
Controle des lumières KNX

last modified by Forestier Robin

on 2022/02/22 08:19

- [1.0 - But](#)
- [2.0 - Liens](#)
- [3.0 - Composants](#)
- [4.0 - Schéma Bloc](#)
- [5.0 - Shield Opto](#)
 - [5.1 - Shéma Shield Opto](#)
 - [5.2 - PCB Shield Opto](#)
- [6.0 - GUI](#)
 - [6.1 - Utilisateurs](#)
 - [6.2 - Pages](#)
 - [6.2.1 - Login](#)
 - [6.2.2 - page1](#)
 - [6.2.3 page 2](#)
 - [6.2.4 - Settings](#)
 - [6.2.5 - téléphone](#)
- [7.0 - Programmation](#)
 - [7.1 - Code Python](#)
 - [7.1.1 - Config](#)
 - [7.2 - HTML](#)
 - [7.3 - CSS](#)
- [8.0 - Instalation du Raspberry](#)
 - [8.1 - Instalation](#)
 - [8.1.1 - Image](#)
 - [8.1.2 - Git clone](#)
 - [8.1.3 - Configuration](#)
 - [8.1.4 - Configuration Gunicorn](#)
 - [8.1.5 - Création du certificat SSL](#)
 - [8.1.6 - Configuration NGINX](#)
 - [8.1.7 - Lancement Navigateur](#)
 - [9.0 - Sauvgarde de l'image](#)
- [9.0 - Câblage](#)
- [10.0 - Mécanique](#)
 - [10.1 - Fixation](#)

1.0 - But

Le but de ce projet est de réaliser une interface graphique pour le contrôle des lumières de l'atelier.
Un écran Raspberry Pi 7" Touchscreen Display servira de contrôleur principal, directement branché sur un Raspberry Pi 3b+.
Une interface à l'aide d'optocoupleurs est utilisée et directement contrôlée par les GPIO du Raspberry.
Pour le contrôle des lumières, nous utilisons deux types de module, trois US/U 4.2 de ABB et deux SA/S 4.16.2.2.

2.0 - Liens

[Gitlab](#)
[ABB - US/U 4.2](#)
[ABB - SA/S 4.16.2.2](#)
[Explication KNX](#)

3.0 - Composants

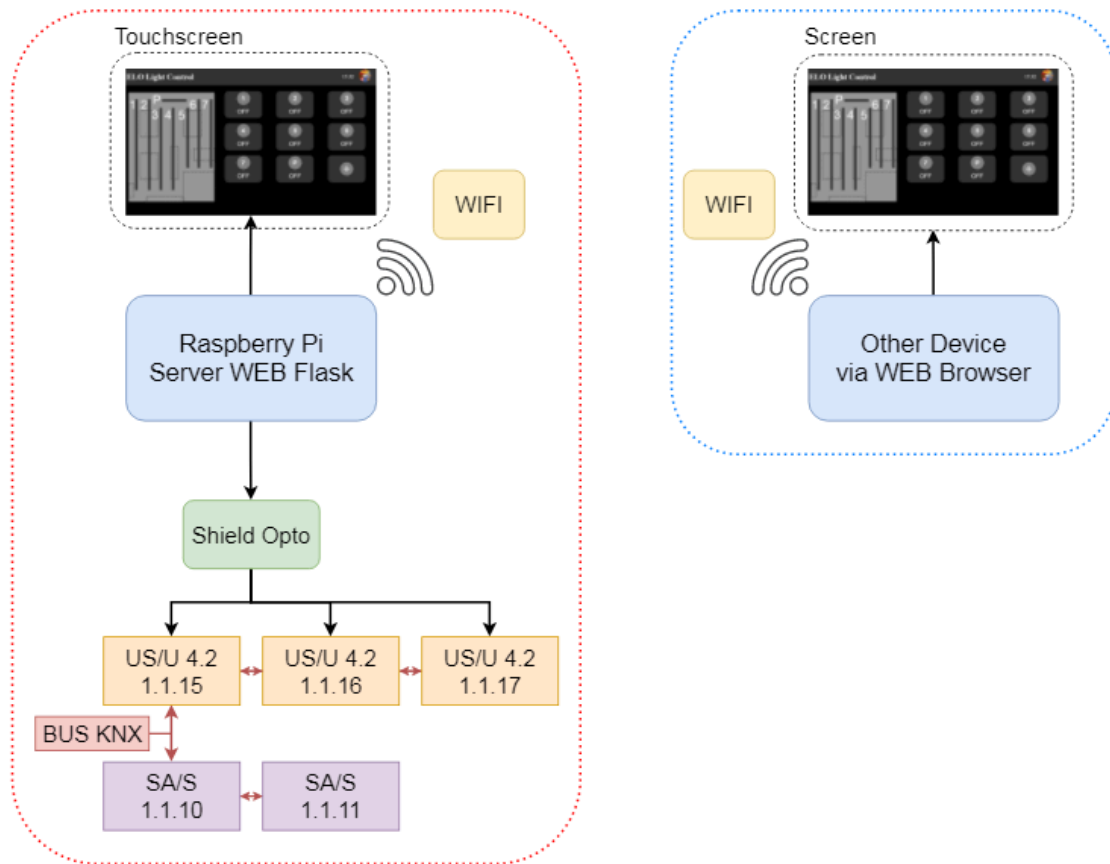
Pour le projet :

Nom	Nombre	Prix unité	Prix total
Raspberry Pi 3B+	1	40.90	40.90
Raspberry Pi 7" Touchscreen	1	74.90	74.90
US/U 4.2	3	83.80	251.40
SA/S 4.16.2.2	2	169.60	339.20
TBLC 25-105	1	52.00	52.00
Fibox ARCA 403015	1	96.90	96.90
Total			857.30

Pour la carte Shield Opto :

