Munkahelyi csapatkommunikációs VoIP és chat alkalmazás fejlesztése

1. Bevezetés

A szakdolgozat célja egy modern, webes alapú csapatkommunikációs platform megtervezése és megvalósítása, amely hanghívások, valós idejű szöveges üzenetküldés és projektmenedzsment funkciókat egyaránt kínál. A platform támogatja az egyéni és csoportos kommunikációt, valamint a feladat- és projektkezelést, így átfogó megoldást biztosít a csapatteljesítmény és együttműködés javítására.

2. Célkitűzések

A projekt főbb technikai és funkcionális céljai az alábbiak:

- WebRTC-alapú peer-to-peer hanghívások (hívásindítás, -fogadás, némítás, bontás)
- Valós idejű szöveges chat Socket.IO/WebSocket használatával, privát és csoportos szobák támogatásával
- Projekt- és feladatkezelő modul fejlesztése, amely tartalmazza a határidők, feladatleírások és státuszok kezelését
- Jogosultsági rendszer felhasználói szerepkörökkel (admin, felhasználó)
- Biztonságos hozzáférés JWT-alapú hitelesítéssel, bcrypt jelszókezeléssel
- SQL Server alapú adatbázis Docker konténerben
- Letisztult, karbantartható frontend React + Material UI kombinációjával

3. Követelményelemzés

3.1 Funkcionális követelmények

A platformnak lehetőséget kell biztosítania a felhasználók regisztrációjára, bejelentkezésére és profiladatainak kezelésére. A kommunikáció WebRTC és Socket.IO segítségével valósul meg, emellett a projekt- és feladatkezelés is részét képezi a rendszernek. Az adminisztrációs felület segítségével kezelhetők a felhasználói szerepkörök.

3.2 Nem-funkcionális követelmények

- REST API válaszidő: 200–300 ms
- Valós idejű események válaszideje: < 100 ms
- HTTPS-en keresztüli titkosított kommunikáció
- Frontend támogatott böngészői: Chromium alapú böngészők
- Kódminőség: TypeScript strict mód, ESLint, Prettier

4. Architektúra és rendszerterv

4.1 Rendszerkomponensek és kommunikáció

A rendszer három fő rétegre épül: frontend (React + TypeScript), backend (NestJS) és adatbázis (SQL Server, Docker konténerben). A kommunikáció REST API és WebSocket alapú.

4.2 Backend modulok (NestJS)
A fő modulok a következők:

- **Auth modul** – JWT, bcrypt
- **User modul** – felhasználói adatok, képfeltöltés
- **Communication modul** – híváskezelés WebRTC signalinggal
- **Chat modul** – üzenetküldés, olvasottsági státusz
- **Project modul** – projekt- és feladatkezelés
- **Admin modul** – jogosultságkezelés

4.3 Adatbázisterv
A legfontosabb táblák:

- `Users` (id, username, email, passwordHash, role, createdAt)
- `Profiles` (userId, displayName, avatarUrl, bio)
- Kommunikációhoz: `Conversations`, `Messages`, `Calls`
- Projektkezeléshez: `Projects`, `Tasks`

Az adatbázis struktúrája a fejlesztés alatt még nagy mértékben változhat. A jelenlegi lista egy PoC megvalósítás.

5. Frontend kialakítás

5.1 Technológiai stack és UI

- React és Material UI komponensek felhasználásával készül a felhasználói felület. például: AppBar, Drawer, DataGrid, Dialog.
- Támogatott a világos és sötét mód, valamint az egyedi CSS modulok.

5.2 Állapotkezelés és routing

- Context API segítségével kezeljük az authentikációs és felhasználói állapotokat
- Egyedi hook-okat alkalmazunk a logika elkülönítéséhez
- Útvonalkezelés: React Router v6, dinamikus és védett útvonalakkal

5.3 Felhasználói élmény
A felület reszponzív kialakítású, mobilon és asztali eszközön is jól használható. A sötét/világos téma közti váltás segíti a hozzáférhetőséget.

6. Backend részletek

6.1 NestJS és TypeORM
A backend TypeORM-mel kezeli az entitásokat és migrációkat. DTO-kon keresztül történik a bemeneti adatok validálása. Konfiguráció `.env` fájlokon keresztül történik.

6.2 WebSocket kommunikáció
A Socket.IO gateway-ek kezelik az üzenetküldést, szobák kezelését és a VoIP hívások signaling folyamatait. A kommunikáció namespace-ekbe és szobákba szervezett.

6.3 Biztonság

- JWT guard a védett útvonalakra
- HTTPS minden kommunikációs csatornán
- Bcrypt-tel salted & hashed jelszavak

7. DevOps és üzemeltetés

7.1 CI/CD pipeline
A GitHub Actions automatizálja a buildelést, tesztelést és deploy-t. Docker image-ek kerülnek a registry-be, és minden sikeres pull request után automatikus staging deploy történik. Hibás verzió esetén rollbackra is van lehetőség.

8. Tesztelési stratégia

- **Jest unit tesztek** a kulcsmodulokra (auth, chat, task)
- **Frontend komponens tesztek** a React Testing Library-vel
- **Integrációs tesztek** végpontok és adatbázis-műveletek ellenőrzésére
- **Hibakezelési szcenáriók** szimulálása valós környezetben

9. Fejlesztési ütemterv

- **1–2. hét**: Környezet előkészítés, alapmodulok
- **3–4. hét**: Auth, User, Profile modulok + unit tesztek
- **5–6. hét**: Chat modul, valós idejű kommunikáció
- **7–8. hét**: WebRTC VoIP hívások
- **9–10. hét**: Project, Task, Admin modulok
- **11–12. hét**: Frontend fejlesztés (MUI, CSS)
- **13. hét**: E2E tesztek, CI/CD finomhangolás, dokumentáció

10. Kockázatelemzés és kezelés
A főbb kockázatok közé tartozik a technikai nehézségek, kódminőség kérdései és fejlesztői túlterheltség. Ezek kezelésére priorizálás, work itemek használata, rendszeres kódellenőrzés és reális ütemezés kerül alkalmazásra.

11. Összefoglalás

A választott technológiai stack – React a frontendhez, NestJS a backendhez, valamint Docker az üzemeltetéshez – lehetővé teszi a gyors fejlesztést és a skálázható, stabil működést. A szakdolgozat dokumentálása folyamatosan történik: egyrészt a fejlesztés alatt, a döntések és megoldások rögzítésével, másrészt annak lezárulta után, a végső tapasztalatok és műszaki részletek összegzésével. Ez a kettős dokumentációs megközelítés biztosítja a rendszer hosszú távú fenntarthatóságát és továbbfejleszthetőségét.

Ilyó-Kovács Levente
NLFUA8