

Face Key

Quentin GERARD, Louis L'HARIDON et Matthieu VILAIN
Licence 3 informatique CMI Systèmes Intelligents et Communicants
Université de Cergy-Pontoise

Résumé

Face Key est un projet de gestionnaire de mot de passe par reconnaissance faciale réalisé dans le cadre du projet de synthèse de Licence 3. Il embarque des composantes réseau, base de données, sécurité, application web et machine learning/reconnaissance faciale.

TABLE DES MATIÈRES

I	Présentation de Face Key	2
I-A	Situation initiale	2
I-B	Notre solution : Face Key	2
I-C	Organisation des différents projets	2
I-D	Nos ambitions, nos limites	3
II	Gestion de projet	4
II-A	Répartition des tâches	4
II-B	Les deadlines	4
II-C	Organisation du travail	4
III	Base de Données	6
III-A	Introduction	6
III-B	Analyse du besoin	6
III-C	Modèle Conceptuel des Données	7
III-D	Modèle Logique de Données	7
III-E	Requêtes significatives	8
III-F	Panel d'administration	11
III-G	Conclusion	11
IV	Réseau	12
IV-A	Spécification	12
IV-B	Fonctionnement global	12
IV-C	Les requêtes	12
IV-D	Diagrammes applicatifs	13
V	Application Android	15
V-A	Introduction	15
V-B	Description de l'application	15
V-C	Activités	15
V-D	Intents	20
V-E	Background service	21
V-F	Conclusion	24
VI	Interface Homme-Machine	25
VI-A	Introduction	25
VI-B	Analyse du Marché	25
VI-C	Les technologies Chrome	26
VI-D	Développement	28
VI-E	Intégration au reste du projet	31
VI-F	Conclusion	31
VII	Reconnaissance Faciale	33
VII-A	contexte	33
VII-B	Intégration de la reconnaissance faciale dans le projet	33
VII-C	Construction de la base de données d'images	35
VII-D	Apprentissage/Reconnaissance des visages	37
VII-E	Conclusion : remarques, avancement et améliorations possibles	46
VIII	Securité	47
VIII-A	Introduction au problème	47
VIII-B	RSA	47
VIII-C	Fonctions de Hachages	49
VIII-D	Bilan de la sécurité	50
IX	Conclusion	51
IX-A	point d'avancement par rapport au cdc	51
IX-B	problème rencontrés + discussion	51
IX-C	amélioration possible + discussion	51
IX-D	ressentit par rapport au projet	51

I. PRÉSENTATION DE FACE KEY

A. Situation initiale

1) *Problème des mots de passe:* De nos jours on observe une augmentation du nombre de sites qui nécessite la création d'un compte : réseaux sociaux, différent compte mail, opérateur téléphonique, journaux en ligne ... Il devient difficile d'avoir un mot de passe différent par site, qu'il soit assez complexe pour être sécurisé et qu'on n'ait pas besoin de la noter quelque part. La plupart des utilisateurs utilisent donc un seul mot de passe, souvent très simple, en dépit des problèmes de sécurité que cela pose. Leurs comptes et les informations qu'ils contiennent deviennent donc facilement hackable et comme les comptes sont souvent liés, on risque un hacking en chaîne de beaucoup de ses comptes. Une solution développée pour pallier ce problème est celle du gestionnaire de mot de passe (GMDP). Le principe est simple : l'utilisateur n'a qu'un seul mot de passe maître (plus élaboré) à connaître et c'est le GMDP qui s'occupe des combinaisons identifiant/mot de passe sécurisé pour chaque site. L'un des problèmes de cette solution est qu'il faut toujours retenir un mot de passe maître élaboré pour déverrouiller le GMDP ; ce déverrouillage pouvant prendre plus de temps. Avec cette méthode il peut aussi être fastidieux, contre intuitif et peu sécurisé de partager l'accès à un compte (partager son compteur d'opérateur téléphonique avec son conjoint par exemple).

2) *Maturisation de la technologie:* Avec l'émergence du deeplearning et d'autres technique de traitement d'image et de machine learning, les techniques de reconnaissance faciale deviennent de plus en plus performante. Certain algorithme arrive même à battre l'humain dans certaines conditions (Chaochao Lu et Xiaou Tang 2014). De plus beaucoup de grands acteurs économiques s'attaquent au problème, obtiennent d'excellents résultats et en sortent des applications, par exemple Facebook avec deepFace en 2014, Microsoft avec Hello en 2015 et enfin très récemment Apple avec FaceID (en 2017). Il a également été prouvé que la reconnaissance faciale pouvait être plus difficilement hackable que des mots de passe de par la complexité et l'unicité du visage humain.

B. Notre solution : Face Key

1) *Présentation:* Notre idée est de combiner les technologies émergentes avec un gestionnaire de mot de passe traditionnel afin de supprimer le mot de passe maître afin de rendre instantanée et sécurisée la connexion à nos sites préférés.

2) *Fonctionnement:* Notre application se présentera sous forme d'un plug-in web, il suffira de se présenter sur la page du site où l'on veut se connecter et cliquer sur le plug-in en haut à droite du navigateur. L'application va alors prendre une photo avec la webcam du l'ordinateur, l'envoyer sur l'application Face Key qui tourne sur l'ordinateur de l'utilisateur ; un algorithme va trouver le visage présent d'en l'image et l'identifier. Si il s'agit bien du visage de l'utilisateur une requête sera faite aux bases de données des serveurs Face Key afin d'obtenir les identifiants pour la page visitée. La page s'actualise avec les identifiants du compte et d'un coup d' $\frac{1}{2}$ il vous êtes connecté. Nous proposons également un système de partage de compte. L'utilisateur peut choisir de partager certains de ses comptes avec d'autres utilisateurs Face Key. Il choisira alors quel personne est autorisé à utiliser quel compte et de la même manière l'autre utilisateur pourra accéder uniquement avec son visage au compte du premier.

C. Organisation des différents projets

Le projet Face Key s'intègre dans le projet d'intégration et recoupe donc les projets de base de données, de réseau, de développement d'application mobile et potentiellement celle de système d'exploitation.

1) *Base de données:* Nous utiliserons une base de données pour stocker différentes informations et paramètres de nos utilisateurs. Par exemple les informations de connexion à la plateforme, les couples identifiant/password pour se connecter aux différents sites, différents paramètre de préférence de l'utilisateur, des données utilisateur collecter sur chaque site (heure moyenne de connexion, fréquence de connexion ...), des données sur l'utilisation de notre application(date de création, fréquence d'utilisation ...). Un des objectif est de rendre la base de données "anonyme", c'est-à-dire qu'on ne demande pas de nom, prénom à notre utilisateur, uniquement une adresse mail, afin que si quelqu'un a accès aux données que nous collectons il ne puisse pas remonter à l'identité de nos utilisateurs. Les images nécessaires pour l'apprentissage des visages ne seront pas stockées dans une base de données sql, après quelques

recherche nous pensons que cet outil n'est pas adapté. Pour plus d'informations sur la partie base de données, consultez le schéma relationnel ci joint.

2) Réseau: La partie réseau de l'application est décomposée en 3 parties :

- Le Plugin Web
- Un Client local (tournant sur la machine de l'utilisateur)
- Un Serveur Distant relié à une Base de Donnée

Le Plugin Web communiquera avec le serveur local et lui transmettra toutes les entrées de l'utilisateur. Le client local lui s'occupera de traiter toutes les demandes de l'utilisateur en effectuant les tâches nécessaires quant à la résolution du problème donné par l'utilisateur. Le client local communiquera avec le serveur distant (lui-même relié à une base de données) pour obtenir les données nécessaires à la résolution du problème. La connexion entre le client et le serveur sera assuré par le protocole TCP permettant un transport des informations fiable et en mode connecté. Les informations transitant sur le réseau seront de plusieurs natures : Images, textes, fichiers, etc ...

3) Développement d'application mobile: L'application Face Key est une application qui sera utilisée à la production du projet. Elle servira en effet à collecter les données pour l'apprentissage de nos modèles en recueillant des photos d'utilisateurs.

4) Système d'exploitation: A DEFINIR

5) Projet de synthèse: Pour la partie projet de synthèse, nous apportons chacun un plus au projet en y incorporant des domaines qui nous ont intéressés cette année :

- Louis L'Haridon : plug-in web/interface utilisateur
- Quentin Gerard : cryptographie/sécurité réseau
- Matthieu Vilain : machine learning/reconnaissance faciale

D. Nos ambitions, nos limites

Tout ce qui a été présenté précédemment est la vision idéale du projet. Dans un premier temps nous nous concentrerons sur le gestionnaire de mot de passe avec reconnaissance faciale sans implémenter le partage de compte mais en concevant l'architecture du logiciel pour que le partage soit possible. En revanche si nous arrivons à tout finir en temps et en heure, nous nous laissons libre d'ajouter des fonctionnalités comme analyse de l'émotion de l'utilisateur lorsqu'il déverrouille l'application ou autre. L'idéal initial était de pousser le projet à fond en développant un protocole d'installation, une documentation détaillée etc... afin de proposer le logiciel final en open source. Cet objectif ne pourra pas être réalisé au court du projet de L3 par manque de temps.

II. GESTION DE PROJET

A. Répartition des tâches

Ce projet entre dans le cadre du projet de synthèse de Licence 3, il englobe donc les matières de base de données, réseau, système d'exploitation et de développement Android. Nous avons décidé de travailler en commun sur la partie impliquant un cours que nous avons eu cette année. Nous avons donc tous travaillé sur la partie réseau, base de données et sur l'application Android. Nous n'avons pas avancé dans notre projet de système d'exploitation, nous n'en parlerons donc pas dans ce rapport.

Pour profiter au maximum de ce projet de fin de licence qui nous offre la possibilité de développer un logiciel complet, nous avons décidé de chacun développer une partie qui nous intéresse et dans lequel nous voudrions apprendre et prendre de l'expérience. Ainsi, en plus des parties reliées à un cours, nous avons :

- Une partie Interface homme machine qui sera faite par Louis
- Une partie reconnaissance faciale/machine learning qui sera faite par Matthieu
- Une partie Sécurité/cryptographie qui sera faite par Quentin

Cela nous permet d'ajouter à notre projet des domaines que nous n'avons pour l'instant pas vu en cours mais qui nous passionne.

Le but est de laisser le plus de liberté possible à chacun dans sa partie, qu'il puisse expérimenter et tourner son problème de la manière qu'il veut. La seule contrainte étant que le travail reste puisse être intégré dans un gestionnaire de mot de passe.

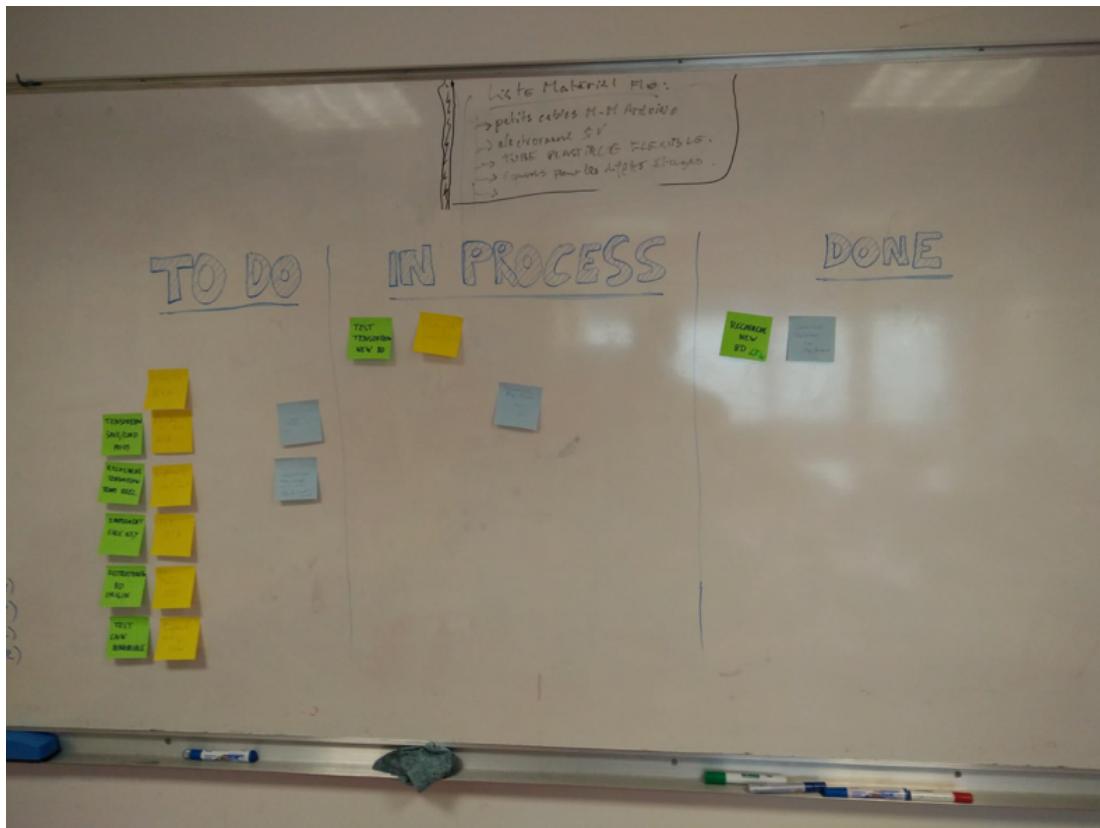
B. Les deadlines

Les dead-lines qui nous ont été fixés pour ce projet sont :

- 08/12/2017 : Rendu de BDD
- Le 12/12/2017 : Soutenance de BDD
- 08/01/2018 : Soutenance de Réseau
- 08/01/2018 : Rendu développement Android
- 16/01/2018 : Soutenance développement Android
- 09/03/2018 : Rapport projet de synthèse
- 15/03/2018 : Soutenance projet de synthèse

C. Organisation du travail

Pour l'organisation nous avons fait des réunions de travail régulières pour mettre au point les tâches que nous avions à réaliser, nous les répartir et nous donner des dead-lines. Nous avons également programmé des séances de travail pour avancer rapidement à certains moments.



Pour le code, nous nous sommes synchronisé sur GIT.

III. BASE DE DONNÉES

A. Introduction

Qui dit gestionnaire de mot de passe dit données à stocker. En effet dans ce genre de projet la Base De Données est un aspect majeur, qu'elle serve à stocker les informations basiques telles que les couples identifiant/mot de passe ou bien tout simplement les informations nécessaires au bon fonctionnement de l'application, il faut lui fournir une architecture robuste et habile. On peut également imaginer vouloir stocker et accéder à tout un tas d'information intéressantes sur les sites visités. Perdre du temps à chercher des informations dans la BDD c'est ralentir tout le fonctionnement de l'application, il faut donc être précautionneux quand à la conception de la BDD.

La BDD de Face Key est située sur le serveur comme le rappelle sur le schéma suivant (Fig. 1) :

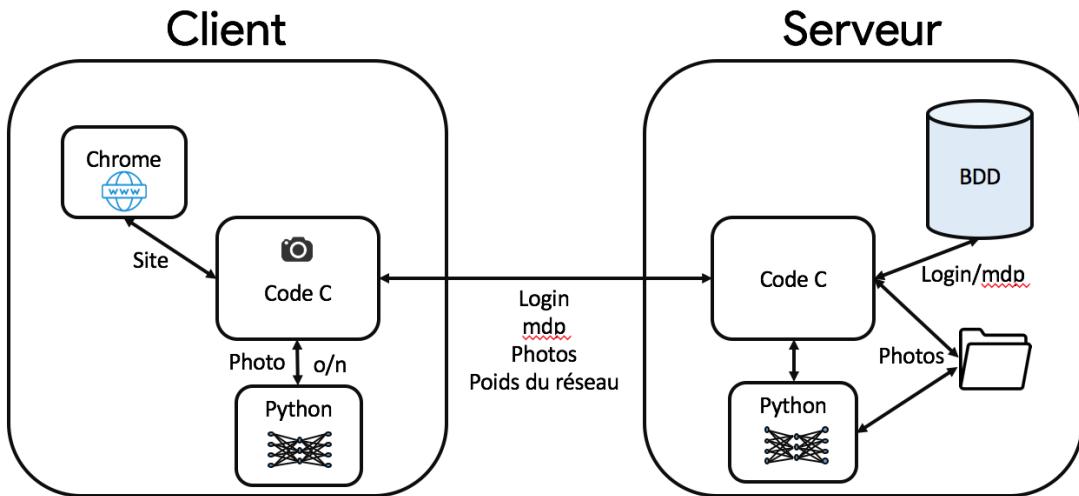


FIGURE 1. Localisation des blocs de BDD dans l'architecture de Face Key

B. Analyse du besoin

Pour répondre aux attendus du projet nous avons donc évalué les besoins d'une telle Base de Données. Au delà du simple aspect nécessaire de certaines données, nous avons eu la volonté d'étoffer notre BDD avec des statistiques sur les sites et les utilisateurs qui nous semblent pertinentes à la compréhension de l'usage fait de FaceKey. Il nous semble aussi important d'utiliser certaines données pour détecter les activités anormales sur les comptes et ainsi, augmenter la sécurité de notre application. Nous avons retenu qu'elle devait contenir les informations suivantes :

- Les informations concernant un utilisateur
 - Nom, Prénom
 - Identifiant maître / Mot de passe Maitre
 - Comptes enregistrés dans Face Key
 - Site sur lequel le compte est utilisé
 - Identifiant du compte
 - Mot de Passe du compte
 - Géolocalisation des dernières connexions
- Les informations concernant les sites utilisables dans FaceKey
 - Nom de domaine
 - Id des champs de connexions
 - Statistiques sur les utilisateurs des sites
 - Géolocalisation
 - Date de dernières connexions

- Utilisateurs inscrits
- Fréquence d'utilisation

Pour réaliser cette Base De Données nous avons donc schématisé sa conception à partir des informations ci-dessus.

