

Slash

Development of Progressive Web Applications

Created By: Artur Kozubov, Artem Zaitsev

Instructed by: [Ing. Matúš Krajčovič](#)

Lectured by: [Ing. Eduard Kuric, PhD.](#)

Requirements

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

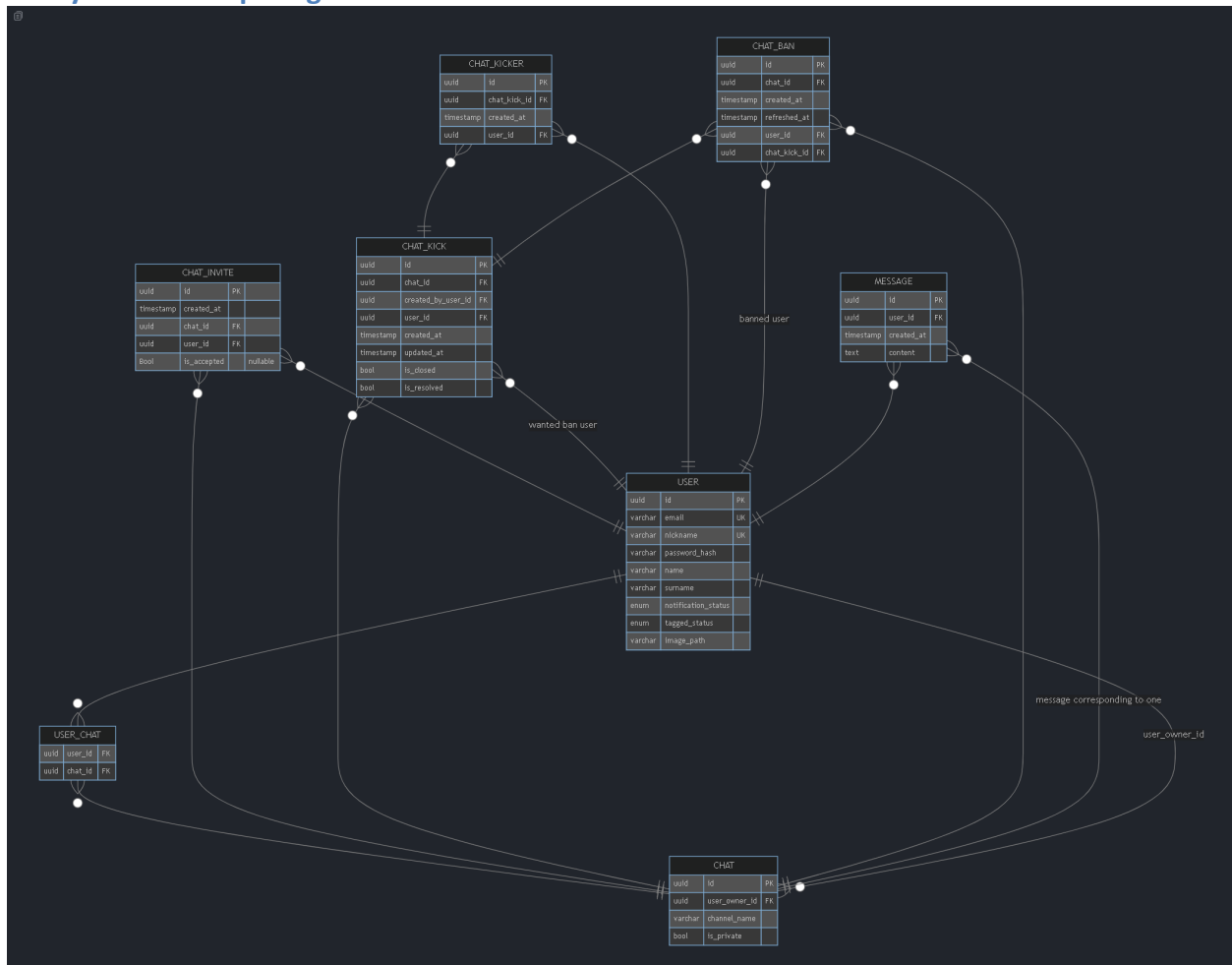
1. registrácia, prihlásenie a odhlásenie používateľa
 - používateľ má meno a priezvisko, nickName a email
2. používateľ vidí zoznam kanálov, v ktorých je členom
 - pri opustení kanála, alebo trvalom vyhodení z kanála je daný kanál odobratý zo zoznamu
 - pri pozvánke do kanála je daný kanál zvýraznený 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 ľubovoľný používateľ cez príkaz /join channelName [private]
 - do súkromného kanála môže pridávať/odoberať používateľovi iba správca kanála cez príkazy /invite nickName a /revoke nickName
 - do verejného kanála sa môže pridať ľubovoľný 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 spraví 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 históriu 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 :-)

