# *Task 4*
# Browser Game

The fourth and last goal is to design and implement a simple Browser-based application or game which uses the two thingies as (very ordinary) input devices/game controllers. The following figure shows such a configuration:

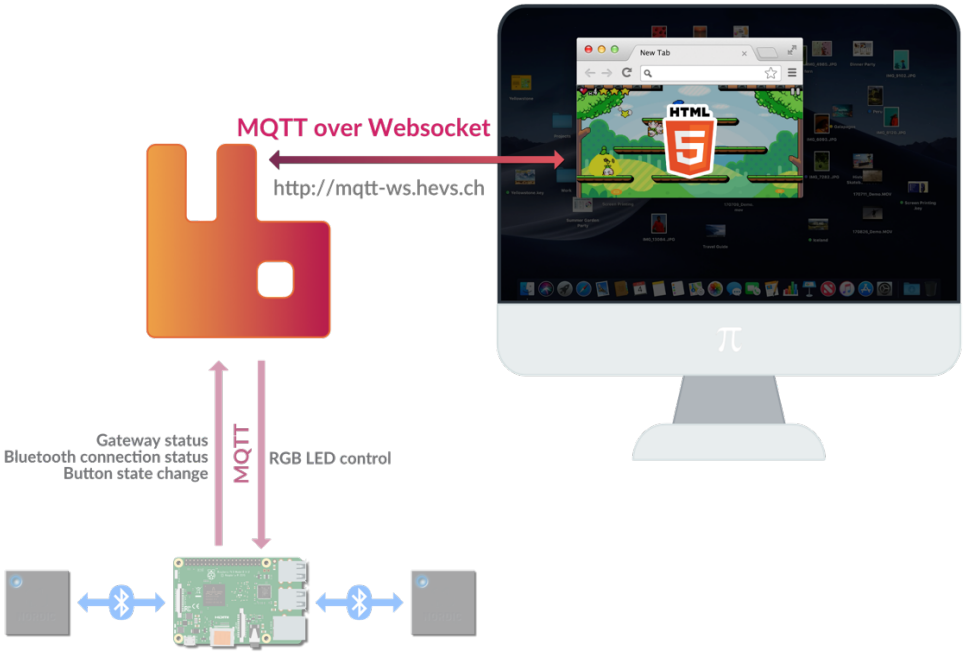


*Figure 1: HTML5 based Browser game connected via MQTT over WebSocket to the message broker.*

The HTML/JavaScript browser application or game resides on your local machine and establishes an **MQTT over Websocket** connection to the message broker **mqtt-ws.sdi.hevs.ch**. The gateway software on the Raspberry Pi will be used as we had it at the end of Task 2 and will connect to the message broker via **MQTT**.

Using this architecture, the web application/game can process button events (or other sensor events) from the two Nordic Thingies or publish messages in order to control the LED (or other actuators) of the Thingies.

You are completely free what kind of application or game you want to develop for this task; however we require you to add some key features:

- The **buttons** on the thingies are used for **input**.
- The **RGB LED** on the thingies are used to signal **game state** and/or to **identify a player**.
- Either **game data** is **read** from or **results** are **written** to the **MongoDB** (or both).

**We are curious to see all your creative ideas!**

### Connecting to RabbitMQ via MQTT over WebSocket:

MQTT is transport agnostic – This means that the protocol can be theoretically on top of any protocol. There exist multiple implementations of the MQTT protocol on top of **WebSockets**. The most popular one is **Eclipse Paho** (https://www.eclipse.org/paho/clients/js).

We provide here some example code to get you started with Eclipse Paho JavaScript:

```
var client = new Paho.MQTT.Client("mqtt-ws.sdi.hevs.ch", 80, "/ws", "rgubsdflkoihisohieg");

client.onConnectionLost = function(responseObject) {
    if (responseObject.errorCode !== 0) {
        console.error("Lost connection:" + responseObject.errorMessage);
    } else {
        console.log("Connection closed.");
    }
}

client.onMessageArrived = function(message) {
    console.log("Got message: topic=" + message.destinationName + ', payload=' +
                message.payloadString);
}

client.connect({
    userName: 'sdiXX',
    password: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    keepAliveInterval: 30,
    cleanSession: true,
    onSuccess: function() {
        console.log("Connected.");
        client.subscribe("sdiXX/status");
        client.subscribe("sdiXX/+/button");
        client.send('sdiXX/XX:XX:XX:XX:XX:XX/led', JSON.stringify({
            red: 0,
            green: 0,
            blue: 255
        }));
    },
    onFailure: function() {
        console.error("Failed to connect.");
    }
});
```

You need to include the Paho Javascript MQTT client implementation in your HTML file in order to be able to use the Paho client functionality in your scripts:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.js">
</script>
```

(You can add the **mqttws31.min.js** JavaScript file to your project if you prefer that too)

## 1. GitHub Classroom

1. You can add the content for this task to the repository used for the previous Tasks.
2. Create a folder **Web** and implement the HTML5 application/game inside that folder.

## 2. Tools

You are free to use the tools of your choice during the practical work - use your favorite editor or IDE. We want to give you just some tips and point you to the tools we use and love ourselves for web development.

### *JetBrains WebStorm*

**WebStorm** is an IDE specialized for Web development using **HTML**, **CSS** and **JavaScript**. It is one of the few IDEs that actually understands the relations between all those files in such a project.

Additionally, there is an integrated web server that serves the files you are actually working on and you can see the changes to your website in real-time and debug your code directly inside the IDE instead of using the developer tools on a web browser (Chrome required).

As all products from JetBrains, you can use the IDE for free if you register with your student email address at JetBrains.

**https://www.jetbrains.com/webstorm/**

### MQTT Tester

You can use the MQTT Tester in order to debug the messages exchanged between your application/game and the virtual game pad:

**https://tester.sdi.hevs.ch**

### Phaser

**Phaser** is a free software 2D game framework for creating HTML5 games for desktop and mobile. Phaser uses both a **Canvas** and **WebGL** renderer internally and can automatically swap between them based on browser support. This allows for fast rendering across desktop and mobile. It uses the pixi.js library for rendering.

**https://phaser.io**

### pixi.js

**Pixi.js** is an open source library allowing to create 2D animations in a web browser. The display is based on **WebGL** technology, and the library is manipulated in JavaScript.

**https://www.pixijs.com**

### Babylon.js

**Babylon.js** is a **real time 3D engine** using a JavaScript library for displaying 3D graphics in a web browser via HTML5.

**https://www.babylonjs.com**

### Tree.js

**Three.js** is a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser. Three.js uses **WebGL**.

**https://threejs.org**

## 3. Game implementation

1. Implement your HTML5 application or game inside the folder **Web** in your repository.
2. Connect to the RabbitMQ message broker on **mqtt-ws.sdi.hevs.ch** using **Paho** and handle **button events** and control the **RGB LED** of your Nordic Thingies to control your application or game and control the RGB LED.
3. Once finished, commit your changes, create the tag **Task4-1** and push your changes to the GitHub repository.

## 4. Report

4. Prepare a **simple report** about your application or game.
   Focus on:
   - Idea
   - Design (Software architecture)
   - Implementation
   - Problems
   - Instructions to run the game
5. Save the report document as PDF to the folder **Report** in your repository, commit your changes, tag them as **Task4-2** and push the changes to the repository on GitHub.