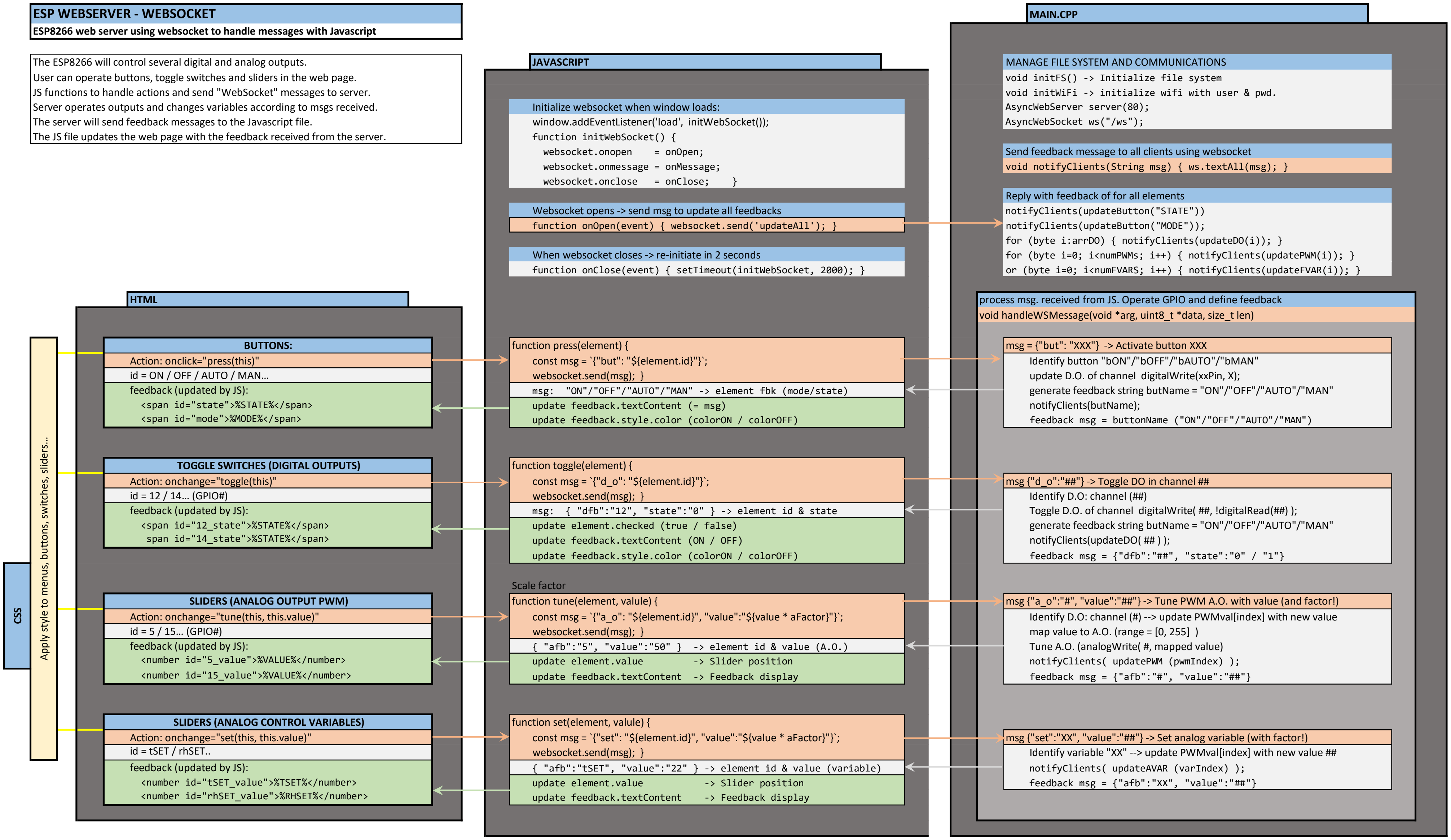
ESP WEBSERVER - WEBSOCKET

ESP8266 web server using websocket to handle messages with Javascript

The ESP8266 will control several digital and analog outputs.
User can operate buttons, toggle switches and sliders in the web page.
JS functions to handle actions and send "WebSocket" messages to server.
Server operates outputs and changes variables according to msgs received.
The server will send feedback messages to the Javascript file.
The JS file updates the web page with the feedback received from the server.



CSS

Apply style to menus, buttons, switches, sliders...

