

VPWA Dokumentácia – 2. (finálna) fáza

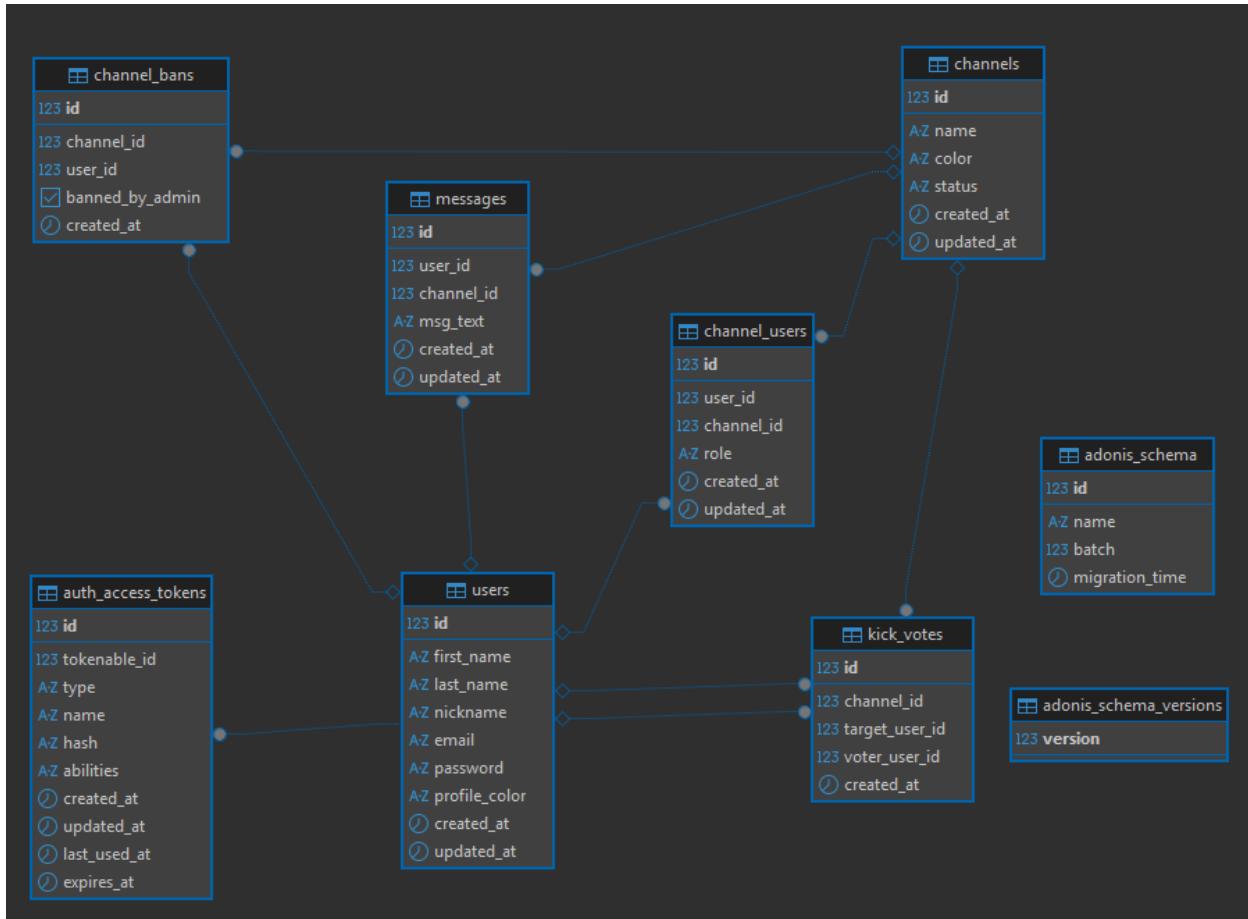
Zadanie

1. registrácia, prihlásenie a odhlásenie používateľa
 - používateľ má meno a priezvisko, nickName a email
2. používateľ vidí zoznam kanálov, v ktorých je členom
 - pri opustení kanála, alebo trvalom vyhodení z kanála je daný kanál odobratý zo zoznamu
 - pri pozvánke do kanála je daný kanál zvýraznený a topovaný
 - v zozname môže cez používateľské rozhranie kanál vytvoriť, opustiť, a ak je správcom aj zrušiť
 - dva typy kanálov - súkromný (private channel) a verejný kanál (public channel)
 - správcom kanála je používateľ, ktorý kanál vytvoril
 - ak nie je kanál aktívny (nie je pridaná nová správa) viac ako 30 dní, kanál prestáva existovať (následne je možné použiť channelName kanála pre "nový" kanál)
3. používateľ odosielá správy a príkazy cez "príkazový riadok", ktorý je "fixným" prvkom aplikácie. používateľ môže odoslať správu v kanáli, ktorého je členom
4. vytvorenie komunikačného kanála (channel) cez príkazový riadok
 - kanál môže vytvoriť ľubovoľný používateľ cez príkaz /join channelId [private]
 - do súkromného kanála môže pridávať/odoberať používateľov iba správca kanála cez príkazy /invite nickName a /revoke nickName
 - do verejného kanála sa môže pridať ľubovoľný používateľ cez príkaz /join channelId (ak kanál neexistuje, automaticky sa vytvorí)
 - do verejného kanála môže člen kanála pozvať iného používateľa príkazom /invite nickName

- vo verejnem kanáli môže člen "vyhodiť" iného člena príkazom /kick nickName. ak tak spravia aspoň 3 členovia, používateľ má "trvalý" ban pre daný kanál. správca môže používateľa vyhodiť "natrvalo" kedykoľvek príkazom /kick nickName, alebo naopak "obnoviť" používateľovi prístup do kanála cez príkaz /invite
 - nickName ako aj channelName sú unikátne
 - správca môže kanál zatvoriť/zrušiť príkazom /quit
5. používateľ môže zrušiť svoje členstvo v kanáli príkazom /cancel, ak tak spraví správca kanála, kanál zaniká
 6. správu v kanáli je možné adresovať konkrétnemu používateľovi cez príkaz @nickname
 - správa je zvýraznená danému používateľovi v zozname správ
 7. používateľ si môže pozrieť kompletnú história správ
 - efektívny infinite scroll
 8. používateľ je informovaný o každej novej správe prostredníctvom notifikácie
