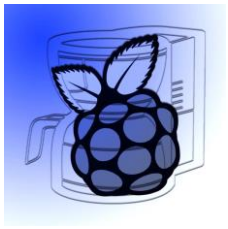


SZAMVEBER Matthias
FREIN Mathias
LP SEICOM

Projet Tuteuré

CAFETIÈRE-PI

Tout les fichiers ressource de ce projet sont accessible sur :
https://github.com/bobytruk/Projet_tutore-CafetierePi

I. CONTENU

II. TABLE DES FIGURES

III. TABLE DES TABLEAUX

IV. PRESENTATION ET CAHIER DES CHARGES

De nos jours, on entend dire que « *Nos smartphones peuvent tout faire sauf le café* ». Ce ne sera plus le cas grâce à ce projet qui a pour but de contrôler une machine à café grâce à une application smartphone et/ou une interface web. La cafetière sera commandée via un RaspberryPi qui gérera la mise en route et la communication entre la cafetière et le smartphone via WIFI. Si l'avancement du projet le permet, il sera possible de programmer la cafetière à une certaine heure/date via l'application.

FIGURE 1-PRESENTATION PROJET

- Étude de la cafetière
  - Choix du type de cafetière
  - Choix des capteurs à intégrer (présence d'eau, présence du bol)
  - Voir si besoin d'une adaptation de tension pour les commandes
- La mise en œuvre du RaspberryPi
  - Documentation sur le fonctionnement de la carte
  - Installation de l'OS
  - Test des ports GPIO (entrées / sorties)
- Développement d'une application Android
  - Documentation
  - Utilisation du SDK Android Development Kit
  - Création d'une interface graphique
  - Utilisation de la communication WIFI
- Fusion des différentes étapes
  - Connectivité entre Smartphone et RaspberryPi
  - Traitement des données envoyées par le Smartphone et les capteurs
  - Gestion par le RaspberryPi de la cafetière
- A terme
  - Notification lorsque le café est terminé
  - Programmation de la cafetière via le Smartphone
  - Authentification des utilisateurs
  - Statistiques en fonction de la consommation des utilisateurs
  - Lancer la machine via internet

V. ELABORATION DU PLANNING

Dans la gestion d'un projet, l'élaboration du planning fait partie d'une des tâches les plus importantes en ce qui concerne la gestion du temps de notre projet. Pour le réalisé, nous avons bien analyser notre cahier des charges pour définir les différentes tâches à effectuer, de plus il nous a fallu nous projeter dans un futur proche pour savoir combien de temps allais nous prendre ces tâches pour les réaliser et ainsi vérifier leurs bons fonctionnements. C'est pourquoi nous avons décidé de découper notre projet en étapes distinctes en sachant que nous avions que 6 semaines pour réaliser en globalité notre projet.

Ci-dessous le tableau du découpage initial de notre projet tuteuré :

Etapes	n° sem. 45	n° sem. 48	n° sem. 49	n° sem. 04	n° sem. 05	n° sem. 09
Cahier des charges	Redéfinition du cahier des charges : approfondissement des taches à réaliser					
Etude de la cafetière	choix de la cafetière et étude l'électronique présente, conception et réalisation d'une carte d'adaptation tension/puissance					
Mise en œuvre du Raspberry Pi		Documentation, installation de l'OS, test des GPIO	Interaction Raspberry Cafetière	Création d'un point d'accès WIFI		
Développement Application		Documentation, début interface graphique	Interface graphique fini	Interaction Application WIFI		
Mise en commun					Choix d'un protocole de communication Interaction Smartphone, Cafetière	

TABLEAU 1- DÉCOUPAGE INITIAL DU PROJET

VI. DOCUMENTATION

A. LA CAFETIÈRE

Pour ce projet nous avons dû choisir une cafetière. Nous nous sommes limités à une cafetière simple, à filtre. Il existe deux catégories de ces cafetières, les programmables et les non programmables. Notre choix c'est porté sur une cafetière programmable, car dans ce type d'appareil, il y a déjà tout un système électronique permettant de programmer la machine à café. Et cela nous évite donc de le concevoir ce qui nous permet un gain de temps et de moyens qui n'est pas négligeable au vu du temps qui nous a été alloué pour réaliser ce projet.

Une fois avoir déterminé le type de cafetière voulu, nous avons comparé chez différents fournisseurs les modèles qu'ils proposent. Notre choix a donc été la cafetière programmable de la marque Listo vendue chez Boulanger au prix de 20€ ce qui nous semble être un prix raisonnable au vu des fonctionnalités qu'elle propose et de ce nous voulons en faire.

Le démontage

Le démontage n'a pas posé de problèmes majeurs à part la présence de vis de sécurité en forme d'étoile de type TORX.

FIGURE 2 - VU DE DESSOUS DE LA CAFETIERE

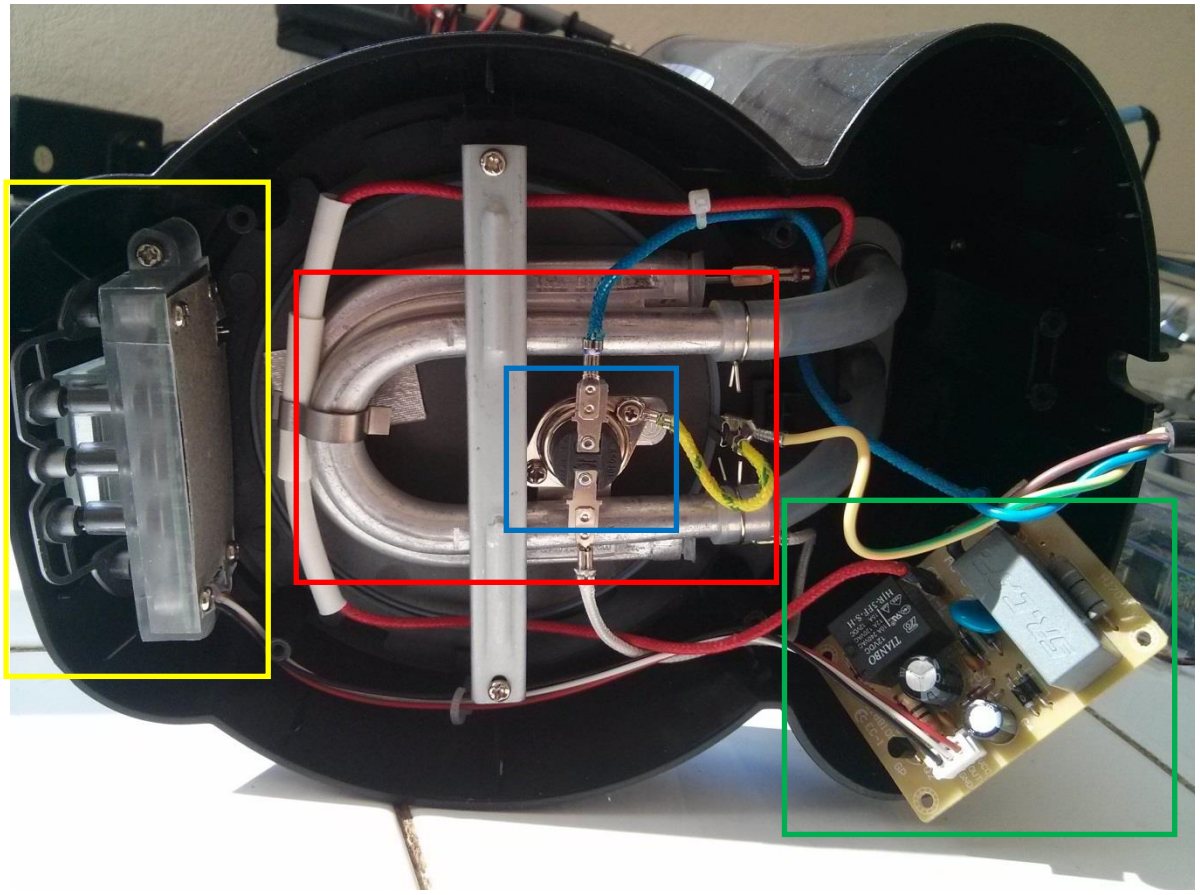


FIGURE 3 - VU DE DESSOUS DE LA CAFETIERE SANS LE CAPOT

Une fois le capot retiré, on aperçoit les différents éléments qui composent la cafetière.

-Le programmateur
-Le transformateur et le relais de contrôle
-La résistance de chauffe
-Le thermostat

VUE DE PLUS PRÈS

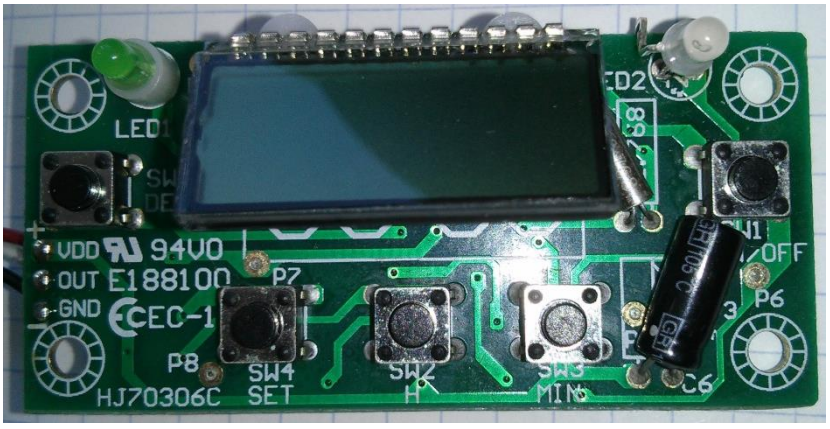
La carte transformateur/relais de contrôle

FIGURE 4 – CARTE TRANSFORMATEUR/RELAIS DE CONTROLE

Cette carte convertit la tension 220V AC en 5V et 12V DC. Le 5V va alimenter le programmateur et le 12V DC le relais.

CONNECTEUR PROGRAMMATEUR

Le programmateur



Le programmateur envoi du 5V ou du 0V sur la broche « out » lorsque le bouton m/a est pressé.

Il envoie aussi du 5V lorsque l'heure programmée est atteinte.

FIGURE 5 – CARTE DU PROGRAMMATEUR

Pour contrôler cette cafetière à distance, il nous suffit donc de placer un dispositif entre la carte du relais et le programmateur qui injectera les mêmes commandes que le programmateur au relais

B. LE RASPBERRY-PI

Le dispositif qui va permettre de contrôler la cafetière à distance que nous avons choisis est une carte RaspberryPi.

Mais qu'est-ce que le RaspberryPi ?

D'après Wikipédia, il s'agit d'un ordinateur, qui a la taille d'une carte de crédit, permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération.

Cet ordinateur est destiné à encourager l'apprentissage de la programmation informatique. Il est cependant suffisamment ouvert (ports USB voire réseau) et puissant (ARM 700 MHz, 256 Mo de mémoire vive pour le modèle d'origine, 512 Mo sur les dernières versions) pour permettre une grande palette d'utilisations. Son circuit graphique BMC Videocore 4 en particulier permet de décoder des flux Blu-Ray full HD (1080p 30 images par seconde), d'émuler d'anciennes consoles et d'exécuter des jeux vidéo relativement récents.

Son prix de vente était estimé à 35 $, soit 29,09 €, début mai 2011. Les premiers exemplaires ont été mis en vente le 29 février 2012 pour environ 35 €. Début 2013, plus d'un million de RaspberryPi Pi ont déjà été vendus.

Nous avons donc choisi d'utiliser ce système pour notre projet. D'une part pour son bas coût au vue de ses performances mais aussi, car il dispose de ports d'entrée/sortie GPIO (General Purpose Input/Output) dont nous allons nous servir pour contrôler la cafetière. De plus, il peut se connecter facilement à un réseau et son utilisation est relativement simple.

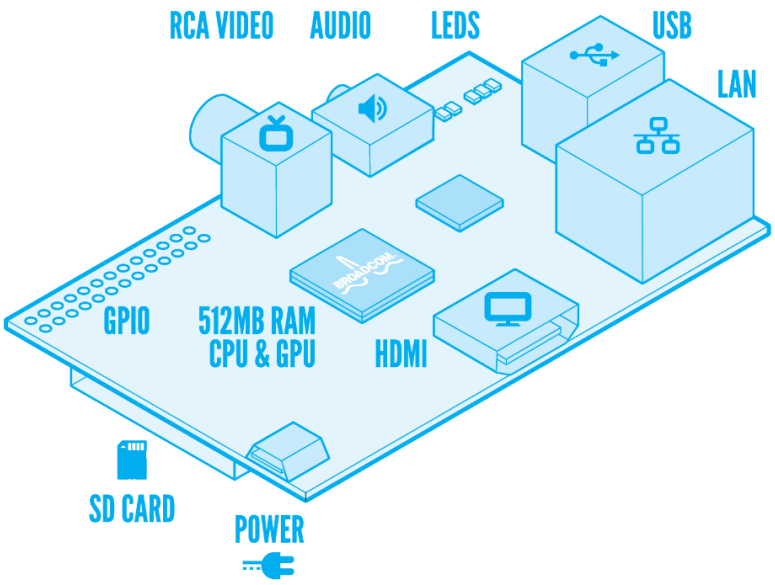


FIGURE 6 – CARTE RASPBERRY PI

Voici une infographie réalisée par kubii.fr, le distributeur officiel de RaspberryPi en France, retraçant l'histoire de cette carte.

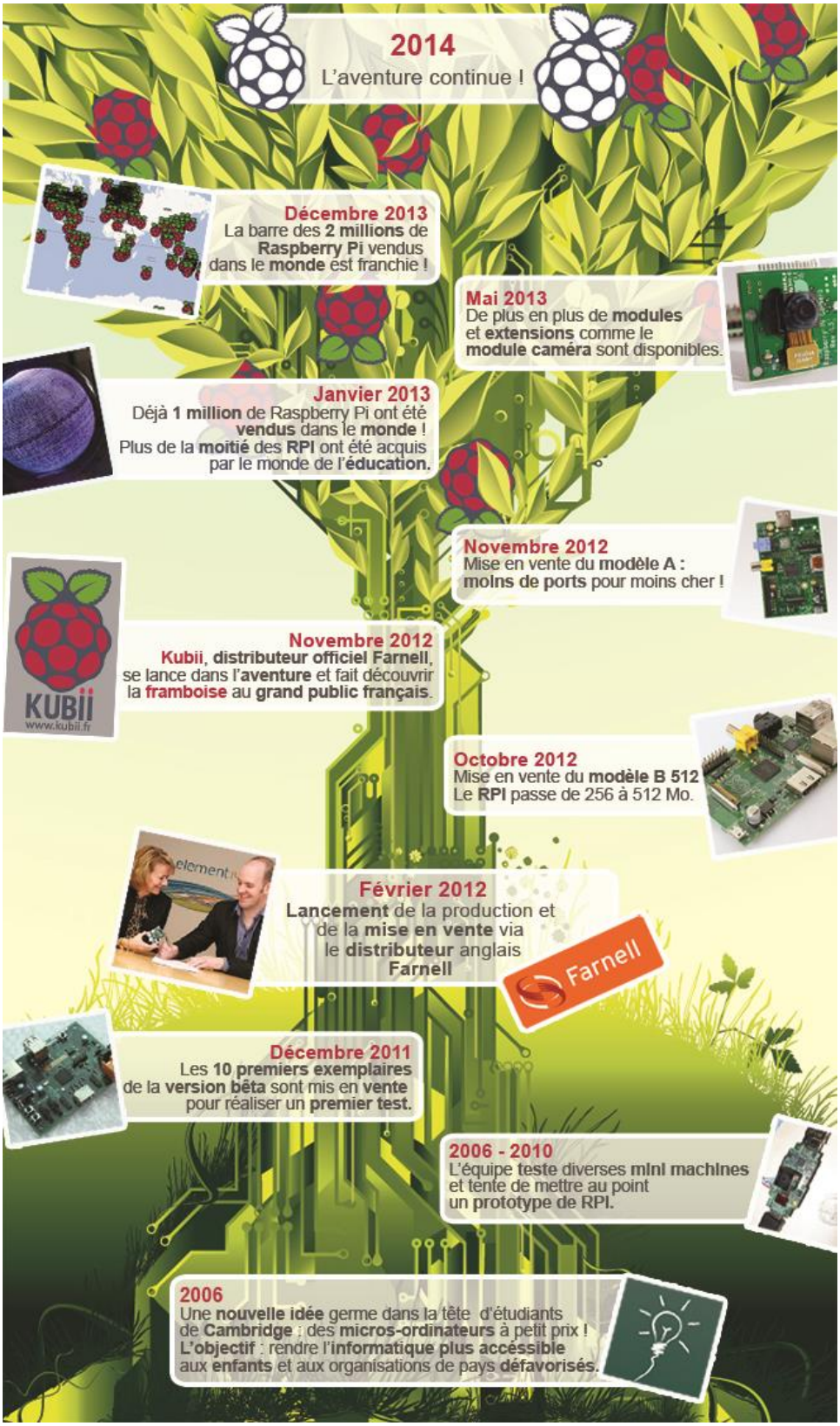


FIGURE 7 - HISTORIQUE

C. L'APPLICATION ANDROID

Le Web mobile a depuis longtemps envahi notre quotidien de concepteur de sites web : aujourd'hui, smartphones et tablettes font partie intégrante du parc de périphériques sur lesquels nous jouissons de notre dose quotidienne d'Internet, sur lesquels nous consultons nos sites web préférés, et pestons - à juste titre - lorsque celui-ci ne s'affiche pas correctement. Dans ce projet pour interagir avec le RaspberryPi nous avons mis en œuvre une application web Android, alors une application web est le nom que l'on donne à un site web "classique" que l'on a adapté pour les mobiles. La structure existante reste en place (HTML, scripts, bases de données, médias) et est complétée par une couche graphique dynamique adaptée aux écrans de taille réduite dans un principe général de design adaptatif. Des retouches ou surcouches JavaScript sont parfois également à l'ordre du jour.

