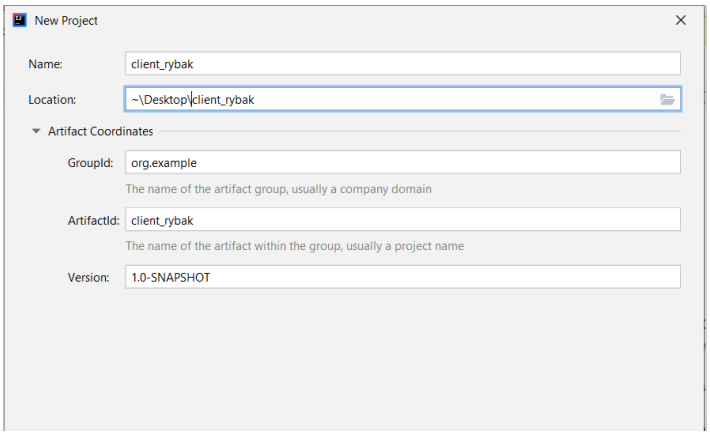
Zaawansowane Programowanie Obiektowe - **Laboratorium 1**

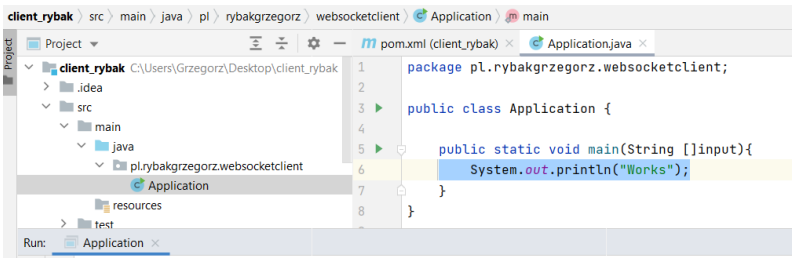
Wielowątkowość (mechanizmy komunikacji)

1. **Utwórz projekt Java w Intelij lub innym IDE**
   a. **Nazwij go: klient_[etykieta] (etykieta – dowolna nazwa)**
   b. **Projekt java w mvn**

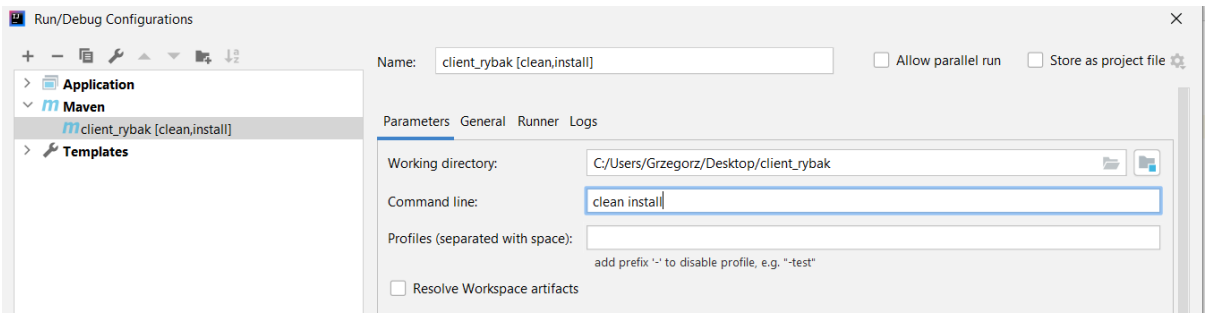


2. **Utwórz pakiet, w którym znajdzie się twój kod**
   a. **Np. : pl.rybakgrzegorz.websocketclient (oczywiście student nazywa swoim nazwiskiem lub pseudo czy nazwą firmy etc...)**
3. **Utwórz klasę: Application.java z funkcją** public static void main(String []input){}
4. **Sprawdź czy działa:**

```
System.out.println("Works");
```