Database structure

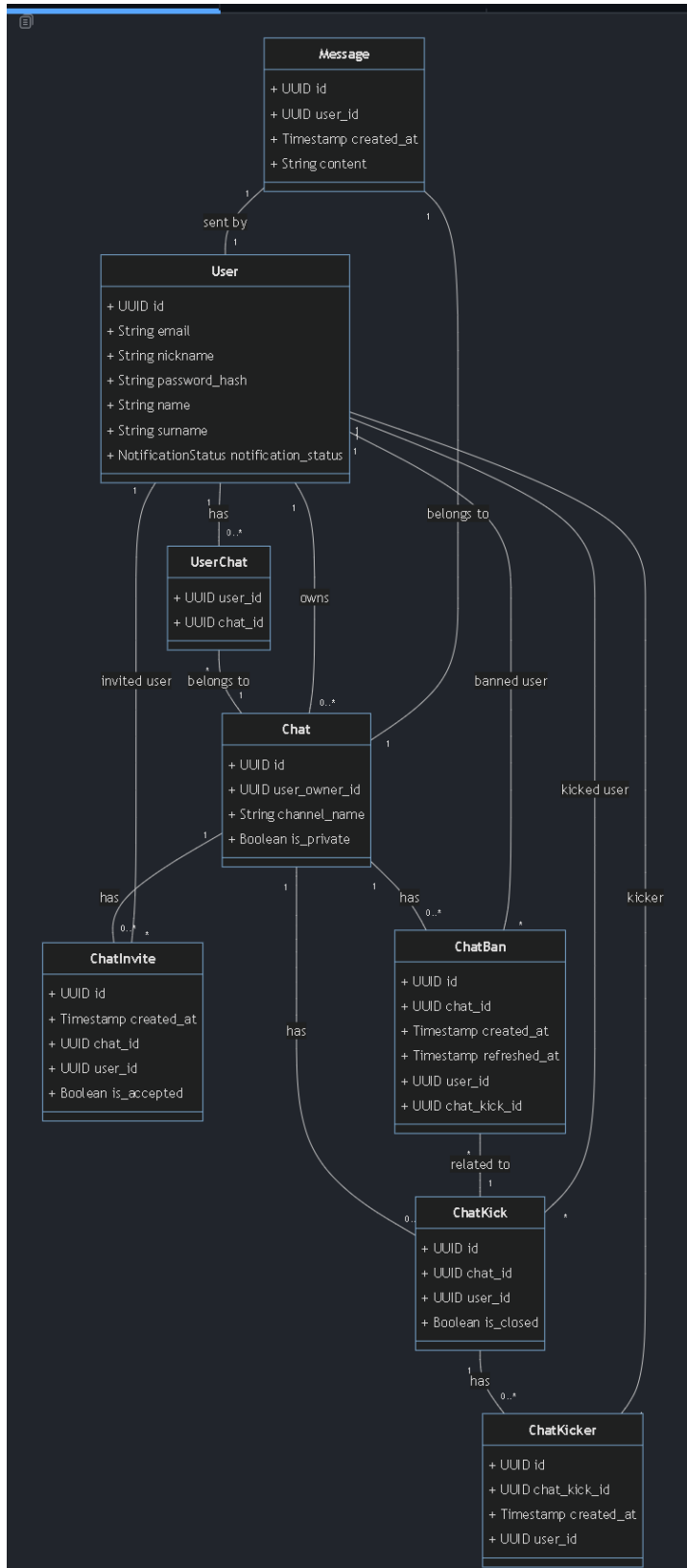
Latest migrations

- `create_user_notify_when_tagged_table.ts`
 - Added new status column to control if notification are only when tagged (@nickname)
- `create_add_image_to_users_table.ts`
 - Added `image_path` column to store user profile image
- `create_is_resolved_table.ts`
 - Added `is_resolved` column to `chat_kicks` table if the user is unbanned
- `create_chat_invites_updated_at_table.ts`
 - Added `updated_at` column to `chat_invites` table to control when the invite was accepted
- `create_update_to_have_created_by_users_table.ts`
 - Added `created_by_user_id` column to `chat_kicks` table to control who kicked the user

