

ConnectObjects

Projet préparé sous la direction de M. Antoine DUTOT

Lien du Git : <https://github.com/DeadMeon/ConnectObjects>

M'hirsi Aziz

ma174899@etu.univ-lehavre.fr

Sommaire

Sommaire	1
Vue d'ensemble	3
Objectifs	3
Quels sont les problèmes rencontrés actuellement ?	3
Les objectifs attendus avec ce nouveau logiciel.	3
Marché ciblé	4
Le profil des futurs utilisateurs	4
Les logiciels concurrencer	4
Caractéristiques	4
Les fonctionnalités	4
Les contraintes	4
Langage de programmation	4
Charte Graphique	5
La structure de l'Interface graphique	6
Recherche	7
Communication entre ConnectObjects et l'Arduino	7
Le fonctionnement des échanges sur un réseau.	7
TCP/IP	8
UDP	8
Trouver les appareils sur le réseau	9
Broadcast et Multicast	10
Routage	10
Multicast	11
Identification	11
Communication entre ConnectObjects et l'API	12
L'Authentification	12
La Vérification	13
Les Requêtes	13
Interface graphique	14
Les FXML	14
Le CSS	14
Les Controller	14
Interactivité	14

Développement	15
Communication entre ConnectObjects et l'Arduino	15
Côté Arduino	15
Côté ConnectObjects	15
Communication entre ConnectObjects et l'API	17
La Vérification	18
Les Requêtes	19
L'Authentification	19
Interface graphique	20
AuthTwitchController	20
CreateObjectController	21
MenuPrincipalController	22
ModifyObjectController	23
OptionController	24
Conclusion	25
Bibliographie	26

Vue d'ensemble

ConnectObjects est un logiciel qui permet de connecter des objets via le Wi-Fi et de les interfacier avec des API.

Le but de cette application est de faciliter la création d'objets connectés et de pourvoir les interfacier avec des API, site ou application via [IFTTT](#) (IFTTT) par exemple.

Objectifs

1. Pouvoir interfacier un [Arduino](#) (Arduino) avec l'application et pouvoir interagir via le wifi.
2. Créer une interface utilisateur permettant une gestion des objets.
3. Connexion et ajout de diverses API ([Twitch](#) (Twitch), [Youtube](#) (Google)...).

Quels sont les problèmes rencontrés actuellement ?

Aujourd'hui, quand on désire créer un objet connecté et que l'on souhaite le contrôler via le wifi, il n'existe aucune application permettant de facilement interagir avec les objets tout en les centralisant. Mais aussi aucune ne permet de pouvoir communiquer avec des applications tierces. Il existe certes des sites permettant d'interagir avec certaines API, mais elles ne prennent pas en compte nativement les Arduinos. Ainsi cette application a pour but de résoudre ce problème en interagissant nativement avec des Arduinos avec une antenne Wifi.

Les objectifs attendus avec ce nouveau logiciel.

Créer un programme qui génère le code à mettre sur l'Arduino et il doit permettre d'interagir entre le logiciel et avec un ou plusieurs Arduino via le même WiFi en simultanément. Le logiciel doit au minimum pouvoir permettre la personnalisation de l'objet connecté et intégrer l'API de Twitch pour interagir avec des événements. L'interface utilisateur nécessite d'être simple et compréhensible pour une grande accessibilité.

Marché ciblé

Le profil des futurs utilisateurs

Le logiciel cible tout le monde. Plus précisément ceux qui sont intéressés par le DIY (Do It Yourself) ou du bricolage. Mais aussi les entreprises pour automatiser des tâches.

Les logiciels concurrencer

Aucun.

Caractéristiques

Le logiciel sera un programme d'application gratuit.

Les fonctionnalités

- Gérer plusieurs objets en même temps
- Interaction avec Twitch

Les contraintes

- Option pour modifier le code directement.
- Utiliser des Json pour la sauvegarde des information et faciliter leur exportation.

Langage de programmation

Le logiciel sera codé sous [Java](#) (Oracle) sous [Maven](#) (The Apache Software Foundation) avec la bibliothèque graphique [JavaFX](#) (OpenJFX). Il gèrera toute la partie des requêtes au APIs. Mais aussi C++ pour toute la partie de programmation sur l'Arduino

Charte Graphique

La police d'écriture sera Montserrat avec une taille de 16 pixels, pour les titres la police d'écriture sera Regular 400 en taille 25 pixels. Les polices d'écriture ont été choisies sur [Google Fonts](#) (Google)

Les couleurs seront des nuances de violet sombre proche #62438C pour les options et des nuances de bleu sombre #4D4DFF pour le centre de l'interface. Ces couleurs permettent un excellent contraste avec du texte blanc. Pour les nuances, j'ai utilisé l'outil de couleurs [Adobe Color](#) (Adobe).

Pour chaque objet, on utilisera une image pour définir le type de carte, ou si cette dernière n'existe pas alors on utilisera un logo du type de carte.

Nous utilisons aussi les SVG pour faciliter la compréhension de l'interface. Ces derniers viennent de la banque d'icône de [Google Material Icônes](#) (Google)

La structure de l'Interface graphique

- Un menu principal
 - Accès à tous les objets déjà enregistrés
 - Bouton permettant d'en créer un nouveau.
 - La présence d'un bouton pour accéder aux options.
- Une page de création.
 - Choix du Nom et de l'icône pour le menu principal
 - Choix du type carte (seulement Arduino pour l'instant, peut être Raspberry Pi pico plus tard).
 - Choix de la carte et choix de leur pin
 - Une zone de sélection des événements reçus.
 - Avec un onglet par API.
 - Avec la liste des événements qui peuvent être traités.
- Une page de l'objet
 - Un onglet de modification des informations de l'objet.
 - Modification du nom.
 - Modification du SSID et du mots de passe Wifi
 - Visualisation de l'identifiant de la carte ou de l'ordinateur
 - Un onglet de sélection des événement reçus.
 - Avec un sous-onglet par API.
 - Avec la liste des événements qui peuvent être traités.
 - Un onglet pour modifier les variables générales.
 - Un onglet pour modifier le code d'initialisation
 - Un onglet pour modifier le code lors de la réception d'un événement
 - Un bouton pour supprimer l'objet.
 - Un bouton pour générer le code de l'objet
 - Un bouton pour sauvegarder l'objet.
- Une page d'option
 - Le chemin du dossier où seront enregistrés les fichiers générés.
 - Un bouton pour activer ou désactiver chaque API
 - Si le token n'est pas généré, alors on ouvre une fenêtre Web interne et on se connecte au lien fourni par l'API.
 - Un bouton de sauvegarde des options.
 - Un bouton de remise à zéro des options.

Recherche

Communication entre ConnectObjects et l'Arduino

On va chercher à trouver un moyen d'effectuer des échanges de message entre ConnectObject et L'Arduino. La compréhension du fonctionnement des échanges réseau est nécessaire pour comprendre cette partie. Pour cela un petit cours sur le fonctionnement des communications réseau. Je vous conseille aussi fortement le livre *Les Réseaux Pour les Nuls* (Lowe)

Le fonctionnement des échanges sur un réseau.

