SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

**Slagg**

VPWA semestrálne zadanie

**Viktória Bukovská, Martin Szabo**

2024

Obsah

Obsah 2

1 Zadanie 5

1.1 Aplikácia na textovú komunikáciu v štýle IRC (zjednodušený Slack) 5

2 Dátové modely 8

2.1 Logicky dátový model 8

2.2 Fyzicky dátový model 8

3 Architektúra 9

3.1 Frontend 9

3.1.1 Komponenty 9

3.1.2 Layouts 9

3.1.3 Pages 9

3.1.4 Služby 10

3.1.5 Stores 10

3.1.6 Boot 10

3.2 Backend 11

3.2.1 Controllers 11

3.2.2 Models 13

3.2.3 Validatory 14

3.2.4 Start 14

3.3 Diagram architektúry 14

4 Návrhové rozhodnutia 15

4.1 Pinia (namiesto Vuex) 15

4.2 Notify plugin 15

5 Ukážky 16

5.1 Chat 16

5.1.1 Na PC 16

5.1.2 Na smartphone 17

5.2 Vytvorenie nového kanála 18

5.3 Pripojenie sa do verejného kanála 18

5.4 Hlasovanie o vyhodení z kanála 19

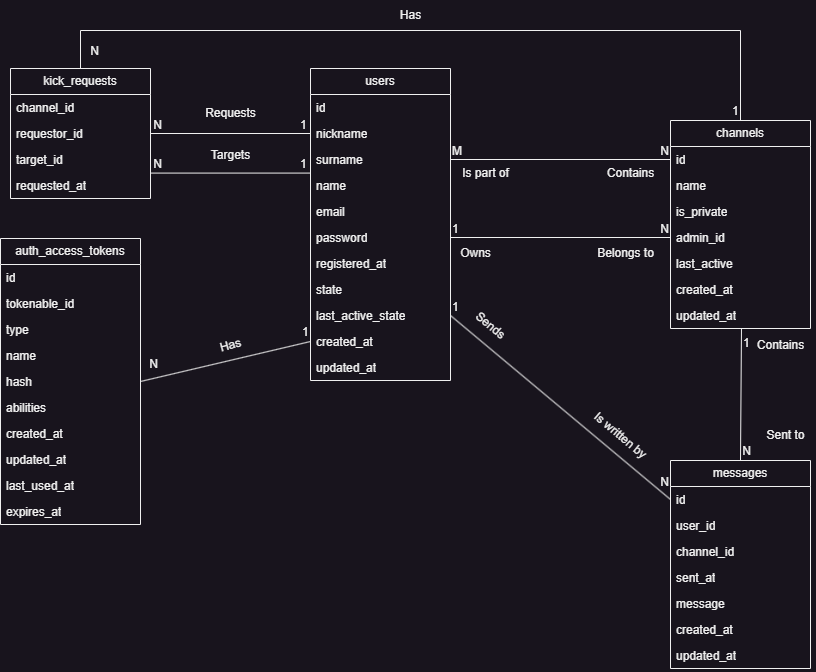
1. Zadanie

Vytvorte progresívnu webovú aplikáciu na textovú komunikáciu v štýle IRC (Slack), ktorá komplexne rieši nižšie definované prípady použitia.

