

**Wydział fizyki i informatyki stosowanej**

Przedmiot: Zaawansowane Technologie Internetowe

Temat: Komunikator internetowy

Autor: *Bartłomiej Leśnicki*

Kraków, 2023

# Cel projektu.

Celem projektu było zaprojektowanie oraz implementacja aplikacji typu klient serwer realizującej zaawansowane technologie internetowe. Należało wytworzyć oprogramowanie zgodnie z wzorcem RESTful z wykorzystaniem technologii WebSocket. Cześć serwerowa miła być napisana w języku Java wykorzystując technologie platformy Jakarta EE lub dodatkowo z wykorzystaniem technologii Spring i technologii Spring Boot. Aplikacja kliencka opracowana z wykorzystaniem serwisu WWW i języka JavaScript, aplikacja typu SPA.

Aplikacja po stronie serwera została napisana w języku Java z użyciem frameworka Spring Boot. Nadrzędny celem projektu było przedstawienie zaawansowanych technologii internetowych, został więc użyty protokół http oraz WebSocket.

Aplikacja Kliencka została napisana w języku JavaScript z użyciem frameworka React.js. Integrowana jest z serwerem za pomocą protokołu http oraz protokołu WebSocket.

# Temat projektu

Tematem projektu jest komunikator internetowy. Komunikator internetowy to aplikacja umożliwiająca użytkownikom komunikację w czasie rzeczywistym za pomocą Internetu. Projekt oparty jest na technologiach Java Spring Boot, React, WebSocket, HTTP i PostgresSQL. Celem projektu jest stworzenie platformy, która umożliwi użytkownikom swobodną wymianę wiadomości, prowadzenie rozmów grupowych i zapewni bezpieczne przechowywanie danych.

# Architektura aplikacji

Aplikacja została napisana w stylu RESTful wykorzystując wzorzec architektoniczny MVC( model-view-controller). Projekt podzielony jest na 3 części:

* Aplikacje Serwerową
* Aplikacje Kliencką
* Bazę Danych

Architekturę aplikacji można przedstawić za pomocą diagramu przepływu danych(rysunek 1). Użytkownik porozumiewa się z aplikacją kliencką. Użytkownik, może się zarejestrować lub zalogować do aplikacji (http) a następnie wysyłać wiadomości (webSocket) w formie postów do odpowiednich chatroom’ów. Aplikacja kliencka przesyła dane za pomocą odpowiedniego protokołu do serwera. Serwer wykonuje operacje na odebranych danych z użyciem Bazy danych. Po wykonaniu logiki biznesowej na danych, informacje przekazywane są ponownie do aplikacji klienckiej oraz wizualizowane użytkownikowi.

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

Rysunek Diagram przepływu danych

# Serwer

W ramach projektu został zaprojektowany oraz zaimplementowany serwer aplikacyjny obsługujący bazę danych oraz udostępniający API, z którym łączy się klient. Serwer został napisany w języku Java w wersji 17 z użyciem frameworka Spring Boot oraz biblioteki Jackson do serializacji. Do kompilacji użyte zostało openJDK firmy Oracle w wersji 19.0.2 (rysunek2).

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rysunek wersja JDK

Moduły Spring Boota oraz dodatkowe biblioteki wraz z ich wersjami zostały przedstawione na Listingu 1.

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-websocket</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

<version>3.0.4</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

<version>20230227</version>

</dependency>

</dependencies>

Struktura klas aplikacji została podzielona na jeden główny pakiet „pl.Komunikator” zawierający klasę Main - główny punkt wejścia programu oraz sześć pozostałych pakietów. Są to pakiety chatroom, post, user, znajomi zawierające klasy odpowiedzialne za:

* Reprezentacje encji bazy danych
* Obsługę zapytań do bazy danych (klasy z sufixem Repository z adnotacją @Repository)
* Obsługę logiki biznesowej przetwarzanych danych (klasy z sufixem Service z adnotacją @Service)
* Obsługę endpointów http (klasy z sufixem Controller z adnotacją @Controller)

Oraz w przpadku pakietów chatroom oraz user klasy ChatroomDeserializer oraz UserDeserializer odpowiedzialne za ręczną obsługę procesu serializacji oraz deserializacji. Ostatnimi pakietami są pakiet WebSocket i pakiet logger. Pakiet WebSocket zawiera klasy odpowiedzialne za obsługę konfiguracji oraz kontroli komunikacji protokołu WebSocket. Pakiet logger dostarcza klasę realizującą logowanie informacji do pliku za pomocą programowania aspektowego(Spring AOP). Logowane są zawarte przyjaźnie między użytkownikami. Plik z tymi danymi to „dodaniZnajomi.txt” znajdujący się w katalogu „app” w kontenerze backend. Pełen wykaz klas znajduje się na diagramie klas przedstawionym na rysunku 3. Zależności między klasami przedstawione są na rysunku 4. Dokumentacja poszczególnych klas oraz funkcji znajduje się w dostarczonej dokumentacji javaDoc w katalogu „ZTI\_Projekt\backend\dokumentacja-JavaDoc” oraz w dostarczonym kodzie źródłowym. Endpointy http znajdują się na http://localhost:8080/.