 - notifikácia sa vystavuje iba ak aplikácia nie je v stave "visible" (pozrite quasar docu App Visibility)
 - notifikácia obsahuje časť zo správy a odosielateľa
 - používateľ si môže nastaviť, aby mu chodili notifikácie iba pre správy, ktoré sú mu adresované
 9. používateľ si môže nastaviť stav (online, DND, offline)
 - stav sa zobrazuje používateľom
 - ak je nastavený DND stav, neprichádzajú notifikácie
 - ak je nastavený offline stav, neprichádzajú používateľovi správy, po prepnutí do online sú kanály automaticky aktualizované
 10. používateľ si môže pozrieť zoznam členov kanála (ak je tiež členom kanála) príkazom /list
 11. ak má používateľ aktívny niektorý z kanálov (nachádza sa v okne správ pre daný kanál) vidí v stavovej lište informáciu o tom, kto aktuálne píše správu (napr. Ed is typing)

- po kliknutí na nickName si môže pozrieť rozpísaný text v reálnom čase, predtým, ako ju odosielateľ odošle (každá zmena je viditeľná)

Diagram fyzického dátového modelu



- **messages**

- zmenené názvy polí (vykonané kvôli jednoznačnejším označovaním jednotlivých údajov, používanie auditných polí)
 - senderId => user_id
 - channelId => channel_id
 - text => msg_text
 - timestamp => created_at (+ updated_at)
- odstránené pole (odstránené kvôli nepotrebnosti)
 - recipientId

- **channel_members => channel_users** (jednoznačnejšie pomenovanie)

- **pridané** pole „role“ (kvôli efektívному zachovaniu roly používateľa v danom kanály – jeden používateľ môže byť správcom v jednom kanály, a obyčajným používateľom v druhom)
- **pridané** auditné pole

- **channels**

- ownerId **posunuté** do channel_users.role (vlastenectvo kanála pokrýva práve stĺpec role, ownerId bolo tam zbytočne)
- type **premenované** na status (jasnejší názov – status je teda “private” a “public”)
- **pridané** auditné pole

- **users**

- color **premenované** na profile_color (kvôli jednoznačnosti pola – farbu priradujeme aj pre kanál, tým sme to semanticky rozdelili)

- **kick_votes**

- **pridaná** tabuľka
- obsahuje aktívne hlasovania na “vyhodenie” používateľa z kanála
- hlasovania sa rozlišuje podľa identifikátora kanála, a cieľového používateľa
- ak bude počet hlasov pre používateľ v danom kanáli nad 3:
 - používateľ je vyhodený a banovaný z kanála
 - všetky hlasovania patriace k tomuto hlasovaniu sa odstránia