D'après Wikipédia Android, est un système d'exploitation pour smartphones, tablettes tactiles, PDA et terminaux mobiles. C'est un système open source utilisant le noyau Linux. Il a été lancé par une startup rachetée par Google en 2005. D'autres types d'appareils possédant ce système d'exploitation existent, par exemple des téléviseurs, des radioréveils, des montres connectées, des autoradios et même des voitures.

Concernant notre projet le développement de notre application c'est fait en Java, le développement d'applications pour Android s'effectue avec un ordinateur personnel sous Mac OS, Windows ou Linux en utilisant le JDK de la plate-forme Java et des outils pour Android. Des outils qui permettent de manipuler le téléphone ou la tablette, de la simuler par une machine virtuelle, de créer des fichiers APK - les fichiers de paquet d'Android, de déboguer les applications et d'y ajouter une signature numérique. Ces outils sont mis à disposition sous la forme d'un plugin pour l'environnement de développement Eclipse.

La bibliothèque d'Android permet la création d'interfaces graphiques selon un procédé similaire aux Framework de quatrième génération que sont XUL, JavaFX ou Silverlight: l'interface graphique peut être construite par déclaration et peut être utilisée avec plusieurs *skins* - chartes graphiques. La programmation consiste à déclarer la composition de l'interface dans des fichiers XML; la description peut comporter des *ressources* - des textes et des pictogrammes. Ces déclarations sont ensuite transformées en objets tels que des fenêtres et des boutons, qui peuvent être manipulés par de la programmation Java. Les écrans ou les fenêtres (*activités* dans le jargon d'Android), sont remplis de plusieurs *vues*; chaque vue étant une pièce d'interface graphique (bouton, liste, case à cocher…). Android 3.0, destiné aux tablettes, introduit la notion de *fragments*: des panneaux contenant plusieurs éléments visuels. Une tablette ayant - contrairement à un téléphone - généralement suffisamment de place à l'écran pour plusieurs panneaux.

Pour la petite histoire Android doit son nom à la startup éponyme spécialisée dans le développement d'applications mobiles rachetée par Google en août 2005, nom venant

lui-même d'« androïde » qui désigne un robot construit à l'image d'un être humain. Le logiciel, qui avait été surnommé *gPhone* par les rumeurs de marchés et qui selon un de ses concepteurs Andy Rubin était initialement prévu pour être un système d'exploitation pour appareil photo, est proposé de façon gratuite et librement modifiable aux fabricants de téléphones mobiles, ce qui facilite son adoption. Le gPhone a été lancé en octobre 2008 aux États-Unis dans un partenariat de distribution exclusif entre Google et T-Mobile. Anticipant les annonces officielles, les marchés financiers se ruent massivement sur les actions Google les faisant monter jusqu'au plus haut historique de 724 dollars le 5 novembre 2007 (Le vendredi 18 octobre 2013 les actions Google franchissent les1 000 dollars). En 2004, le prix du cours d'introduction du moteur de recherche était de 85 dollars l'action.

VII. PROCESSUS DU PROJET

Une fois nous être bien documentés sur les différents éléments ainsi que les différentes parties que nous devons réalisés, nous avons établi un processus qui va nous servir de fil directeur durant notre projet. Il résume les choix que nous avons faits et ceux que nous avons choisi d'écarter. Ce processus montre aussi les différents acteurs qui sont intervenus et quel ont été les supports que nous avons utilisés.

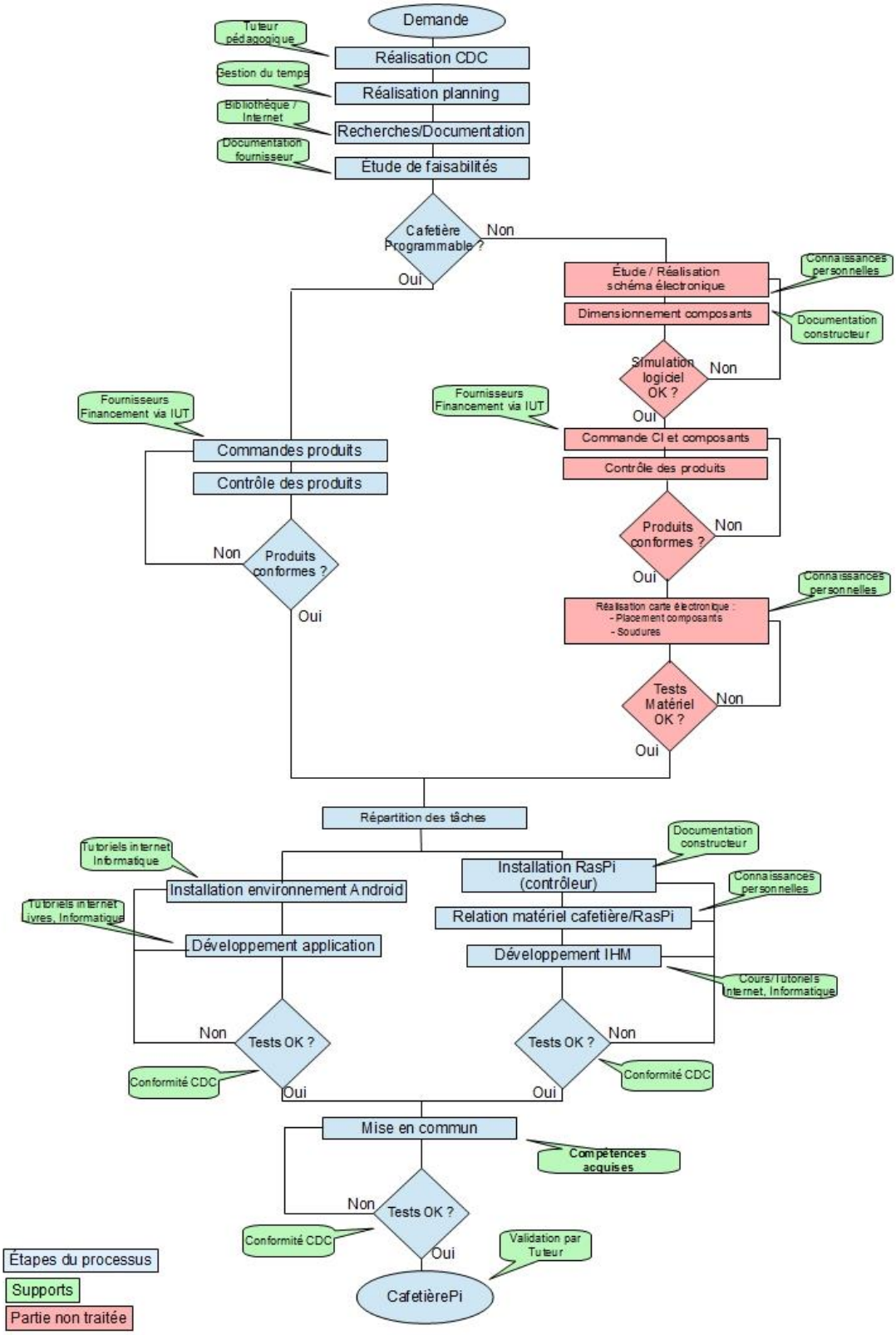


FIGURE 8 – PROCESSUS DU PROJET

VIII. APPLICATION ANDROID

A. INSTALLATION DE L'ENVIRONNEMENT ANDROID SOUS ECLIPSE

La première étape pour programmer des applications pour Android est d'installer le SDK Android. Dans un premier temps qu'est un SDK ?

SDK signifie « Software Development Kit », ce qui veut dire en français « kit de développement logiciel ». C'est donc un ensemble d'outils qui nous permettra de développer nos applications Android.

Vous trouverez ci-dessous les différentes étapes d'installation :

- Etape1 : Téléchargement d'Eclipse

La première chose à faire est donc de télécharger l'environnement de développement Eclipse. Pour cela il suffit, d'aller sur le site d'Eclipse et de télécharger **Eclipse IDE for Java Developers**. Aucune installation n'est à réaliser il suffit simplement de décompresser le fichier téléchargé et de faire un double clic sur le fichier eclipse.exe pour lancer le logiciel.

- Etape2 : Téléchargement du SDK Android

Passons maintenant au téléchargement du SDK Android sur le site Android. Téléchargez la version pour Windows, et décompressez le fichier zip où vous le souhaitez sur votre disque dur (*C:\Users\Matthias\Documents\LP_SEICOM\Projet_Tut\adt-bundle-windows-x86_64-20131030*).

Nous allons maintenant ajouter le chemin du SDK Android à la variable Path du système. Ceci permettra d'utiliser les outils du SDK sans devoir spécifier le chemin complet à chaque fois. La manipulation est légèrement différente en fonction du système d'exploitation Windows dans notre cas il s'agira de Windows7.

Clic droit sur **Ordinateur > Propriétés > Protection du système** (à gauche), ensuite cliquez sur l'onglet **Paramètres système avancés** puis sur le bouton **Variable d'environnement**. Sélectionnez dans les Variables système, la variable **Path** puis cliquez sur Modifier. Il suffit de rajouter à la fin de la ligne un point-virgule et d'ajouter le chemin du SDK décompressé.

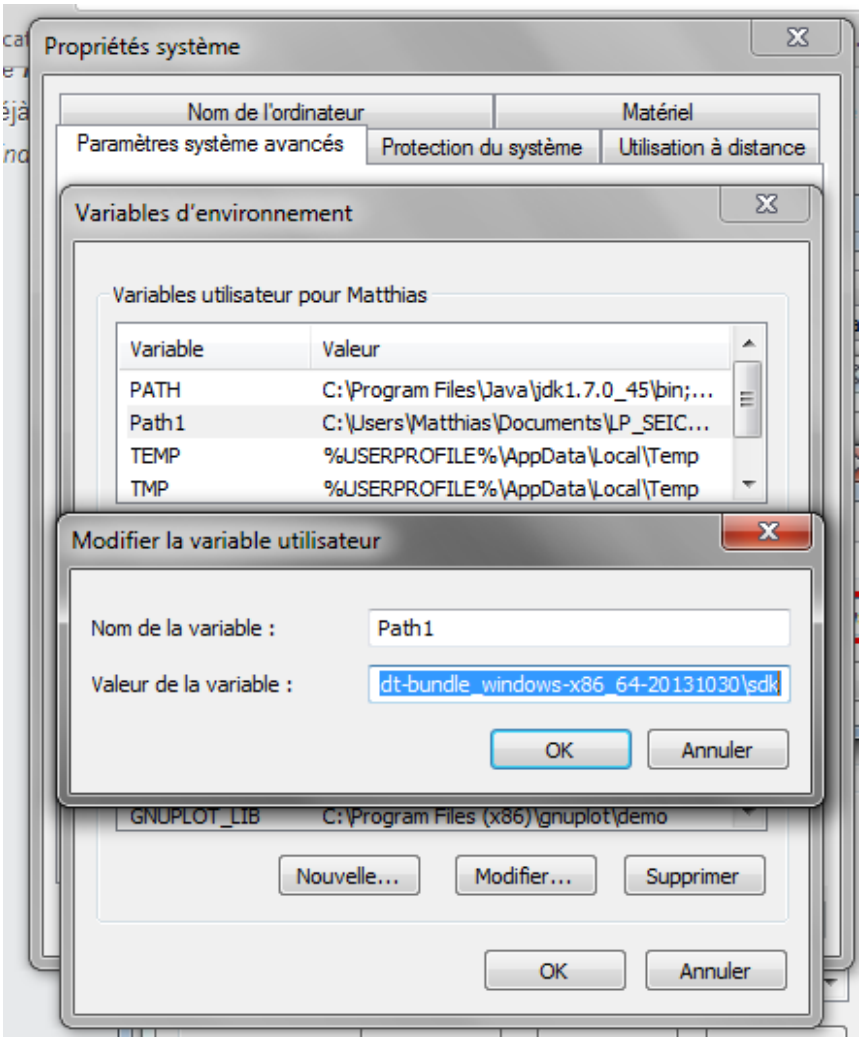


FIGURE 9 - MODIFICATION VARIABLE

- Etape3 : Installation du plugin Android pour Eclipse

Lancez Eclipse, puis allez dans le menu **Help > Install New Software**, puis cliquez sur le bouton **Add** et remplissez le formulaire. Dans le champ Name écrivez par exemple Plugin Android et dans le champ Location indiquez l'adresse suivante : **https://dl-ssl.google.com/android/eclipse/** puis cliquez sur OK.

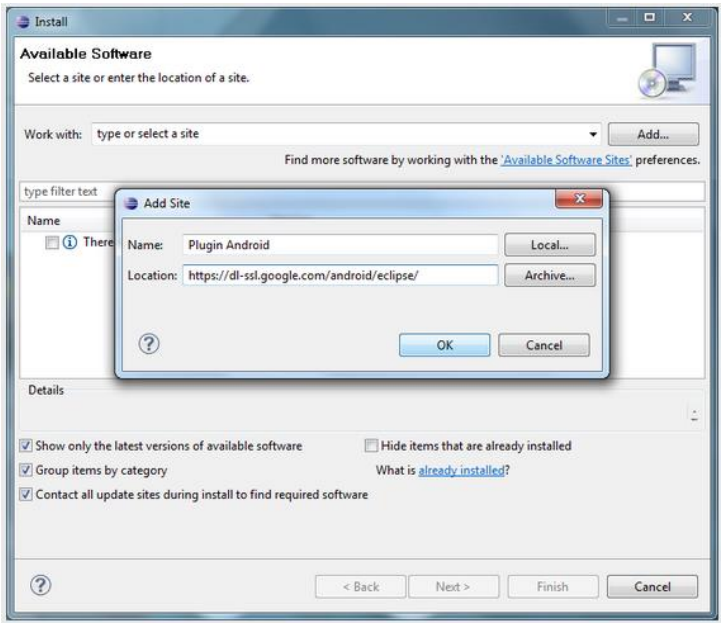


FIGURE 10 - INSTALLATION PLUGIN ANDROID

Cochez **Developer Tools** puis cliquez sur **Next** et encore une fois sur **Next**. Cochez ***« I accept the terms of the license agreements »*** puis cliquez sur **Finish**

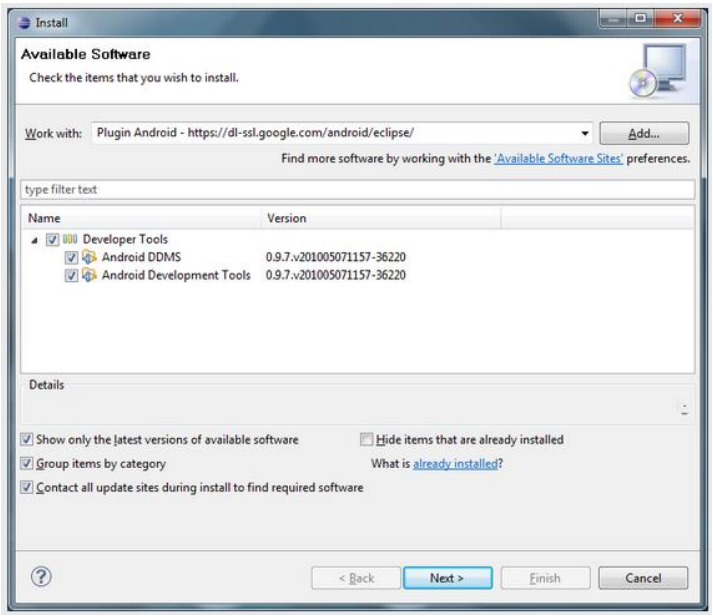


FIGURE 11 - INSTALLATION PLUGIN ANDROID ETAPE 2

Un message d'avertissement devrait apparaître vous signalant que le logiciel que vous essayez d'installer n'est pas signé, cliquez sur OK puis cochez le certificat Eclipse.org Foundation puis cliquez sur OK.

