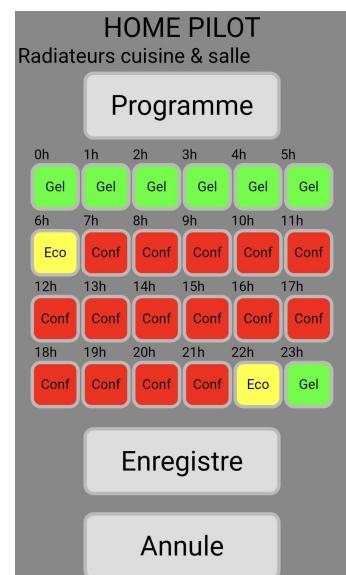
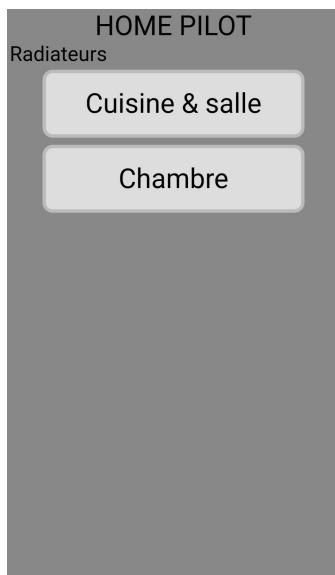
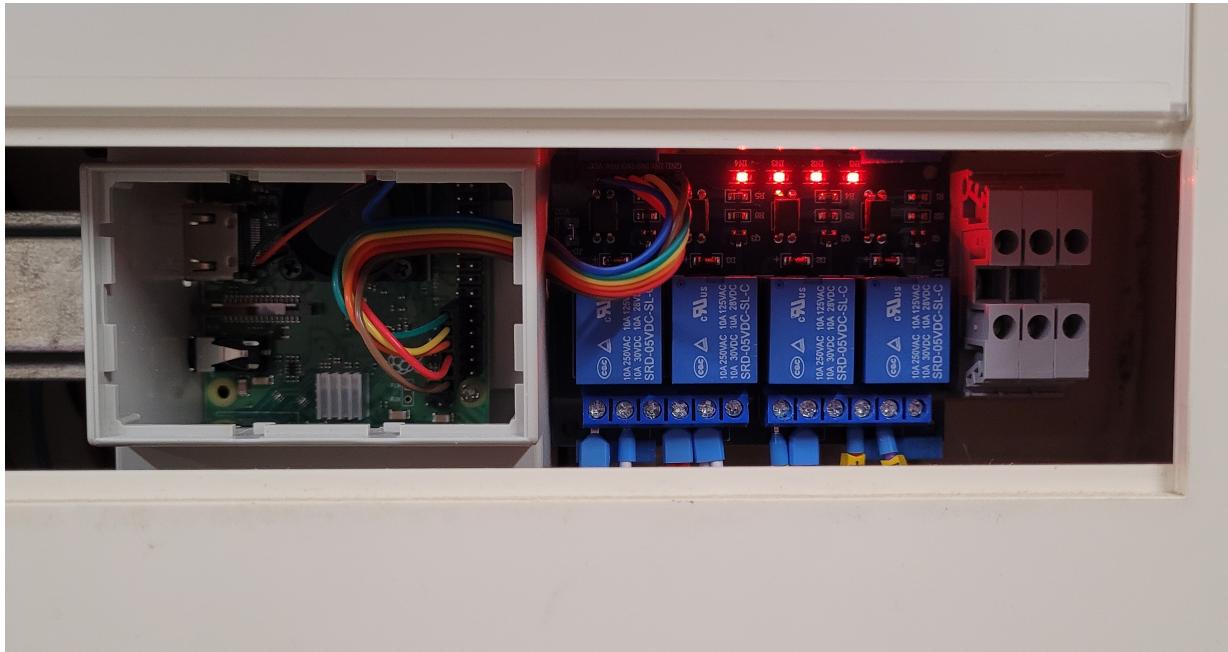


Home Pilot

"Pilotage à distance de circuits multi-états"



Origine du projet

La maison est équipée de "radiateurs" électriques, installés il y a plus de 20 ans.

A l'origine, il y avait une centrale "fil pilotes" 2 zones. Ce truc là, c'est bien joli, mais ça fonctionne sur des programmes horaires pré-enregistrés. Tout au plus peut-on définir un mode "customisé" (assez lourd à paramétrier), ce qui peut convenir aux personnes ayant des horaires immuables, mais ce n'est pas notre cas.

Donc, je n'ai pas tardé à démonter ce machin qui ne servait à rien.



Donc, je me suis mis à chercher une solution qui pourrait remplacer cette centrale, toujours en utilisant le fil pilote, mais avec les contraintes supplémentaires:

- Pouvoir modifier facilement la programmation, sans être obligé de retourner au tableau électrique. Cette modification doit pouvoir être faite par des personnes n'ayant pas forcément une grande attirance pour les "nouvelles technologies".
- Pouvoir commander à distance. Par exemple: on part en week-end, on ne sait pas trop exactement quand on va rentrer, donc je veux pouvoir relancer le chauffage une fois sur la route du retour.

Pour atteindre cet objectif, j'ai examiné plusieurs pistes:

Les domotiques commerciales: (DeltaDore, Ikea, Google, LeroyMerlin, Lidl...)

Inconvénients:

La solution est totalement fermée, il faut s'adapter au service offert.

La gestion du chauffage m'a paru compliquée (plages horaires).

Besoin d'installer des modules radio dans les boîtiers de raccordement des convecteurs.

La radio c'est utile quand on ne peut pas passer de câbles mais là j'ai tout ce qu'il faut.

La radio c'est bien quand ça marche, mais quand on commence à avoir des problèmes de portée, de piles à remplacer, d'adressage...

