Safe Chicken

Release 0.1.0

Claudio Nold

Mar 06, 2021

CONTENTS

So what is *Safe Chicken*?

- a Raspberry Pi controlling a chicken door electronics so they are safe at night
- the door can be opened or closed according to:
  - sunrise & sunset (calculated)
  - static time
  - force command (web app open/close buttons)
- the web application helps checking the current state of the door and allowing some user commands
- the output is just a relay signal to open and close the door
- the door motor and the motor electronics are not part of this project; this setup just controls the relay signals (and two LEDs for displaying the status)
- uses an MQTT broker which runs on your NAS or a home server (but could also run on the Raspberry itself)
- allows unstable WLAN connections from the Raspberry to the home network

Get *Safe Chicken* on GitHub.

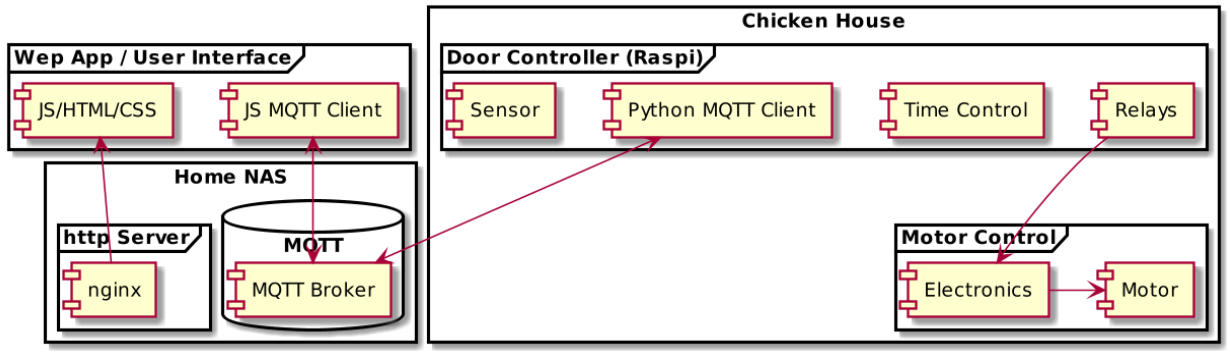
