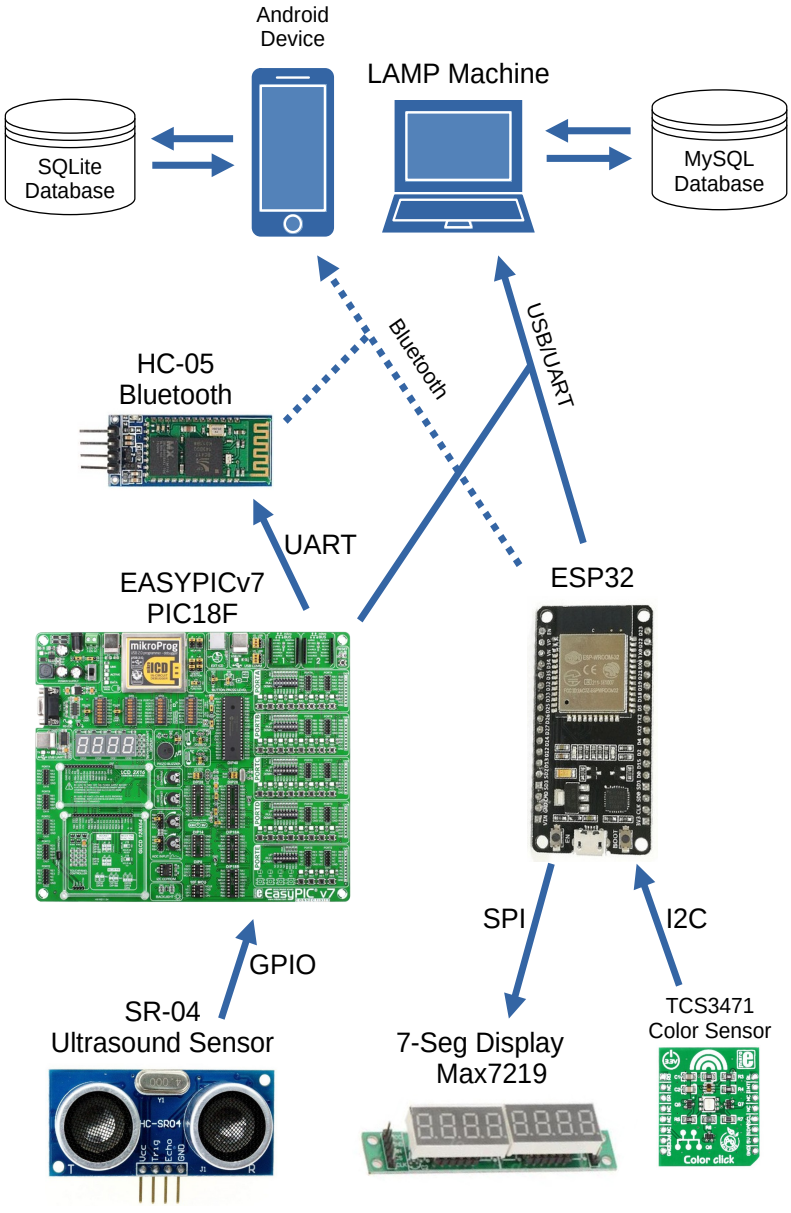
Nom : Andrew O’Shei
Num Etud. 20001943

**Projet Objet Connecté / Bases des données**

Ce qui suit est la documentation de mon projet de démonstration d'un pipeline complet pour la création des objets connectés. Il démontrera l'application pratique de plusieurs approches et outils, notamment les microcontrôleurs, les réseaux sans fil, les applications Web et les bases de données. Bien que ce projet puisse avoir une portée limitée en raison des matériaux et du temps disponibles, il n'y a aucune raison pour que les méthodes utilisées ne puissent pas être transformées en une application IoT complète.

Les plates-formes utilisées pour construire mon projet incluent l’Espressif ESP32, MicroChip Pic18F, Android Studio, MySQL et SQLite et LAMP web service stack. Tout au long de cet aperçu, je discuterai des outils que j'ai utilisés, des raisons pour lesquelles je les ai choisis et de leurs avantages et inconvénients par rapport aux options alternatives. Tout comme l'approche que j'ai adoptée pour concevoir et construire le système, nous commencerons par les microcontrôleurs et les différents composants et capteurs utilisés pour collecter les données.



