Programowanie systemów rozproszonych	Projekt
Distributed Chat System	Imię i nazwisko: Jakub Olejarczyk Numer albumu: s086748 Kielce 2024/2025

Spis treści

1. Wstęp

Aplikacja czatowa działa w czasie rzeczywistym, umożliwiając użytkownikom komunikację tekstową w ramach dedykowanych pokoi czatowych. System opiera się na architekturze rozproszonej, co pozwala na integrację z wieloma serwerami, zapewniając płynne i spójne doświadczenie użytkownika niezależnie od obciążenia sieci. Dzięki temu rozwiązanie efektywnie obsługuje dużą liczbę użytkowników jednocześnie, oferując niskie opóźnienia, wysoką dostępność oraz skalowalność w miarę rosnących potrzeb.

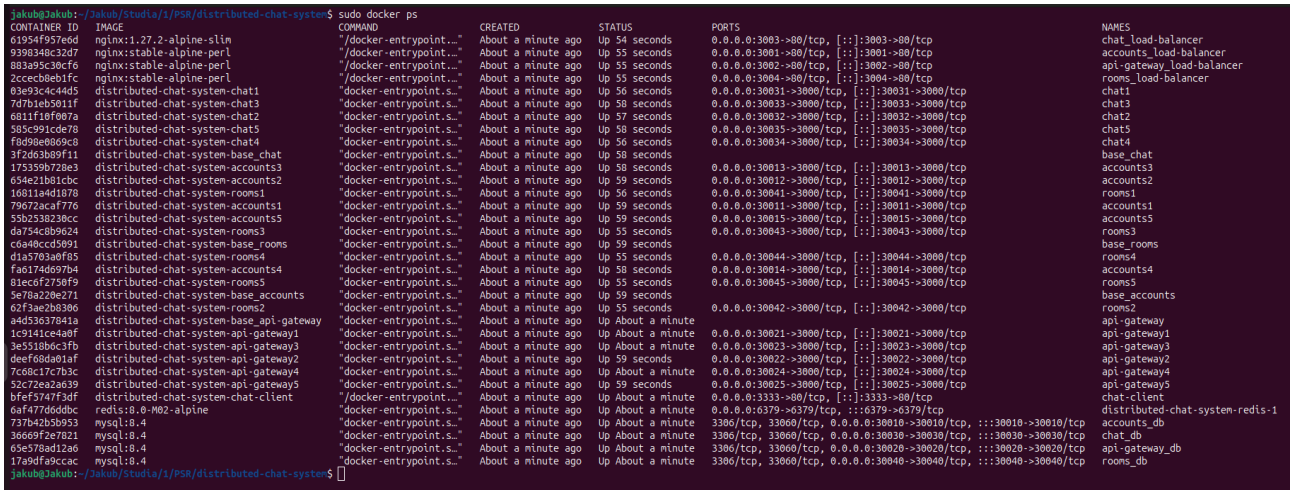
Aplikacja posiada zaawansowane funkcje, takie jak rejestracja i logowanie użytkowników z wykorzystaniem bezpiecznych tokenów JWT, zarządzanie profilami użytkowników oraz możliwość tworzenia pokoi czatowych. Powiadomienia w czasie rzeczywistym informują o nowych wiadomościach lub działaniach w pokojach, w których użytkownik bierze udział, co zwiększa interaktywność platformy.

Rozwiązanie zostało zbudowane w oparciu o nowoczesne technologie. Komunikację w czasie rzeczywistym obsługuje Socket.io, backend działa na Node.js, a Redis pełni rolę bufora wiadomości i mechanizmu kolejkowania zdarzeń. Architektura kontenerowa oparta na Dockerze zapewnia łatwość wdrażania i niezależność od środowiska.

Interfejs użytkownika został zaprojektowany z wykorzystaniem Angulara oraz biblioteki komponentów PrimeNG, co gwarantuje spójny wygląd i intuicyjną obsługę. Zarządzanie stanem aplikacji odbywa się za pomocą NgRx, co zapewnia wysoką wydajność i niezawodność systemu.

Aplikacja jest stabilna, skalowalna i przyjazna dla użytkownika, doskonale sprawdzając się zarówno w zastosowaniach prywatnych, jak i biznesowych.

2. Struktura kontenerów



Aplikacja składa się z wielu kontenerów, z których każdy realizuje pojedynczą funkcjonalność, składającą się na kompletną aplikację czatu.