C. Modèle Conceptuel des Données

Dans un premier temps nous avons décrit sous forme d'un schéma les données à traiter.. (Fig. 2)

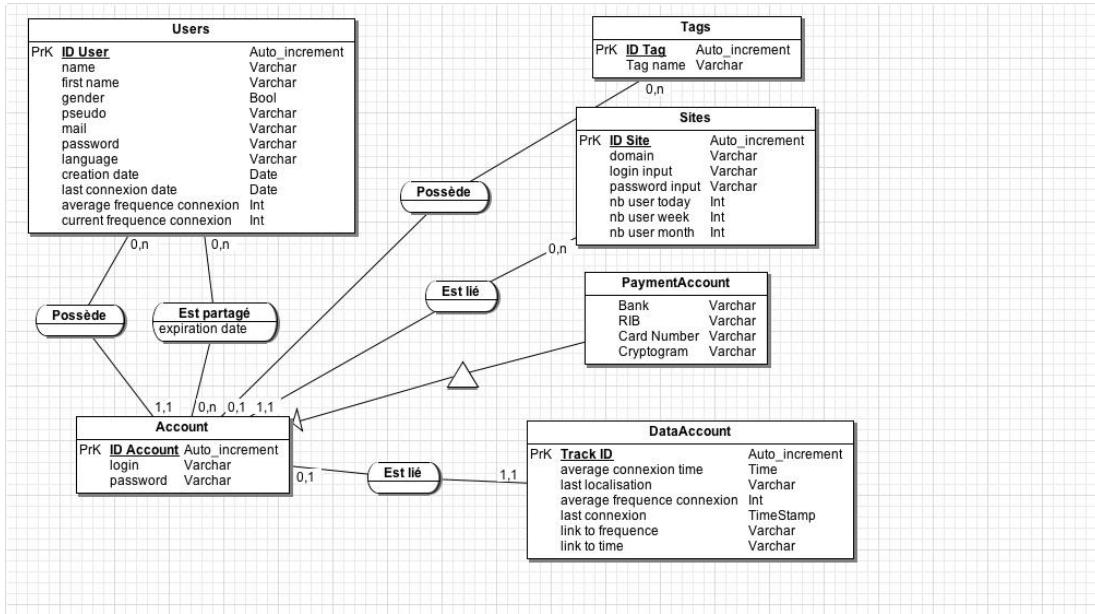


FIGURE 2. Modélisation Conceptuelle des Données

D. Modèle Logique de Données

A partir du MCD nous avons réalisé le Modèle Logique des Données qui est le formalisme adapté à l'implémentation de notre BDD en PostgreSQL. (Fig. 3)

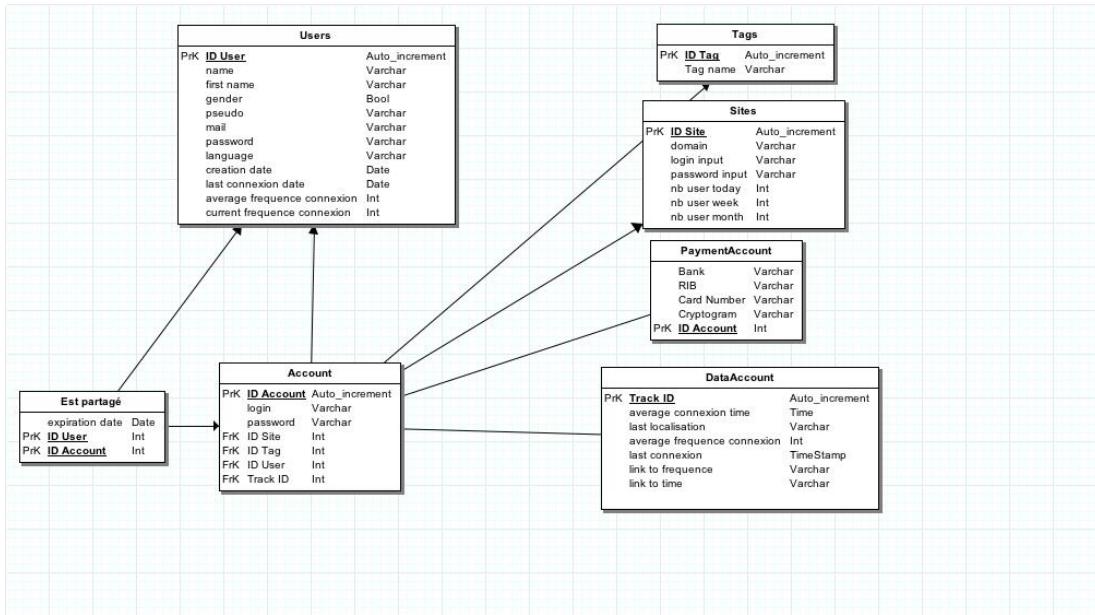


FIGURE 3. Modélisation Logique de Données

C'est grâce à ce modèle que nous créons en PostgreSQL notre Base De Données.

E. Requêtes significatives

Depuis notre BDD nous pouvons effectuer les requêtes pour accéder à toutes les informations nécessaires au bon fonctionnement de l'application. Nous notons ici quelques requêtes significatives qui représentent le fonctionnement de l'application et de l'accès aux données via la BDD.

1) Liste les comptes d'un user: Code SQL :

```
SELECT id_account, domain, login
FROM Account
  INNER JOIN Sites
    ON account.id_site = sites.id_site
WHERE account.id_user = 2
;
```

Résultat :

id_account	domain	login
6	google.com	matthieu.vilain@etu.u-cergy.fr
4	evernote.com	matthieu.vilain@etu.u-cergy.fr
5	nytimes.com	matthieu.vilain@etu.u-cergy.fr

2) Liste sharedAccount d'un user: Code SQL :

```
SELECT id_sharedAccount, domain, name, first_name
FROM SharedAccount
  INNER JOIN Account
    ON sharedAccount.id_account = account.id_account
  INNER JOIN Sites
```

```

    ON account.id_site = sites.id_site
    INNER JOIN Users
        ON account.id_user = users.id_user
    WHERE sharedAccount.id_receiver = 3
;

```

Résultat :

id_sharedaccount	domain	name	first_name
4	google.com	Louis	LHARIDON
2	spotify.com	Jérôme	VABOIS

3) Liste des positions GPS des utilisateurs d'un site: Code SQL :

```

SELECT last_loc
FROM DataAccount
    INNER JOIN Account
        ON DataAccount.id_account = Account.id_account
    INNER JOIN Sites
        ON Account.id_site = Sites.id_site
WHERE domain = 'twitter.com'
;

```

Résultat :

last_loc
46°37.5120N, 2°22.8760E
43°37.3120N, 2°24.2760E

4) Somme du temps d'utilisation et nombre utilisateurs d'un site pour calcul moyenne: Code SQL :

```

SELECT domain, SUM(average_conn_time) AS sum_time, COUNT(average_conn_time) AS nb_user
FROM Account
    INNER JOIN Sites
        ON Account.id_site = Sites.id_site
    INNER JOIN DataAccount
        ON Account.id_account = DataAccount.id_account
GROUP BY domain;

```

Résultat :

domain	sum_time	nb_user
trello.com	00:11:25	1
amazon.com	00:06:52	1
nytimes.com	02:24:34	2
apple.com	00:05:30	1
spotify.com	01:16:34	2
google.com	00:53:24	2
twitter.com	00:47:59	2
evernote.com	00:52:36	1
facebook.com	01:43:01	3

5) Dernier compte utilisé par l'utilisateur: Code SQL :

```
SELECT id_user AS user , domain AS site , last_conn AS last_connexion FROM (SELECT Account .
    id_user , Account.id_account , domain , last_conn , account.id_site FROM Account
    INNER JOIN sites
        ON account.id_site = sites.id_site
    INNER JOIN DataAccount
        ON account.id_account = DataAccount.id_account
    WHERE account.id_user = 2) AS list_conn
ORDER BY last_conn DESC
FETCH FIRST 1 ROWS ONLY;
```

Résultat :

user	site	last_connexion
2	nytimes.com	2017-12-14 12:14:20

6) Liste des comptes partagé avec un utilisateur: Code SQL :

```
SELECT id_sharedAccount , domain , name , first_name , login , id_receiver
FROM SharedAccount
INNER JOIN Account
    ON sharedAccount.id_account = account.id_account
INNER JOIN Sites
    ON account.id_site = sites.id_site
INNER JOIN Users
    ON account.id_user = users.id_user
WHERE sharedAccount.id_receiver = 2 ;
```

Résultat :

id_sharedaccount	domain	name	first_name	login	id_receiver
4	google.com	Louis	LHARIDON	louis.lharidon@etu.u-cergy.fr	2
2	spotify.com	Jérôme	VABOIS	jerome.vabois@etu.u-cergy.fr	2

F. Panel d'administration

Pour administrer notre BDD nous avons implémenté un panel d'administration en PHP. Ce dernier fonctionne sur un modèle simple : sur une page d'accueil on choisit si on veut accéder aux données relatives aux sites ou aux utilisateurs.

1) *Utilisateurs*: Dans cette partie nous allons pouvoir ajouter/supprimer un utilisateur, accéder au profil d'un utilisateur et gérer ses informations. Pour chaque utilisateur nous pouvons lister ses comptes, les supprimer/modifier ou en ajouter de nouveaux, modifier les comptes partagés avec d'autres utilisateurs et afficher les informations relatives à cet utilisateur (géolocalisation des connexions, fréquence d'utilisation...).

2) *Sites*: Dans cette partie du panel d'administration on va pour ajouter/supprimer des sites utilisables par FaceKey et modifier les informations de ceux déjà présents dans la BDD. nous allons également avoir accès à un certain nombre d'information sur l'usage de ces sites par les utilisateurs (géolocalisation représentée sur une carte, fréquence d'utilisation...).

Nous avons représenté le fonctionnement du panel sur le schéma suivant (Fig. 4).

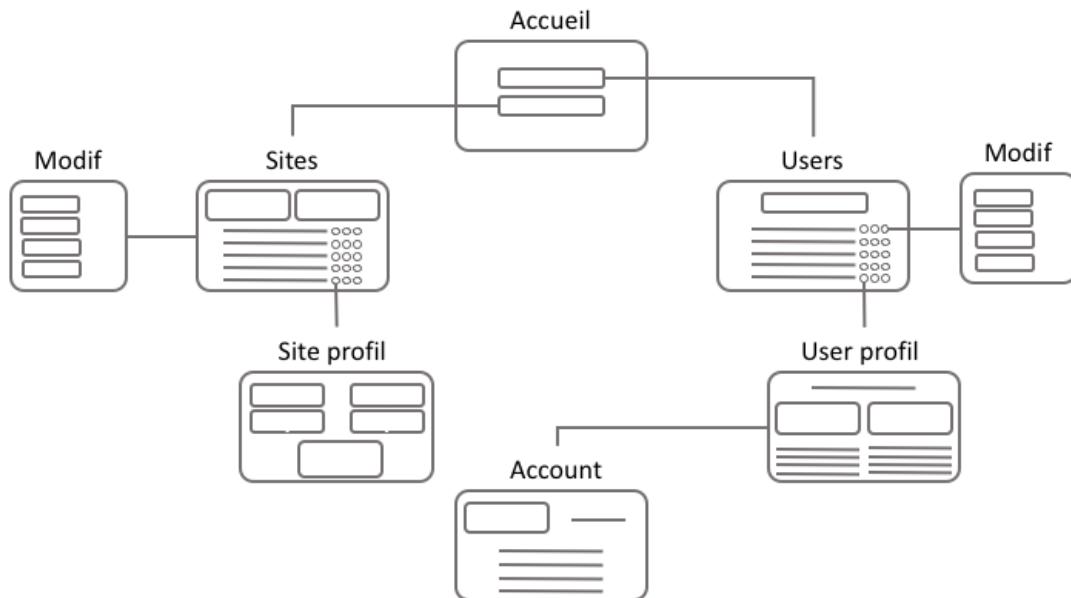


FIGURE 4. Schématisation du fonctionnement du Panel d'Administration de la BDD

G. Conclusion

Nous avons donc développé une BDD robuste et utilisable ainsi qu'un panel d'administration graphique. Nous devons maintenant penser à la sécurisation des données sur cette BDD qui sont pour la plupart sensibles (couples identifiant/mot de passe, numéro de Carte Bleue, géolocalisation des utilisateurs...). Une amélioration possible à cette BDD serait l'ajout de statistiques plus poussées qui seraient intéressantes à partir d'une masse critique d'utilisateurs.

IV. RÉSEAU

A. Spécification

Pour le projet Face Key nous avons besoin de développer un serveur qui communique avec un client. Le serveur aura pour mission d'authentifier l'utilisateur et de fournir au client les données dont il a besoin au bon fonctionnement du logiciel, dans la limite de ce qui est accessible à l'utilisateur (un utilisateur ne peut avoir accès qu'à ses données). C'est le serveur qui assurera le lien avec la Base de données.

Le serveur devra :

- Authentifier l'utilisateur
- Créer un compte
- Fournir la combinaison Identifiant/Mot de passe pour un site donné à l'utilisateur
- Recevoir les photos de l'utilisateur pour mise à jour du réseau de neurone
- Envoyer les poids du réseau de neurones à l'utilisateur

B. Fonctionnement global

Pour réaliser le Client/Server, nous avons choisi d'utilisé le protocole TCP, qui nous permet de nous assurer que chaque message a bien été transmis à l'utilisateur, et ne pouvant pas être utilisé en diffusion. Le serveur est également multi-client : un processus est créé, à partir du processus père, à chaque connexion au serveur. Voici le fonctionnement globale du Client/Serveur de Face Key :

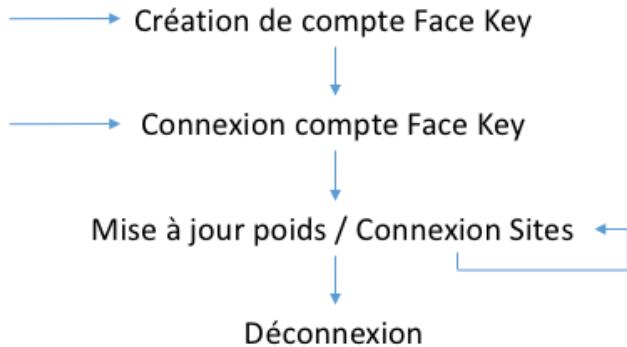


FIGURE 5. Fonctionnement globale de Face Key

C. Les requêtes

Une requête est constitué de 2 parties :

- Le code de la requête (voir tableau ci-dessous)
- Les informations

Dans la requête, les informations sont séparées du code de la requête par un ";". Ainsi la requête est composé de la manière suivante :

 |

. Les informations sont quant à elles séparé par une virgule, nous avons donc par exemple :

 |

. Cela donne en pratique "100;toto,facebook.com".

Liste des codes :

Code Diagramme (000-099)	000	OK
	001	Code pour démarrer le diagramme de connexion à un site web
	002	Code pour démarrer le diagramme de création d'un compte
	003	Code pour démarrer le diagramme de mise à jour du réseau de neurones
Client -> Serveur (100-149)	100	Demande la liste des IDs disponibles pour un utilisateur sur un site à l'instant t
	101	Demande le mot de passe pour un ID (accessible par l'utilisateur) sur un site
	102	Envoie d'une photo de l'utilisateur ainsi que ses coordonnées GPS
	103	Envoie de l'identifiant et du mot de passe Face Key pour authentification
	110	Envoie de l'email et du pseudo pour création du compte Face Key
	111	Envoie du mot de passe pour création du compte Face Key
	112	Envoie des informations de l'utilisateur pour création du compte Face Key (Genre, Nom, Prénom et Langue)
	113	Signal le serveur de l'envoi d'une photo
	114	Signal le serveur de l'envoi de la dernière photo
	136	Demande l'envoi du fichier des poids du réseau de neurones aux serveurs
Serveur -> Client (200-249)	200	Envoie la liste des IDs disponibles pour un utilisateur sur un site à l'instant t
	201	Envoie le mot de passe pour un ID (accessible par l'utilisateur) sur un site
Erreurs côté serveur (400-499)	400	Requête inconnue
	401	Il manque des informations dans la requête reçue
	402	Utilisateur introuvable lors de la phase d'authentification : Mauvais Identifiants
	403	Utilisateur introuvable lors de la phase d'authentification : Mauvais mot de passe
	405	Timeout Reached : l'utilisateur (ou le serveur) a mis trop de temps à répondre
	406	La requête reçue n'est pas pris en charge à ce point de l'application
	407	Aucun compte n'a été trouvé sur le domaine envoyé par l'utilisateur
	408	Le compte demandé pour le domaine n'est pas dans la liste des comptes accessibles par l'utilisateur
	409	Création du compte : l'email est déjà utilisé
	410	Création du compte : le pseudo est déjà utilisé

D. Diagrammes applicatifs

Dans cette partie nous allons présenter les différents diagrammes applicatifs du Client/Server de Face Key.

1) *Création d'un compte:* Pour démarrer ce diagramme il faut préalablement envoyer au serveur le code diagramme correspondant (code 002 pour la création de compte).

Ici les photos et les poids du réseau de neurone sont envoyés sous forme de fichiers. Les fichiers sont ouverts par les deux programmes (client et serveur) et envoyés (et reçus) octet par octet sans code en tête. Le transfert d'un fichier est donc décomposé en plusieurs parties : tout d'abord les deux premiers messages envoient le nom du fichier ainsi que le nombre d'octets qui va être envoyé (la taille du fichier), ensuite les octets sont envoyés un par un. Le receveur compte le nombre d'octets reçus et s'arrête une fois que l'intégralité des octets ont bien été reçus.

2) *Phase d'authentification:* Ici, le client envoie son identifiant et mot de passe Face Key, le serveur lui répondra par un code "OK" si la combinaison est correcte, ou par une erreur spécifiant quel partie de la combinaison est erronée.