5. Dodaj konfiguracje mvn (clean, install) - uruchom budowanie

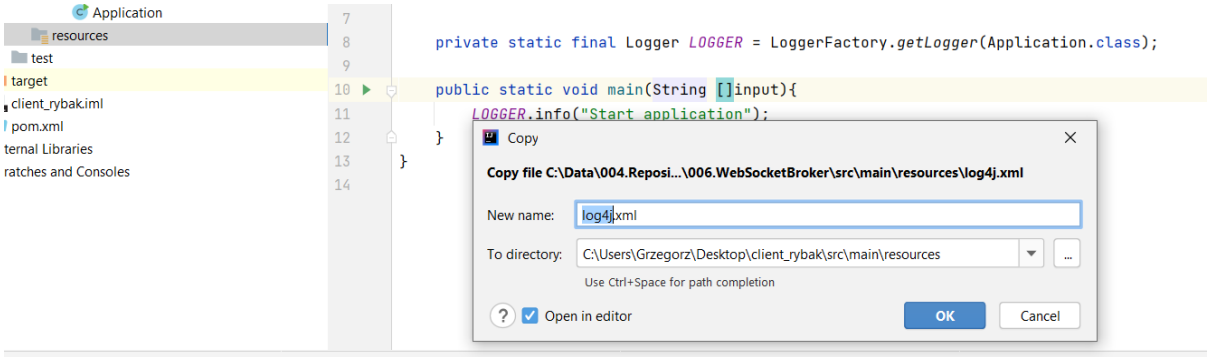


6. Do klasy dodaj logger (slf4j z implementacją log4j)
    a. `private static final Logger LOGGER = LoggerFactory.getLogger(Application.class);`
    b. W pomie dodaj zależności

```
<dependencies>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.7</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.16</version>
    </dependency>
</dependencies>
```

    c. Jeśli Ide ma “problem” to zrób: maven->reload project (prawym guzikiem myszki na projekcie w menu kontekstowym
    d. Przebuduj projekt (użyj wcześniej zrobionej konfiguracji: mvn install”)

7. Dodaj konfigurację log4j



Utwórz nowy plik log4j.xml w katalogu resources o treści:

8. Zamień sysout na log4j

```
LOGGER.info("Start application");
```

9. Utwórz nową klasę, która będzie naszym serwerem

1. `public class Server extends WebSocketServer`
2. Dodaj zależność do zewnętrznego repozytorium kodu maven

```
<repositories>
    <repository>
        <id>Central Maven repository</id>
        <name>Central Maven repository https</name>
        <url>https://repo.maven.apache.org/maven2</url>
    </repository>
</repositories>
```

3. Dodaj zależności mvn do biblioteki websocket

```
<dependency>
    <groupId>org.java-websocket</groupId>
    <artifactId>Java-WebSocket</artifactId>
    <version>1.5.1</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

4. Jak są problemy z zależnością to: maven-> reload project
5. Teraz można zrobić import w klasie Serwer do WebSocketSerwer

```
import org.java_websocket.server.WebSocketServer
```

6. Utwórz metody jakie są wymagane przez klasę abstrakcyjną WebSocketSerwer

pom.xml (client_rybak) | Application.java | Server.java | log4j.xml

```
4   import org.java_websocket.handshake.ClientHandshake;
5   import org.java_websocket.server.WebSocketServer;
6
7   public class Server extends WebSocketServer {
8
9
10      public Server(Object p0) {
11      }
12
13      @Override
14      public void onOpen(WebSocket webSocket, ClientHandshake clientHandshake) {
15
16      }
17
18      @Override
19      public void onClose(WebSocket webSocket, int i, String s, boolean b) {
20
21      }
22
23      @Override
24      public void onMessage(WebSocket webSocket, String s) {
25
26      }
27
28      @Override
29      public void onError(WebSocket webSocket, Exception e) {
30
31      }
32
33      @Override
34      public void onStart() {
35
```

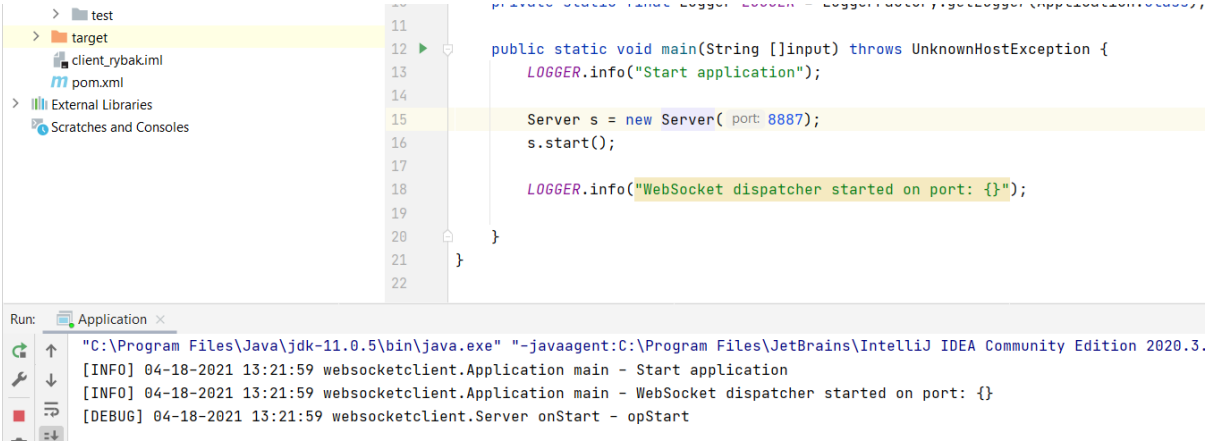
7. Doklej Logger wraz z odpowiednimi komunikatami do każdej z metod
8. Wywołaj klasę Server w klasie Application