Un coût total non négligeable.

Les domotiques Raspberry: (JeeDom, Domotix...)

Mis à part le fait que ces applications sont plus customisables que précédemment, on garde finalement un peu les mêmes inconvénients.

Objectifs

Les objectifs que je me suis fixés pour ce projet sont:

- Simple à utiliser:

Il s'agit juste de pouvoir commander des circuits depuis un smartphone. Un enfant doit pouvoir le faire. Au maximum, une table horaire sur la journée avec des modes heure par heure.

- Ouvert:

On n'utilise que des logiciels open-source parmi les plus répandus. Toute personne avec des notions d'informatique doit pouvoir intervenir dessus.

- Économique et évolutif:

Que des composants simples, non liés à un constructeur en particulier, disponibles dans le commerce local ou en ligne.

- Fiable et facile à dépanner:

Câblé en fil-à-fil et en RJ45. Pas de liaison radio ou autre protocole compliqué.

Descriptif

Je résume le projet réalisé comme "pilotage à distance d'actionneurs multi-états".

Pilotage à distance = depuis n'importe où sur Internet via n'importe quel navigateur HTML (plutôt sur smartphone).

Actionneurs multi-états = circuit pouvant prendre plusieurs état logiques.

Par exemple, un éclairage peut prendre 2 états (éteint/allumé);
un store motorisé peut prendre 3 états (arrêt/montée/descente);
etc...

Pour les radiateurs électriques, c'est un peu plus compliqué:

Les vieux modèles n'ont que 2 états (confort/éco);
d'autres 3 ou 4 états (confort/éco/hors gel/arrêt).

Certains modèles ont même d'autres modes qui ne sont pas traités ici car plus compliqués (trains d'impulsions).

J'ai commencé par quelques essais sur Arduino, avec une carte Ethernet ENC28J60, vite abandonnés à cause des limitations de cette dernière.

Donc naturellement, j'en arrive au Raspberry Pi, beaucoup plus efficace pour créer un petit serveur de "domotique". Inconvénient: le nombre de ports GPIO plus limité (apparemment, 17 ports utilisables). Pour une application plus étendue, il faudrait envisager une liaison SPI avec un Arduino, mais là c'est une autre paire de manches.

Matériel

Un Raspberry Pi 3 Model B+

J'ai pris ce modèle, assez ancien, car il est moins cher et largement assez puissant pour ce qu'on veut faire avec.

Bien sur, vous pouvez essayer un autre modèle mais, attention, il se peut que ça vous demande quelques adaptations (histoires de version d'OS, de compatibilité de bibliothèques et autres réjouissances).

Environ 54€ en 2024.



Une alimentation micro USB 5V 3A

Attention, le Raspberry a besoin d'une alimentation assez puissante.

J'ai perdu beaucoup de temps et gaspillé pas mal d'argent à racheter des composants avant de finalement m'apercevoir que le problème venait d'une alimentation défaillante.

Certains modèles ont un petit interrupteur, ce qui est bien pratique pour éteindre le Raspberry au besoin.

Environ 10€ en 2024.



Une carte micro SD

16 GB sont largement suffisants.

Étant donné qu'il y a des accès en écriture assez répétés, j'ai privilégié une carte de bonne qualité.

Mais j'ai aussi fait tous mes essais avec une carte à pas cher et je n'ai eu aucun problème.

Prévoir un petit graveur si vous n'en avez pas.

Environ 10€ en 2024.



Un boîtier rail DIN pour Raspberry Pi 3B

Pour permettre l'intégration dans le coffret électrique.

Vendu avec quelques radiateur et un petit ventilo, c'est pas plus mal.

Attention, d'un modèle de Raspberry à l'autre ce n'est pas le même boîtier.

Environ 18€ en 2023.



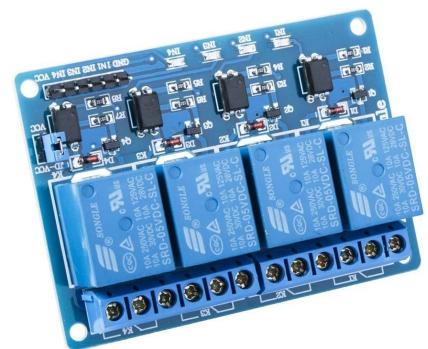
Une carte 4 relais

Pour interfaçer le 3,3V des sorties GPIO avec le 230V du fil pilote.

Environ 13€ en 2023.

Voir pour 2, 4 ou 8 relais suivant le nombre de circuits à piloter. J'aurais préféré une solution "statique" à base d'électronique pure, sans électromécanique pour des raisons de longévité, mais la manipulation du 230V n'est pas si simple. On verra plus tard.

Là aussi il faudrait un boîtier pour intégration sur rail DIN mais pour l'instant je me contente d'un support vite fait à l'imprimante 3D.



Quelques fils Dupont femelle-femelle

Pour relier les sorties GPIO au bornier de la carte relais.

Environ 10€ en 2023.



J'ai fini par trouver une petite valise de connecteurs à monter soi-même chez Lidl. C'était pas très cher et ça permet de faire un câblage plus propre.



On est donc aux alentours de 120€ (2024) au total. Il faut cependant ajouter de quoi amener l'Ethernet au niveau du Raspberry Pi (câble, connecteurs, jarretières, switch...).

Éventuellement quelques composants électroniques (LEDs, résistances) si vous voulez tester sur breadboard avant, ce que je recommande quand même.

Fichiers

Avertissement:

Mon niveau et mes connaissances en informatique datent du début des années 80 ! Ne soyez pas surpris si mon code est plus proche de la calculatrice TI57 que de ce qui se fait maintenant et s'il est honteusement peu optimisé. En dehors du fait que ça fonctionne quand même, j'y vois deux avantages:

- N'étant pas compacté, les débutants auront moins de mal à le comprendre et pourront plus facilement l'adapter à leur besoin.
- Les programmeur chevronnés auront tout le loisir de s'éclater à l'optimiser au maximum.