Création de compte

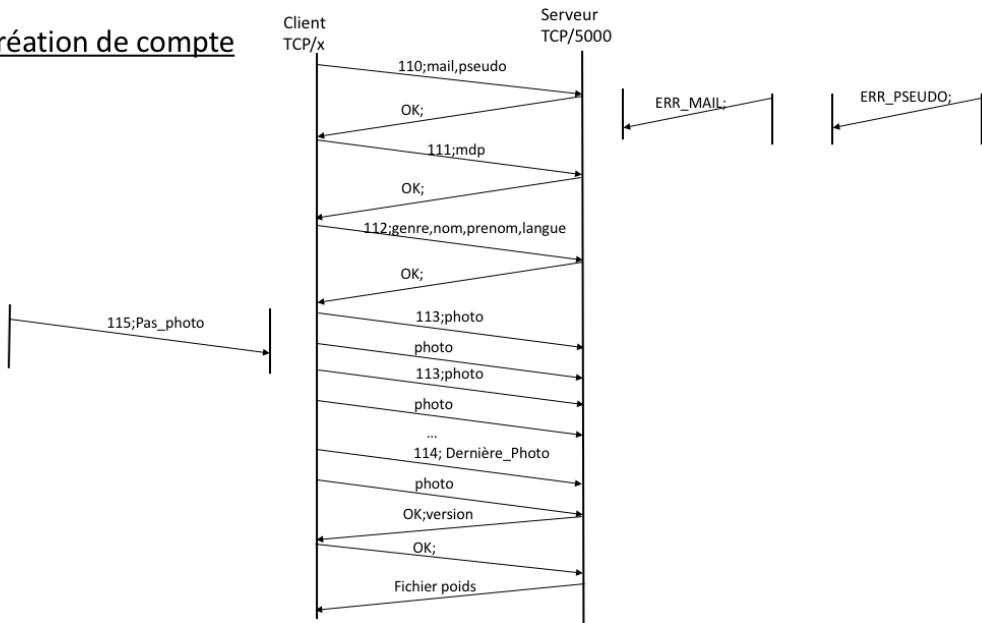


FIGURE 6. Crédation d'un compte

Connexion Application

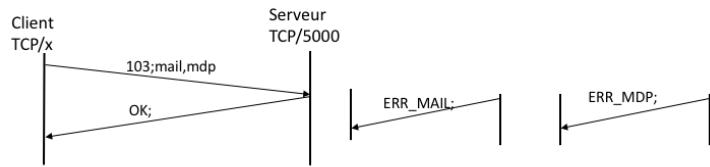


FIGURE 7. Authentification au compte Face Key

3) Connexion à un site: Pour démarrer ce diagramme il faut préalablement envoyer le code diagramme (code 001 pour la connexion à un site) au serveur **et** effectué une phase d'authentification (code 103).

Connexion Site

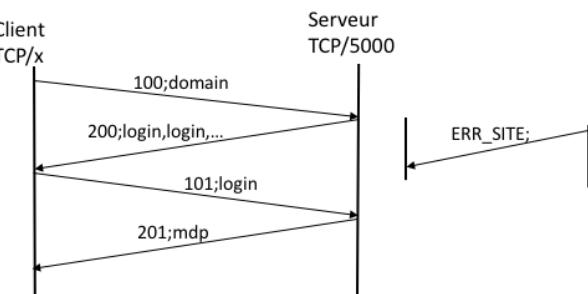


FIGURE 8. Connexion à un site web

4) Mise à jour des poids du réseau de neurones: Pour démarrer ce diagramme il faut préalablement envoyer le code diagramme (code 003) au serveur **et** effectué une phase d'authentification (code 103).
Tout comme dans le diagramme de création d'un compte, les poids du réseau de neurones sont envoyés de la même manière : sous forme de fichier avec une transmission octet par octet.

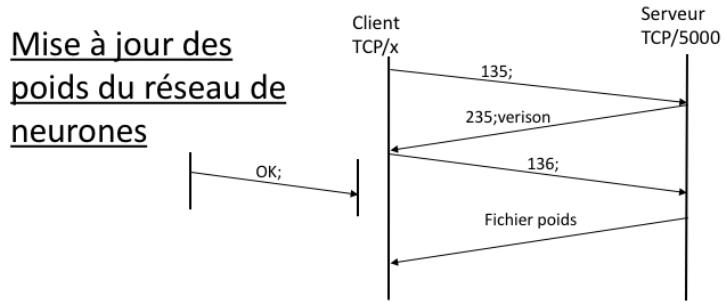


FIGURE 9. Mise à jour des poids du réseau de neurones

V. APPLICATION ANDROID

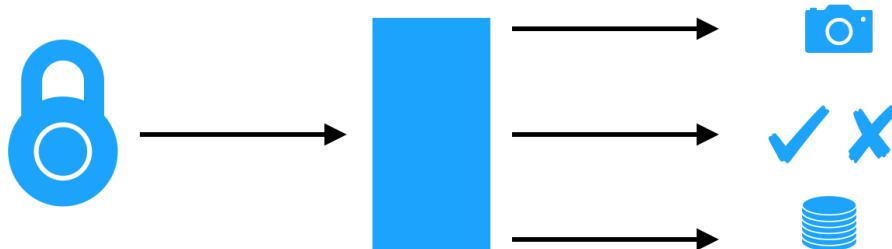
A. Introduction

Pour le projet FaceKey nous avons besoin d'un grand jeu de données pour entraîner nos modèles de reconnaissance faciale. Pour augmenter ce jeu de données nous en avons besoin d'un processus plus orienté utilisateur. Nous avons donc décidé de créer une application Android qui va prendre des photos d'un utilisateur et les sauvegarder sur notre Base de Données pour pouvoir entraîner nos modèles. Nous voulons donc une application qui propose à l'utilisateur de se connecter à son compte Face Key et lui propose de prendre des photos de lui. Après vérification des photos par l'utilisateur, l'application lui propose de télécharger les photos sur notre Base De Données.

B. Description de l'application

Le modèle choisi pour l'application est donc simple. Sur un premier écran on peut se connecter, ce qui nous amène à un second écran où on peut choisir entre trois options :

- Prendre une photo
- Trier les photos
- Uploader vers la BDD mySQL



C. Activités

L'application contient 3 activités différentes.

- Activité de connexion
- Activité de Choix
- Activité de tri des photos

1) *Login Activity:* Il s'agit de la première activité que nous avons codé. Elle contient 2 text input, une pour le login et une autre pour le mot de passe. En cliquant sur le bouton et en ayant le couple login/mot de passe correct on peut accéder à la seconde activité.

Dans Facekey, nous utilisons PostgreSQL pour stocker les informations des utilisateurs. Comme la Base De Données est sur un ordinateur isolé par le réseau de l'Université, nous ne pouvons pas y accéder depuis un téléphone android. Nous allons donc juste stocker dans un simple tableau à deux dimensions des couples login/mdp pour tester l'application.

Si nous pouvions accéder à la BDD depuis le smartphone nous aurions placé un fichier PHP sur un serveur web qui, connecté à la BDD, aurait retourné vérifié via une http request l'exactitude du couple login/mdp testé.

Ici nous utiliserons simplement le tableau à 2 dimensions.

```
public static String [][] login = {{"admin","",""}, {"Login","p@ssw0rd"}, {"louis.lharidon@etu.u-  
cergy.fr", "azerty"}};
```

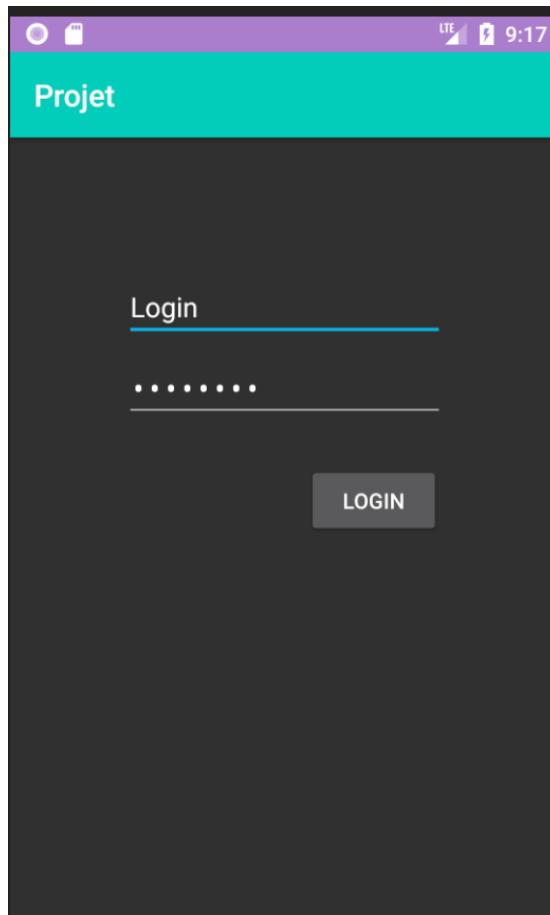
Et, en cliquant sur le bouton soumettre on vérifie si le couple login/mdp correspond à un autre couple présent dans le tableau. present on the array.

```
for (String [] s : login) {  
    if ((name.equals(s[0])) && (pass.equals(s[1]))) {  
        // login ok : starting intent  
    }  
}
```

La méthode aurait été similaire avec PostgreSQL mais, au lieu de créer un tableau, on aurait vérifier la présence du couple dans la Base De Données via l'httpprequest.

2) *Choice Activity*: Cette activité contient simplement 3 boutons.

- Un qui lancera la caméra
- Un autre qui lancera la troisième activité, celle du tri de photos
- Une dernière qui lancera le service qui uploadera les photos sur la Base de Données MySQL



3) Activité de tri de photos: Cette activité est la plus complexe. Elle est composée de deux éléments : une gridview et une checkbox. La gridview est remplie d'imageviews. Pour faire cela, nous avons du créer un ImageAdapter. Cet adaptateur va simplement prendre une image au format bitmap, la redimensionner pour la faire rentrer dans l'imageview et la placer dans le gridview tout en gardant sa position et son id en mémoire.



L'imageAdapter calcule les proportions pour que les imageview remplissent chacune de ses view, les stocke dans un tableau et autorise certaines opérations.

```
Storing views into array
ArrayList<String> itemList = new ArrayList<>();
```

Il y a trois fonctions intéressantes dans l'imageAdapter : decodeSampledBitmapFromUri qui va retourner un bitmap depuis un chemin et un couple hauteur/largeur.

```
Bitmap decodeSampledBitmapFromUri(String path, int reqWidth, int reqHeight) {
    Bitmap bm ;
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, options);
    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
    // Decode bitmap with inSampleSize
    options.inJustDecodeBounds = false;
    bm = BitmapFactory.decodeFile(path, options);
    return bm;
}
```

Une autre qui va calculer le couple hauteur/largeur requis.

```

int calculateInSampleSize(BitmapFactory.Options options, int reqWidth, int reqHeight) {
    // height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;
    if (height > reqHeight || width > reqWidth) {
        if (width > height) {
            inSampleSize = Math.round((float)height / (float)reqHeight);
        } else {
            inSampleSize = Math.round((float)width / (float)reqWidth);
        }
    }
    return inSampleSize;
}

```

Enfin nous avons une fonction add qui va ajouter une image à la gridview via son chemin.

```

void add(String path){
    itemList.add(path);
}

```

Nous avons besoin de onCreate pour déclarer les imageview en tant qu'adaptateurs du gridview.

```

final GridView gridview = findViewById(R.id.gridView);
myImageAdapter = new ImageAdapter(this);
gridview.setAdapter(myImageAdapter);

```

Pour ajouter toutes les images depuis un dossier nous avons besoin de lister tous les fichiers d'un dossier. La première étape est l'accès au dossier avec les lignes suivantes :

```

String root = Environment.getExternalStorageDirectory().getAbsolutePath();
File myDir = new File(root + "/saved_images");

```

Ensuite nous pouvons lister tous les fichiers, y accéder et les ajouter au gridview via notre imageAdapter.

```

files = myDir.listFiles();
for(File f : files){
    // add to image adapter the file
    myImageAdapter.add(f.getAbsolutePath());
}

```

Pour gérer les fichiers avant l'envoi sur le serveur nous avons implémenté une suppression des fichiers. L'utilisateur peut être amené à décider si les photos prises sont bonnes à l'envoi. Nous avons placé une checkbox pour qu'il puisse indiquer si il veut supprimer une image et un onclicklistener pour qu'il appuie simplement sur l'image à supprimer.

```

gridview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View v,

```

```

        int position, long id) {
    // if checkbox is enabled
    if (ch.isChecked()) {
        // getting file from position
        File item = files[position];
        // deleting and checking deleting
        boolean deleted = item.delete();
        Log.d("Files", "Removed " + deleted);
        // reload page to delete from the grid
        reload();
    }
});
```

La fonction reload rafraîchit l'UI en redémarrant l'activité.

```

// reload function, simply reload activity
public void reload(){
    finish();
    startActivity(getIntent());
}
```

D. Intents

Pour cette application nous utilisons une intent externe, celle de la caméra.

1) *Camera intent:* Cette intent est ouverte depuis l'activité de choix. en utilisant les permissions nous pouvons accéder à la caméra. En ouvrant cette intent, l'application demander la permission à l'utilisateur d'utiliser la caméra et de gérer les fichiers du téléphone et, après acceptation, ouvrira la caméra. L'utilisateur va ensuite prendre une photo, l'intent lui demandera sa confirmation et, ensuite, transférera la photo vers l'activité ou elle sera convertie en png et sauvegardée dans un dossier spécifique.

Pour démarrer l'intent de la caméra nous avons donc besoin de déclarer nos permissions (nous avons aussi besoin de permissions pour nos autres activités et intent mais nous expliquerons ici comment gérer les permissions pour utiliser la caméra).

Pour déclarer une activité nous avons tout d'abord besoin d'ajouter la use-permission spécifique dans le manifeste android.

```
<uses-permission android:name="android.permission.CAMERA" />
```

Ensuite, dans l'activité ou la permission est requise, nous ajoutons les lignes suivante à la fonction oncreate pour vérifier si la permission est bien accordée et, si ce n'est pas le cas, demander la permission.

```

// CAMERA
if( (ContextCompat.checkSelfPermission(SecondActivity.this, Manifest.permission.CAMERA)
!= PackageManager.PERMISSION_GRANTED)) {
    if (ActivityCompat.shouldShowRequestPermissionRationale(SecondActivity.this,
        Manifest.permission.CAMERA)) {
        ActivityCompat.requestPermissions(SecondActivity.this, new String []{ Manifest.
            permission.CAMERA}, PERMISSION_REQUEST_CODE);
    } else {
        ActivityCompat.requestPermissions(SecondActivity.this, new String []{ Manifest.
            permission.CAMERA}, PERMISSION_REQUEST_CODE);
    }
}
```

```
}
```

Ensuite nous lançons l'intent caméra avec les lignes suivantes :

```
Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(cameraIntent, CAMERA_REQUEST);
```

Après que l'utilisateur est pris une photo nous avons besoin de gérer le résultat de l'activité caméra pour stocker l'image dans un dossier. Pour faire cela nous allons utiliser la fonction onActivityResult en vérifiant si nous obtenons bien le résultat de la caméra et, ensuite, en stockant le bitmap résultant compressé au format png dans le dossier correct avec un nom de fichier unique.

```
if (requestCode == CAMERA_REQUEST && resultCode == this.RESULT_OK) {
    // getting photo bitmap
    Bitmap photo = (Bitmap) data.getExtras().get("data");
    FileOutputStream out = null;
    try {
        // Getting custom picture folder
        String root = Environment.getExternalStorageDirectory().getAbsolutePath();
        File myDir = new File(root + "/saved_images");
        if (!myDir.exists()) myDir.mkdir();
        Log.d("Photos", "directory " + myDir.toString());
        // Writing filename
        String fname = "img-" + System.currentTimeMillis() + ".png";
        // Creating file
        File file = new File(myDir, fname);
        Log.d("Photos", "file " + file.toString());
        try {
            out = new FileOutputStream(file);
            if (photo != null) {
                // putting bitmap photo to file after png compression
                photo.compress(Bitmap.CompressFormat.PNG, 100, out);
            }
            out.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

E. Background service

Dans notre seconde activité nous lançons un service en arrière plan qui va uploader toutes les photos d'un dossier spécifique vers une Base De Données MySQL. Pour faire cela nous avons besoin de lister toutes les images d'un dossier et, pour chacune d'entre elles, les convertir en bitmap et les uploader sur notre Base de Données.