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

Rysunek Lista pakietów wraz z klasami

Obraz zawierający tekst, zrzut ekranu, diagram

Opis wygenerowany automatycznie

Rysunek Diagram klas aplikacji serwerowej

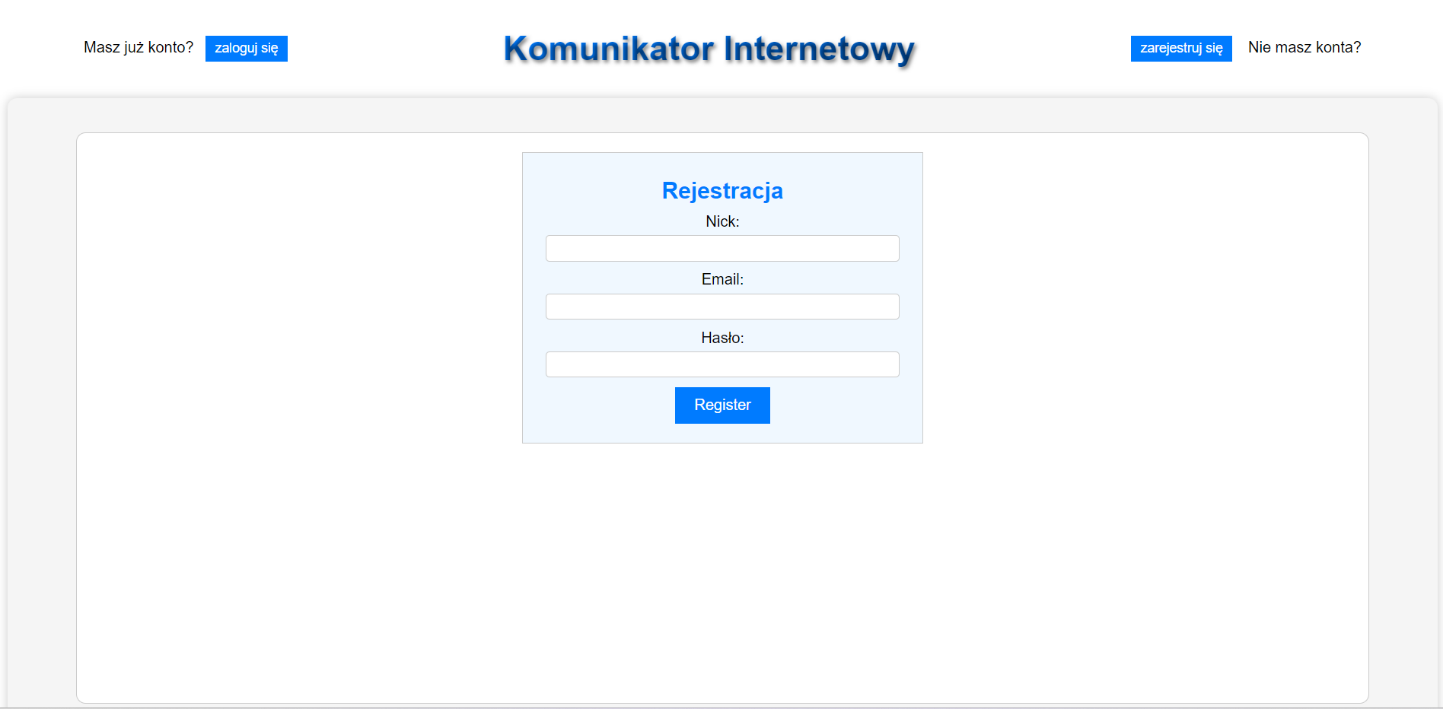
# Klient

Aplikacja kliencka została napisana w języku JavaScript z użyciem frameworka React.js. Aplikacja obsługuje endpointy http udostępnione przez backend, oraz łączy się protokołem WebSockets z backendem w celu wymiany postów. Aplikacja dostępna jest pod adresem <http://localhost:3000/> Technologie wykorzystane w aplikacji klienckiej:

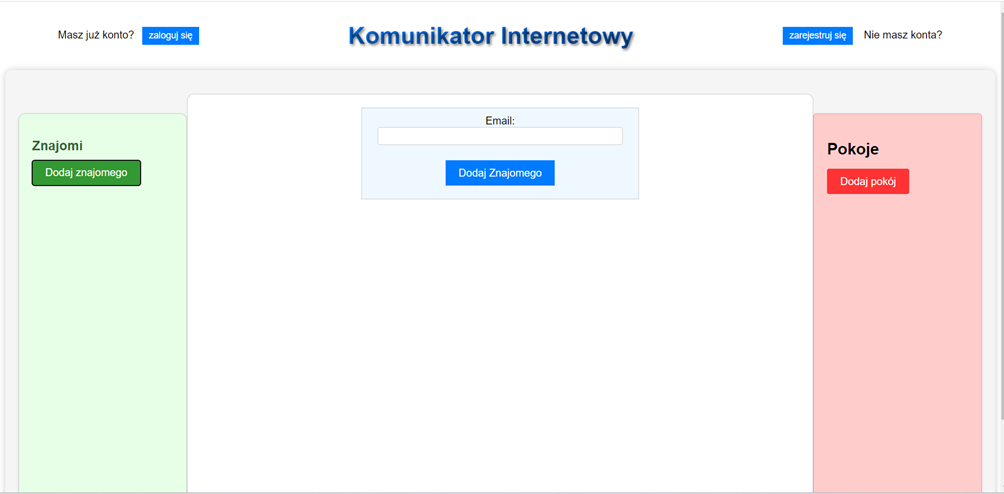
* JavaScript,
* React,
* useState,
* useEffect,
* axios,
* SockJS,
* Stomp.

Program udostępnia następujące funkcjonalności:

* Logowanie oraz rejestracje obsługiwane protokołem http. Email jest odpowiednio parsowany i sprawdzany w przypadku błędnego formatu wyświetlany jest odpowiedni komunikat. Wpisywanie hasła jest tajne. Podczas logowania w przypadku wpisania błędnego hasła lub emailu zostanie wyświetlony odpowiedni komunikat. Logowanie odbywa się za pomocą mechanizmu sesji. Dane dotyczące danej sesji przechowywane są w przeglądarce. Po odświeżeniu strony użytkownik nie zostaje wylogowany lecz wraca do strony głównej(rysunek 8). Otwarcie strony w nowej karcie przechowuje sesje tego samego użytkownika. W celu wylogowania się należy nacisnąć przycisk zaloguj się lub zarejestruj się. Użytkownik zostanie przeniesiony do strony z wyborem między wylogowaniem się a powrotem do strony głównej. Po naciśnięciu przycisku „Wyloguj” użytkownik zostaje wylogowany oraz przeniesiony do strony logowania a sesja zostaje zakończona. Użytkownik w jednym momencie może zalogować się na to samo konto na kilku przeglądarkach/komputerach. Sesja jest realizowana tylko po stronie klienta. Backend realizuje akcje dla danych użytkowników za pomocą odpowiednich endpointów oraz messageBrokera.
* Dodawanie znajomych protokołem http. W lewej części interfejsu graficznego znajduje się lista znajomych w raz z przyciskiem „Dodaj Znajomego”. Po naciśnięciu przycisku wyświetlany zostaje formularz dodania znajomego w centralnej części interfejsu użytkownika (Rysunek 6). Obok nicków znajomych znajduje się przycisk „usuń” do usuwania znajomych. Po usunięciu znajomego usuwana jest relacja między znajomymi i w przypadku usunięcia wszystkich znajomych z chatromu usuwany jest ten chatroom.



Rysunek Strona rejestracji



Rysunek Strona dodawania znajomych

* Dodawanie Chatroomów protokołem http. W prawej części interfejsu graficznego znajduje się lista chatroomów w których uczestniczy dany użytkownik (Rysunek 7). Dodawanie pokoi odbywa się poprzez nadanie nazwy, wybranie uczestników z listy przyjaciół za pomocą checkboxa i potwierdzenie przyciskiem „DodajChatroom”.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, design