CHAPTER

ONE

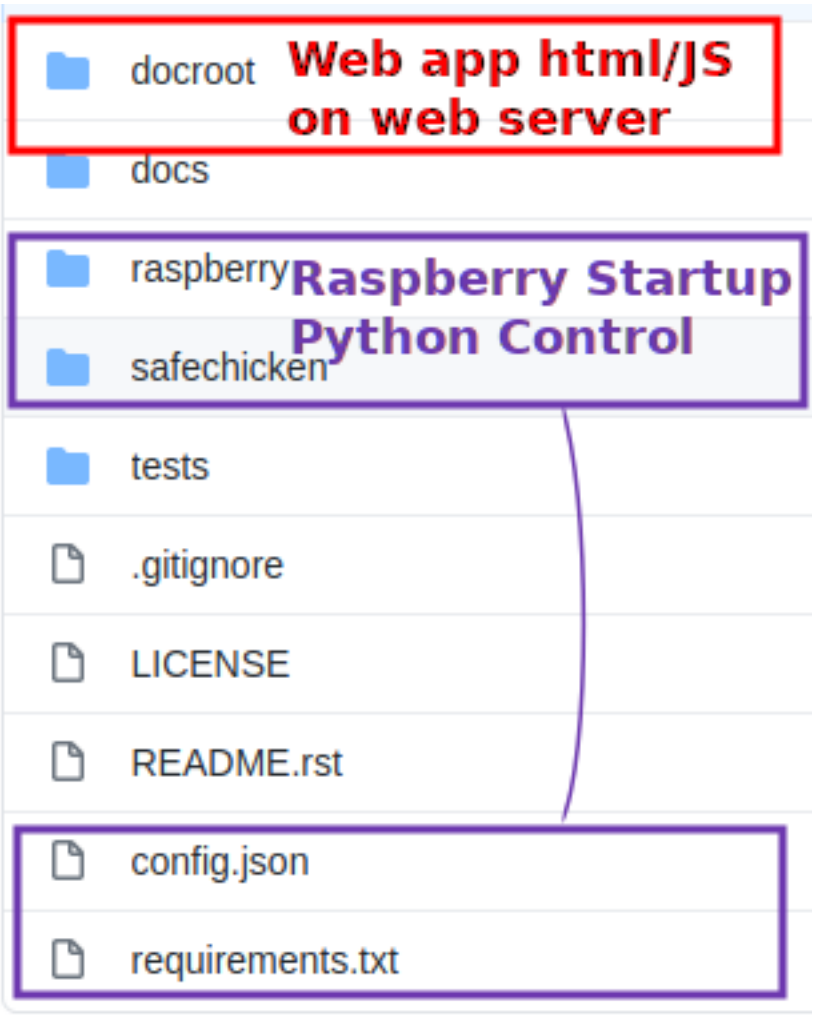
TABLE OF CONTENTS

1.1 Overview

So the main components are these:



So which software parts run where?



1.2 Door Control Hardware

1.2.1 Parts

Table 1: Hardware Components

No	Part Name	Usage
1	Raspberry 3 or 4 with power supply	door controller
2	M5Stack Mini 3A Relay Unit U023	door open relay
3	M5Stack Mini 3A Relay Unit U023	door close relay
4	M5Stack Hall Sensor U084	magnetic sensor for door closed
5	M5Stack 1 to 3 HUB Unit U006	sensor cable extension
6	M5Stack Unbuckled Grove Cable 50cm A034-C	sensor cable extension
7	Adafruit Jumperset 40 wires male/female, 15cm	Raspi GPIO connection
8	orange LED 11 mA + resistor 82 Ohms	automatic is working (= time synced)
9	green LED 11 mA + resistor 82 Ohms	MQTT connection to broker working

1.2.2 Connections

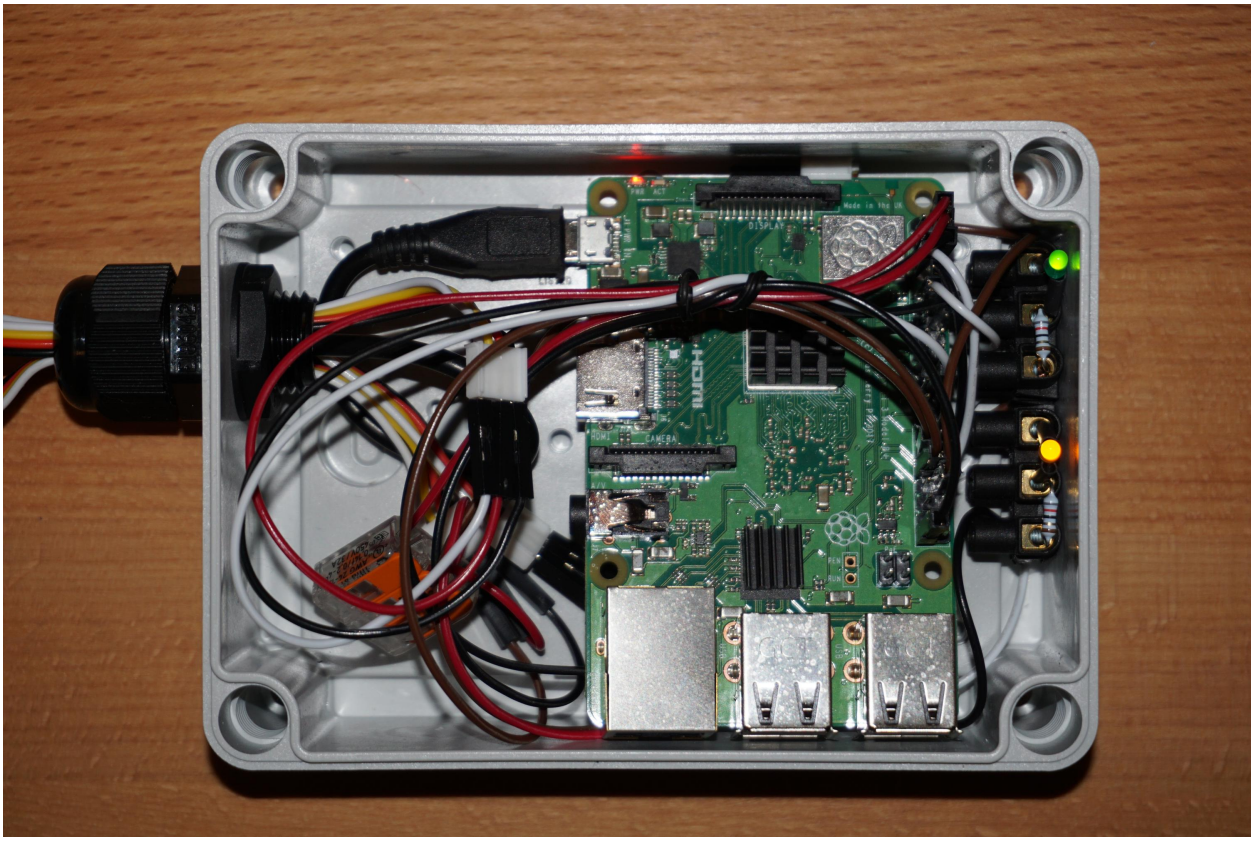
Raspberry GPIO connection:

This I/O configuration can be found on the Python/Raspberry side as *config.json*:

```
{
    "io": {
        "out_ready_led": 4,
        "out_network_status_led": 17,
        "out_open_command": 5,
        "out_close_command": 6,
        "in_door_closed": {"pin": 25, "active_state": true},
        "command_out_pulse_time_s": 2
    }
}
```

1.2.3 Outdoor Case

The Raspberry is enclosed in a case which is waterproof so there will not be any condensed water. It's good enougth to control the relays.



1.2.4 Electrical Considerations

Some notes about the electrical possibilities and limitations of the Raspberry 3+4:

- the GPIO devices must not consume more than 50 mA altogether
- one single GPIO output must not consume more than 16 mA
- Some outputs have a pullup and others have a pulldown behavior. If we choose the wrong behavior the startup or a reboot of the system lets the output relays switching unexpectedly. A pullup output sets the output to high until the application has been started which could be a big problem. For this project this means that the door will open or close on each startup which is an unwanted operation of the door.
- So the relay outputs have to be connected to GPIO 5 and 6 which have a pulldown resistor. A reboot of the system will not lead to an unwanted switch of the relays.
- GPIO power considerations this project:
  - Estimated limiting resistor for testing: (3.3V (measured output) - 1.7V (typical LED voltage))/20mA = 80Ohms
  - Voltage measurement of the used LED (limiting resistor 82Ohms): I=11mA (measured), R=82Ohm => U=R*I=82Ohm*0.011A=0.9V (limiting Resistor)
  - so a limiting resistor of 82 Ohms fits here for both LEDs used.
  - Relay coils: nominal power=0.2W, resistance=125Ohms => I=U/R=3.3V/125Ohms=26.4mA

Note: The coil current consumption is too high (26.4mA instead of 16mA) but it's only for 2 seconds so this should damage nothing.

1.3 MQTT Broker Docker Setup

1.3.1 Docker Guest OS

Docker eclipse-mosquitto

version 1.6, no SSL because of certificates

1.3.2 Host Persistent Directories

Make directory for `/Container/eclipse-mosquitto-16`

Create following directories in it:

```
config
data
docroot
log
```

Create an initial configuration in `config/mosquitto.conf`:

```
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
```

(continues on next page)

(continued from previous page)

```
listener 9001
protocol websockets
http_dir /mosquitto/docroot
```

1.3.3 Container Configuration

Name: eclipse-mosquitto-2020 Command: /usr/sbin/mosquitto -c /mosquitto/config/mosquitto.conf Entrypoint: /docker-entrypoint.sh CPU Limit: 10% Memory Limit: 1200 MB

Container Hostname: mqtt-16

Network, exposed ports:

- for MQTT protocol: QNAP-Host: 1883, Container: 1883, TCP
- for Websockets: QNAP-Host: 9001, Container: 9001, TCP

Shared Folders:

- Volume from host:
  - /Container/eclipse-mosquitto-16/config => /mosquitto/config
  - /Container/eclipse-mosquitto-16/data => /mosquitto/data
  - /Container/eclipse-mosquitto-16/log => /mosquitto/log

1.3.4 HTTP server for the Web App

Note: The MQTT broker could also provide some files via http but is made too simple as the mime types cannot be configured. So JS files will not be interpreted by the browers. Therefore an nginx is used here to provide the web app files.