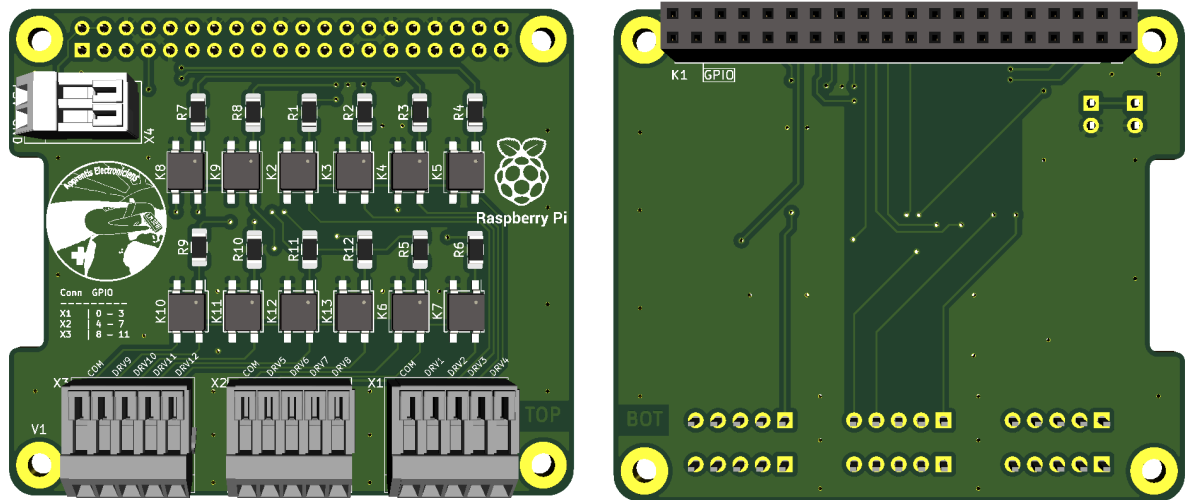
Reference	Value	Prix unité	Prix total
K2-K13	ASSR-1218	1.75	21
x1-x3	WAGO 218-505	3.25	9.75
x4	WAGO 218-502	1.55	1.55
Total			32.30

4.0 - Schéma Bloc



Comme on peut le voir sur ce schéma block, on utilise plusieurs systèmes de communication. Les Gpio du Raspberry contrôlent directement le shield opto puis les US/U 4.2. Les SU/U 4.2 communiquent avec les SA/S en KNX (www.knx.ch). Et les autres appareils communiquent en TCP via le wifi.

5.2 - PCB Shield Opto



6.0 - GUI

Pour la partie graphique, j'ai tout d'abord essayé avec la librairie Python TKinter.

<https://docs.python.org/fr/3/library/tkinter.html>

Les versions de test se trouvent sur le gitlab sous 5_Programmation.

Mais pour simplifier le tout et pouvoir dans le futur ajouter une partie réseau à ce projet, j'ai donc choisi d'héberger un site web. En utilisant Flask sur Python, je peux donc facilement héberger le site sur mon Raspberry pi et me connecter dessus simplement en connaissant l'adresse IP.

6.1 - Utilisateurs

Pour assurer la sécurité du site, il existe 3 utilisateurs différents.

1: elo

L'utilisateur elo peut, s'il connaît le mot de passe, allumer ou éteindre les lumières sans pouvoir accéder aux paramètres.

2: admin

L'utilisateur admin a tous les accès.

3: local

Cet utilisateur est réservé au Raspberry pi (server).

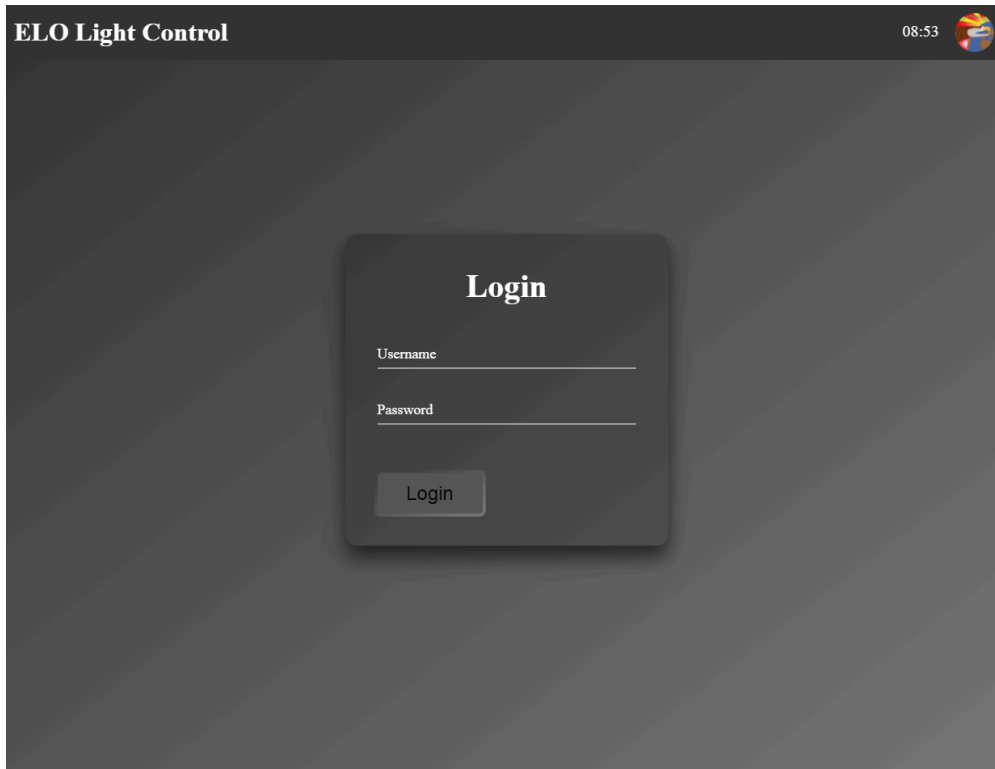
L'utilisateur local n'a pas besoin de se connecter pour accéder au site, son authentification se fait à l'aide de son IP. Il ne peut pas accéder aux réglages de l'application.

6.2 - Pages

Le site est constitué de 4 pages distinctes.

6.2.1 - Login

La page de login permet de se connecter via un périphérique autre que l'écran tactile.