3. Konfiguracja projektu

1. Link do projektu: www.github.com/jaqba98/distributed-chat-system
2. Wymagane oprogramowanie:
    1. NodeJS www.nodejs.org
    2. PNPM www.pnpm.io
    3. Docker www.docker.com
    4. Git www.git-scm.com
    5. NX www.nx.dev
3. Kroki do wykonania w celu uruchomienia projektu:
    1. Instalacja oprogramowania
        1. Zainstaluj NodeJS z oficjalnej strony oprogramowania
        2. Zainstaluj PNPM za pomocą odpowiedniej komendy

            ```
            npm install -g pnpm@latest-10
            ```

        3. Zainstaluj NX za pomocą odpowiedniej komendy

            ```
            pnpm install -g nx
            ```

        4. Zainstaluj Docker z oficjalnej strony oprogramowania
        5. Zainstaluj Git z oficjalnej strony oprogramowania

2. Przygotowanie projektu
    1. Uzyskaj link do projektu
        1. Przejdź na stronę projektu na serwisie GitHub
        2. Wciśnij zielony przycisk „Code” i wybierz zakładkę „HTTPS”
        3. Skopiuj wyświetlony link (w następnych krokach będzie nam potrzebny)
    2. Przejdź do dowolnej lokalizacji na twoim komputerze i wykonaj polecenie klonujące repozytorium z serwisu GitHub na twój komputer lokalny

        ```
        git clone https://github.com/jaqba98/distributed-chat-system.git
        ```

    3. Przejdź do nowo utworzonego folderu i wykonaj polecenie instalacji pnpm.

        ```
        pnpm install -r
        ```

    4. Zbuduj projekt

        ```
        nx run-many --target=build
        ```

3. Uruchomienie projektu
    1. Za pomocą docker compose uruchom projekt

        ```
        sudo docker compose up -d
        ```

    2. Weryfikacja czy wszystko działa – aby mieć pewność że wszystko działa wykonaj polecenie sudo docker logs na dowolnym serwisie

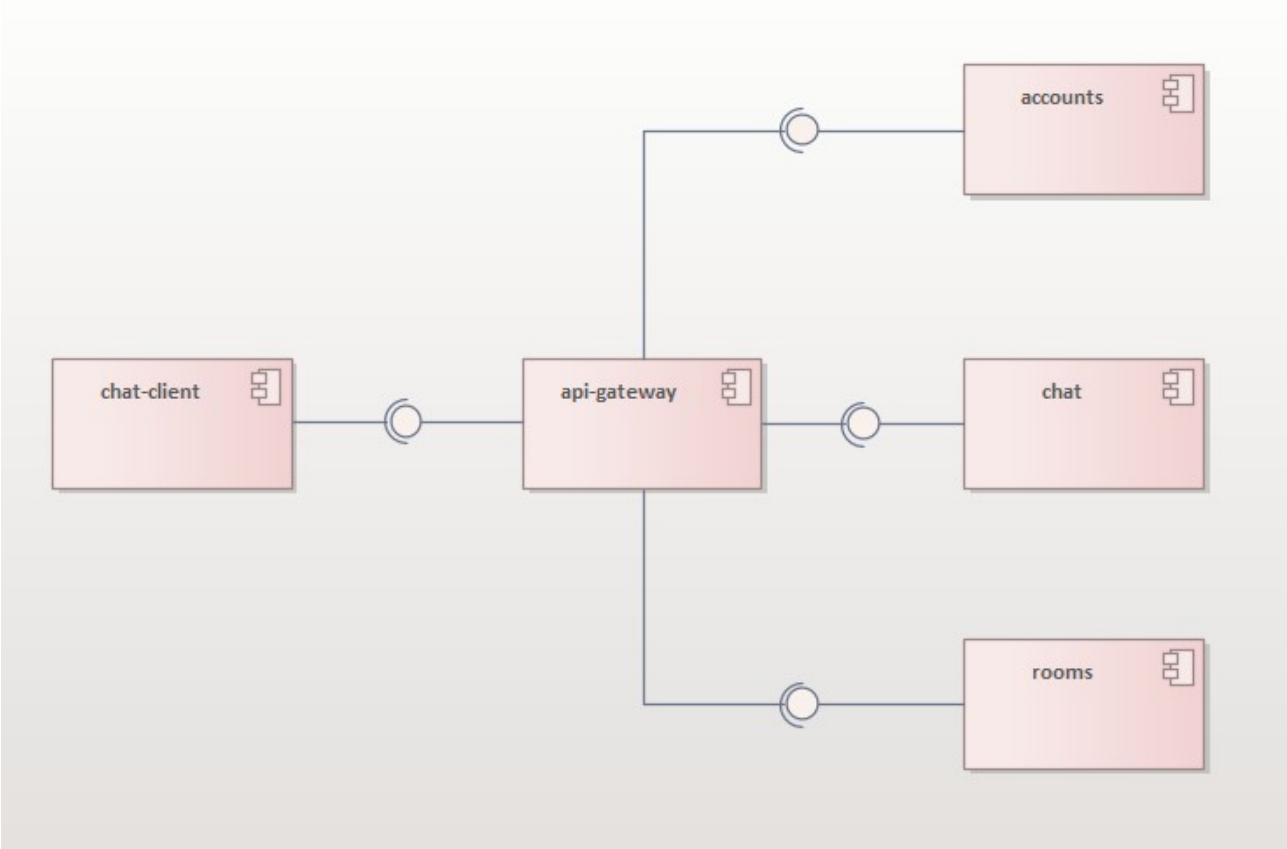
        ```
        sudo docker logs accounts1
        ```

