|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A close up of a logo  Description automatically generated | Oulu University of Applied Sciences | | Product design and implementation | |
| Made by  Michał Kowalski,  Natalie Breu,  Artur Łomozik | | Learner's groups:  XVAI24S  EXT24SDIN22SP | | Classes conducted by  Teemu Korpela |

**Independent chat system project**

Main goals:

1. **Building independent connection - Using our own design to make connectivity possible - that includes simple protocol and physical connection.**
2. **Designing our own message coding system - To code the message and make it hardware compatible we needed to make our own code.**
3. **Implementing translation – The system that enables the translation from human readable language to hardware language and back to the human readable language.**
4. **Making user friendly interface – The WebSocket base web server on which the user can access the system**

Every part of the project was assigned to one of the group members and was responsible for it. Michał Kowalski was handling the physical circuit, python code that enables the communication and the script for statistics. Natalie Breu made the web server, user interface and was the one taking care of the github repository. Artur Łomozik made the communication code and the translation. He was also responsible for joining the group efforts into one project.

1. **Building independent connection – Michał Kowalski**

The base idea was to connect 2 raspberry pies with each other using GPIO inputs and outputs. The devices read the 3,3 V signal to send “1” and a 0 V signal is interpreted as “0”. The communication is time based so that every symbol (0 or 1) has a set time slot to be sent by Rp\_1(raspberry pie 1) and received by Rp\_2(raspberry pie 2).

The components used for the final version were:

* Bread board
* Red LED
* Wires
* 1000 ohm resistor

The first design included the separate circuit for the special clock circuit running on shepherd thread, but proved to be inefficient and it didn’t keep the same frequency. After that realization Michał decided to switch the approach for the event ration base system. This was a better solution and it worked much better considering the python programming language.

**Finished design:**

A diagram of a circuit board

Description automatically generated

**Picture 1** Design of the circuit

A circuit board with wires and wires

Description automatically generated

**Picture 2** Actual physical circuit on the breadboard

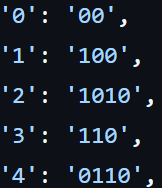
1. **Designing own message coding system – Artur Łomozik and Michał Kowalski**

For this part Artur used the script made by Michał to run the statistics on the Harry Potter book series. The main point was to calculate how often each symbol appears in the English language in percent. Thanks to the statistics Artur used a Huffman algorithm to make the code.

A graph with numbers and lines

Description automatically generated

**Picture 3** Huffman tree

A screenshot of a computer code

Description automatically generated

**Picture 4** The symbols from the Tree

The code was not ideal because the alphabet took 24 symbols which prove to be hard to implement. Michał and Artur decided to modify Huffman code so it takes only 10 symbols, by assigning the letter to the number:

|  |  |
| --- | --- |
| Symbols | Assigned number |
| a - z | 01-26 |
| A - Z | 27-52 |
| 0 - 9 | 53-62 |

Extra characters:

' ' = 63, ' . ' = 64, ' , ' = 65, " ' " = 66, 'null' = 67, '-' = 68, ' ! ' = 69, '?' = 70, '(' = 71, ')' = 72, ':' = 73, ';' = 74

