Face Key

Quentin GERARD, Louis L'HARIDON et Matthieu VILAIN
CMI Systèmes Intelligents et Communicants, Université de Cergy-Pontoise
mail@etu.u-cergy.fr

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.

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, diffèrent 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 chaine 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 connaitre 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) Maturation 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 à 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 taches 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 utilise 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 DEFININIR

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 développent un protocole d'installation, une documentation détaillé 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

test

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) :

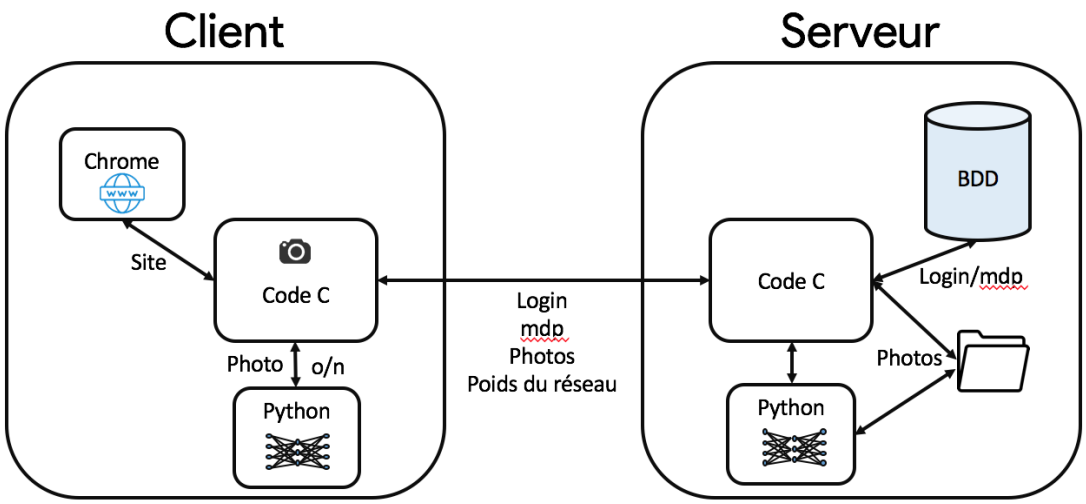


Figure 1. Architecture

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 maitre / 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)

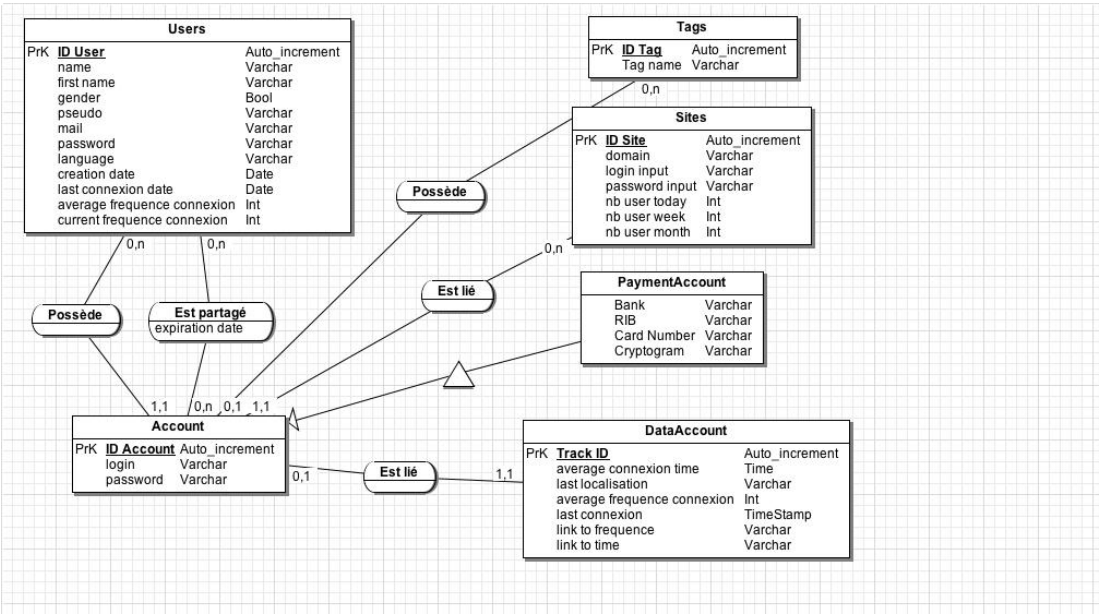


Figure 2. Modélisation Conceptuelle des Données

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)