        Jeśli w konsoli wyświetlił się napis "" wtedy mamy pewność że wszystko uruchomiło się poprawnie.
4. Wyłączenie projektu

```
sudo docker logs accounts1
```

3. Architektura systemu

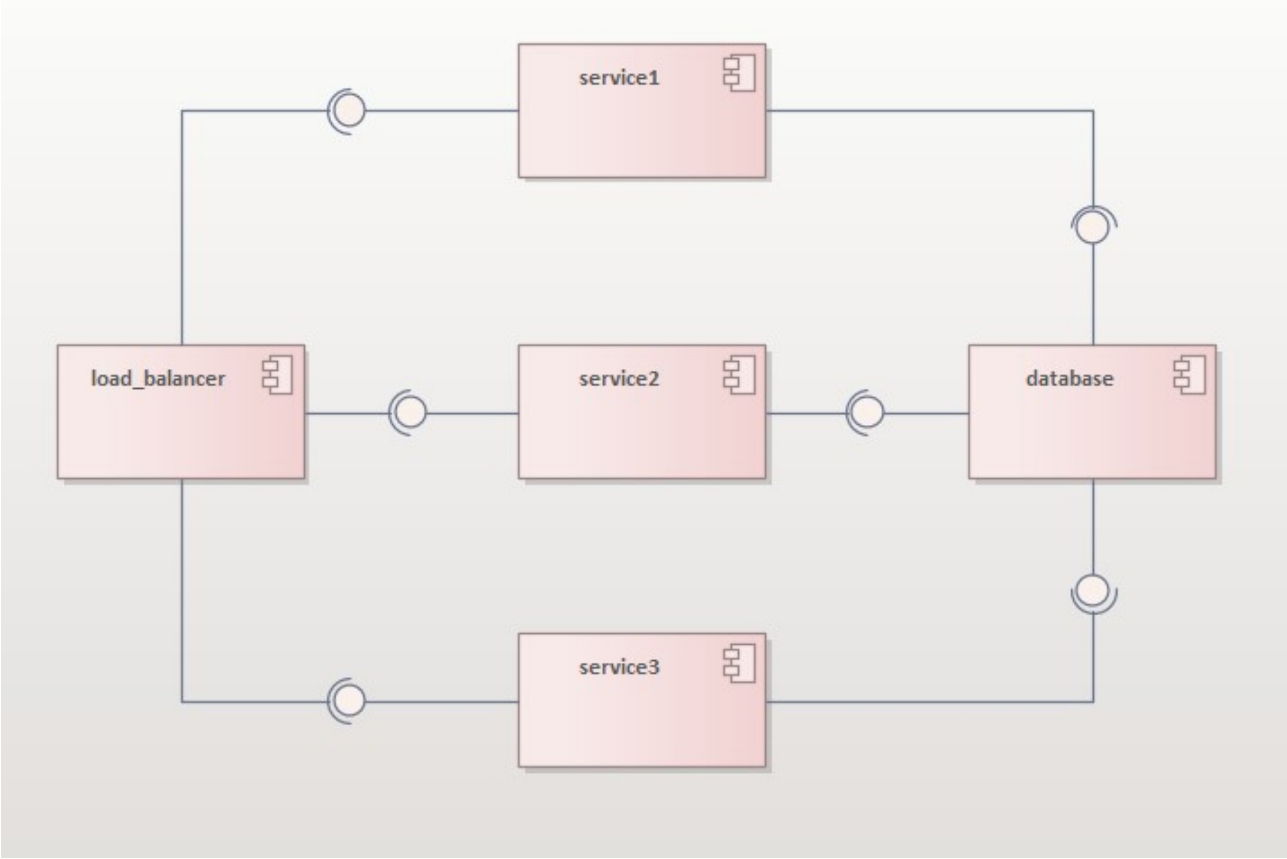
Architektura microservices na najwyższym poziomie



Składa się z:

- chat-client – Aplikacja frontendowa działająca po stronie klienta, zawierająca pełny interfejs użytkownika oraz funkcjonalności umożliwiające interakcję z systemem w czasie rzeczywistym.
- api-gateway – Mikroserwis pełniący funkcję centralnego punktu dostępu, zarządzający ruchem sieciowym pomiędzy klientem a poszczególnymi mikroserwisami.
- accounts – Mikroserwis odpowiedzialny za zarządzanie danymi użytkowników, w tym rejestrację, logowanie, autoryzację oraz przechowywanie.
- chat – Mikroserwis zajmujący się obsługą komunikacji w czasie rzeczywistym, zarządzający wymianą wiadomości pomiędzy użytkownikami, w tym kolejkowaniem i dostarczaniem ich do odpowiednich odbiorców.
- rooms – Mikroserwis dedykowany zarządzaniu pokojami czatowymi, obejmujący tworzenie.

Architektura pojedyńczego microservice



Każdy z mikroserwisów składa się z następujących komponentów:

- load-balancer – Odpowiedzialny za równoważenie obciążenia pomiędzy wieloma instancjami serwisu, zapewniając wysoką dostępność oraz optymalne wykorzystanie zasobów.
- service – Główna jednostka logiczna mikroserwisu, realizująca jego funkcjonalności; może być powielana wielokrotnie, aby skalować aplikację w zależności od obciążenia i wymagań.
- database – Dedykowana baza danych dla danego mikroserwisu, służąca do przechowywania oraz zarządzania danymi związanymi z jego działaniem.

3. Komunikacja w czasie rzeczywistym

Główną funkcją aplikacji jest umożliwienie użytkownikom prowadzenia rozmów w czasie rzeczywistym w ramach wybranego pokoju czatowego. Oznacza to, że jeśli użytkownik „A” wyśle wiadomość, to wszyscy pozostali użytkownicy znajdujący się w tym samym pokoju natychmiastowo ją otrzymują, bez opóźnień. Użytkownicy aplikacji łączą się z jedną z wielu instancji mikroserwisu „chat”, które są rozmieszczone w różnych lokalizacjach. Pomimo tego, że użytkownicy mogą być podłączeni do różnych serwerów, rozproszonych geograficznie, na przykład na dwóch różnych kontynentach, komunikacja między nimi jest synchronizowana dzięki technologii „Redis”. Redis zapewnia efektywne buforowanie i synchronizację danych pomiędzy serwerami, co pozwala na utrzymanie płynnej i natychmiastowej wymiany wiadomości niezależnie od miejsca, w którym znajduje się użytkownik. Dzięki temu użytkownicy mogą bezproblemowo i w czasie rzeczywistym komunikować się, niezależnie od lokalizacji geograficznej i obciążenia poszczególnych serwerów.

4. Technologie

1. Frontend:
    1. HTML: Język znaczników używanym do tworzenia struktury stron internetowych.
    2. CSS / SCSS: CSS odpowiada za stylizację stron internetowych, a SCSS to jego rozszerzenie oferujące zaawansowane funkcje, takie jak zmienne i zagnieżdżanie stylów.
    3. TypeScript: Jest rozszerzeniem JavaScript z obsługą statycznego typowania, co ułatwia tworzenie i utrzymanie złożonych aplikacji.
    4. Angular: Jest frameworkiem frontendowym do budowy dynamicznych i modułowych aplikacji internetowych w oparciu o TypeScript.
    5. RxJs: Biblioteka do reaktywnego programowania w JavaScript, umożliwiająca obsługę strumieni danych i zdarzeń asynchronicznych.
    6. Nx: Narzędzie do zarządzania projektami monorepo, które usprawnia pracę nad wieloma aplikacjami i bibliotekami w jednym repozytorium.
    7. NgRx: Biblioteka do zarządzania stanem w aplikacjach Angular z wykorzystaniem wzorca Redux.
    8. PrimeNG: Zbiór komponentów UI dla Angulara, umożliwiający szybkie tworzenie atrakcyjnych interfejsów użytkownika.
2. Backend:
    1. Socket.io: Biblioteka do obsługi komunikacji w czasie rzeczywistym między klientem a serwerem, często używana w aplikacjach czatowych.
    2. Node.js: Środowisko uruchomieniowe dla JavaScript, umożliwiające tworzenie aplikacji serwerowych.
    3. JWT: Standard tokenów służący do bezpiecznej autoryzacji użytkowników w aplikacjach.
    4. Docker: Platforma do konteneryzacji aplikacji, pozwalająca na łatwe uruchamianie i skalowanie środowisk.
    5. MySQL: Relacyjna baza danych, często używana do przechowywania i zarządzania danymi w aplikacjach internetowych.
    6. Redis: Szybki magazyn danych w pamięci, używany do buforowania i obsługi kolejek.