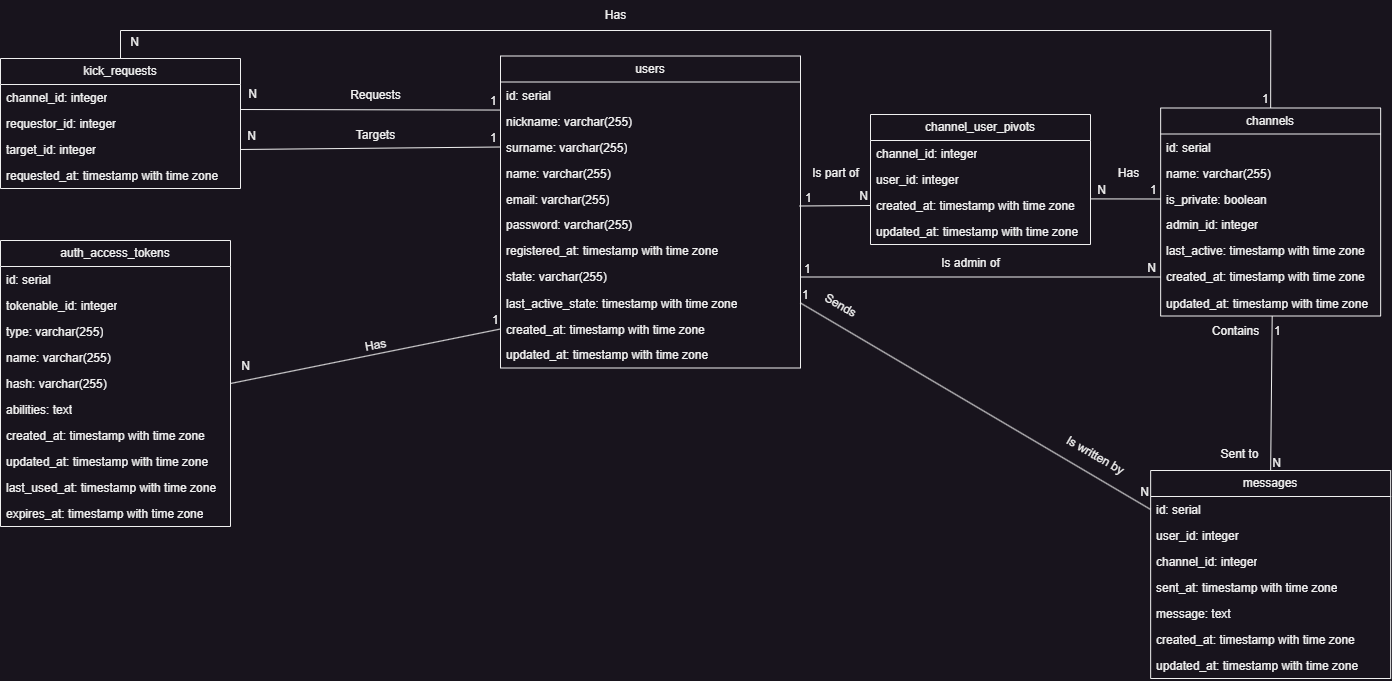
* 1. Aplikácia na textovú komunikáciu v štýle IRC (zjednodušený Slack)

**Aplikácia musí realizovať tieto prípady použitia:** Akékoľvek iné vylepšenia sú vítané a potešia ma :-)

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 odosiela 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 ľubovolný používateľ cez príkaz /join channelName [private]
   * do súkromného kanála môže pridávať/odoberať používateľov iba správca kanála cez príkazy /invite nickName a /revoke nickName
   * do verejného kanála sa môže pridať ľubovolný používateľ cez príkaz /join channelName (ak kanál neexistuje, automaticky sa vytvorí)
   * do verejného kanála môže člen kanála pozvať iného používateľa príkazom /invite nickName
   * vo verejnom 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 "obnovit" 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 históriu správ
   * efektívny inifinite 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) :-)
12. Dátové modely
    1. Logicky dátový model



* 1. Fyzicky dátový model



1. Architektúra
   1. Frontend

Frontend je vytvorený pomocou Vue.js, univerzálneho frameworku JavaScript, a Quasar, frameworku určeného na vytváranie responzívnych a výkonných webových aplikácií.

* + 1. Komponenty

Jednotlivé stránky, ktoré predstavujú rôzne časti aplikácie:

* Registračný komponent (RegistrationComponent.vue): Spracúva registráciu používateľa vrátane overenia a odoslania formulára.
* Prihlasovací komponent (LoginComponent.vue): Spravuje prihlásenie používateľa s funkciami overovania a odosielania.
  + 1. Layouts

Rozloženia definujú celkovú štruktúru aplikácie. Medzi kľúčové rozloženia patria:

* Hlavný layout (MainLayout.vue): Obsahuje záhlavie, pätu a navigačné menu hlavnej aplikácie.
* Layout na prihlásenie (SignInLayout.vue): Špecifické pre stránky súvisiace s overovaním, ako je prihlásenie a registrácia.
  + 1. Pages
* ChatPage.vue: Slúži ako stránka chatu aplikácie. Je zodpovedná za zobrazenie rozhrania chatu vrátane chat logu, vstupného poľa a tlačidla odoslať.
* IndexPage.vue: Je zodpovedná za zobrazenie uvítacej správy a poskytovanie odkazov na ďalšie stránky aplikácie.
* ErrorNotFound.vue: Je zodpovedná za zobrazenie chybovej správy, keď používateľ prejde na stránku, ktorá neexistuje.
* RegisterPage.vue: slúži ako registračná stránka aplikácie. Je zodpovedná za zobrazenie registračného formulára a spracovanie vstupných údajov používateľa.
* LoginPage.vue: Je zodpovedná za zobrazenie prihlasovacieho formulára a spracovanie vstupných údajov používateľa.
  + 1. Služby
* socket.ts exportuje inštanciu služby socketService, ktorá poskytuje komunikačný kanál v reálnom čase medzi klientom a serverom. Táto služba je zodpovedná za vytvorenie a správu spojenia socketu, ako aj za odosielanie a prijímanie správ. Inštancia socketService poskytuje nasledujúce funkcie:
  + vytvorí spojenie socketu so serverom
  + odosiela správy na server
  + prijíma správy zo servera
  + spracúva chyby spojenia a odpojenia
    1. Stores

Moduly storov pre aplikáciu. Moduly storov sú zodpovedné za správu stavu aplikácie a poskytujú centralizované miesto pre prístup k tomuto stavu a jeho úpravu.

* user.ts: Tento modul storu je zodpovedný za správu stavu používateľa vrátane overovania a autorizácie. Poskytuje metódy na prihlásenie, odhlásenie, načítanie informácií o používateľovi, inicializáciu socketov a iné. Používa knižnicu na store management Pinia.
* model.ts: Obsahuje dátové modely používané modulmi storov. Používajú sa v celej aplikácii na reprezentáciu a manipuláciu s údajmi. Napríklad rozhranie User (Používateľ) sa môže použiť na reprezentáciu údajov používateľa, zatiaľ čo rozhranie Channel (Kanál) sa môže použiť na reprezentáciu údajov kanála a podobne.
  + 1. Boot

Obsahuje niekoľko súborov, ktoré sa používajú na spustenie aplikácie.