Docker OS used: offical nginx

Name: nginx-safechicken CPU Limit: 10% Memory Limit: 1200 MB

Container Hostname: nginx-safechicken

Network, exposed ports:

- for MQTT protocol: QNAP-Host: 9000, Container: 80, TCP
- Volume from host:
  - /Container/http-safechicken/docroot => /usr/share/nginx/html

Set directory permissions:

cd /usr/share/nginx/html chown nginx:nginx . chown nginx:nginx -R ./*

1.3.5 MQTT Client for Testing

Proposed client: https://mqtt-explorer.com/

snap install mqtt-explorer

Run it like this: `mqtt-explorer`

Name: <some name> Host: <your Docker host> Port: 1883

1.3.6 MQTT Manual Test

Using mqtt-explorer > Publish json

test1/topic1: {“value1”:11, “array1”: [“val1”, “val2”]}

Select “retain” to store it in a database so you could better test it.

1.4 Raspberry Pi Base Setup

1.4.1 OS Installation

Download the light version of the Raspberry OS from Raspberry Pi: Raspberry Pi OS Lite (size around 440 MB)

Write it on the SD card using some Flash Card Writer Tool (for instance USB Image Tool on Windows, or just “dd” on Linux).

Then boot it and do the base settings directly via keyboard and HDMI screen as SSH is not active yet.

1.4.2 Base Setup

`sudo raspi-config`

- Localisation Options (Timezone is important)
- Interface Options: enable SSH
- change the default password for user *pi*

1.4.3 Disable Bluetooth

`sudo systemctl disable bluetooth`

1.4.4 Disable Serial Console

```
sudo systemctl disable hciuart
sudo apt purge bluez
sudo apt autoremove
```

1.4.5 Copy your SSH public keys

In order to avoid to always enter the Raspberry's user password you could copy the public key. So from your Linux workstation desktop do this:

```
ssh-copy-id pi@<IP of Raspi>
```

So now it should not be necessary to always enter your password on Raspi login or scp copy operations.

1.4.6 Misc

For convinience, the widely used shell aliases could be enabled:

```
nano /home/pi/.bashrc
```

Uncomment the already existing lines so they look like this:

```
# some more ls aliases
alias ll='ls -l'
alias la='ls -A'
alias l='ls -CF'
```

To give out the processor temperature it is recommended to make an alias called `temp`:

```
nano /home/pi/.bash_aliases
```

```
alias temp='/opt/vc/bin/vcgencmd measure_temp'
```

After a re-login you can get the temperature by calling *temp*:

```
pi@raspberrypi:~ $ temp
temp=48.0'C
```

1.5 Application Setup

1.5.1 Copy to Raspi

Use the helper script to copy everything needed to the Raspi:

```
cd path/to/safechicken
raspberry/copy_to_raspi.sh <IP of your Raspberry>
```

This script also installs the package `python3-venv` and installs the virtual environment.

1.5.2 Manual Start

```
cd ~/safechicken
source venv/bin/activate
python3 -m safechicken.main config.json
```

Check the console logs it it's running or not and also the status LEDs.

1.5.3 System Auto Start

The following code is automatically executed in the copy_to_raspi.sh script, but could be done manually on the Raspberry side:

```
sudo cp /home/pi/safechicken/raspberry/safechicken.service /etc/systemd/system/
sudo chmod +x /etc/systemd/system/safechicken.service
sudo systemctl daemon-reload
sudo systemctl enable safechicken.service
```

1.5.4 Safe Chicken Configuration

Set your home coordinates in the *config.json* file otherwise the open/close times according to sunrise/sunset times does not match. Also set the MQTT broker host name according to your home network.

```
{
  "time_control": {
    "latitude": 47.0,
    "longitude": 7.0,
    "minutes_after_sunrise": 15,
    "minutes_after_sunset": 30
  },
  "mqtt_client": {
    "broker_hostname": "192.168.21.5",
    "broker_port": 1883,
    "client_name": "SafeChickenController"
  },
  "topic_conf": {
    "sun_times": "safechicken/sun_times",
    "door_sun_times_conf": "safechicken/door_sun_times_conf",
    "door_prio": "safechicken/door_prio",
    "static_time": "safechicken/static_time",
    "force_operation": "safechicken/force_operation",
    "command_out": "safechicken/command_out",
    "last_commands": "safechicken/last_commands",
    "door_closed_log": "safechicken/door_closed_log",
    "door_lifesign": "safechicken/door_lifesign"
  },
  "controller": {
    "force_time_expire_minutes": 180
  },
  "dispatcher": {
    "start_action_delay": 2
  },
  "io": {
    "out_ready_led": 4,
```