```
Server s = new Server(8887);
s.start();
```

9. Dostosuj konstruktor (parametryzacja konstruktora)

```
public Server(int port) throws UnknownHostException {
        super(new InetSocketAddress(port));
}
```

10. Uruchom aplikację



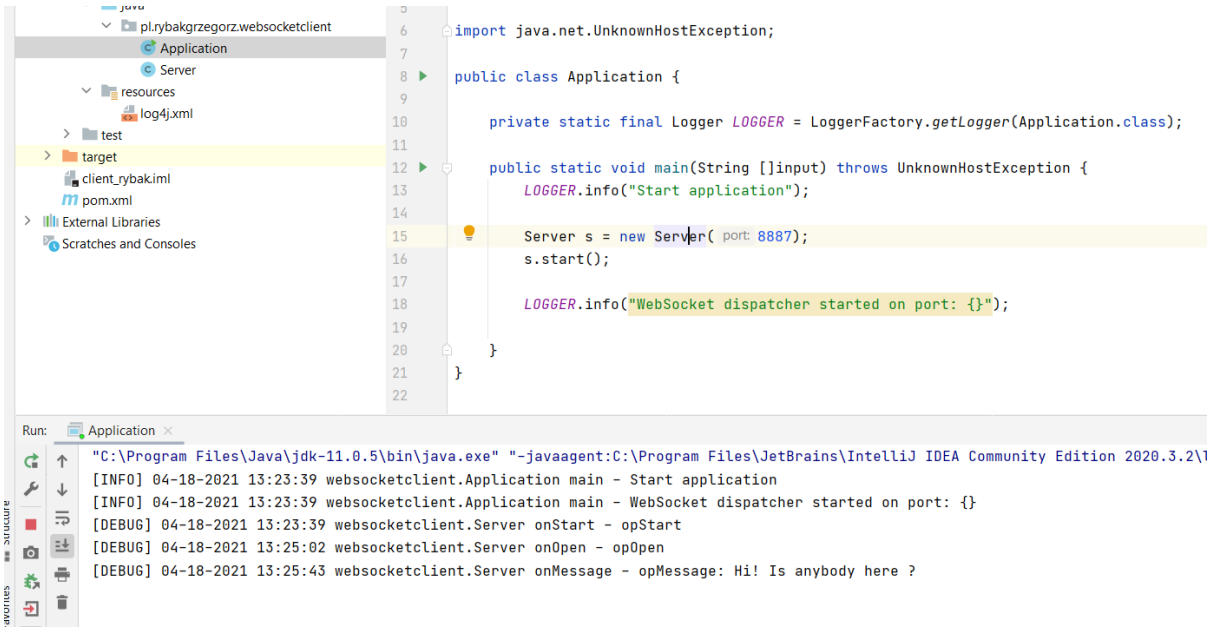
10. Wyświetl komunikat jaki przyjdzie do serwera z treścią wiadomości
11. Przetestuj działanie serwera
    a. Uruchom przeglądarkę - najlepiej Chrome
    b. Wejdź w konsolę (narzędzia developerskie)
    c. Utwórz klienta WebSocket

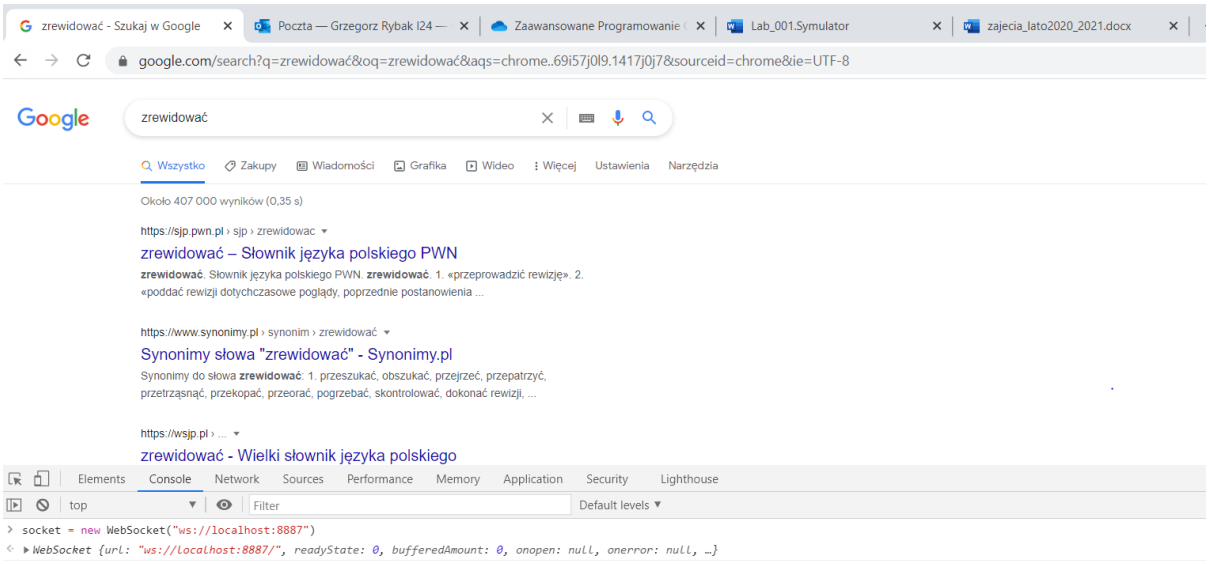
```
socket = new WebSocket("ws://localhost:8887")
```

    d. Wyślij wiadomość do uruchomionej aplikacji w JAVA

