ChatApp

Użyte technologie i biblioteki

- **NextJS**

- **TypeScript** - to język programowania opierający się na JavaScript, który dodaje do niego typy i bardziej zaawansowane funkcjonalności. TypeScript jest transpilowany do JavaScript, co umożliwia wykorzystanie go we wszystkich miejscach, gdzie obsługiwany jest JavaScript. TypeScript jest szczególnie przydatny do tworzenia dużych i skomplikowanych aplikacji, ponieważ pozwala na lepszą współpracę między programistami i zapewnia lepszą diagnostykę błędów w czasie kompilacji.

- **NextAuth** - biblioteka do uwierzytelniania i autoryzacji dla aplikacji opartych na frameworku Next.js. **Pozwala ona na łatwe i szybkie integracje** z popularnymi dostawcami uwierzytelniania takimi jak Google, GitHub, Twitter i inne. NextAuth.**js używa nowoczesnych technologii jak JSON Web Tokens (JWT) i OAuth 2.0,** aby zapewnić bezpieczne i skalowalne uwierzytelnianie dla aplikacji.

- **Prisma** - to narzędzie do zarządzania bazami danych, które umożliwia łatwe i szybkie tworzenie schematów bazy danych i wykonywanie zapytań SQL. Prisma jest w pełni zintegrowana z językiem programowania GraphQL

- **MongoDB** - jest dokumentowym systemem baz danych NoSQL. Został stworzony, aby umożliwić elastyczne i skalowalne przechowywanie danych, które są zorganizowane jako kolekcje dokumentów zamiast tabel z kolumnami i wierszami. Jest często wybierany do aplikacji webowych i aplikacji mobilnych ze względu na jego elastyczność i łatwość użycia.

- **TailwindCss** - to narzędzie do tworzenia stylów CSS, które polega na korzystaniu z gotowych klas i utworzeniu własnych. Jest to biblioteka utility-first CSS, co oznacza, że ​​skupia się na udostępnianiu wielu gotowych klas, zamiast na zdefiniowaniu stylów dla każdego elementu. Dzięki temu można szybko i łatwo tworzyć interfejsy użytkownika bez potrzeby pisania dużej ilości kodu CSS. Tailwind CSS jest bardzo popularny w środowisku front-end i jest często używany do tworzenia szybkich i responsywnych aplikacji webowych.

- **GraphQL** - to język zapytań i zarządzania danymi dla API, opracowany przez Facebooka. Umożliwia on żądanie dokładnie tych danych, których aplikacja potrzebuje, oraz unikanie nadmiarowych danych i niedopasowania schematów. GraphQL jest alternatywą dla tradycyjnych API REST i pozwala na elastyczne i precyzyjne żądanie danych.

**Apollo GraphQL** - Apollo GraphQL to narzędzie do tworzenia i utrzymywania aplikacji opartych na GraphQL. Zawiera ono zestaw narzędzi i bibliotek, takich jak biblioteka klienta i serwera, które umożliwiają łatwą integrację GraphQL z aplikacjami webowymi i mobilnymi. Apollo GraphQL umożliwia także optymalizację i caching żądań, co przekłada się na szybsze i bardziej efektywne działanie aplikacji. W skrócie, Apollo GraphQL to kompleksowe rozwiązanie dla aplikacji opartych na GraphQL, które umożliwia łatwe i efektywne tworzenie aplikacji z wykorzystaniem tego języka zapytań.

Lodash - to biblioteka JavaScript, która zawiera szereg funkcji użytecznych dla programowania. Lodash jest szczególnie przydatny w React, ponieważ umożliwia łatwiejsze i bardziej efektywne programowanie poprzez dostarczanie gotowych funkcji i narzędzi, takich jak funkcje pomocnicze do manipulowania tablicami, obiektami i ciągami znaków. Użycie Lodash może znacznie skrócić kod i zwiększyć jego czytelność, co może przyczynić się do lepszego zrozumienia i utrzymania kodu. Lodash jest dostępny jako zewnętrzna biblioteka i można go łatwo zintegrować z aplikacją React.

**Struktura folderów**

🡸=========================== **Backend** ======================================🡺

1. **Backend** – Część aplikacji odpowiedzialna za logikę biznesową, przechowywanie danych i komunikację z innymi systemami.