Pour qu'un appareil puisse communiquer, il faut qu'il puisse envoyer des informations à un autre ordinateur sur le même réseau que lui. Malheureusement, on ne peut pas envoyer les informations n'importe comment, sinon ça revient à mettre plusieurs personnes qui ne parlent pas la même langue dans une même pièce et espérer qu'ils se comprennent. Pour qu'ils puissent se comprendre, ils doivent parler une même langue. Notre langue ici sont les protocoles de communication et les mots sont les paquets !

Mais pourquoi utiliser un protocole de communication ?

On utilise les protocoles de communication car ils nous permettront de ne pas avoir à gérer les parties communication bas niveau, c'est-à-dire de ne pas avoir à gérer nous-même la partie des communications entre les ordinateurs. Aussi bien physique (l'envoi de données par le câble ethernet en passant par la carte réseau) que traitement (la gestion du paquet et son traitement pour être compréhensible par notre programme).

Il existe plusieurs protocoles de communication et qui ont des caractéristiques différentes. Les principaux sont :

- TCP/IP
- UDP

Nous allons voir les avantages et les inconvénients de chacun.

TCP/IP

Le protocole TCP (Transmission Control Protocol) ou TCP/IP est un protocole qui va vérifier que chaque paquet est bien arrivé. Pour cela il va envoyer le paquet et attendre que l'autre programme lui envoie un paquet pour lui dire si il l'a reçu ou non. Cette méthode est très pratique pour s'assurer qu'un message arrive bien à destination. Ce protocole est très utilisé pour le web ou les téléchargements par exemple.

Les avantages

- Meilleur fiabilité
- Détection des erreurs et corrections
- Accusé de réception

Les inconvénients

- Fiabilité vitesse

UDP

Le protocole UDP (User Datagram Protocol) à l'inverse fonctionne comme un flux de données. C'est à dire qu'il envoie les paquets sans se soucier de qui va les recevoir ou bien s'ils sont corrompus ou non. Ce protocole est très utilisé pour le streaming de vidéo ou ce n'est pas très grave d'avoir un paquet manquant au vu de la vitesse d'affichage.

Les avantages

- Meilleure vitesse
- Non bloquant par l'autre machine dont on attend une réponse

Les inconvénients

- Des paquets peuvent être perdus ou corrompus.

Pour ConnectObjects, l'objectif étant d'envoyer à tous les objets ayant souscrit à un événement à l'instant T. Comme la quantité de paquet envoyé est très faible, le risque de perte ou de corruption est très faible et même s'il existe ne pose pas de problème car on veut plus de flexibilité que de fiabilité. Et de plus si un objet se déconnecte ou vient à planter. On se retrouverait alors bloqué en attendant qu'un accusé de réception qui n'arrivera jamais ce qui est encore plus problématique.

Ainsi on va préférer le protocole UDP au protocole TCP/IP.

Trouver les appareils sur le réseau

Maintenant que l'on a le protocole, il faut un moyen de trouver les objets sur le réseau et de pouvoir les reconnaître de manière efficace et de les relier aux identifiants de chaque objet. Tout en utilisant seulement des outils natif d'Arduino pour que cela reste compatible avec le plus d'appareil possible.

Ma première approche était d'envoyer sur le réseau un paquet à toutes les adresses IP du sous réseau. C'est à dire plus concrètement envoyer un paquet à toutes les adresses IP qui peuvent être prises par un appareil électronique sur le même routeur ou modem (comme par exemple la box d'une maison). Ainsi chaque objet créé par ConnectObjects ayant reçu la requête renverra un message de confirmation pour que ConnectObjects identifie les objets présents sur le même réseau que lui. Le message de confirmation ne peut pas être l'identifiant de l'objet car sinon n'importe quelle réponse sur le réseau peut être prise pour une confirmation.

Malheureusement avec cette méthode, comme on ne peut pas identifier les objets, on est obligé de renvoyer un message via la liste d'ip récupérée pour les objets qui ont répondu au premier message. Pour pouvoir les identifier, on va leur envoyer une requête leur demandant de s'identifier pour qu'il nous envoie leur identifiant et qu'on puisse les stocker dans ConnectObjects. Et comme les nouveaux objets ne peuvent pas connaître l'adresse IP de l'ordinateur où ConnectObjects est installé, le seul moyen pour vérifier s'il y a de nouveaux objets sur le réseau c'est de refaire toutes les étapes précédentes en boucle pour actualiser la liste d'objets.

Ce système pose beaucoup de problèmes. Le premier est que pour maintenir la liste des objets connectés à jour on doit créer un [Thread](#) (Wikipedia) (C'est un programme indépendant créé par l'application principale qui permet de faire des tâches en parallèle) spécialement dédié à faire des requêtes sur le réseau tous les x temps. Ce qui pose dans la foulée, le problème du spam de requête sur le réseau et parasite tous les objets présents sur le réseau. Cela pose aussi un problème de sécurité, car tout appareil sur le réseau reçoit un message de ConnectObjects et donc peut le copier pour le renvoyer sur le réseau et facilement récupérer toutes les informations des objets. Il a aussi la possibilité que le programme reste bloqué en attente de réponse si un objet se déconnecte durant la phase d'enregistrement de l'identifiant comme il sera en attente d'un message qui n'arrivera jamais.

Dû à tous ces problèmes, il m'a fallu penser à une autre méthode de communication qui puisse utiliser tous les avantages du protocole UDP.

Broadcast et Multicast

Pour respecter l'objectif de faire le programme de communication de manière native sur l'Arduino, j'ai passé plusieurs heures à lire la documentation d'Arduino pour trouver une méthode plus simple. Et je suis tombé sur une méthode de routage, le Multicast. Directement intégré dans la bibliothèque de gestion de la carte wifi, Cette méthode de routage permet d'envoyer un message sur une adresse de Multicast un message et tous les appareils écoutant cette adresse en multicast recevront le message.

Pour mieux comprendre, on va voir les différentes méthodes de routage possible puis, voir les avantages du Multicast dans notre cas.

Routage

Le Broadcast est une méthode de routage. Mais qu'est-ce qu'une méthode de routage ? Le routage c'est le dispositif qui choisit le chemin par lequel va passer le paquet. Par exemple, c'est lui qui décide par quel serveur passer pour atteindre le serveur du site auquel vous voulez accéder. Ainsi on peut envoyer à un destinataire le paquet si on connaît à l'avance le destinataire. Or si on ne connaît pas le destinataire, il y a trois méthodes pour envoyer le paquet : Le Broadcast, L'Anycast et le Multicast.

Le Broadcast revient à envoyer à tous les appareils sur le même réseau sans distinction. Ce qui est très pratique pour interagir avec d'autres appareils sur le réseau mais a pour défaut de parasiter tout le réseau à chaque envoi et on veut éviter ça au maximum.