Il m'est aussi arrivé de recopier des exemples trouvés sur Internet, sans en comprendre toutes les finesse. Là aussi, des améliorations doivent être possibles.

Les fichiers constituant le projet sont:

serveur.js

C'est le noyau central du système. Comme son nom l'indique, c'est un serveur développé en JavaScript qui est lancé au démarrage du Raspberry Pi par une application appelée NodeJS. Ce serveur écoute le port 3000 (modifiable) et, suivant la requête:

- Présente l'interface avec l'utilisateur.
- Enregistre les modifications demandées dans un fichier.
- Mets à jour les sorties GPIO.

regen.js

Un clone simplifié extrait de serveur.js. Il sert juste à remettre régulièrement à jour les sorties GPIO (en cas de fonctionnement sur table horaire). Il est donc lancé au début de chaque heure.

Attention, en cas de modification du serveur.js, il faudra aussi penser à répercuter ces modifications sur regen.js.

IOon.js et looff.js

Deux petits utilitaires qui allument (ou éteignent) les ports GPIO 21 à 24. Utilisés uniquement lors de la phase de mise au point, pour tester le câblage.

Le reste est classique d'un serveur HTML:

index.html

Page d'accueil appelée à la connexion.

index.js

Scripts liés à index.html.

styles.css

Feuille de style. Commune à tous les fichiers .html.

cuisine.html et chambre.html

Interfaces utilisateurs pour chaque circuit.

A chacun de les renommer, modifier, multiplier suivant l'application.

cuisine.js et chambre.js

Scripts liés aux fichiers .html.

cuisine.json et chambre.json

Ces fichiers stockent les informations de configuration de chaque circuit (mode et table horaire). Indispensable pour ne pas perdre les réglages, en cas de coupure de courant par exemple.

Ils sont écrits et lus par serveur.js.

Ils sont lus par regen.js.

A chacun, donc, d'analyser ces fichiers et de les adapter à ses besoins.

Configuration

1) Installer le système

Insérer la carte micro-SD dans le lecteur.

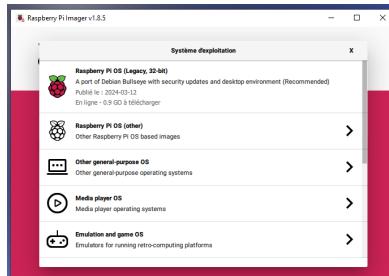
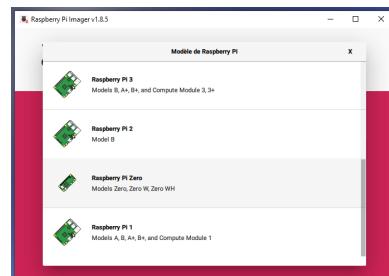
Lancer l'application **Raspberry Pi Imager**.



Choisir le modèle:

Raspberry Pi 3

(sauf si...)



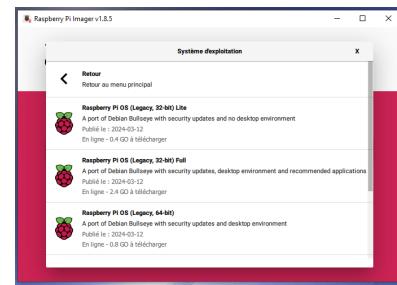
Choisir l'OS:

Raspberry Pi (other)

(deuxième ligne)

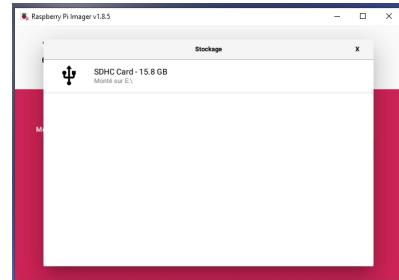
Raspberry Pi OS (Legacy, 32 bit) Lite (Debian Bullseye)

(attention à ne pas vous tromper)



Choisir le stockage

Selectionner la carte microSD



Suivant

Modifier les réglages



GENERAL:

Nom d'hôte: **pidom** (par exemple)

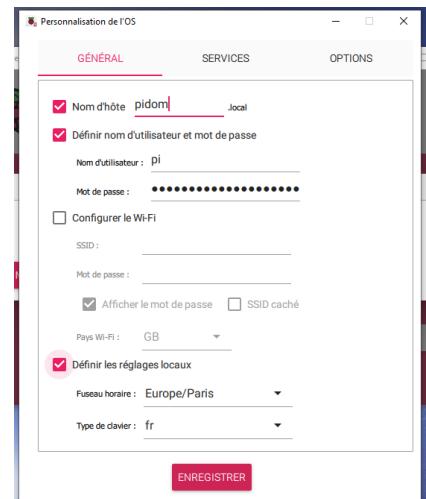
Nom d'utilisateur: **pi**

Mot de passe: **à noter précieusement !**

Pas de Wi-Fi (sauf si...)