```
String root = Environment.getExternalStorageDirectory().getAbsolutePath();
File myDir = new File(root + "/saved_images");
// Listing files
File[] filelist = myDir.listFiles(IMAGE_FILTER);
// browsing files
int i = 0;
for (File f : filelist) {
    // Sleeping 0.01s between each files
    try {
        Thread.sleep(10);
    } catch (Exception e) {
        Log.e("Error", "exception");
    }
}
```

```

if (isRunning) {
    Log.d("Files", "FileName:" + f.getName());
    // getting file path
    String filePath = f.getPath();
    // converting to bitmap
    bitmap = BitmapFactory.decodeFile(filePath);
    // starting upload fonction
    uploadImage();
}

```

Pour l'upload, nous créons une fonction qui convertit les images en base64 et qui lance une httprequest contenant une requête POST vers une page PHP stockées sur un serveur web. Pour cela nous utilisons cette fonction de conversion bitmap vers base64 :

```

// Converting bitmap to base64 function
public String getStringImage(Bitmap bmp){
    // new array
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    bmp.compress(Bitmap.CompressFormat.JPEG, 100, baos);
    byte[] imageBytes = baos.toByteArray();
    // return converted to base64 array
    return Base64.encodeToString(imageBytes, Base64.DEFAULT);
}

```

Et cette fonction qui effectuera la requête POST :

```

public void POST(String... params){
    String urlString = params[0]; // URL
    String data = params[1]; // data POST
    try {
        // Opening connexion
        URL url = new URL(urlString);
        // URL
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000);
        conn.setConnectTimeout(15000);
        // POST METHOD
        conn.setRequestMethod("POST");
        conn.setDoInput(true);
        conn.setDoOutput(true);
        // Building request with url and data
        Uri.Builder builder = new Uri.Builder().appendQueryParameter("image", data);
        Log.d("Upload", "POST : " + builder.toString());
        String query = builder.build().getEncodedQuery();
        // outputstream on connexion
        OutputStream os = conn.getOutputStream();
        // Buffered writer to write on outputsteam
        BufferedWriter writer = new BufferedWriter(
            new OutputStreamWriter(os, "UTF-8"));
        writer.write(query);
        writer.flush();
        writer.close();
        os.close();
        Log.d("Upload", "url : " + conn.getURL().toString());
        // Request
        conn.connect();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

```

Qui sont toutes deux combinées dans une fonction uploadImage :

```

//upload image fonction
private void uploadImage() {
    Log.d("Upload", "uploading");
    // converting to base64
    String uploadImage = getStringImage(bitmap);
    // Posting to PHP
    POST(UPLOAD_URL, uploadImage);
}

```

Ce script PHP va prendre le fichier base64 et l'insérer dans une table mySQL sur une colonne de type blob :

```

$conn = new mysqli(HOST,USER,PASS,DB) ;
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$image = $_POST['image'];
$sql = "INSERT INTO photo (image) VALUES ('$image')";
if ($conn->query($sql) == TRUE) {
    echo "New image successfully inserted";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();

```

Pour notifier l'utilisateur de l'avancement de la mise ligne des photos nous utiliserons un Toast. Mais, comme nous sommes dans un service, nous ne pouvons pas simplement utiliser un Toast. Nous avons besoin de récupérer le contexte du Thread principal pour y placer le Toast.

```

private Context applicationContext;
// Function to toast on main thread
void showToast(final String toShow){
    // checking app context
    if(null != applicationContext){
        // geting main theard
        Handler handler = new Handler(Looper.getMainLooper());
        handler.post(new Runnable() {
            @Override
            public void run() {
                // threading
                Toast.makeText(applicationContext, toShow, Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

Ensuite, via cette fonction, nous allons simplement faire apparaître un Toast sur l'écran de l'utilisateur :

```

String toToast = "Uploading" + (i+1) + "/" + filelist.length + "...";
showToast(toToast);

```

Après avoir téléchargé une image sur la BDD nous la retirons du dossier en la supprimant : After having uploaded an image we remove it from the folder by deletion :

```
boolean del = f.delete();
```

Après avoir téléchargé toutes les images sur la BDD, le service s'arrête.

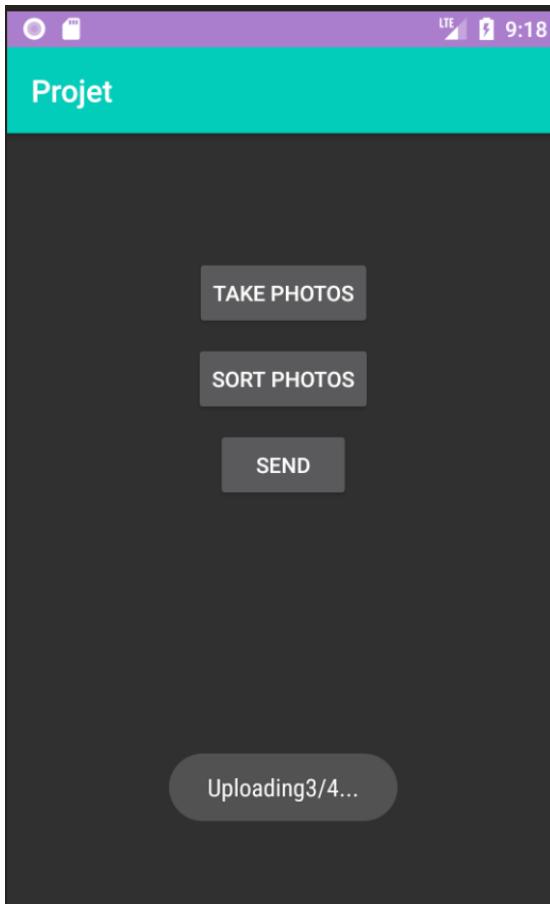


FIGURE 10. Envoi de photos au serveur

SELECT * FROM `photo`	
<input type="checkbox"/> Tout afficher	Nombre de lignes : 25
Filtrer les lignes: Chercher dans cette table	
+ Options	
<input type="checkbox"/>	id image
<input type="checkbox"/> Modifier	Copier
<input type="checkbox"/> Effacer	0 [BLOB - 4 KiB]

FIGURE 11. Réception des photos sur la BDD

F. Conclusion

En conclusion, nous avons développé une application robuste qui effectue différentes actions : une activité de connexion, un service qui effectue une httprequest vers un script php, une activité qui affiche les images d'un dossier et permet de les supprimer et nous avons utilisé la caméra via une intent. Pour améliorer cette application nous aurions pu développer notre propre activité pour prendre des photos, ce qui nous aurait permis de prendre en rafale les photos de l'utilisateur. Nous aurions pu améliorer l'UI de l'activité de tri de photos. Le principal défaut de cette application est qu'elle n'est pas connectée aux autres parties de notre projet comme nous n'avons pas la possibilité de télécharger directement les images vers notre base de données. En effet le réseau de l'Université bloque les accès extérieurs aux machines et empêche les machines sur le même réseau de s'accéder. Cette mesure de sécurité nous empêche d'intégrer pleinement notre application android dans notre écosystème et nos données FaceKey

VI. INTERFACE HOMME-MACHINE

A. Introduction

Dans le cadre d'un gestionnaire de mot de passe comme Face Key l'IHM est une composante primordiale. Discrète, elle doit savoir se faire oublier mais doit rester facilement accessible quand on en a besoin. L'expérience utilisateur doit donc être la plus simple possible. Afin de rendre cette expérience optimale nous isolons les besoins d'une telle IHM :

- Accessibilité au plus grand public
- Facilité d'utilisation
- Se connecter à Face Key (via la reconnaissance faciale)
- Se connecter à un site via Face Key
- Ajouter un compte à Face Key

Notre IHM touche donc deux composantes de notre architecture : la partie navigateur et la partie client C comme on peut le voir sur le schéma suivant (Fig. 12).

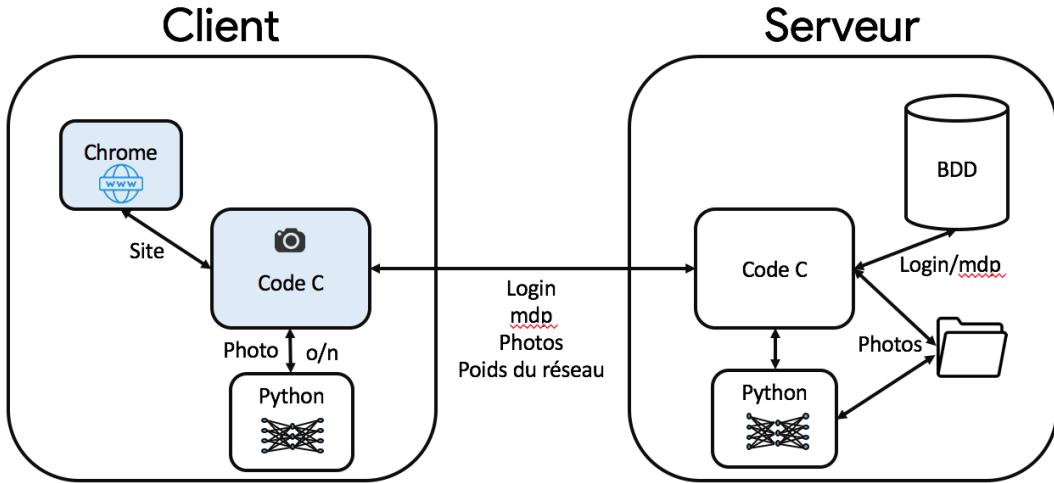


FIGURE 12. Localisation des blocs du plug-in dans l'architecture de Face Key

La connexion à Face Key, l'ajout de compte et la récupération des informations de connexions sont déjà implémentés dans notre client. Il nous faut donc une IHM capable d'interagir avec un navigateur.

Nous faisons donc logiquement le choix de développer un plug-in pour navigateur permettant de se connecter aux sites depuis le ce dernier. Nous avons donc établi le cahier des charges suivant pour la réalisation de ce plug-in :

- Il doit être accessible depuis un navigateur grand public
- Il pourra communiquer avec le client présent sur l'ordinateur
- Il doit pouvoir interagir avec le contenu d'une page web affiché dans un navigateur.

B. Analyse du Marché

Pour répondre au premier besoin : " Il doit être accessible depuis un navigateur grand public " nous avons analysé les parts de marchés des navigateurs web (Table. I). De cette analyse nous tirons la conclusion qu'il faut nous concentrer sur les 2 principaux navigateurs ayant assez de part de marché : Google Chrome et Apple Safari.

TABLE I
LES PARTS DE MARCHÉ DES NAVIGATEURS WEB DANS LE MONDE, TOUTES PLATES-FORMES CONFONDUES (FÉVRIER 2018 - W3COUNTER)

Chrome	Safari	Firefox	IE + Edge	Opera	UC Browser	Android	Autres
59,9 %	15,7 %	8,5 %	7,3 %	3,4 %	1,5 %	0,0 %	3,7 %

Nous devons ensuite nous concentrer sur les opportunités qu’offrent ces deux navigateurs pour le développement d’un plug-in. Nous savons que les deux offrent cette possibilité mais nous devons étudier si les API des deux permettent de répondre aux deux besoins techniques suivants “pouvoir communiquer avec le client présent sur l’ordinateur” et “pouvoir interagir avec le contenu d’une page web affiché dans un navigateur”. Pour cela nous isolons 3 fonctionnalités indispensables à implémenter :

- Interaction avec le contenu d’une page web.
- Récupération du contenu (pour les champs de connexion par exemple).
- Manipulation du contenu (injection dans la page du couple identifiant/mot de passe par exemple).
- Interaction avec le client : ici on choisira l’UDP pour sa facilité d’utilisation et la rapidité de la transmission des messages.

En comparant les possibilités offertes par les interfaces de programmation (API) des extensions de ces deux navigateurs nous voyons que seul un des deux répond à nos besoins (Table. VII-D4b).

TABLE II
COMPARAISON DES FONCTIONNALITÉS UTILISABLES SELON LE NAVIGATEUR

Technologie	Chrome extension/application	Safari extension
Récupération du contenu d’une page	✓	✓
Injection de contenu dans la page	✓	✓
UDP	✓	X

Nous choisissons donc logiquement de développer notre plug-in sur Google Chrome. Avant de passer au développement intrinsèque du plug-in nous devons étudier les possibilités de développement offertes par Google Chrome.

C. Les technologies Chrome

Le navigateur Google Chrome permet, via des API riches, de développer des plug-in sous différentes formes. Il propose notamment deux types de web applications :

- Les Applications
- Les Extensions

C’est en analysant ces deux types de web-application que nous choisirons sur lequel implémenter notre IHM.

1) *Application chrome*: Les applications chrome sont des web-applications développées en HTML5, CSS et javascript et dont l’objectif est de se créer une expérience d’une application native.

L’avantage majeur des applications Chrome, en comparaison avec des applications natives ou avec des web-applications, est qu’elles sont multi-plateformes. En effet, une fois développée, une application chrome fonctionnera aussi bien sur n’importe quel système d’exploitation équipé d’un navigateur chrome (Windows, OSX, Linux, Chromium et même iOS et Android via une pré-compilation).

Les applications Chrome, bien que démarrées depuis l’application Chrome (ou le terminal) fonctionnent indépendamment de chrome. Elles ont un processus propre. Ces applications peuvent fonctionner en arrière plan ou en mode fenêtré. En mode fenêtré, elles n’ont pas de barre d’adresses, contrairement au navigateur chrome, et leur fonctionnement en terme d’interface est le même que celui d’une application native.

Le plus important est qu’elles ont accès à toutes les composantes matériel de l’ordinateur hôte. Nous pouvons donc utiliser le bluetooth, les ports USB et, ce qui nous intéresse ici, la carte réseau pour l’UDP.

On peut résumer leur fonctionnement sur le schéma suivant (Fig. 13). Ici on crée une application chrome qui contient un fichier main.html et un fichier app.js. L'application ne peut pas interagir avec le contenu de chrome, elle fonctionne de manière indépendante, mais peut accéder aux composantes matériel du système hôte comme le bluetooth ou l'USB.

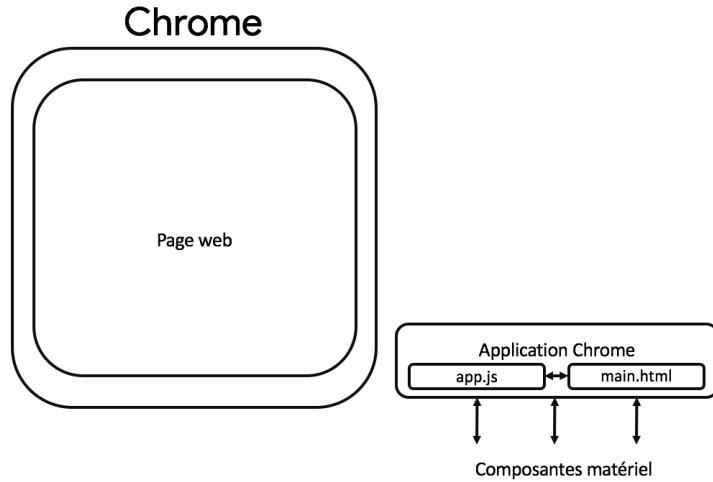


FIGURE 13. Schématisation du fonctionnement d'une application Chrome

Nous devons donc utiliser une application Chrome pour la communication entre l'IHM et le client C, mais, en raison de leur fonctionnement indépendant du navigateur nous ne pouvons pas interagir avec les sites visités par l'utilisateur.

2) Extension chrome: Les extensions chrome sont, quand à elles, des petits logiciels qui permettent de personnaliser l'expérience utilisateur du navigateur Chrome.

Basées sur les technologies web HTML, JavaScript et CSS les extensions doivent servir à un seul objectif bien défini et facile à comprendre. Elles peuvent inclure de multiples composants et avoir une série de fonctionnalités, tant qu'elles servent un objectif clairement défini.

A l'instar des applications Chrome, les extensions fonctionnent à l'intérieur du navigateur Chrome. Elles sont accessibles depuis une icône la barre d'utilisateur en haut à droite de la fenêtre chrome. Tout comme les applications Chrome, les extensions ne sont pas dépendantes du contenu du navigateur et ne nécessitent pas une connexion internet.

Les extensions chrome, contrairement aux applications Chrome, ne permettent pas d'accéder aux composantes matériel du système hôte. Elles ont cependant un accès complet au contenu des pages visitées par l'utilisateur sur le navigateur.

Le fonctionnement de ces extensions peut être résumé par le schéma suivant (Fig. 16). Ici on a une extension composée d'un fichier popup.html, d'un fichier contentscript.js et d'un fichier extension.js et une page web composée de fichiers html, js et css par exemple. Notre extension fonctionne de manière isolée via les fichiers popup et extension et peuvent interagir avec la page web via un troisième fichier contentscript qui agira sur le contenu de cette dernière.

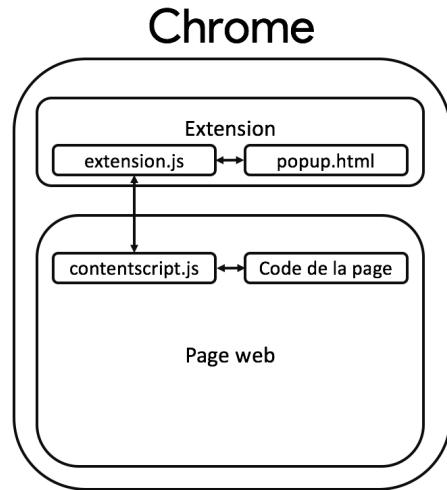


FIGURE 14. Schématisation du fonctionnement d'une extension Chrome

Nous devons donc utiliser une extension chrome pour l'interaction avec les sites visités par l'utilisateur de Face Key.

Nous venons d'identifier que nous avons besoin des deux types de web-application proposées par l'API de Chrome. Ce qui soulève le problème de la communication entre ces composantes. Il existe dans l'API des outils pour établir un système de communication entre les extensions les applications Chrome.

D. Développement

Après analyse des deux technologies offertes par chrome nous avons donc choisi de décomposer notre IHM en deux parties :

- Une extension pour interagir avec le contenu des pages web
- Une application pour interagir avec le client

Nous relierons les deux composantes via une des fonctionnalités proposées via l'API de développement Google Chrome qui s'appelle Messaging.

1) Application: Dans un premier temps nous développons la première partie de notre IHM qui sera l'application. Cette application aura pour objectif d'être le relai entre le client C et l'extension qui interagira avec le contenu des sites visité.

Une interface n'est pas nécessaire pour cette application puisque l'utilisateur n'aura jamais besoin de la manipuler, nous créons donc cette application pour un fonctionnement en arrière plan.

La réelle difficulté technique du développement de cet IHM se situe à ce niveau, comment établir une connexion UDP depuis une web-application ? Il existe deux méthodes qui demandent différents niveaux d'implémentation de notre part : la première demande d'utiliser Chromium, la version Open Source de Google Chrome et de créer des exceptions dans le code de Chromium pour nous ouvrir l'accès direct du navigateur, la deuxième méthode est plus simple. L'API suffisamment riche de chrome fournit la possibilité d'ouvrir des sockets udp et, à partir de ces derniers, envoyer et recevoir des messages.