L'Anycast est l'opposé du Broadcast, cela revient à envoyer le paquet à n'importe lequel des appareils sur le réseau, (généralement le plus proche ou le plus efficace). C'est une méthode de routage très utile pour choisir entre plusieurs serveurs. Par exemple, lorsque l'on veut se connecter au site de [Google](https://www.google.com) (Google), comme il existe plusieurs serveurs au tour du monde ayant un copy du site de Google, le navigateur va se connecter au serveur le plus proche de votre position géographique pour avoir la plus faible latence (temps de réponse) possible. C'est exactement le travail de l'Anycast.

Le Multicast quant à lui fonctionne de la même manière qu'un groupe sur une messagerie instantanée, seules les personnes intéressées par ce groupe recevront tous le paquet. Ce qui permet de ne pas se préoccuper de tout parasitage des appareils qui ne sont pas intéressés par les paquets.

Multicast

Maintenant que l'on a vu différentes méthodes de routage, nous allons voir le fonctionnement du Multicast et voir en quoi le Multicast nous arrange pour ConnectObjects.

Pour mieux comprendre le fonctionnement du multicast, on peut le comparer à une [CB](#) (Wikipédia) ("Citizens band radio"). Sur une CB, on se connecte à une fréquence et on reçoit en temps réel les messages émis sur cette fréquence. On peut aussi envoyer des message pour répondre sur cette même fréquence et tout le monde le recevra aussi pas que l'émetteur. Donc c'est dans les deux sens.

Ici, dans notre comparaison la fréquence va être une adresse ip de multicast, ces adresses sont entre 224.0.0.0 à 239.255.255.255 ou certaines sont réservées et ne peuvent être utilisées librement mais le reste sont libres. Ainsi en se connectant à une même adresse de multicast, les objets peuvent facilement communiquer avec ConnectObjects.

Cette méthode de routage fonctionne comme des observateurs (réagissant à un événement). On n'a pas besoin de vérifier si l'appareil est présent sur le réseau car on attend pas de réponse. Et du coup cela permet de ne pas avoir à synchroniser et mettre en mémoire une liste d'adresses sur le réseau. On a juste une liste des appareils créés avec ConnectObjects et on écoute si on reçoit un message qui nous est destiné.

Identification

Malheureusement avec le multicast on doit gérer la partie identification au niveau de l'application et pas du protocole. Pour cela il fallait créer un identifiant unique pour chaque appareil et les utiliser pour transmettre.

Une méthode simple de faire cela revient a lors de la création de l'objets de lui génère un identifiant et lui mettre dans le code que l'arduino embarquera avec l'identifiant de l'application qui l'a créé comme cela sur chaque carte on enregistre le destinataire et l'expéditeur et ConnectObjects enverra le paquet en y ajoutant l'identifiant de l'objet ciblé suivi de son identifiant et en fin du message.

De cette manière on peut facilement traiter les différents paquets et savoir lesquels sont pour l'objet. Cela permet aussi d'avoir plusieurs ordinateur ayant ConnectObjects sur le même réseau, car ils ont tous l'identifiant de l'application qui les a créé.

Communication entre ConnectObjects et l'API

Maintenant que l'on a vu comment faire fonctionner la partie entre ConnectObjects et L'Arduino. On va s'occuper de la partie entre ConnectObjects et L'API. Tout d'abord, qu'est ce que Twitch et pourquoi l'API de Twitch plutôt qu'une autre.

Twitch est un site de divertissement destiné au partage de contenus en direct, on y retrouve tout type de contenu, de la création, des jeux vidéos ou des cours par exemple. Pourquoi eux ? Leur API a une utilisation assez générique et leur système est basé sur un système d'abonnement payant ou bien gratuit donc on peut récupérer ainsi les personnes ayant souscrit à un abonnement payant (Subscription souvent raccourci en Sub) ou bien à un abonnement gratuit (Follow).

Pour faire mes tests, j'ai utilisé un outil qui s'appelle [Insomnia](#) (Kong Inc.), Cet outil permet de facilement tester des requêtes et voir ce que va recevoir l'application. Ainsi on va voir comment récupérer ces informations, commencent par l'authentification à leur API.

L'Authentification

En utilisant la documentation, on apprend qu'il faut enregistrer sur la [console de développeur](#) Twitch (Twitch) une clé client qui va servir à identifier l'application pour les requêtes.

Une fois cela fait, on fait une requête à l'API pour s'identifier. Cette dernière va nous renvoyer une page html qu'il faudra utiliser par le ConnectObjects pour permettre de s'identifier via le formulaire fourni. De cette manière, l'application n'a pas à gérer la partie identification chez Twitch.

Une fois le formulaire envoyé, si il est correct il on est redirigé vers une URL que l'on a choisie lors de l'enregistrement de la clé client dans lequel il y a le token, une suite de caractères permettant d'identifier l'utilisateur pour une application spécifique qui permet de récupérer les informations nécessaires. Le token nécessite des permissions qui sont émises lors de la demande d'envoi du formulaire. Ici ConnectObjects n'aura besoin que de la permission de lecture des abonnements payants. Ainsi même si le token est récupéré, il reste complètement inoffensif pour la sécurité du compte de l'utilisateur. Ainsi, on pourra le stocker avec les autres informations sur la machine sans avoir de problème de sécurité.

La Vérification

Pour rendre le token utilisable il faut le vérifier, en envoyant une requête de validation avec notre clé client et le token. Si la requête est validée alors on récupère l'identifiant de l'utilisateur connecté, il sera utile pour les requêtes.

Les Requêtes

Maintenant que l'on a un token valide, on peut demander à récupérer les informations de l'utilisateur, mais avant le nom du compte suffisait sauf que la nouvelle API change tout cela en nécessitant l'identifiant du compte que l'on récupère lors de la validation. Ainsi on peut récupérer toutes les informations validées par le token. Ces dernières seront retournées sous forme de [JSON](#) (Crockford) que l'on devra rendre compréhensible par ConnectObjects.

Interface graphique

Pour l'interface visuel, comme par le passé, nous avons utilisé en cours Swing, j'ai décidé de partir sur une autre bibliothèque graphique et essayer JavaFX. Donc pour le reste du projet j'utiliserai JavaFX.

Pour comprendre le fonctionnement de JavaFX, j'ai dû regarder la documentation et des vidéos, tutoriels pour tout comprendre mais pour résumer JavaFX est composé en trois parties.

Les FXML

Les FXML sont le squelette de code, il sont l'emplacement des objets visuels avec lesquels l'utilisateur va interagir, la structure de l'interface. Avec cela on peut facilement déterminer la position de chaque objet dans l'interface.

Chaque objet placé va permettre des interactions différentes en fonction de l'objet. Par exemple, un Button ne pourra pas retourner s'il est sélectionné ou non mais les CheckBox ou les ToggleButton le peuvent. Ou bien l'utilisateur ne peut modifier le texte que d'un TextField ou d'un TextArea.

Le CSS

Le CSS est comme le CSS pour les l'HTML sauf que cette fois-ci il est appliqué au FXML. Il permet la customisation des parties visuel et permet d'appliquer la charte graphique sur l'application.

Les Controller

Les Controllers sont quant à eux le code qui va réagir aux interactions avec l'utilisateur. Pour continuer notre comparaison, ça correspond au JavaScript d'une page web.