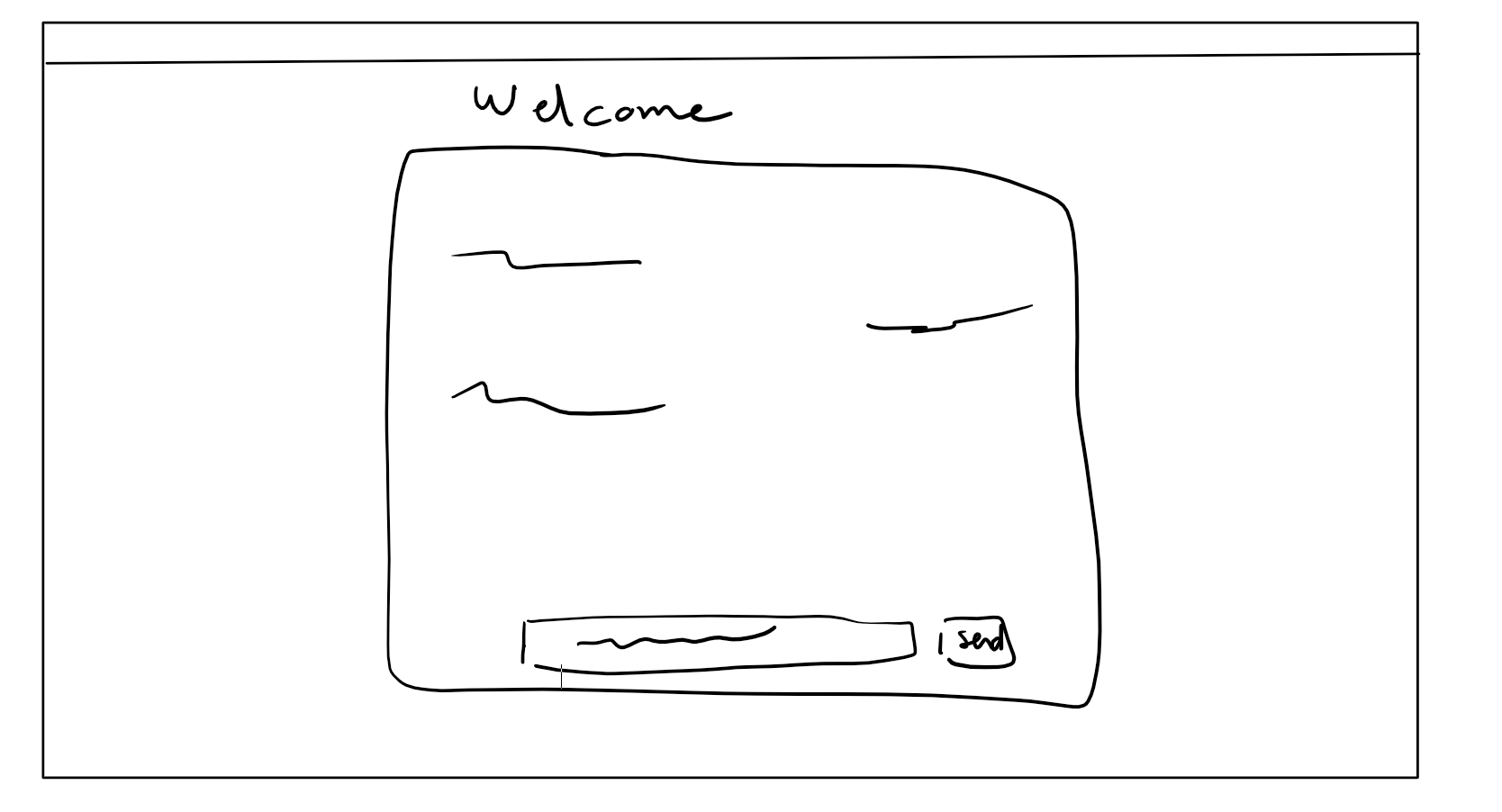
1. **Implementing translation – Artur Łomozik**

Artur wrote the script that implemented Huffman and the new solution that made transfer of information possible. This is how the translation works:

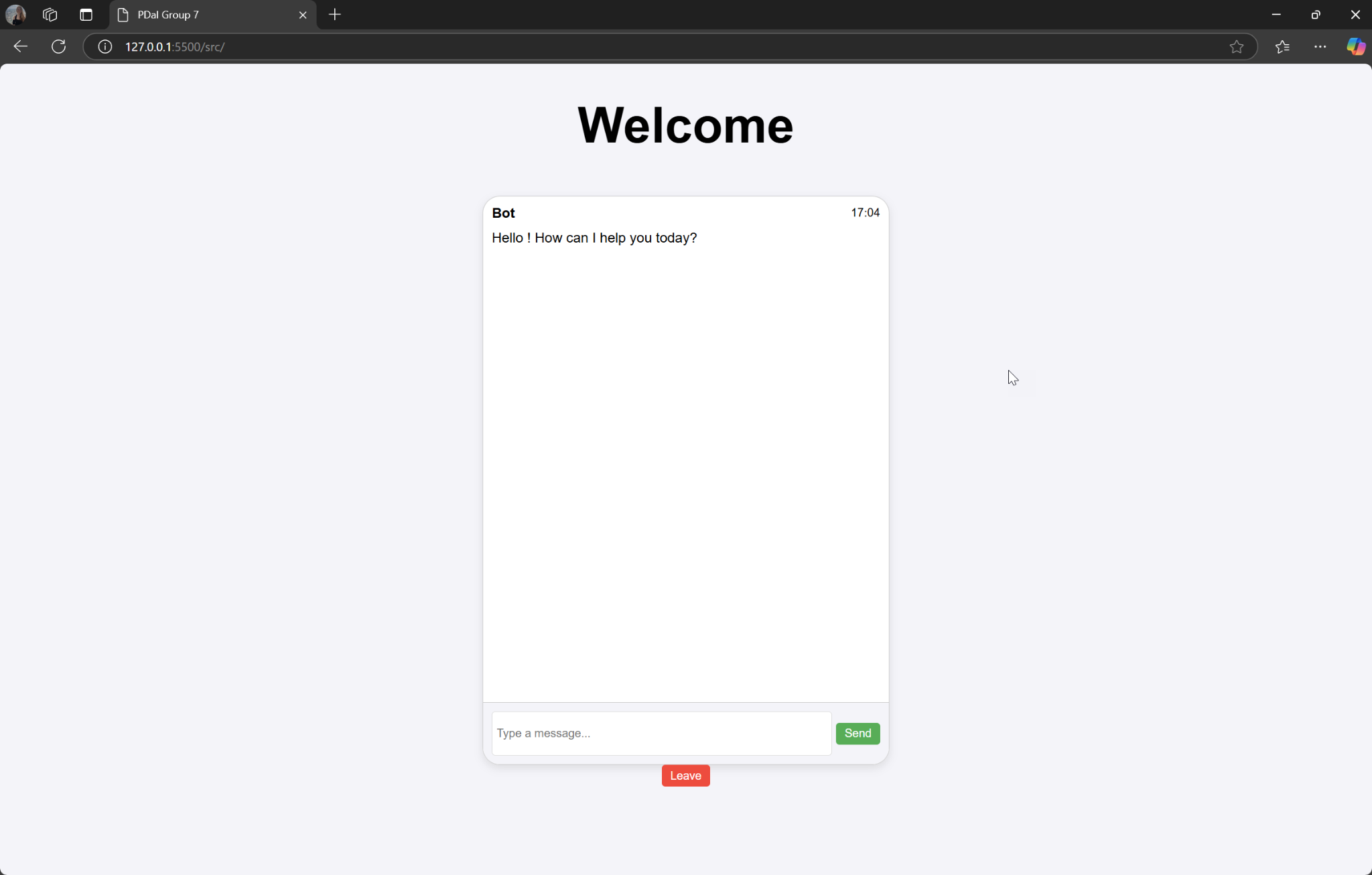
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | Assigning the number | Splitting the number | Assigning the Huffman code | Output |
| “abc” | a = “01”,  b = “02”,  c = “03” | “0”,  “1”,  “0”,  “2”,  “0”,  “3” | “0” = 00,  “1” = 100,  “0” = 00,  “2” = 1010  “0” = 00  “3” = 110 | “0010000101000110” |

1. **Making user friendly interface – Natalie Breu**

Our first idea was to just create a simple web interface with an input field and the chat history. We created this quite fast with HTML and JavaScript.

