29.06.2019, Poznań, CDV

Czat

Nowoczesne metody przetwarzania danych
Projekt zaliczeniowy

Wykonali:
Filip Baumgart, 22412,
Jakub Dolata, 22419

Opis projektu:

Implementacja komunikatora w czasie rzeczywistym. Docelowo zastosowanie tego czatu to użycie go jako komponent większej aplikacji, np. gry sieciowe, live-chaty, shoutboxy na stronach i w aplikacjach internetowych, gdzie nie ma konieczności zachowania trwałości danych. W celu prezentacji projektu stworzono klienta - prosty czat podzielony na pokoje tematyczne. Użytkownik po rejestracji uzyskuje dostęp do predefiniowanych pokoi, gdzie może dołączyć do rozmowy z innymi użytkownikami.

Użyte technologie:

Front-end:

- HTML,
- CSS/SASS,
- Bootstrap,
- JavaScript

Back-end:

- Python, Flask,
- JavaScript, Node.js, Express,
- klienci Redisa (redis-py, node-redis)

Baza danych:

- Redis

Model danych:

Rolę bazy danych w naszym projekcie pełni Redis. Jest to rozwiązanie open-source z nurtu NoSQL przechowujące dane w pamięci w postaci par klucz - wartość.

Dlaczego Redis?

Zastosowaliśmy Redis w naszym projekcie ze względu na:

- szybkość działania - Redis potrafi zapisywać i odczytywać informacje w bardzo szybkim tempie - szczególnie należy tu położyć nacisk na operacje zapisu, gdzie wydajność może sięgać ok. 110000 operacji SET na sekundę (odczyt to ok. 81000 operacji GET na sekundę),
- dostępne typy danych - w naszym projekcie wykorzystaliśmy:
    - stream - strumień zastosowano do przechowywania wysłanych wiadomości oraz do odczytu wiadomości dla danego pokoju - ten typ danych doskonale nadaje się do przechowywania wiadomości ze względu na swoją strukturę: każdy nowy wpis jest złożony z jednego lub wielu pól, do których przyporządkowana jest wartość, w tym przypadku jest to nazwa użytkownika, treść wiadomości oraz unikalny identyfikator wiadomości,
    - hash - np. do przechowywania danych użytkowników - hash umożliwia reprezentację danych w formie podobnej do obiektu, jest to mapa między kluczami, a wartościami,
    - sorted set - zbiór posortowany zastosowano do stworzenia listy najbardziej aktywnych użytkowników, czyli takich, którzy napisali najwięcej wiadomości - zbiór posortowany to zbiór unikalnych wartości, do każdej z których przyporządkowany jest określony wynik stanowiący o pozycji danego elementu w zbiorze,
    - string - podstawowy typ danych w Redisie umożliwiający szybki dostęp do wartości na zasadzie klucz - wartość - w projekcie zastosowany m.in. do: przechowywania ID następnego użytkownika
- atomowość operacji - w przypadku, gdy dwóch konkurencyjnych klientów stara się o dostęp do serwera to otrzymają oni aktualne dane

Opis implementacji:

Schemat bazy danych:

Poniżej przedstawiono tabelę ze schematem bazy danych i z wykorzystanymi kluczami:

Nazwa klucza	Wartość	Typ danych	Zastosowanie	Przykład wywołania

user:<userid>	username <username>
	password <password>
	auth <auth>
		Hash	Przechowywanie danych użytkowników: nazwa użytkownika, hasło oraz token nadawany przy rejestracji do obsługi sesji	`hgetall user:1`
				