Tout comme sur Android, nous devons déclarer que nous utilisons l'udp dans les permissions de l'application. Ici on autorisera toutes les connexions depuis toutes les connexions, même s'il est possible de limiter celles ci en remplaçant * par une liste dans le manifeste de l'application.

```

"sockets": {
  "udp": {
    "send": [ "*" ],
    "bind": [ "*" ]
  }
}

```

Sur le code suivant on observe un exemple de création de socket udp et d'envoi de données :

```

// Creation of the socket
chrome.sockets.udp.create({}, function(socketInfo) {
  // Socket created, sending some data
  var socketudp = socketInfo.socketudp;
  chrome.sockets.udp.send(socketudp, arrayBuffer,
    '127.0.0.1', 3001, function(sendInfo) {
      console.log("sent " + sendInfo.bytesSent);
    });
});

```

Sur le code suivant on peut observer un exemple de d'écoute de données via un socket udp :

```

var socketudp;
// Handle the "onReceive" event to wait for data on the socket
var onReceive = function(info) {
  if (info.socketudp !== socketudp)
    return;
  //display data
  console.log(info.data);
};

// Creation of the socket
chrome.sockets.udp.create({}, function(socketInfo) {
  socketudp = socketInfo.socketudp;
  // Setup event handler and bind socket.
  chrome.sockets.udp.onReceive.addListener(onReceive);
  chrome.sockets.udp.bind(socketudp,
    "0.0.0.0", 0, function(result) {
      if (result < 0) {
        console.log("Error binding socket.");
        return;
      }
      chrome.sockets.udp.send(socketudp, arrayBuffer,
        '127.0.0.1', 3001, function(sendInfo) {
          console.log("sent " + sendInfo.bytesSent);
        });
    });
});

```

Ces bouts de codes d'exemple nous permettent d'établir la connexion de manière efficace et sûre vers notre client en C. Chrome étant assez peu permissif il est ardu pour nous d'implémenter notre propre protocole udp sans utiliser leurs sockets.

C'est à partir de ces blocs que nous développons notre application qui tournera en arrière plan et effectuera le lien avec le client C.

2) Extension: Nous développons ensuite l'extension qui est la partie visible de l'IHM.

Elle est simplement composée d'une icône qui permet d'afficher fenêtre pop-up (Fig. 15). Cette fenêtre pop-up contient un bouton log qui, à son déclenchement va appeler l'application Chrome pour qu'il demande au client C une

authentification, si l'authentification est valide, il retourne le couple identifiant/mot de passe pour le site, sinon un message d'erreur. Une fois le couple d'identifiant/mot de passe récupéré il faut l'injecter dans le site.



FIGURE 15. Capture d'écran du plug-in utilisé sur une page de connexion Facebook

Nous devons donc ajouter un click listener sur le bouton connexion qui lancera cette procédure :

```
document.addEventListener('DOMContentLoaded', () => {
  getCurrentTabUrl((url) => {
    link = document.getElementById('log');
    link.addEventListener('click', () => {
      //ask identification
      $true = askid();
      // access granted
      if($true){
        injecttext($login , $loginfield);
        injecttext($password , $passwordfield);
      }
      // access denied
      else{
        warn("authentification failed");
      }
    });
  });
});
```

Pour cela nous utilisons une fonction injecttext qui va injecter dans une textedit d'id id du texte. C'est la fonction utilisée pour injecter le couple identifiant/mot de passe dans le site visité en cas d'authentification réussie.

```
function injecttext(string , id) {
  var script = 'document.getElementById(" + id + ").value = "' + string + '";';
  chrome.tabs.executeScript({
    code: script
  });
}
```

Nous concevons donc notre extension qui, sur ce principe, permet la connexions aux sites gérés par Face Key.

3) Communication Application-Extension: Pour relier les deux parties de notre plug-in nous utilisons une des fonctionnalités offertes par l'API de Google Chrome. Cette fonctionnalité, appelée Messaging, permet qui permet la communication inter-extensions (et également avec les applications dans notre cas).

Pour établir cette connexion nous utilisons les id de l'application et de l'extension qui nous permettent, comme elles sont toutes deux reliées à chrome, une communication facilitée. On établit ensuite une connexion qui permet d'envoyer des messages et d'attendre des réponses. Sur le code suivant on montre un exemple simplifié de communication entre deux applications via l'API Messaging.

```
// App 1
var app2id = "abcdefghijklmnabcdefghijklmnab2";
```

```

chrome.runtime.onMessageExternal.addListener(
  function(request, sender, sendResponse) {
    if(sender.id == app2id && request.data) {
      // Data sent
      // Pass an answer with sendResponse() if needed
    }
  }
);

// App 2
var app1id = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz";
chrome.runtime.sendMessage(app1id, {data: /* some data */},
  function(response) {
    if(response) {
      // Connexion established
    } else {
      // Could not connect; or App 1 not installed
    }
  }
);

```

E. Intégration au reste du projet

En résumant on obtient donc l'architecture suivante pour notre IHM (Fig. 16).

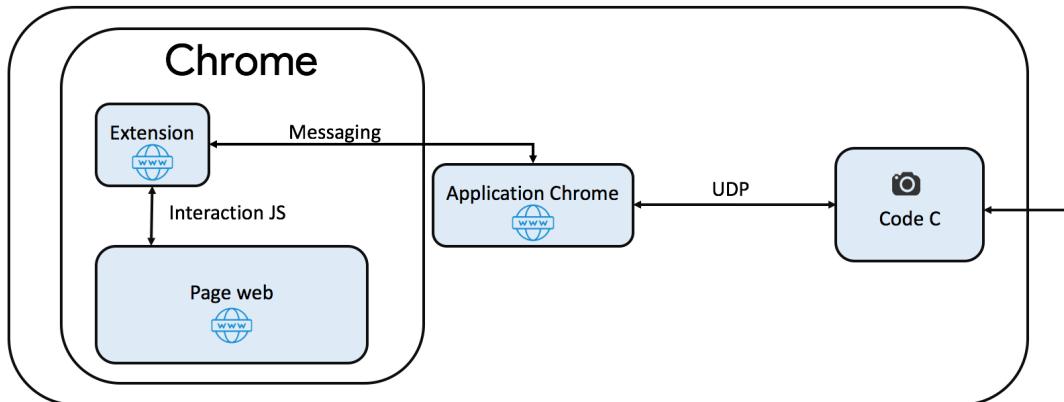


FIGURE 16. Architecture de notre plug-in

Nous avons une extension chrome qui interagit avec le contenu des sites, c'est elle que l'utilisateur utilise pour se connecter à ses sites via Face Key. Cette extension communique avec une application Chrome via Messaging. Cette Application Chrome est le relai via le client C qui est relié au reste de l'architecture de Face Key. Cette connexion est établie en udp.

F. Conclusion

Nous avons donc choisi un plug-in pour IHM nous nous devons de valider si il correspond bien aux attendus de l'IHM de Face Key et comment nous pourrions l'améliorer.

1) *Validation:* Notre plug-in répond aux attendus de l'IHM de Face Key.

- Accessibilité au plus grand public : En effet le plug-in étant développé sur chrome il couvre plus de la moitié des navigateurs web.

- Facilité d'utilisation : L'extension chrome est simple, il suffit de cliquer sur une icône en haut de son navigateur puis sur un bouton connect pour lancer l'authentification.
- Se connecter à Face Key (via la reconnaissance faciale) : Le plug-in permet bien, via le client C et l'application, une authentification via le système Face Key
- Se connecter à un site via Face Key : Le plug-in permet bien, via le client C et l'application, une authentification à un site via le système Face Key
- Ajouter un compte à Face Key : Le plug-in permet bien, via le client C et l'application, un ajout de compte au système Face Key

Le plug-in répond donc au cahier des charges de notre projet.

2) *Perspectives*: Bien qu'il réponde aux attendus, notre plug-in peut être amélioré. Un des pivots majeurs d'amélioration serait l'administration de son compte Face Key.

Un autre point d'amélioration majeur est celui de la suppression du client C. Chrome permet d'établir des connexions TCP et UDP pour ses applications. A moyen terme, une amélioration possible du projet serait donc que le client soit intégralement intégré au plug-in. On obtiendrait l'architecture suivante (Fig. 17).

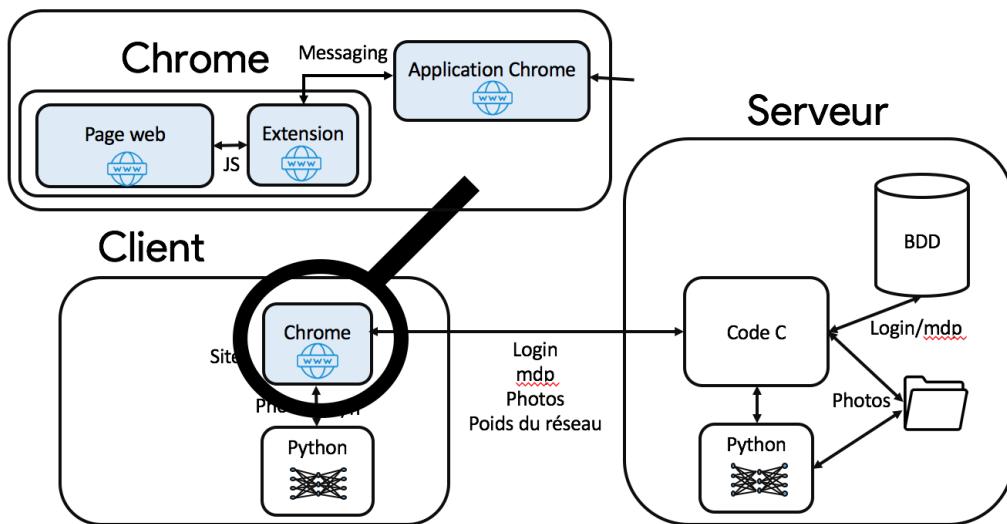


FIGURE 17. Architecture possible de Face Key avec une extension à la place du client C

Une dernière piste d'amélioration serait d'intégrer la prise d'image directement depuis le plug-in. Ce qui nous permettrait d'adopter une UI plus moderne et plus proche des standards du marché.

VII. RECONNAISSANCE FACIALE

A. contexte

Depuis quelques mois, avec la sortie de l'iPhone X et sa technologique Face ID pour le déverrouillage du téléphone, la reconnaissance faciale est sur le devant de la scène des technologies en vogue. En réalité ce problème de reconnaissance de visages est un problème qui remonte aux années 60 avec les travaux de Bledsoe, Helen Chan et Charles Bisson. Ce n'est que récemment avec le développement du deeplearning que le niveau de reconnaissance devient suffisant pour être utilisé de manière fiable dans des applications. 2014 DeepFace 97,35% de reconnaissance et 2015 FaceNet 99,63% de reconnaissance.

Nous avons donc décidé d'intégrer de la reconnaissance faciale à notre gestionnaire de mots de passe afin de rendre l'expérience utilisateur la plus simple possible. Le but du gestionnaire de mots de passe est de réduire le nombre de mot de passe que nous utilisons quotidiennement à un seul mot de passe sécurisé. Avec la reconnaissance faciale nous pouvons réduire à zéro le nombre de mot de passe et rendre la connexion à un site rapide, facile et fiable. Contrairement à Face ID de Apple qui nécessite un capteur infrarouge spécial, nous voulions que le système puisse tourner sur n'importe quelle machine, ordinateur voir même mobile, et donc n'utilisant qu'un capteur optique. Nous avons donc décidé d'expérimenter différentes techniques de machine learning pour reconnaître nos utilisateurs.

Le machine learning est un ensemble d'algorithme nécessitant un apprentissage, supervisé ou non, permettant d'effectuer des tâches de classification d'images par exemple. Cela nous permettrait donc de classifier nos utilisateurs de manière que si on lui présente une photo d'une personne, il puisse nous dire de quel utilisateur il s'agit. Pour nous, le principe est d'entraîner de manière supervisée un modèle avec une base de données de visages de nos utilisateurs afin qu'il puisse apprendre à les reconnaître et que si on lui présente une nouvelle image d'un utilisateur qu'il a appris, il puisse dire de quelle personne il s'agit.

B. Intégration de la reconnaissance faciale dans le projet

Le problème de reconnaissance est découpé en 3 parties :

- La localisation du visage dans l'image
- L'apprentissage du visage.
- La reconnaissance de ce visage

Le problème de certaines techniques de machine learning, en particulier le deeplearning, c'est qu'il demande beaucoup de ressources au moment de l'apprentissage (temps et puissance de calcul). Cette étape ne peut donc pas être réalisée sur la machine de l'utilisateur. Nous allons donc devoir la déporter sur le serveur qui devra avoir une puissance de calcul suffisante.

Pour réaliser ce genre de calcul qui devient de plus en plus courant, le département informatique de L'Université en partenariat avec Orange à investi dans un ordinateur pour réaliser des calculs sur carte graphique. En effet les cartes graphiques permettent de paralléliser un maximum de calculs, ce qui est particulièrement efficace pour les calculs de type vectoriel et matriciel utilisé en machine learning. Nous avons donc eu l'honneur et la responsabilité de monter, installer, configurer pour le calcul sur carte graphique et inaugurer cette machine dans le cadre de notre projet. C'est donc cette machine qui nous servira de serveur.



Si la partie d'apprentissage du modèle demande beaucoup de ressources, l'utilisation du modèle entraîné pour reconnaître les visages peut se faire avec très peu de ressources et peut donc être réalisée sur la machine de l'utilisateur.

La détection du visage n'est également pas très coûteuse, elle peut aussi être faite sur la machine de l'utilisateur.

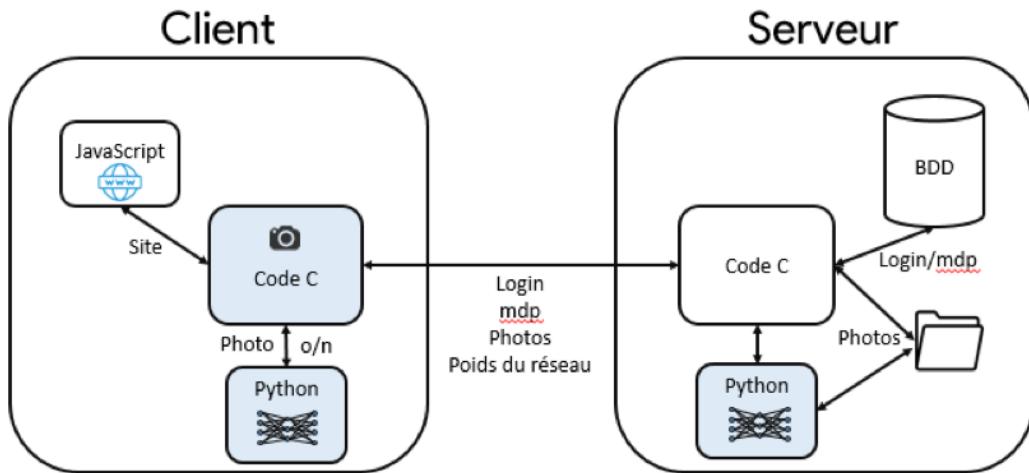


FIGURE 18. Localisation des blocs de reconnaissance dans l'architecture de Face Key

Nous avons donc sur notre client le code principal en C qui possède une fonction qui prend une photo, détecte le visage de l'utilisateur et envoie la partie de l'image correspondant au visage au code de reconnaissance. Ce dernier, en Python, possède une version entraînée du modèle de reconnaissance et va prédire la classe de l'image qu'il a reçue et la renvoyer au code principal. Le code principal vérifie qu'il s'agit bien de la bonne classe en fonction du numéro de l'utilisateur et si c'est bien le cas il fait la demande de login/mot au serveur. En réalité, pour augmenter la fiabilité, le système va prendre plusieurs images et regarder la moyenne des identifications. Nous avons fixé le nombre d'images

prisent à 5 avec un très léger temps entre les images pour minimiser la possibilité d'erreur. Si la personne est bien identifiée les images du visage sont envoyées au serveur pour l'apprentissage continu.

Sur le serveur se déroule l'apprentissage du modèle de reconnaissance, il est réalisé une fois par jour. Il prend en compte les nouvelles classes créées (nouveaux utilisateurs) et les images utilisateur courant qu'il a reçu à chaque connexion à un site. Ce système permet un apprentissage continu des classes. Si un utilisateur se laisse pousser la barbe tout en continuant à utiliser notre application alors le modèle apprendra à le reconnaître avec sa nouvelle barbe. Une fois le nouveau modèle appris, il le transmet à tous les utilisateurs pour que les prochaines reconnaissances sur les machines des utilisateurs prennent bien en compte les nouvelles images apprises.

C. Construction de la base de données d'images

La partie commune des différentes méthodes de machine learning que nous allons utiliser est qu'elles nécessitent une base de données pour entraîner le modèle de reconnaissance. Cette base de données doit être étiquetée car nous utilisons des méthodes d'apprentissage dites supervisées, c'est-à-dire que lors de l'entraînement, on présente l'image à apprendre au modèle et on lui dit à quelle classe appartient cette image. Ensuite, selon la technique d'apprentissage, le modèle va se modifier pour faire la corrélation entre l'image et la classe qui lui ont été données. La construction de la base de données est donc une étape primordiale car si les images et les labels qui lui sont présentés ne sont pas cohérents alors le modèle apprendra de mauvaises choses.

Pour notre application il faut donc construire une base d'images des visages de nos utilisateurs, étiqueté avec leur nom. Pour des raisons de relatif anonymat nous n'utilisons pas le nom des utilisateurs comme label mais leur ID client que nous sommes les seuls à connaître. Le problème est de réunir assez de photos du visage d'un utilisateur sans que ce processus soit trop lourd pour l'utilisateur.

