RAPPORT DU PROJET : Application Mobile de Reconnaissance Faciale

Introduction

------------

Dans le cadre du projet final du module de développement mobile (L3 ING SEC – USTHB), il nous a été demandé de réaliser une application mobile permettant une authentification sécurisée par reconnaissance faciale, en utilisant Flutter et Firebase. Ce projet s’inscrit dans le contexte actuel où les solutions biométriques prennent une place de plus en plus importante dans le domaine de la sécurité informatique.

Vue d’ensemble et objectif

--------------------------

L’objectif principal de cette application est de permettre à un utilisateur de créer un compte et de s’authentifier en utilisant soit une adresse email et un mot de passe, soit par reconnaissance faciale. Une fois connecté, l’utilisateur peut tester la reconnaissance faciale à travers des images capturées ou sélectionnées, afin de vérifier l’authenticité de l'identité.

Les objectifs secondaires incluent :

* Intégrer un système sonore pour indiquer le succès ou l’échec de l’authentification.
* Offrir la possibilité à l’utilisateur de modifier ses informations et l’image utilisée pour la reconnaissance faciale.
* Fournir une interface conviviale et ergonomique.

----------

**Conception**

La conception de l’application repose sur deux volets :

1. **Authentification**
   * Inscription avec username, email, mot de passe et photo faciale.
   * Authentification par email et mot de passe ou reconnaissance faciale.
2. **Reconnaissance faciale**
   * Capture d’image via la caméra ou sélection depuis la galerie.
   * Comparaison des visages à l’aide d’un algorithme de similarité.
   * Retour sonore (audio succes ou failed) selon le résultat.

L’architecture suit un modèle basé sur des écrans interconnectés : écran d’inscription, écran de connexion, page d’accueil, page home de test de reconaissance faciale, etc.

Scénario de fonctionnement :

1. Lancement de l’application

2. Page d’accueil : choix entre inscription et connexion

3. Si l’utilisateur n’a pas de compte :

- Redirection vers l’écran d’inscription

- Saisie de nom d’utilisateur, l’email, du mot de passe et capture d’une photo

- Création du compte + enregistrement de la photo dans Firebase

4. Si l’utilisateur a déjà un compte :

- Choix entre connexion par mot de passe ou par reconnaissance faciale

- En cas de reconnaissance faciale :

- Capture d’une nouvelle photo

- Comparaison avec la photo de référence

- Accès autorisé ou message d’erreur (avec signal sonore)

5. Une fois connecté :

- Accès à la page principale avec options :

- Modifier les informations ou l’image faciale

- Tester la reconnaissance avec une image de la galerie ou une nouvelle capture

Technologies et langages utilisés

---------------------------------

- Flutter : framework principal pour le développement mobile multiplateforme.

- Firebase : Authentification, Firestore, Storage

- Dart : langage de programmation utilisé avec Flutter.

- Packages Flutter utilisés :

- google\_ml\_kit : détection et traitement des visages.

- firebase\_auth, cloud\_firestore, firebase\_storage : intégration des services Firebase.

- image\_picker, camera : gestion de la capture et sélection d’images.

- audioplayers : ajout de sons pour signaler le succès ou l’échec.

Implémentation

--------------

Dans cette partie, nous présentons comment nous avons effectué une authentification faciale avec le package « flutter\_face\_api»  
Dans la prochaine partie, nous détaillons comment nous avons utiliser **google\_ml\_kit\_face\_detection** pour regrouper les visages similaires en détectant leurs points caractéristiques (landmarks). Cela sera utile pour stocker les visages dans une base de données et, plus tard, lorsque l’utilisateur capturera son visage avec la caméra, nous pourrons le reconnaître à partir des visages enregistrés.

Cette étape est nécessaire car, imaginons que tu as 100 visages enregistrés dans la base de données. Lorsqu’un utilisateur capture son visage, pour pouvoir l’identifier, le package doit comparer ce visage capturé avec chacun des visages déjà stockés. Et il faut savoir que la reconnaissance faciale peut prendre environ 3 à 6 secondes par comparaison.