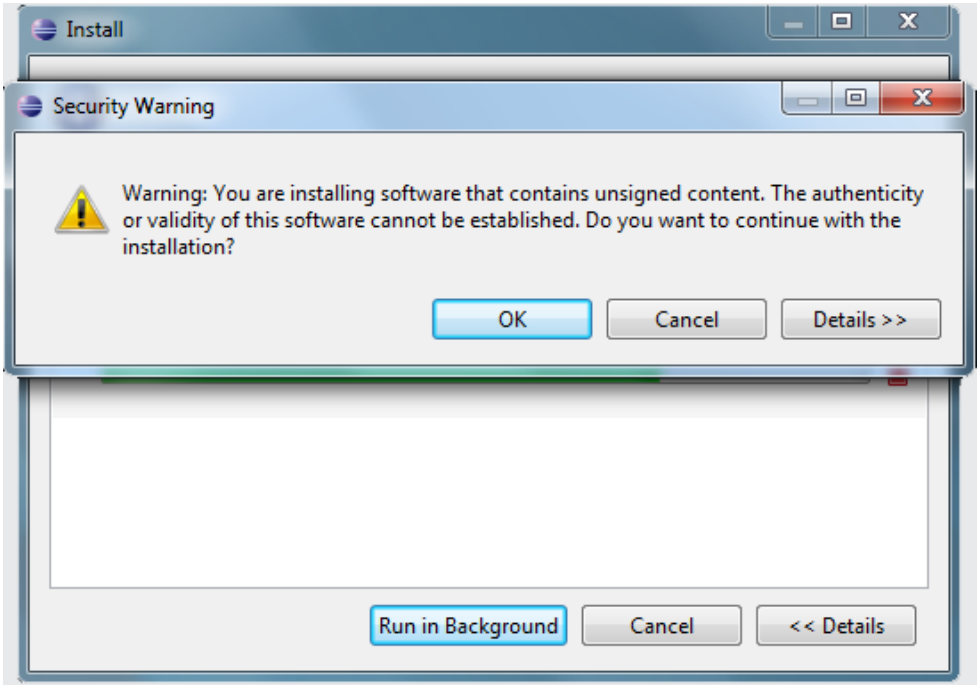


FIGURE 12 - AVERTISSEMENT

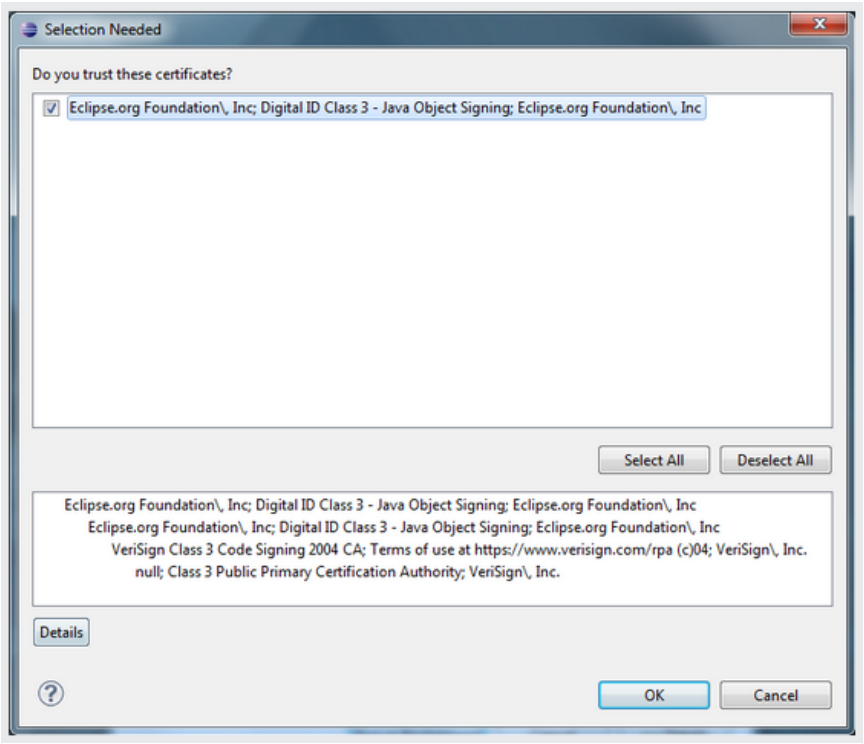


FIGURE 11 – INSTALLATION

Une fois l'installation finie redémarrez Eclipse. Une fois Eclipse relancé, allez dans le menu **Windows > Préférences > Android** puis **SDK Location** allez chercher le dossier dans lequel se trouve le SDK que l'on a décompressé puis cliquez sur OK.

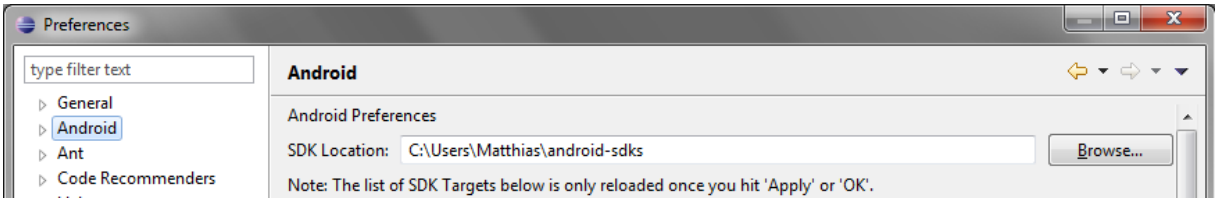
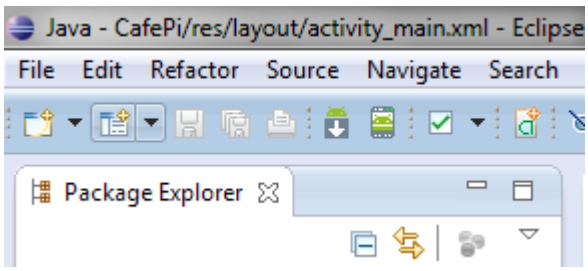


FIGURE 124 - CHEMIN DE LA SDK

Désormais le SDK Android et le plugin Android pour Eclipse sont installés. Il ne reste plus qu'à configurer le plugin avec le SDK.

- <u>Etape4 : Configuration du plugin Android</u>



Cliquez sur le petit icône encadré en rouge sur l'image ci-dessous

Cliquez sur Installed Packages puis sélectionnez le package Android SDK Tools et cliquez sur Update All

Cochez Accept All puis cliquez sur Install.

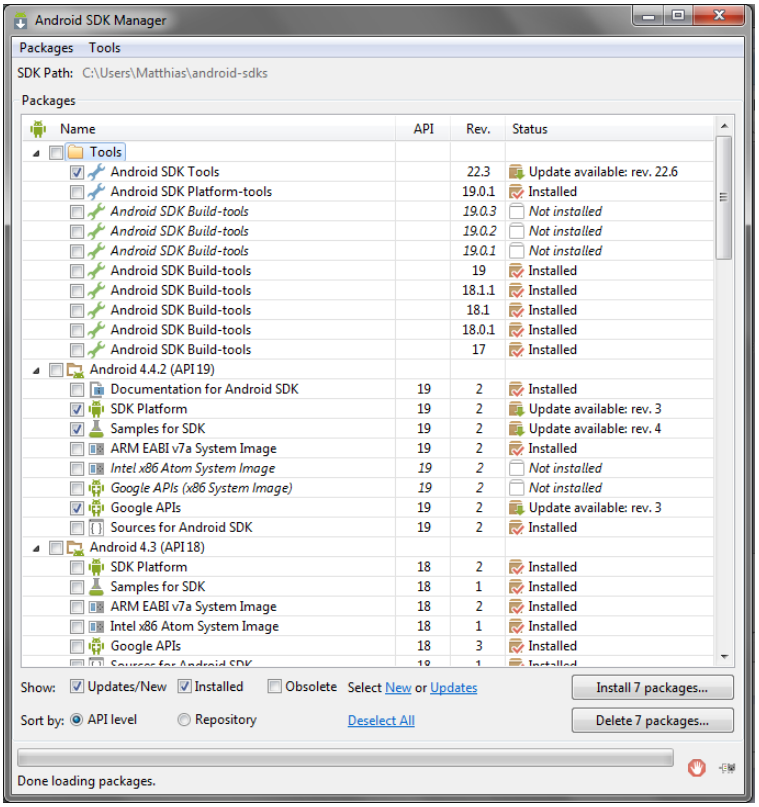


FIGURE 15 - INSTALLATION PACKAGES

B. DEVELOPPEMENT DE L'APPLICATION

Pour le développement de notre application Android nous avons choisis de faire une WebApplication comme préciser précédemment celle-ci a était développé en Java sous Eclipse. Pour les explications nous allons suivre le fil suivant :

- Présentation de l'outil
- Présentation des différents layout
- Emulateur d'application Android

1. PRÉSENTATION DE L'OUTIL

Sous Eclipse nous pouvons créer des interfaces graphiques à la souris, il est en effet possible d'ajouter un élément et de le positionner grâce à la souris. C'est à l'aide du menu en haut, celui visible à la figure suivante, que vous pourrez observer le résultat avec différentes options

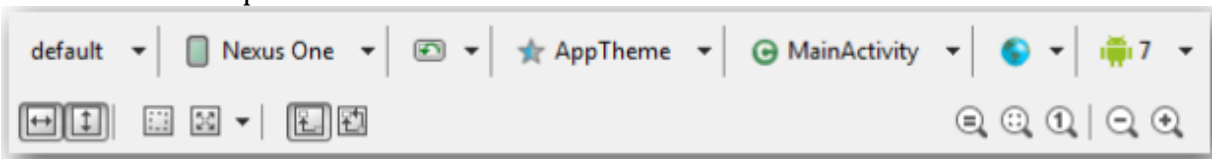


FIGURE 13 - MENU D'OPTION

Ce menu est divisé en deux parties : les icônes du haut et celles du bas. Nous allons nous concentrer sur les icônes du haut pour l'instant (voir figure suivante).

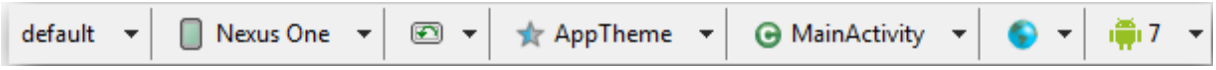


FIGURE 14 - BARRES DU HAUT MENU D'OPTION

- La première liste déroulante vous permet de naviguer rapidement entre les répertoires de layouts.
- La deuxième permet d'observer le résultat en fonction de différentes résolutions. Le chiffre indique la taille de la diagonale en pouces (sachant qu'un pouce fait 2,54 centimètres, la diagonale du **Nexus One** fait 3,7x2,54 = 9 ,4 cm) et la suite de lettres en majuscule la résolution de l'écran.
- La troisième permet d'observer l'interface graphique en fonction de certains facteurs. Se trouve-t-on en mode portrait ou en mode paysage ? Le périphérique est-il attaché à un matériel d'amarrage ? Enfin, fait-il jour ou nuit ?
- La suivante permet d'associer un thème à votre activité.
- L'avant-dernière permet de choisir une langue si votre interface graphique change en fonction de la langue.
- Et enfin, la dernière vérifie le comportement en fonction de la version de l'API, si l'on définit des quantificateurs à ce niveau-là.

Occupons-nous maintenant de la deuxième partie, tout d'abord avec les icônes de gauche, visibles à la figure suivante.

FIGURE 15 - LES ICONES DE GAUCHE DU MENU

Ces boutons sont spécifiques à un composant et à son layout parent, contrairement aux boutons précédents qui étaient spécifiques à l'outil. Ainsi, si vous ne sélectionnez aucune vue, ce sera la vue racine qui sera sélectionnée par défaut. Comme les boutons changent en fonction du composant et du layout parent, je ne vais pas les présenter en détail. Enfin, l'ensemble de boutons de droite, visibles à la figure suivante.

FIGURE 16 - LES ICONES DE DROITE DU MENU

- Le premier bouton permet de modifier l'affichage en fonction d'une résolution que vous choisirez. Très pratique pour tester, si vous n'avez pas tous les terminaux possibles.
- Le deuxième fait en sorte que l'interface graphique fasse exactement la taille de la fenêtre dans laquelle elle se trouve.
- Le suivant remet le zoom à 100%.
- Enfin les deux suivants permettent respectivement de dézoomer et de zoomer.

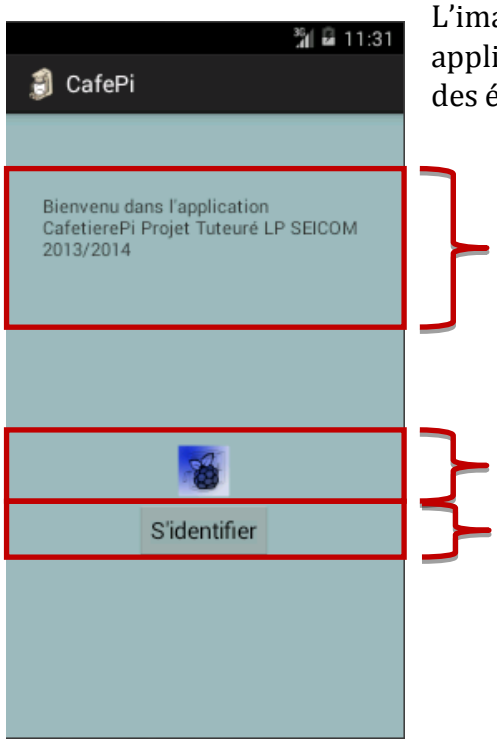
Apres vous avoir expliqué comment s'articule Eclipse autour de la création de Layout Android, je vous expliquerai comment nous avons développé et créé nos layout.

2. PRÉSENTATION DES DIFFÉRENTS LAYOUT

Dans cette partie je vais vous présenter les différent layout de notre application vous y trouverez une vue graphique avec les codes associés (*xml* et *java*).

A) LAYOUT DE PRESENTATION :

L'image suivante représente le premier layout de notre application Android de la CafetierePi celui est composé des éléments suivant :



TextView : Un élément basique, il s'agit d'une simple zone de texte que l'on peut personnaliser aussi bien graphiquement qu'au niveau du comportement selon les besoins de l'application.

ImageView : est une boite qui peut contenir une image

Button : A pour but que l'on puisse effectuer un click dessus.

FIGURE 20 - LAYOUT DE PRESENTATION

Pour l'implémentation de ce premier layout nous avons dû définir les différents paramètres concernant ces trois éléments ces paramètre sont répertorier dans le fichier *xml. (voir annexe application android source xml)*

En *annexe application android source java* vous trouverez le code java correspondant au fonctionnement des différents éléments présents sur ce premier layout de notre application Android.

B) LAYOUT D'IDENTIFICATION

L'image suivante représente le deuxieme layout de notre application Android de la cafetierePi celui est composé des éléments suivant :

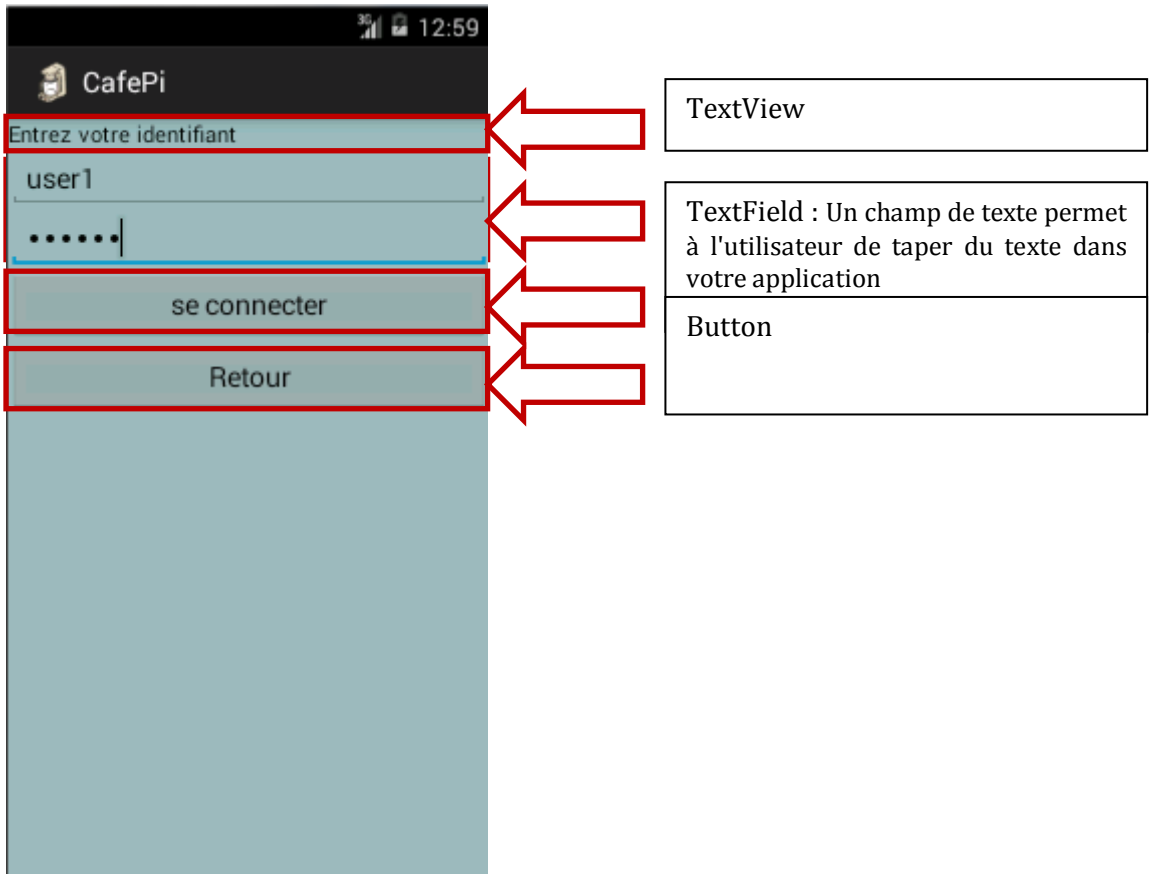


FIGURE 18 - LAYOUT D'IDENTIFICATION

Pour l'implémentation de ce layout nous avons dû définir les différents paramètres concernant ces cinq éléments ces paramètres sont répertorier dans le fichier *xml. (Voir annexe application android source xml)*

En *annexe application android source java* vous trouverez le code java correspondant au fonctionnement des différents éléments présents sur ce premier layout de notre application Android.