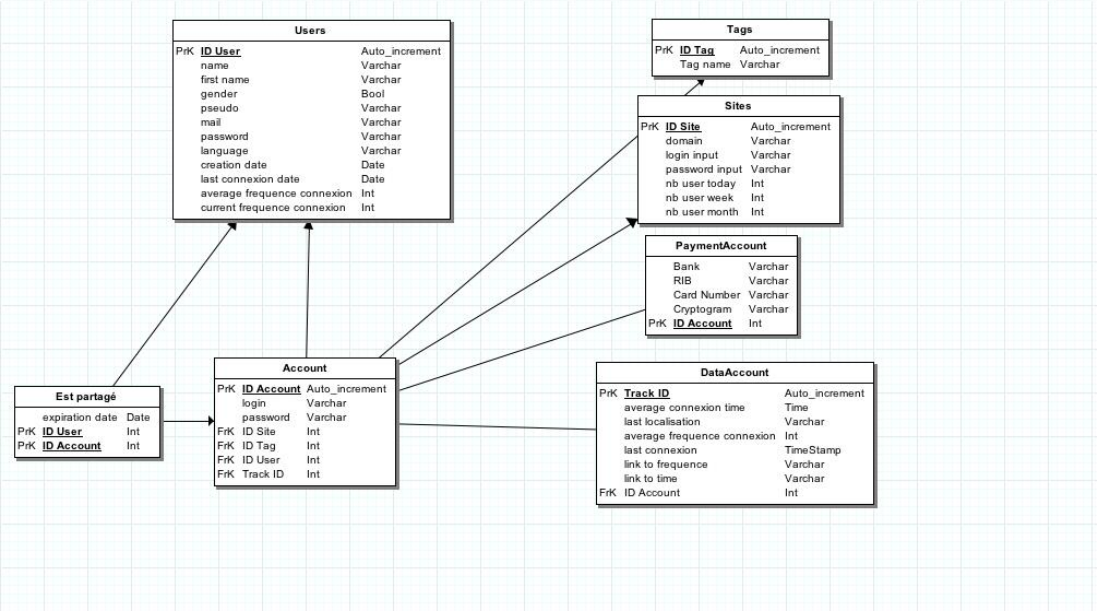


Figure 3. Modélisation Logique de Données

C'est grâce à cd modèle que nous créé 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).