- src

- graphql

- resolvers – Tworzenie Resolverów

- typeDefs – Tworzenie definicji typów

- index.ts – zawiera wszystkie definicje typów

- index.ts

🡸=========================== **Frontend** ======================================🡺

1. **Frontend** – Część strony widoczna dla użytkownika, odpowiada głównie za interfejs graficzny i interakcję z użytkownikami:

- src – niedomyślny folder, w którym znajdują się podfoldery:

- pages – strony i podstrony

- Chat – Podstrona konwersacji

- Auth – Podstrona z ekranem logowania

- api

[...nextauth].js – konfiguracja NextAuth.js

- styles – style strony

- lib - biblioteki

next-auth.d.ts – plik z deklaracjami danych, dostarcza informacji o typie

- prisma.ts – klient prisma

- prisma

- prisma.schema – zawiera schemat bazy danych

- graphql

- apollo-client.ts – Tworzenie klienta apollo

- operations -

3**. inne**

**.env.local – plik konfiguracyjny, zawiera zmienne środowiskowe dla aplikacji.**

Routing w **NextJS** – zarządzany jest przez moduł **‘next/router’**. Oparty jest o koncepcję dynamicznych ścieżek, zawierających zmienne, co pozwala nam na np. **automatyczne generowanie odpowiedniej strony**. Można tworzyć zagnieżdżone struktury podstron np. blog/details

**NextAuth**

**Zastosowani dostawcy:**

**- Google – możliwość logowania się za pomocą google**

**Zastosowane adaptery (model bazy):**

**- prisma**

**Generowanie ciągu znaków „secret”:**

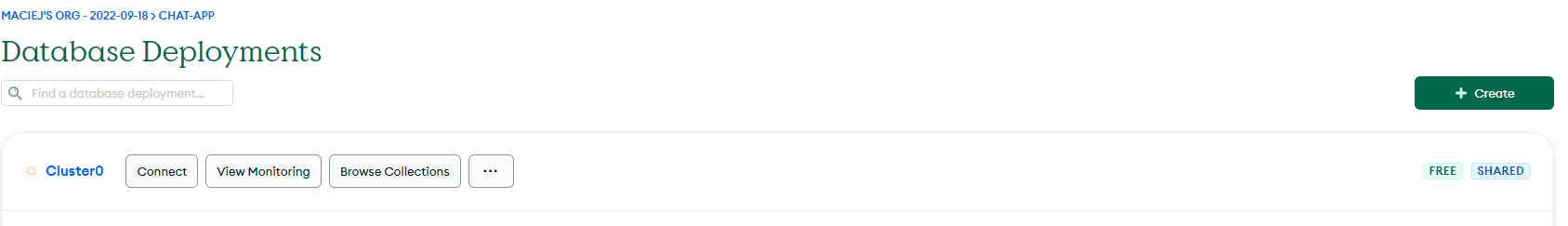
**Declaration Files, Pliki deklaracji** (znane również jako pliki ".d.ts") to sposób dostarczania informacji o typie dla kodu JavaScript w TypeScript. Służą do opisania typów zmiennych, funkcji i modułów w bibliotece JavaScript, tak aby TypeScript mógł wykorzystać te informacje do zapewnienia sprawdzania typów i lepszego narzędzia. Pliki deklaracji są ważne w TypeScript, ponieważ JavaScript jest językiem dynamicznie typowanym, ale TypeScript jest językiem statycznie typowanym. Pliki deklaracji pozwalają TypeScript zrozumieć typy obiektów w bibliotece JavaScript i zapewniają lepsze doświadczenie dla programisty pracującego z tą biblioteką.:

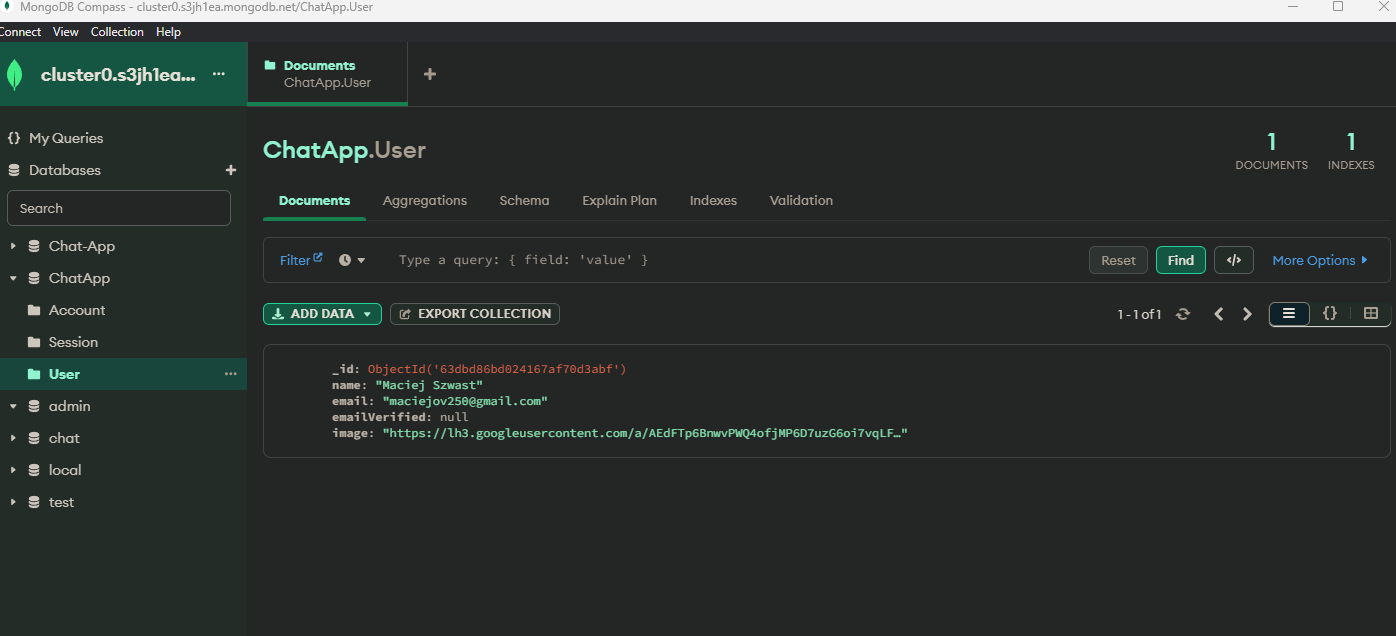
Dodawanie typu username do pliku types.d.ts

**Text

Description automatically generated**

**Baza danych – MongoDB**

****



**Zapobieganie migotania przy odświeżaniu strony**. Sprawia, że użytkownik ma lepsze odczucia podczas korzystania ze strony:

getServerSideProps - jest specjalną funkcją w frameworku Next.js, która umożliwia wykonanie operacji serwerowych przed wyrenderowaniem strony. Jest to bardzo przydatne, gdy potrzebujesz dostępu do danych, które nie są dostępne na kliencie, np. z bazy danych, API lub pliku.

jest wywoływana tylko na serwerze i może zwracać dane, które są następnie dostępne dla komponentu jako właściwość. Dzięki temu można uzyskać bardziej bezpieczne i niezawodne działanie aplikacji, ponieważ dane są pobierane na serwerze, a nie na kliencie.

**Apollo i graphql**

**Resolver** w React jest funkcją, która umożliwia asynchroniczne pobieranie danych przed renderowaniem komponentu. Dzięki temu można uniknąć pojawienia się błędu "brakujących danych" podczas wyświetlania strony. Resolver jest używany w celu zapewnienia, że dane są dostępne dla komponentu przed jego wyrenderowaniem na stronie.

Apollo **InMemoryCache** to pamięć podręczna opracowana przez Apollo Client, która umożliwia przechowywanie i śledzenie danych zwracanych z zapytań GraphQL na poziomie klienta. InMemoryCache pozwala na szybką i efektywną odpowiedź na żądania, ponieważ dane są pobierane z pamięci podręcznej, a nie z serwera. W przypadku aktualizacji danych na serwerze, InMemoryCache automatycznie je aktualizuje, co zapewnia, że dane są zawsze aktualne i nie trzeba ponownie ich pobierać z serwera.

**Schemat GraphQL** to formalny opis typów danych, operacji i relacji w aplikacji GraphQL. Schemat określa, jakie dane są dostępne w aplikacji i jakie operacje można na nich wykonać. Schemat jest używany przez serwer GraphQL do weryfikacji i wykonania żądań klienta. Zawiera on opis typów danych, które są dostępne w aplikacji, jak również reguły dotyczące tego, jak dane te są powiązane i jakie operacje można na nich wykonać. Schemat jest jednym z ważniejszych składników aplikacji GraphQL i jest kluczowy dla zapewnienia bezpieczeństwa i łatwego rozwoju aplikacji.

**makeExecutableSchema** jest funkcją w bibliotece "graphql-tools", która pozwala na tworzenie pełnego schematu GraphQL na podstawie opisu typów i resolverów. Funkcja makeExecutableSchema łączy opis typów i resolverów w jeden spójny schemat, który może być używany przez serwer GraphQL do weryfikacji i wykonywania żądań od klienta. makeExecutableSchema jest przydatne, ponieważ umożliwia łatwe i efektywne tworzenie schematów GraphQL, które są spójne i zgodne z wymaganiami aplikacji. W rezultacie, użycie makeExecutableSchema może przyczynić się do skrócenia czasu rozwoju i poprawy jakości aplikacji GraphQL.