* axios.ts: Tento súbor sa používa na konfiguráciu knižnice Axios, ktorá je populárnou klientskou knižnicou HTTP pre JavaScript.
* auth.ts: Tento súbor sa používa na konfiguráciu overovania pre aplikáciu.
* socket.ts: Tento súbor sa používa na konfiguráciu knižnice Socket.IO, čo je knižnica na komunikáciu v reálnom čase medzi klientom a serverom.
  1. Backend
     1. Controllers
* AuthController.ts: Autentifikačný controller, ktorý spracováva HTTP požiadavky na prihlásenie a registráciu. Trieda AuthController má nasledujúce metódy:
  + zaregistrovať: spracuje požiadavku POST /register. Vytvorí nového používateľa, uloží ho do databázy a v odpovedi vráti token.
  + Prihlásenie: Spracúva požiadavku POST /login. Overí používateľa, vygeneruje token a vráti ho v odpovedi.
  + odhlásenie: Spracúva požiadavku DELETE /logout. Odhlási autentifikovaného používateľa a v odpovedi vráti správu o úspechu.
  + me: Spracúva požiadavku GET /me. Získava informácie o overenom používateľovi, formátuje ich a vracia ich v odpovedi.
* UserController.ts: Spracúva požiadavky HTTP a interakcie medzi klientmi a serverom súvisiace so správou používateľov. Trieda UserController má nasledujúce metódy:
  + getAllOtherUsers: Spracúva požiadavku GET /users. Získa všetkých používateľov okrem overeného používateľa, sformátuje ich a vráti ich v odpovedi.
  + getJoinablePublicChannels: Spracúva požiadavku GET /joinable-channels. Získa verejné kanály, ku ktorým sa môže autentifikovaný používateľ pripojiť, naformátuje ich a vráti ich v odpovedi.
  + changeUserStatus: Spracúva požiadavku POST /change-status. Aktualizuje stav overeného používateľa a v odpovedi vráti nový stav.
  + createNewChannel: Spracúva požiadavku POST /create-channel. Vytvorí nový kanál, pridá overeného používateľa ako správcu a v odpovedi vráti nový kanál.
  + deleteChannel: Spracúva požiadavku DELETE /delete-channel. Vymaže kanál a v odpovedi vráti správu o úspechu.
  + leaveChannel: Spracúva požiadavku DELETE /leave-channel. Odstráni overeného používateľa z kanála a v odpovedi vráti správu o úspechu.
  + kickUserFromChannel: spracúva požiadavku DELETE /kick-user-from-channel. Odstráni používateľa z kanála a v odpovedi vráti správu o úspechu.
  + addUserToChannel: Spracúva požiadavku POST /add-user-to-channel. Pridá používateľa do kanála a v odpovedi vráti správu o úspechu.
  + joinPublicChannel: Spracúva požiadavku POST /join-public-channel. Pridá overeného používateľa do verejného kanála a v odpovedi vráti správu o úspechu.
  + requestKickUserFromChannel: Spracúva požiadavku POST /request-kick-user-from-channel. Vytvorí požiadavku na kopnutie pre používateľa v kanáli a v odpovedi vráti správu o úspechu.
  + getMessages: Spracúva požiadavku GET /messages. Získava správy pre kanál, formátuje ich a vracia ich v odpovedi.
  + saveMessage: Spracúva požiadavku POST /messages. Vytvorí novú správu, uloží ju do databázy a vráti novú správu v odpovedi.
  + isUserInChannel: Spracúva požiadavku GET /is-user-in-channel. Kontroluje, či je overený používateľ v kanáli, a v odpovedi vracia logickú hodnotu.
  + cleanupInactiveChannels: Spracúva požiadavku GET /cleanup-inactive-channels. Odstráni neaktívne kanály a v odpovedi vráti správu o úspechu.
* SocketController.ts: Spracúva udalosti socketu a interakcie medzi klientmi a serverom. Trieda SocketController má nasledujúce metódy:
  + hello: Spracováva udalosti pripojenia a odpojenia. Keď sa klient pripojí, načíta informácie o kanáli a vyšle klientovi udalosť kanála. Keď sa klient odpojí, vyšle udalosť kanála s nulovou hodnotou.
  + addMessage: Spracúva udalosť addMessage. Vytvorí novú správu, uloží ju do databázy a vyšle udalosť newMessage všetkým klientom v tom istom kanáli.
  + getMessages: Spracúva udalosť getMessages. Získava správy z databázy, formátuje ich a vysiela klientovi udalosť loadedMessages.
  + deletedChannel: Spracúva udalosť deletedChannel. Kontroluje, či kanál existuje, a vysiela udalosť deletedChannel všetkým klientom v tom istom kanáli.
  + memberLeftChannel: Spracúva udalosť memberLeftChannel. Kontroluje, či je člen v kanáli, a vysiela udalosť memberLeftChannel všetkým klientom v tom istom kanáli.
  + addedMember: Spracúva udalosť addedMember. Získava informácie o členovi, formátuje ich a vysiela udalosť addedMember všetkým klientom v tom istom kanáli.
  + reloadUser: Spracúva udalosť reloadUser. Získava informácie o používateľovi, formátuje ich a vysiela udalosť reloadUser všetkým klientom v tom istom kanáli.
  + typing: Spracúva udalosť písania. Vysiela udalosť písania všetkým klientom v tom istom kanáli.
  + stopTyping: Spracúva udalosť stopTyping. Vysiela udalosť stopTyping všetkým klientom v tom istom kanáli.
    1. Models

ORM modely. Reprezentujú entity v aplikácii / DB.

* User.ts: reprezentuje model user tabuľky z fyzického modelu.
* Message: reprezentuje model message tabuľky z fyzického modelu.
* Channel: reprezentuje model channel tabuľky z fyzického modelu.
* KickRequest.ts: reprezentuje model kick\_request tabuľky z fyzického modelu.
  + 1. Validátory
* LoginValidator: Overuje prihlasovacie údaje (e-mail a heslo) zadané používateľom.
* RegisterValidator: Overuje registračné údaje (e-mail, heslo, nick, priezvisko a meno) zadané používateľom.

Každá trieda validátora má metódu schémy, ktorá definuje pravidlá validácie pre príslušné údaje. Metóda schémy vracia objekt schémy, ktorý definuje pravidlá validácie.

