Projekt „Chatbot”

Projekt „Chatbot” to aplikacja komunikacyjna zbudowana w oparciu o technologie Node.js oraz NestJS, które umożliwiają tworzenie skalowalnych, wydajnych i łatwych do rozwoju aplikacji backendowych. Aplikacja wykorzystuje TypeScript, który zapewnia większe bezpieczeństwo i czytelność kodu, a jej struktura opiera się na wzorcu Model-View-Controller (MVC), co sprzyja utrzymaniu czystości kodu i modularności.

System umożliwia użytkownikom rejestrację, logowanie oraz komunikację za pomocą wiadomości tekstowych. Aplikacja jest zintegrowana z zewnętrznym modelem sztucznej inteligencji – Mistral AI, który umożliwia generowanie odpowiedzi bota na pytania i zapytania użytkowników. Wszystkie dane, takie jak informacje o użytkownikach, czatach i wiadomościach, są przechowywane w bazie danych Microsoft SQL Server (MSSQL), co zapewnia trwałość i bezpieczeństwo przechowywanych informacji.

Aplikacja może być wykorzystana w wielu różnych scenariuszach, zarówno przez firmy, jak i użytkowników indywidualnych.

Technologie użyte w aplikacji

Aplikacja została zbudowana przy użyciu następujących technologii i narzędzi:

- **Node.js** – środowisko uruchomieniowe do obsługi aplikacji backendowej
- **NestJS** – framework dla aplikacji backendowych opartych na TypeScript
- **TypeScript** – statycznie typowany język zapewniający większą bezpieczeństwo i czytelność kodu
- **TypeORM** – ORM do interakcji z bazą danych
- **Microsoft SQL Server (MSSQL)** – relacyjna baza danych używana do przechowywania danych użytkowników, wiadomości i czatów
- **Mistral AI API** – zewnętrzny model AI do generowania odpowiedzi
- **JWT (JSON Web Token)** – mechanizm uwierzytelniania użytkowników

Instalacja i konfiguracja

2.1 Wymagania systemowe

Aby uruchomić aplikację, wymagane jest środowisko obsługujące **Node.js, NPM** oraz baza danych **Microsoft SQL Server (MSSQL)**. Dodatkowo, aplikacja korzysta z API **Mistral AI**, dlatego konieczne jest posiadanie klucza API.

2.2 Instalacja

1. **Pobranie kodu źródłowego** – kod aplikacji powinien zostać pobrany na lokalny komputer lub serwer.
2. **Instalacja zależności** – aplikacja wymaga bibliotek NestJS, TypeORM, a także obsługi MSSQL i API Mistral AI.
3. **Konfiguracja plików środowiskowych** – klucz API do Mistral AI oraz dane dostępowe do bazy MSSQL powinny zostać umieszczone w pliku .env.
4. **Konfiguracja bazy danych** – aplikacja wykorzystuje Microsoft SQL Server i automatycznie tworzy wymagane tabele na pods.
   Aplikacja wykorzystuje **Microsoft SQL Server (MSSQL)** jako system zarządzania bazą danych. Do obsługi połączenia i operacji na bazie danych używany jest **TypeORM**, który automatycznie zarządza encjami i migracjami.

   - **Konfiguracja połączenia**
     Połączenie z bazą danych jest zdefiniowane w pliku data-source.ts. Aplikacja łączy się z lokalnym serwerem MSSQL i dynamicznie wykrywa wszystkie encje w katalogu projektu.
   - **Główne parametry konfiguracji:**
     1. type: 'mssql' – określa typ bazy danych jako Microsoft SQL Server.
     2. host: 'localhost' – domyślnie baza działa lokalnie, ale można skonfigurować zdalny serwer.
     3. port – numer portu, na którym działa MSSQL.
     4. username, password – dane uwierzytelniające do bazy danych.
     5. database – nazwa bazy danych, która będzie używana przez aplikację.
     6. synchronize: true – opcja umożliwiająca automatyczne tworzenie tabel na podstawie encji (zalecane tylko w środowisku deweloperskim).
     7. entities – dynamiczne wykrywanie wszystkich encji w katalogu projektu.
     8. trustServerCertificate: true – opcja wymagająca zaufania do certyfikatu serwera MSSQL (przydatne dla lokalnych instancji).

2.3 Uruchomienie aplikacji

Aplikacja może być uruchomiona zarówno lokalnie, jak i w środowisku produkcyjnym. Po poprawnej konfiguracji aplikacja będzie dostępna pod odpowiednim adresem URL. W przypadku problemów należy sprawdzić logi aplikacji, poprawność połączenia z bazą danych oraz konfigurację pliku .env.