HTML

BUTTONS:
Action: onclick="press(this)"
id = ON / OFF / AUTO / MAN...
feedback (updated by JS):

```
<span id="state">%STATE%</span>
<span id="mode">%MODE%</span>
```

TOGGLE SWITCHES (DIGITAL OUTPUTS)
Action: onchange="toggle(this)"
id = 12 / 14... (GPIO#)
feedback (updated by JS):

```
<span id="12_state">%STATE%</span>
 span id="14_state">%STATE%</span>
```

SLIDERS (ANALOG OUTPUT PWM)
Action: onchange="tune(this, this.value)"
id = 5 / 15... (GPIO#)
feedback (updated by JS):

```
<number id="5_value">%VALUE%</number>
<number id="15_value">%VALUE%</number>
```

SLIDERS (ANALOG CONTROL VARIABLES)
Action: onchange="set(this, this.value)"
id = tSET / rhSET..
feedback (updated by JS):

```
<number id="tSET_value">%TSET%</number>
<number id="rhSET_value">%RHSET%</number>
```

JAVASCRIPT

Initialize websocket when window loads:

```
window.addEventListener('load', initWebSocket());
function initWebSocket() {
  websocket.onopen    = onOpen;
  websocket.onmessage = onMessage;
  websocket.onclose   = onClose;    }
```

Websocket opens -> send msg to update all feedbacks

```
function onOpen(event) { websocket.send('updateAll'); }
```

When websocket closes -> re-initiate in 2 seconds

```
function onClose(event) { setTimeout(initWebSocket, 2000); }
```

```
function press(element) {
  const msg = `{"but": "${element.id}"}`;
  websocket.send(msg); }
msg:  "ON"/"OFF"/"AUTO"/"MAN" -> element fbk (mode/state)
update feedback.textContent (= msg)
update feedback.style.color (colorON / colorOFF)
```

```
function toggle(element) {
  const msg = `{"d_o": "${element.id}"}`;
  websocket.send(msg); }
msg:  { "dfb":"12", "state":"0" } -> element id & state
update element.checked (true / false)
update feedback.textContent (ON / OFF)
update feedback.style.color (colorON / colorOFF)
```

Scale factor

```
function tune(element, value) {
  const msg = `{"a_o": "${element.id}", "value":"${value * aFactor}"}`;
  websocket.send(msg); }
{ "afb":"5", "value":"50" }  -> element id & value (A.O.)
update element.value         -> Slider position
update feedback.textContent  -> Feedback display
```

```
function set(element, value) {
  const msg = `{"set": "${element.id}", "value":"${value * aFactor}"}`;
  websocket.send(msg); }
{ "afb":"tSET", "value":"22" } -> element id & value (variable)
update element.value           -> Slider position
update feedback.textContent    -> Feedback display
```

MAIN.CPP

MANAGE FILE SYSTEM AND COMMUNICATIONS

```
void initFS() -> Initialize file system
void initWiFi -> initialize wifi with user & pwd.
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");
```

Send feedback message to all clients using websocket

```
void notifyClients(String msg) { ws.textAll(msg); }
```

Reply with feedback of for all elements

```
notifyClients(updateButton("STATE"))
notifyClients(updateButton("MODE"));
for (byte i:arrDO) { notifyClients(updateDO(i)); }
for (byte i=0; i<numPWMs; i++) { notifyClients(updatePWM(i)); }
or (byte i=0; i<numFVARS; i++) { notifyClients(updateFVAR(i)); }
```

process msg. received from JS. Operate GPIO and define feedback
void handleWSMessage(void *arg, uint8_t *data, size_t len)

msg = {"but": "XXX"} -> Activate button XXX
Identify button "bON"/"bOFF"/"bAUTO"/"bMAN"
update D.O. of channel digitalWrite(xxPin, X);
generate feedback string butName = "ON"/"OFF"/"AUTO"/"MAN"
notifyClients(butName);

```
feedback msg = buttonName ("ON"/"OFF"/"AUTO"/"MAN")
```

msg {"d_o":"##"} -> Toggle DO in channel ##
Identify D.O: channel (##)
Toggle D.O. of channel digitalWrite(##, !digitalRead(##));
generate feedback string butName = "ON"/"OFF"/"AUTO"/"MAN"
notifyClients(updateDO(##));

```
feedback msg = {"dfb":"##", "state":"0" / "1"}
```

msg {"a_o":"#", "value":"##"} -> Tune PWM A.O. with value (and factor!)
Identify D.O: channel (#) --> update PWMval[index] with new value
map value to A.O. (range = [0, 255])
Tune A.O. (analogWrite(#, mapped value)

```
notifyClients( updatePWM (pwmIndex) );
feedback msg = {"afb":"#", "value":"##"}
```

msg {"set":"XX", "value":"##"} -> Set analog variable (with factor!)
Identify variable "XX" --> update PWMval[index] with new value ##

```
notifyClients( updateAVAR (varIndex) );
feedback msg = {"afb":"XX", "value":"##"}
```