**EasyPICv7, PIC18F et le HC-SR04 capteur ultrason**

L'EasyPICv7 est une carte de développement centrée sur le microcontrôleur 8 bits PIC18F45k22. Il comporte une variété de périphériques intégrés qui sont utiles pour évaluer la fonctionnalité du microcontrôleur et des projets de prototypage comme celui-ci. En raison du temps limité et de l'accès au matériel, je n'ai pas complètement développé le projet en utilisant la plate-forme EasyPICv7. J'ai cependant développé un programme qui prouve que le design pouvait être facilement patché dans le pipeline que j'ai créé pour l'ESP32. En ce qui concerne l'EasyPICv7, plusieurs composants embarqués ont été utilisés et, comme indiqué dans le schéma ci-dessus, le capteur ultrasons HC-SR04 et le module Bluetooth HC-05 ont été ajoutés pour prouver une gamme de fonctionnalités similaire à celle de l'ESP32.

L'EasyPICv7 génère des données à l'aide du capteur ultrason HC-SR04. Le capteur ultrason HC-SR04 est un module relativement simple pour ajouter aux projets la fonctionnalité pour mesurer la distance et détecter les collisions. Le principe de base est qu'il déclenche une impulsion ultrasonore de huit cycles (10us) puis il détecte la réflexion de cette impulsion lorsqu'elle revient au transducteur situé sur le module. Pour mesurer la distance, nous pouvons mesurer le temps de retour de l'impulsion et calculer à l'aide de la formule suivante:

Distance(mètres) = ½ * Time(seconds) * 343m/s

343 est la distance approximative en mètres parcourue par le son vers l'air par seconde. Le temps est divisé par deux car nous devons tenir compte du temps que prend le son pour arriver à destination et puis revenir. Cependant, dans notre programme, le calcul est encore simplifié car nous comptons le temps de retour par incréments de 10 nanosecondes. En utilisant la formule que j'ai énoncée, nous pouvons déterminer qu'avec un temps de retour de 600 comptes de 10 nanosecondes, la distance serait d'environ 1 mètre. {½ * .006 * 343 = 1.029 mètre}. Dans le programme, nous pouvons simplifier quelque peu les mathématiques avec la division. Nous savons {6000(us) / X = 1.029 mètres}. On trouve {X = 6000(us) / 1.03}, donc X = 5825. Puisque nous sommes intéressés à obtenir la distance en centimètres, simplifions les choses et utilisons:

Temps(us) / 58.0 = Distance(cm)

Pour mesurer la distance avec le HC-SR04 le microcontrôleur envoie un signal logique haut(1) vers la broche TRIG pendant 10us afin de produire une impulsion. Puis le microcontrôleur commence à compter en attendant que la broche HC-SR04 ECHO lise la tension élevée (logique 1). En détectant que la broche ECHO a une tension élevée, il termine le comptage et calcule la distance. Sur le PIC18F, nous pouvons simplement utiliser n'importe quelle broche GPIO pour cette opération car nous ne traitons pas de protocoles de communication spécifiques. Pour optimiser d'avantage les performances du code, j'ai plafonné le nombre maximum à 6000us car j'ai trouvé que le HC-SR04 n'était pas fiable pour mesurer des distances de plus de 1 mètre. De plus, il y a place à l'amélioration, car le code actuel se bloque pendant la phase de comptage. Comme nous le verrons dans un instant, cela réduit le taux de rafraîchissement de l'affichage à 7 segments utilisé pour lire la valeur de distance. Une meilleure approche serait d'utiliser une interruption pour détecter l'écho ultrasonore afin que d'autres opérations puissent se poursuivre en attendant une mesure de distance.

Après avoir pris avec succès une mesure de distance, le PIC18F écrit les données dans l'EEPROM embarquée de l'EasyPIC. L'EasyPICv7 contient une EEPROM ST Microelectronics M24C08, bien que dans le cadre de ce projet nous l'utilisions principalement pour tester la fonctionnalité, il convient de noter qu'elle s'interface avec le PIC18F via le protocole I2C. Cela démontre le potentiel de mise en cache des lectures de données dans le cas où le PIC perd la connectivité avec le périphérique Android ou le serveur Linux.