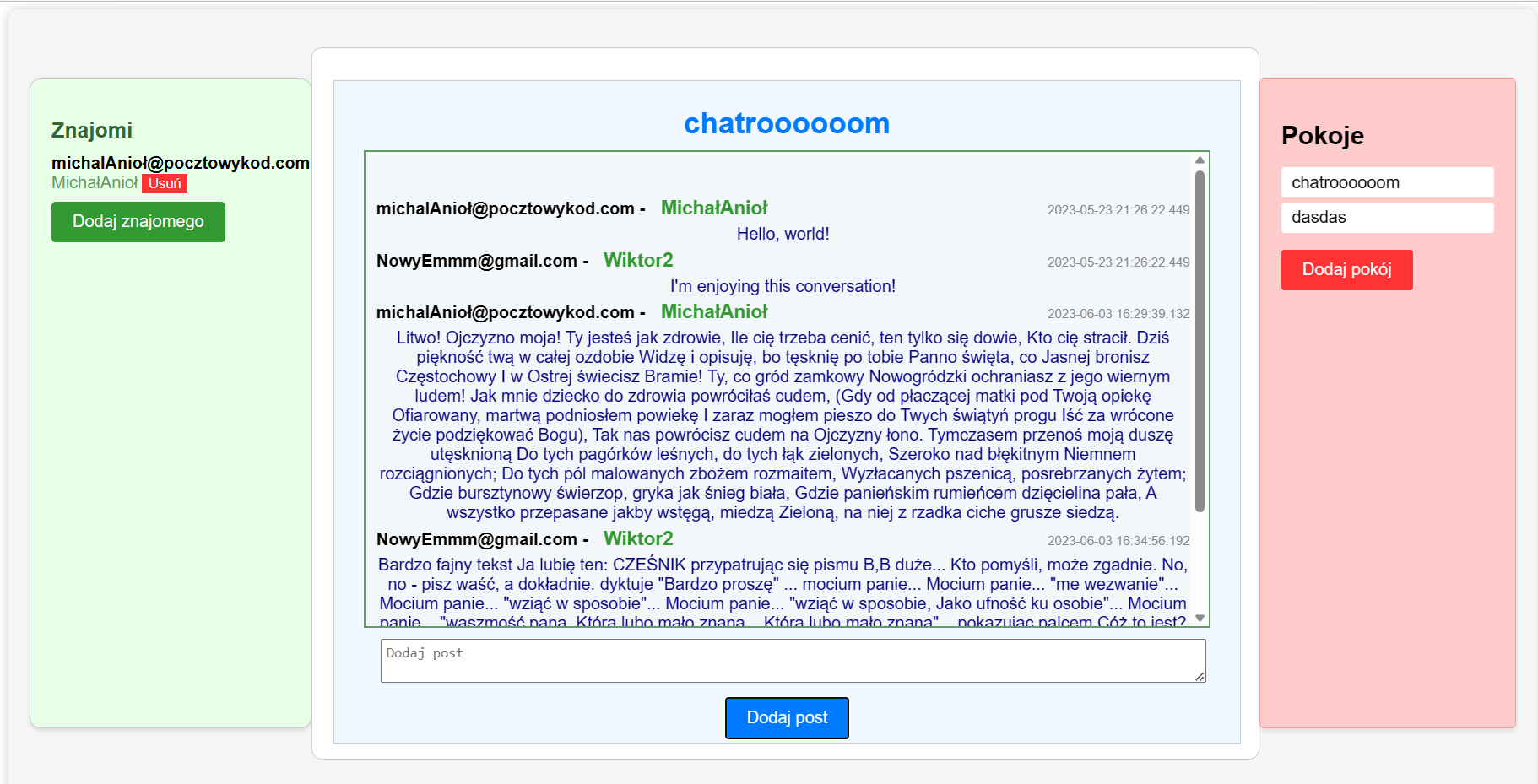
Opis wygenerowany automatycznie

Rysunek Strona Dodawania pokoju



Rysunek Strona główna

* Wysyłanie Postów protokołem WebSocket oraz http. Aby wysyłać posty należy wybrać jeden z chatroomów. Po naciśnięciu nazwy chatroomu z listy po prawej stronie interfejsu graficznego pojawia się podstrona służąca do wysyłania postów(Rysunek 9). Środkowa część podstrony zawiera nazwę pokoju, okno wyświetlające posty (email i nick autora, datę dodania postu oraz jego treść), pole do wpisywania treści postu oraz przycisk do wysyłania postów- „Dodaj post”. Po naciśnięciu na nazwę pokoju posty ładowane są za pomocą protokołu http, a następnie aktualizowane oraz wysyłane protokołem WebSocket. Maksymalna długość postu wynosi 10000 znaków. Postów nie można bezpośrednio edytować przy użyciu aplikacji klienckiej.