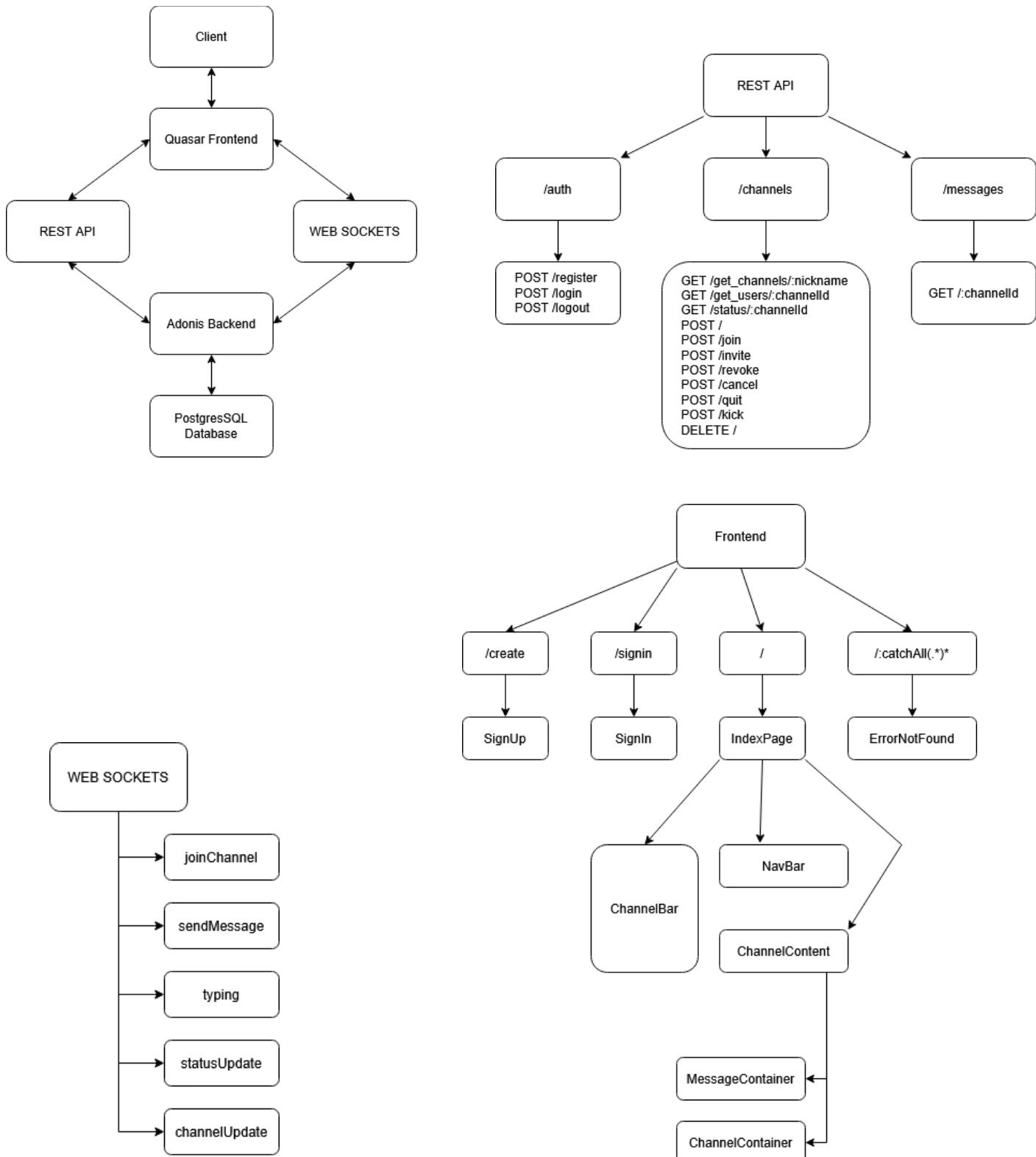
- **channel_bans**

- **pridaná** tabuľka
- obsahuje tých používateľov daných kanálov, ktorí boli v dôsledku (používateľského) hlasovania o vyhodenie odstránené a banované z kanála
- po opäťovnej pozvánke od správcovi kanála sa daný ban z tejto tabuľky odstráni (správca obnoví prístup do kanála)

- **auth_access_tokens**

- **pridaná** tabuľka
- zachová tokeny na tokenovú autentifikáciu používateľov
(hashované)
- obsahuje aj auditné polia
- použité na overenie identity používateľa pri komunikácii
s backendom

Diagramy architektúry aplikácie



Návrhové rozhodnutia (pridanie externej knižnice - zdôvodnenie, ...)