Ainsi c'est la partie qui va interagir avec les autres parties du programme, ajouter des nouveaux objets, en supprimer des objets, créer les fichiers.

Interactivité

Maintenant que l'on a l'outil il fallait réfléchir à un moyen de rendre cela facile d'accès. LE meilleur moyen c'est de le faire de manière simple et d'indiquer les actions possibles avec des petites images pour faciliter la compréhension de ConnectObjects . Donc j'ai décidé de faire une application qui ressemble à une application mobile dans le design, quelque chose de simple mais efficace au vue du public ciblé.

Développement

Communication entre ConnectObjects et l'Arduino

Côté Arduino

Mon objectif étant de le faire de manière native sur tous les Arduino, je ne peux utiliser que la bibliothèque officielle d'Arduino et qui sont inclus de base. Pour cela, j'ai utilisé un exemple de la documentation de la bibliothèque officielle de la gestion du Wifi, [WiFiNINA library](#) (Arduino), qui permet d'envoyer et de recevoir des message via le protocole UDP, [WiFiUdpSendReceiveString](#) (Arduino). Je l'ai modifié pour pouvoir fonctionner en Multicast en changeant simplement la fonction begin() de [WiFiUdp](#) (Arduino) qui permet d'envoyer un message a une adresse IP précise via le protocole UDP en beginMulticast() qui envoie vers une adresse de multicast déterminée.

Maintenant que l'on peut envoyer et recevoir des messages il reste a traiter les paquets reçus. On vérifie que le message nous est bien destiné en séparant le paquet avec un séparateur défini des deux côtés, ici ":". Le paquet peut être traité de manière différente de s'il vient de ConnectObjects ou d'un autre Arduino par exemple. Une fois traité, si le message vient de l'application et que ce dernier est un message prédéfini, il sera traité automatiquement sinon le message sera envoyé à la fonction runCode qui est le code écrit par l'utilisateur.

Maintenant que le code de l'arduino fonctionne il faut que ConnectObjects générer le fichier.

Côté ConnectObjects

Grâce au code Arduino qui gère la communication vu ci-dessus, on peut facilement créer de nouveaux programmes en ajoutant simplement les fonctions runCode et InitCode ainsi qu'en début de fichier les variables générales qui seront utilisées par les fonctions ci-dessus.

On va aussi générer le fichier arduino_secret.h qui est un fichier qui va contenir les identifiants ainsi que le SSID et le Mots de passe utilisé pour se connecter au Wifi. Ils sont mis dans ce fichier pour compliquer la récupération de ces informations confidentielles.

Pour rendre tout cela accessible le fichier sera enregistré dans le fichiers Document de l'utilisateur sur Windows pour l'instant mais à terme l'objectif est de le rendre multiplateforme.

Tout cela est géré par le manager des Arduinos qui est sur un thread à part pour écouter si un message lui est destiné. Il contient aussi une liste des objets créés, ces derniers contiennent toutes les informations nécessaires pour créer le code.

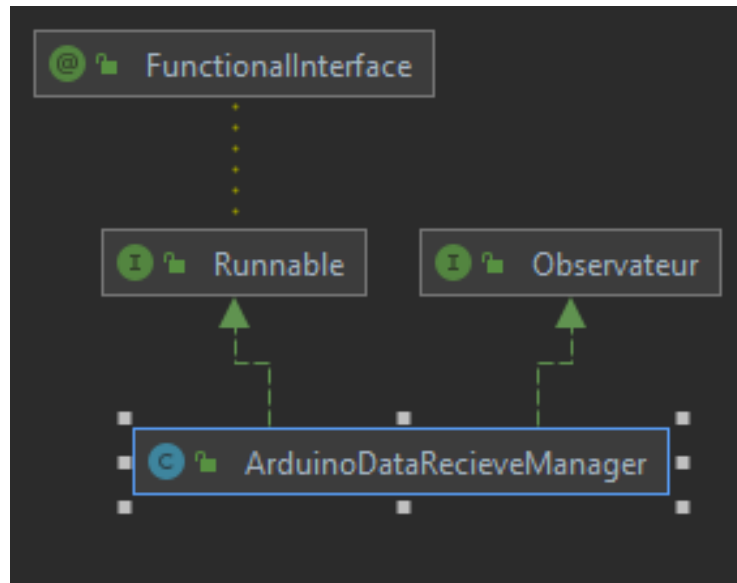


Diagramme de la classe `ArduinoDataReceiveManager`

Chaque objet de la liste sont des objets créés à partir de la classe `ArduinoEntity` ou des classes héritant de `ArduinoEntity`. Cette classe contient le nom de l'objet, le nombre de pin sur la carte ou bien la liste des objets. En somme toutes les informations manquantes pour créer le fichier.

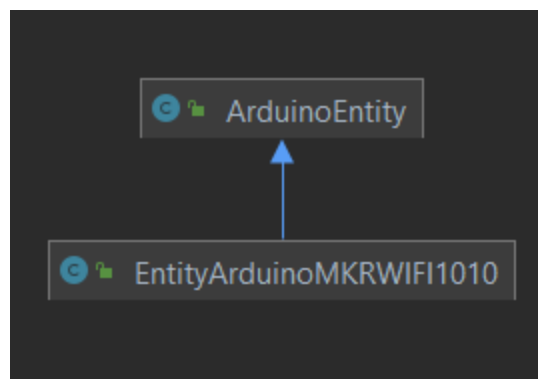
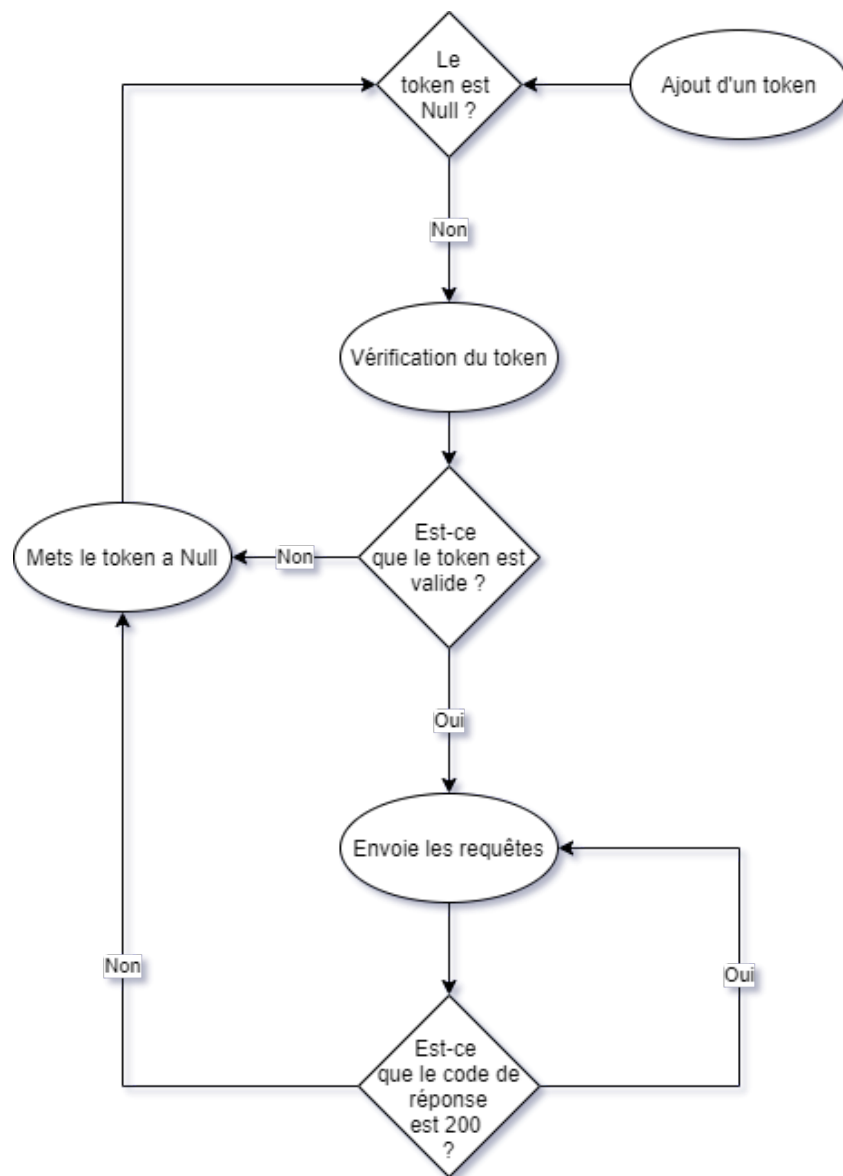


