Web Socket Example

```
 $ tree -L 2
.
├── build
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   ├── Makefile
│   ├── templates
│   └── websocket_server
├── CMakeLists.txt
├── server.cpp
└── templates
    └── ws.html

5 directories, 7 files
```

> CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 3.16)
project(websocket_example)

set(CMAKE_CXX_STANDARD 17)

# Find Crow
find_package(Crow REQUIRED PATHS ../libs/crow/lib/cmake/Crow)

# Create executable
add_executable(websocket_server server.cpp)

# Find required libraries
find_package(OpenSSL REQUIRED)
find_package(ZLIB REQUIRED)
find_package(Threads REQUIRED)

# Link libraries
target_link_libraries(websocket_server
    Crow::Crow
    OpenSSL::SSL
    OpenSSL::Crypto
    ZLIB::ZLIB
    Threads::Threads
)

# Copy templates directory to build directory
file(COPY ${CMAKE_SOURCE_DIR}/templates DESTINATION ${CMAKE_BINARY_DIR})
```

> server.cpp

```cpp
#include "crow.h"
#include <unordered_set>
#include <mutex>

std::mutex mtx;
std::unordered_set<crow::websocket::connection*> users;

// GPIO states (simulating hardware)
struct GPIOState {
    bool gpio1 = false;
    bool gpio2 = false;
    bool gpio3 = false;
};

GPIOState gpio_state;

void broadcast_gpio_state() {
    std::string message = "{\"type\":\"gpio_update\",\"gpio1\":"
        + std::string(gpio_state.gpio1 ? "true" : "false")
        + ",\"gpio2\":" + std::string(gpio_state.gpio2 ? "true" : "false")
        + ",\"gpio3\":" + std::string(gpio_state.gpio3 ? "true" : "false") +
"}";

    std::lock_guard<std::mutex> _(mtx);
    for (auto u : users) {
        u->send_text(message);
    }
}

int main()
{
    crow::SimpleApp app;

    CROW_WEBSOCKET_ROUTE(app, "/ws")
```

```cpp
      .onopen([&](crow::websocket::connection& conn) {
          CROW_LOG_INFO << "new websocket connection from " <<
conn.get_remote_ip();
          std::lock_guard<std::mutex> _(mtx);
          users.insert(&conn);
      })
      .onclose([&](crow::websocket::connection& conn, const std::string&
reason, uint16_t) {
          CROW_LOG_INFO << "websocket connection closed: " << reason;
          std::lock_guard<std::mutex> _(mtx);
          users.erase(&conn);
      })
      .onmessage([&](crow::websocket::connection& /*conn*/, const
std::string& data, bool is_binary) {
          std::lock_guard<std::mutex> _(mtx);
          for (auto u : users)
              if (is_binary)
                  u->send_binary(data);
              else
                  u->send_text(data);
      });

    CROW_ROUTE(app, "/")
    ([] {
        char name[256];
        gethostname(name, 256);
        crow::mustache::context x;
        x["servername"] = name;

        auto page = crow::mustache::load("ws.html");
        return page.render(x);
    });

    // HTTP endpoint for GPIO 1 toggle
    CROW_ROUTE(app, "/gpio1").methods("POST"_method)
    ([] {
        gpio_state.gpio1 = !gpio_state.gpio1;
        broadcast_gpio_state();
        return crow::response(200, gpio_state.gpio1 ? "ON" : "OFF");
    });

    // HTTP endpoint for GPIO 2 toggle
    CROW_ROUTE(app, "/gpio2").methods("POST"_method)
    ([] {
        gpio_state.gpio2 = !gpio_state.gpio2;
        broadcast_gpio_state();
        return crow::response(200, gpio_state.gpio2 ? "ON" : "OFF");
    });

    // HTTP endpoint for GPIO 3 toggle
    CROW_ROUTE(app, "/gpio3").methods("POST"_method)
    ([] {
        gpio_state.gpio3 = !gpio_state.gpio3;
        broadcast_gpio_state();
        return crow::response(200, gpio_state.gpio3 ? "ON" : "OFF");
    });

    app.port(8080)
      .multithreaded()
      .run();
}
```

> ws.html

```
<!DOCTYPE html>
<html>
<head>
    <title>GPIO Control Panel</title>
</head>
<body>
    <h1>GPIO Control Panel</h1>

    <h2>Hardware Controls</h2>
    <button onclick="toggleGPIO(1)">GPIO 1: <span id="gpio1-
status">OFF</span></button><br><br>
    <button onclick="toggleGPIO(2)">GPIO 2: <span id="gpio2-
status">OFF</span></button><br><br>
    <button onclick="toggleGPIO(3)">GPIO 3: <span id="gpio3-
status">OFF</span></button><br><br>

    <h2>Connection Status</h2>
    <div id="status">Connecting to WebSocket...</div>

    <script>
    var sock = new WebSocket("ws://{{servername}}:8080/ws");
    sock.onopen = function() {
        console.log('WebSocket connected');
        document.getElementById('status').innerHTML = 'Connected - Ready to
receive updates';
    }

    sock.onerror = function(e) {
        console.log('WebSocket error', e);
        document.getElementById('status').innerHTML = 'Connection Error';
    }

    sock.onclose = function(e) {
        console.log('WebSocket closed', e);
        document.getElementById('status').innerHTML = 'Disconnected';
    }

    sock.onmessage = function(e) {
        try {
            var data = JSON.parse(e.data);
            if (data.type === 'gpio_update') {
                // Update all button states when any client changes GPIO
                document.getElementById('gpio1-status').innerHTML =
data.gpio1 ? 'ON' : 'OFF';
                document.getElementById('gpio2-status').innerHTML =
data.gpio2 ? 'ON' : 'OFF';
                document.getElementById('gpio3-status').innerHTML =
data.gpio3 ? 'ON' : 'OFF';

                document.getElementById('status').innerHTML =
                    'GPIO States - 1:' + (data.gpio1 ? 'ON' : 'OFF') +
                    ', 2:' + (data.gpio2 ? 'ON' : 'OFF') +
                    ', 3:' + (data.gpio3 ? 'ON' : 'OFF');
            }
        } catch (e) {
            console.log('Message received:', e.data);
        }
    }

    function toggleGPIO(gpioNum) {
        fetch('/gpio' + gpioNum, {
            method: 'POST'
        })
        .then(response => response.text())
        .then(data => {
            console.log('GPIO' + gpioNum + ' toggled to:', data);
        })
        .catch(error => {
            console.log('Failed to toggle GPIO' + gpioNum, error);
        });
    }
    </script>
</body>
</html>
```