MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Universitatea Tehnică din Cluj-Napoca

Facultatea de Automatică și Calculatoare

DeChat – Sistem de Mesagerie Descentralizat

Comunicare P2P via WebRTC cu Semnalizare și Persistență

Programul de studii: CALCULATOARE

Autor:

Vancea Paul Alexandru

2026

Contents

1 Introducere

1.1 Context și Motivație

În peisajul tehnologic actual, centralizarea datelor în mari centre de date ridică probleme majore de securitate și suveranitate a informației. DeChat propune o alternativă în care serverul nu mai este un punct central prin care trec toate mesajele, ci doar un facilitator pentru conexiunea directă între utilizatori. Utilizând tehnologia **WebRTC**, aplicația permite transferul de date criptate direct între browserele clienților.

1.2 Obiectivele Sistemului

Proiectul își propune atingerea următoarelor obiective tehnice:

- Implementarea unui protocol de semnalizare robust peste WebSockets.
- Reducerea latenței în comunicare prin utilizarea `RTCDataChannel`.
- Asigurarea persistenței mesajelor globale prin SQLite (sql.js) pentru utilizatorii care se conectează ulterior.
- Oferirea unei experiențe de utilizare moderne (Real-time UI) folosind React și Zustand.

Table 1: Prioritizarea cerințelor proiectului prin metoda MoSCoW

Categorie	**Cerințe / Funcționalități**
Must Have	• Semnalizare P2P și mesagerie directă; • Afișarea listei de peer-i activi; • Handshake securizat utilizând API Key.
Should Have	• Persistența mesajelor în baza de date; • Containerizare utilizând Docker; • Management de stare globală în interfața frontend.
Could Have	• Mecanism de fallback pe REST API (în caz de eșec WebRTC); • Suport pentru transferul de fișiere între clienți.
Won't Have	• Autentificare complexă (SaaS/OAuth/OpenID); • Apeluri video/audio (planificate pentru versiuni viitoare).

2 Arhitectura Generală

2.1 Modelul Hibrid P2P

DeChat nu este un sistem P2P pur, ci unul hibrid. Acesta utilizează un **Signaling Server** pentru descoperirea nodurilor, dar fluxul principal de mesaje directe circulă prin tuneluri WebRTC.

- **Control Plane:** WebSockets (pentru schimbul de metadate și semnale ICE).
- **Data Plane (Direct):** WebRTC (pentru chat privat).
- **Data Plane (Persistat):** REST API (pentru mesaje tip broadcast și istoric).

2.2 Stiva Tehnologică

Componentă	**Tehnologie Utilizată**
Frontend Framework	React 18 + Vite
Limbaj de programare	TypeScript (strict mode)
State Management	Zustand
Backend Runtime	Node.js + Express
Bază de date	SQLite (`sql.js`)
Comunicare	WebSockets (`ws`) + WebRTC
Containerizare	Docker & Kubernetes

Table 2: Stiva tehnologică DeChat

3 Implementare Backend

3.1 Signaling Gateway și Validare

Serverul de semnalizare acționează ca un router de mesaje tehnice. La conectare, fiecare client trebuie să trimită un eveniment de tip `hello` care conține un `apiKey`. Fără acesta, conexiunea este închisă imediat. Evenimente principale gestionate:

- `signal`: Redirecționează pachetele SDP/ICE către destinarul specificat.
- `peers`: Returnează lista nodurilor active cu metadatele lor (nume, ID, status).

3.2 Gestiunea Datelor cu SQLite

Am ales `sql.js` pentru portabilitate. Sistemul include un mecanism de **migrații automate** care rulează la pornirea serverului, asigurând existența tabelelor `messages` și `peers`.

```
export interface StoredMessage {
  id: string;
  senderId: string;
  senderName: string;
  receiverId: string | null; // null inseamna broadcast
  content: string;
  timestamp: number;
}
```

Listing 1: Interfața pentru mesaje stocate

3.3 Securitate și Observabilitate

Backend-ul implementează politici CORS configurabile și un endpoint de `/health` care monitorizează starea bazei de date și a serverului WebSocket, esențial pentru probele de Liveness din Kubernetes.

4 Implementare Frontend

4.1 Hook-ul `usePeerClient`

Acesta este piesa centrală a aplicației. Acesta incapsulează toată complexitatea WebRTC:

1. Creează instanța `RTCPeerConnection`.
2. Adaugă `ICE Candidates` primiți prin WebSocket.
3. Gestionează ciclul de viață al `RTCDataChannel` (onOpen, onClose).
4. **Fallback Logica:** Dacă un canal P2P nu poate fi stabilit în 10 secunde, mesajele sunt trimise automat prin API-ul REST către backend.

4.2 Structura Componentelor UI

Aplicația utilizează CSS Modules pentru a asigura izolarea stilurilor.

- **App.tsx**: Layout-ul principal și inițializarea serviciilor.
- **PeerList.tsx**: Monitorizează în timp real starea (*Connected, Connecting, Failed*) a celorlalți utilizatori.
- **Composer.tsx**: Editor de mesaje cu suport pentru selecția tipului de trimitere.

4.3 Testare și Calitate

Calitatea codului este menținută prin:

- **Vitest:** Pentru teste unitare rapide în mediul Vite.
- **React Testing Library:** Pentru verificarea comportamentului componentelor UI la interacțiunea utilizatorului.
- **ESLint & Prettier:** Pentru consistența stilului de cod și evitarea erorilor comune de TypeScript.

5 Infrastructură și Deploy

5.1 Containerizarea cu Docker

Am utilizat **Multi-stage builds** pentru a optimiza dimensiunea imaginilor:

- Etapa 1: Build-ul aplicației (Node 20).
- Etapa 2: Servirea statică (Nginx pentru frontend) sau runtime-ul minimal pentru backend.

5.2 Orchestrarea în Kubernetes

Pentru deployment-ul în cloud/cluster, am definit manifeste pentru:

- **ConfigMap:** Stocarea URL-urilor de API.
- **Secrets:** Stocarea cheilor API și a credențialelor TURN.
- **Ingress:** Configurarea rutării traficului (HTTPS la nivel de edge).

5.3 Pipeline-ul CI/CD (GitHub Actions)

La fiecare *Pull Request* pe ramura `main`, se declanșează automat: 1. Instalarea dependențelor și cache-uirea acestora. 2. Execuția testelor (Lint, Vitest, Jest). 3. Construirea imaginilor Docker și verificarea vulnerabilităților.

6 Concluzii și Direcții Viitoare

Proiectul DeChat a demonstrat că tehnologiile moderne de browser (WebRTC) pot fi folosite pentru a crea aplicații sigure, scalabile și mai puțin dependente de infrastructura centralizată.

6.1 Limitări Actuale

- Lipsa criptării **End-to-End (E2EE)** la nivel de aplicație (se bazează pe securitatea canalului WebRTC).
- Nu există livrare de mesaje offline (peer-ul trebuie să fie activ).

6.2 Plan de Dezvoltare

Viitoarele iterații se vor concentra pe implementarea unui sistem de **mesaje asincrone** (offline queue) și pe adăugarea criptării cu chei publice/private (RSA/AES) pentru a garanta că nici măcar serverul de semnalizare nu poate citi conținutul pachetelor de date.

7 Bibliografie

1. Resurse MDN WebRTC - `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API`
2. Tanenbaum, A. S. - *Computer Networks*, 5th Edition.
3. Documentation Vite & React - `https://vitejs.dev/`
4. Kubernetes Official Documentation - `https://kubernetes.io/docs/`