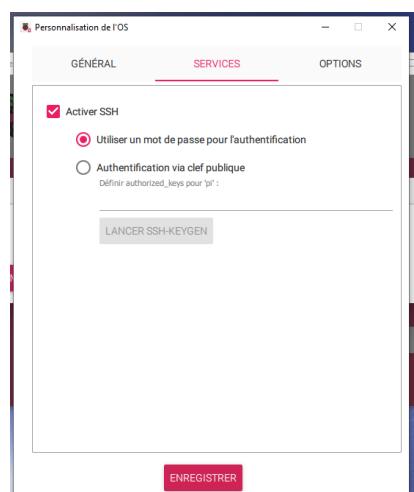
Fuseau horaire: Europe/Paris

Type de clavier: fr



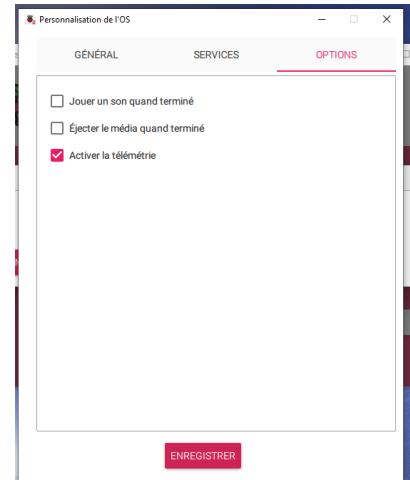
SERVICES:

Activer SSH: Utiliser un mot de passe pour l'identification



OPTIONS

On ne touche à rien



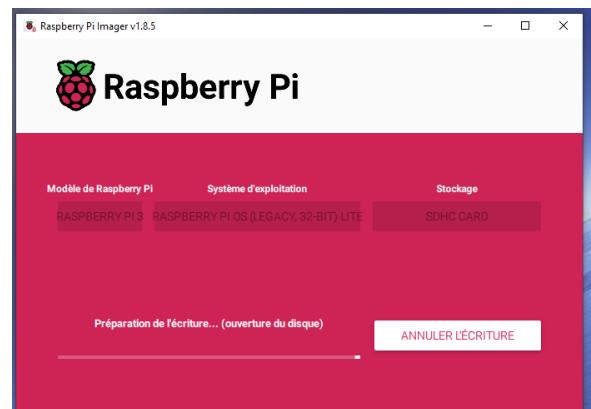
ENREGISTRER

Voulez-vous appliquer les réglages... : OUI

Toutes les données...
... voulez-vous continuer ?: OUI



La gravure démarre
On laisse dérouler jusqu'au bout.



On éjecte la carte micro SD.
On insère la carte dans le Raspberry Pi hors tension.
On raccorde le Raspberry Pi sur l'Ethernet.
On allume le Raspberry Pi et on **attend** un certain temps que toutes les installations se fassent.
Dans des conditions normales, il y en a pour moins d'une heure.

J'utilise une petit application sur Android, **Net Analyser** pour scanner le réseau LAN. A partir du moment où "pidom" apparaît dans la liste, c'est qu'il tourne.

2) Donner une adresse IP statique

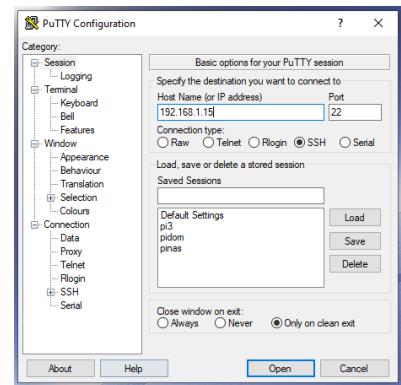
On se connecte sur le Raspberry avec l'application **PuTTY ***

Host Name (or IP address): **l'adresse IP relevée précédemment.**

Port: **22**

Connection type **SSH**

Puis **Open**



login as: pi

pi@xxx.xxx.xxx.xxx's password: le mot de passe configuré lors du formatage de la carte micro SD.

(* A la place de PuTTY, on peut aussi ouvrir une fenêtre "invite de commande" et taper **ssh pi@192.168.1.8**)

Ensuite, on édite le fichier /etc/dhcpcd.conf:

cd /etc

sudo nano dhcpcd.conf

A la fin de ce fichier, on ajoute les deux lignes:

interface eth0

static ip_address=192.168.1.8/24 (adresse IP que vous voulez)

static routers=192.168.1.254 (si c'est bien l'adresse IP de votre box)

On sauvegarde (**Ctrl O**), on quitte (**Ctrl X**)

On reboot le Raspberry Pi

sudo reboot.

On laisse redémarrer et on surveille avec Net Analyser que l'adresse à bien changé.

Vérifiez bien que l'adresse n'est pas utilisée par autre chose et qu'elle n'est pas dans la plage DHCP (voir notice de la box).

Autre solution: donner une adresse fixe par un bail DHCP (voir notice de la box).

3) Créer le dossier projet

A partir de maintenant, on utilisera PuTTY (ou SSH) avec la nouvelle adresse IP.

On peut même sauvegardes ces paramètres avec **Save**, et les rappeler avec **Load** pour gagner du temps.

On crée le dossier /homepilot (ou un autre nom si vous préférez):

```
cd /  
sudo mkdir /homepilot
```

On donne la propriété de ce dossier à pi:

```
sudo chown -R pi /homepilot
```

4) Quelques bibliothèques à installer

NodeJS permet l'exécution de scripts JavaScript:

```
sudo apt-get update  
sudo apt-get install -y nodejs  
node -v pour vérifier que c'est bien installé
```

Wiring Pi permet d'agir sur les ports GPIO.

Un peu plus compliqué, il faut télécharger puis compiler:

```
cd /  
sudo apt-get install -y git  
sudo git clone https://github.com/WiringPi/WiringPi  
cd WiringPi  
sudo git pull origin ( doit donner "already up to date")  
sudo ./build  
gpio -v pour vérifier  
gpio readall donne la table complète des ports GPIO
```

Si possible ,vous pouvez vérifier en câblant une LED avec une résistance sur la broche 29 (BCM5 / wPi21)

```
gpio mode 21 out configure la voie en sortie  
gpio write 21 1 allume la LED  
gpio write 21 0 éteint la LED
```

5) Copier les fichiers dans le dossier projet

J'utilise **FileZilla** pour transférer entre l'ordi et le Raspberry Pi.