Entity-Relationship Diagram



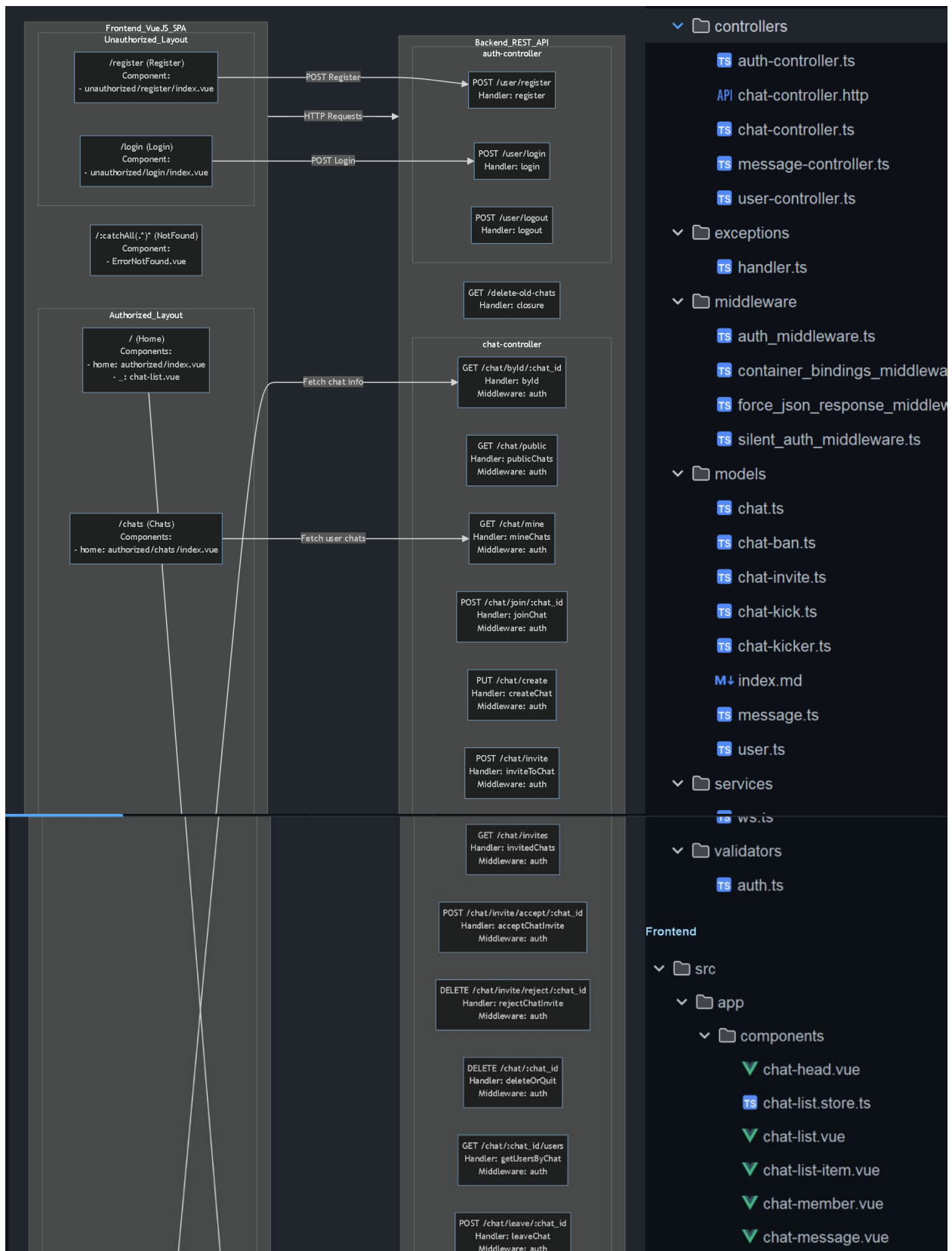
UML Class Diagram

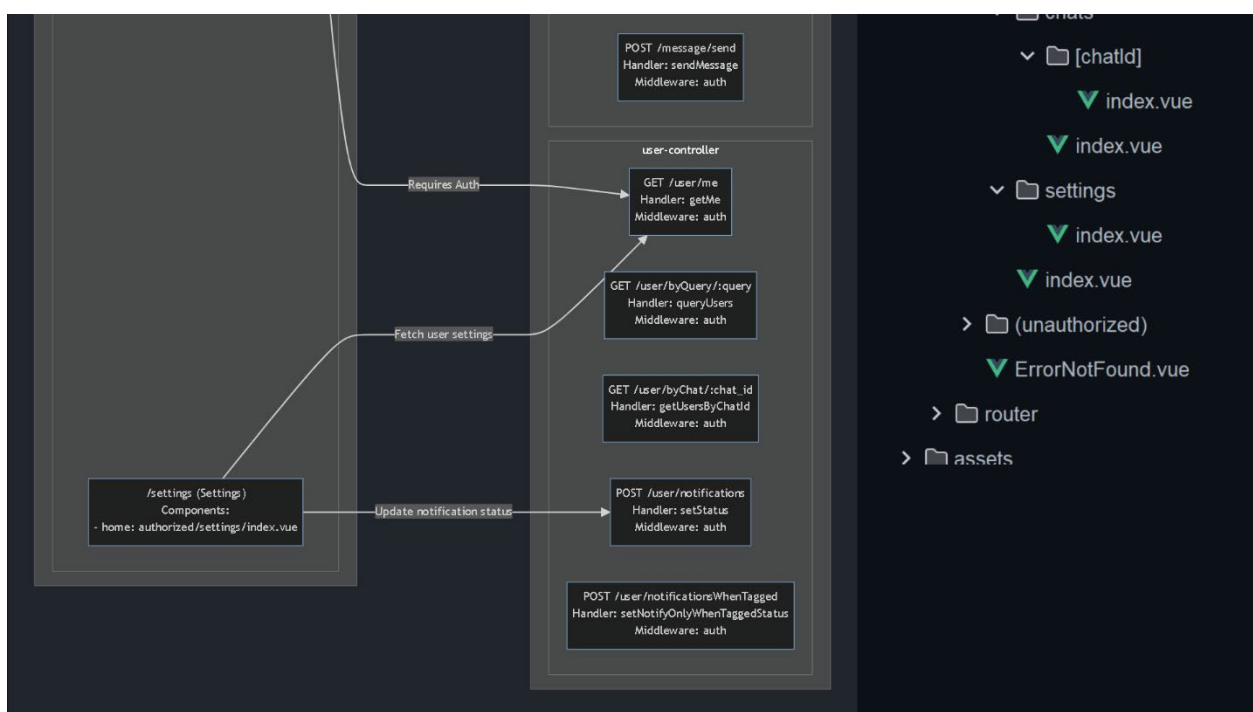


Project Architecture

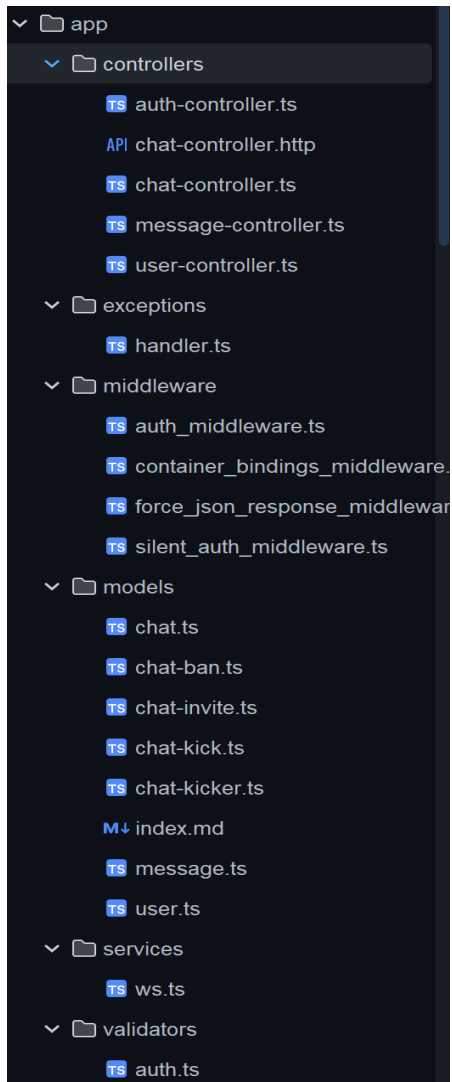
Pages to Controller & File Structure

Next chart shows the relationship between the frontend and backend structure. The frontend is a Vue.js SPA and the backend is a REST API. Shows which page requires which endpoint to be fetched.

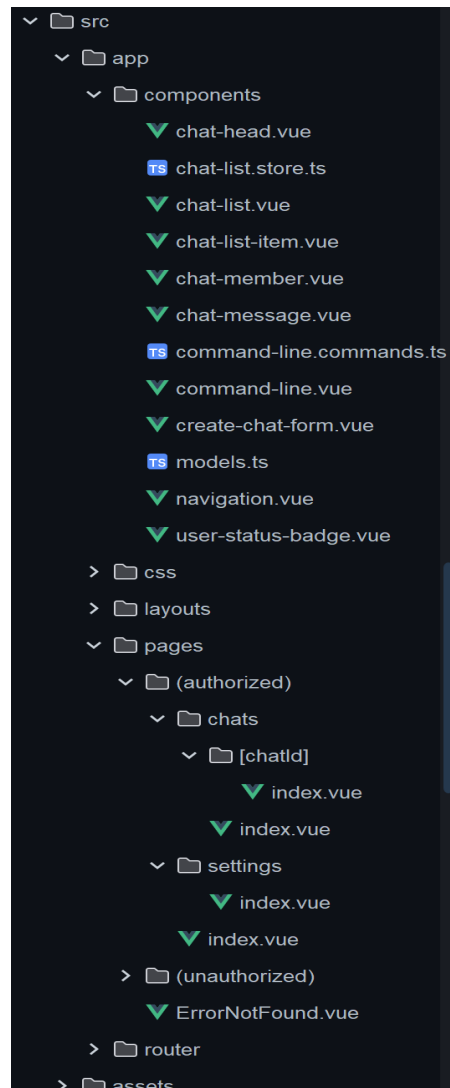




Backend



Frontend



Project Structure

Libraries

On the frontend side, we in addition to standard Vue.js + Quasar structure with libraries we use:

```
"@quasar/extras": "^1.16.4",  
"axios": "^1.7.7", // Used for auth interceptors and easy-to-use HTTP  
requests  
"core-js": "^3.31.1",  
"quasar": "^2.16.0",  
"vue": "^3.4.18",  
"vue-router": "^4.0.12", // Used for routing in SPA. We also used a Next.js  
like pages structure.  
"@slash/backend": "*" // We import the backend package, as we use npm  
monorepo, what allows us to use the backend types and interfaces in the  
frontend for type safe development.  
"socket.io": "^4.8.1" // Used for real-time communication (sending messages,  
notifications, etc.)
```

On the backend side, we use:

```
"@adonisjs/auth": "^9.2.4",  
"@adonisjs/core": "^6.14.1",  
"@adonisjs/cors": "^2.2.1",  
"@adonisjs/lucid": "^21.3.0",  
"@adonisjs/session": "^7.5.0",  
"@vinejs/vine": "^2.1.0", // In register and login controllers, we use vine  
for validation user input  
"luxon": "^3.5.0",  
"node-cron": "^3.0.3", // For checking and deleting inactive chats  
"pg": "^8.13.0",  
"reflect-metadata": "^0.2.2",  
"socket.io": "^4.8.1" // Used for real-time communication (sending messages,  
notifications, etc.)
```

User Interface