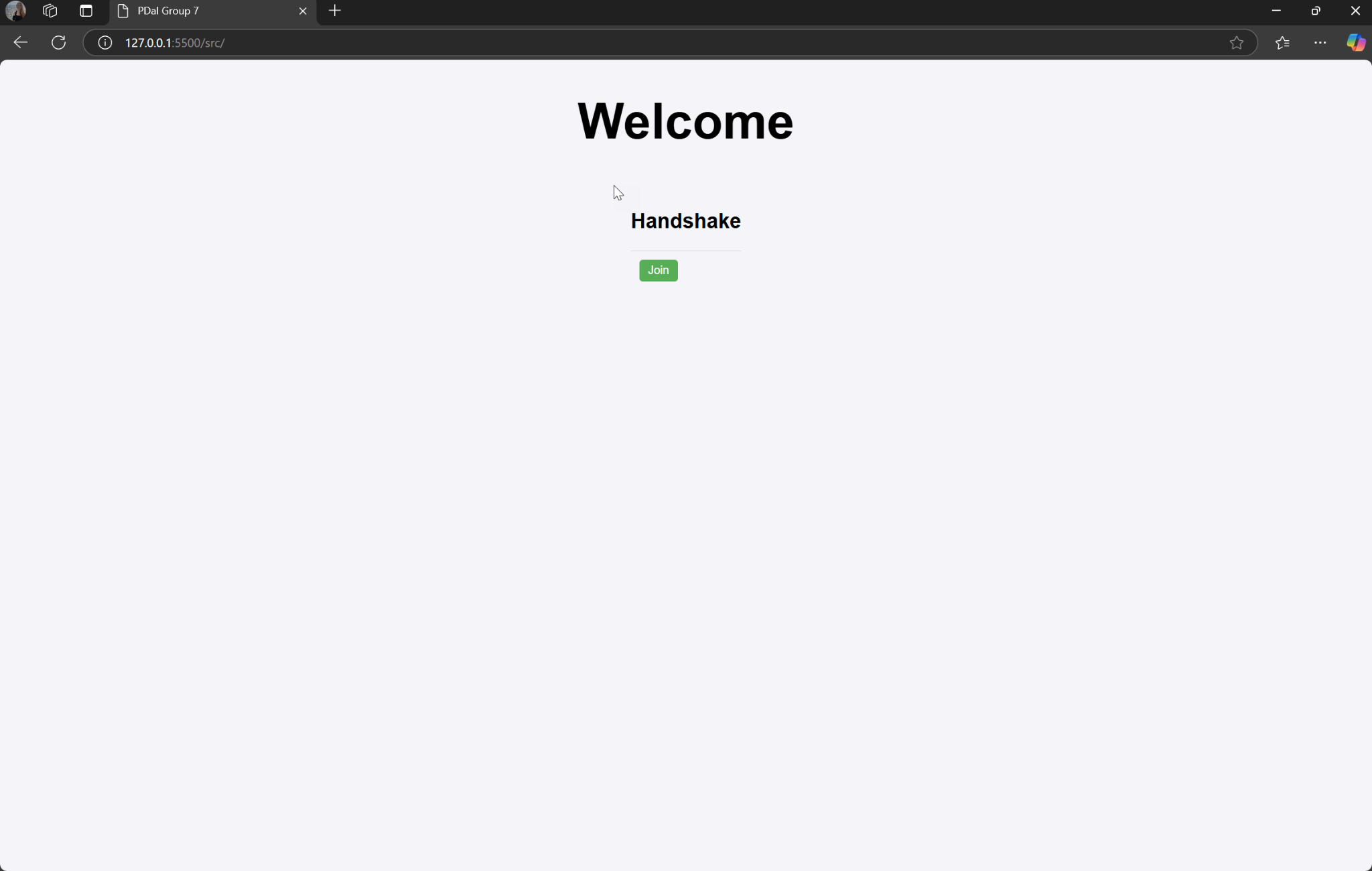


**Picture 5** Initial design of UI



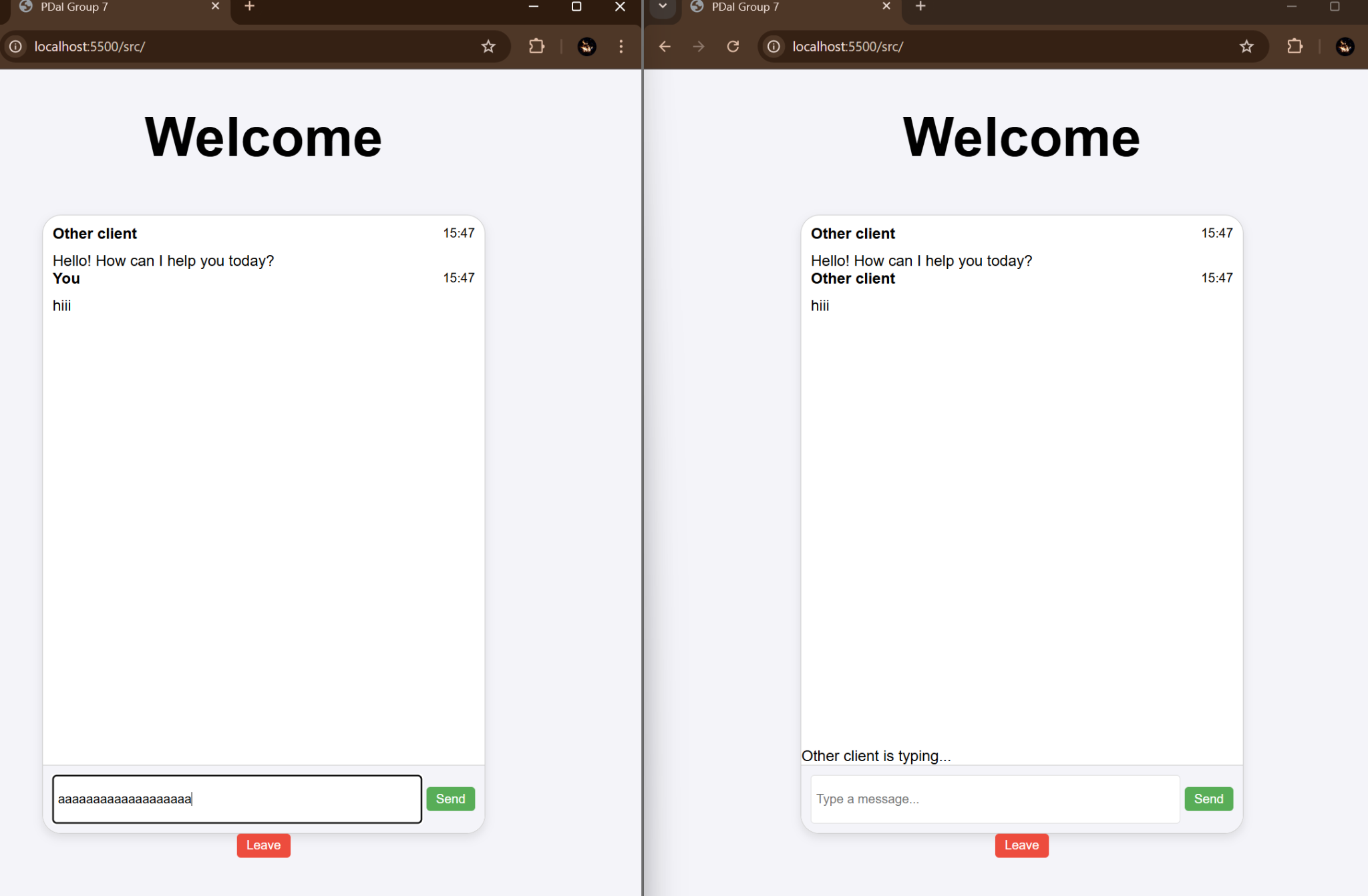
**Picture 6** The one Implemented

In addition we created a loading page where the two clients can click “Join” and wait until the two Raspberries are connected with each other. To keep it as simple as possible we made it as a one page.