Rozdelenie komunikácie medzi WebSocket + REST API

REST API volania sme sa rozhodli používať pri deterministických operáciach, ktoré patria iba danému príjemcovi, nie ako verejná informácia pre viacerých používateľov.

Napríklad: inicializačné načítanie dát, CRUD operácie, vrátenie statusov a popisu stavu po spracovaní požiadavky na strane servera, a pod.

WebSocket sa používa na komunikáciu, prenos správ a spracovanie viacerých udalostí v reálnom čase. Je užitočné pre synchronizáciu medzi viacerými klientami.

Napríklad: typing indikátor, odosielanie správ, spracovanie udalostí a notifikácií

Používa sa jeden **globálny WebSocket** pre každého používateľa, tým sa zabezpečí možnosť odosielania **broadcastových správ**.

Multicast komunikácia je zabezpečená cez logické zoskupovanie WebSocket spojení do miestností na strane servera (udalosti sa emitujú iba do príslušnej skupiny používateľov).

Napríklad: Po tom, ako používateľ vstúpi do kanála, alebo je pozvaný, socket sa pridá do multicast skupiny. Pri neskoršom prenose správ do danej skupiny dostanú správu všetci členovia danej miestnosti. – channel:<channelId>

Spracovanie udalostí cez WebSocket

Navrhli sme politiku na spracovanie udalostí, ktorý spočíva na komunikáciu cez WebSocket.

Správy obsahujú:

- typ udalosti (**type**)
- payload (**data**)
 - kód udalosti (**code**)
 - iné informácie správy

Tým sme dosiahli efektívne spracovanie vykonaných príkazov na strane servera, a štrukturovanú komunikáciu na spracovanie odpovedí od servera na frontende.

Napríklad:

Frontend – vytvorí WebSocket správu podľa stanovenej štruktúry

```
{  
  "type": "sendMessage",  
  "data": {  
    "channelId": 1,  
    "nickname": "JozkoMrkvicka",  
    "msgText": "Hello World"  
  }  
}
```

Frontend – odošle správu cez globálny WebSocket spojenie na server
 Backend – prijme udalosť (“event”) a podľa **type** parametru spracuje danú správu (type == sendMessage => socket.handleSendMessage())
 Backend – spracuje správu, komunikuje s databázou (nájde používateľa, pridá novú správu podľa získaných údajov, priradí správu používateľovi) a vytvorí WebSocket odpoved'

```
{
  "type": "message",
  "data": {
    "id": 10,           // id spravy z DB, ktorý patrí používateľovi
    "channelId": 1,     // id kanala, do ktorého patrí sprava
    "nickname": "JozkoMrkvicka", // používateľské meno odosielatela
    "profileColor": "#EF4444", // hex farba profile používateľa
    "msgText": "Hello World", // samotný text spravy
    "timestamp": createdAt // čas vytvorenia spravy
  }
}
```

Backend – odošle túto správu ako ‘event’ do multicast miestnosti kanála (channel:1)
 Frontend – používateľ získa ‘event’ správu, spracuje ju (handleEvent), a pridá telo správy do lokálneho úložiska (pole správ)
 Frontend – zachytí zmenu pola správ, reaktívne aktualizuje stav zobrazených údajov

Ďalší spôsob prenášania a spracovania udalostí je cez kódovanie udalostí, kde do pola “data” je pridaný parameter “code”. Tieto kódované správy sa používajú na signalizovanie jednotlivých udalostí, na základe čoho frontend zobrazuje notifikačné hlášky používateľovi.