(continues on next page)

(continued from previous page)

```
    "out_network_status_led": 17,
    "out_open_command": 5,
    "out_close_command": 6,
    "in_door_closed": {"pin": 25, "active_state": true},
    "command_out_pulse_time_s": 2
  }
}
```

1.6 Functions

1.6.1 Time Synchronisation

SafeChicken door control only works when the system time is correct and synchronized at least once. This can manually be checked with the command:

```
pi@raspberrypi:~ $ timedatectl show -p NTPSynchronized
NTPSynchronized=yes
```

Or in Python:

```
import subprocess

command = ['timedatectl', 'show', '-p', 'NTPSynchronized']
res = subprocess.check_output(command)

if '=yes' in res.decode("utf-8"):
    print('timesync working')
else:
    print('timesync NOT working')
```

1.6.2 Output Signals

The door open and door close signals are both an **impulse of 2 seconds** each which let the door motor electronics activate its motor on the requested side.

1.7 Raspberry Network Setup

1.7.1 Static IP

Which network interface do you use? Check it with `ip r`, then edit `sudo nano /etc/dhcp/dhclient.conf`.

Example in my network:

```
interface wlan0
static ip_address=192.168.21.12/24
static routers=192.168.21.1
static domain_name_servers=192.168.21.1
```

1.7.2 WLAN

See WLAN setup

`sudo raspi-config`

- Change WLAN country option

If you have two networks in range, you can add the priority option to choose between them. The network in range, with the highest priority, will be the one that is connected.

`sudo nano /etc/wpa_supplicant/wpa_supplicant.conf`

```
network={
    ssid="HomeOneSSID"
    psk="passwordOne"
    priority=1
    id_str="homeOne"
}

network={
    ssid="HomeTwoSSID"
    psk="passwordTwo"
    priority=2
    id_str="homeTwo"
}
```

1.8 Backup and Restore

Here you learn how to backup the flash card containing the operating system and your application in case it's getting lost and you want to restore it later.

Plug in your Raspberry's flash card in a desktop PC, then start a partition manager like *gparted* or *kdepartitionmanager*.

Note: In this example, the flash disk is on **/dev/sdb**, /dev/sdb2 is the main partition.

1.8.1 Prepare Flash Card

First fill the unused space with zeros before you want to compress it.

```
sudo mkdir -p /mnt/work
sudo mount /dev/sdb2 /mnt/work
sudo dd if=/dev/zero of=/mnt/work/zero bs=1M
sudo rm /mnt/work/zero
```

Then shrink the partition to a minimum:

- First unmount the partition `sudo umount /mnt/work`
- start the partition manager
- in my example: shrink sdb2 from 16/32GB flash card size to around 2 GB

1.8.2 Backup on Linux

Write the flash disk into a file; use gzip which compresses enough (using the compressor *pigz*).

```
sudo umount /mnt/work
time sudo bash -c "dd if=/dev/sdb bs=1M | pigz -9 > safechicken_16gb_image_raspberry_
↪$(date -I).dd.gz"
# size: around 623 MiB
sudo chown ${USER}:${USER} safechicken*
```

1.8.3 Restore on Linux

Warning: This may overwrite your harddisk if you choose the wrong disk! Here, it is **/dev/sdb**.

```
time sudo bash -c "pigz -dc safechicken_16gb_image_raspberry_DATE.dd.gz | dd of=/dev/
↪sdb bs=1M"
```