Ensuite, le PIC lit les données de l'EEPROM et les affiche sur l'affichage à 7 segments intégré. L'affichage à 7 segments nécessite 4 broches GPIO (A0-A3) pour basculer chacun des 4 chiffres de

l'affichage. Lorsque le PIC fait basculer une broche vers le haut, un transistor ouvre un chemin vers la masse, complétant le circuit pour le chiffre en cours de basculement. Pour contrôler la valeur affichée, les chiffres de l'affichage à 7 segments partagent un bus sur le PortD du PIC. Chacune des 8 broches de PortD est utilisée pour activer et désactiver un segment spécifique sur l'affichage. Plusieurs valeurs différentes sont affichées simultanément en activant et désactivant chaque caractère à une vitesse imperceptible à l'œil humain. Malheureusement, comme je l'ai mentionné ci-dessus dans ce projet particulier, il y a un léger scintillement sur l'écran car le temps nécessaire pour lire le capteur à ultrasons plus le cycle de lecture / écriture EEPROM fait chuter le taux de rafraîchissement de l'écran dans une plage perceptible.

La dernière étape utilisée dans le PIC est l'interface série UART. Il l'utilise à la fois pour transmettre des données au serveur Linux et au module Bluetooth HC-05. Heureusement, l'EasyPICv7 dispose d'une interface UART vers USB intégrée qui transmet automatiquement tout ce qui se trouve sur le bus UART via un port USB. La connexion au serveur Linux est donc aussi simple que de brancher un câble USB. Le HC-05 est d'une facilité d'utilisation similaire. Branchez les broches d'alimentation et le Tx au Rx du PIC et au Rx sur le Tx du PIC et le HC-05 transmettra les données via Bluetooth sans autre configuration. Il est important de noter que le HC-05 est réglé par défaut sur 9600 bauds et que cela doit être configuré par le PIC. Cependant, il est possible d'augmenter le débit de données (et configurer des autres paramètres) en envoyant des messages AT au HC-05 bien que cela n'entre pas dans le cadre de ce projet.

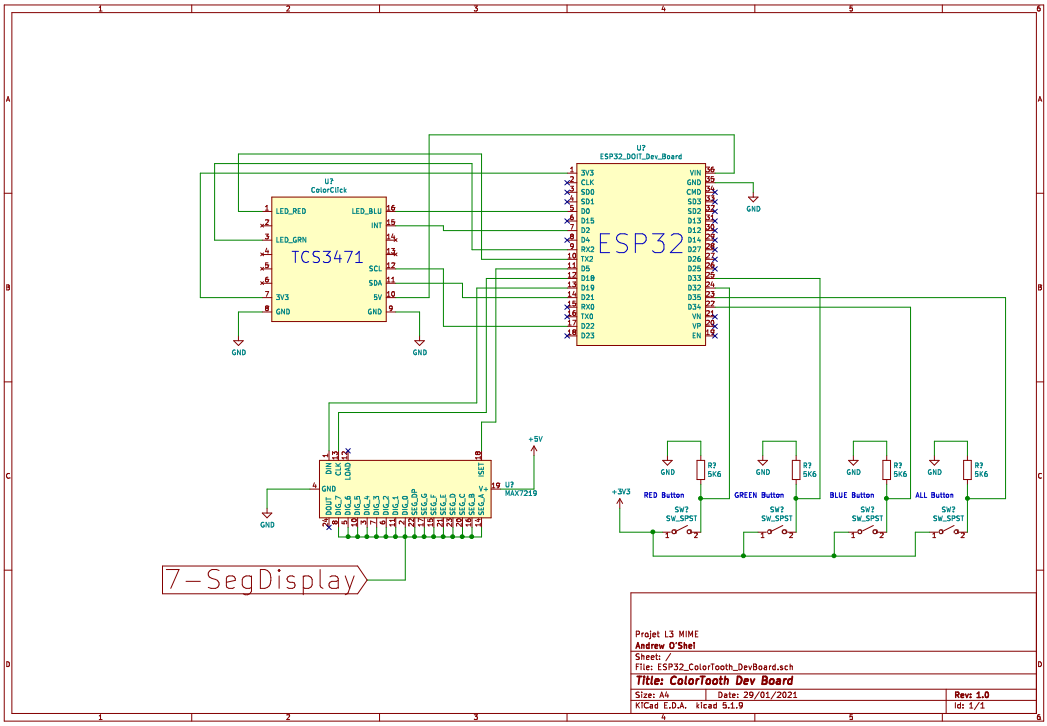
C'est là que la partie PIC du projet s'est terminée pour moi. J'ai testé que le serveur Linux reçoit correctement les données transmises par le PIC via USB avec un simple script Python. De même, j'ai testé et reçu correctement les données transmises à l'aide de l'application Serial Bluetooth Terminal disponible sur le Google Play Store (https://play.google.com/store/apps/details?hl=en&id=de.kai_morich.serial_bluetooth_terminal). Malheureusement, en raison de contraintes de temps, je n'ai pas pu patcher la fonctionnalité pour analyser, recevoir et afficher ces données dans mes applications Android et Web. Si je devais terminer l'intégration du PIC avec mes applications Android et Web, j'utiliserais les mêmes méthodes que celles que nous examinerons avec l'ESP32.