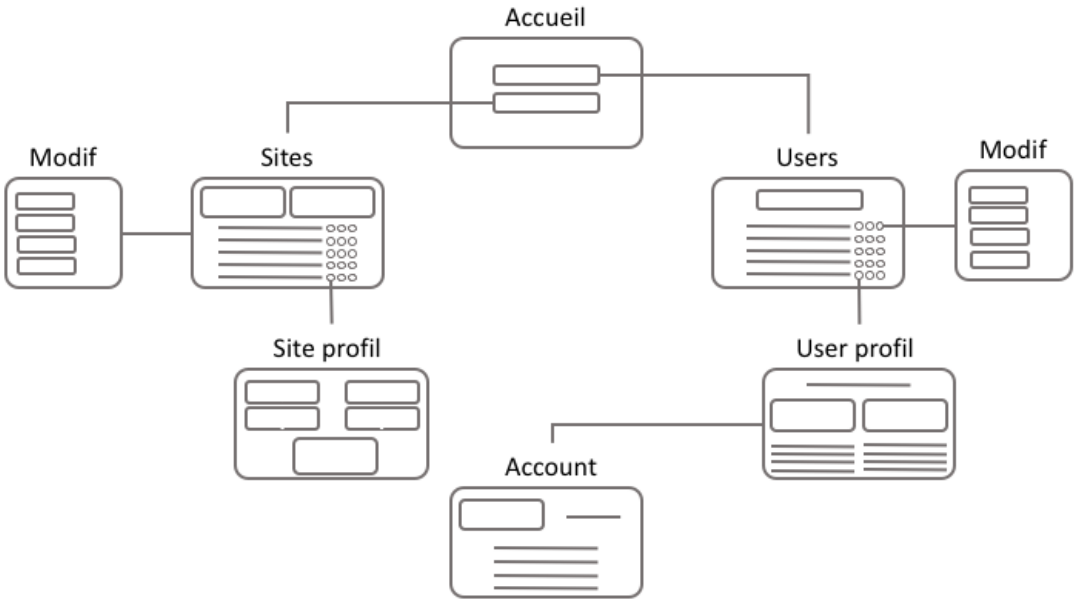


Figure 4. Schématisation du fonctionnement du Panel d'Administration de la BDD

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

V. Application Android

A. Introduction

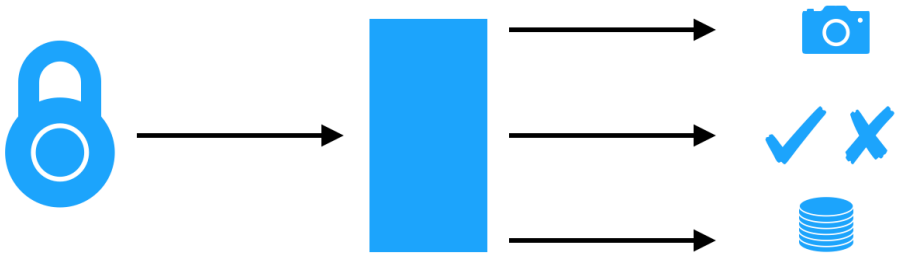
Pour le projet FaceKey nous avons besoin d'un grand jeu de données pour entraîner nos modèles de reconnaissance faciale. Pour augmenter ce jeu de données nous eu avons besoin d'un processus plus orienté utilisateur. Nous avons donc

décidé de créer une application Android qui va prendre des photos d'un utilisateur et les sauvegarder sur notre Base de Données pour pourvoir entraîner nos modèles. Nous voulons donc donc une application qui propose à l'utilisateur de so 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 ou on peut choisir entre trois options :