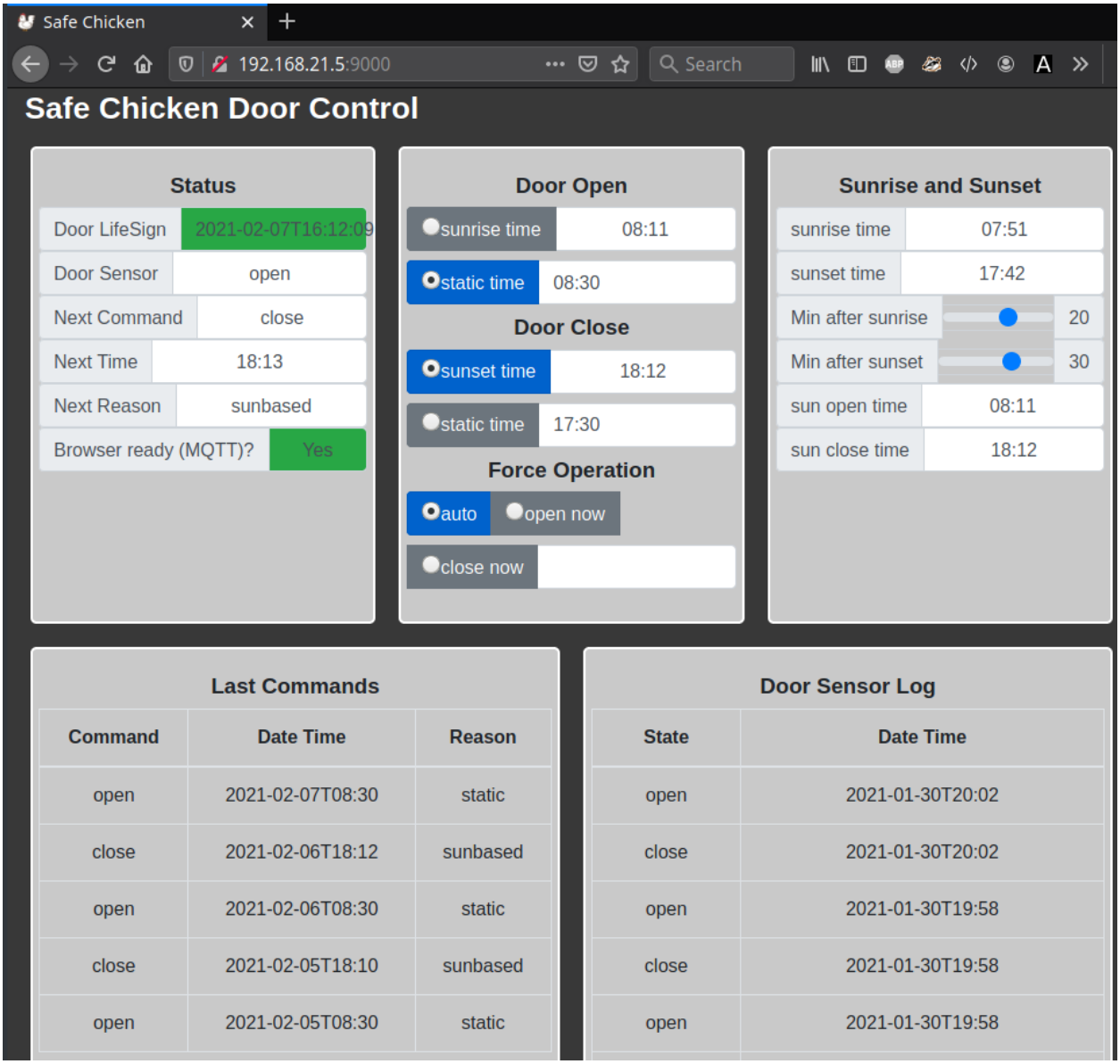
1.9 Web Application

The web application's functions are generally described in the index and overview pages already.

Here are some technical details:

- The Paho MQTT JavaScript client communicates to the MQTT broker running on the NAS
- Bootstrap UI components
- self-created simple communication adapter between MQTT and bootstrap html components

As the web app uses bootstrap containers it's responsive and looks good on multiple devices. On a desktop it looks like this:


Safe Chicken
192.168.21.5:9000
Search
Safe Chicken Door Control
Status
Door LifeSign
2021-02-07T16:12:09
Door Sensor
open
Next Command
close
Next Time
18:13
Next Reason
sunbased
Browser ready (MQTT)?
Yes
Door Open
sunrise time
08:11
static time
08:30
Door Close
sunset time
18:12
static time
17:30
Force Operation
auto
open now
close now
Sunrise and Sunset
sunrise time
07:51
sunset time
17:42
Min after sunrise
20
Min after sunset
30
sun open time
08:11
sun close time
18:12
Last Commands
Command
Date Time
Reason
open
2021-02-07T08:30
static
close
2021-02-06T18:12
sunbased
open
2021-02-06T08:30
static
close
2021-02-05T18:10
sunbased
open
2021-02-05T08:30
static
Door Sensor Log
State
Date Time
open
2021-01-30T20:02
close
2021-01-30T20:02
open
2021-01-30T19:58
close
2021-01-30T19:58
open
2021-01-30T19:58

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search