**ESP32 et le TCS3471 capteur de couleur**

J'ai commencé ce projet au premier semestre avec l'idée d'utiliser le capteur de couleur Mikroelectronica Color Click. Cependant, en testant mon code original sur l'EasyPICv7 en janvier, j'ai rapidement réalisé que j'allais être sévèrement limité par l'absence de licence du software pour le MikroC pour PIC IDE, ce qui limite la taille du code compilé pour le PIC à moins de 4kb. Avec toutes les fonctionnalités que j'espérais mettre en œuvre, cette restriction était tout simplement trop limitative. Je suis donc rapidement allé travailler à la construction de ma propre carte de développement personnalisée pour tester les fonctionnalités du Color Click avec une plate-forme de microcontrôleur offrant plus de flexibilité. Avec Bluetooth intégré, un environnement de développement ouvert et après avoir travaillé avec lui sur mon projet Android pour le premier semestre, l'ESP32 était un choix évident.

L'ESP32 présente plusieurs avantages par rapport au PIC18F étant donné qu'il s'agit d'une plate-forme de développement moderne conçue spécifiquement pour les applications IoT. L'ESP32 est à la base un système sur puce doté d'un microprocesseur double cœur 32 bits avec Bluetooth et Wifi intégrés. Il est destiné au développement à faible consommation d'énergie et à faible coût. Cependant, des fonctionnalités supplémentaires s'accompagnent d'une complexité accrue et d'un ensemble différent de considérations de conception. Cela étant dit, mon idée était de créer une carte aussi proche que possible des fonctions et fonctionnalités que nous prévoyions d'utiliser avec l'EasyPICv7. Heureusement, j'avais sous la main un écran à 7 segments d'un projet précédent qui ne

s'est jamais concrétisé. À des fins de test, je voulais également ajouter des boutons pour contrôler la LED RVB intégrée sur la carte Color Click.




Au centre de la carte Color Click se trouve un capteur de couleur AMS TCS3471. J'ai décrit sa fonctionnalité en profondeur dans le rapport que j'ai rédigé en décembre, je serai donc plus bref ici. Le principe de base du TCS3471 est qu'il dispose de quatre diodes de détection de lumière, claires, rouges, vertes et bleues. Chacune des diodes de détection de lumière est connectée à un convertisseur analogique-numérique (ADC) qui est capable de déterminer l'intensité de la lumière frappant la diode. Il y a alors un étage de gain pour booster la lecture des diodes de détection de lumière si l'on a l'intention d'utiliser la puce dans des conditions de faible luminosité. Il dispose également d'une fonction d'interruption pour déclencher des fonctions lorsqu'un certain seuil de lumière est dépassé, mais au final, je n'ai pas fini par implémenter cette fonctionnalité.