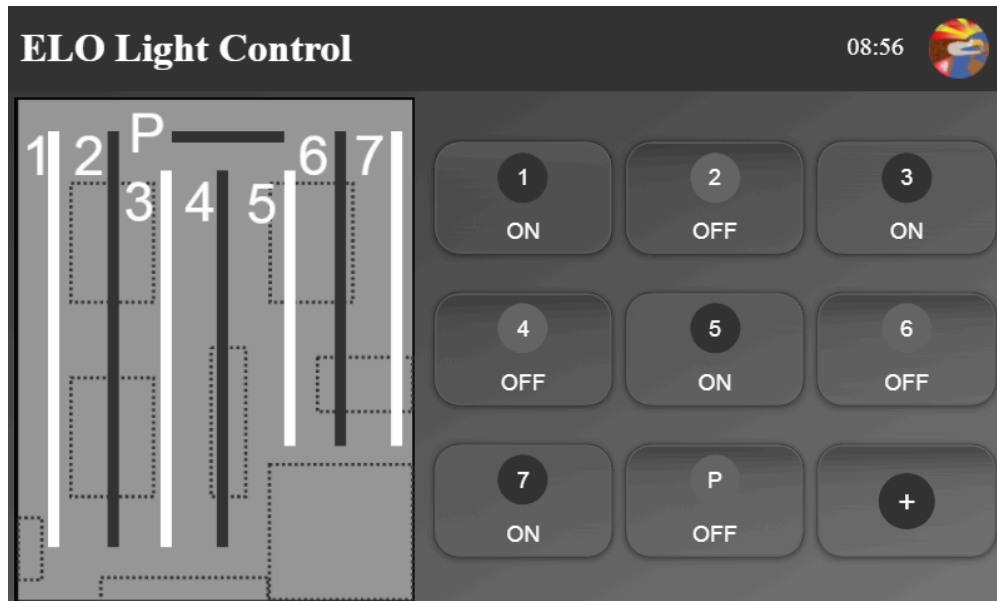
The screenshot shows a web interface for 'ELO Light Control'. The title bar at the top left displays 'ELO Light Control' and the top right shows the time '08:53' next to a small circular icon. The main content area has a dark gray background with a subtle diagonal line pattern. In the center, there is a dark gray rounded rectangle containing the 'Login' form. The form includes the title 'Login' in white, followed by two input fields labeled 'Username' and 'Password' in a light gray font. Below these fields is a 'Login' button with a light gray background and dark gray text.

6.2.2 - page1

La page 1 contient à gauche un plan de l'atelier qui se modifie selon les lumières allumées.

À droite, 9 boutons permettent d'allumer séparément chaque rail de lumières. Le bouton en bas à droite permet de se rendre sur la page 2.

Le logo ELO en haut à droite permet d'accéder aux réglages (admin only).



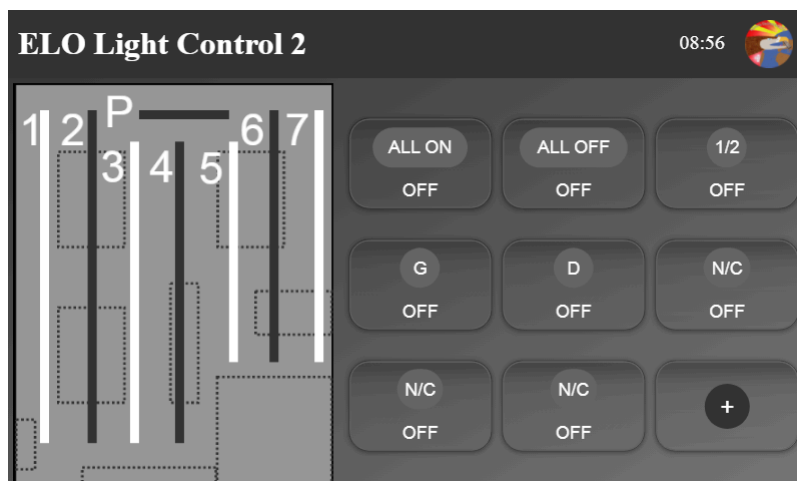
6.2.3 page 2

La page 2 est une copie de la page 1 avec comme modification l'action des boutons de droite.

On y retrouve:

- All ON | tout allumer
- ALL OFF | tout éteindre
- 1/2 | allumer 1 sur 2
- G | allumer la gauche de l'atelier
- D | allumer la droite de l'atelier
- N/C | non connecté (réserver pour usage futur)

Le bouton en bas à droite permet de se rendre sur la page 1.

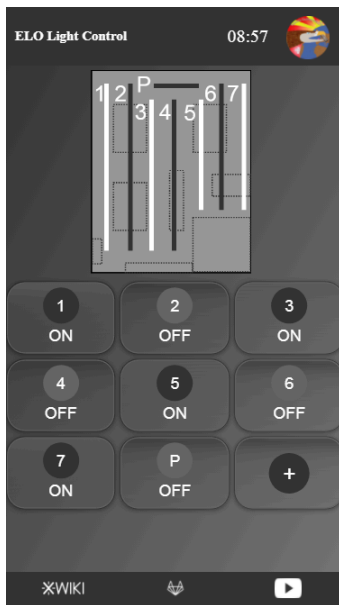


6.2.4 - Settings

La page settings accessible uniquement par le compte admin permettra de régler les paramètres importants.

6.2.5 - téléphone

J'ai aussi réalisé une mise en page pour téléphone.



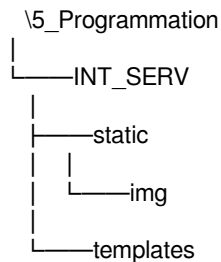
7.0 - Programmation

Pour la programmation de cette interface, j'ai utilisé python. Plus précisément flask.

<https://flask.palletsprojects.com>

Flask est un micro framework open-source de développement web en Python. (Wikipedia)

Avec flask, je peux facilement héberger un server web. Il suffit de créer les fichiers: static & templates.



Le programme python se trouve dans le dossier principal (INT_SERV).

Les fichiers HTML se trouvent dans templates.

Les fichiers CSS se trouvent dans static.

Les images se trouvent dans static\img.

On retrouve aussi le fichier config.csv dans le dossier principal.

7.1 - Code Python

Pour ce code j'ai besoin de ces bibliothèques.

```
from flask import Flask, render_template, request, session, url_for, redirect, flash, abort
from flask_sqlalchemy import SQLAlchemy
from flask_session import Session
import time
import datetime
import threading
import csv
import os
import logging
from werkzeug.exceptions import HTTPException
```

Quand le Raspberry Pi avec l'écran veut se logger sur le site, on veut que cela soit fait automatiquement. Pour le faire, j'ai créé une liste des IP autorisées.

```
authorize_ip = ["localhost", "127.0.0.1", "172.16.32.133"]
```

Pour pouvoir créer une session pour chaque utilisateur se connectant sur le site, j'ai créé une class User.

```
class User:
    """User : creat different user for the login"""
    def __init__(self, id, username, password):
        self.id = id
        self.username = username
        self.password = password
```

Il y a 3 sortes d'utilisateur.

