Przedmiot: Zaawansowane programowanie w języku JAVA

RAPORT

Filip Jeżak *jezakfilip@gmail.com*
Dominik Seweryn *domsew@gmail.com*

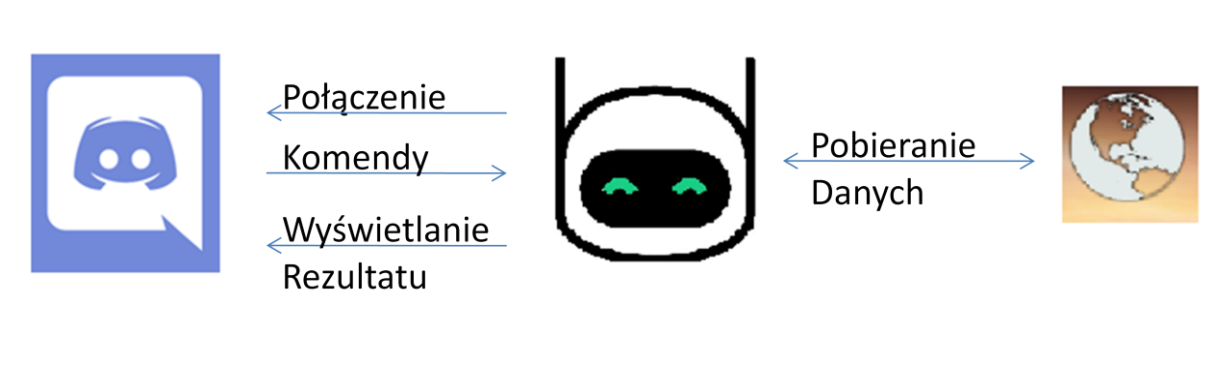
GitHub: https://github.com/fjezak/Discord-E-sim-Bot

E6C1S4

Spis treści

Wstęp

Celem naszego projektu było zaprojektowanie aplikacji konsolowej łączącej się z serwisem komunikacyjnym Discord, która pobiera dane z innej witryny (serwera gry), przetwarza je oraz zwraca rezultat na czacie Discorda, jako wiadomość. Nasz program udało się uruchomić na platformie Rasbery PI, co daje możliwość zarządzania aplikacją na takiej samej zasadzie, gdyby była wyhostowana na zdalnym serwerze.



1 Opis projektu

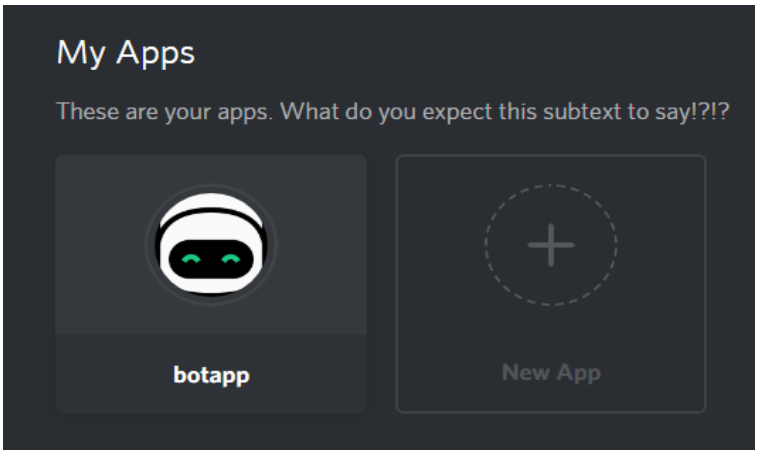
Program ma za zadanie reagować na komendy wysyłane na kanale tekstowym discorda i w zależności od rodzaju komendy wysłać na kanał wiadomość zwrotną. Wiadomość zwrotna bazuje na przetworzonych danych, które są pozyskiwane z serwerów gry E-Sim.

1.1 Zakres odpowiedzialności

- Filip Jeżak - komunikacja z Discordem (w tym przegląd dostępnych rozwiązań) oraz obsługa biblioteki do obsługi logów,
- Dominik Seweryn - komunikacja z serwerami HTTP oraz przetwarzanie pozyskiwanych danych.

2 Discord

Pierwszym krokiem do obsługi komunikatora Discord było wygenerowanie bota, możliwe to jest dzięki oficjalnemu API producenta. Po tym działaniu otrzymywaliśmy unikalny token, przez który kojarzymy bota z naszą aplikacją.



Do połączenia bota z naszym serwerem Discord, dobierania oraz wysyłania wiadomości posłużyła nam biblioteki Javy. Poniżej opisane zostaną 3 dostępne biblioteki, z której jedna została wykorzystana w naszym projekcie.