				`>1) "username"`
				`>2) "admin"`
				`>3) "password"`

				>4) “admin” >5) “auth” >6)”1q2w3e4r5”
leaderboard	<username> <score>	Sorted set	Przechowywanie informacji o ilości wiadomości wysłanych przez każdego użytkownika	zrevrange leaderboard 0 -1 WITHSCORES >1) “admin” >2) “5” >3) “user1” >4) “3”
auths	<auth> <userid>	Hash	Secondary index - Przechowywanie tokenów przyporządkowanych do użytkowników	hgetall auths > ”2qw34e5r3w” > “1” > ”5ewasfe25x” > “2”
<username>	<userid>	String	Secondary index - Przechowywanie identyfikatora użytkownika przyporządkowanego do jego nazwy	get admin > “1”
next_user_id	<userid>	String	Przechowywanie identyfikatora użytkownika, który zostanie utworzony jako następny	get next_user_id > “2”
room: <roomname>	<messageid> username <username> message <message>	Stream	Przechowywanie wiadomości z określonego pokoju	xread STREAMS room:general 0-0 >1) 1) >"room:general>" > 2) 1) 1) >"1560968578227>7-0" > 2) >1) "username"

				> >2) "admin" > >3) "message" > >4) "Hello, >World!"

Opis zaimplementowanych metod:

checkUser():
Funkcja sprawdza czy użytkownik istnieje i czy podał prawidłowe hasło.
Pobiera nazwę użytkownika oraz hasło, a następnie sprawdza czy użytkownik istnieje i pobiera z bazy danych identyfikator, token oraz hasło użytkownika. Wykonuje walidację hasła i zwraca w odpowiedzi token lub error jeśli hasło jest niepoprawne lub użytkownik nie istnieje.

getToken():
Funkcja służy do sprawdzania czy użytkownik jest zalogowany.
Pobiera token zapisany w session storage przeglądarki, po czym sprawdza czy istnieje taki użytkownik w bazie. Jeśli użytkownik istnieje to zwraca kod 200, wraz z nazwą użytkownika, a jeśli nie istnieje - zwraca błąd.

newUser():
Dodaje nowego użytkownika do bazy danych.
Funkcja pobiera nazwę użytkownika oraz hasło, które przeszło już walidację po stronie front-end. Inkrementuje klucz next_user_id aby przypisać do nowego użytkownika poprawny identyfikator. Następnie generuje losowy token i zapisuje w bazie danych: nazwę użytkownika, hasło oraz unikalny token. Dodaje również do klucza auths token powiązany z identyfikatorem danego użytkownika. Funkcja zwraca status 200 lub błąd.

sendMessage() :
Funkcja pobiera parametry: nazwę użytkownika, treść wiadomości oraz nazwę pokoju, a następnie za pomocą klienta Redisa zapisuje w określonym strumieniu nową wiadomość. Parametry są pobierane z front-endu za pomocą metody POST protokołu HTTP. Funkcja zwraca 1 jeśli umieszczenie wiadomości w strumieniu się powiodło.

readLastMessage(string *roomname*):
Funkcja pobiera parametr roomname za pomocą metody GET, a następnie odczytuje ostatnią wiadomość z właściwego strumienia.
Funkcja zwraca obiekt JSON, w którym zapisana jest nazwa użytkownika i treść wiadomości.

readAllMessages(string *roomname*):
Funkcja pobiera parametr *roomname* za pomocą metody GET, a następnie odczytuje wszystkie wiadomości z właściwego strumienia.
Funkcja zwraca obiekt JSON, w którym zapisana jest lista obiektów z nazwą użytkownika i treścią wiadomości.

displayLeaderboard():

Funkcja za pomocą klienta Redisa pobiera dane z klucza o nazwie *leaderboard*, gdzie składowane są nazwy użytkowników z przyporządkowanymi do nich wynikami, tj. liczbą wysłanych wiadomości. Dane są pobierane w porządku malejącym według ilości wysłanych wiadomości. Funkcja zwraca obiekt JSON z listą użytkowników i wynikami.

dumpDatabase():

Funkcja wywołuje poprzez klienta Redisa polecenie BGSAVE, które zapisuje w tle zrzut bazy danych do pliku dump.rdb