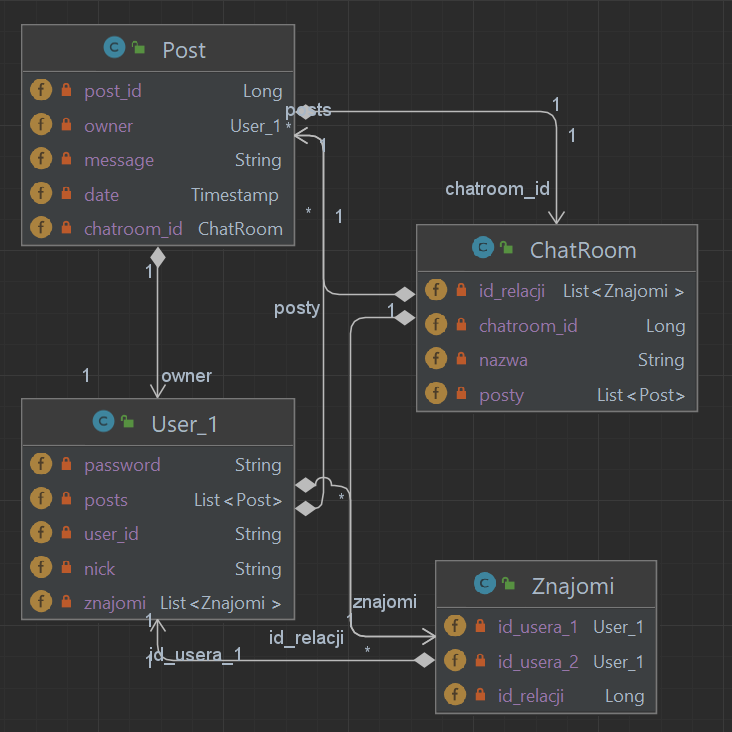
Rysunek Strona do wysyłania postów

# Baza Danych

W projekcie została użyta relacyjna baza danych oparta o system zarządzania bazą danych PostgreSQL. Baza danych znajduje się w chmurze. Realizowana jest przez system database as a service – ElephantSQL. Zawiera 4 tabele:

* User\_1
* Znajomi
* Chatroom
* Post

Relacje między tabelami przedstawione są na Rysunku 10. Aplikacja serwerowa łączy się z bazą danych za pomocą SpringBoot’a. konfiguracja połączenia znajduje się w pliku src/main/resources/application.properties.



Rysunek Schemat Bazy danych

# Informacja uruchomieniowa

Projekt znajduje się na githubie : <https://github.com/PanBartlomiej/ZTI_projekt>. Wraz z projektem został dostarczony docker-compose.yaml konfigurujący oraz uruchomiający kontenery z aplikacjami. W celu uruchomienia należy wpisać komendę docker-compose up w głównym katalogu projektu gdzie znajduje się plik docker-compose.yaml (katalog ZTI\_Projekt). Aplikacja kliencka znajduje się na <http://localhost:3000/>.

# Wnioski

Technologie internetowe odgrywają kluczową rolę we współczesnym świecie, umożliwiając globalne połączenie, wymianę informacji i dostęp do ogromnego zasobu wiedzy. Przez nieustanny rozwój protokołów i narzędzi, takich jak HTTP, WebSocket, HTML, CSS i JavaScript, możliwe stało się tworzenie zaawansowanych aplikacji internetowych, interaktywnych stron internetowych i mobilnych aplikacji. Technologie internetowe umożliwiają także rozwój e-commerce, społeczności online, usług streamingowych i wiele innych innowacyjnych rozwiązań, które wpływają na nasze życie codzienne, zarówno w obszarze pracy, jak i rozrywki. W ramach rozwoju technologii internetowych, framework Spring Boot odegrał istotną rolę, umożliwiając programistom szybkie i efektywne tworzenie zaawansowanych aplikacji webowych opartych na platformie Java. Spring Boot zapewnia wiele gotowych modułów i narzędzi, które usprawniają proces projektowania, konfiguracji i uruchamiania aplikacji, co przyspiesza czas tworzenia i wdrażania rozwiązań internetowych. Technologia WebSocket jest szczególnie przydatna w aplikacjach wymagających komunikacji w czasie rzeczywistym, takich jak czat online, aplikacje do udostępniania treści na żywo, systemy monitoringu itp. Dzięki WebSocket możliwe jest natychmiastowe przesyłanie danych w obu kierunkach, co pozwala na dynamiczną aktualizację zawartości strony bez konieczności odświeżania. Technologia ta wydajnie realizuje wymianę postów miedzy użytkownikami, bez zbędnych odświerzań czy pullingów. Wykorzystanie WebSocket w połączeniu z innymi technologiami internetowymi, takimi jak HTTP i bazy danych, pozwala na budowę kompleksowych i wydajnych aplikacji webowych, które oferują zarówno funkcjonalność żądanie-odpowiedź, jak i komunikację w czasie rzeczywistym co można zaobserwować w omawianym projekcie komunikator internetowy.