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



C. Activitées

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"}};
```

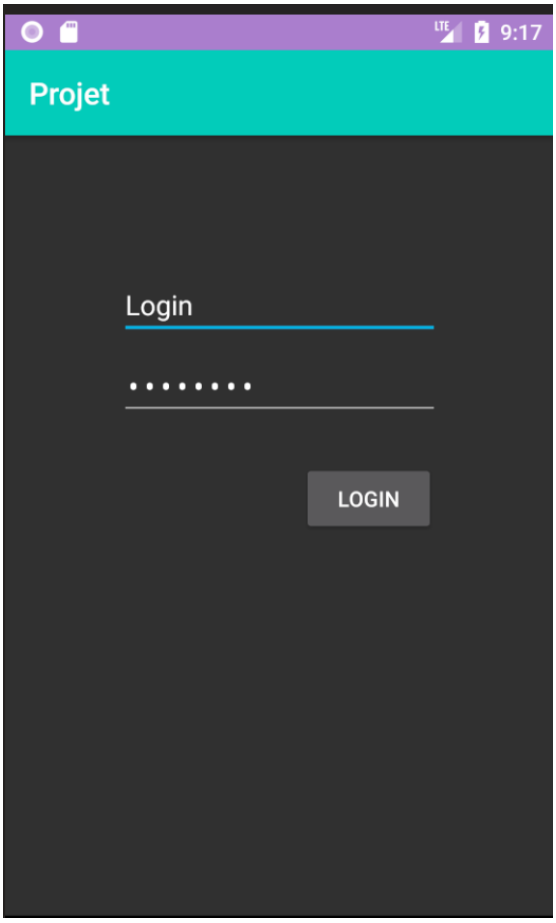
Et, en cliqaunt sur le bouton soumettre on vérifie si le couple login/mdp correspond à un autre couple présent dans le tableau. present on the array.

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

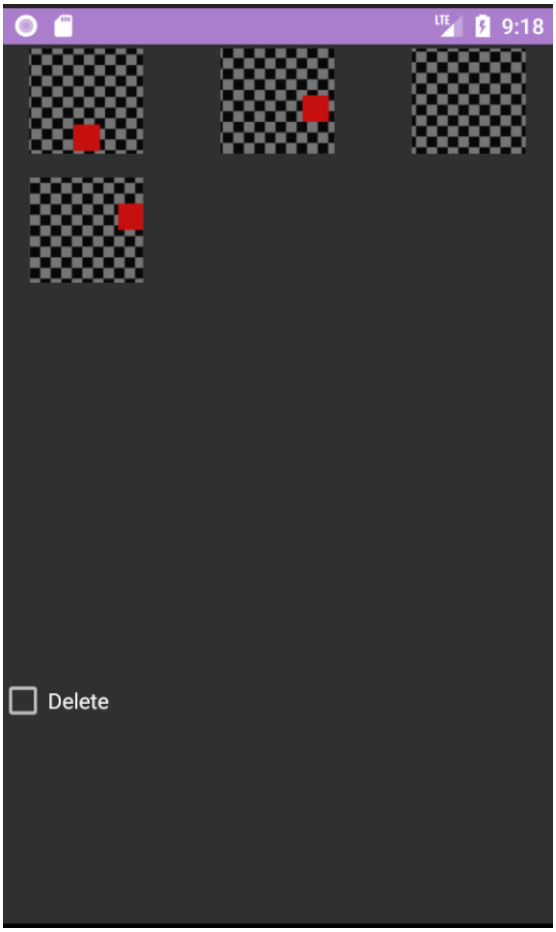
```
}
```

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

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 adapteur va simplement prendre une image au format bitmap. la redimensionner pour la faire rentrer dans l'imageview et la placer dans le griview 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 lsiter tous les fichier, y accéder et les ajouter au griview 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 supprimier.

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

```
                                    int position, long id) {
                // if checkcbox 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 raffraîchit 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, transfèrera 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 s i 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 bitmat
    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("Updload", "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("Updload", "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("Updload", "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 appContext;
// Function to toast on main thread
void showToast(final String toShow){
    // checking app context
    if(null !=appContext){
        // geting main theard
        Handler handler = new Handler(Looper.getMainLooper());
        handler.post(new Runnable() {
            @Override
            public void run() {
                // threading
                Toast.makeText(appContext, 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.

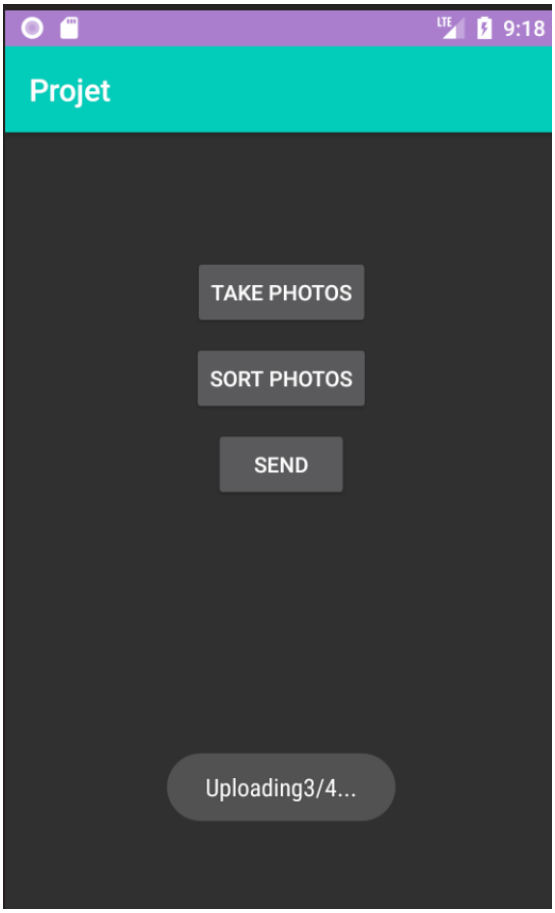


Figure 5. Flower one.