C) LAYOUT WEB

Les images suivantes représentent les deux layout de la partie web de notre application Android de la cafetierePi celui est composé des éléments suivant :

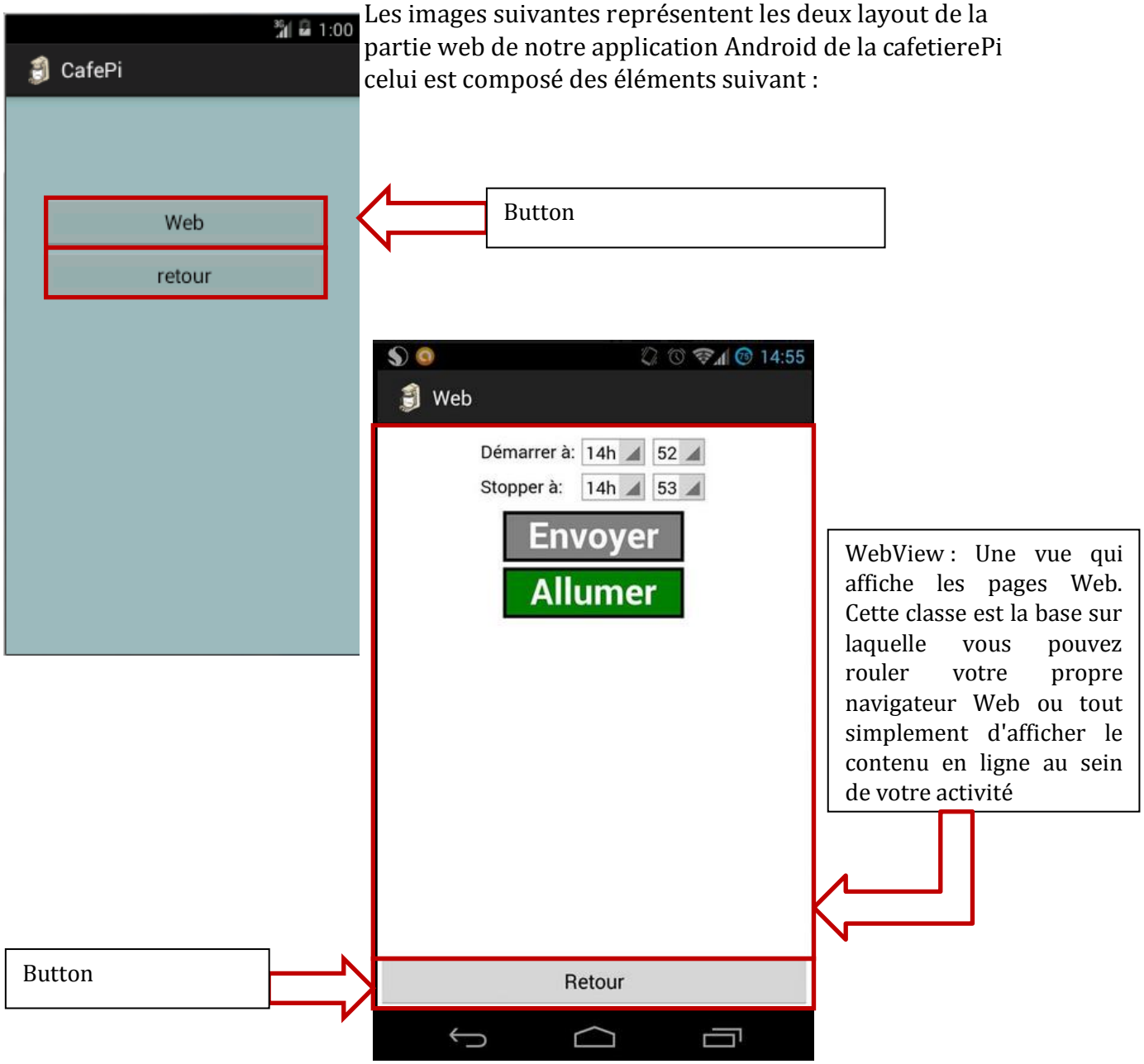


FIGURE 19 - LAYOUT WEB

Pour l'implémentation de ces layout nous avons dû définir les différents paramètres concernant les éléments ces paramètre sont répertorier dans le fichier *xml. (voir annexe application android source xml)*

En *annexe application android source java* vous trouverez le code java correspondant au fonctionnement des différents éléments présents sur ce premier layout de notre application Android.

IX. CAFETIERE ET RASPBERRY-PI

A. INSTALLATION DU RASPBERRY-PI

Lorsque l'on sort un Raspberry de sa boite, il ne peut pas fonctionner tout de suite. Il faut au préalable installer un système d'exploitation (OS). Plusieurs OS sont disponibles pour le RasPi et nous avons choisi d'utiliser Raspbian qui est un système d'exploitation libre et gratuit basé sur Linux/GNU/Debian et optimisé pour fonctionner sur un Raspberry Pi. Étant donné les ressources limitées de cet ordinateur, Raspbian utilise des logiciels réputés pour être légers tels que le gestionnaire de fenêtres LXDE et le navigateur internet Midori.

L'installation est très simple, voici comment faire sous **Windows** :

- Connecter la **carte SD** au PC et noter la lettre attribuée par **Windows.**
- Télécharger l'utilitaire Win32DiskManager et décompresser le fichier ZIP obtenu.
- Télécharger l'image de Raspbian sur le site raspberrypi.org et la décompresser.
- Ensuite, lancer **Win32DiskImager** en administrateur.
- Dans **Win32DiskImager**, choisir la lettre de la **carte SD** et ajouter le fichier 2013-xx-xx-wheezy-raspbian.img téléchargé préalablement.
- Une fois que l'image est bien ajoutée et le lecteur de votre **carte SD** correctement renseigné, il suffit de cliquer sur **Write**.

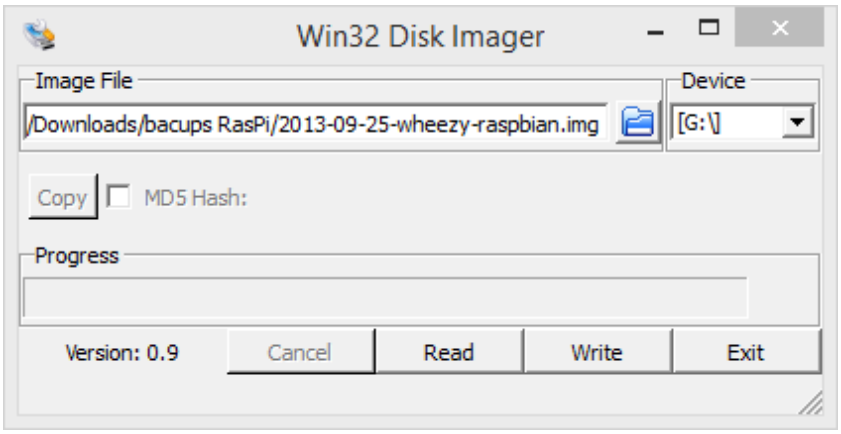


FIGURE 20 – WIN32 DISK IMAGER

Une fois l'installation terminée, éjectez la carte SD puis L'insérer dans le RasPi.

Sous linux, l'installation ce fait dans un terminal, pas besoin de logiciel.

-Ouvrir un terminal et connecter la carte SD au lecteur. Dans le terminal, taper :

```
df -h
```

-Un tableau comme celui-ci apparait :

```
Filesystem        Size   Used  Avail  Capacity  Mounted on
/dev/disk0s2      698Gi  429Gi  269Gi 62%      /
devfs             194Ki  194Ki  0Bi   100%     /dev
map -hosts        0Bi    0Bi    0Bi   100%     /net
map auto_home     0Bi    0Bi    0Bi   100%     /home
/dev/disk1s1      3.8Gi  2.2Mi  3.8Gi 1%       /Volumes/UNTILTED
```

-La ligne correspondant à la carte mémoire va nous permettre de savoir quel nom le PC lui a attribué. Il s'agit de la ligne avec la valeur qui se rapproche le plus de la carte SD, ici 3.8Go pour une carte de 4Go c'est la bonne ligne :

```
/dev/disk1s1
```

-Il faut maintenant démonter la carte sd :

```
diskutil unmount /dev/disk1s1
```

-Puis nous allons écrire le contenu du fichier .img sur la carte.

```
dd bs=1M if=~/Downloads/2013-09-25-wheezy-raspbian.img of=/dev/rdisk1
```

-Une fois terminé, le terminal retourne :

```
1850+0 records in
1850+0 records out
1939865600 bytes transferred in 101.411969 secs (19128567 bytes/sec)
```

-On éjecte la carte avec la commande :

```
diskutil eject /dev/rdisk1
```

L'installation de **Raspbian** est terminée, il ne reste plus qu'à brancher le **Raspberry Pi à un écran,** au câble Ethernet, puis enfin à l'alimentation électrique, un chargeur de téléphone 5V 700mA, afin qu'il démarre.

Au premier démarrage un panneau de configuration s'affiche (aussi accessible via la commande Raspi-config):

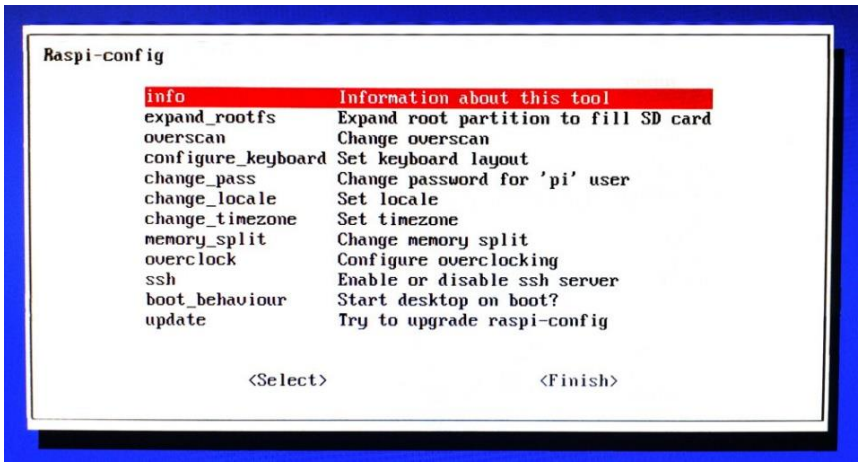


FIGURE 21 – RASPI-CONFIG

- **expand_rootfs :** Va permettre d'utiliser tout l'espace de la carte SD
- **configure_keyboard :** Pour mettre le clavier en français on sélectionne Generic 105-key (intl) PC puis Other puis French puis French puis The default for the keyboard layout puis No compose key puis No
- **change_pass :** modifier le mot de passe de l'utilisateur 'pi' (par défaut : raspberry)
- **change_locale :** changer la langue du système
- **change_timezone :** changer l'heure locale
- **ssh :** activer le SSH
- **boot_behavior :** Pour choisir si on veut arriver directement sur le bureau au démarrage du système et non pas sur une console.

Une fois les configurations terminées, le RaspberryPi redémarre et nous demande login (pi) et mot de passe (raspberry).

B. LA CONNEXION SSH

Pour éviter d'utiliser un second écran et clavier, il est possible de prendre la main sur le RasPi à distance via un PC grâce à la connexion SSH (Secure Shell). Il faut l'avoir préalablement activée dans le panneau de configuration vu précédemment.

Sous Windows, nous utilisons le logiciel Putty qui permet d'établir une connexion SSH. Sa configuration est très simple.

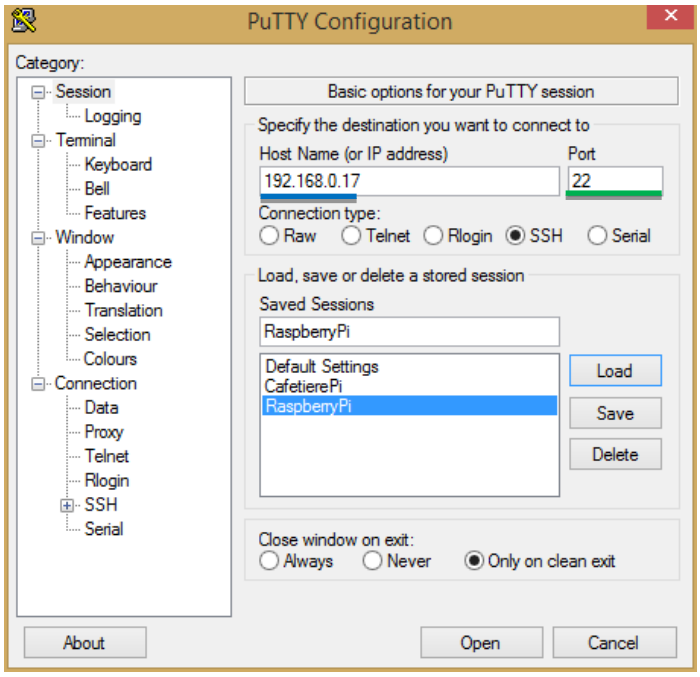


FIGURE 22 – CONFIGURATION PUTTY

Il suffit d'entrer l'adresse IP du RasPi ainsi que le numéro de port puis de cliquer sur Open.

On peut connaitre l'adresse IP via la commande ifconfig eth0

Le port 22 est un port réservé au SSH ainsi que pour l'échange de fichiers sécurisés SFTP (Secure File Transfer Protocol).

Une fois la connexion établie, un terminal s'ouvre, nous demande de ce logger et il nous est possible de contrôler le RasPi comme avec un second écran/clavier.

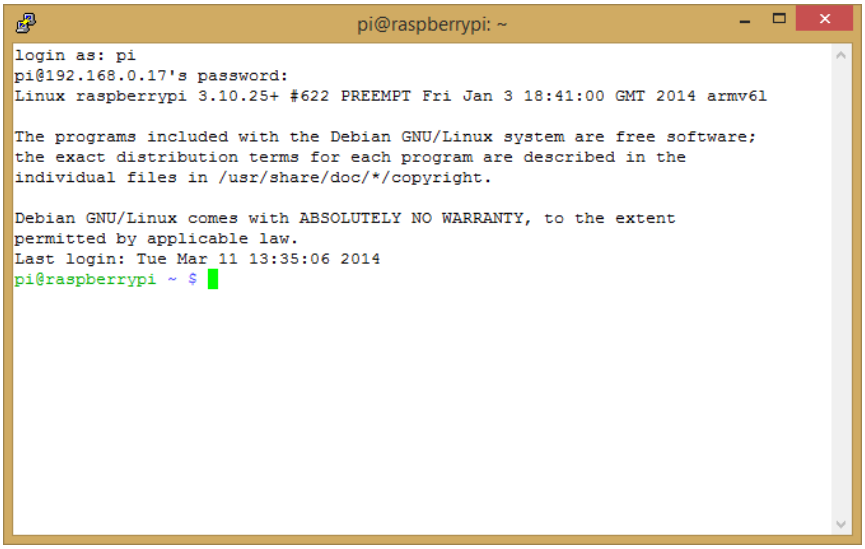


FIGURE 23 – TERMINAL PUTTY

C. L'UTILISATION DES PORTS GPIO

Le RaspberryPi possède, en plus des connectiques classiques (USB, HDMI, VGA…) des ports GPIO ce qui signifie General Purpose Input/Output. Ce qui peut se traduire par entrées/sorties numérique.

Ces entrées/sorties permettent d'étendre les fonctionnalités du RasPi en lui donnant la possibilité d'agir sur des LEDs ou des afficheurs LCD par exemple, lire l'état d'un interrupteur, d'un capteur, etc...

Ce connecteur GPIO dispose de différents types de connexion :

- Des broches utilisables en entrée ou sortie numériques tout ou rien
- Des broches pour une interface I2C (permettant de se connecter sur du matériel en utilisant uniquement 2 broches/pins de contrôle.)
- Une interface SPI pour les périphériques SPI,
- Les broches Rx et Tx pour la communication avec les périphériques séries.
- De broches pouvant être utilisées en PWM ("Pulse Width Modulation") permettant de contrôler des servo moteurs par exemple.

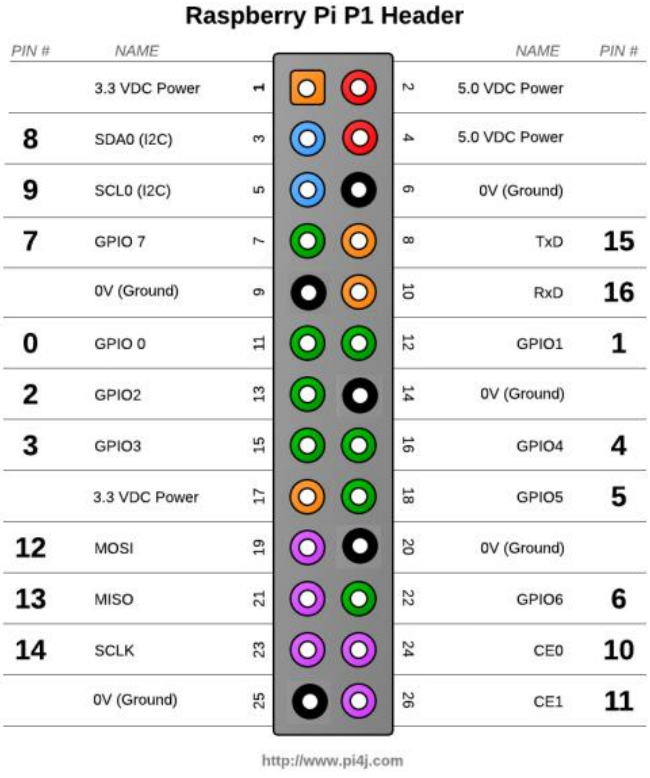


FIGURE 24 – BROCHAGE GPIO

Les broches du port GPIO sont alimentées en **3,3V**. Seules les deux broches d'arrivée de courant (2 et 4) fournissent une tension de **5V**.