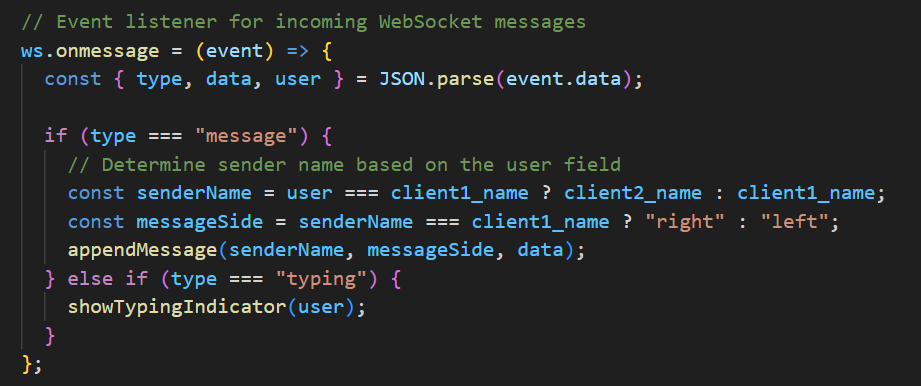
**Picture 7** Before connection window

As a special extra, Teemu suggested adding a Websocket so the clients can see when the other one is typing a message.

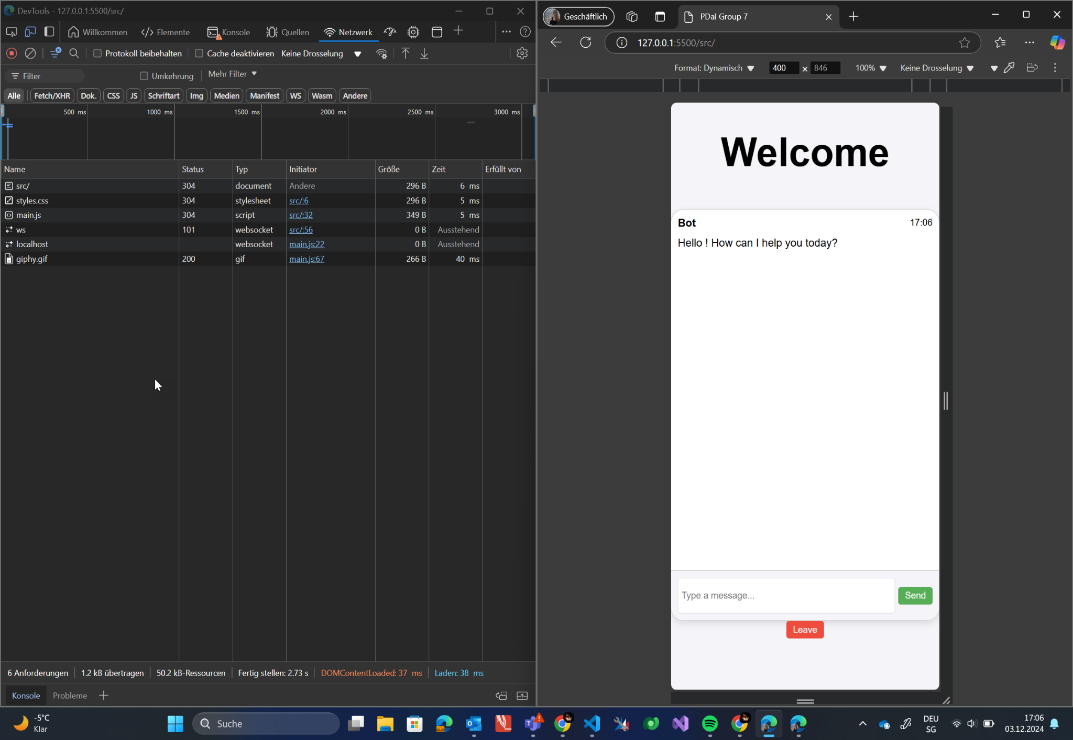


Picture 8 WebSocket added

We implemented that with Node.js and the Websocket API and two messages: One for when a user sent a message and one for when a user is typing a message.



**Picture 9** Code for listening event



**Picture 10** Presentation of communication; stage 1

A screenshot of a computer

Description automatically generated

**Picture 11** Presentation of communication; stage 2