- 1 - le raspberry pi en local, **username = local**.
- 2 - un utilisateur, **username = elo**.
- 3 - un admin, **username = admin**.

```
users.append(User(id=1, username='elo', password='elo'))
users.append(User(id=2, username='admin', password='admin'))
users.append(User(id=3, username='local', password='local'))
```

Chaque page web a sa propre fonction pour pouvoir interagir avec les request. Dans l'ordre, login, page1, page2 et settings.

```
@app.route("/login", methods=['POST', 'GET'])
@app.route("/", methods=['POST', 'GET'])
def login():
    @app.route("/page1", methods = ['POST', 'GET'])
    def page1():
        @app.route("/page2", methods = ['POST', 'GET'])
        def page2():
            @app.route("/settings", methods = ['POST', 'GET'])
            def settings():
```

Pour que l'utilisateur puisse voir quelque chose, il faut lui retourner la page. On va donc retourner le fichier HTML correspondant à l'aide de la fonction `render_template()`. On ajoute aussi toutes les variables qui seront affichées sur la page come l'heure (time).

```
return render_template('page1.html',
                        button=buttonSts_p1,
                        color=color,
                        time=current_time,
                        warning=warning)
```

Une dernière fonction est la fonction **activate_job()**.
Cette fonction s'exécute au moment où la première personne essaie de se connecter au site.
Un thread est créé pour réaliser toutes les autres activités comme: les automatisations ou le warning de température.

```
@app.before_first_request
def activate_job():
    """activate_job : fonction to run job on background of the web server (with thread)"""
    def run_job():
        """
        run_job : run background job (run every minute).
        Auto - on/off
        Check temperature of the Raspberry Pi
        """
        thread = threading.Thread(target=run_job)
        thread.start()
```

Et pour finir le **main**.

```
if __name__ == "__main__":
    logging.info(" *** Strating server *** ")
    app.run(host='0.0.0.0', port=80, debug=False)
    GPIO.cleanup()
```

7.1.1 - Config

Le fichier **config.csv** contient les informations pour la page settings.
Ces informations sont stockées en externe du programme pour assurer qu'elles soient sauvgardées même après un redémarrage.
Pour le moment, les seules informations stockées sont: Auto on et Auto off.
Ce qui correspond à l'allumage automatique des lumières à une certaine heure.

Voici son contenu :

name	state	param1
Auto on	on	07:30
Auto off	on	18:00

7.2 - HTML

Les fichiers HTML se trouvent sous `\5_programmation\nom_de_version\templates`
On retrouve évidemment un fichier HTML par page.

Détaillons le fichier **page1.html**.

Premièrement, j'utilise une balise *meta* pour venir reload automatiquement ma page.
Ceci permet, en cas de modification d'un autre utilisateur, d'afficher ses dites modification.

```
<meta http-equiv="refresh" content="60">
```

Dans le header, on peut voir l'heure affichée à gauche.
Pour le faire, dans mon code python, je viens ajouter *time* quand j'appelle la fonction *render_template()*.
Et pour l'afficher sur ma page web, il suffit d'écrire `{{ time }}`

```
<div class="header">
  <a href="/settings">  </a>
  <p> {{ time }} </p>
<p class="warning"> {{ warning }} </p>
<h1>ELO Light Control</h1>
</div>
```

Vient ensuite le plan de l'atelier. Ce plan est une image où je viens simplement dessiner les néons allumés.
Le script récupère la couleur des différents néons et crée un rectangle à la bonne place.

```
<div class="row" id="row">
  <div class="column-1">
    
    <canvas id="myCanvas" width="356px" height="451px">
      <script>
        window.onload = function() {
          var c = document.getElementById("myCanvas");
          var ctx = c.getContext("2d");
          var img = document.getElementById("plan");
          ctx.drawImage(img, 0, 0);
          ctx.fillStyle = "{{ color[0] }}";
          ctx.fillRect(0, 30, 10, 370);
          (...)
        }
      </script>
    <canvas>
    <noscript>
      <p style="color:red">JavaScript is Disable on you brother !</p>
      <p style="color:red">To have the best use of this site, please turn it on.</p>
    </noscript>
  </div>
```

Pour finir, il reste à afficher les boutons.
Pour cela j'ai réalisés une boucle *for*.
Il faut utiliser Jinja: <https://jinja.palletsprojects.com/en/3.0.x/templates/>

Puis on crée le dernier bouton pour accéder à la page suivante.
Il faut aussi créer deux faux texte pour qu'il agisse correctement avec les règles css.

```
<form method="post" action="/page1" class="column-r" id="grid">
  {% for bt in button %}
    <div class="button_num">
      <button class="btn" id={{ bt }} type="submit" name="button_p1" value={{ loop.index }}>
        {% if loop.index == 8 %}
          <span class="text_num" id={{ bt }}> P </span>
        {% else %}
          <span class="text_num" id={{ bt }}> {{ loop.index }} </span>
        {% endif %}
        {% if bt == "off" %}
          <p class="text_info"> OFF </p>
        {% else %}
          <p class="text_info"> ON </p>
        {% endif %}
      <button>
    </div>
  {% endfor %}
  <div class="button_num">
    <button class="btn" id="off" type="submit" name="button_p1" value="page_2">
      <span class="page_next"> + </span>
      <p class="text_on" style="visibility: hidden"> OFF </p>
    <button>
  </div>
</form>
```

7.3 - CSS

Les fichiers CSS se trouvent dans \5_programmation\nom_de_version\static.
Il y a 3 fichiers css.

style.css - utilisé pour les pages: page1 et page2.
login.css - utilisé pour la page: login.
settings.css - utilisé pour la page: settings.

Il y a quelques points importants que je vais expliquer ci-dessous.

Pour l'affichage du plan à côté des boutons, j'utilise deux colonnes (l et r).

```
.column-r {  
  position: relative;  
  width: auto;  
  height: auto;  
  padding: 10px;  
  padding-top: 5%;  
}  
.column-l {  
  float: left;  
  width: auto;  
  height: 100%;  
  padding: 5px;  
  margin-top: 20px;  
  margin-left: 5px;  
}
```

Puis pour afficher les 9 boutons, j'utilise une grille.

```
#grid {  
  display: grid;  
  column-gap: 10px;  
  row-gap: 30px;  
  grid-template-columns: repeat(3, 1fr);  
}
```

Pour avoir un meilleur affichage sur l'écran du Raspberry, j'ai créé une partie du css exprès.
@media all and (max-width: 800px) and (max-height: 480px) and (min-width: 799px)

Et j'ai aussi créé une partie pour les téléphones portables.
@media only screen and (hover: none) and (pointer: coarse) and (orientation: portrait)

8.0 - Instalation du Raspberry

8.1 - Instalation

Pour l'installation du Raspberry et de la partie logicielle, vous pouvez soit, copier l'image de mon Raspberry et repartir de là où j'ai laissé le projet ou repartir de zéro.