Diagramme de la classe `ArduinoEntity`

Communication entre ConnectObjects et l'API

Maintenant que l'on a les requêtes qu'il faut envoyer et que l'on connaît ce que l'on va recevoir, il ne reste plus qu'à les implémenter en Java. Pour cela on va utiliser une librairie de java.net qui se nomme `URLConnection`. Cette bibliothèque permet de faire des requêtes HTTP. Une fois que l'on exécute on peut récupérer le code de réponse avec la fonction `getResponseCode()`. Et si le code de réponse est 200 ce qui correspond à tout c'est bien passé, on peut alors récupérer la réponse du serveur.



On va donc voir ce que l'on fait pour chaque requête. On va commencer par celle qui retourne un JSON.

Pour gérer toute cette partie, on a un manager, ce dernier va tourner sur un thread à part pour lui permettre de faire les requêtes tous les x temps si il est activé dans les options de l'application.

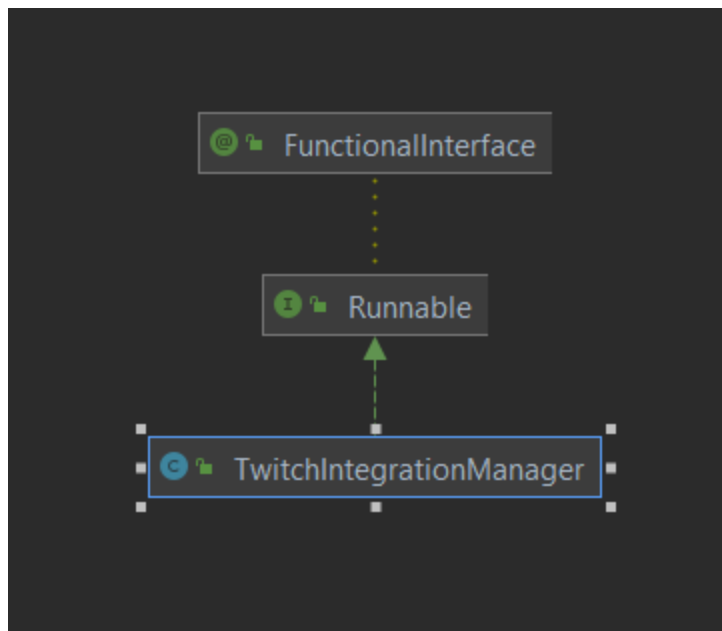


Diagramme de la classe TwitchIntegrationManager

Pour nous simplifier la vie, on va utiliser la librairie [GSON](#) (Google) créée par Google qui permet de convertir les Json en Objets et de pouvoir les utiliser comme de simples objets.

La Vérification

Pour la vérification, si le code de réponse est 200, on récupère le JSON que l'on le transforme en Objet Validate qui contient les mêmes champs que le JSON. On ne récupère que l'identifiant de l'utilisateur.

Si le code de réponse est différent de 200, on remet le token a null pour qui soit automatiquement remis à zéro par l'interface. Comme cela on est sûr que le token est valide.

Les Requêtes

On va faire pareil que pour la vérification sauf que dans ce cas si on va garder les objets pour pouvoir les comparer avec les anciens et si ils sont différents alors on envoie à tous les objets ayant souscrit à cet événement.

L'Authentification

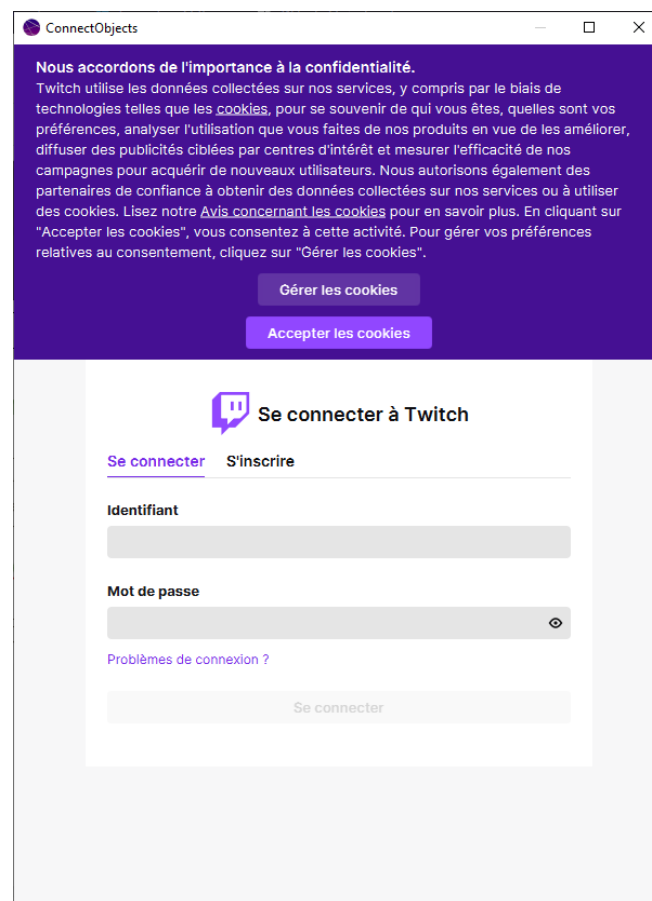
Comme L'authentification requiert d'être affichée, on la reverra dans la partie interface. Mais notre manageur va, lorsqu'il récupère, un token vérifier s'il est valide et s' il ne l'est pas, alors le token sera remis à zéro.