2.1 Przegląd bibliotek Javy

		Channel events			Message events			Role events			Server events					User events		Voice		
Name	Language	Create	Delete	Update	Delete	Receive	Update	Create	Delete	Update	Ban	Create	Delete	Unban	Typing	Join	Leave	Receive	Send	Multi-server
Discord4J	Java	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
JDA	Java	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Javacord	Java	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	

2.1.1 Dicord4J

Pierwszą testowaną biblioteką była Discord4J. Jest to najbardziej popularny i rozbudowany interface Javy do obsługi Discorda. Posiada dużą bazę przykładowych programów, oraz rozbudowaną społeczność co zapewnia dobre wsparcie. Jedną z zalet która wyróżnia ją na tle innych dostępnych narzędzi był spis projektów które wykorzystywały bibliotekę. Niestety jedną z wad które, pojawiły się w podczas wyboru , były problemy z obsługą Javy w wersji 9. Dlatego nie zdecydowaliśmy się na wykorzystanie jej w naszym projekcie.

https://github.com/austinv11/Discord4J

2.1.2 Javacord

Javacord jest najprostszą biblioteką z wszystkich trzech tu opisywanych. Niemniej jednak posiada ona możliwość obsługi potrzebnych nam funkcjonalności takich jak m.in. logowanie na serwer, wysyłanie i przetwarzanie wiadomości, oraz przykładowe programy do obsługi tych zdarzeń, wspiera ona także potrzebne nam Log Framework-i . Wadami które zadecydowały o nie wybraniu jej do projektu były przede wszystkim gorsze wsparcie społeczności oraz brak obsługi kanałów głosowych, co mogło stanowić problem gdy chcielibyśmy dalej rozwijać nasz projekt.

https://github.com/BtoBastian/Javacord

2.1.3 JDA

Do naszego projektu wybraliśmy bibliotekę JDA. Cechowała się ona przejrzystym tutorialem, bazą przykładów niezbędnych nam do realizacji projektu, oraz wsparciem bibliotek logów. Posiada szeroką społeczność przez co jest ciągle rozwijana. Nie napotkaliśmy problemów przy implementacji potrzebnych nam funkcji udostępnionych przez tę bibliotekę .

https://github.com/DV8FromTheWorld/JDA

Poniżej znajdują się najważniejsze funkcje których użyliśmy w naszym projekcie.

2.1.3.1 Kody

-logowanie się bota do aplikacji Discord

```
JsonReader reader = new JsonReader(new FileReader("settings.json"));
Gson gson = new Gson();
APP_SETTINGS = gson.fromJson(reader, AppSettings.class);
JDA jda = new
JDABuilder(AccountType.BOT).setToken(APP_SETTINGS.botToken).buildBlocking()
;
eSimClient = new ESimClient();
jda.addEventListener(new Jdabot());
```

-Odebranie wiadomości, rozpoznanie formatu komendy(np. „.help”) oraz wysyłanie sformatowanej wiadomości na serwer Discord.

```
public void onMessageReceived(MessageReceivedEvent event) {
    if (event.isFromType(ChannelType.PRIVATE)) {
        System.out.printf("[PM] %s: %s\n", event.getAuthor().getName(),
                event.getMessage().getContent());
    } else {
        System.out.printf("[%s][%s] %s: %s\n", event.getGuild().getName(),
                event.getTextChannel().getName(),
event.getMember().getEffectiveName(),
                event.getMessage().getContent());


        String message = event.getMessage().getContent();
        String name = event.getMember().getEffectiveName();
        MessageChannel channel = event.getChannel();
if (!message.isEmpty() &&  message.charAt(0) == '.') {
            try {
                String str = eSimClient.processCommands(message, name);
                EmbedBuilder eb = new EmbedBuilder();
                eb.setColor(Color.red);
                eb.setTitle("!!!ESIM BOT!!!");

                eb.setDescription(str);
                channel.sendMessage(eb.build()).queue();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2.2 Biblioteka do logów SLF4J

Aby móc kontrolować prace programu z poziomu konsoli, zaimplementowaliśmy Framework do obsługi logów SLF4J. Pozwala on śledzić m.in. ruch na serwerze do którego podłączony jest bot(użytkowników logujących się na serwer, wysyłane wiadomości).