Ce qui veut dire qu'en mode ouput (sortie) la tension fournie par une broche est de 3,3V et délivrera un courant de 50mA max. Cela risque d'être un problème du fait que le programmateur envois du 5V au relais mais nous verrons cela plus tard.

Le contrôle des ports ce fait depuis le terminal.

Exemple pour passer l'état de la broche 18 (GPIO 24) à 1 :

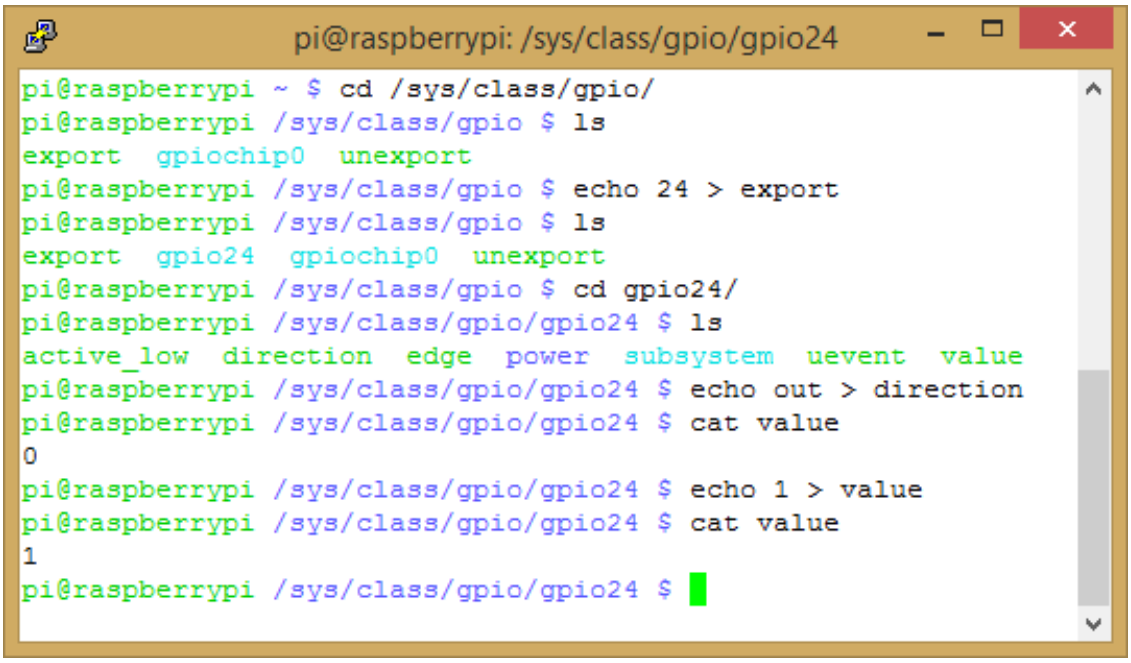


FIGURE 25 – UTILISATION GPIO TERMINAL

Cette méthode n'étant pas très pratique, nous avons cherché d'autres manières de procéder.

Après quelques recherches, nous avons trouvé la librairie wiringPi qui permet de contrôler les ports plus facilement, toujours dans le terminal :

Installation de la librairie :

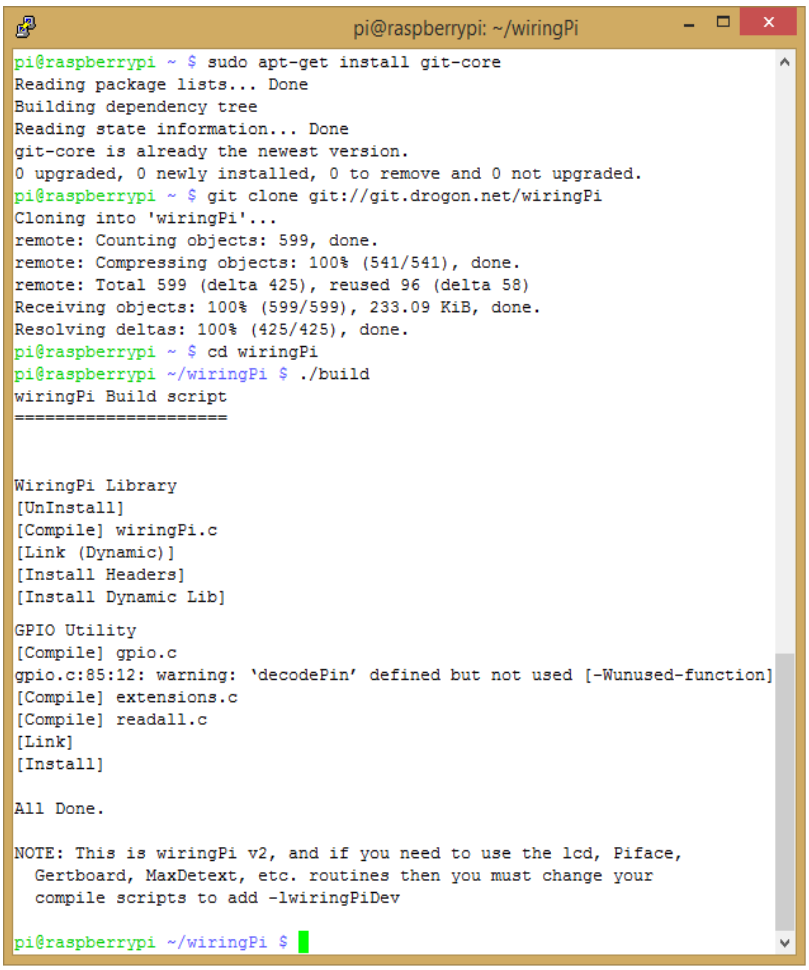
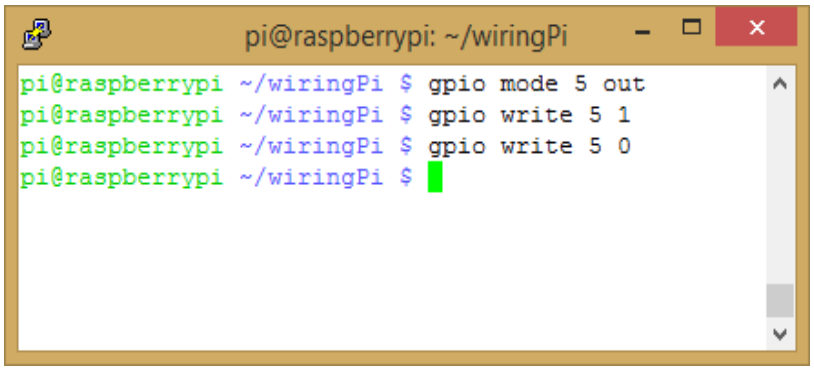


FIGURE 26 – INSTALLATION WIRINGPI

Utilisation de wiringPi :



Pour utiliser cette librairie, il suffit de dire dans quel mode nous voulons que le port soit. Dans l'exemple ci-dessus, le GPIO5 est mis en sortie (remplacer out par in pour le mettre en entrée). Ensuite, nous écrivons sur le GPIO5 la valeur 1 puis la valeur 0

FIGURE 27 – UTILISATION WIRINGPI

D. RELATION MATÉRIELLE CAFETIÈRE/RASPBERRY-PI

Maintenant que nous savons comment manipuler les GPIO, nous allons réaliser une interface afin d'interagir le RasPi avec la cafetière.

Nous avons réalisé un schéma structurel de cette interface :

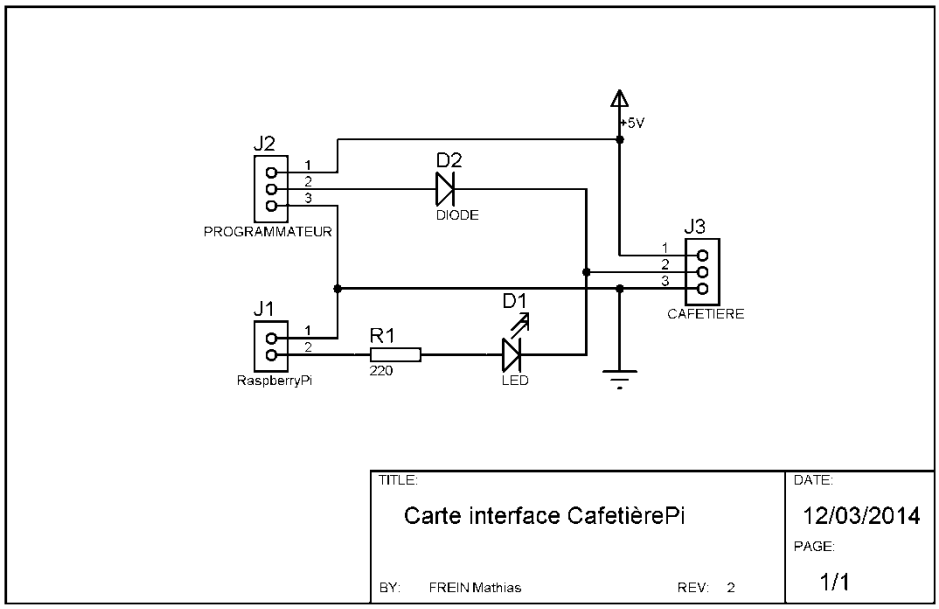


FIGURE 28 – SCHEMA STRUCTUREL CARTE INTERFACE

Cette carte est assez simple. La LED D1 va servir à avertir que la cafetière a été allumée à distance. Elle va aussi servir de diode pour éviter que si le programmateur envoi un du 5V, celui-ci n'aille pas vers le RasPi qui lui n'enverrai rien et qui serai donc au niveau logique 0. La diode D2 a la même utilité mais pour le cas inverse.

Pour voir si ce schéma fonctionne, nous avons réalisé un prototype sur une platine d'essai.

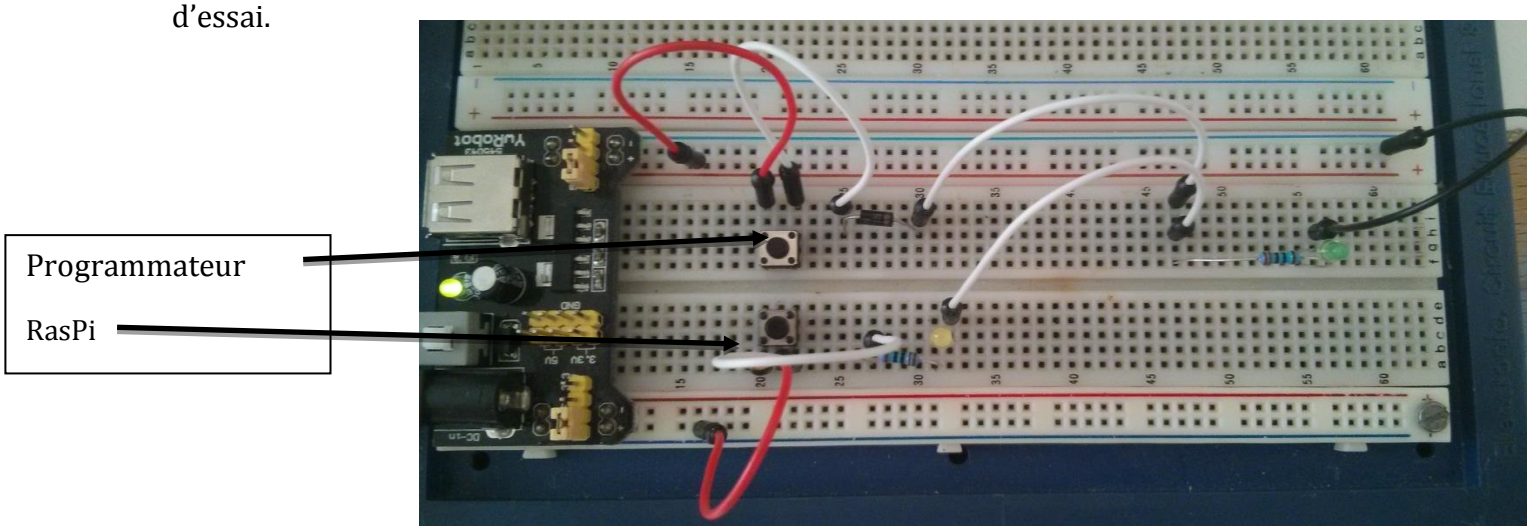


FIGURE 29 – PROTOTYPE CARTE INTERFACE

Les boutons poussoirs simulent les niveaux logiques du Programmateur et du RasPi.

La LED verte tout à droite simule l'état du relais.

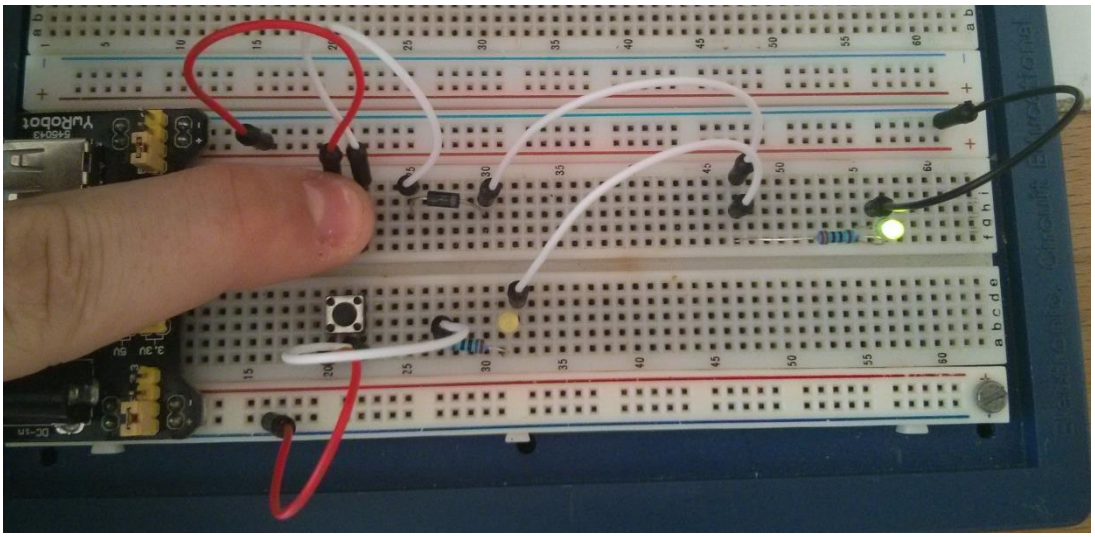


FIGURE 30 – TEST 1 PROTOTYPE

Si le programmateur est à 1et le RasPi à 0, la LED relais (verte) s'active mais pas la LED de contrôle à distance (jaune).

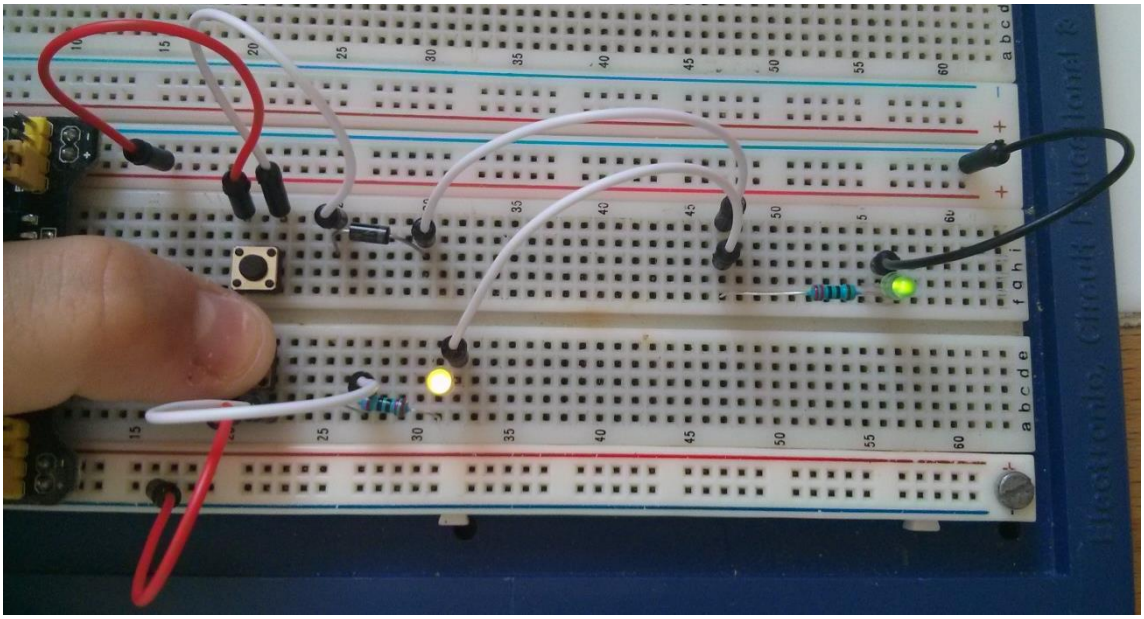


FIGURE 31 – TEST 2 PROTOTYPE

Si le programmateur est à 0 et le RasPi à 1, la LED relais (verte) s'active ainsi que la LED de contrôle à distance (jaune).