Interface graphique

L'interface est composée de plusieurs contrôles pour gérer chaque interface visuelle. On va voir chacun d'entre eux.

AuthTwitchController

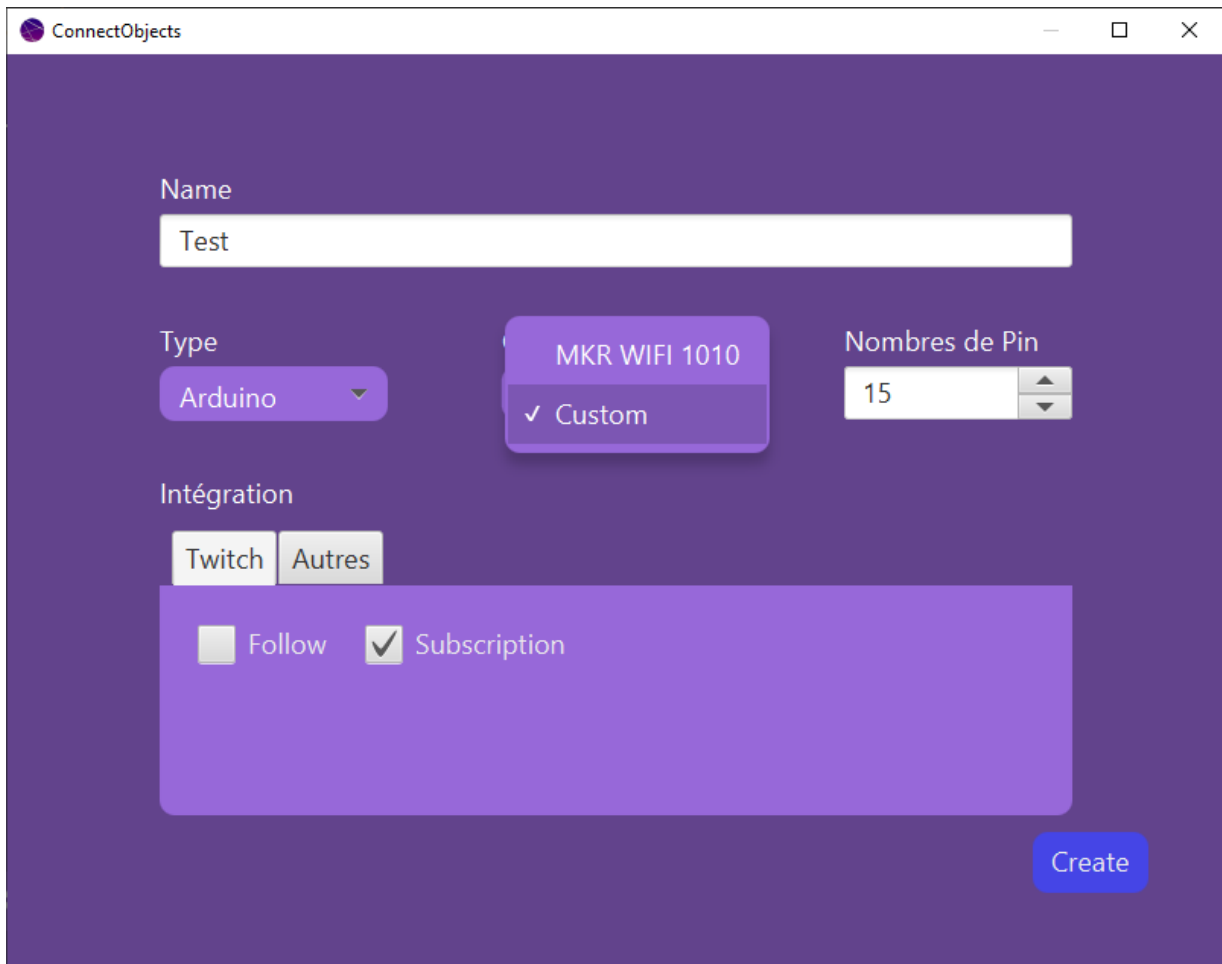
C'est le contrôleur de l'affichage et du traitement de l'authentification pour Twitch. On va y charger la page et à chaque changement d'URL, le programme va vérifier si la page affichée commence par le site qui a été enregistré dans la console de Twitch. Dans ce cas là, il va récupérer dans le lien le token et l'envoyer au manager des requêtes vers l'API de Twitch pour pouvoir l'utiliser.



Capture d'écran de la fenêtre d'authentification

CreateObjectController

Ce contrôleur est celui du formulaire, il va vérifier que le nom de l'objet n'existe pas déjà dans la liste des objets enregistrés ou bien qu'il n'y a aucune case vide. Si toutes les vérifications sont bonnes, l'objet sera créé en fonction des paramètres entrant. Ensuite il sera ajouté à la liste d'objets avant d'en faire une sauvegarde locale.

