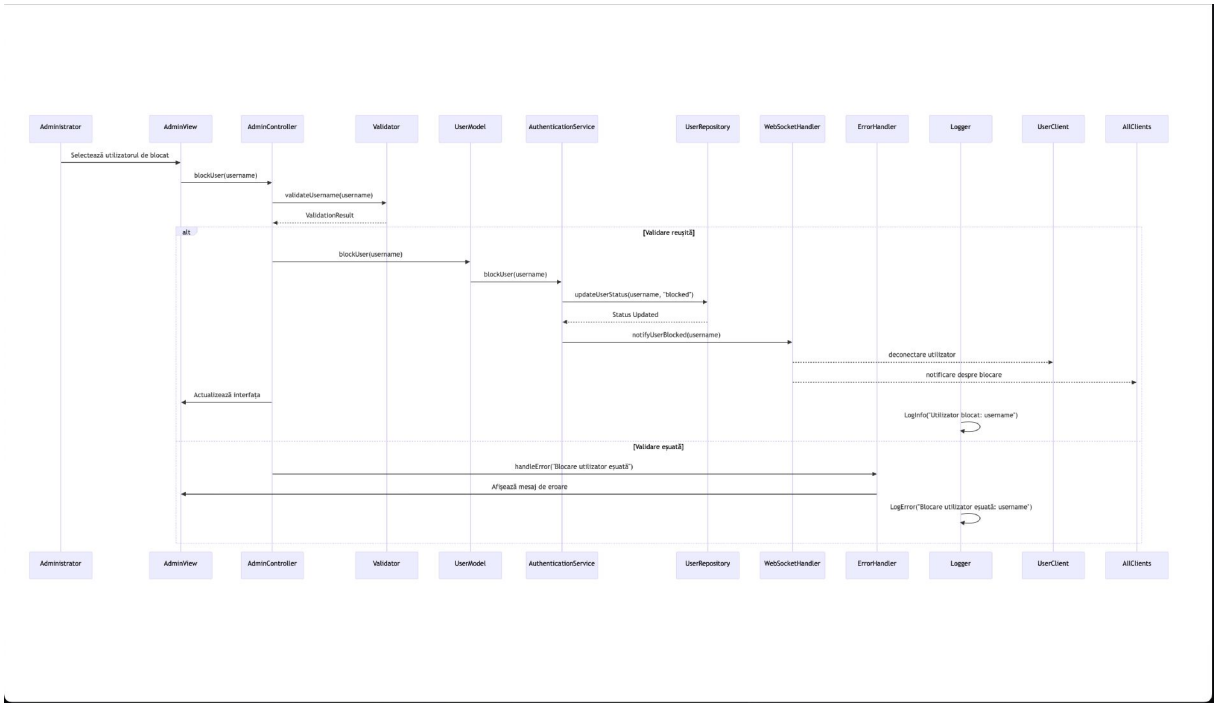

Figura 24: Diagrama de Secvență: Blocare Utilizatori

Descriere:

- **Adminul** identifică userul "problemă", apasă "Block" în **AdminView**.
- **AdminController** apelează **AuthenticationService** sau un **UserService**, care în **UserRepository** setează `status=blocked`.
- **WebSocketHandler** forțează deconectarea userului, iar toți ceilalți useri pot fi notificați că "X a fost blocat".

4.9 Secvență: Creare Camere Predefinite (Admin)

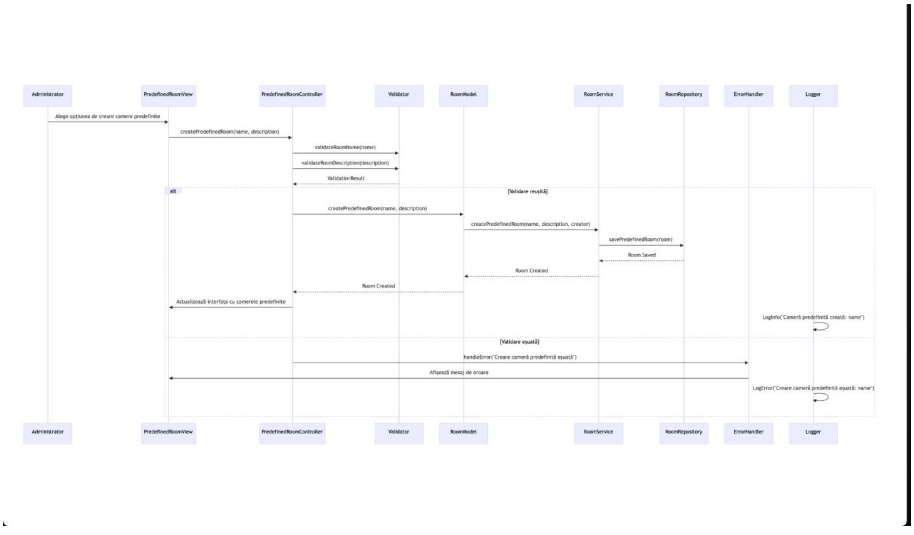

Figura 25: Diagrama de Secvență: Creare Camere Predefinite (Admin)