Commençons avec la première méthode.

Pour cela rendez-vous sur le OMVSERVER et sur : `formation/project/ControleDesLumiersKNX`, récupérer l'image "lightcontrol.img" et copiez la sur votre bureau.

Avec le logiciel Raspberry Pi Imager installer l'image sur une carte micro sd (≥ 16 Gb).

Pour vous assurer que vous avez la dernière version du programme, il faut réaliser un pull du GitLab.

```
cd controle-des-lumieres-knx
git fetch
git pull
sudo systemctl restart nginx
```

Si vous n'avez aucune erreur, redémarrez votre Raspberry et votre installation sera terminée.

Si vous avez une erreur au git pull, faites simplement : `git checkout`.

Si vous avez d'autres erreurs testez :

```
sudo systemctl status lightcontrol
sudo nginx -t
```

8.1.1 - Image

Si vous souhaitez réaliser une installation depuis le départ, voici la marche à suivre.

Pour faire fonctionner votre Raspberry, il vous faudra une image.

Vous pouvez simplement télécharger Raspberry Pi Imager.

<https://www.raspberrypi.com/software/>

Ou installer la version se trouvant sur le OMVSERVER.

OMVSERVER\formation\Projets\ControleDesLumiersKNX\raspberry_imager_1.7.1.exe

Installez-le sur votre PC.

Sur l'interface, choisissez l'image de votre choix puis votre carte micro SD.

Vous pouvez aussi déjà configurer votre Raspberry en cliquant sur la roue dentée en bas à droite.

Quand tout est bon, il suffira de cliquer sur écrire.

Vous pouvez maintenant lancer votre Raspberry pi avec sa nouvelle image.

Commencez par le configurer et le connecter au wifi.

Vous pouvez installer **Xscreen saver** pour désactiver la mise en veille.

```
sudo apt-get install xscreensaver
```

Puis dans l'app, choisir désactiver la mise en veille.

Vous pouvez aussi activer VNC et le SSH pour vous faciliter la vie.

```
sudo raspi-config
```

8.1.2 - Git clone

Pour cloner le gitlab, copiez cette commande dans votre terminal.

(vérifiez s'il n'a pas été déplacé sous Projets_Ancien_Apprentis)

```
git clone http://172.16.32.230/Forestier/controle-des-lumieres-knx
```

Vous trouverez ensuite le dossier de la dernière version du server sous *5_Programmation/server* .

8.1.3 - Configuration

Tout d'abord, on va installer quelques outils de base.

```
sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev python3-setuptools
```

Maintenant, on va se rendre dans notre dossier server puis installer tous les "requirements".

```
cd controle-des-lumieres-knx/5_Programmation/server  
source serverenv/bin/activate  
pip install -r requirements.txt  
deactivate
```

Vous pouvez tester le server en exécutant le flask_test.py dans le dossier server/test.

```
cd controle-des-lumieres-knx/5_Programmation/server/test  
sudo python3 flask_test.py
```

Pour s'y connecter, tapez l'ip de votre Raspberry dans la barre de recherche de votre navigateur.
Si vous ne connaissez pas l'ip de votre Raspberry, tapez "ifconfig" dans votre terminal.

Vous devriez voir une page blanche avec marqué "Hello World".

Pour que le server WEB se lance automatiquement au démarrage du Raspberry Pi et pour que le server ne soit plus un serveur de développement, j'ai choisi d'utiliser un server Gunicorn, et le proxy Nginx.

Pour l'installation vous pouvez suivre ce tuto :

<https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-18-04>

(/1_Documentation/Flask_server_Gunicorn_DigitalOcean.pdf)

Mais je vais quand même expliquer comment je l'ai fait.

Si tout va bien vous avez normalement dans le dossier server un fichier WSGI.py et un fichier lightcontrol.services.

8.1.4 - Configuration Gunicorn

Commencons la configuration du server. La première chose sera de configurer Gunicorn pour que le server se lance automatiquement.

Pour cela, on va créer le fichier *lightcontrol.service*.

sudo nano /etc/systemd/system/lightcontrol.service

Puis on y écrit ceci :

```
[Unit]
Description=Gunicorn instance to serve ELO light control
After=network.target
[Service]
User=pi
Group=www-data
WorkingDirectory=/home/pi/controle-des-lumieres-knx/5_Programmation/server
Environment="PATH=/home/pi/controle-des-lumieres-knx/5_Programmation/server/serverenv/bin"
ExecStart=/home/pi/controle-des-lumieres-knx/5_Programmation/server/serverenv/bin/gunicorn --
workers 1 --bind unix:lightcontrol.sock -m 007 wsgi:app
[Install]
WantedBy=multi-user.target
```

Unit : Description du project

Service : On indique sous quel utilisateur on va utiliser le server et où se trouvent les dossiers.

Install : Pour activer le démarrage automatique.

Vous pouvez fermer et enregistrer le fichier.
(ctrl + x puis o puis enter)

On va maintenant démarrer Gunicorn.

sudo systemctl start lightcontrol

On l'autorise à se lancer au démarrage.

sudo systemctl enable lightcontrol

On va "checker" le status.

sudo systemctl status lightcontrol

Vous devriez voir:

```
● lightcontrol.service - Gunicorn instance to serve ELO light control
   Loaded: loaded (/etc/systemd/system/lightcontrol.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-02-15 09:41:00 CET; 45min ago
     Main PID: 515 (gunicorn)
       Tasks: 3 (limit: 2059)
    CGroup: /system.slice/lightcontrol.service
           ┌─ 515 /home/pi/controle-des-lumieres-knx/5_Programmation/server/serverenv/bin/python3
           └─ 1231 /home/pi/controle-des-lumieres-knx/5_Programmation/server/serverenv/bin/python3
```

Si vous avez une erreur à ce niveau, corrigez-la avant de poursuivre la configuration.