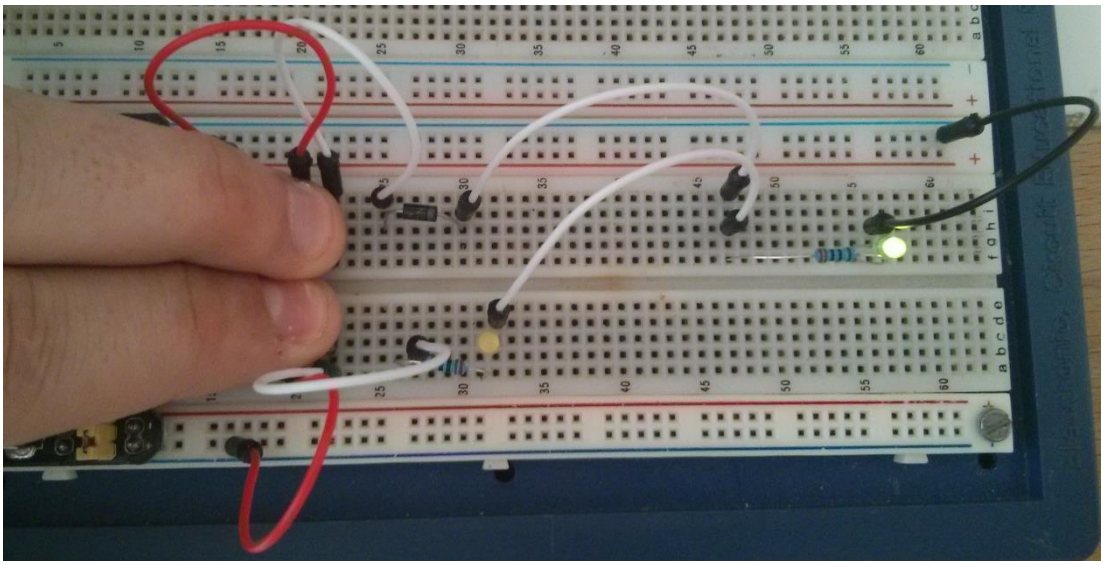


FIGURE 32 – TEST 3 PROTOTYPE

Si le programmateur est à 1 et le RasPi à 1, la LED relais (verte) s'active mais pas la LED de contrôle à distance (jaune).

Nous avons ensuite testés sur la carte de contrôle de la cafetière. Aucuns soucis, nous entendons 'claquer' le relais cela signifie que le 3,3V du GPIO suffi à saturer le transistor qui le commande. Tout fonctionne comme prévu.

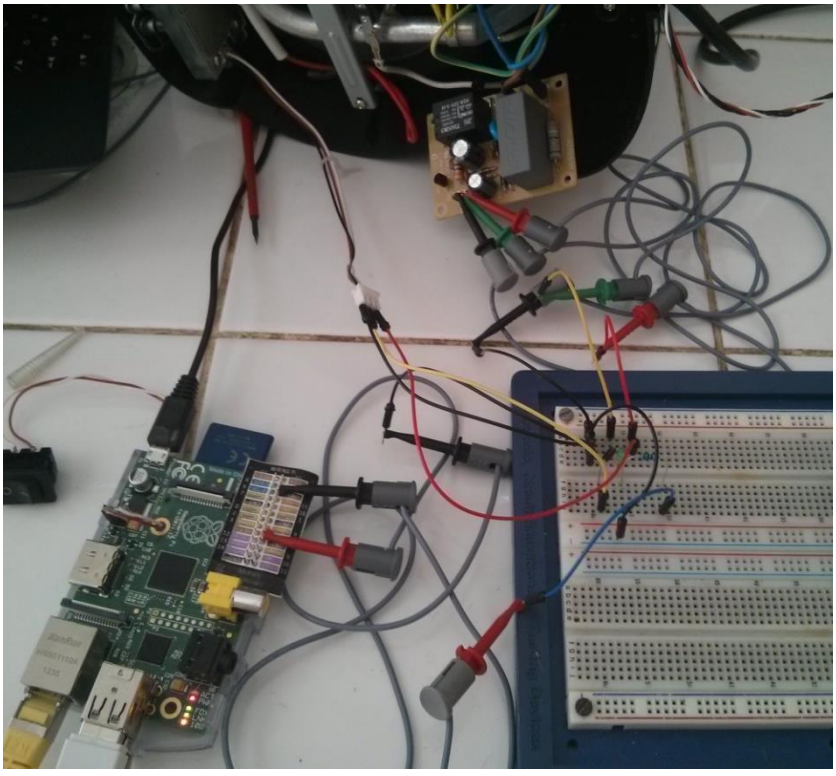


FIGURE 33 – TEST PROTOTYPE SUR CAFETIERE

Une fois le prototype validé, nous sommes passés à la réalisation de la maquette.

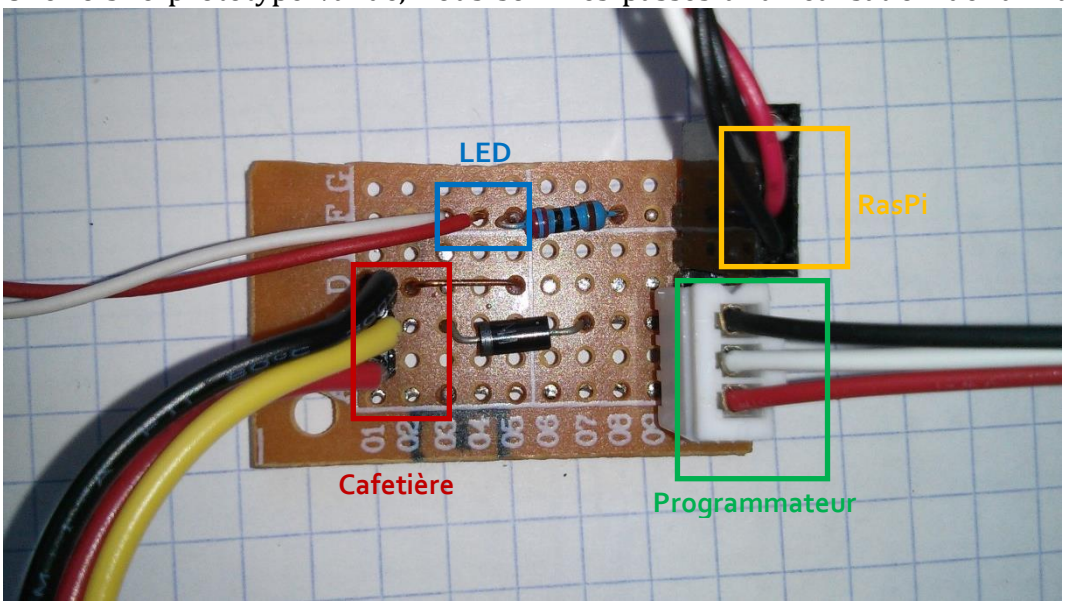
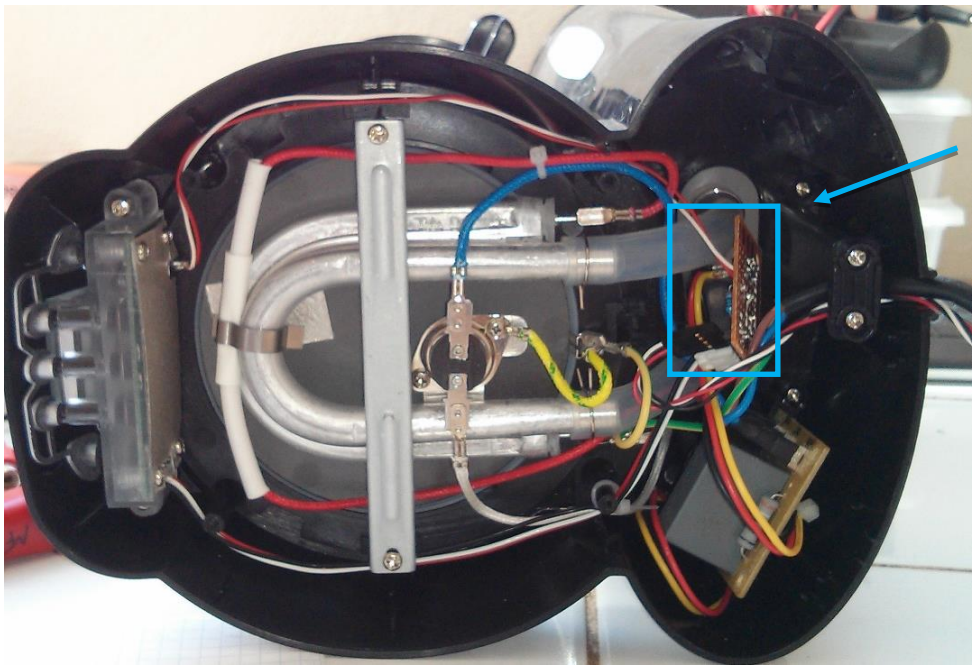


FIGURE 34 – MAQUETTE INTERFACE

Nous pouvons dire que cette carte n'a rien coutée car elle a été réalisée avec du matériel de récupération et des composants que je possédais déjà.

Nous avons choisis de déporter la LED de contrôle à distance vers le programmateur, car une fois la carte d'interface placée dans la cafetière, celle-ci ne sera plus visible.

Pour éviter d'avoir une troisième LED, nous avons décidés de changer celle qui était déjà présente d'origine sur le programmateur indiquant par une lumière rouge que la cafetière est en marche. Nous l'avons donc remplacée par une LED bicolore. De ce fait, lorsque le programmateur allume la cafetière, la LED est rouge et lorsque c'est RasPi, la LED est verte.



La carte d'interface loge parfaitement dans la cafetière. Nous avons dût faire attention à ce que ni la carte, ni les fils ne soient trop près de la résistance de chauffe afin qu'ils ne soient pas détériorés par la chaleur.

FIGURE 35 – INSTALLATION MAQUETTE

FIGURE 36 – TEST MAQUETTE

La LED bicolore joue parfaitement son rôle et l'on entend claquer le relais. Tous les contrôles d'origines du programmateur sont restés fonctionnels.

NB :

Je tiens à préciser que par mesure de sécurité la résistance de chauffe a été laissée débranchée afin d'éviter d'éventuelles brulures.

E. DÉVELOPPEMENT DE L'INTERFACE HOMME MACHINE

Pour qu'un smartphone puisse contrôler la cafetière, il faut avant tout qu'il puisse s'y connecter. Pour ce faire, nous avons choisis d'utiliser la technologie de communication sans fil WIFI. Il faut donc créer un point d'accès (AP) sur le RasPi.

Dans un premier lieu, il faut vérifier que le dongle (clé wifi) est bien reconnu par le RasPi grâce à la commande lsusb qui va lister l'ensemble des périphériques connecté sur le bus USB.

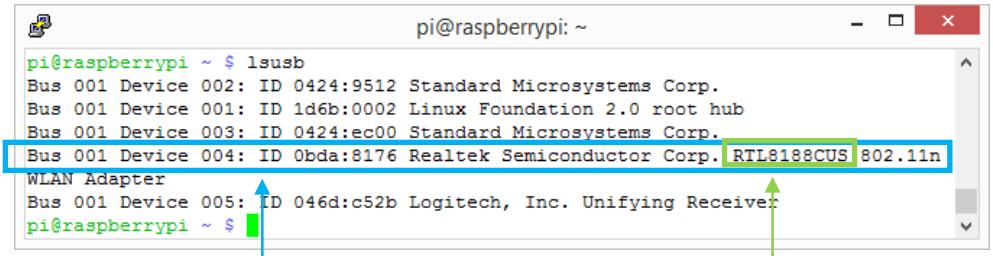


FIGURE 37 – INSTALLATION DONGLE WIFI

On peut voir ici que le dongle est bien reconnu et installé. On peut aussi remarquer quel chipset il utilise. Ceci nous sera utile pour la suite. Le logiciel que nous allons utiliser pour créer l'AP est hostapd. Il s'installe simplement avec la commande

sudo apt-get install hostapd.

La version d'origine de hostapd le fonctionne qu'avec les dongles qui utilisent le chipset nl80211. Il ne fonctionnera donc pas avec notre dongle ce qui nous à value de nombreux problèmes... Heureusement, nous avons fini par trouver une version qui a été modifiée pour fonctionner avec notre dongle sur www.daveconroy.com.

Nous allons donc la télécharger et la remplacer grâce aux commandes suivantes :

```
wget http://www.daveconroy.com/wp3/wpcontent/upload/2013/07/hostapd.zip
unzip hostapd.zip
sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.bak
sudo mv hostapd /usr/sbin/hostapd.edimax
sudo ln -sf /usr/sbin/hostapd.edimax /usr/sbin/hostapd
sudo chown root.root /usr/sbin/hostapd
sudo chmod 755 /usr/sbin/hostapd
```

Maintenant que la bonne version d'hostapd est installée, il faut maintenant le configurer.

Pour ce faire, il faut éditer le fichier de configuration.

```
sudo nano /etc/hostapd/hostapd.conf
```

Voici ce qu'il doit contenir :

```
interface=wlan0
driver=rtl871xdrv
ssid=CafetièrePi
channel=6
wmm_enabled=1
wpa=1
wpa_passphrase=SEICOM2014
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
auth_algs=1
macaddr_acl=0
```

Maintenant que la bonne version d'hostapd est installée, il faut maintenant le configurer.
Pour ce faire, il faut éditer le fichier de configuration.

```
sudo nano /etc/hostapd/hostapd.conf
```

Il faut ensuite définir une adresse IP fixe au dongle.

Pour cela, il faut modifier le fichier interface

```
sudo nano /etc/network/interfaces
```

Il doit contenir ceci :

```
auto lo
iface lo loopback
iface eth0 inet static
    address 192.168.0.17
    netmask 255.255.255.0
    gateway 192.168.0.1

allow hotplug
iface wlan0 inet static
    address 192.168.42.1
    netmask 255.255.255.0
```

Ensuite, nous devons installer puis configurer un serveur DHCP (Dynamic Host Configuration Protocol) afin que les périphériques qui se connectent au point d'accès puissent obtenir une adresse IP.

Pour ce faire nous allons installer isc-dhcp-server via la commande suivante :

```
sudo apt-get install isc-dhcp-server
```

Pour sa configuration il faut modifier le fichier suivant:

```
sudo nano /etc/dhcp/dhcpd.conf
```

Premièrement, il faut s'assurer que les lignes suivantes soient bien décommentées. Il ne doit pas y avoir de # au début de la ligne :

```
authoritative;
ddns-update-style none;
default-lease-time 600;
max-lease-time 7200;
log-facility local7;
```

Puis il faut ensuite ajouter ceci à la toute fin:

```
subnet 192.168.42.0 netmask 255.255.255.0 {
range 192.168.42.25 192.168.42.50 ;
option domain-name-servers 8.8.8.8, 8.8.4.4 ;
option routers 192.168.42.1
interface wlan0 ;
}
```

Ceci détermine la plage d'adresses IP qui vont pouvoir être alloués aux différents périphériques qui vont venir se connecter

Une fois la configuration terminée, il faut redémarrer le RasPi pour qu'il prenne bien en compte les changements.

Maintenant que le RasPi est prêt, il faut redémarrer le serveur DHCP :

```
sudo /etc/init.d/isc-dhcp-server restart
```

Puis démarrer hostapd avec le paramètre –B qui va permettre que hostapd s'exécute en arrière-plan et ainsi nous pouvons continuer à utiliser le RasPi :

```
sudo hostapd -B /etc/hostapd/hostapd.conf
```

Pour que hostapd se lance seul au démarrage nous allons en faire un deamon :

```
sudo nano /etc/default/hostapd
```

Puis il décomenter et mettre à jour la ligne suivante :

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Le point d'accès est maintenant opérationnel on peut s'y connecter avec un smartphone.

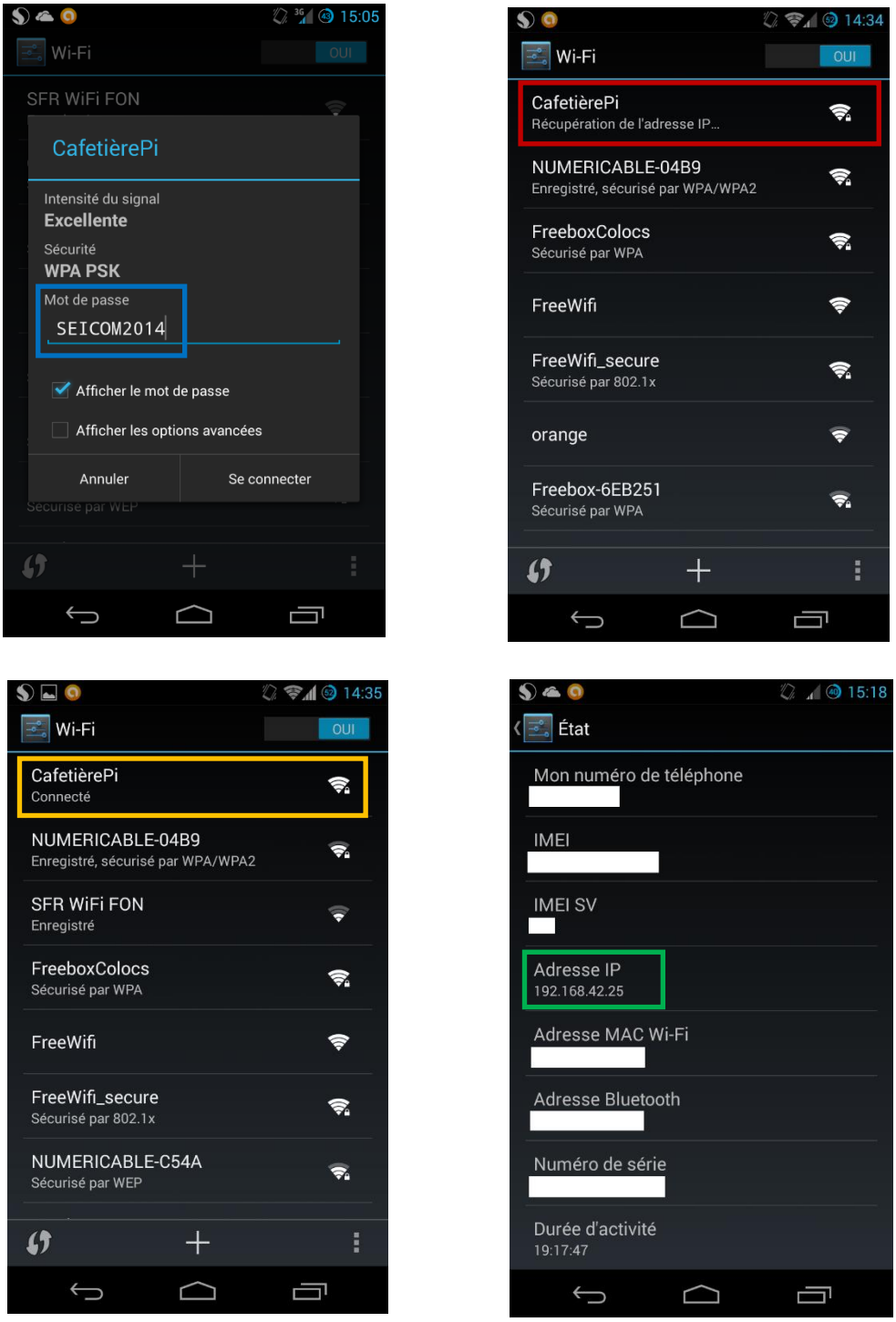


FIGURE 38 – CONNEXION POINT ACCES WIFI

Une fois avoir rentré le mot de passe que nous avions choisis, le téléphone se voit attribué une adresse IP par le serveur DHCP et réussi à se connecter au réseau que nous venons de créer. Sa nouvelle adresse IP est 192.168.42.25, la 1ère disponible par le sous réseau.