Descriere:

- **Adminul** intră în secțiunea "Predefined Rooms", introduce parametrii noii camere (ex. "anunțuri globale").
- **PredefinedRoomController** validează datele, apelează **RoomService** → **RoomRepository** pentru a salva camera cu un flag `isPredefined = true`.
- **PredefinedRoomView** actualizează UI-ul, afișând camera în lista globală la toți userii.

Observație: Aceasta extinde functionalitatea normală de *createRoom*, dar forțează un statut special sau anunță toți userii imediat.

4.10 Alte Diagrame de Secvență

Pe lângă cele prezentate, se pot imagina:

- *Forgot Password* (user trimite un request, sistemul validează userul, rescrie parola etc.).
- *Change Role* (admin schimbă rolul altui user, sistemul actualizează DB).
- *Clear Chat* (admin golește istoricul public/room).

Acestea urmează același pattern: **Controller** → **Service** → **Repository** → DB, plus **ErrorHandler** și **Logger** pentru cazurile speciale.

5 Interfața Grafică (Client)

Aplicația client rulează într-un browser. Figurile de mai jos ilustrează ecranul principal, formularul de login etc.

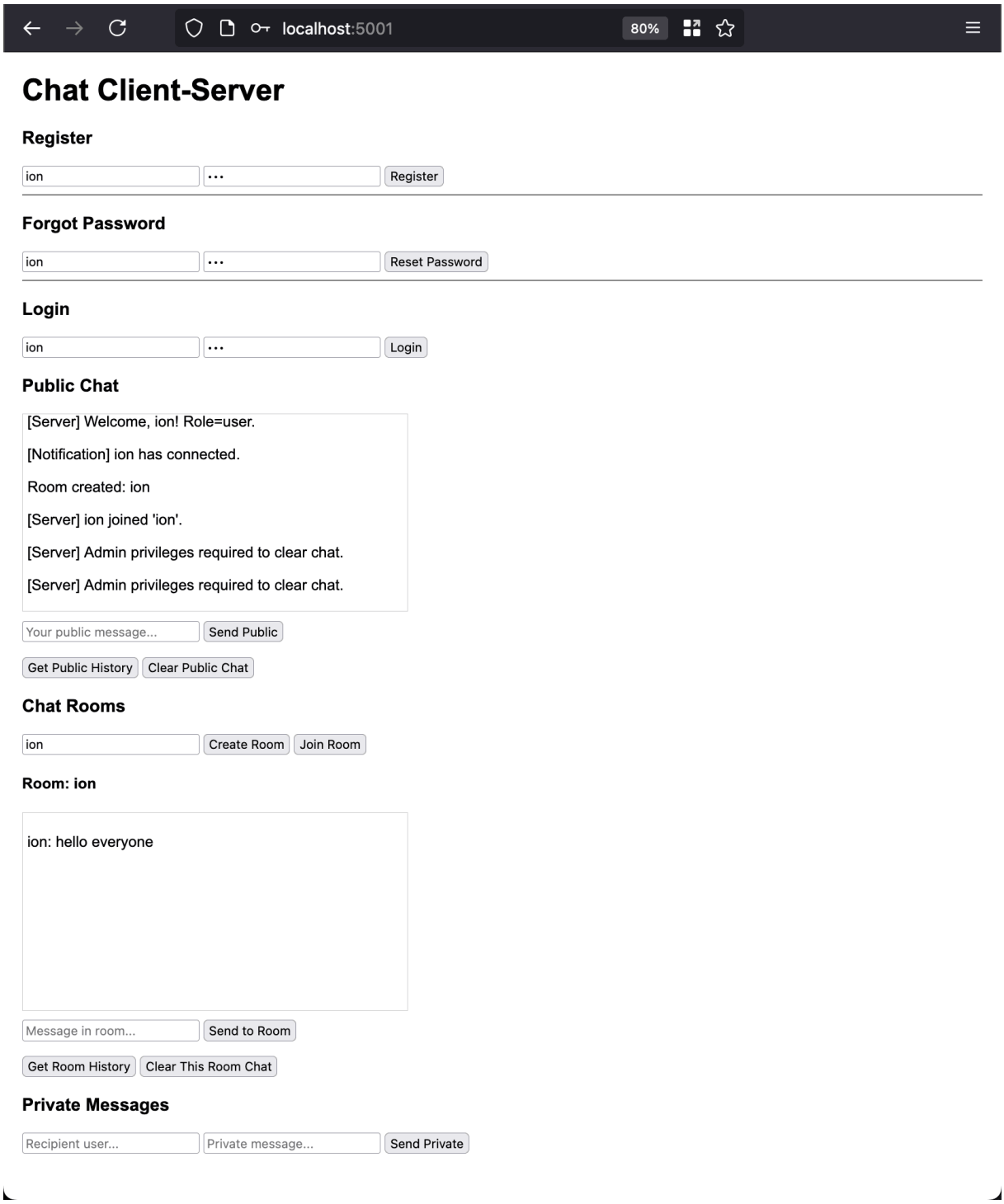

Figura 26: Interfața principală de Chat

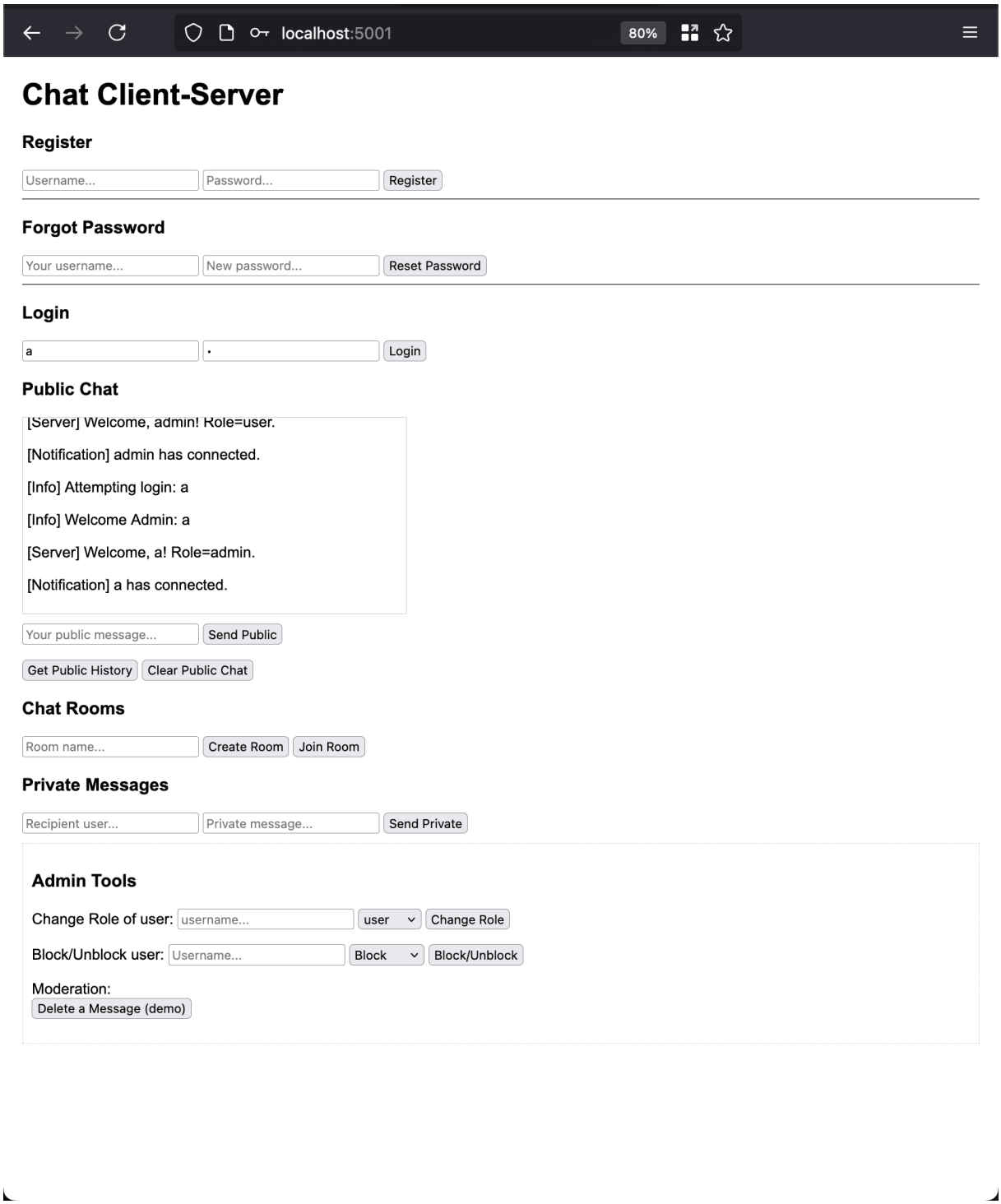

Figura 27: Funcționalități Admin (blocare, schimbare rol)

Dupa autentificare, dacă rolul este **admin**, se afişează secțiunea de “Admin Tools”. Astfel pot fi schimbate roluri, pot fi blocați utilizatori etc.