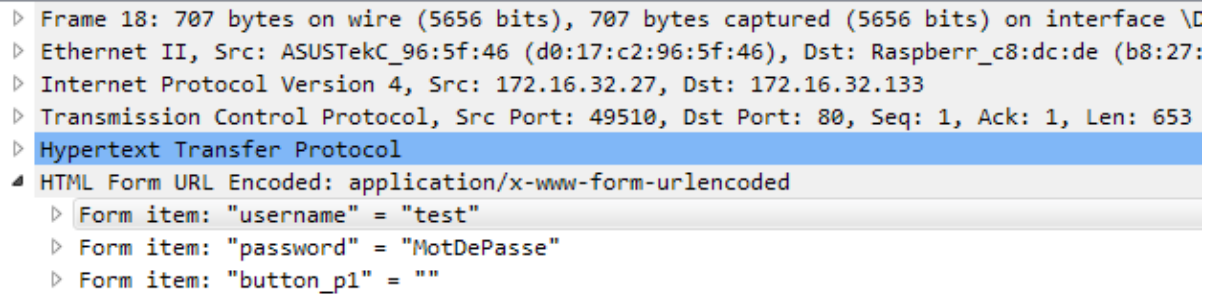
8.1.5 - Création du certificat SSL

Vous êtes maintenant avec un serveur http fonctionnel grâce à Unicorn.

Mais le problème reste qu'en terme de sécurité, l'http ne crypte pas les communications.

La preuve, en utilisant WireShark (www.wireshark.org) pour "sniffer" les communication entre mon Pc et le Raspberry (server).

Voici ce que j'ai obtenu:



```

▶ Frame 18: 707 bytes on wire (5656 bits), 707 bytes captured (5656 bits) on interface \D
▶ Ethernet II, Src: ASUSTekC_96:5f:46 (d0:17:c2:96:5f:46), Dst: Raspberr_c8:dc:de (b8:27:
▶ Internet Protocol Version 4, Src: 172.16.32.27, Dst: 172.16.32.133
▶ Transmission Control Protocol, Src Port: 49510, Dst Port: 80, Seq: 1, Ack: 1, Len: 653
▶ Hypertext Transfer Protocol
▶ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▶ Form item: "username" = "test"
  ▶ Form item: "password" = "MotDePasse"
  ▶ Form item: "button_p1" = ""
```

On peut clairement voir toutes les informations de mon login.

Pour palier à cela, on va passer notre serveur en HTTPS.

Il nous faudra un "Secure Socket Layer" ou SSL certificate, cela nous permettra de sécuriser toutes les transactions entre le serveur et le PC.

Normalement, une clé SSL est payante et fonctionne avec un nom de domaine mais travaillant sans nom de domaine, je vais simplement utiliser openssl (<https://www.openssl.org>).

Commencez par installer nginx.

sudo apt install nginx

Ensuite on va créer le dossier ssl.

sudo mkdir /etc/nginx/ssl/

cd /etc/nginx/ssl/

Puis, on va générer la clef privée RSA:

sudo openssl genrsa -des3 -out server.key 2048

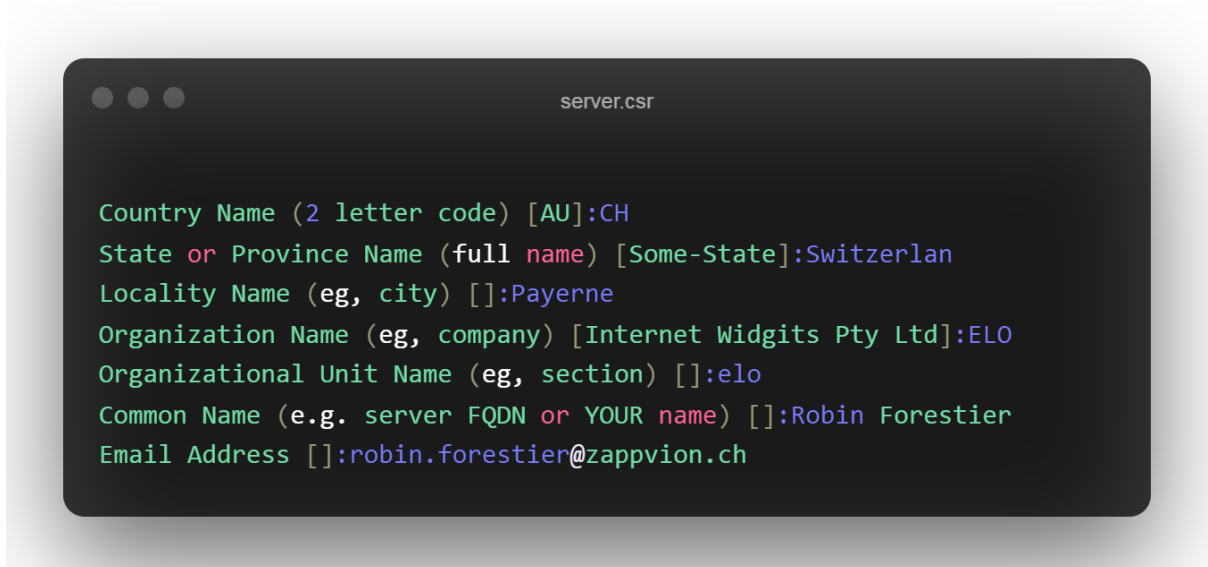
Entrez une phrase (plus longue qu'un simple mot de passe) et répétez-la pour confirmer.

Rien ne s'affiche à l'écran lorsque vous tapez. C'est normal.

Après quelques secondes, on obtient le fichier *server.key*.

A partir de la clef privée RSA, on va maintenant générer un *CSR* (Certificate Signing Request). C'est-à-dire un formulaire de demande de certificat.

```
sudo openssl req -new -key server.key -out server.csr
```



La seule indication importante est votre nom de domaine "Common Name".

domaine.tld protégera tous les sous-domaines. sousdomaine.domaine.tld ne protégera que le sous domaine mentionné.

Laissez les « extra attributes » vides.

Voilà, la demande de certificat `server.csr` est créée. Il ne reste plus qu'à la valider. Normalement, c'est une autorité de certification (CA) qui doit signer votre CSR. Elle se porte ainsi garante de l'authenticité des données qui se trouvent dans le certificat. Malheureusement, ce service est souvent payant.

