Université des Sciences et de la Technologie Houari Boumediene

Faculté d'informatique

Application Mobile de Reconnaissance Faciale

Membres de l'équipe :

Akchiche Meriem Lyna
Baroud Lina

Année Scolaire: 2024/2025

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Contexte

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.

1.2 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.

.

CHAPTER 2

CONCEPTION

Ce chapitre présente la conception menée pour le développement de notre application mobile Visionary. Il est structuré en deux parties que nous allons détailler ci-après:

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 succès ou échec) 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. 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.

CHAPTER 3

IMPLÉMENTATION ET INTERFACES GRAPHIQUES

Dans ce chapitre, nous allons présenter les outils et les langages utilisés pour développer notre application mobile, par la suite nous allons présenter l'implémentation du code. Enfin, nous présenterons les différentes interfaces de notre application mobile Visionary.

3.1 Les outils 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.

3.2 Implémentation de l'application

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 comme FireBase 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, dans le cas où nous avons 100 visages enregistrés dans la base de données FireBase. 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 de mise en œuvre de l'authentification faciale :

- **Étape 1** : Configuration de FireBase avec le code d'application.
- **Étape 2** : Capturer le visage.

  Pour capturer les images, nous avons utilisé le plugin : image_picker.

  Nous avons ajouté la dépendance suivante dans le fichier pubspec.yaml: image_picker: 1.0.4

  Pour sélectionner et extraire une image, nous avons utilisé le code suivant :

  Bloc de code 1 :

```dart
  Future _getImage() async {
    setState(() {
      _image = null;
    });
    final pickedFile = await _imagePicker?.pickImage(
      source: ImageSource.camera,
      maxWidth: 400,
      maxHeight: 400,
      // imageQuality: 50,
    );
    if (pickedFile != null) {
      _setPickedFile(pickedFile);
    }
    setState(() {});
  }
  Future _setPickedFile(XFile? pickedFile) async {
    final path = pickedFile?.path;
    if (path == null) {
      return;
    }
    setState(() {
      _image = File(path);
    });

    Uint8List imageBytes = _image!.readAsBytesSync();
    widget.onImage(imageBytes);

    InputImage inputImage = InputImage.fromFilePath(path);
    widget.onInputImage(inputImage);
  }
}
```

Figure 3.1: Flux de Sélection et de Traitement d'Images

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 la base de données FireBase.

- **Étape 3** : Préparer les images pour la comparaison.

  Supposons que l'image capturée à l'inscription ait été stockée dans FireBase. Cette étape consiste à récupérer cette image et à la préparer pour la comparaison.

  Ceci fait partie du flux d'authentification.

  1. Récupérer l'image enregistrée depuis la FireBase et stockez-la dans une variable nommée storedImage.
  2. Capturer le visage de l'utilisateur à l'instant de la connexion avec le même code que le bloc 1, et stocker 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 4** : Comparaison des visages.

  Nous avons ajouté la dépendance suivante dans pubspec.yaml : flutter_face_api: 4.1.1
  Pour effectuer la comparaison entre les deux visages, nous avons utilisé le code suivant :

  Bloc de code 2 :

```
  const AuthenticateFaceView({Key? key}) : super(key: key);

  @override
  State<AuthenticateFaceView> createState() => _AuthenticateFaceViewState();
}

class _AuthenticateFaceViewState extends State<AuthenticateFaceView> {
  final AudioPlayer _audioPlayer = AudioPlayer();
  final FaceDetector _faceDetector = FaceDetector(
    options: FaceDetectorOptions(
      enableLandmarks: true,
      performanceMode: FaceDetectorMode.accurate,
    ),  // FaceDetectorOptions
  );  // FaceDetector
  FaceFeatures? _faceFeatures;
  var image1 = regula.MatchFacesImage();
  var image2 = regula.MatchFacesImage();


  bool _canAuthenticate = false;

  List<dynamic> users = [];

  UserModel? loggingUser;

  bool isMatching = false;
```

Figure 3.2: Classe d'État pour l'Authentification Faciale

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

Partie 2 – Détection de visage et comparaison via ML Kit

Lors de l'enregistrement du visage par l'utilisateur, nous pouvons sauvegarder ce visage sous deux formats :

- Premièrement, sous forme de chaîne de caractères.
- Deuxièmement, en extrayant les caractéristiques faciales de l'utilisateur et en les enregistrant également.