Wygląd pliku .env:

MISTRAL_API_KEY=

DATABASE_HOST=

DATABASE_PORT=

DATABASE_USER=

DATABASE_USER_PASSWORD=

DATABASE_NAME=

JWT_SECRET_WORD=

2.4 Uruchomienie systemu

Aby poprawnie uruchomić aplikację, wykonaj poniższe kroki:

1. Uruchomienie serwera bazy danych
    - Sprawdź, czy Microsoft SQL Server jest uruchomiony.
    - Zweryfikuj, czy baza danych działa na odpowiednim porcie
2. Wypełnienie pliku .env
    - Upewnij się, że plik .env zawiera poprawne dane dostępu do bazy oraz klucz API.
3. Załadowanie zmiennych środowiskowych
    - Przed uruchomieniem aplikacji załaduj zmienne z pliku .env, na przykład za pomocą komendy source path/to/.env
4. Instalacje zależności
    - W katalogu projektu uruchom: npm install
5. Uruchomienie aplikacji:
    - Aplikację można uruchomić w katalogu projektu za pomocą ts-node: npx ts-node src/main.ts
6. Sprawdzenie działania aplikacji
    - Po uruchomieniu aplikacja powinna być dostępna pod adresem: http://localhost:3000

Architektura aplikacji

Aplikacja opiera się na wzorcu **Model-View-Controller (MVC)**, co zapewnia logiczny podział komponentów:

- **Model (Entity)** – reprezentuje strukturę danych w bazie MSSQL, np. użytkownicy, wiadomości, czaty
- **Controller** – obsługuje żądania HTTP i przekazuje je do odpowiednich usług
- **Service** – zawiera logikę biznesową i zarządza komunikacją z bazą danych

Modularna budowa aplikacji pozwala na łatwe rozszerzanie funkcjonalności. Każda kluczowa funkcja została podzielona na dedykowane moduły, w tym:

- **UserModule** – zarządzanie użytkownikami
- **ChatModule** – obsługa czatów
- **MessageModule** – obsługa wiadomości
- **MistralModule** – integracja z Mistral AI

Struktura tabel w bazie danych

Baza danych zawiera trzy główne tabele odpowiadające modułom aplikacji:

1. Tabela User

Przechowuje informacje o użytkownikach.
Kolumny:

- id (int, PK, auto-increment) – unikalny identyfikator użytkownika
- username (nvarchar, unique) – nazwa użytkownika
- password (nvarchar) – zaszyfrowane hasło użytkownika

Relacje:

- User posiada wiele Chat (relacja 1:N)

2. Tabela Chat

Reprezentuje konwersacje między użytkownikami.
Kolumny:

- id (int, PK, auto-increment) – unikalny identyfikator czatu
- userId (int, FK) – identyfikator użytkownika tworzącego czat

Relacje:

- Chat posiada wiele Message (relacja 1:N)

- Chat jest powiązany z User (relacja N:1)

Podstawowe funkcjonalności

- **Rejestracja i logowanie użytkowników** (UserModule)
- **Obsługa wiadomości** – użytkownicy mogą wysyłać wiadomości (MessageModule)
- **Integracja z AI** – aplikacja korzysta z Mistral AI do generowania odpowiedzi (MistralModule)
- **Zarządzanie czatami** – system obsługuje rozmowy użytkowników (ChatModule)
- **Bezpieczeństwo** – szyfrowanie haseł, autoryzacja za pomocą JWT

Opis modułów

Moduł User

Moduł **User** odpowiada za zarządzanie użytkownikami w systemie. Wspiera operacje takie jak tworzenie użytkowników, wyszukiwanie użytkowników po nazwie oraz identyfikatorze, a także przechowywanie informacji o użytkownikach w bazie danych.

Komponenty Modułu User

1. **User Controller**:
    - Obsługuje żądania HTTP związane z użytkownikami, np. rejestrację, edytowanie danych użytkownika.
2. **User Entity**:
    - Definiuje strukturę tabeli użytkowników w bazie danych. Zawiera atrybuty takie jak: identyfikator, nazwa użytkownika, hasło, relacja z tabelą **Chat**.
3. **User Module**:
    - Grupa wszystkich komponentów związanych z użytkownikami. Importuje **TypeOrmModule** do pracy z bazą danych.
4. **User Service**:
    - Zawiera logikę biznesową dla operacji na użytkownikach, takich jak tworzenie użytkowników, wyszukiwanie ich po nazwie i ID.

Funkcjonalności

- Tworzenie nowych użytkowników.
- Wyszukiwanie użytkowników po nazwie lub ID.
- Relacja z czatami utworzonymi przez użytkowników.

Moduł Auth

Moduł **Auth** zapewnia mechanizm logowania oraz rejestracji użytkowników. Wykorzystuje tokeny JWT do autentykacji i ochrony dostępu do chronionych zasobów w systemie.

Komponenty Modułu Auth

1. **Auth Controller**:
    - Obsługuje żądania HTTP związane z logowaniem i rejestracją użytkowników.
2. **Auth Service**:
    - Odpowiada za logikę rejestracji, logowania oraz weryfikacji użytkowników.
3. **JwtAuthGuard**:
    - Chroni dostęp do chronionych tras, sprawdzając poprawność tokenu JWT.
4. **Jwt Strategy**:
    - Definiuje strategię używaną do weryfikacji tokenu JWT.