Napríklad:

Frontend – používateľ sa prihlási do kanála, odošle POST správu na /channels/join cestu backendu

Backend – spracuje členskú správu pripojenia do kanála, získa údaje a vykonáva zmeny v databáze, a odošle verejnú multicastovú správu členom kanála:

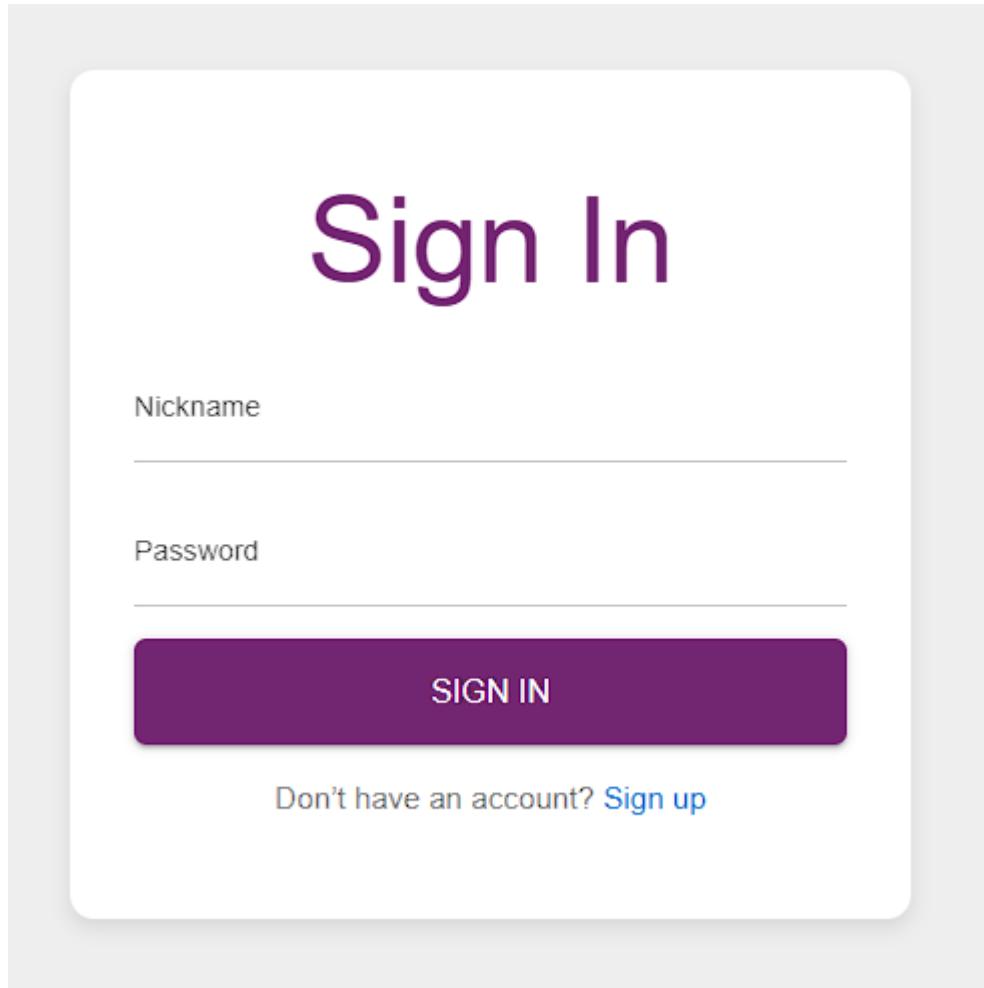
```
{
  "type": "channelUpdate",
  "data": {
    "code": 'JOINED_CHANNEL',
    "channelId": 2,
    "nickname": "JozkoMrkvicka"
  }
}
```

Frontend – spracuje udalosť typu channelUpdate, kde na základe kódu JOINED_CHANNEL aktualizuje lokálny zoznam kanálov