Validačné pravidlá sa definujú pomocou objektu rules, ktorý poskytuje súbor preddefinovaných validačných pravidiel. Napríklad pravidlo email kontroluje, či je vstup platnou e-mailovou adresou, zatiaľ čo pravidlo minLength kontroluje, či má vstup minimálnu dĺžku 3 znaky.

Každá trieda validátora definuje aj sadu chybových správ, ktoré sa vrátia, ak validácia zlyhá.

* + 1. Start

Obdobne ako pri boot obsahuje niekoľko súborov, ktoré sa používajú na spustenie aplikácie.

* kernel.ts: Je vstupným bodom pre aplikáciu. Nastavuje rámec AdonisJS a definuje middleware.
* routes.ts: Definuje routy pre aplikáciu. Na definovanie route pre rôzne funkcie, ako je napríklad autentifikácia a správa používateľov.
* socket.ts: Konfiguruje namespace Socket.IO na komunikáciu v reálnom čase medzi klientmi a serverom.
  1. Diagram architektúry

1. Návrhové rozhodnutia
   1. Pinia (namiesto Vuex)

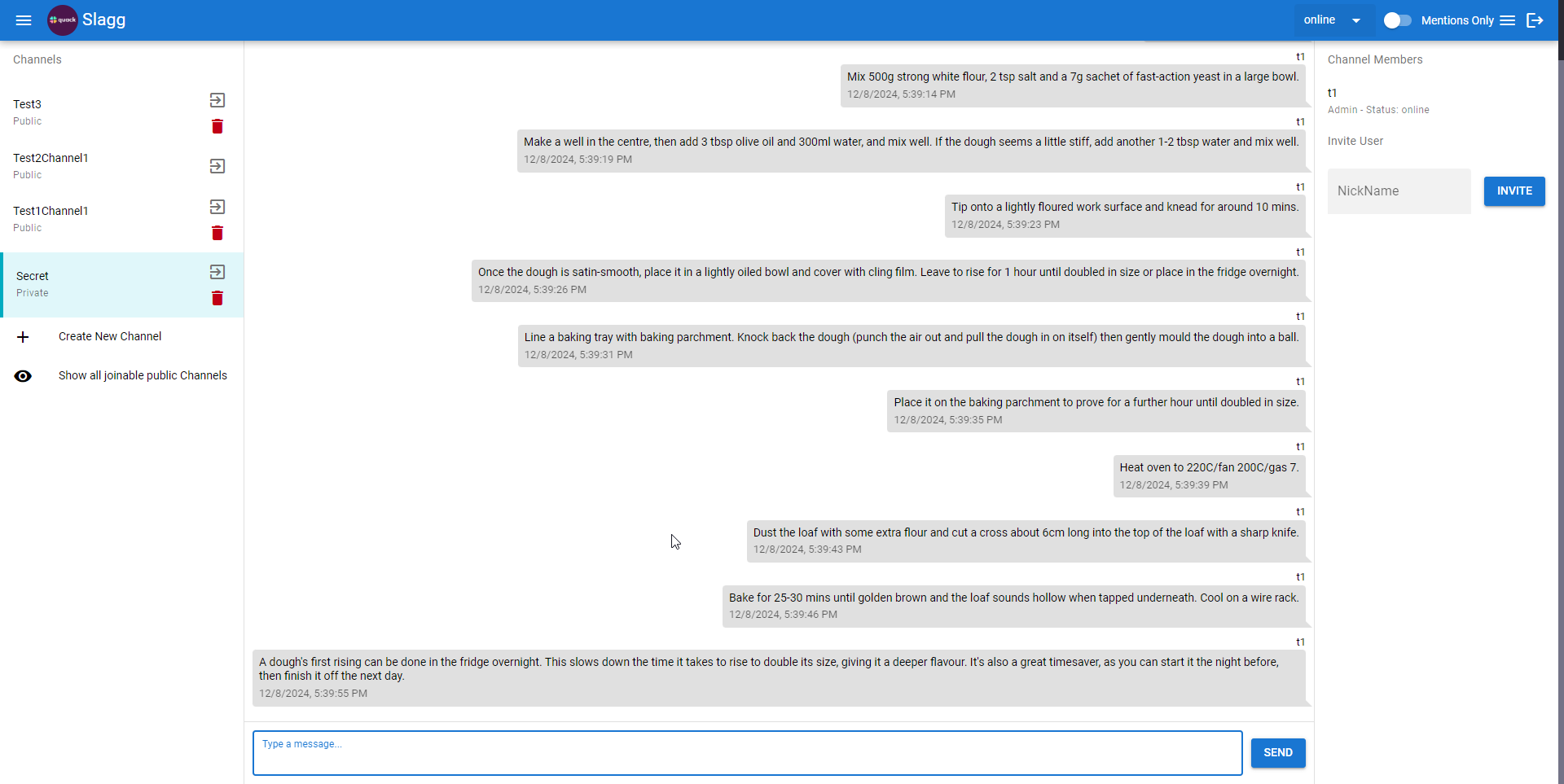
V našom projekte sme prešli z Vuex na Piniu, predovšetkým kvôli jednoduchosti používania, najmä pri práci so socketmi. Pinia nám ponúkla viaceré výhody, ktoré z nej robia lepšiu voľbu:

* + 1. Prečo sme si vybrali Piniu:
* Jednoduchosť: Pinia má intuitívne a ľahko pochopiteľné API, čo zjednodušuje prácu s dátovým manažmentom.
* Podpora TypeScriptu: Vstavaná podpora TypeScriptu poskytuje vyššiu úroveň typovej bezpečnosti, čo minimalizuje chyby.
* Výkon: Pinia využíva efektívny systém reaktivity, ktorý je zároveň jednoduchší na použitie v porovnaní s Vuexom.
* Škálovateľnosť: Modulárny dizajn umožňuje vytvárať viaceré story, ktoré môžu byť jednoducho importované priamo tam, kde sú potrebné.
* DevTools: Pinia disponuje vlastnými vývojárskymi nástrojmi, ktoré umožňujú zobrazovať obsah store v reálnom čase počas používania aplikácie.
  + 1. Porovnanie s Vuexom:
* Komplexnosť: Vuex má oveľa komplikovanejšie API, čo môže byť náročné na pochopenie.
* Podpora TypeScriptu: Vuex nemá natívnu podporu TypeScriptu, čo zvyšuje riziko chýb pri práci s dátami.
* Reaktivita: Systém reaktivity vo Vuexe nie je taký flexibilný ani výkonný ako v Pinia.
* Škálovateľnosť a Modularita: Vuex ponúka len jeden hlavný store s viacerými modulmi, čo môže byť menej flexibilné.
  1. Notify plugin

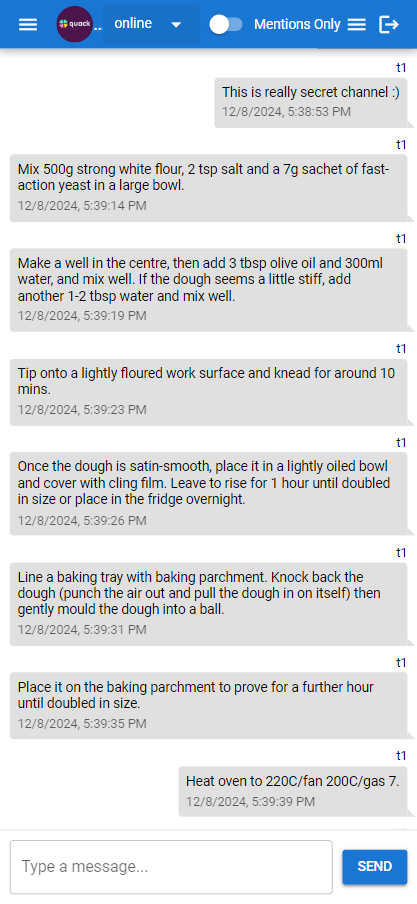
Hlavne dôvody prečo sme sa rozhodli použiť Quasar notify plugin miesto alternatív ako Toasted alebo Vue-Toast:

* Jednoduchosť používania: Zásuvný modul Notify je jednoduchý a ľahko použiteľný modul, ktorý poskytuje jednoduchý spôsob zobrazovania oznámení.
* Integrácia so systémom Quasar: Notify plugin je navrhnutý tak, aby bezproblémovo spolupracoval s Quasarom. Táto integrácia poskytuje ucelené a konzistentné používateľské prostredie.
* Nenáročný: Notify plugin je lightweight plugin, čo znamená, že nepridáva aplikácii výraznú réžiu, takže je vhodnou voľbou pre rôzne prostredia vrátane produkčného.
* Podpora komunity: Notify plugin má aktívnu komunitu a je dobre udržiavaný, čo zaručuje, že bude naďalej dostávať aktualizácie a opravy chýb.
* Súbor funkcií: Notify plugin poskytuje funkcie, ktoré spĺňajú potreby aplikácie, vrátane podpory viacerých typov oznámení, pozícií a možností prispôsobenia.