```
socket.send("Hi! Is anybody here ?")
```

    e. Sprawdź czy wszystko działa





12. Rozszerz działanie serwera, tak aby odpowiadał przywitaniem na wysłaną wiadomość

```
0       @Override
1   @   public void onMessage(WebSocket webSocket, String s) {
2           LOGGER.debug("opMessage: {}",s);
3
4           webSocket.send( s: "Yes, How Can I serve you ?");
5       }
```

13. Zrestartuj serwer i połącz się ponownie z serwerem konfigurując jednocześnie Konsumenta wiadomości po stronie JavaScript

```
socket = new WebSocket("ws://localhost:8887");
socket.onmessage=function(data){console.log(data);}
```

Wyślij wiadomość raz jeszcze:

```
socket.send("Hi! Is anybody here ?")
```

Sprawdź, czy masz odpowiedź:

```
> socket.send("Hi! Is anybody here ?")
< undefined
  ▶ MessageEvent {isTrusted: true, data: "Yes, How Can I serve you ?", origin: "ws://localhost:8887", lastEventId: "", source: n
>
```

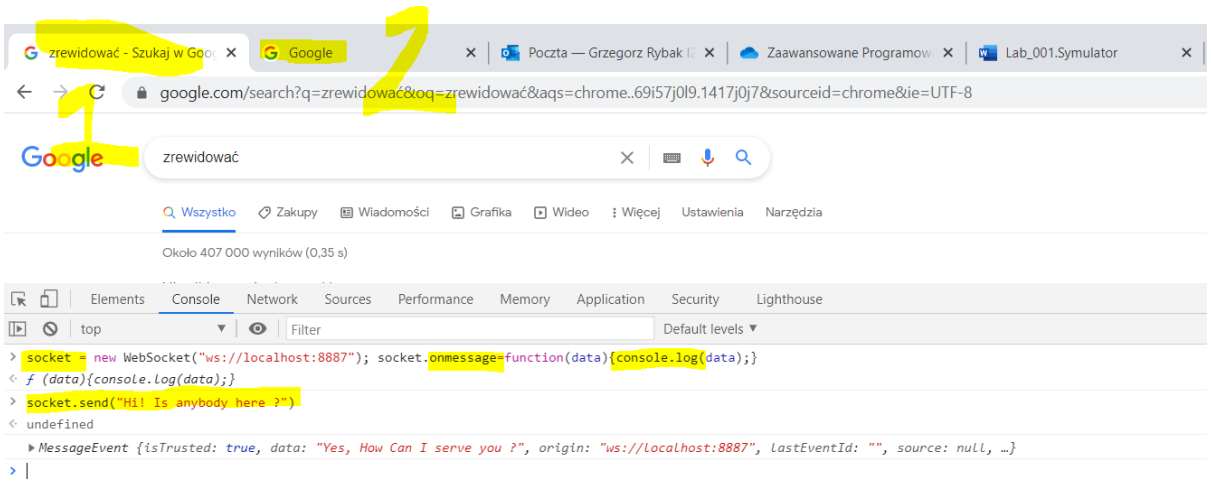
14. Rozszerz serwer, tak aby gromadzić referencje do wszystkich aktywnych połączeń
    a. Zastosuj HashMapę lub kolekcję
    b. OnOpen – dodaj wpis do kolekcji
    c. OnClose - usuń wpis z kolekcji
    d. OnMessage – dla wszystkich wpisów przekaż dalej wiadomość “message”