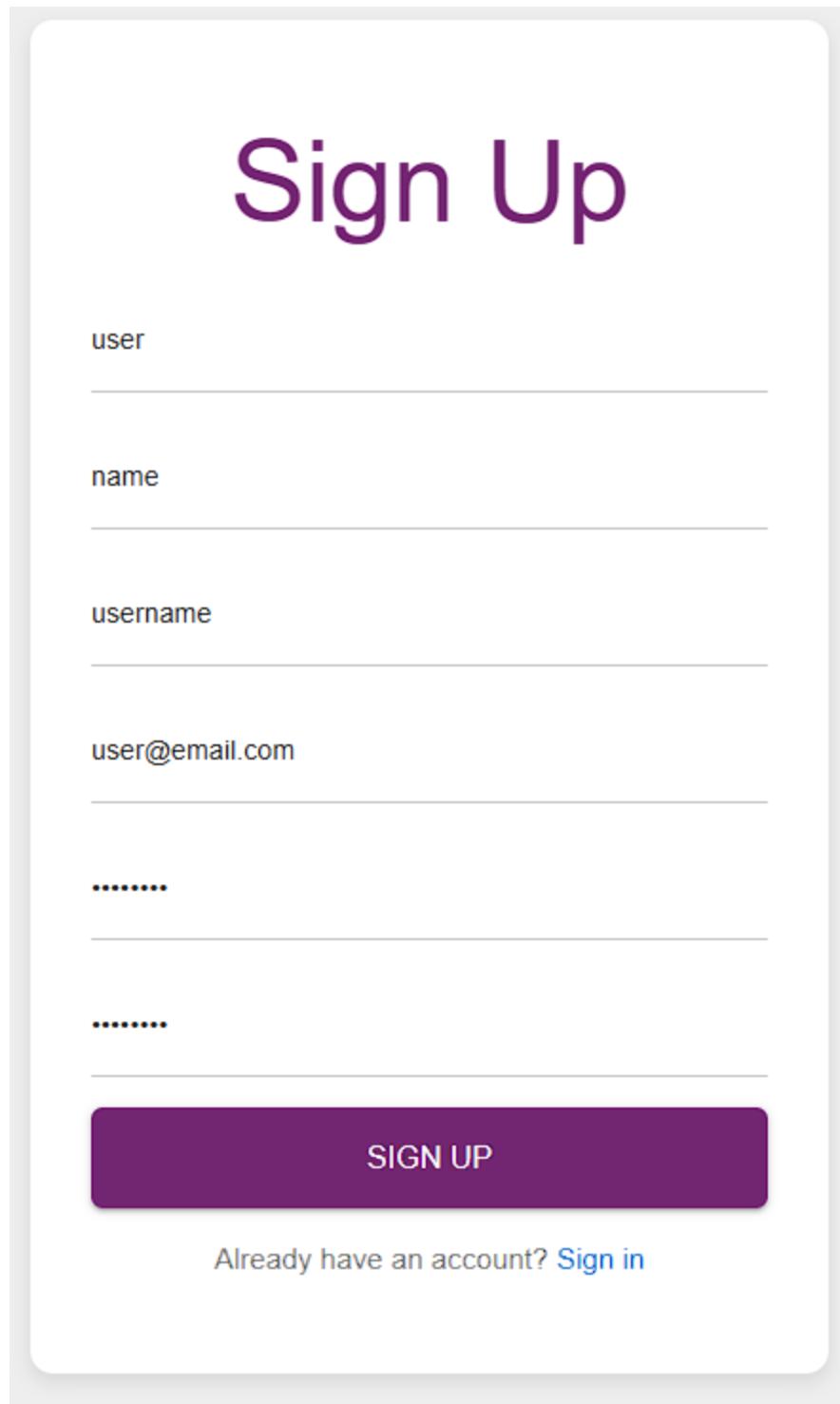
Frontend – aktualizuje stav premennej CHANNEL_EVENT podľa kódu a údajov získanej správy

Frontend – na základe kódu vytvorí notifikačnú hlášku, ktorá sa neskôr zobrazí na hlavnej obrazovke.