F. DÉVELOPPEMENT DE L'INTERFACE UTILISATEUR

En ce qui concerne l'interface utilisateur (UI) nous avons choisi de créer une webapp. Une webapp est une application que l'on manipule grâce à un navigateur web. De la même manière que les sites web, une application web est généralement placée sur un serveur et se manipule en actionnant des *widgets* à l'aide d'un navigateur web, via un réseau informatique.

Il existe WebioPi qui est un Framework REST qui permet de contrôler les GPIO du RasPi à partir d'un navigateur. WebIOPi intègre un serveur HTML. Il est possible de personnaliser et créer facilement une application web. WebioPi inclut aussi d'autres fonctionnalités comme le PWM logiciel pour tous les GPIO.

Son installation ce fait en 4 lignes :

```
wget http://webiopi.googlecode.com/files/WebIOPi-0.7.0.tar.gz
tar xvzf WebIOPi-0.7.0.tar.gz
cd WebIOPi-0.7.0
sudo ./setup.sh
```

Voici l'interface basique qui est proposée au travers d'un navigateur. Il est même possible d'y accéder via un smartphone en se connectant au point d'accès WIFI que nous avons créés au pare avant:

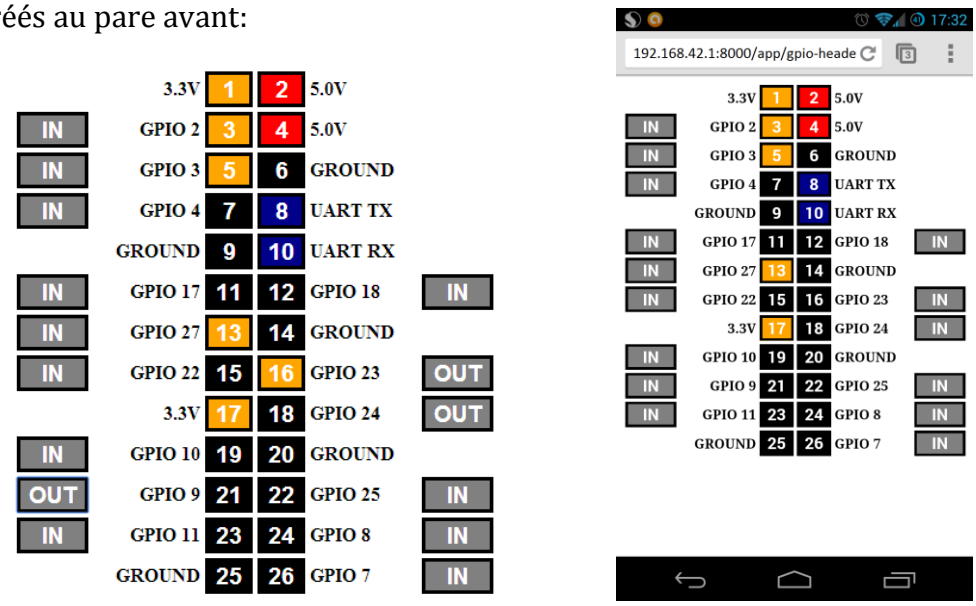


FIGURE 39 – WEBIOPI

Cette interface permet de 'jouer' avec les GPIO. Il est possible de les paramétrer en sortie et de changer leurs valeurs ou en entrer et voir dans quel état ils sont, et tout cela, de manière graphique.

Il est possible de modifier cette interface à notre gré pour faire ce que l'on veut.

Nous avons choisi de réduire l'Interface Utilisateur (IU) au plus simple afin de ne pas complexifier l'utilisation. Avec notre WebApp il est donc possible de programmer l'heure à laquelle l'on veut que la machine à café ce lance. Il est aussi possible de programmer l'heure d'arrêt afin de ne pas faire un bol complet, mais aussi si l'on veut garder le café au chaud un certain temps. Une fois la programmation effectuée, il suffit d'appuyer sur « Envoyer » afin que la CafetièrePi la reçoive. Il est aussi possible de lancer la CafetièrePi directement en appuyant sur « Allumer ».

Il est possible de voir si la machine est en marche directement sur la webapp. Le bouton « Allumer » change de couleur en fonction de l'état de la CafetièrePi. Noir si elle est éteinte, vert si elle est allumée.

Nous avons aussi choisis de conserver les commandes présentes physiquement sur la cafetière. Il est donc toujours possible d'allumer et de programmer la cafetière via les boutons présents d'origine sur la cafetière. La seule modification que nous avons apportée à l'interface d'origine est la LED bicolore qui indique d'une couleur verte si la machine à café est contrôlée à distance, via la webapp.

Le contrôle de la cafetière par les boutons physiques et par contrôle à distance est indépendant. Si la cafetière a été allumée par l'application, il faudra l'éteindre par celle-ci et de même si elle a été directement allumée par les boutons physiques. La programmation via les boutons est prioritaire à celle via la webapp.

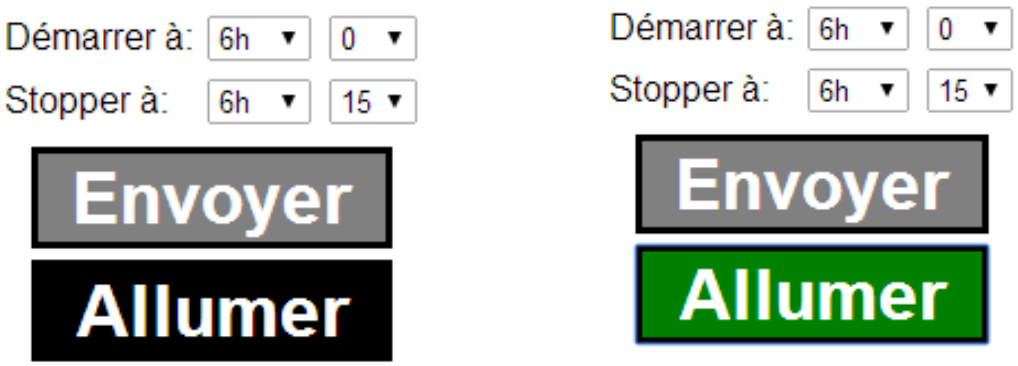


FIGURE 40 – INTERFACE UTILISATEUR

Les codes de cette WebApp ce trouve en annexe. Il y a une partie en langage HTML qui gère la mise en page et l'interaction de la WebApp et une partie en langage PYTHON qui gère l'interaction de la WebApp et le RasPi.

X. MISE EN COMMUN

A. INSTALLATION DE L'APPLICATION

Maintenant que nous avons chacun fini notre partie, il faut les mètres en relations afin de voir s'il n'y a pas de problème.

Premièrement, il faut installer l'application qui à été réalisée sur un smartphone Android. Par mesure de sécurité, l'installation d'applications ne provenant pas du PlayStore est bloquée. Il faut donc pour lever cette interdiction, aller dans les **Paramètres** du téléphone puis aller dans la catégorie **Sécurité** et cocher la case **Sources inconnues**. Il ne faudra pas oublier de décocher cette case une fois l'application CafetièrePi installée.

L'application est à disposition sur le serveur du RasPi. Il faut donc s'y **connecter via wifi** comme expliqué précédemment. Ensuite, il faut **ouvrir le navigateur internet** que vous avez l'habitude d'utiliser et de **taper dans la barre de recherche** l'adresse IP du RasPi qui est **192.168.42.1**.

Il suffit ensuite de **cliquer sur CafetierePi.apk**, pour télécharger le fichier d'installation. Une fois le téléchargement terminé, faite **dérouler la barre de notification** qui ce trouve en haut de l'écran puis **cliquez sur « CafetièrePi.apk Téléchargement terminé »** puis sur **Installer**. Quand l'installation est terminée, il vous suffit de cliquer sur **Ouvrir** pour que l'application ce lance.

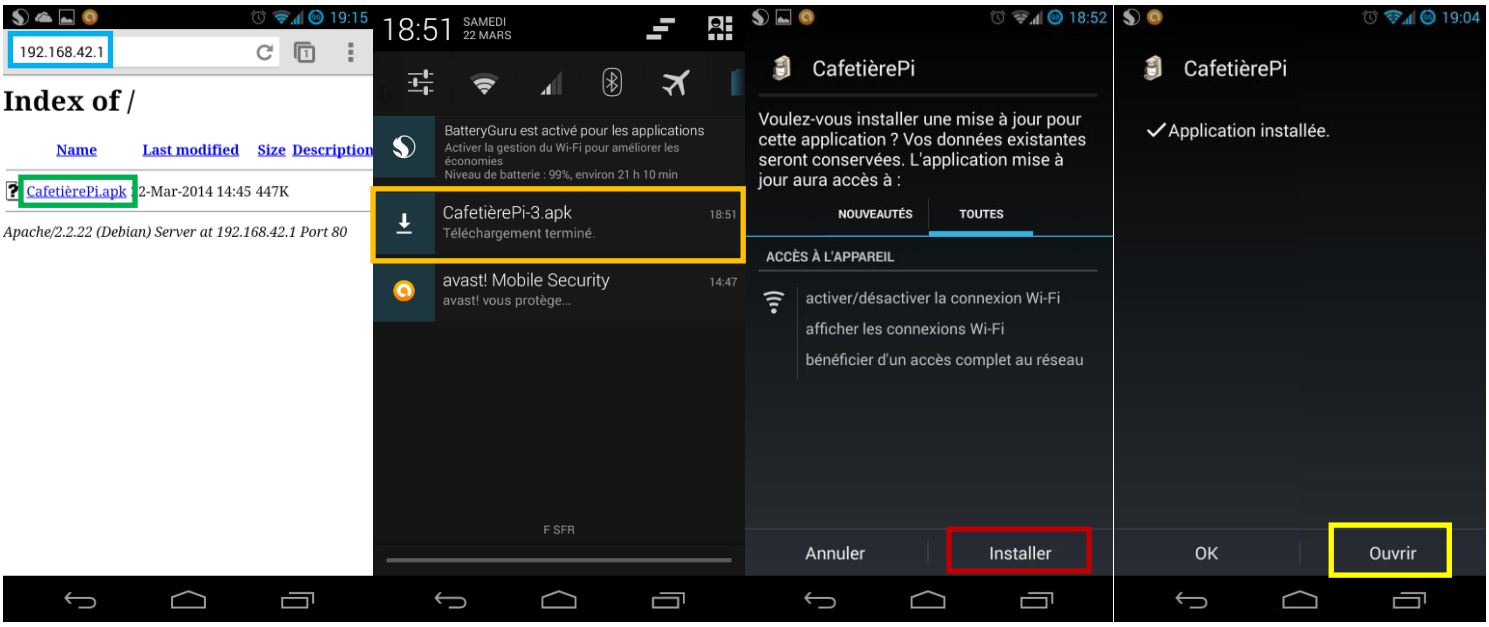


FIGURE 41 – INSTALLATION APPLICATION

B. UTILISATION DE L'APPLICATION

Mainteneant que l'application est installée, nous allons voir comment elle fonctionne. Son utilisation est simple, une fois l'application lancée, vous arrivez sur la page d'accueil. Cliquez sur **« S'identifier »**. Vous arrivez sur la page d'identification. Les identifiants de base sont, pour le login **user1** et le mot de passe **cafepi**. Une fois avoir entré ces identifiants, cliquez sur **« Se connecter ».** Vous arrivez sur une page permettant d'accéder à la WebApp ou de retourner en arrière pour changer ses identifiants. Enfin, après avoir cliqué sur le bouton **« CafetièrePi »**, vous arrivez sur la WebApp et il vous est possible d'allumer et/ou de programmer la CafetièrePi.

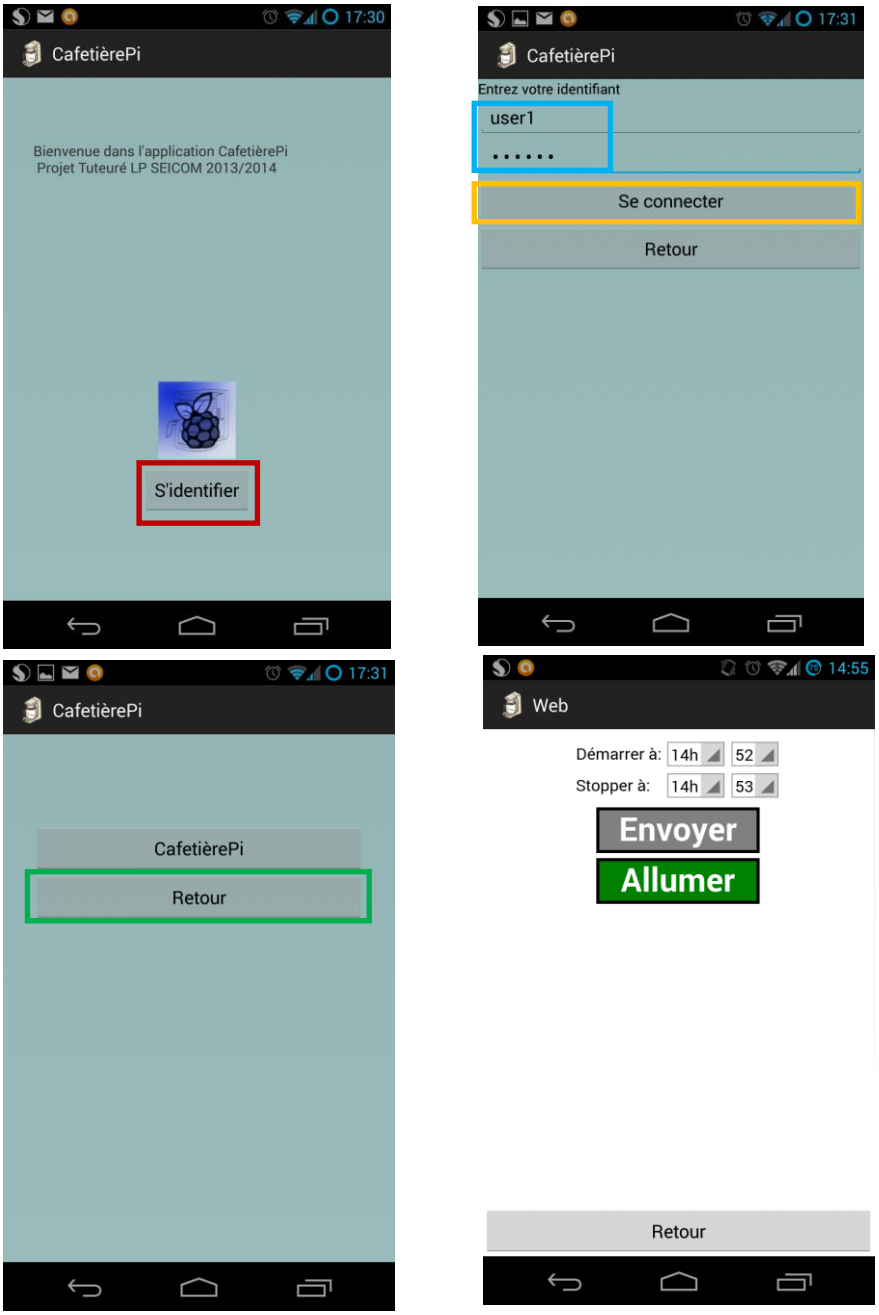


FIGURE 42 – UTILISATION APPLICATION

XI. CONCLUSION

Pour conclure, lorsque l'on nous a parlé que l'on pouvait choisir un projet tuteuré mon binôme et moi-même avons discuté sur un sujet qui pouvait être original. Suite à quelques recherches sur le net nous sommes tombés d'accord sur cette phrase que l'on entend souvent de nos jours que : « Nos smartphones peuvent tout faire sauf le café ».