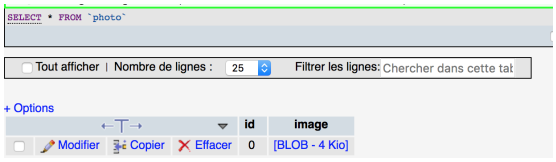


Figure 6. Flower two.

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 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. 7).

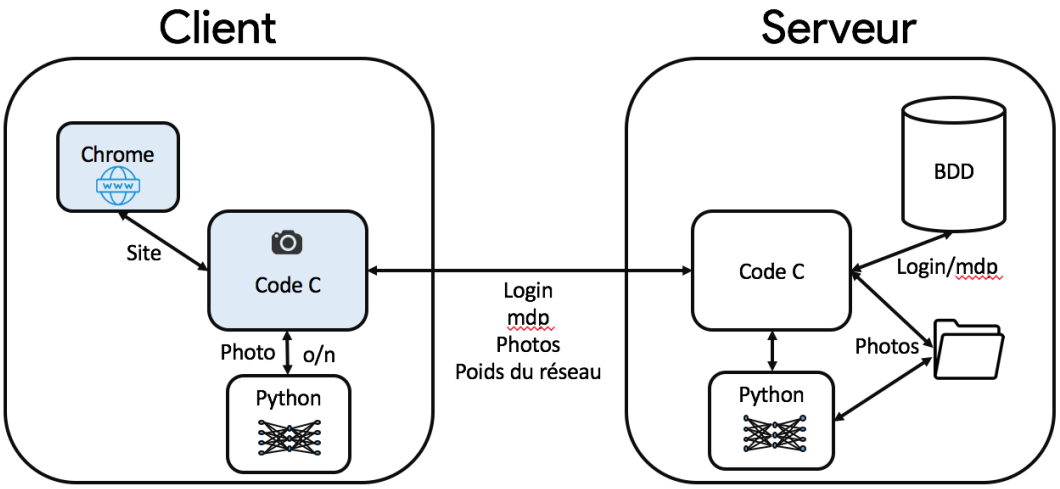


Figure 7. Architecture

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 pouvoir 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 besoin 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 fonctionnalité 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. II).

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 application chrome sont des web-applications développées en HTML5, CSS et javascript et dont l'objectif est de se créer une l'expérience d'une application native.

L'avantage majeur des application Chrome, en comparaison avec des application 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 application Chrome, bien que démarrées depuis lanceur d'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. 8). 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.

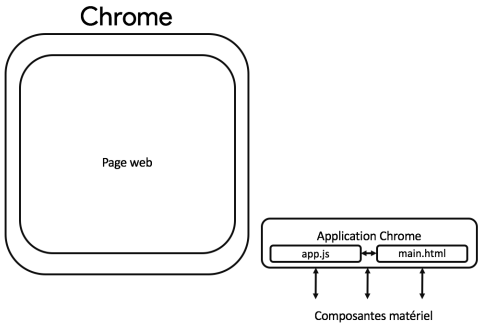


Figure 8. Architecture

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 chromes sont, quand à elles, des petits logiciels qui permettent de personnaliser l'experience 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 que 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 application 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. 11). 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 fichier 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.