1) *Comment l'utilisateur entre-t-il dans la base de données Face Key ?:* Pour cela, à la création du compte Face Key, nous demandons à l'utilisateur s'il veut que le programme apprenne à reconnaître son visage. S'il refuse, il ne pourra pas utiliser la reconnaissance faciale mais pourra utiliser Face Key comme un simple gestionnaire de mot de passe. Si il accepte, la fonction d'enregistrement d'images se lance, sa webcam s'allume, on détecte le visage de l'utilisateur dans chaque image du flux vidéo et on les envoie au serveur pour entraînement. Le flux vidéo est pris pendant 1 minute, pendant ce temps nous demandons à l'utilisateur de tourner sa tête sous différents angles et de réaliser différentes expressions faciales pour que le modèle apprenne à reconnaître l'utilisateur dans différents contextes. Les images transmises au serveur sont des images couleur de 100px/100px.

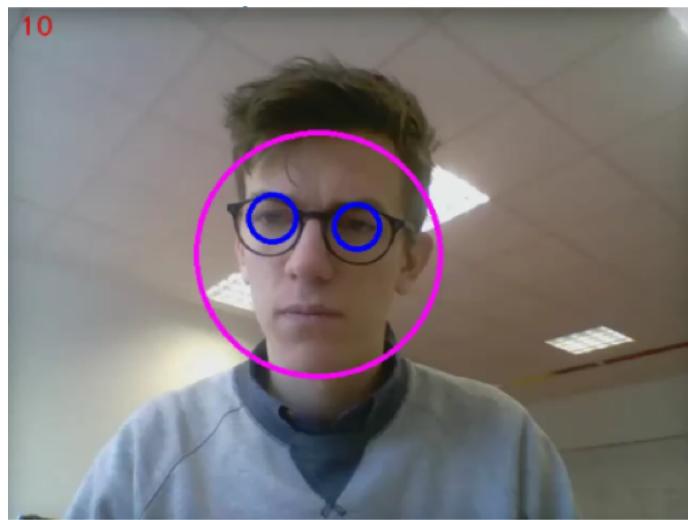
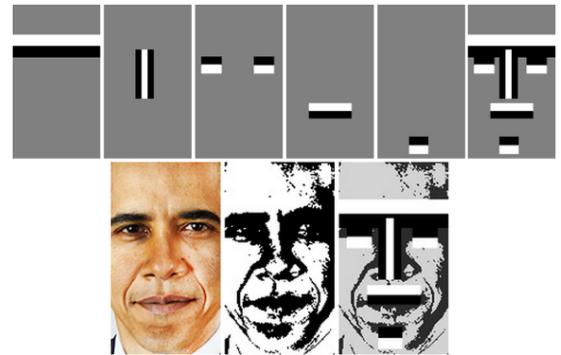
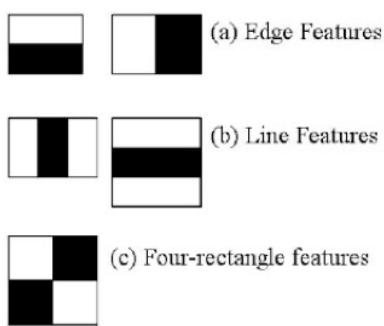


FIGURE 19. Localisation des blocs de reconnaissance dans l'architecture de Face Key

2) *Comment détecte-on le visage de l'utilisateur dans une image ?:* Pour ce faire, nous utilisons un algorithme appelé Haar Cascade qui utilise le principe de détection de features. Un visage est décomposable selon un certain

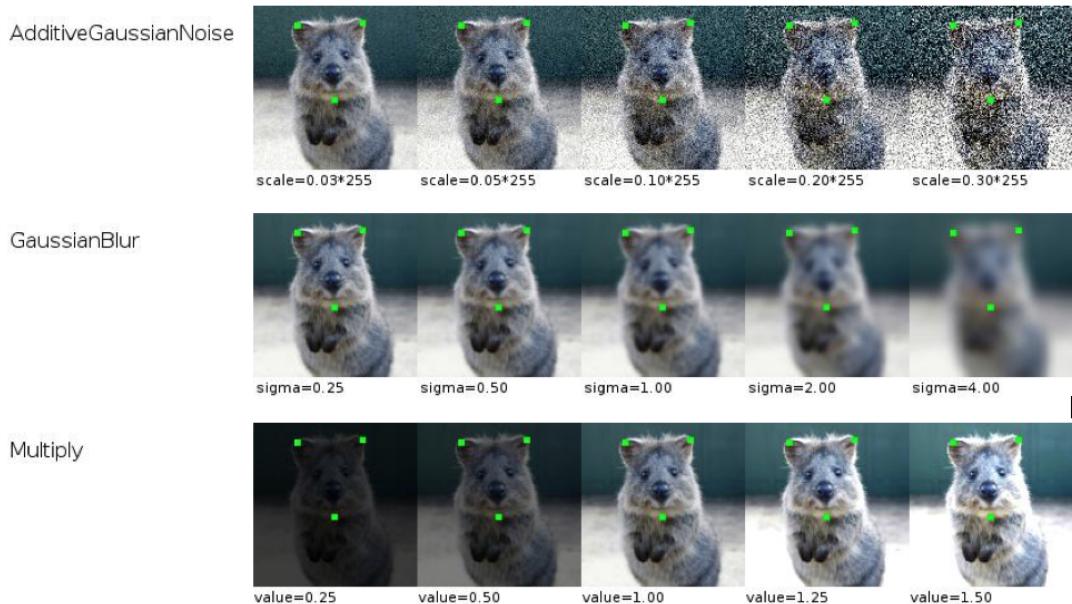
nombre de features simples.



Ensuite on utilise nos features simples comme des convolutions sur notre image et on regarde à quel endroit de la nouvelle image obtenue se trouve le pattern ressemblant le plus à notre modèle de visage simplifié.

Le projet portant plus sur la reconnaissance de visage que la localisation de visage, nous avons fait le choix de ne pas implémenter l'algorithme même si initialement nous voulions implémenter une technique appelée HOG¹ (Histogram Oriented Gradient) permettant également de détecter des visages. Cependant en vue de la charge de travail et du peu de temps imparti nous avons décidé de ne pas traiter cette partie. Nous avons donc utilisé la fonction HaarCascadeDetection d'OpenCV.

3) *Et une fois les photos collectées sur le serveur ?:* Une fois les photos des utilisateurs enregistrées sur le serveur, elles sont passées dans un augmenteur de données. En effet certaines techniques de machine learning, principalement le deep learning, nécessitent un grand nombre d'exemples d'entraînement pour être performants. Le but de l'augmenteur de données va être d'à partir d'une image de la base, créer de nouvelles images en ajoutant du bruit, flouant l'image, baisser et augmenter la luminosité ?



Ce système va permettre de créer artificiellement de nouveaux contextes et donc d'aider notre modèle à bien généraliser les classes qu'il apprend. Pour cela nous avons utilisé la bibliothèque imgaug² dédiée au machine learning et nous avons appliqué 13 transformations différentes sur nos images.

1. <https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78>
2. github.com/aleju/imgaug

4) Comment avez-vous concrètement construit votre base ?: Pour construire la base de données il nous fallait des utilisateurs à pouvoir identifier. Nous avons donc fait un appel au prêt des étudiants du département informatique et des étudiants de CMI. 13 de ces derniers ainsi que 2 professeurs ont répondu à notre appel et nous avons pu construire une base de données de 17 personnes (en ajoutant nos 3 classes et en retirant la classe de Pierre F. qui a été victime d'effacement involontaire). Nous les avons fait suivre le protocole d'une création de compte Face Key comme décrit plus haut. Cette collecte nous a permis d'enregistrer 19462 images réparties relativement uniformément sur 17 classes et de construire une base de données de 225306 images après augmentation (soit 1.3Go d'images). Ces chiffres sont relevés après le tri de la base. En effet Haar Cascade (ou du moins son implémentation dans OpenCV) est relativement sensible à l'environnement. Des conditions de luminosité changeante pouvaient générer entre 2% et 15% d'images indésirables, des images où Haar Cascade voyait un visage alors qu'il n'y en avait pas. N'ayant pas trouvé de moyen simple pour automatiser la déletion des images indésirables, nous avons dû nettoyer la base de données à la main.



5) Remarques: La construction de la base de données est une étape extrêmement importante avant d'entrainer un modèle de machine learning. Nous avons passé beaucoup de temps à construire la base de données cependant, avec le temps, nous remarquons que cette étape a été mal faite. En effet nous nous sommes laissé embarquer dans l'idée d'avoir une base de données très large car nous avions entendu dire que le deeplearning nécessite un très grand nombre de données. Dans notre cas la base est beaucoup trop redondante. Nous tirons les images d'un flux vidéo donc de l'une à l'autre les images se ressemblent beaucoup. De plus nous avons mal fait l'augmentation de données car nous étions trop hésitants à faire de grosses transformations. On se retrouve donc avec des images qui, successivement, se ressemblent beaucoup et on les duplique 13 fois sans faire de transformations suffisamment importantes. La base est donc constituée d'énormément de redondances, ce qui n'aidera pas lors de la phase d'apprentissage. Il aurait mieux valu une base beaucoup plus petite mais beaucoup plus diversifiée.

D. Apprentissage/Reconnaissance des visages

Maintenant que la base de données des visages des utilisateurs est constituée et le système de captation des visages mis en place, il nous faut entraîner notre modèle pour qu'il soit capable par la suite d'identifier les utilisateurs. Nos travaux de références sont ceux de DeepFace et FaceNet qui obtiennent respectivement 97.35% et 99.63% de reconnaissance, c'est-à-dire que dans plus de 97% des cas ils arrivent à bien classifier le visage qui leur est présenté. Notre objectif est de se rapprocher le plus possible de ces scores. Nous allons donc essayer différentes méthodes d'apprentissage.

1) Découpe de la base de données: Avant de commencer à entraîner nos modèles il nous faut, encore un peu, travailler sur notre base de données. L'un des problèmes en machine learning est le sur-apprentissage (overfitting). En effet, quand on entraîne notre modèle sur notre base de données, si on le fait trop apprendre alors il se peut qu'il apprenne par cœur la base de données. Il aura alors de très bonne performances sur les éléments de la base mais lorsque l'on déploie l'application dans le monde réel alors le modèle ne sera plus capable de correctement généraliser et aura de très mauvaises performances. Pour ne pas se faire surprendre au moment du déploiement, la solution est de découper notre base en 3 sous-ensembles : le training set, le validation set et le test set. On entraîne le modèle avec le training set, à chaque époque (chaque fois que l'on passe le training set tout entier) on fait un suivi des performances du modèle en regardant son taux de réussite sur le validation set sans entraîner le modèle sur ces images. À la toute fin de l'apprentissage on regarde les performances finales sur le test set, sans apprentissage, et on peut voir directement si l'on a sur-appris ou pas. Mais pourquoi utiliser un test set et un validation set ? Un des deux ne suffirait-il pas ? Même si il n'y a pas d'apprentissage sur le validation set, le modèle peut l'apprendre indirectement. En effet on va continuer à apprendre si l'on remarque que le taux de performance sur le validation set ne suffit pas, on va donc apprendre implicitement à s'adapter au validation set. Le test set est donc là pour tester le modèle sur des images qu'il n'a jamais vu, même pas implicitement. On pourra donc se fier aux chiffres de performance du test set.

2) Méthode par régression logistique:

a) Théorie: La régression logistique est un modèle de classification linéaire probabiliste. C'est un outil simple de classification en machine learning, que l'on utilise souvent en premier pour tester la complexité de son problème. Il est initialement fait pour faire de la classification binaire, classification en 2 classes.

Il est composé d'une fonction de coût S :

$$S(X^i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Avec :

X^i : une observation de la base de données

x_i : un élément de X

θ_i : un poids de la fonction hypothèse, ce qu'on cherche à apprendre

θ_0 : un biais

Cette fonction est ensuite passée dans une fonction sigmoïde qui donnera un résultat y entre 0 et 1. Ce résultat peut être interprété comme la probabilité que le vecteur d'entrée X soit de la classe 1. On a donc :

$$\text{Sigmoid}(S(X)) = P(y \geq 0,5 | X; \theta) \text{ et donc } P(y < 0,5 | X; \theta) = 1 - P(y \geq 0,5 | X; \theta)$$

Pour faire de la classification multi-classes on peut utiliser la technique du One-vs-Rest, c'est-à-dire que de la même manière, on calcule la probabilité que 1 sorte par rapport au reste des classes. On obtient donc un ensemble de poids θ pour chaque classe.

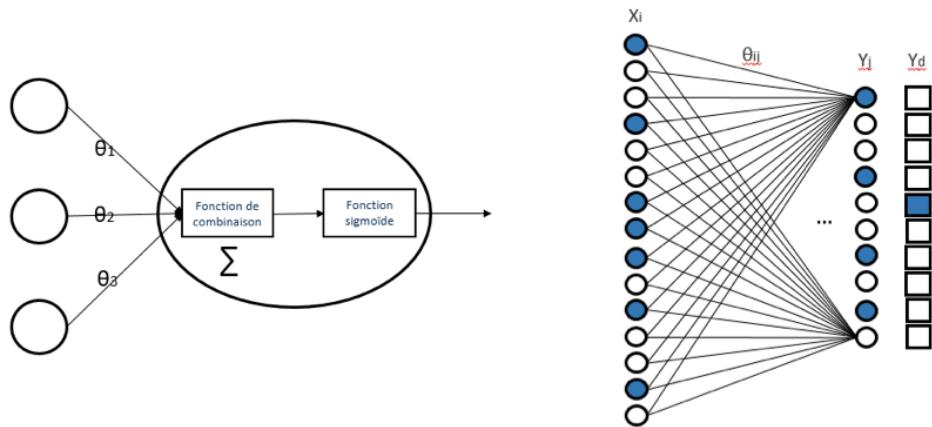
Pour qu'il y ait adaptation il faut que les poids θ_i soient appris. On va donc les mettre à jour en fonction de l'erreur faite par le modèle.

Tout d'abord on calcule l'erreur faite par la fonction Sigmoid par rapport au résultat attendu y_d : $err = y_d - y$. Ensuite on calcule combien on va mettre à jour le poids θ_i : $\Delta\theta_i = \lambda \cdot err \cdot x_i$ avec λ un coefficient d'apprentissage.

Enfin on met à jour le poids : $\theta_i(t+1) = \theta_i(t) + \Delta\theta_i$

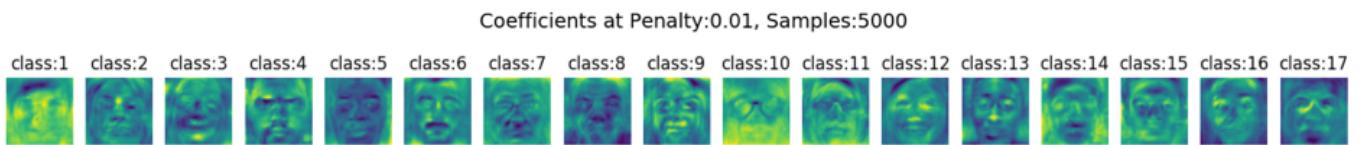
On met ainsi à jour tous les poids du modèle de façon à ce qu'ils s'adaptent en fonction de l'erreur qu'ils ont provoquées.

La régression logistique peut aussi se représenter sous forme de réseau de neurones (perceptron simple). Les neurones prennent un vecteur d'entrée qu'ils passent dans une fonction sigmoïde et renvoient le résultat en sortie. Le réseau sera donc défini comme suit :



b) *Implémentation:* On utilise donc les images de visages de nos utilisateurs (en échelle de gris) sous forme de vecteur, c'est-à-dire avec les colonnes mises les unes à la suite des autres, pour alimenter ce modèle. L'implémentation de l'algorithme est faite « from scratch » en C.

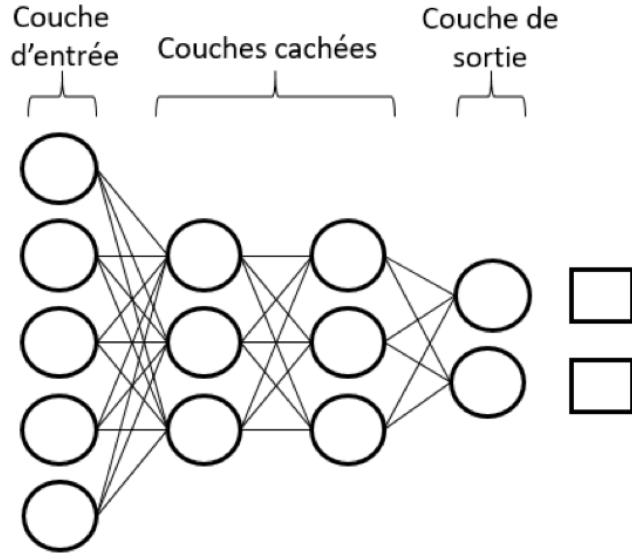
c) *Résultats:* Après avoir passé la totalité du training set on obtient 53,2% de reconnaissance sur le test set. Ce résultat très faible, mais au-dessus de l'aléatoire qui est à 6%, n'est pas étonnant car la régression logistique est une méthode de classification linéaire et notre problème semble être non linéaire. Nous ne retiendrons pas cette méthode pour le système d'identification de Face Key. Nous avons également essayé avec la bibliothèque sklearn qui utilise une version améliorée avec une fonction softmax à la place de la sigmoïde et offre un affichage facile des résultats. Nous obtenons un score plus élevé de 62.3% de reconnaissance sur 5000 exemples d'apprentissage. Si l'on s'amuse à afficher les ensembles θ de chaque classe, on peut voir quels pixels ont de l'importance dans la reconnaissance de chaque classe et on voit se dessiner des ébauches des visages.