C'est pourquoi nous nous sommes attardés à rechercher un maximum d'information pour un projet de ce type et avons élaboré et proposé un descriptif de notre projet : une Cafetière Intelligente.

Après validation du projet par le professeur nous avons donc élaboré un cahier des charges pour fixer nos objectifs à atteindre dans le temps impartis.

En sommes, pour la répartition des tâches nous avons séparé le projet en deux parties: la partie concernant la cafetière et le RaspberryPi et la partie application Android.

Pour ces deux parties nous avons donc découpé notre projet en trois grandes étapes : la recherche, le développement et la mise en commun.

Concernant l'aspect éducatif le contenu du projet celui-ci nous a permis de voir de nouvelle chose et d'apprendre de nouveaux langages de programmation notamment pour le développement de l'interface web et également pour l'application Android et également sur la création d'un planning de projet, d'utiliser les outils découverts en cours (GitHub, processus de projet, programmation).

Concernant l'aspect humain ce projet nous a permis d'apprendre à travailler en équipe (binôme) et de réaliser ce projet en total autonomie, et gérer correctement notre temps afin d'arriver au terme de ce projet. Cela nous a également permis de d'échanger nos savoir ou connaissance sachant chacun arrivent d'horizon différent.

XII. ANNEXE

index.php 1/3

```
<!-- Règles de mise en forme à appliquer par le navigateur -->
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <!-- Elément racine de la pag -->
    <head> <!-- Informations d'entête utilisées -->
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <meta name="viewport" content="width=device-width"> <!--Compatibilité pour mobile -->
        <title>Cafeti&egrave;rePi</title> <!-- Titre de l'onglet -->
        <script type="text/javascript" src="/webiopi.js"></script>
        <script type="text/javascript"> <!-- Début du code javascript-->
            webiopi().ready(function() {
                // Cette fonction traite les infos reçus de la macro set/getCafetierePiHours
                var updateCafetierePiHours = function(macro, args, response) {
                    var hours = response.split(";");
                    // Ces lignes utilises des fonctions jQuery
                    $("#inputOn").val(hours[0]);
                    $("#inputOff").val(hours[1]);
                }

                // Cette fonction traite les infos reçus de la macro set/getCafetierePiMinutes
                var updateCafetierePiMinutes = function(macro, args, response) {
                    var minutes = response.split(";");
                    $("#inputOnMin").val(minutes[0]);
                    $("#inputOffMin").val(minutes[1]);
                }

                // Immediately call getCafetierePiHours macro to update the UI with current values
                // "getCafetierePiHours" refers to macro name
                //[] is an empty array, because getCafetierePiHours macro does not take any argument
                // updateCafetierePiHours is the callback function, defined above
                webiopi().callMacro("getCafetierePiHours", [], updateCafetierePiHours);
                webiopi().callMacro("getCafetierePiMinutes", [], updateCafetierePiMinutes);

            // Création d'un bouton appellant les macro setCafetierePiHours et setCafetierePiMinutes
                var sendButton = webiopi().createButton("sendButton", "Envoyer", function() {
                    // Arguments envoyés aux macro
                    var hours = [$("#inputOn").val(), $("#inputOff").val()];
                    var minutes = [$("#inputOnMin").val(), $("#inputOffMin").val()];
                    // Appelle des macros
                    webiopi().callMacro("setCafetierePiHours", hours, updateCafetierePiHours);
                    webiopi().callMacro("setCafetierePiMinutes", minutes,
                    updateCafetierePiMinutes);
                });

                // Append the button to the controls box using a jQuery function
                $("#controls").append(sendButton);

                // Création d'un bouton nommé "Allumer" pour contrôler le GPIO 24
                var button = webiopi().createGPIOButton(24, "Allumer");

                // Append the button to the controls box
                $("#controls").append(button);

                // Rafraichissement du bouton
                // true pour rafraichir répétitivement ou false pour le rafraichir qu'une fois
                webiopi().refreshGPIO(true);
```

index.php 2/3

```
// Les fonctions suivantes permettent de créer les choix des listes déroulantes
        (function() {
            var elm = document.getElementById('inputOn'),
            df = document.createDocumentFragment();
            for (var i = 0; i <= 23; i++) {
                var option = document.createElement('option');
                option.value = i;
                option.appendChild(document.createTextNode(i+"h"));
                df.appendChild(option);
            }
            elm.appendChild(df);
        });

        (function() {
            var elm = document.getElementById('inputOff'),
            df = document.createDocumentFragment();
            for (var i = 0; i <= 23; i++) {
                var option = document.createElement('option');
                option.value = i;
                option.appendChild(document.createTextNode(i+"h"));
                df.appendChild(option);
            }
            elm.appendChild(df);
        });

        (function() {
            var elm = document.getElementById('inputOnMin'),
            df = document.createDocumentFragment();
            for (var i = 0; i <= 59; i++) {
                var option = document.createElement('option');
                option.value = i;
                option.appendChild(document.createTextNode(i));
                df.appendChild(option);
            }
            elm.appendChild(df);
        });

        (function() {
            var elm = document.getElementById('inputOffMin'),
            df = document.createDocumentFragment();
            for (var i = 0; i <= 59; i++) {
                var option = document.createElement('option');
                option.value = i;
                option.appendChild(document.createTextNode(i));
                df.appendChild(option);
            }
            elm.appendChild(df);
        });
});

</script> <!-- Fin du code javascript -->
```

index.php 3/3

```
    <body> <!-- Début du corps de la page -->
       <div align="center"> <!-- Permet de centrer le contenu de la page -->
           <TABLE > <!-- Début du tableau -->
               <TR> <!-- Début de la 1ere ligne du tableau -->
                   <TD> <!-- Début de la 1ere colonne du tableau -->
                           <FONT face="Helvetica"> D&eacute;marrer &agrave;: </FONT> <!-- Insertion
                                                                                    d'un texte -->
                   </TD> <!-- Fin de la 1ere colone du tableau -->
                   <TD>
                       <select id="inputOn"></select> <!-- Insertion d'une liste déroulante -->
                   </TD>
                   <TD>
                       <select id="inputOnMin"></select>
                   </TD>
               </TR> <!-- Fin de la 1ere ligne du tableau -->
               <TR>
                   <TD>
                       <FONT face="Helvetica">Stopper &agrave;: </FONT>
                   </TD>
                   <TD>
                       <select id="inputOff"></select>
                   </TD>
                   <TD>
                       <select id="inputOffMin"></select>
                   </TD>
               </TR>
           </TABLE> <!-- Fin du tableau-->
           <div id="controls"></div> <!-- Ajout des boutons -->
       </div>
    </body> <!-- Fin du corps de la page -->
</html> <!-- Fin du code HTML -->



<!-- Les éléments de la programmation de l'heure son placés dans un tableau 2x3
                ----------------------------------------
                |  Texte  |  inputOn  |   inputOnMin   |
                ----------------------------------------
                |  Texte  |  inputOff |  inputeOffMin  |
                ----------------------------------------          -->
```

script.py 1/1

```
import webiopi  #import de la librairie webiopi
import datetime #import de l'heure

GPIO = webiopi.GPIO

CAFETIEREPI = 24 # broche GPIO utilisant la numérotation BCM

HOUR_ON    = 0 # Heure d'allumage de la CafetièrePi
HOUR_OFF   = 0 # Heure d'extinction de la CafetièrePi
MINUTE_ON  = 0 # Minute d'allumage de la CafetièrePi
MINUTE_OFF = 0 # Minute d'extinction de la CafetièrePi

# Fonction de paramétrage automatiquement appelée au démarrage de WebIOPi
def setup():
    # Configuration du GPIO utilisé par la CafetièrePi en sortie
    GPIO.setFunction(CAFETIEREPI, GPIO.OUT)

# La fonction loop est appelée répétitivement par WebIOPi
def loop():
    # Récupération de l'heure courante
    now = datetime.datetime.now()

    # Allumer la CafetièrePi tous les jours à l'heure correcte
    if ((now.hour == HOUR_ON) and (now.minute == MINUTE_ON ) and (now.second == 0)):
        if (GPIO.digitalRead(CAFETIEREPI) == GPIO.LOW):
            GPIO.digitalWrite(CAFETIEREPI, GPIO.HIGH)

    # Eteindre la CafetièrePi tous les jours à l'heure correcte
    if ((now.hour == HOUR_OFF) and (now.minute == MINUTE_OFF) and (now.second == 0)):
        if (GPIO.digitalRead(CAFETIEREPI) == GPIO.HIGH):
            GPIO.digitalWrite(CAFETIEREPI, GPIO.LOW)

    # Donne au CPU du temps avant de boucler à nouveau
    webiopi.sleep(1)

# La fonction destroy est appelée lors de l'arrêt de WebIOPi
def destroy():
    GPIO.digitalWrite(CAFETIEREPI, GPIO.LOW)

# Fonction macro permettant d'extraire les heures paramétrées et de les placer dans des variables
@webiopi.macro
def getCafetierePiHours():
    return "%d;%d" % (HOUR_ON, HOUR_OFF)

# Même chose pour les minutes
@webiopi.macro
def getCafetierePiMinutes():
    return "%d;%d" % (MINUTE_ON, MINUTE_OFF)

@webiopi.macro
def setCafetierePiHours(on, off):
    global HOUR_ON, HOUR_OFF
    HOUR_ON = int(on)
    HOUR_OFF = int(off)
    return getCafetierePiHours()

@webiopi.macro
def setCafetierePiMinutes(on, off):
    global MINUTE_ON, MINUTE_OFF
    MINUTE_ON = int(on)
    MINUTE_OFF = int(off)
    return getCafetierePiMinutes()
```

Annexe application android source xml:

Source Layout d'intro

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/background"
    android:padding="30dip"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="33dp"
        android:text="TextView" />

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="90dp"
        android:layout_height="90dp"
        android:layout_above="@+id/button1"
        android:layout_centerHorizontal="true"
        android:maxHeight="@dimen/logo"
        android:maxWidth="@dimen/logo1"
        android:src="@drawable/ic_1" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="56dp"
        android:text="S&apos;identifier" />

</RelativeLayout>
```

Source Layout d'identification

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/couleurFond"
    android:orientation="vertical"
    tools:context=".Intro" >

    <TextView
        android:id="@+id/textViewIdentifiant"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/identifiant"
        android:textColor="@color/couleurTitre" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <EditText
            android:id="@+id/EditTextPrenom"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom"
            android:layout_weight="1"
            android:hint="@string/identifiantHint" />
    </LinearLayout>

    <EditText
        android:id="@+id/editTextPasswd"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:password="true"
        android:inputType="textPassword" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/buttonEnvoyer"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/bouton" />
```

```
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Retour" />

</LinearLayout>
```

Source Layout Web1

```
<?xml version="1.0" encoding="utf-8"?>
        <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:background="@color/background"
            android:padding="30dip" >

            <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text=""
            android:layout_gravity="center"
            android:layout_marginBottom="25dip"
            android:textSize="24.5sp" />

            <Button
            android:id="@+id/button1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/button1_label"
            android:textSize="18sp" />

            <Button
            android:id="@+id/button2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/button2_label"
            android:textSize="18sp" />

        </LinearLayout>
```

Source Layout Web2

```
<?xml version="1.0" encoding="utf-8"?>
        <LinearLayout
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="vertical"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">

            <WebView
                android:id="@+id/webView1"
                android:layout_width="320dp"
                android:layout_height="427dp" />

            <Button
                android:id="@+id/button1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Retour" />

        </LinearLayout>
```

Annexe application android source java:

Source Layout d'intro:

```
/* PROJET TUTEURE CAFEPI
 * SZAMVEBER MATTHIAS
 * FREIN MATHIAS
 * LAYOUT D'INTRO DE L'APPLICATION*/

package com.example.cafepi;
// importation des packages
import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Intent;
//import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;



public class MainActivity extends Activity {

         @SuppressLint("CutPasteId")
```

```
        @Override
          public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            //Affichage de l'image du logo
            ImageView image = (ImageView)findViewById(R.id.imageView1);
       image.setImageResource(R.drawable.ic_1);

             //Affichage du texte de presentation
            TextView monTexte = (TextView)findViewById(R.id.textView1);
            monTexte.setText("Bienvenu dans l'application CafetierePi"+" Projet Tuteuré
"+"LP SEICOM 2013/2014");

         //Click sur le bouton genere une nouvelle activité vers le second layout
         Button Bouton1 = (Button)findViewById(R.id.button1);
         Bouton1.setOnClickListener(new OnClickListener() {
          @Override
                  public void onClick(View v) {
                          Intent t = new Intent(MainActivity.this, Intro.class);
//déclaration de notre nouvelle activité
                          startActivity(t);
                  }});

         };}
```

Source Layout identification

```
/* PROJET TUTEURE CAFEPI
 * SZAMVEBER MATTHIAS
 * FREIN MATHIAS
 * LAYOUT DE CONNECTION*/
package com.example.cafepi;


import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Intro extends Activity {

         public EditText editText;
         private Button button;
```

```
        public String identifiant;
        //private EditText identifiant;
        private EditText password;

@Override
         public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.intro);

                editText = (EditText) findViewById(R.id.EditTextPrenom);
                password = (EditText) findViewById(R.id.editTextPasswd);
               // identifiant = (EditText) findViewById(R.id.EditTextPrenom);
                button = (Button) findViewById(R.id.buttonEnvoyer);

                button.setOnClickListener(new OnClickListener() {
                          public void onClick(View v) {
                                   identifiant = editText.getText().toString();


        if(password.getText().toString().equals("cafepi") &&
(identifiant.toString().equals("user1"))){
                                                          //Toast.makeText(Intro.this,
"Bonjour " + identifiant + " !", Toast.LENGTH_LONG).show();
                                                 Intent t = new Intent(Intro.this,
WebView.class);
                                                 startActivity(t);
                                                 }else{   Toast.makeText(Intro.this, "Erreur
de connexion !", Toast.LENGTH_LONG).show();}

                          }});

                //retour au premier layout
                Button Bouton1 = (Button)findViewById(R.id.button1);
                 Bouton1.setOnClickListener(new OnClickListener() {
                 public void onClick(View v) {
                                   finish();

                          };
            });
        }}
```

Source Layout web1

```
/* PROJET TUTEURE CAFEPI
 * SZAMVEBER MATTHIAS
 * FREIN MATHIAS
 * LAYOUT WEB1*/
package com.example.cafepi;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebSettings;
import android.widget.Button;


//import android.webkit.WebView;

         public class WebView extends Activity implements OnClickListener{
                   WebView webview;

             @Override
             public void onCreate(Bundle savedInstanceState) {
                 super.onCreate(savedInstanceState);
                 setContentView(R.layout.webview);

              // déclaration des element present sur le layout
                  View firstButton = findViewById(R.id.button1);
                  firstButton.setOnClickListener(this);
                  View secondButton = findViewById(R.id.button2);
                  secondButton.setOnClickListener(this);
             }
                  @Override
                  //declaration de la method onClick
                  public void onClick(View v) {
                           // TODO Auto-generated method stub
                           switch(v.getId()){
            case R.id.button1: // initialisation de notre activité
                Intent j = new Intent(WebView.this, WebScreen.class);
                startActivity(j);
            break;

            case R.id.button2:
                                     finish();
            break;
        }
    }
}
```

Source Layout web2

```
/* PROJET TUTEURE CAFEPI
 * SZAMVEBER MATTHIAS
 * FREIN MATHIAS
 * LAYOUT D'AFFICHAGE DU CONTENU WEB*/
package com.example.cafepi;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.widget.Button;
import android.util.Log;

public class WebScreen extends Activity {

    public static final String URL = "";
    private static final String TAG = "Class Webscreen";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.webscreen);

        WebView Mywebview = (WebView) findViewById (R.id.webView1);
        WebSettings webSettings = Mywebview.getSettings();
        Mywebview.loadUrl("http://192.168.42.1:8000");// adresse Ip
        webSettings.setJavaScriptEnabled(true);

    Button Bouton1 = (Button)findViewById(R.id.button1);
         Bouton1.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
                  finish();
                  }});

}}
```

XIII. CHECK-LIST A VERIFIER AVANT DE RENDRE UN RAPPORT.

	Auteurs	Correcteur
Le titre du document reflète bien le contenu du rapport.		
Les noms des auteurs figurent sur la première page.		
La date du rapport est indiquée sur la première page.		
Le résumé synthétise bien le contenu du rapport.		
Le plan du rapport avec les numéros de pages est inclus.		
Le contexte du travail est présenté dans l'introduction.		
Les objectifs du travail sont indiqués dans l'introduction.		
Le plan du rapport est expliqué dans l'introduction.		
La conclusion synthétise les conclusions du travail présenté.		
Si nécessaire, des perspectives sont proposées.		
L'orthographe a été vérifiée.		
Le rapport n'est pas rédigé à la première personne		
Les phrases ne sont pas trop longues.		
Les sections sont correctement numérotées.		
Le format des différents paragraphes est uniforme.		
Toutes les figures sont référencées dans le texte.		
Les figures sont numérotées et correctement légendées.		
Les axes des courbes sont explicités.		
Les figures ne sont pas trop petites.		
La taille et le format des équations sont uniformisés.		
Les notations des équations sont explicitées.		
Les calculs longs sont placés en annexe.		
Les références bibliographiques sont citées dans le texte.		
Les références bibliographiques permettent de retrouver le document cité.		
Le droit d'auteur est respecté.		