Le TCS3471 communique via le protocole I2C qui permet à un microcontrôleur de définir ses différents modes de fonctionnement et paramètres en écrivant dans une série de registres de contrôle. Mais la communication principale a lieu lors de la lecture des 8 registres de couleurs. L'ADC de chaque diode de détection de lumière écrit sur une paire de registres 8 bits, ce qui signifie finalement que nous nous retrouvons avec une valeur de 16 bits indiquant l'intensité de la lumière détectée par chacune des 4 diodes. Bien que j'apprécie la possibilité d'avoir une résolution de 16 bits sur les lectures du capteur, j'ai choisi de réduire les lectures du capteur d'échantillons à des valeurs de 8 bits, car cela rend plusieurs opérations beaucoup plus faciles en aval. Il est également important de noter que le TCS3471 permet à l'utilisateur de régler le temps entre la lecture des données du capteur. Un détail qui est devenu clair après les tests est que le délai entre les lectures a un impact sur la précision des données du capteur. Ceci est configuré avec le registre de temps d'intégration. J'ai finalement trouvé qu'un temps d'intégration d'environ 350 ms est aussi rapide que possible et produit toujours des données fiables.

En développant le programme pour l'ESP32 avec l'IDE Arduino j'ai pu utiliser la bibliothèque TCS3471 créée par Raivis Rengelis disponible sur Github (https://github.com/raivisr/TCS3471-Arduino-Library). Le choix d'utiliser cette bibliothèque était principalement pour gagner du temps et éviter de réécrire du code. Si vous jetez un œil à la bibliothèque Arduino, vous verrez qu'elle est

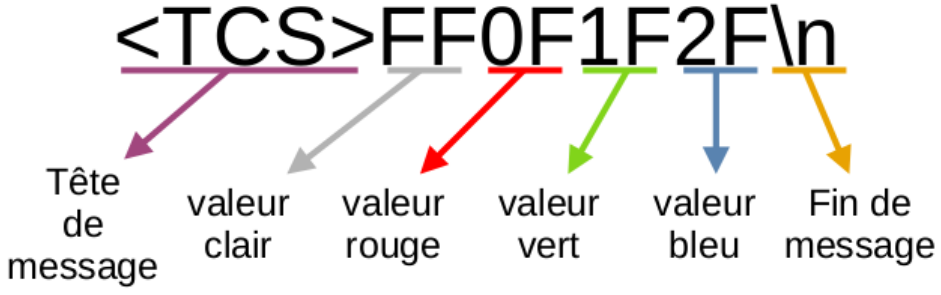
très similaire au programme que j'ai soumis en décembre. La principale différence est qu'il est écrit en C ++ et non en C et tire parti de certains types de données spécifiques à C ++.

Bien que j'aie pu me procurer un affichage à 7 segments pour ma carte de développement, son fonctionnement est quelque peu différent de l'affichage à 7 segments trouvé sur l'EasyPICv7. L'écran à 7 segments que j'ai utilisé est basé sur le pilote d'affichage LED Max7219. Il est donc contrôlé via le protocole SPI (Serial Peripheral Interface). Le Max7219 gère également la commutation entre chaque caractère de l'écran en interne afin d'éviter le problème de scintillement que j'ai rencontré avec l'affichage à 7 segments sur l'EasyPICv7. Le Max7219 offre un mode où les caractères sont affichés en envoyant une simple valeur Int ou Char, mais il est quelque peu limité. Je voulais pouvoir afficher les messages d'état de la connexion ainsi que les données en valeurs hexadécimales, il fallait donc programmer l'affichage en mode manuel. En mode manuel, les caractères sont écrits sur l'écran de la même manière que sur l'EasyPICv7. Chaque caractère de l'affichage comporte sept segments et un point correspondant à une valeur d'entrée de 8 bits. Les bits définis sur 1 ou 0 permettent d'activer ou de désactiver un segment particulier. Les données lues à partir du TCS3471 sont affichées sous forme de quatre valeurs hexadécimales 8 bits (une valeur et donc deux caractères d'affichage pour chacune des diodes de détection de couleur). Les messages de connexion s'affichent également lorsqu'un périphérique USB série ou Bluetooth se connecte ou se déconnecte de l'ESP32.

