**Pump API**

This is a minimal REST API specification and implementation for controlling a dosing pump.

**Specification**

The minimal API is documented using the OpenAPI specification, the corresponding file can be found in /ESP8266/ESPMinPumpEx/minpumpapi.yaml

The specification file can also be rendered in an interactive form using the Swagger [editor](https://editor.swagger.io/) (go to file -> import yaml file)

The API specification for running pump\_water operations can be seen in /Operations/pump\_water/pump\_waterapi.yaml (Implementation in progress)

**Implementation**

**Computer**

The API can be implemented using different computer systems from the API specification. The example in this repository has been implemented using an ESP8266 NodeMCU V3 Wifi [board](http://prometec.org/communications/nodemcu/arduino-ide/).

**Source Code**

The software implementation is done in the Arduino language and makes use of the ESP8266WebServer and ArduinoJson libraries. Most languages have similar tools and frameworks for handling requests and providing responses (i.e. FastAPI), and for handling JSON documents. There are two important parts in the code:

* Routing: This part is in charge of matching the incoming request command (i.e GET) and path (i.e. /pump), with the corresponding program routine or action (get\_pump function).

void config\_rest\_server\_routing() { //Routing function: Set actions for incoming commands and locations

http\_rest\_server.on("/", HTTP\_GET, []() {

http\_rest\_server.send(200, "text/html",

"Welcome to Minimal REST Server Example");//Welcome page

});

http\_rest\_server.on("/pump", HTTP\_GET, get\_pump); //Return pump info

http\_rest\_server.on("/pump/on", HTTP\_GET, get\_pump\_on); //Turn pump on and return status

http\_rest\_server.on("/pump/off", HTTP\_GET, get\_pump\_off);//Turn pump off and return status

}

* Action: The program routine that is executed when handling a request, it is in charge of interpreting the incoming body of the request, building the JSON response (serialization) and sending it through the server.

void get\_pump\_on(){//GET /pump/on action

pump\_resource.status = LOW; //Change resource status

digitalWrite(pump\_resource.port, pump\_resource.status);//Change external pin status

DynamicJsonDocument doc(capacity);

doc["id"] = pump\_resource.id;

doc["port"] = pump\_resource.port;

doc["status"] = pump\_resource.status;

String buf;

serializeJson(doc, buf);

http\_rest\_server.send(200, F("application/json"), buf);

}

The JSON document is populated using information from a struct "database".

struct Pump { //Resource "database"

byte id;

byte port;

byte status;

} pump\_resource;

The web server is initialized by the setup function, which also attempts to connect to the wifi network. The main program loop waits for and handles incoming requests.

void setup() {

Serial.begin(115200);

init\_pump\_resource();

if (init\_wifi() == WL\_CONNECTED) { //Connect to Wifi

Serial.print("Connected to ");

Serial.print(wifi\_ssid);

Serial.print("--- IP: ");

Serial.println(WiFi.localIP());

}

else {

Serial.print("Error connecting to: ");

Serial.println(wifi\_ssid);

}

config\_rest\_server\_routing(); //Configure routing

http\_rest\_server.begin();//Start web server

Serial.println("HTTP REST Server Started");

}

void loop() {

http\_rest\_server.handleClient();//Handle incoming requests

}

**Hardware**

The following hardware is connected to the ESP8266 device:

* L298N motor driver [board](https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/)
* 12V, 3W peristaltic dosing [pump](https://www.adafruit.com/product/1150)
* 12V and 5V power source

**Connections**

* Pin D4 (GPIO 2) of the ESP8266 is connected to ENA in the L298N board.
* In1 and In2 in the L298N board are connected to 5V and Gnd respectively.
* Ground pins on the ESP8266 and the L298N board are connected together and to the main power source ground.
* The pump terminals are connected to OUT1 and OUT2 on the L298N board.