Donc, faire en sorte que le système compare un à un les 100 visages ferait attendre l’utilisateur trop longtemps, ce qui n’est clairement **pas une bonne expérience utilisateur**. C’est là que cette partie entre en jeu.  
En utilisant le package **google\_ml\_kit**, on peut appliquer une logique de présélection entre le visage capturé et les visages enregistrés. Cela permet d’établir une **liste réduite de visages probables**. Ensuite, on appelle flutter\_face\_api pour effectuer la reconnaissance précise, ce qui permet d’authentifier le visage avec seulement **2 ou 3 comparaisons**, idéalement.

### Voici les étapes **Étapes de mise en œuvre de l’authentification faciale**

**Étape 1 : Capturer le visage**

**Pour capturer les images à partir de la caméra ou de la galerie, nous utilisons le plugin image\_picker.**

**Ajoutez la dépendance suivante dans le fichier pubspec.yaml :**

**image\_picker: ^0.8.6**

**Puis, exécutez la commande :**

**flutter pub get**

**Pour sélectionner et extraire une image, on utilise le code suivant :**

**Bloc de code 1 : Capture du visage et conversion en chaîne de caractères**

**> Ce code permet de prendre une photo avec la caméra ou de sélectionner une image, puis de la convertir en base64 (chaîne de caractères). Cette chaîne peut ensuite être enregistrée dans une base de données ou dans les préférences locales (SharedPreferences).**

**Étape 2 : Préparer les images pour la comparaison**

**--------------------------------------------------**

**Supposons que l’image capturée à l’inscription ait été stockée dans une base de données. Cette étape consiste à récupérer cette image et à la préparer pour la comparaison.**

**Remarque : Ceci fait partie du flux d’authentification.**

**1. Récupérez l’image enregistrée depuis la base de données et stockez-la dans une variable nommée storedImage.**

**2. Capturez le visage de l’utilisateur à l’instant de la connexion avec le même code que le bloc 1, et stockez cette nouvelle image dans une variable nommée capturedImage.**

**NB : Les deux variables (storedImage et capturedImage) sont des chaînes de caractères représentant des images en base64.**

**Étape 3 : Comparaison des visages**

**---------------------------------**

**Ajoutez la dépendance suivante dans pubspec.yaml :**

**flutter\_face\_api: ^4.1.1**

**Puis exécutez à nouveau :**

**flutter pub get**

**Pour effectuer la comparaison entre les deux visages, utilisez :**

**Bloc de code 2 : Code pour authentifier un visage**

**> Ce code utilise l’API Flutter Face pour comparer les deux chaînes (images) et déterminer si le visage capturé correspond à celui enregistré.**

**Et voilà, c’est tout !**

**Vous avez maintenant une méthode fonctionnelle pour capturer, stocker, et comparer les visages dans une application Flutter à des fins d’authentification faciale.**

**"""**

Partie 1 – Mise en place de Firebase et de la structure de base

- Configuration initiale de Firebase

- Définition du modèle utilisateur

- Capture d’image à l’inscription

- Envoi des données vers Firebase

- Navigation via Navigator

Partie 2 – Détection de visage et comparaison via ML Kit

 Utilisation de google\_ml\_kit pour détecter les visages

 Calcul de la **distance euclidienne** entre les vecteurs de visages

 Lecture de feedback sonore

 Comparaison d’images capturées ou sélectionnées

Extrait de code de comparaison :

double compareFaces(List<double> vector1, List<double> vector2) {

double sum = 0;

for (int i = 0; i < vector1.length; i++) {

sum += pow(vector1[i] - vector2[i], 2);

}

return sqrt(sum);

}

**Conclusion**

----------

Ce projet a permis de concevoir une application mobile complète combinant authentification traditionnelle et reconnaissance faciale à l’aide de technologies modernes telles que Flutter et Firebase. Il a mis en valeur l’importance des solutions biométriques et a offert l’opportunité de manipuler des outils avancés comme google\_ml\_kit.

Voici les perspectives d’amélioration possible pour l’application :

- Optimiser la précision de la reconnaissance avec des modèles plus avancés

- Ajouter la reconnaissance multi-visages

- Renforcer la sécurité des données (cryptage)

- Améliorer l’interface utilisateur (UX/UI)

- Ajouter un support partiel hors-ligne