Alors que le module Bluetooth HC-05 se superpose essentiellement au bus UART, le Bluetooth intégré sur l'ESP32 nécessite quelques considérations lors de la programmation. Lors de l'utilisation de la bibliothèque BluetoothSerial intégrée à ESP32, comme je l'ai fait, l'interface Bluetooth est présentée comme une classe d'objet accessible comme une deuxième interface UART. Une différence clé est qu'au lieu de définir le débit en bauds lors de l'initialisation de l'interface Bluetooth, vous devez définir le nom du périphérique Bluetooth tel qu'il apparaîtra sur un réseau. En dehors de cela, les appels de lecture et d'écriture sur le Bluetooth sont identiques aux appels utilisés pour une interface série UART. Cependant, la répartition des fonctionnalités entre deux interfaces différentes est pratique lors de la mise en œuvre de protocoles de connexion, comme je l'ai fait dans ma programmation. C'est-à-dire que les connexions via Bluetooth et via UART peuvent être gérées dans des fonctions séparées.

Pour qu'un appareil Android ou un serveur Linux se connecte à l'ESP32, j'ai utilisé un handshake. Après le démarrage, l'ESP32 écoute une fois à chaque cycle de programme les messages entrants sur les interfaces série Bluetooth et UART, il reçoit un message «<CONNECT>» puis une connexion est établie et l'ESP32 commencera à transmettre des données sur cette interface. Cela permet à la fois à l'ESP32 de réduire la charge lorsqu'une interface n'est pas utilisée, il permet également aux appareils Android et Linux de synchroniser le transfert de données, l'une des méthodes que j'ai employées pour la correction d'erreurs. De même, si un appareil transmet un message «<DISCONN>», la connexion sur cette interface sera fermée et l'ESP32 cessera de transmettre des données. L'ESP32 détecte également les messages erronés et affiche «ErrOr 0» sur l'affichage à 7 segments, ce qui facilite la communication de débogage.

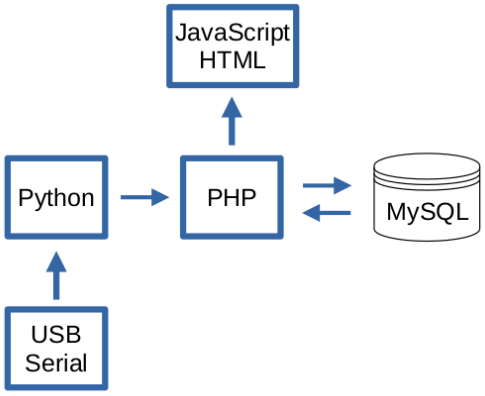
Les messages de données transmis par l'ESP32 sont envoyés au format suivant:



L'en-tête et le saute de ligne de fin du message permettent d'éviter les erreurs qui pourraient être causées par la réception d'un message partiel. Chaque valeur de couleur varie de 0 à 255 (8 bits) représentée en hexadécimal. Afin d'obtenir des caractères hexadécimaux à partir des valeurs initiales reçues du TCS3471, l'ESP32 utilise un tableau de recherche avec chaque groupe de quatre bits (à partir des valeurs de couleur reçues) comme index pour le tableau.

**LAMP Web App**

L'application Web de réception et d'affichage des valeurs générées par la carte ESP32 / ColorClick repose sur une variété de technologies centrées autour de la pile de services LAMP. LAMP, Linux, Apache, MySQL et PHP / Python créent une collection d'outils qui se complètent bien. Dans cette application, Python est utilisé pour accéder à l'interface série USB et analyser les données, PHP sert d'intermédiaire entre la base de données MySQL et la page Web, MySQL stocke les données reçues et Apache propose une page Web qui utilise JavaScript pour afficher les données reçues de manière dynamique. Il est également intéressant de noter que j'utilise des scripts Bash pour gérer les opérations du système et éviter les erreurs de processus en double.




Pour fournir plus de détails, j'ai écrit un programme python qui, à l'aide de la bibliothèque pySerial (https://pythonhosted.org/pyserial/), se connecte à l'interface USB UART, initie une connexion avec l'ESP32, puis commence à collecter des données. Les données collectées sont horodatées, le nom de la couleur détectée est trouvé, puis le paquet de données nouvellement formé est poussé vers PHP pour être stocké dans la base de données MySQL. Afin de trouver le nom d'une couleur, python convertit d'abord la valeur reçue de ARVB en espace colorimétrique HSV. L'espace colorimétrique HSV présente l'avantage de représenter la teinte sous forme de valeur linéaire. J'utilise cette valeur et la compare avec une table de correspondance (LUT) pour déterminer le nom de la couleur.