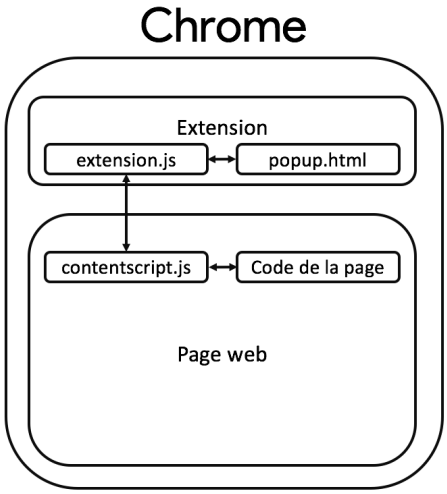


Figure 9. Architecture

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 relions 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'example nous permettent d'établir la connexion de manière efficace et sure 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. 10). 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 10. Architecture

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 identifient/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 example simplifié de communication entre deux applications via l'API Messaging.

```
// App 1
var app2id = "abcdefghijklmnoabcdefhijklmnoab2";
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 = "abcdefghijklmnoabcdefhijklmnoab1";
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. 11).

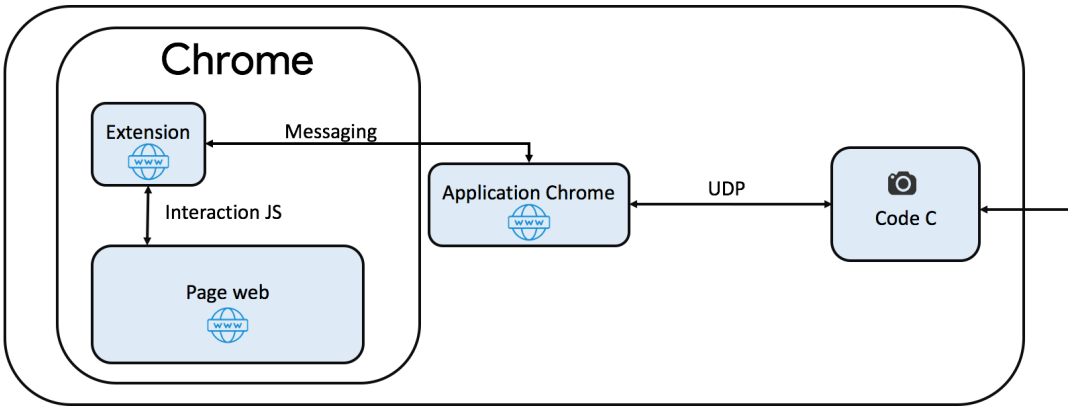


Figure 11. Architecture

Nous avons une extensions 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 développe 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'extension, 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. 12).

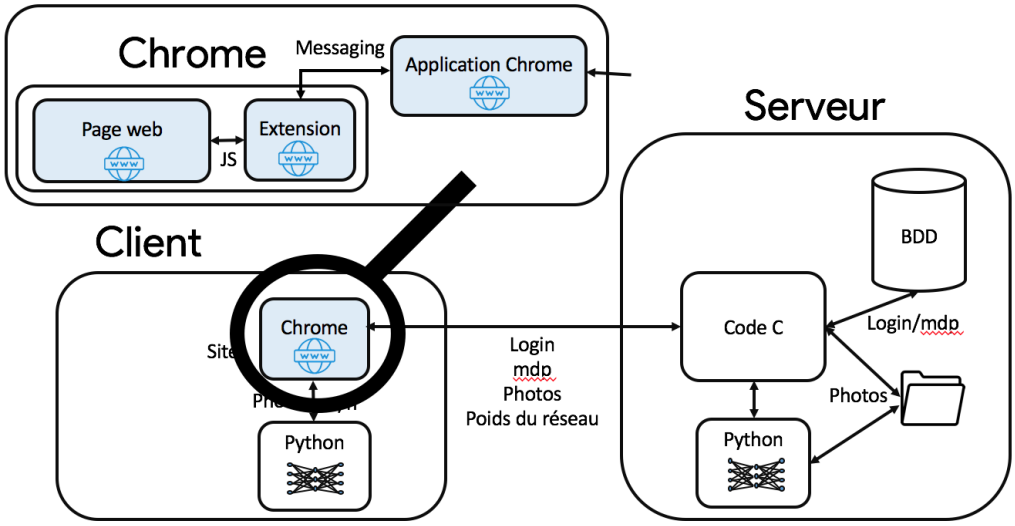


Figure 12. Architecture

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

VIII. Securité

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