3) Méthode par perceptron multicouches:

a) *Théorie:* Le perceptron multicouches (Hinton 1986) est basé sur le principe de la régression logistique sous forme de réseau de neurones. Cependant contrairement au perceptron simple qui est une méthode de classification linéaire, le perceptron multicouches pourra résoudre des problèmes non linéaires.

Le principe est d'ajouter des couches, dites cachées, entre la couche d'entrée et la couche de sortie.



Les neurones ont le même comportement que dans le perceptron simple (somme des poids d'entrées et sigmoïde). La différence est dans la manière de mettre à jour les poids du réseau. L'idée est de minimiser la fonction d'erreur par le calcul du gradient et la mise à jour des poids en conséquence.

Sous la forme matricielle on suit les étapes suivantes :

Calcul de l'erreur des neurones de sortie :

$$E_i = y_i^d - y_i$$

Calcul du signal d'erreur des neurones :

$$\delta_i = y_i \otimes (1 - y_i) \otimes E_i$$

Calcul de l'erreur de la couche précédente :

$$E_j^n = \sum_j W_{ij} \times \delta_j^{n+1}$$

Mise à jour des poids :

$$W_{ij}^{t+1} = W_{ij}^t + \eta \times y_i^n \times \delta_j^{n+1}$$

Calcul du signal d'erreur :

$$\delta^n = y^n \otimes (1 - y^n) \otimes E^n$$

Et on répète ces étapes pour mettre à jour tous les poids du réseau. Ainsi, de la même manière qu'avec la régression logistique, on peut modifier nos poids en fonction de l'erreur commise par le modèle et donc le faire apprendre. Cet algorithme s'appelle la backpropagation ou gradient-descent.

Pour aider à la généralisation, nous utilisons sur ce genre de réseau une technique de régularisation appelée dropout. Le principe est simple, on désactive aléatoirement un certain pourcentage de neurones sur chaque couche à chaque tour d'apprentissage. Cette technique va permettre un renforcement plus rapide des poids des neurones en fonction de leur importance et prévient également de l'overfitting.

b) Implémentation: Cette technique à également été programmée from scratch mais nous n'avons pas pu tester sa performance sur notre base de données car celle-ci est trop grande et trop complexe. Elle nécessite un réseau avec beaucoup de neurones en couches cachées et le nombre d'images à entraîner a rendu l'apprentissage trop long pour un être calculé sur CPU sur un seul thread.

Nous avons donc utilisé la librairie de machine learning Tensorflow qui propose du calcul sur carte graphique.

Modèle choisi :

Couche	Nombre neurone
Input	10 000
Hidden 1 + dropout 0.5	1024
Hidden 2 + dropout 0.5	512
Hidden 3 + dropout 0.5	256
Output	17
Total du poids :	10 899 712

c) Résultats: Avec le modèle décrit ci-dessus on obtient 67,1% de reconnaissance sur notre base de données. Ce résultat n'est pas significativement supérieur au résultat obtenu avec la régression softmax de sklearn. Le problème vient peut-être donc du type de réseau de neurones utilisé. Avec 67,1% de reconnaissance nous sommes encore loin des 99% de FaceNet, nous ne gardons donc pas le perceptron multicouches comme solution de reconnaissance faciale.

4) Méthode par réseau de neurones convolutif:

a) Théorie: Le problème avec le perceptron multicouches est qu'il est long à entraîner quand on augmente son nombre de neurones car tous les neurones des couches successives sont connectés entre eux. Il y a aussi une perte d'informations lorsque l'on aplatis notre image en un vecteur car la continuité qu'il y a entre les différentes lignes d'une image n'est pas prise en compte. C'est ce qu'apporte l'arrivée des réseaux de neurones convolutif (CNN, Y.Lecun 1998) et le développement du deeplearning. Le principe des CNN est d'ajouter des couches dites de convolutions avant un perceptron multicouches.

Le principe de la convolution est d'utiliser des noyaux, initialisé aléatoirement puis appris, que l'on la convoluer sur l'image d'entrée pour obtenir des filtres. Dans un filtre obtenu, chaque valeur sera le taux de correspondance entre le noyau et la partie de l'image centré en ce point.

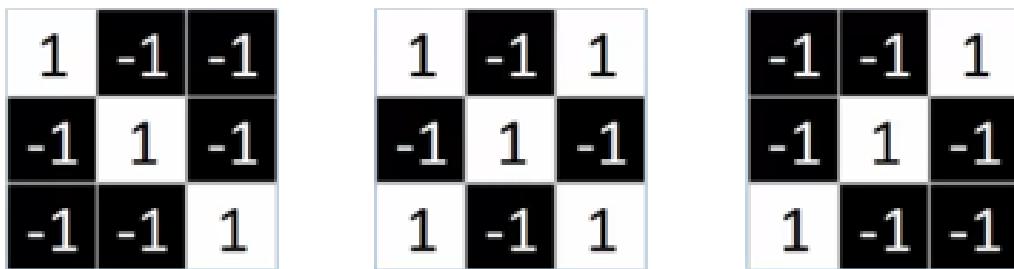


FIGURE 20. Opération de convolution

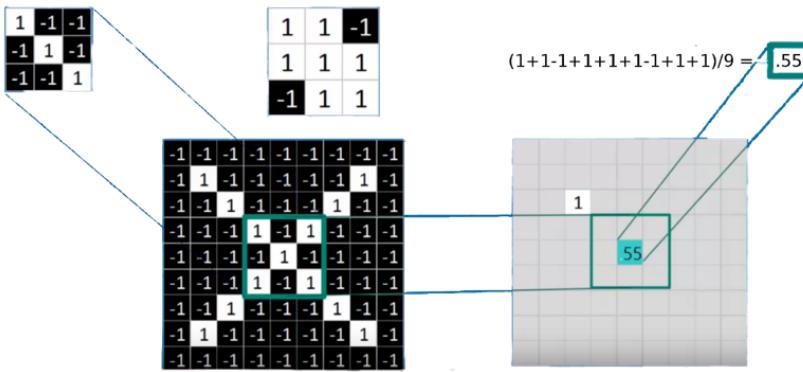


FIGURE 21. Noyaux

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	-0.11	0.11	-0.11

FIGURE 22. Filtre obtenu

On peut utiliser plusieurs paramètres pour customiser l'opération de convolution :

- Le padding : Si l'on effectue une convolution simple sur l'image (avec un ?same padding?) on aura une réduction de dimension des filtres obtenu par rapport à l'image d'entrée. On peut donc entourer notre image d'entrée de 0 pour obtenir la même dimension en sortie (?zero padding?, utilisé dans l'exemple ci-dessus)
- Le stride : Le décalage en pixel du noyau lorsqu'on le ?glisse? pendant l'opération de convolution. Le stride a un impact sur la taille des filtres de sortie.

On peut par la suite enchainer les couches de convolution en utilisant les filtres comme image d'entrée pour la couche suivante. Les couches de convolution vont permettre de décrire l'image en différents patterns. On va donc avoir des filtres qui permettront de décrire de plus en plus précisément notre image originale.

Une autre opération que l'on introduit avec le CNN est le pooling, en particulier le max polling. C'est une couche qui va permettre de réduire la dimension de nos filtres en gardant l'information importante. On prend un noyau d'une certaine taille, en fonction de la façon dont on veut diminuer la dimension, et on ne garde que la valeur maximale de l'image dans le noyau.

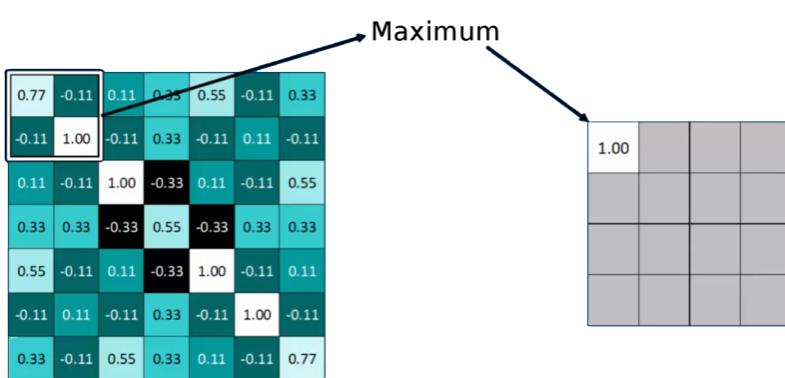


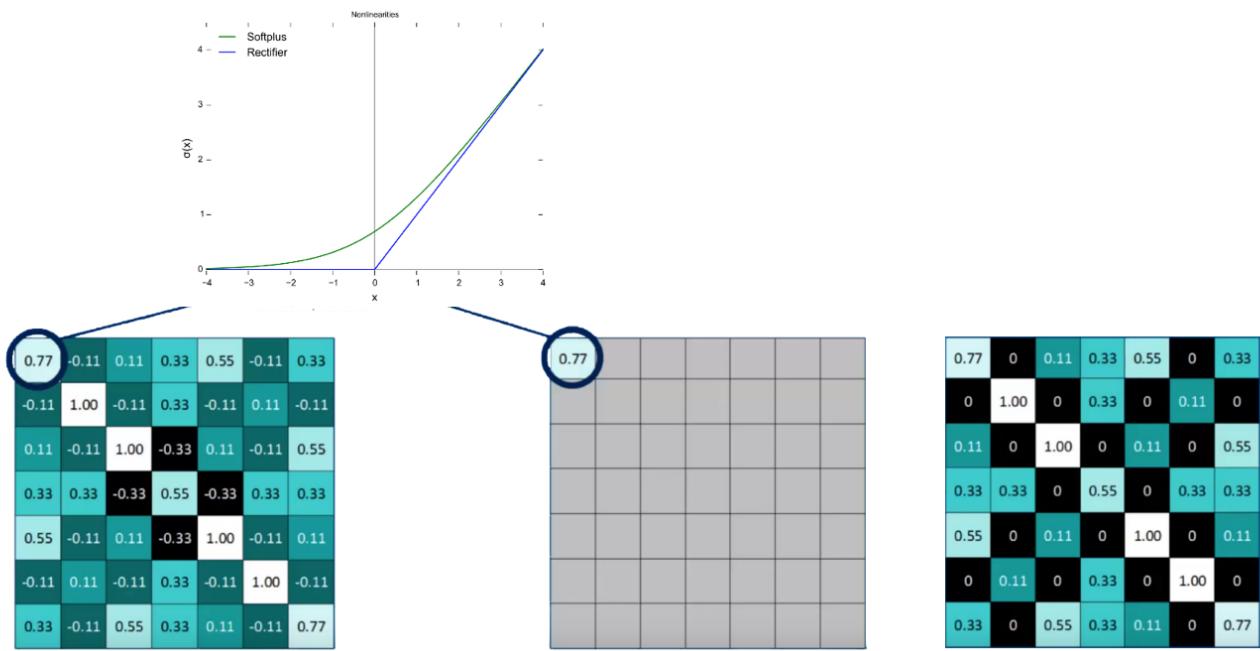
FIGURE 23. Opération de Pooling

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

FIGURE 24. résultat maxPooling

On peut de la même manière jouer sur le stride du pooling.

Enfin on change notre fonction d'activation sigmoïde par une fonction ReLU (Rectify Linear Unit) car la fonction sigmoïde à pour désavantage d'être ?aspirante? à ses extrémités.

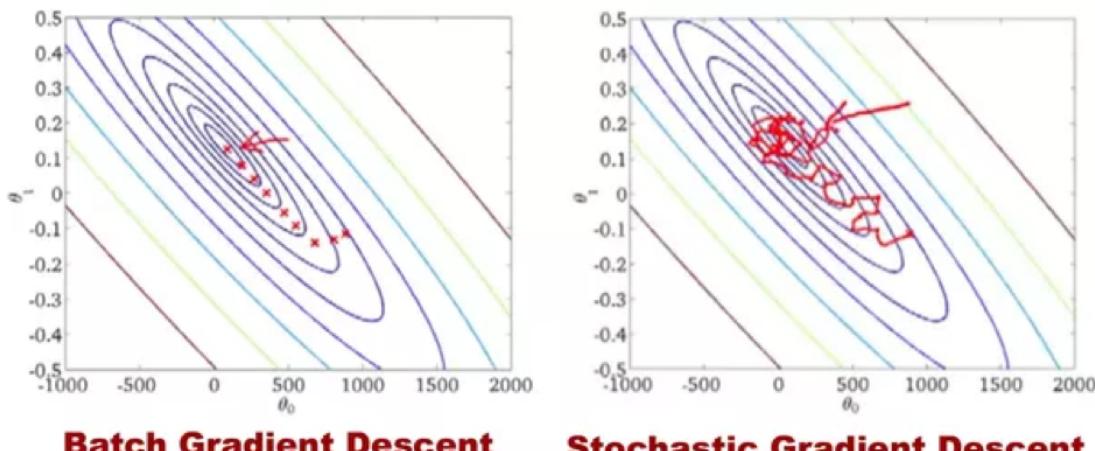


On peut maintenant mettre bout à bout toutes ces couches pour obtenir notre CNN. On suit généralement une architecture comme :

INPUT \rightarrow [[CONV \rightarrow RELU] $*N \rightarrow$ POOL? $]*M \rightarrow$ [FC \rightarrow RELU] $*K \rightarrow$ FC

Avec FC (Fully Connected) des couches de perceptron.

Pour l'apprentissage on utilise le même principe de minimisation par descente du gradient mais de manière stochastique (stochastic gradient descent). Cette technique nécessite plus d'étapes pour converger mais est beaucoup plus rapide à calculer.



b) *Implémentation:* Cette fois ci nous n'avons pas implémenté ce modèle from scratch mais nous avons utilisé la librarie Keras qui tourne avec Tensorflow pour un entrainement rapide sur carte graphique.

Nous voulions initialement implémenter les mêmes architectures que DeepFace et FaceNet mais au moment de la création de la base de données nous avons sauvegarder les images en 100x100 alors que Deepface utilise des images de 152x152 et FaceNet.

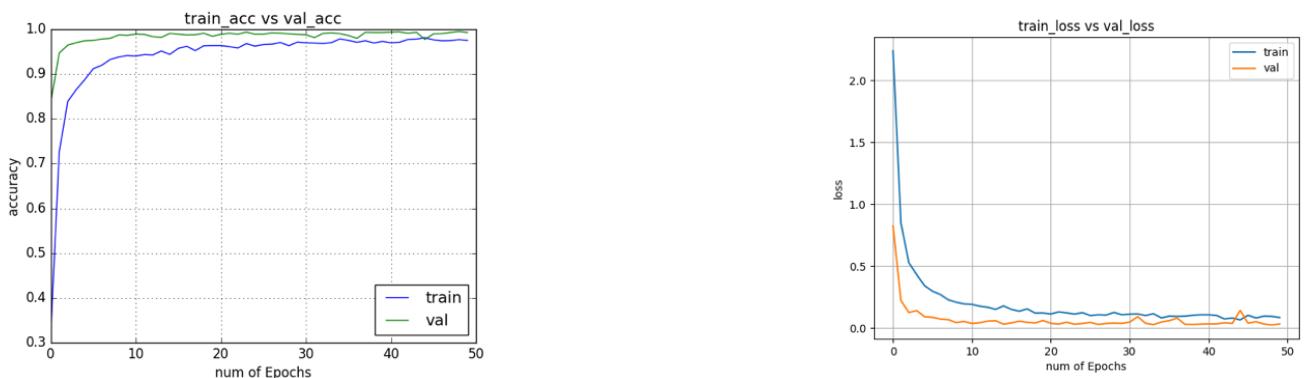
Nous avons donc décidé de designer notre propre architecture. Nous décidons également de rester sur des images en noir et blanc, en pensant que l'humain ne nécessite pas la couleur pour identifier des gens sur une photo et nous voulions que notre réseau se focalise sur les formes des visages. Nous sommes conscients que c'est un parti-pris osé de par la quantité d'informations que l'on retire. Les résultats présentés ci-dessous sont donc réalisés sur des images noir et blanc, de nouveaux résultats seront peut-être présentés en couleur pour la présentation.

Architecture choisie :

Couches	Description
Convolution	32 filtres, noyau (3x3)
ReLU	
Convolution	32 filtres, noyau (3x3)
MaxPooling + Dropout 0.5	Noyau (2x2)
Convolution	64 filtres, noyau (3x3)
ReLU	
Convolution	64 filtres, noyau (3x3)
ReLU	
MaxPooling + Dropout 0.5	Noyau (2x2)
Flatten	Mise en vecteur des filtres
Fully connected	64 neurones
ReLU + dropout 0.5	
Fully connected	17 neurones
SoftMax	
Total du poids : 2 052 000	

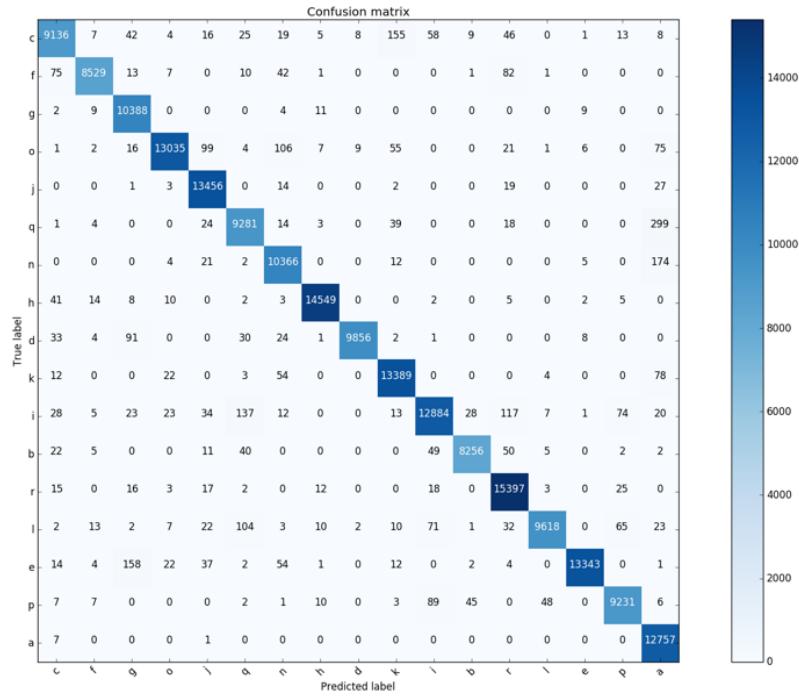
Avec le modèle décrit ci-dessus on obtient 97,1% de reconnaissance sur notre base de données. Avec ce niveau de reconnaissance on se rapproche des résultats de DeepFace et FaceNet, ce qui est logique comme on utilise le même type de réseau qu'eux.