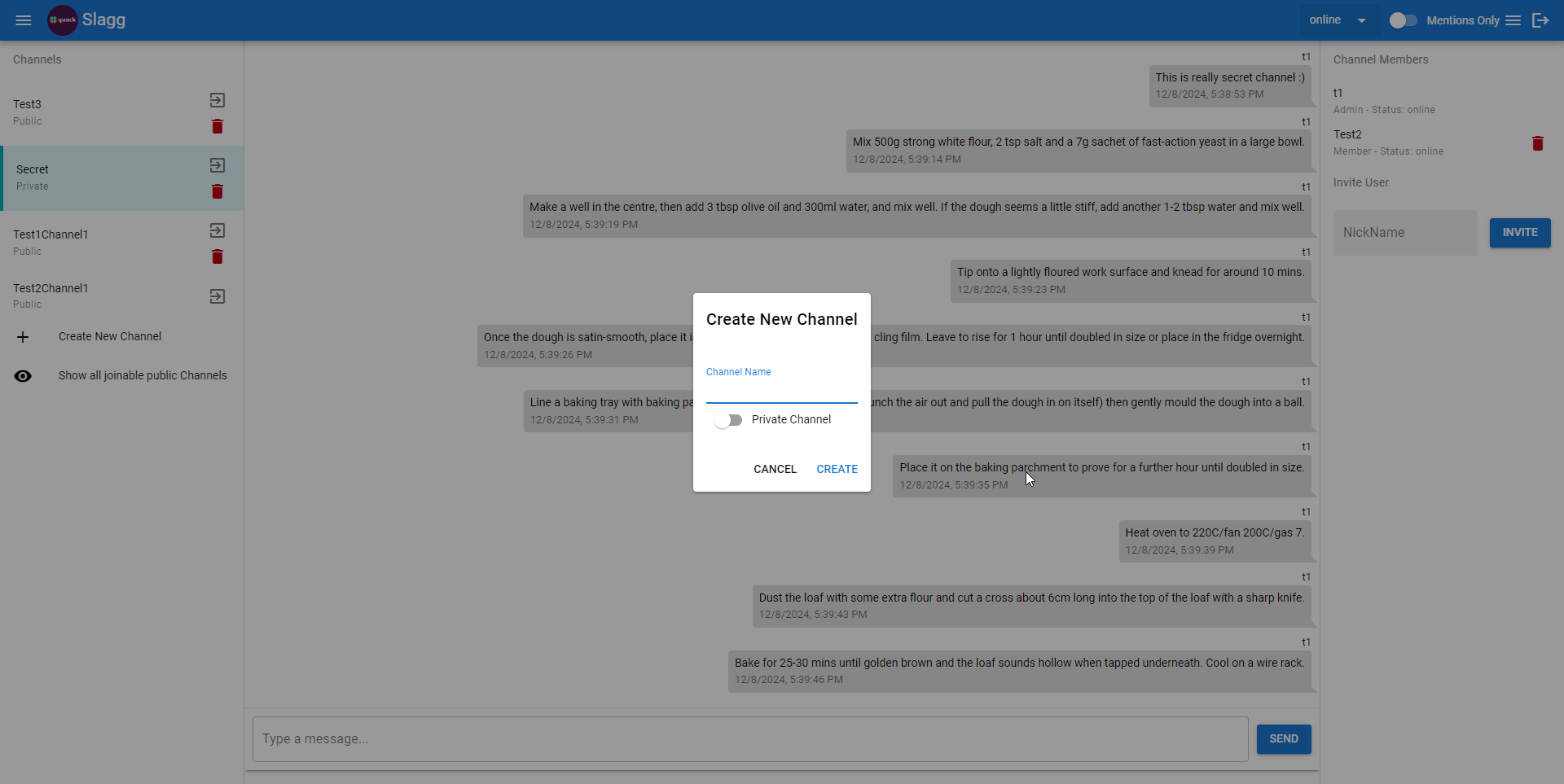
1. Ukážky
   1. Chat
      1. Na PC



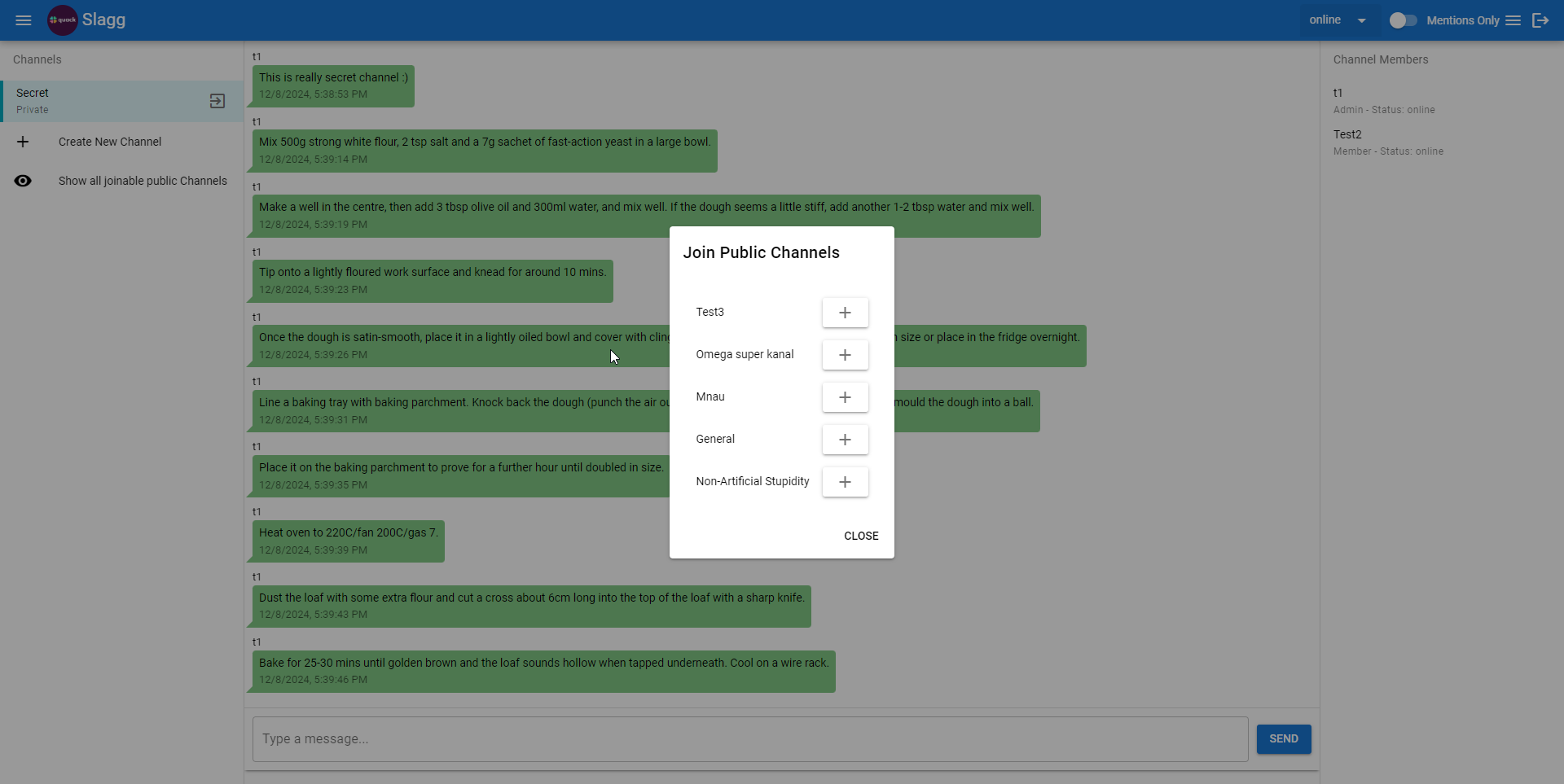
* + 1. Na smartphone



* 1. Vytvorenie nového kanála



* 1. Pripojenie sa do verejného kanála



* 1. Hlasovanie o vyhodení z kanála