Funkcjonalności

- Rejestracja użytkowników.
- Logowanie użytkowników i generowanie tokenów JWT.
- Walidacja tokenów JWT w celu zapewnienia bezpiecznego dostępu do zasobów.

Moduł Message

Moduł **Message** odpowiada za zarządzanie wiadomościami w systemie. Umożliwia użytkownikom wysyłanie i odbieranie wiadomości w ramach czatów. Zawiera także interakcje z botem, który odpowiada na wiadomości użytkowników.

Komponenty Modułu Message

1. **Message Controller**:
    - Obsługuje żądania HTTP związane z tworzeniem nowych wiadomości.
2. **Message Entity**:
    - Zawiera strukturę tabeli wiadomości w bazie danych. Przechowuje informacje o treści wiadomości, identyfikatorze czatu oraz autorze wiadomości.
3. **Message Service**:
    - Odpowiada za logikę tworzenia nowych wiadomości, w tym komunikację z botem, który odpowiada na wiadomości użytkowników.

Funkcjonalności

- Tworzenie wiadomości przez użytkowników.
- Automatyczne odpowiedzi bota na wiadomości użytkowników.
- Powiązanie wiadomości z odpowiednimi czatami.

Moduł Mistral

Moduł **Mistral** integruje aplikację z API Mistral, które umożliwia generowanie odpowiedzi bota na wiadomości użytkowników. Moduł wykorzystuje specjalistyczną usługę do komunikacji z API.

Komponenty Modułu Mistral

1. **Mistral Controller**:
    - Obsługuje żądania związane z komunikacją z API Mistral. Może zostać użyty do testowania odpowiedzi generowanych przez API.
2. **Mistral Service**:
    - Komunikuje się z API Mistral i zarządza generowaniem odpowiedzi na podstawie historii wiadomości.
3. **MistralModule**:

- Moduł, który grupuje **MistralController** i **MistralService** w jeden logiczny blok. Dzięki niemu możliwe jest łatwe zarządzanie tymi komponentami oraz ich zależnościami.

Funkcjonalności

- Wysyłanie zapytań do API Mistral w celu uzyskania odpowiedzi na podstawie rozmowy.
- Obsługuje mechanizm ponawiania zapytań w przypadku problemów z dostępem do API (np. limit przekroczenia).

Przykłady API

Rejestracja użytkownika

- **Endpoint**: POST /auth/register
- **Parametry**:
    - username: Nazwa użytkownika
    - password: Hasło użytkownika
- **Przykładowa odpowiedź**:
    - Wiadomość zwrotna: „User successfully registered” (Użytkownik został pomyślnie zarejestrowany).

Logowanie

- **Endpoint**: POST /auth/login
- **Parametry**:
    - username: Nazwa użytkownika
    - password: Hasło użytkownika
- **Przykładowa odpowiedź**:
    - Token JWT: zwrócony token autoryzacyjny użytkownika w formie ciągu znaków (np. „jwt_token”).

Wysłanie wiadomości

- **Endpoint**: POST /message/send
- **Parametry**:
    - content: Treść wiadomości, którą użytkownik chce wysłać
    - chatId: Identyfikator czatu, do którego wiadomość jest przypisana
- **Przykładowa odpowiedź**:
    - Wiadomość zwrotna: „Message sent successfully” (Wiadomość została pomyślnie wysłana).

Odpowiedź bota (Mistral AI)

- **Endpoint**: POST /mistral/respond
- **Parametry**:
    - message: Treść wiadomości użytkownika, na którą bot ma odpowiedzieć
- **Przykładowa odpowiedź**:

- Odpowiedź bota: „Hello! How can I assist you today?” (Witaj! Jak mogę Ci pomóc dzisiaj?).

Zależności Pomiędzy Modułami

Moduły w aplikacji są ze sobą powiązane w sposób umożliwiający pełną interakcję między użytkownikami, wiadomościami, czatami oraz odpowiedziami generowanymi przez bota. Moduł **User** współpracuje z modułami **Auth** i **Message**, zapewniając dostęp do danych użytkowników i ich wiadomości. Moduł **Message** łączy się z modułem **Mistral**, aby dostarczyć automatyczne odpowiedzi na wiadomości użytkowników.

Dodatkowe uwagi

1. **Bezpieczeństwo**: System wykorzystuje mechanizmy szyfrowania haseł (np. bcrypt) i autoryzacji opartą na JWT, zapewniając bezpieczne przechowywanie danych użytkowników oraz komunikację z API.
2. **Obsługa błędów**: Aplikacja posiada odpowiednie mechanizmy obsługi błędów, takie jak walidacja danych wejściowych, autoryzacja tokenów JWT oraz obsługa wyjątków przy zapytaniach do API Mistral.
3. **Logowanie**: Zastosowano mechanizm logowania błędów i aktywności użytkowników, co pozwala na monitorowanie stanu aplikacji oraz wykrywanie ewentualnych problemów.