Pour contourner le « problème », nous allons nous ériger en CA et signer nous-même le CSR. On commence par archiver notre clé privée :

```
sudo cp server.key server.key.tmp
```

Puis, on va décrypter la clé privée pour permettre à Nginx de s'en servir sans avoir besoin du mot de passe :

```
sudo openssl rsa -in server.key.tmp -out server.key
```

On entre la phrase choisie précédemment. Puis, on signe le CSR avec notre nouvelle clé privée décryptée :

```
sudo openssl x509 -req -days 1000 -in server.csr -signkey server.key -out server.crt
```

8.1.6 - Configuration NGINX

On va maintenant configurer Nginx comme proxy pour votre server Unicorn.

sudo nano /etc/nginx/sites-available/lightcontrol

Voici ma configuration :
(ne pas oublier de modifier l'ip)

```
server {
    listen 80;
    server_name localhost;
    location / {
        include proxy_params;
        proxy_pass http://unix:/home/pi/controle-des-lumieres-knx/5_Programmation/server/lightcontrol.sock;
    }
}
server {
    listen 80;
    server_name 172.16.32.133;
    location / {
        include proxy_params;
        proxy_pass http://unix:/home/pi/controle-des-lumieres-knx/5_Programmation/server/lightcontrol.sock;
    }
}
server {
    listen 443 ssl;
    server_name 172.16.32.133;
    ssl on;
    ssl_certificate /etc/nginx/ssl/server.crt;
    ssl_certificate_key /etc/nginx/ssl/server.key;
    location / {
        include proxy_params;
        proxy_pass http://unix:/home/pi/controle-des-lumieres-knx/5_Programmation/server/lightcontrol.sock;
    }
}
```

On va créer un lien vers sites-enabled.

sudo ln -s /etc/nginx/sites-available/lightcontrol /etc/nginx/sites-enabled

Avec ce lien, vous pouvez vérifier s'il n'y a pas d'erreur de syntaxe.

sudo nginx -t

Si ça ne vous renvoie aucune erreur, vous pouvez redémarrer nginx.

sudo systemctl restart nginx

Vous pouvez maintenant taper l'ip de votre Raspberry dans la barre de recherche de votre navigateur et votre site devrait apparaître.

Si une page avec marqué "Welcome to nginx!" apparaît, il vous faudra encore supprimer le fichier index de nginx.

sudo rm /var/www/html/index.nginx-debian.html

8.1.7 - Lancement Navigateur

Et vous avez presque fini. Il faut encore que votre navigateur se lance automatiquement au démarrage. Pour cela, modifiez le fichier autostart :

```
sudo nano /etc/xdg/lxsession/LXDE-pi/autostart
```

Et rajoutez cette ligne en indiquant l'ip du Raspberry Pi.

```
@chromium --kiosk http://172.16.32.133
```

Vous pouvez ensuite redémarrer votre Raspberry et tout est normalement fonctionnel.

9.0 - Sauvgarde de l'image

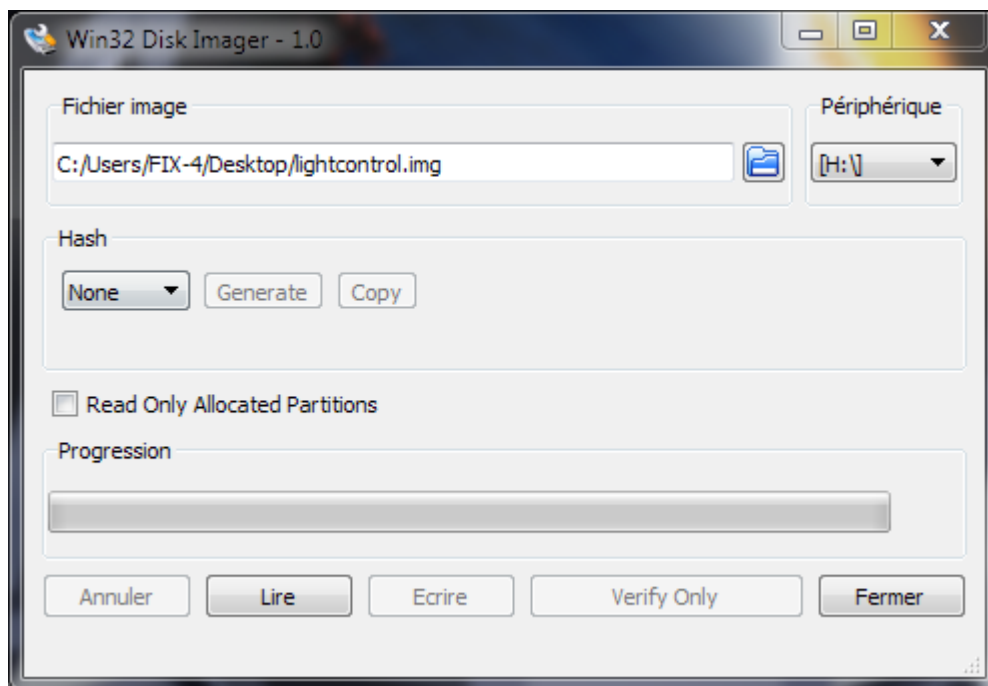
Si vous réalisez des modifications importantes sur le Raspberry (autres que dans le fichier sur le git).

Vous devriez sauvgarder votre version de votre image sur le OMVSERVER.

Pour cela, il vous faudra le logiciel Win32 Disk Imager.

Banchez la carte SD sur votre PC. Dans l'interface indiquez l'emplacement où ira se stocker votre image et son nom.

Sélectionnez le périphérique, puis cliquez sur "Lire". Il faudra attendre que la copie se face, puis votre image sera copiée.