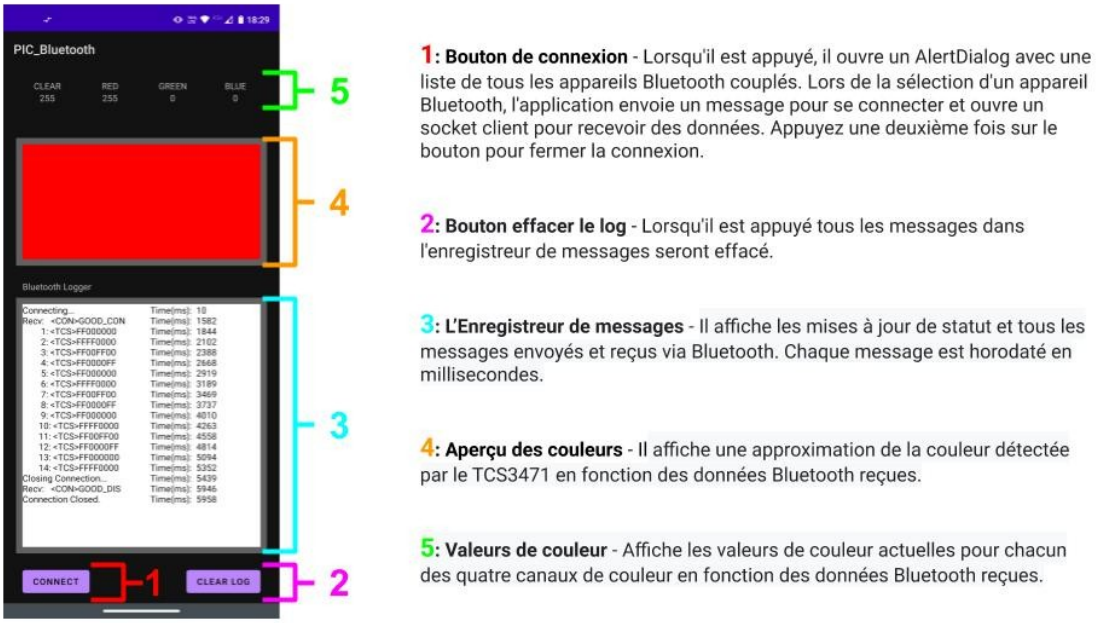
Le composant PHP fonctionne pour transmettre des données entre chacun des composants de l'application. PHP interagit directement avec la base de données MySQL en créant des commandes d'insertion et de requête lorsque les données passent de python puis vers JavaScript. Le composant PHP convertit également les données de la base de données au format JSON afin qu'elles puissent être tracées sur un graphique linéaire. Les méthodes PHP Post() et Get() sont utilisées pour faciliter le flux de données de manière dynamique. PHP insère des données dans une base de données MySQL en utilisant la structure décrite dans ce script MySQL:

```
CREATE TABLE IF NOT EXISTS ColorSerial(
        id INTEGER UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        colorname CHAR(8),
        alpha_v TINYINT UNSIGNED,
        red_v TINYINT UNSIGNED,
        green_v TINYINT UNSIGNED,
        blue_v TINYINT UNSIGNED,
        time_s CHAR(11));
```

Le front-end de l'application Web est écrit en HTML/CSS et JavaScript. Bien que j'aie envisagé d'utiliser PHP pour tracer les données, je trouve qu'avec JavaScript c'est plus facile a mettre en œuvre l'affichage dynamiques des données avec des tâches asynchrones (AJAX). Cela m'a également permis de diviser l'application en éléments modulaires, ce qui facilite la maintenance et le débogage. L'interface utilisateur de l'application Web se compose de quatre composants principaux: les boutons pour démarrer et terminer une connexion, un tableau pour afficher les données du capteur, un graphique linéaire pour tracer les données et une vue couleur pour afficher la couleur actuelle du capteur. Le tableau et le graphique linéaire sont tous les deux utilisés à l'aide de l'API Google Charts (https://developers.google.com/chart). Les boutons de démarrage et d'arrêt déclenchent un script Bash qui lance le programme python.