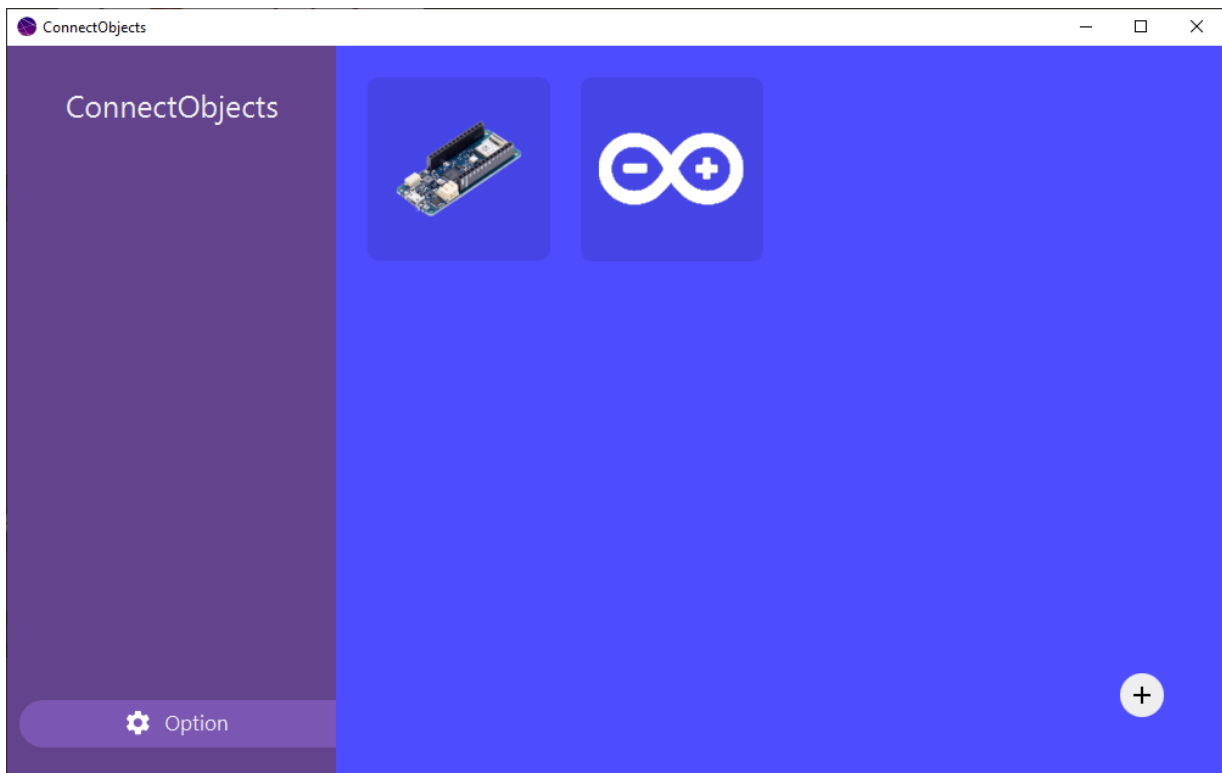


The screenshot shows a web application window titled "ConnectObjects". The form has a purple background. It includes a "Name" input field with the text "Test". Below it, there are two sections: "Type" and "Nombres de Pin". The "Type" section has a dropdown menu currently showing "Arduino", with a modal menu open showing "MKR WIFI 1010" and "✓ Custom". The "Nombres de Pin" section has a numeric input field with the value "15". Below these, there is an "Intégration" section with two tabs: "Twitch" and "Autres". Under the "Twitch" tab, there are two checkboxes: "Follow" (unchecked) and "Subscription" (checked). A blue "Create" button is located at the bottom right of the form.

Capture d'écran du menu de création d'objet

MenuPrincipalController

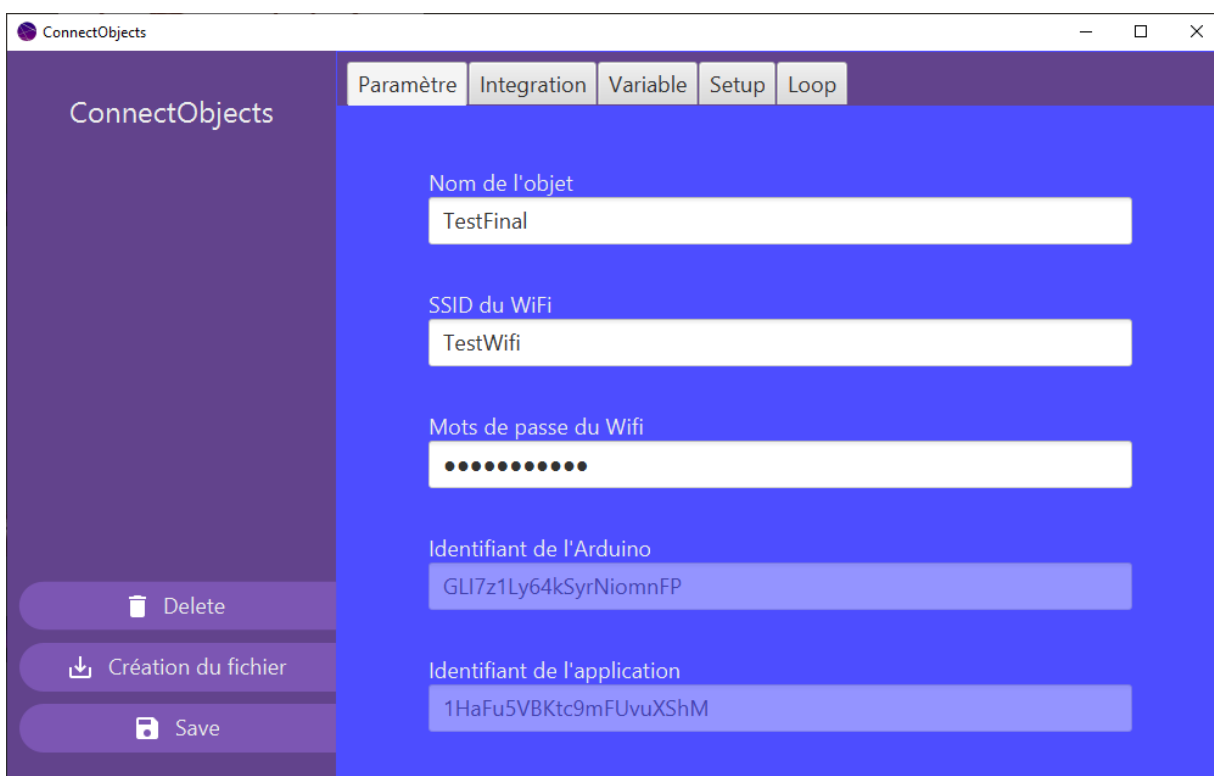
Comme son nom l'indique c'est le controller du menu principal. C'est lui qui est appelé en premier et il va créer un bouton pour chaque appareil dans la liste d'arduino fournis par le manageur des Arduinos.



Capture d'écran du menu principal

ModifyObjectController

A chaque fois qu'un objet est sélectionné dans le menu principal, il ouvre la page de modification de l'objet en question. La partie modification est gérée par ce controller. Il effectue plusieurs actions comme la suppression d'un objet de la liste d'arduino. Mais aussi lance la génération du code pour l'objet. On peut y modifier une grande partie des informations de l'objet.



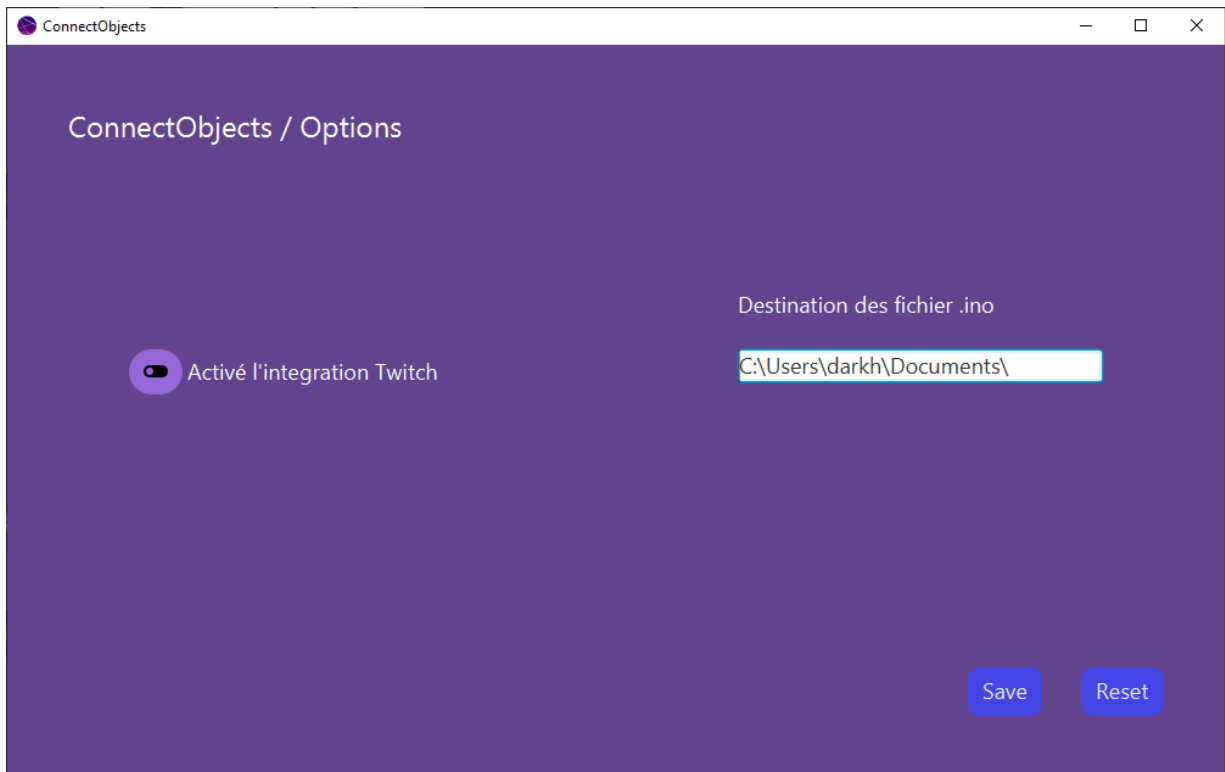
The screenshot shows the 'ConnectObjects' application window. The title bar reads 'ConnectObjects'. The interface has a dark purple sidebar on the left with the text 'ConnectObjects' and three buttons: 'Delete' (with a trash icon), 'Création du fichier' (with a download icon), and 'Save' (with a floppy disk icon). The main area has a blue background and a tabbed interface with tabs: 'Paramètre', 'Integration', 'Variable', 'Setup', and 'Loop'. The 'Paramètre' tab is active. It contains five text input fields with labels and values:

- Nom de l'objet: TestFinal
- SSID du Wifi: TestWifi
- Mots de passe du Wifi: (masked with dots)
- Identifiant de l'Arduino: GLI7z1Ly64kSyrNiomnFP
- Identifiant de l'application: 1HaFu5VBKtc9mFUvuXShM

Capture d'écran du menu de modification

OptionController

Ce dernier controller est celui qui va gérer les options, elle modifie des fonctionnalités à travers toutes l'application. On peut aussi y activer les différentes interfaces avec les API indépendamment les une des autres et si c'est la première fois qu'elle est active alors de l'initialiser en appelant la fenêtre d'authentification.



Capture d'écran du menu des options

Conclusion

L'objectif de cette application est de faciliter l'accès à la créativité pour tout le monde. Pour moi, cette application m'aide à créer des projets qui me passionnent. J'aime créer des petits objets qui automatisent une tâche ou bien simplement pour de la décoration. Mais malheureusement tout cela a un coup, c'est pour cela que je voulais que l'application soit la plus native possible pour pouvoir créer des objets sans avoir à acheter des composants et ainsi réduire le coût au maximum. Et c'est pour cela que je vais rendre mon projet Open Source pour que le plus de personnes puissent l'utiliser à leur manière. Cela reste après tout la bannière du DIY.

ConnectObject en est encore qu'à sa version 1 et j'ai bien dans l'objectif de la faire évoluer et simplifier au maximum l'accès à la création d'objets DIY qui nous permettent de nous changer la vie. L'objectif de la version 2 sera de créer une interface web et pouvoir faire tourner le programme sur un raspberry pi en tant que serveur sur notre réseau local.

Pour l'instant, je vais essayer d'ajouter petit à petit de nouvelles API pour augmenter le nombre d'interactions possibles mais aussi le nombre de cartes supportées par le programme. Ainsi que le support pour Linux et Mac ainsi qu'ajouter avec la version 1.2 les outils de création visuelle (sans écrire la moindre ligne de code).

Pour moi, cet exercice de gestion de projet était très intéressant car j'ai dû apprendre à faire le cahier des charges, à chercher les concurrents direct possible, etc ... Et j'ai adoré faire ça, j'ai toujours aimé bricoler et créer alors réussir à le combiner avec ma passion fut génial. En plus de cela, ça m'a appris à utiliser JavaFX outils que l'on a jamais utilisé ainsi que comment utiliser une API mise à disposition par un site web. Donc cela reste une très bonne expérience malgré les centaines d'heures à me tirer les cheveux.

Et pour finir, je tiens à remercier M. Antoine Dutot pour m'avoir soutenu et guidé sur ce projet.

Bibliographie

Adobe. "Adobe Color." *Adobe Color*, <https://color.adobe.com/fr/create/color-wheel>.

The Apache Software Foundation. "Maven." *Maven*, <https://maven.apache.org/>

Arduino. "Arduino." *Arduino*, <https://www.arduino.cc/>

Arduino. "WiFiNINA library." *Arduino*, <https://www.arduino.cc/en/Reference/WiFiNINA>

Arduino. "WiFiUdp." *GitHub*,

<https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/WiFiUdp.h>

Arduino. "WiFiUdpSendReceiveString." *Arduino*,

<https://www.arduino.cc/en/Tutorial/LibraryExamples/WiFiNINAWiFiUdpSendReceiveString>

Celonis. "Integromat." *Integromat*, <https://www.integromat.com>

Crockford, Douglas. "JavaScript Object Notation." *JSON*, 2002, <https://json.org/json-fr.html>

Dabo, Mohamed, et al. "Débutez la programmation avec Java." *OpenClassRoom*,

<https://openclassrooms.com/fr/courses/6173501-debutez-la-programmation-avec-java>

Diagrams.net. "Draw.io." *Draw.io*, <https://app.diagrams.net/>

Google. "Google." *Google*, <https://www.google.com/>

Google. "Google Fonts." *Google Fonts*, <https://fonts.google.com/>

Google. "Google Material Icônes." *Google Material Icônes*, <https://fonts.google.com/icons>

Google. "GSON." *Github*, <https://github.com/google/gson>

Google. "Youtube." *Youtube*, <https://www.youtube.com/>

IFTTT. "IFTTT." *IFTTT*, <https://ifttt.com/>

Kong Inc. "Insomnia." *Insomnia*, <https://insomnia.rest/>

Lowe, Doug. *Les Réseaux Pour les Nuls*. 13 ed., First Interactive, 11 mars 2021.

n8n.io. "n8n.io." *n8n.io*, <https://n8n.io/>

OpenClassroom. "OpenClassroom." *OpenClassroom*, <https://openclassrooms.com/fr/>

OpenJFX. "JavaFX." *JavaFX*, <https://openjfx.io/>

Oracle. "Java." *Java*, <https://www.java.com/fr/>

Pipedream, Inc. "Pipedream." *Pipedream*, <https://pipedream.com/>

Twitch. "Console." *Twitch Developers*, <https://dev.twitch.tv/console>

Twitch. "Twitch." *Twitch*, <https://www.twitch.tv/>

Wikipédia. "Citizen-band." *Wikipédia*, <https://fr.wikipedia.org/wiki/Citizen-band>

Wikipedia. "Thread." *Wikipedia*, [https://fr.wikipedia.org/wiki/Thread \(informatique\)](https://fr.wikipedia.org/wiki/Thread_(informatique))

Zapier Inc. "Zapier." *Zapier*, <https://zapier.com/>