ref: OMVSERVER\formation\Projets\ControleDesLumiersKNX\lightcontrol.img

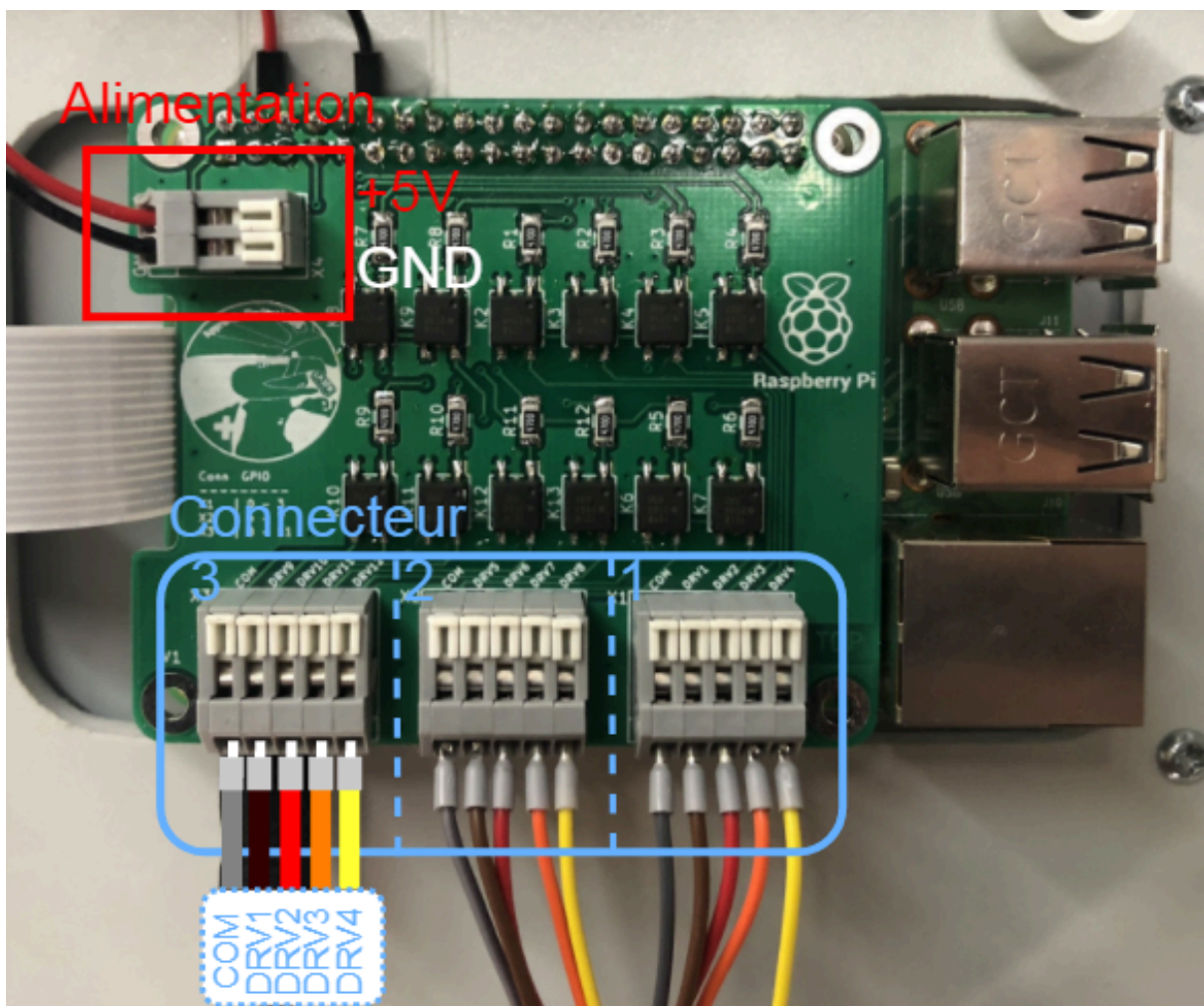
9.0 - Câblage

Pour le câblage, il faut brancher l'alimentation de l'écran sur le connecteur Wago X4 à gauche de la carte. Les US/U 4.2 se branchent dans les connecteurs X1 - X3. Chaque connecteur comporte 5 pôles.

COM	DRV1	DRV2	DRV3	DRV4
-----	------	------	------	------

Les connecteurs sont connectés aux GPIO suivants:

Connecteur	GPIO
X1	0 - 3
X2	4 - 7
X3	8 - 11

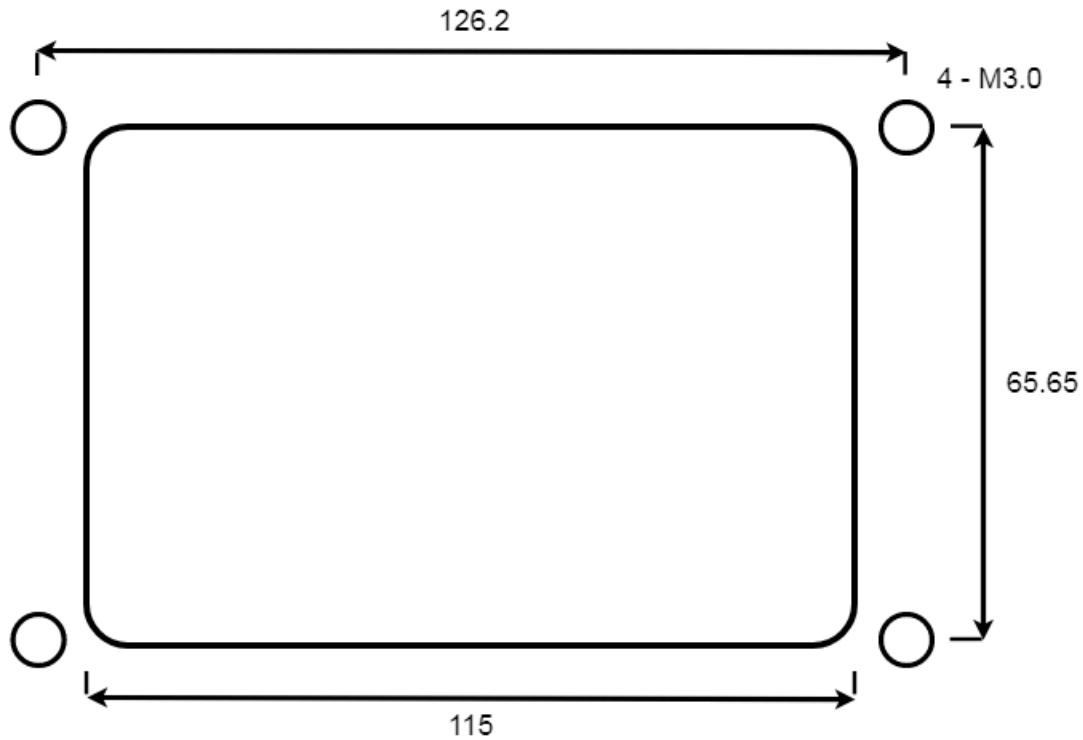


10.0 - Mécanique

Pour placer l'écran sur la porte du boîtier Fibox, j'ai percé 4 trous et réalisé une fenêtre carrée pour que le Raspberry puisse y passer.

ref: /1_Documentation/7-inch-display-mechanical-drawing.pdf &

/1_Documentation/raspberry-pi-3-b-plus-mechanical-drawing.pdf on GitLab.



10.1 - Fixation

Pour la fixation à l'intérieur du boîtier, j'ai utilisé trois rails DIN 35mm.

Sur le premier rail, j'ai placé les trois US/U 4.2 et l'alimentation 5V (TBLC 25-105) pour le Raspberry Pi.

Le deuxième rail sera utilisé par les électriciens.

Sur le troisième rail, il y a les deux boîtiers SA/S 4.16.2.2.