```java
    public Server(int port) throws UnknownHostException {
        super(new InetSocketAddress(port));
    }

    @Override
    public void onOpen(WebSocket webSocket, ClientHandshake clientHandshake) {
        LOGGER.debug("opOpen");
        connections.add(webSocket);
    }

    @Override
    public void onClose(WebSocket webSocket, int i, String s, boolean b) {
        LOGGER.debug("opClose");
        connections.remove(webSocket);
    }

    @Override
    public void onMessage(WebSocket webSocket, String s) {
        LOGGER.debug("opMessage: {}",s);

        connections.stream().forEach(ws->{ws.send( s: "Yes, How Can I serve you ?");});

    }
```

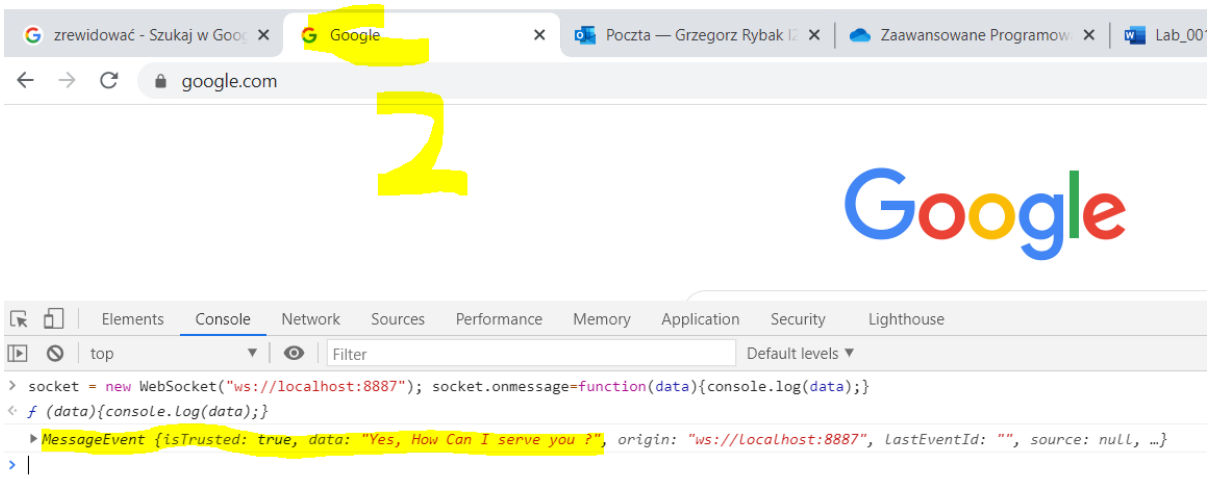
15. Przetestuj

- **a.** Uruchom kartę przeglądarki jak wcześniej