**Application Android**

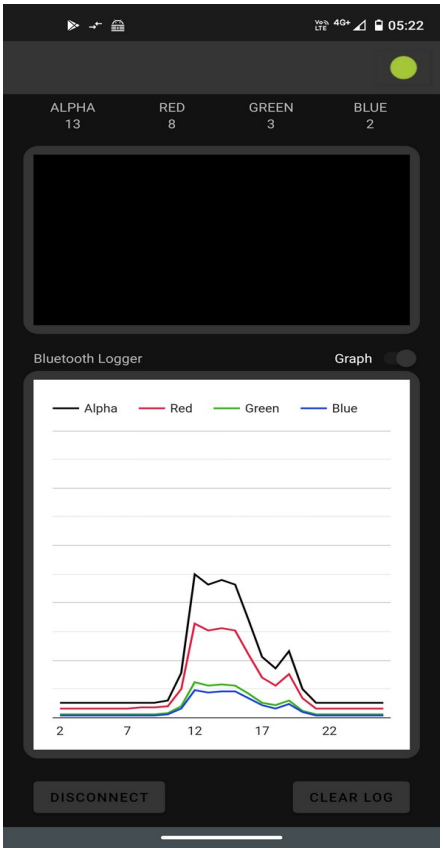
L'application Android a été développée dans Android Studio en utilisant le langage Kotlin. Je republie le diagramme de MainActivity de mon application que j'ai créé en décembre car la fonctionnalité front-end est restée inchangée (à une exception près).




Il y a plusieurs changements majeurs dans le fonctionnement de l'application sur son back-end. Les données sont désormais écrites dans une base de données SQLite à l'aide de la méthode de l'adaptateur Android Room. Dans la version précédente de l'application l'enregistreur de messages affichait des données directement depuis l'interface Bluetooth. L'enregistreur de message a été remplacé par une Recyclerview qui est liée à la base de données SQLite et se met à jour de manière dynamique à mesure que les données s'introduisent dans la base de données. Cette méthode a l'avantage d'être plus efficace en mémoire et elle me permettre d'éliminer une grande partie du code utilisé pour analyser les chaînes de texte.

L'autre changement majeur apporté à l'application est que les données peuvent désormais également être tracées sur un graphique, tout comme dans l'application Web. Le graphique linéaire est créé à l'aide d'une Webview et j'ai ajouté un interrupteur à bascule pour activer le graphique linéaire. La raison pour laquelle j'ai choisi d'utiliser une Webview est qu'elle me permet de réutiliser une partie du code que j'ai développé pour l'application Web LAMP. Les Webviews sont également un outil puissant pour le développement multiplate-forme, je l'ai donc vu comme une opportunité d'acquérir plus de compétences avec Android Studio.

Voici une capture d'écran de la version actuelle de mon application Android:




← Bouton à bascule pour activé le graphique linéaire

← Le Webview contenant le graphique linéaire est superposée sur le tableau qui affiche les données

**Conclusion**

Comme vous pouvez le voir, ce projet a démontré le déploiement d'un large éventail de technologies utilisées dans le développement de l'IoT. Il existe encore des domaines dans lesquels ce projet pourrait être étendu. La collecte de données et l'utilisation de bases de données sont généralement associées à une certaine forme d'analyse des données. Alors peut-être que si je revisite ce projet, je mettrai en œuvre une forme d'analyse ou d'apprentissage d'IA pour utiliser les données collectées. Un autre aspect important dans les déploiements IoT du monde réel est qu'ils utilisent des réseaux de nombreux capteurs déployés. La capacité de collecter des données à partir d'un capteur est utile pour créer une conception de preuve de concept. Cependant, à un moment donné, il devra être étendu avec plusieurs capteurs et le déploiement de réseaux de capteurs sans fil (WSN).

Bien que je n'ai fait qu'effleurer la surface de chaque couche, je pense avoir démontré une solide compréhension dans chaque domaine du développement de l'IoT. En plus de mettre en pratique les compétences acquises tout au long de mes études universitaires, cela m'a donné l'opportunité d'étendre mes compétences préexistantes dans des domaines tels que le développement Web et les bases de données. Je suis certain que je m'appuierai sur ces compétences dans mes projets futurs.

Tout le code de ce projet est disponible sur mon Github: https://github.com/GonzoDMX/ColorTooth