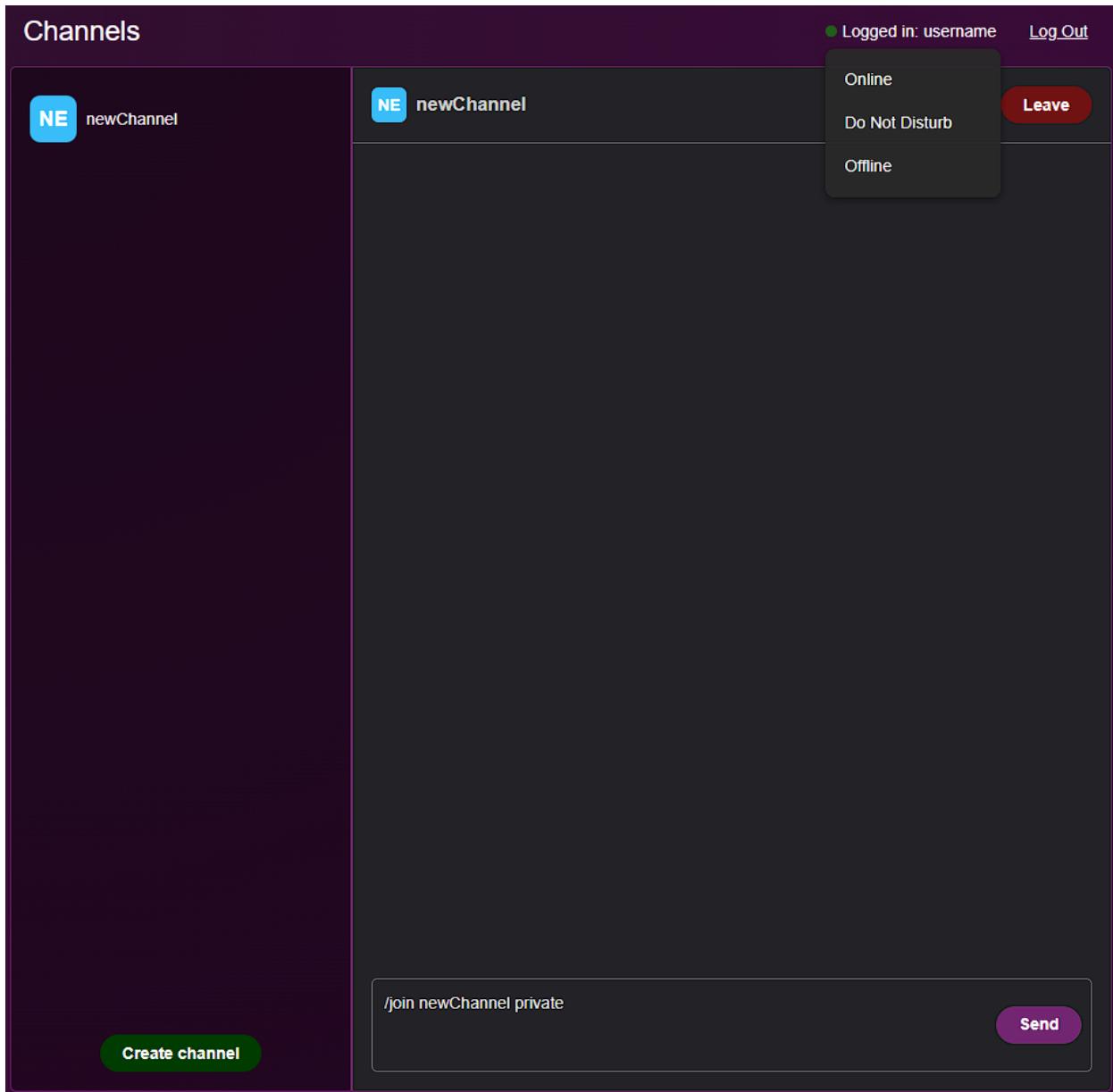
Snímky obrazoviek



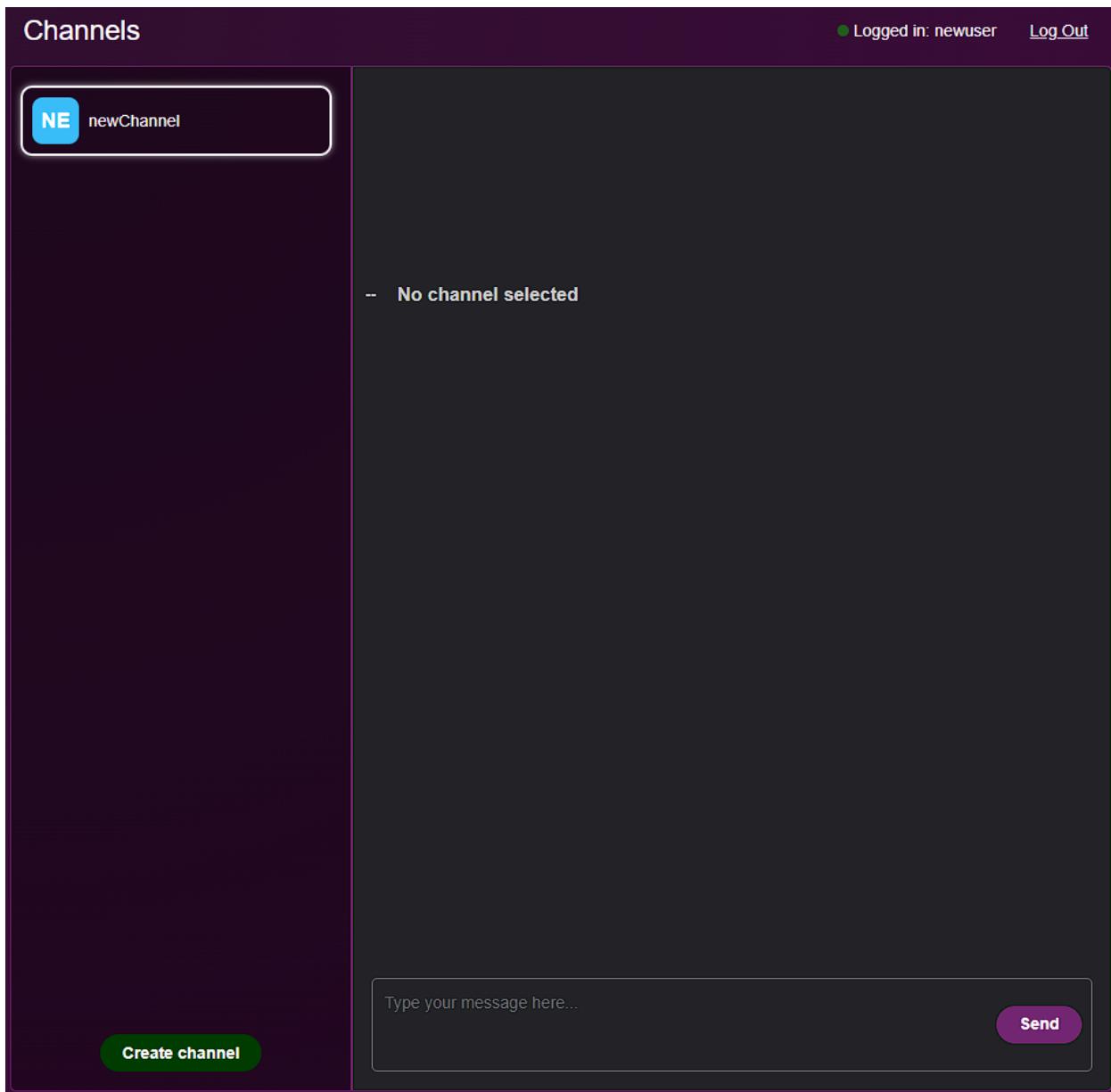
Obrázok 1: Prihlásovacia stránka



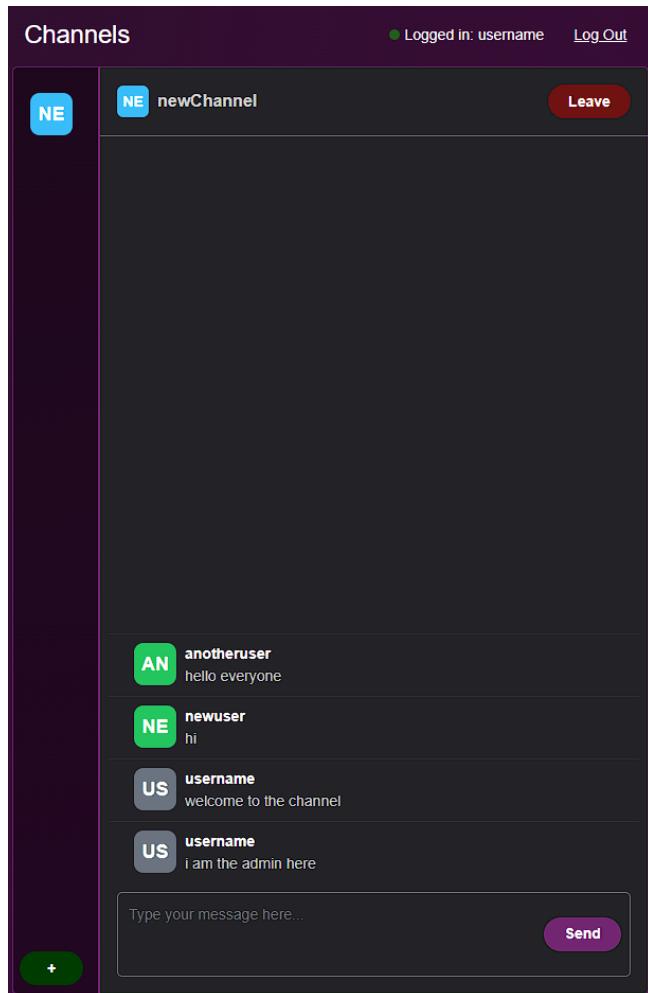
Obrázok 2: Registračná stránka



Obrázok 3: Pripojenie do nového kanálu



Obrázok 4: Používateľ dostal pozvánku do kanála



Obrázok 5: Responzívny vzhľad aplikácie