```
15:21:03.622 [JDA MainWS-ReadThread] DEBUG n.d.j.core.requests.WebSocketClient - Sending Identify-packet...
15:21:03.627 [JDA MainWS-ReadThread] DEBUG n.d.j.core.requests.WebSocketClient - Got HELLO packet (OP 10). In
15:21:03.635 [JDA MainWS-ReadThread] DEBUG n.d.j.core.requests.WebSocketClient - Received a _trace for HELLO
15:21:03.877 [JDA MainWS-ReadThread] DEBUG n.d.j.core.requests.WebSocketClient - Received a _trace for READY
15:21:04.275 [JDA MainWS-ReadThread] INFO  net.dv8tion.jda.core.JDA - Finished Loading!
15:21:04.303 [JDA MainWS-ReadThread] DEBUG n.d.j.core.requests.WebSocketClient - Resending 0 cached events...
15:21:04.305 [JDA MainWS-ReadThread] DEBUG n.d.j.core.requests.WebSocketClient - Sending of cached events fin
[testserver][general] filj: .help
[testserver][general] botapp: "Dostępne komendy: .s, .licz, .link, .dmg, .today, .eq, .spec, .h, .noslap, .ho
```

3 E-Sim

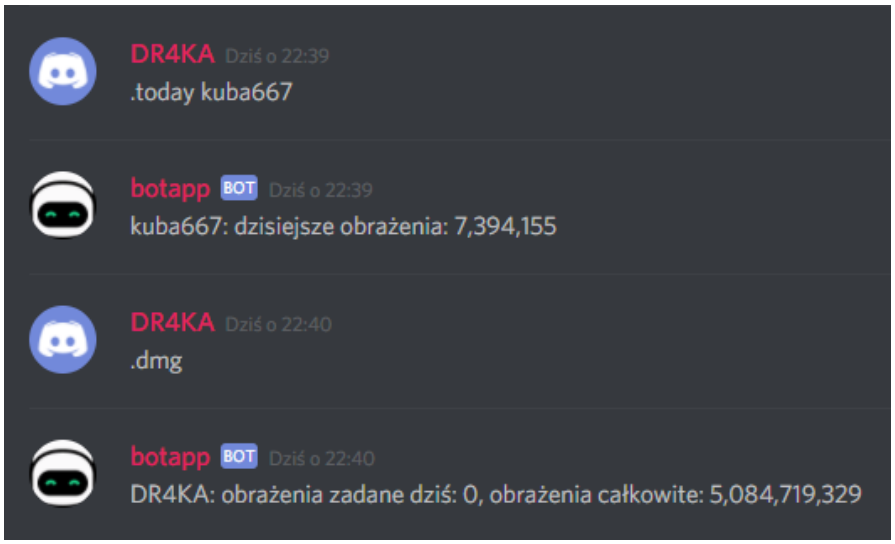
E-Sim jest strategiczną grą przeglądarkową o społeczności złożonej z około dziesięciu tysięcy aktywnych graczy. Twórcy gry udostępnili użytkownikom JSON API, dzięki któremu można pozyskać wiele przydatnych informacji.

Do komunikacji z serwerami gry wykorzystano bibliotekę Retrofit2. Biblioteka ta pozwala na wysyłanie wszelkich zapytań HTTP oraz przetwarzanie odpowiedzi z serwera. Retrofit2 bazuje na kliencie OkHttp3, co pozwala na implementację dowolnej obsługi zapytań (wraz z konfiguracją nagłówków oraz plików Cookies).

W projekcie wykorzystano również parser plików HTML o nazwie jsoup, ponieważ istniała konieczność pozyskiwania niektórych informacji bezpośrednio z zawartości gry (dane nie były dostępne w API).

4 Prezentacja programu

Poniżej zaprezentowano działanie przykładowych komend które wyświetlają informacje pobrane z serwera gry.



Podsumowanie

Zrealizowany projekt pozwala na komunikacje użytkownika z serwerem gry przeglądarkowej za pomocą popularnego serwisu komunikacyjnego. Aplikacja obsługuje kilka użytecznych komend dzięki których możliwe jest pobieranie informacji z kilku serwerów gry oraz wyświetlanie sformatowanych wiadomości zwrotnych.

Główne założenia projektu zostały zrealizowane, a aplikacja działa zgodnie z planem. Wszelkie poufne dane zostały odseparowane w osobnym pliku z ustawieniami aplikacji, który nie jest częścią projektu. Aby możliwe było uruchomienie aplikacji należy stworzyć plik o nazwie settings.json o następującej zawartości:

```
{
	"botToken": String,
	"securaLogin": String,
	"securaPassword": String
}
```

Udało się również skompilować zbudować projekt, jako osobny plik .jar, który ma wbudowane w swoją strukturę wszelkie zależności. Wynikowy plik ma rozmiar około 10 MB.

Wśród funkcjonalności, których nie udało się zrealizować wymienić można, to że nie przetestowano biblioteki Retrofit2 pod kątem łańcuchowania zapytań HTTP. Jest to ciekawa i przydatna czynność, aczkolwiek nie była w bezpośrednich założeniach projektu.