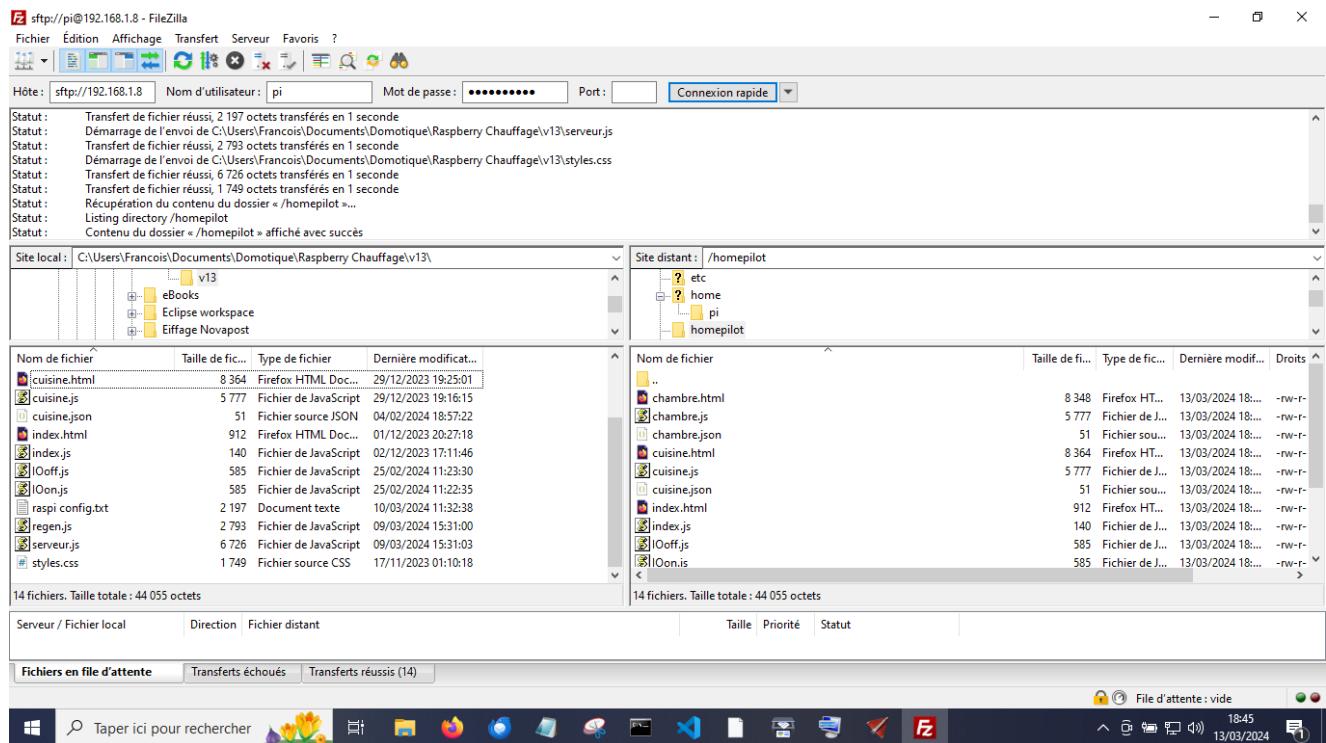
Hôte: **192.168.1.8**

Nom d'utilisateur: **pi**

Mot de passe: **le mot de passe**

Port: **22**

Puis connexion (par la suite, on peut utiliser connexion rapide)



A gauche: le dossier source dans le PC,

A droite: le dossier cible dans le Raspberry Pi: /homepilot

Puis on copie par **glisser-déposer** (ou double clic) l'ensemble des fichiers (à l'exception éventuelle de la présente documentation).

On peut vérifier sur le Raspberry Pi que les fichiers ont bien été copiés:

cd /homepilot

ls

6) Configurer le démarrage automatique des programmes

On édite la "crontab" du Raspberry Pi.

sudo crontab -e

Si le choix est proposé, on choisit **nano**

A la fin du fichier, on ajoute les 3 lignes:

@reboot cd /homepilot && sudo node serveur.js

(lance le serveur à chaque démarrage du Raspberry Pi)

0 * * * * cd /homepilot && sudo node regen.js

(lance la mise à jour des sorties GPIO au début de chaque heure, utile en cas de fonctionnement sur table horaire)

* * * * * sudo reboot

(reboote le Raspberry Pi toutes les nuits à 3h, histoire de faire du propre)

On sauvegarde (**Ctrl O**), on quitte (**Ctrl X**)

On reboot le Raspberry Pi

sudo reboot.

A partir de là, ça doit fonctionner.

7) Prise en main en local

On commence par des essais en local:

Sur n'importe quel navigateur Internet (sur un smartphone par exemple) il suffit de saisir dans la barre d'adresse:

<adresse IP locale>:<numéro de port>/index

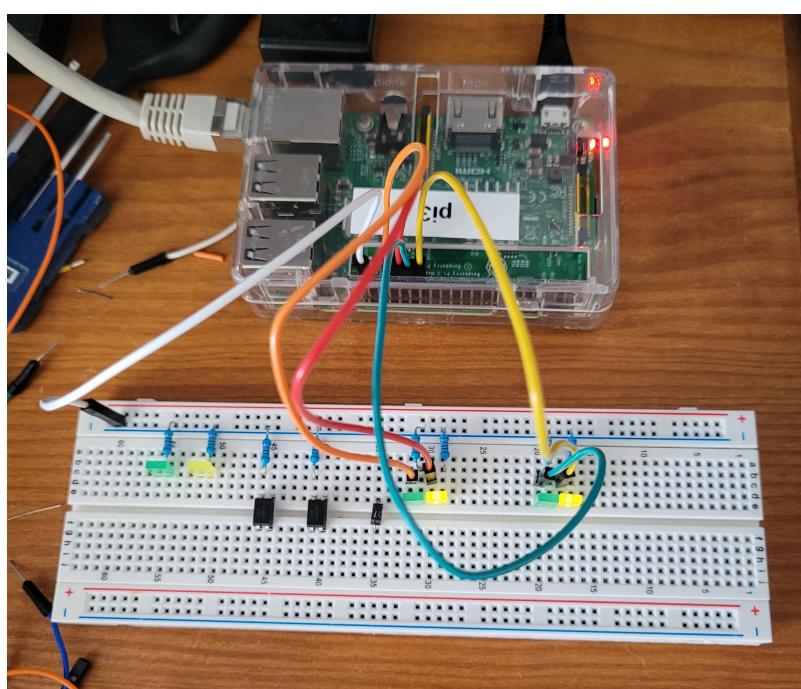
dans mon cas:

192.168.1.8:3000/index

On commence par vérifier que l'interface utilisateur fonctionne et mémorise bien les paramètres.

Un petit coup de coupure d'alimentation pour vérifier que tout repart correctement.

Idéalement, on branche quelques LED et on teste différentes configurations.



8) Prise en main à distance

Plus difficile à expliquer car cela dépend du routeur. Il faut donc ouvrir le gestionnaire de votre box et éditer quelques paramètres. Deux choses:

a- Définir une adresse IP globale fixe

(on parle bien de l'adresse IP de votre box sur le web)

A voir avec votre opérateur.

Certains vous permettent même de créer un nom de domaine.

b- Définir une redirection de port

Il faut indiquer à la box une règle qui redirige une connexion entrante web (WAN) vers la connexion locale sur le Raspberry Pi (LAN).

Sur le gestionnaire de la box, il faut trouver le chapitre "redirection de ports"

(parfois aussi appelé "PAT") et entrer les paramètres:

(je prend l'exemple de la Freebox)

IP destination:	192.168.1.8 (l'adresse locale donnée au Raspberry Pi)
IP source:	toutes
Protocole:	TCP
Port de début:	un numéro au hasard de préférence dans la plage des ports dynamiques, entre 49152 et 65535
Port de fin:	le même numéro
Port de destination:	3000 (le numéro que vous avez mis dans serveur.js)
Commentaire:	comme vous voulez

Enregistrer et redémarrer la box si nécessaire.

Cette solution n'est pas idéale en termes de protection; un pirate ayant intercepté la bonne trame sur Internet pourrait vous couper le chauffage pendant votre sommeil pour rigoler ! Si vous pensez avoir été repéré, il reste possible de changer le numéro de port entrant ou de remplacer le nom du fichier "index" par autre chose de plus tordu.

Il y a sûrement d'autres solutions plus sécurisées (genre VPN ?) mais je ne connais pas. A étudier.

On refait les mêmes essais que précédemment, à distance cette fois-ci:

Sur n'importe quel navigateur Internet il suffit de saisir dans la barre d'adresse:

<adresse IP globale>:<numéro de port entrant>/index

ou

<nom de domaine>:<numéro de port entrant>/index

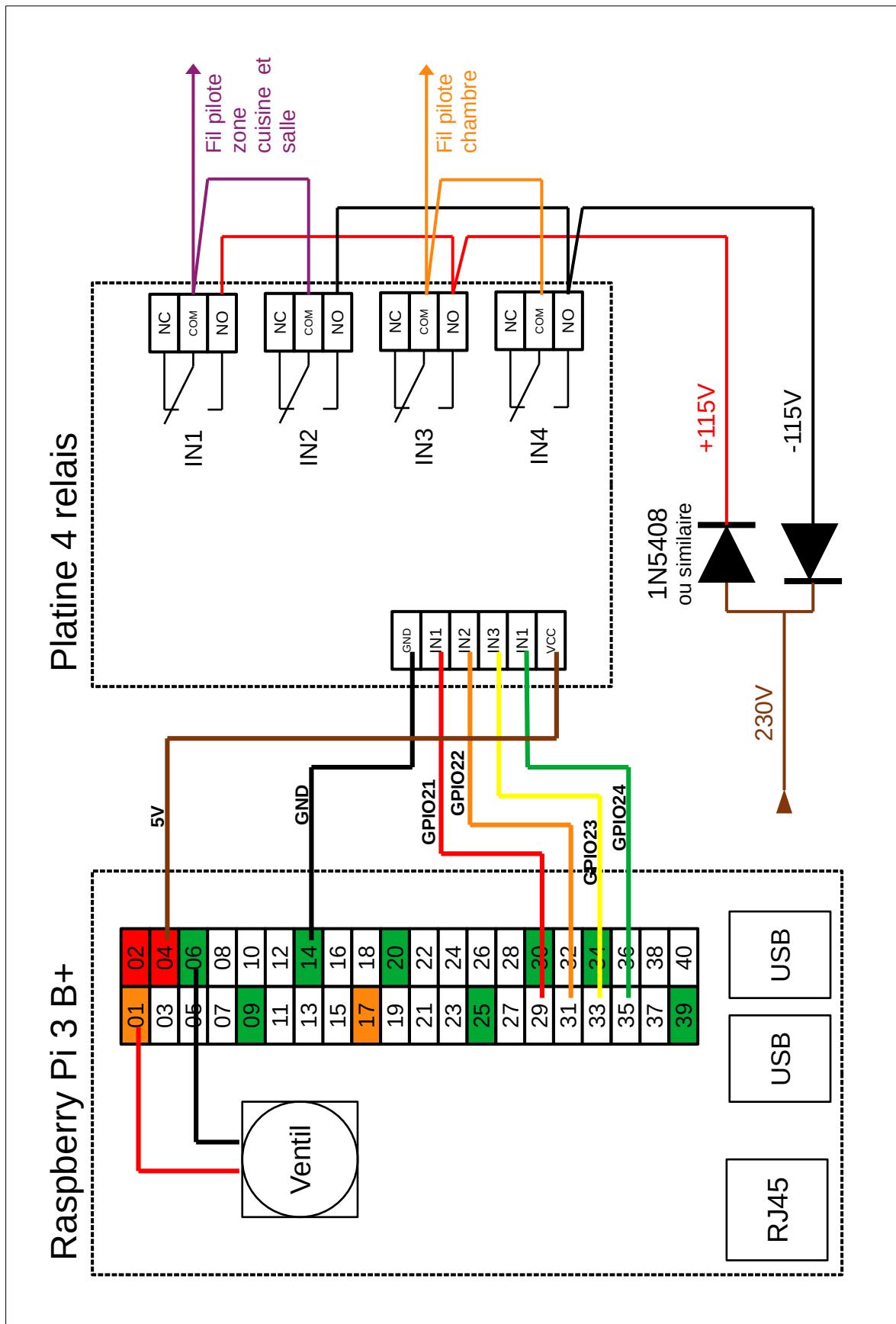
genre:

*****.***.***.***:*****/index**

On refait les mêmes essais que précédemment, à distance cette fois-ci.

Câblage

Je prend comme principe que vous savez câbler et lire un schéma. Si ce n'est pas le cas, faites vous aider par un ami électricien.



Pilotage des radiateurs électriques:

La plupart des radiateurs (ou convecteurs) sont raccordés avec 3 ou 4 fils. Vérifiez sur la notice du radiateur, mais en principe, les couleurs sont:

- Marron** = phase 230 V
- Noir** = fil pilote
- Bleu** = neutre
- Vert-jaune**: terre (sauf si l'appareil est de classe II)

Le fil pilote sert à télécommander le mode de fonctionnement du convecteur.

Les modèles plus anciens n'ont que deux modes:

- Confort: le radiateur chauffe à la consigne normale
- Économie: le radiateur chauffe sur une consigne réduite, soit par un deuxième réglage, soit suivant un abaissement par rapport à la consigne normale

Les modèles plus récents peuvent avoir d'autres modes supplémentaires:

- Hors gel: le radiateur chauffe à une température de 7°C
- Arrêt: arrêt complet
- Confort -1°C
- Confort -2°C

Vérifiez bien sur votre appareil les modes qu'il est capable de gérer.

Le radiateur réagit ensuite suivant le signal qui lui est envoyé sur le fil pilote:

Mode	Signal à transmettre	Mesure par rapport au neutre
Confort	—	0 Volt
Eco	—○—	230 Volts
Hors-gel	—○—	115 Volts négatif
Arrêt Chauffage	—○—	115 Volts positif
Conf. -1°C	3s 297s	230 Volts/3s
Conf. -2°C	7s 293s	230 Volts/7s

Attention: sur le module 4 relais que j'ai utilisé, la logique est inversée; c'est à dire que les relais sont activés lorsque les sorties GPIO sont désactivées. Pour y voir clair, rien de mieux qu'une petite table logique.

	+115V	-115V	GPIO21	GPIO22
Arrêt	On	Off	0	1
Hors gel	Off	On	1	0
Eco	On	On	0	0
Confort	Off	Off	1	1

Brochage GPIO:

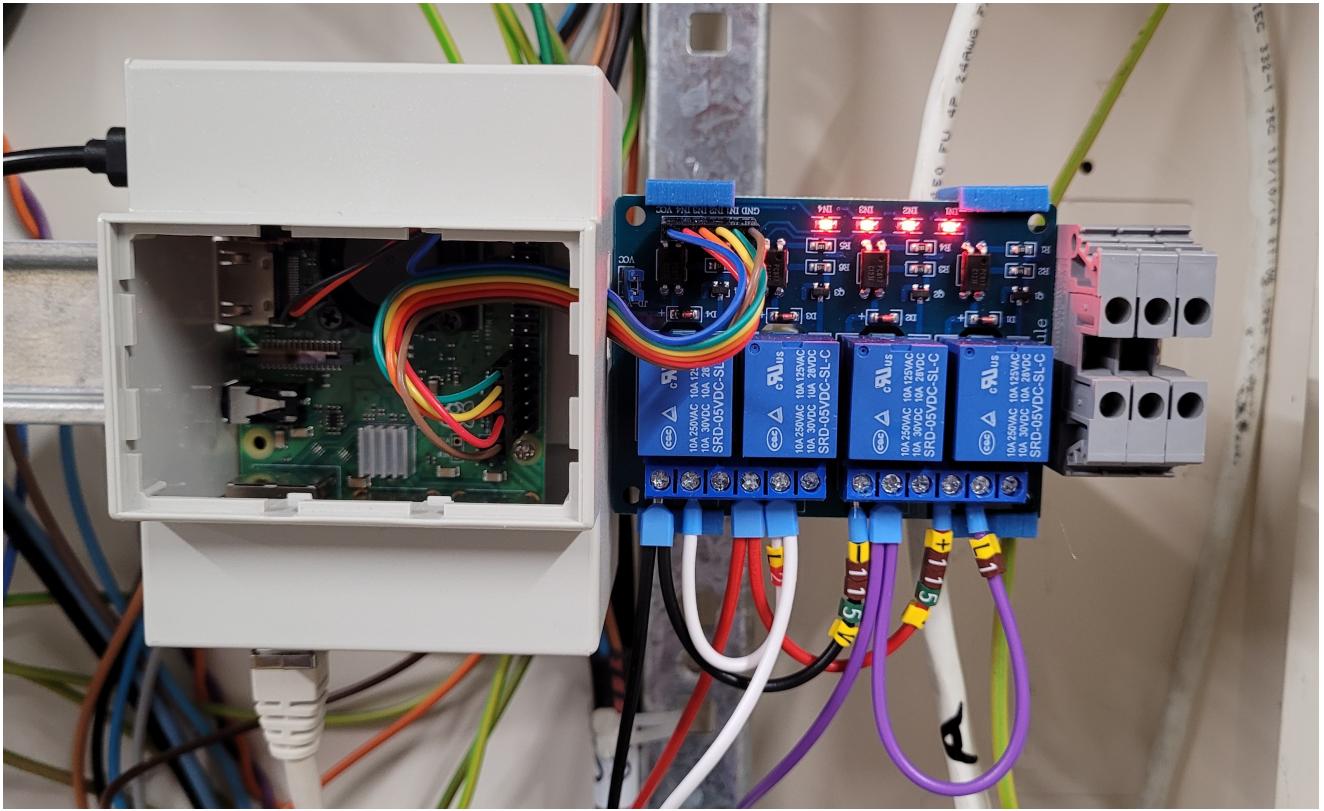
Sur le schéma précédent j'ai utilisé les GPIO 21 à 24. Si vous voulez en utiliser d'autres voici le "PinOut" du Raspberry Pi 3 B+.