- **b.** Uruchom druga kartę przeglądarki jak wcześniej
- **c.** Wyślij wiadomość z jednej karty przeglądarki
- **d.** Odczytaj wiadomość w drugiej karcie przeglądarki

1

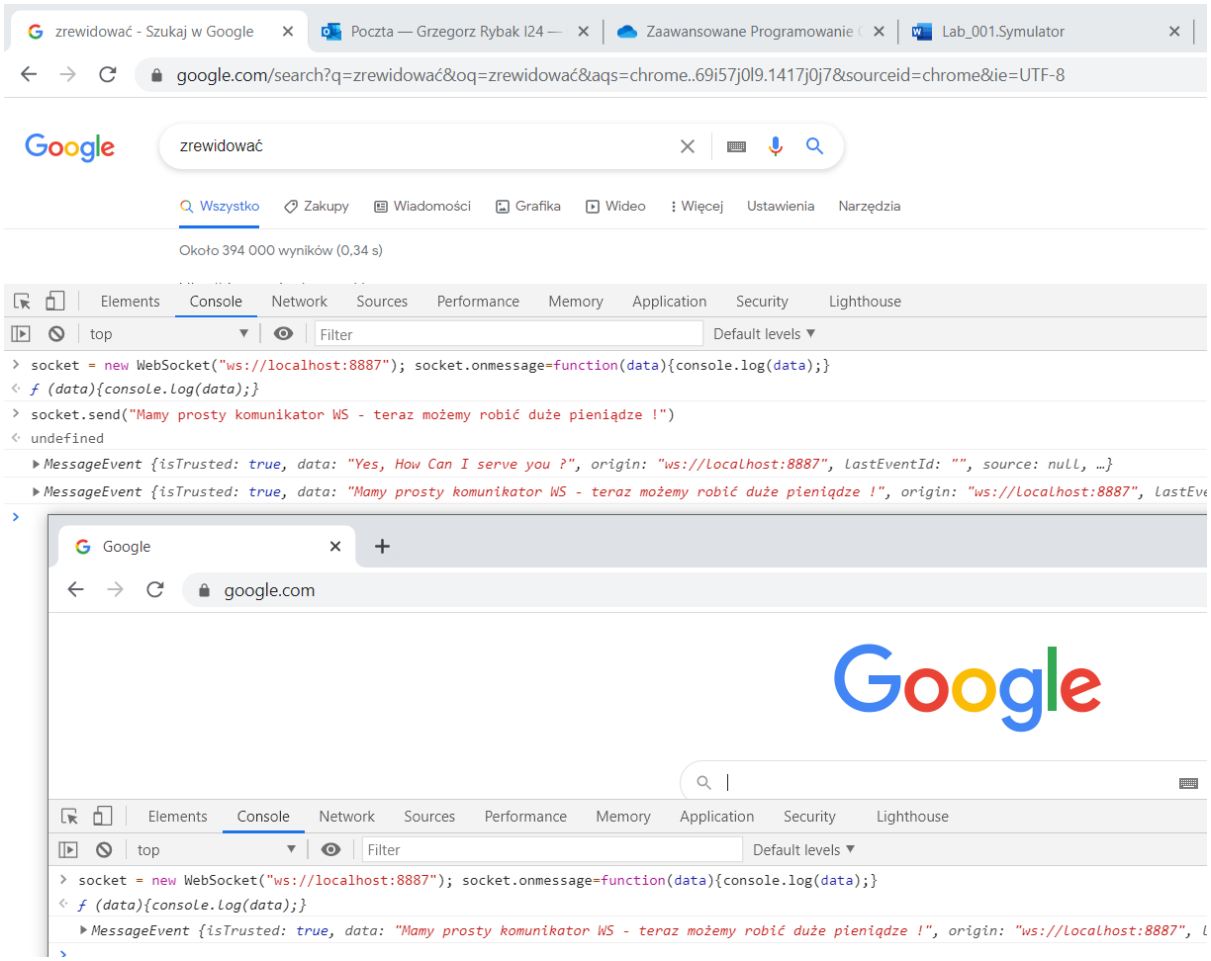


2



16. Popraw treść forwardowanej wiadomości, tak aby była tą, która jest wysłana z pierwszej zakładki (zrestartuj serwer)

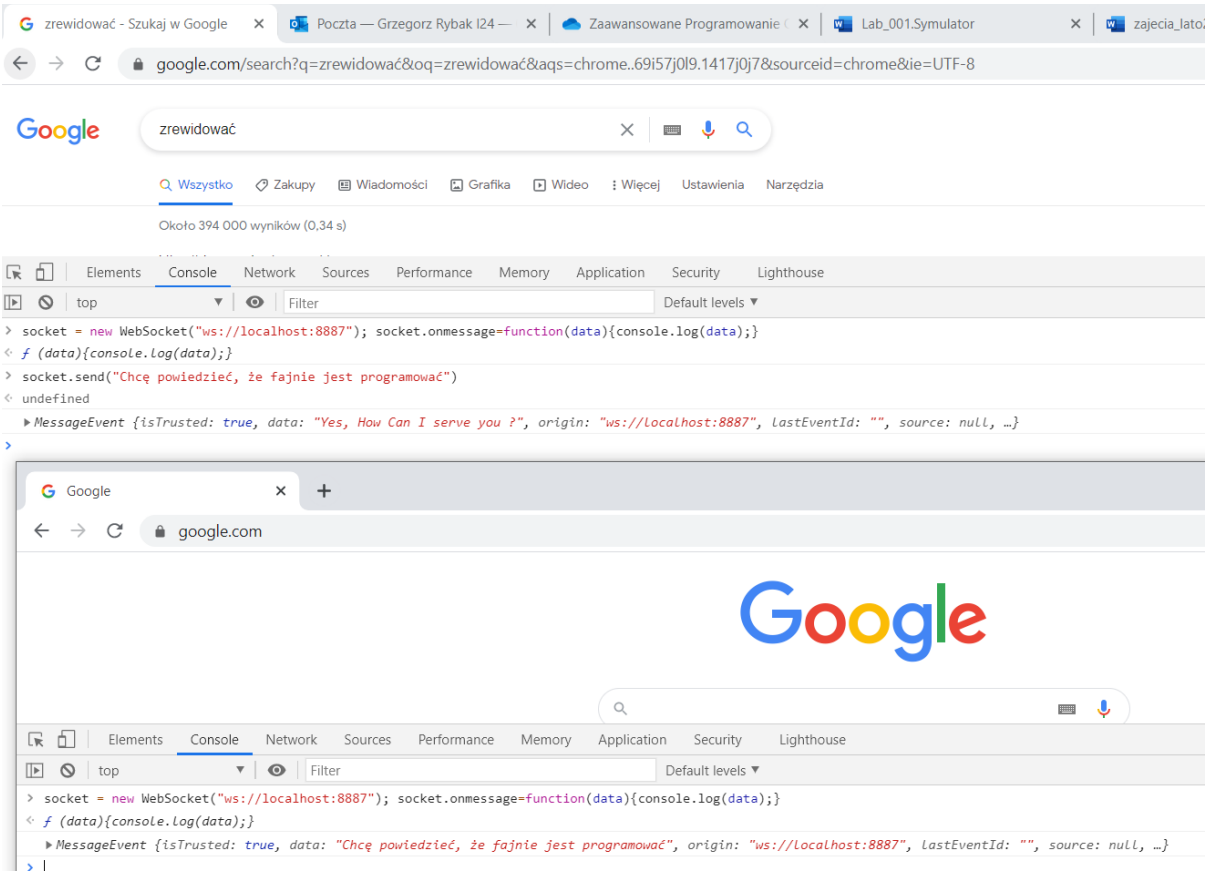
```
36      @Override
37      public void onMessage(WebSocket webSocket, String s) {
38          LOGGER.debug("opMessage: {}",s);
39          webSocket.send( s: "Yes, How Can I serve you ?");
40          connections.stream().forEach(ws->{ws.send(s);});
41
```



17. Nie wysyłaj wiadomości do siebie samego

Zastosowanie Lambd i programowania funkcyjnego będzie na kolejnym wykładzie. (Stream API), ale macie kawałek kodu żeby zobaczyć o co chodzi.

```
@Override
public void onMessage(WebSocket webSocket, String s) {
        LOGGER.debug("opMessage: {}",s);
        webSocket.send("Yes, How Can I serve you ?");
        connections.stream()
                .filter(ws->!webSocket.equals(ws))
              .forEach(ws->{ws.send(s);});
}
```



18. Uruchom serwer z dostępem do IP zewnętrznego (najlepiej stałe IP) - odblokuj dostępy w fireWall jeśli takie są założone etc.
19. Podaj kolegom swoje IP i port, na którym uruchomiłeś komunikator

(w aplikacji MSTeams)

20. Grupa może wysyłać wiadomości na Czat/komunikator tekstowy przy zastosowaniu WebSocket

21. Wejdź na MSTeams na połączenie głosowe i podziel się informacją na temat realizacji lab. 😉