On peut observer l'évolution du loss (erreur que fait le réseau) et de l'accuracy (taux de réussite) sur la base de training et de validation pendant l'apprentissage.

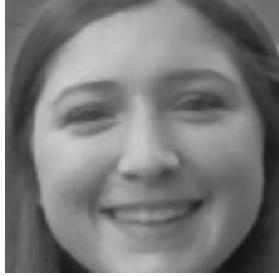


L'entraînement a été fait sur 50 époques mais on constate que l'accuracy n'augmente plus significativement à partir de la 7ème époque, les poids enregistrés sont donc ceux de la 7ème époque pour éviter l'overfitting.

On peut également afficher la matrice de confusion entre les classes :



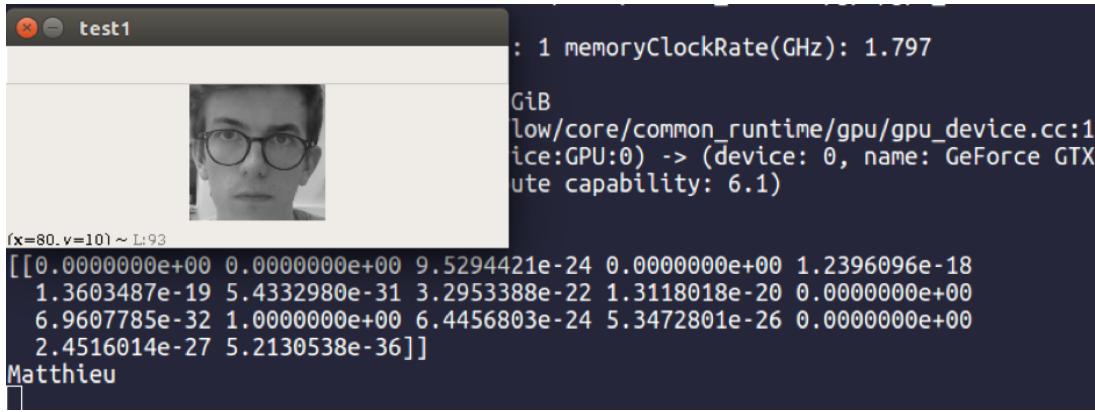
On remarque qu'il y a très peu de confusion entre les classes. De plus certaines confusion sont explicable, par exemple il y a eu $255+12=267$ erreur entre la classe c et la classe k :



Les deux individus ont une forme de visage assez ronde, des yeux qui se ressemblent, des cheveux longs ? Les erreurs commises entre ces deux classes sont donc compréhensible (ce nombre d'erreur reste très faible par rapport au nombre d'images total)

On remarque que l'on obtient de meilleures performances sur la validation que sur le training ce qui s'explique par le fait que comme notre base est très, trop, beaucoup trop, redondante. Lors de la division en 3 sous bases, une grande partie, voir la totalité, de nos validation set et test set sont composés d'images du training set. Nous n'avons donc aucun moyen de prévenir l'overfitting de notre modèle.

On remarque effectivement l'overfitting lorsque l'on déploie notre application et que l'on test des images qui n'appartiennent pas à notre base de données. Nous avons testé sur 3 images de 3 classes différentes. Pour 2 des 3 classes le visage est reconnu.



Nous prenons donc une nouvelle photo de l'individu de la 3eme classe dans d'autres conditions de luminosité, d'angle etc. Le modèle n'arrive toujours pas à l'identifier.

Nous n'avons malheureusement pas pu faire de réel test chiffré pour l'utilisation du modèle dans le monde réel. Nous n'avons donc aucun moyen de savoir de manière fiable la performance de notre solution de reconnaissance faciale. Nous espérons pouvoir faire ces tests avant la soutenance.

E. Conclusion : remarques, avancement et améliorations possibles

Nous remarquons que la plupart de nos problèmes viennent de la mal formation de la base de données. Nous avons passé beaucoup de temps pour la constitué mais il a sans doute manqué un temps de recherche sur comment bien constituer une base de données.

La finition de cette partie n'est pas parfaite, nous avons passé beaucoup de temps à nous documenter, constituer la base, choisir les modèle, prendre en mains les outils et implémenter tous les algorithmes (y compris ceux qui ne se voient pas sur la manipulation de données pour la construction de la base). Il ne nous restait donc plus beaucoup de temps pour faire le lien entre tout ce qui a été développé.

Au niveau des améliorations possible nous pensons bien évidemment à la refonte totale de la base de données (nombre, d'images, taille, augmentation différentes?). Nous pensons également à la recherche d'un modèle réellement efficace, voir sir FaceNet marche aussi bien sur une base de données maison (99,63% ont été obtenu sur la base de données Labeled Faces in the Wild LFW). Et enfin une nouvelle méthode de localisation des visages, implémentation de HOG pour test ou utiliser un réseau qui localise également le visage (Object detector, exemple réseau YOLO).

D'un point de vue pratique, mais qui soulève un problème théorique, comment allons-nous gérer l'ajout d'un nouvel utilisateur ? Il faut rajouter une classe de sortie mais doit-on ré entraîner le réseau avec uniquement les photos de l'utilisateur ? A priori ce posera des problèmes de généralisation de cette classe, mais doit-on alors réapprendre toute la base ? Ce qui est difficile pour une application qui possède plusieurs milliers d'utilisateurs. Ce genre de problème est pour nous sans réponse pour le moment mais ils vaudraient la peine de se pencher dessus.

VIII. SECURITÉ

A. Introduction au problème

Lors de l'utilisation de l'application, beaucoup de données transitent entre les différents blocs que composent l'architecture du logiciel, et/ou sont stockés dans le Base de Donnée. Parmis elles, se trouvent des informations extrêmement sensibles concernant les utilisateurs. Pour un utilisateur, nous pouvons y trouver ces informations (liste non ordonnée et non exhaustive) :

- Adresses mails
- Mot de passe (de l'application Face Key)
- Combinaisons Identifiants/Mot de passe de plusieurs sites
- Coordonnées Bancaires
- Et d'autres

Si un utilisateur mal intentionné arriverait à intercepter les messages entre le client et le serveur, en se plaçant entre les deux grâce à un attaque basique de type Man in the Middle (MITM), alors il aurait accès à toutes ces données. De même si la base de donnée arrivait un jour à être compromise, et accessible par un utilisateur mal intentionné, alors il aurait également accès à toutes les données. Il nous faut donc trouver une solution qui nous permet à la fois de sécuriser les envoies de données Client/Server, et une solution qui nous permet de rendre illisible les informations dans la base de données à tous les utilisateurs autres que l'utilisateur concerné.

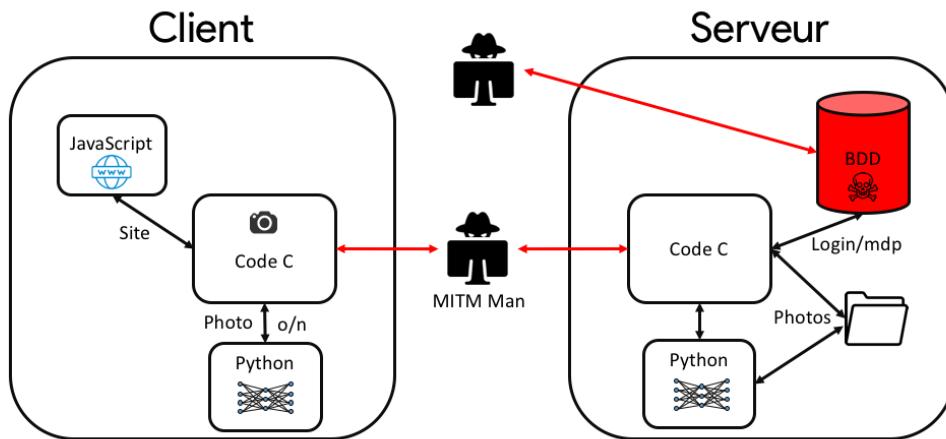


FIGURE 25. Problème de sécurité

B. RSA

1) *Introduction:* Le RSA est un algorithme de cryptographie dit "asymétrique" (une clé privée et une clé publique, contrairement à la cryptographie symétrique qui utilise la même clé pour crypter et décrypter). Il a été rendu public par R.L. Rivest, A. Shamir, et L. Adleman dans un papier publié en 1977 appelé "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*". Il a pour but, à la fois de crypter des messages afin qu'ils puissent être transmis sans risquer d'être lu par une personne tierce, mais aussi de signer les messages. Un des principes du RSA est que la clé privée, permettant de déchiffrer les messages soit impossible à retrouver à partir de la clé publique, qui elle permet d'encrypter les messages. Nous avons donc décidé d'utiliser cette méthode afin de crypter les messages entre le Client et le Server.

2) *Génération des clés:* La génération des clés appartient au receveur des messages, il doit ensuite communiquer la clé publique, qui n'est pas une information sensible, à l'envoyeur du message. Dans notre cas, le serveur et le client vont chacun générer de leur côté une paire de clés et transmettre à l'autre partie, la clé publique. Ainsi, même si les clés sont interceptées, n'importe qui pourra envoyer des messages au client et au serveur, mais personne ne pourra lire les messages cryptés envoyés, à l'exception du client et du serveur.

Pour créer les clés nous devons d'abord choisir deux nombres premiers p et q et ensuite calculer

$$n = p \cdot q$$

Notons que n sera présent dans la clé publique il est donc important de trouver deux nombres premiers p et q assez grands pour qu'il soit difficile de décomposer n (Il est recommandé d'après les auteurs d'utiliser un nombre de premiers composant de 100 chiffres), en effet chaque nombre n a une décomposition **unique** en facteur premiers. On cherche ensuite d tel que :

$$\text{pgcd}(d, (p - 1) \cdot (q - 1)) = 1$$

Nous venons donc de créer notre clé privée (d, n) , pour trouver notre clé publique nous devons calculer e tel que :

$$e \cdot d \equiv 1 \pmod{(p - 1) \cdot (q - 1)}$$

Notre clé publique est donc le couple (e, n) .

3) *Cryptage et Décryptage des messages:* Pour crypter un message M nous avons besoin de notre clé publique (e, n) , notre message crypté C se calcule de la manière suivante :

$$C \equiv M^e \pmod{n}$$

La fonction de décryptage est tout aussi simple, à partir de notre message crypté C , et de notre clé privée (d, n) , nous calculons :

$$M \equiv C^d \pmod{n}$$

La méthode décrite ici est démontrée mathématiquement dans le papier d'origine de R.L. Rivest, A. Shamir, et L. Adleman.

4) *Implémentation:* Pour implémenter l'algorithme nous avons choisi d'utiliser la bibliothèque libre et open source OpenSSL, et plus particulièrement la bibliothèque *libcrypto* qui fournit les algorithmes de cryptographie. OpenSSL est une bibliothèque largement répandue et utilisée, il est donc plutôt facile de trouver de la documentation dessus (suivant les fonctionnalités utilisées).

a) *Génération des clés:* Les clés publiques et privées du serveur et du client sont générées préalablement, et stockées dans les fichiers des deux programmes.

b) *Échange des clés:* Lors de chaque connexion du client au serveur, les clés publiques des deux parties sont échangées, envoyées sous forme de fichier ".pem" et stockées, avec les autres clés pour le client, et dans un dossier nommé par le PID du processus pour le serveur (le serveur étant multi-processus).

c) *Cryptage et décryptage:* Pour le cryptage et le décryptage, les fonctions de OpenSSL sont utilisées après avoir chargé la clé dans la mémoire. Le cryptage et le décryptage sont opérationnels pour peu que les deux clés issues de la même paire soient utilisées pour crypter et décrypter.

d) *État de l'implémentation:* Le système n'est pas complètement implémenté, en effet, malgré le bon fonctionnement de chacune des briques indépendamment des autres, nous pouvons observer un nombre non négligeable de ratés lorsque nous essayons d'utiliser l'ensemble dans le contexte Client/Serveur. En effet, la transmission du message crypté ne se fait pas correctement (contrairement aux messages non cryptés), il apparaît effectivement que la somme de vérification du message crypté de l'une des parties, n'est pas la même que la somme de vérification du même message

mais de l'autre côté. Nous pouvons en déduire que le message s'est mal transmis, cependant nous n'avons pas encore trouvé la raison de ce phénomène.

C. Fonctions de Hachages

1) Introduction: Une fonction de hachage cryptographique, est une fonction avec un message (ou fichier) en entré et une valeur en sortie appelé valeur de hachage, somme de vérification ou encore empreinte numérique. Le principe d'une fonction de hachage est de prendre le fichier en entré et de le transformé en une valeur (une chaîne de caractère la plupart du temps) unique qui sera son "empreinte numérique". Une autre propriété importante de la fonction de hachage est qu'elle est très difficilement (voir presque impossible) inversible, c'est à dire qu'à partir de la valeur en sortie il est impossible (dans un temps raisonnable) de retrouver la valeur en entrée. De même une fonction de hachage idéal n'aurait aucune collision, c'est à dire que pour deux messages différents données ne pourrions pas avoir la même valeur en sortie.

2) Intégration au sein du projet: Au sein du projet Face Key, nous utiliserons une fonction de hachage (MD5) pour transformer les valeurs stockées dans la base donnée et régulièrement envoyé par l'utilisateur tel que : son mot de passe. Ainsi à chaque fois que l'utilisateur enverra son mot de passe, nous récupérerons la valeur hachée stocker dans la base, et nous transformerons le mot de passe envoyé et vérifierons les 2 sommes de vérifications. Si les 2 sommes sont les mêmes alors le mot de passe donné est le même sinon le mot de passe est erroné. Cependant nous pouvons remarquer qu'une fonction de hachage n'est pas suffisante afin de garantir une sécurité satisfaisante. En effet, si la base venait à être compromise et lu par une personne tierce, il lui suffirait d'utiliser une liste des mots de passe les plus utilisés préalablement hachés, et de seulement comparer les sommes de vérifications à celle dans la base donnée pour révéler plusieurs mots de passe. Sachant que 2 même mot de passe obtiennent la même sortie cela diminue grandement le temps nécessaire à retrouver plusieurs mots de passe. De plus, il existe certaines méthodes pour retrouver des mots de passe hachés en clair, comme les attaques par dictionnaires ou Rainbow Table. Nous devons donc trouver une solution pour rendre la tâche plus compliquée.

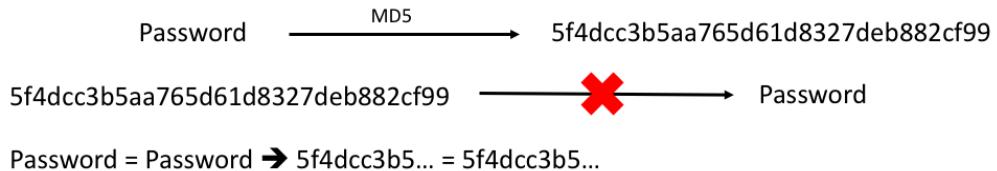


FIGURE 26. Principe du Hachage

3) Le Haching salé: Le principe du hachage salé est simple : ajouter une chaîne de caractères que l'on va concaténer au mot de passe avant de hacher la chaîne finale. Chaque utilisateur a un sel unique stocker sur la base de donnée. Ainsi 2 même mots de passe n'auront pas la même chaîne hachée. Cela rend les attaques par Brute Force, Dictionnaire et Rainbow Table, bien plus fastidieuse en augmentant le temps nécessaire pour retrouver les mots de passe.

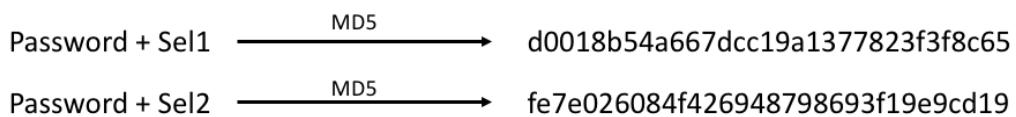


FIGURE 27. Hachage salé

D. Bilan de la sécurité

Pour commencer ce bilan, faisons un point sur ce qui est implémenté ou non. Nous avons vu que le RSA est partiellement implémenté, chaque bloc fonctionne de manière indépendante, mais la transmission du message crypté ne s'effectue pas bien. Pour ce qui est du hachage, seul le hachage classique est implémenté, le hachage salé n'est pas implémenté par manque de temps. En effet trouver d'où venait le problème du RSA nous à pris pas mal de temps, et nous ne pouvons à l'heure actuelle, estimer combien de temps il faudrait pour le résoudre, car nous ne connaissons pas la cause du problème. L'implémentation du Hachage Salé nécessiterait une modification de la base de donnée, pour stocker le "sel" de chaque utilisateur. De plus, il resterait d'autres problème de sécurité à régler, même si nous avions pu tout implémenté. En effet les combinaisons Identifiants/Mot de passe pour chaque site web d'un utilisateur ne sont pas protégé! Il aurait aussi été intéressant de crypter l'envoie et la sauvegarde des fichiers (images et réseau de neurones) sur le serveur, afin de préserver l'identité des utilisateurs.

IX. CONCLUSION

A. point d'avancement par rapport au cdc

B. problème rencontrés + discussion

C. amélioration possible + discussion

D. ressentit par rapport au projet

Acknowledgement