Attention, la bibliothèque Wiring Pi utilise un autre code que le BCM couramment utilisé.

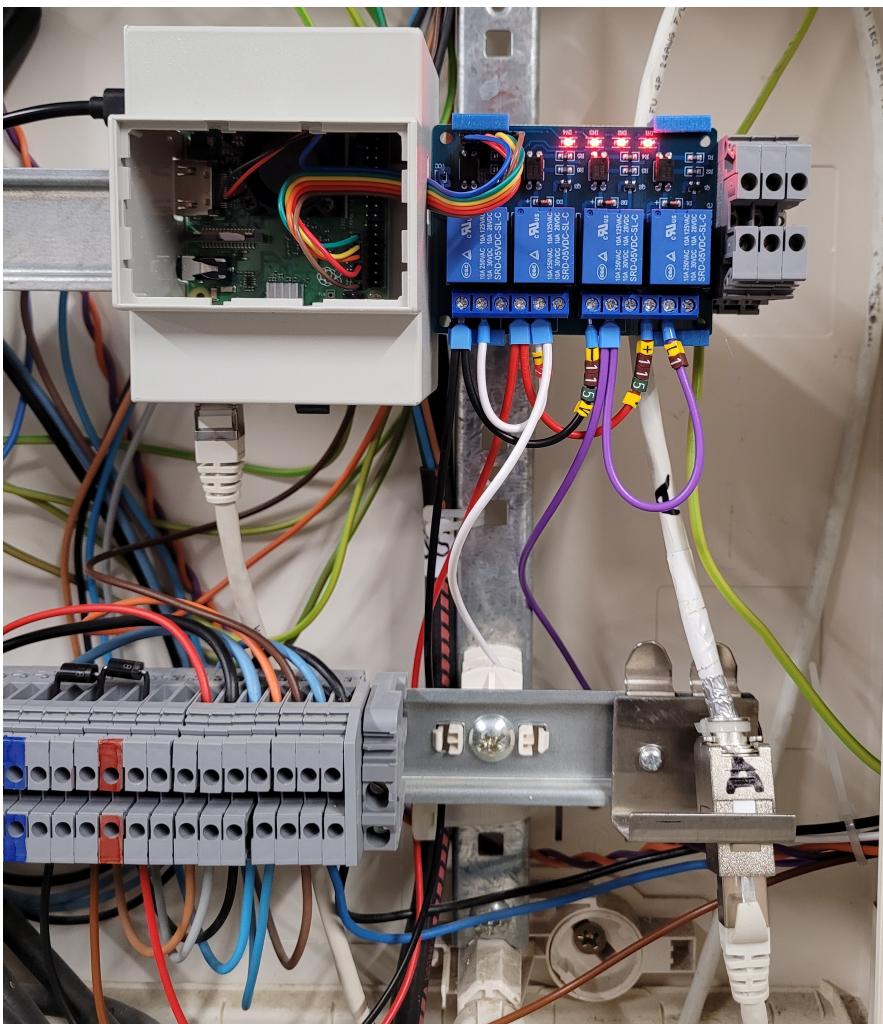
GPIO	pin	pin	GPIO
3V3	1	2	5V
I2C	3	4	5V
I2C	5	6	GND
?	7	8	UART
GND	9	10	UART
?	11	12	1
?	13	14	GND
?	15	16	4
3V3	17	18	5
SPI	19	20	GND
SPI	21	22	?
SPI	23	24	SPI
GND	25	26	SPI
DNC	27	28	DNC
21	29	30	GND
22	31	32	26
23	33	34	GND
24	35	36	27
25	37	38	28
GND	39	40	29

Je n'ai pas réussi à identifier le numéro GPIO des broches marquées ?.

Évitez d'utiliser les cases colorées qui risquent d'être perturbées par d'autres ressources (I2C, SPI, UART...).



Nota: sur ces photos, les couleurs de fil ne sont pas celles du schéma.



Essais

Il ne reste plus qu'à faire les essais en réel.
N'oubliez pas de mettre vos radiateurs en mode "Auto".
On laisse tourner quelques jours pour vérifier que tout va bien.

Bon amusement !



Liste à lapins

Points à voir / améliorer:

- Réponse à la requête "favicon".
- Sécurisation par mot de passe.

Propriété intellectuelle

Ce projet est mis à disposition de la communauté, libre d'utilisation et de modification, aux conditions suivantes:

- Pas de distribution à titre commercial
- Les copies et adaptations devront porter la référence du projet initial
- L'auteur décline toute responsabilité quand aux conséquences de l'utilisation du projet