Pour stocker ces caractéristiques faciales, nous avons utilisé le code suivant:

```
class FaceFeatures {
  Points? rightEar;
  Points? leftEar;
  Points? rightEye;
  Points? leftEye;
  Points? rightCheek;
  Points? leftCheek;
  Points? rightMouth;
  Points? leftMouth;
  Points? noseBase;
  Points? bottomMouth;

  FaceFeatures({
    this.rightMouth,
    this.leftMouth,
    this.leftCheek,
    this.rightCheek,
    this.leftEye,
    this.rightEar,
    this.leftEar,
    this.rightEye,
    this.noseBase,
    this.bottomMouth,
  });

```

Figure 3.3: Définition des Points de Repère Faciaux

Maintenant, pour extraire les caractéristiques faciales, nous avons utilisé les codes ci-dessous:

NB: Pour travailler avec google_ml_kit, nous avons besoin que l'image capturée soit au format InputImage. Il faut donc convertir l'image capturée dans ce format.

Nous avons déjà présenté le code de capture d'image dans la Partie 1 (Figure 3.1). On a ajouté la ligne : **InputImage inputImage = InputImage.fromFilePath(path);**

À partir de l'objet InputImage, nous pouvons extraire les caractéristiques faciales de l'utilisateur à l'aide du code suivant :

Nous avons ajouté la dépendance suivante dans notre fichier pubspec.yaml : google_ml_kit_face_detection: 0.9.0

```
final FaceDetector _faceDetector = FaceDetector(
  options: FaceDetectorOptions(
    enableLandmarks: true,
    performanceMode: FaceDetectorMode.accurate,
  ),  // FaceDetectorOptions
);  // FaceDetector
FaceFeatures? _faceFeatures;
var image1 = regula.MatchFacesImage();
var image2 = regula.MatchFacesImage();


bool _canAuthenticate = false;

List<dynamic> users = [];

UserModel? loggingUser;

bool isMatching = false;
```

Figure 3.4: Configuration du Détecteur Facial et Variables d'État

Pour chaque utilisateur enregistré, nous sauvegardons les données de son visage sous deux formats :

- Une chaîne de caractères (utilisée pour le flux d'authentification avec flutter_face_api)
- Un modèle de données de caractéristiques faciales (FaceFeature) utilisé pour la logique de comparaison avec google_ml_kit.

Remarque : La chaîne est utilisée pour l'authentification avec flutter_face_api, tandis que le modèle FaceFeature est utilisé pour effectuer la comparaison des visages avec google_ml_kit.

Comparaison avec un visage capturé

En supposant que vous avez déjà stocké les visages enregistrés dans une base de données: Lorsqu'un nouvel utilisateur tente de s'authentifier, nous capturons à nouveau les deux types d'informations :

- Le visage sous forme de chaîne
- Les caractéristiques faciales via le même processus décrit plus haut.

Nous avons ensuite utiliser ces caractéristiques faciales pour comparer avec les données des visages déjà enregistrés.

Avant cela, nous avons déclaré une liste pour stocker les visages similaires :
List similarFaces = [];

Chaque utilisateur soit représenté dans Flutter par un modèle UserModel.Voici un exemple:

```dart
class UserModel {
  String? id;
  String? name;
  String? email;
  String? password;
  String? image;
  FaceFeatures? faceFeatures;
  int? registeredOn;

  UserModel({
    this.id,
    this.name,
    this.email,
    this.password,
    this.image,
    this.faceFeatures,
    this.registeredOn,
  });
```

Figure 3.5: Modèle de Données Utilisateur

3.3 Interfaces Graphiques

Les interfaces de notre application mobile Visionary sont les suivantes :

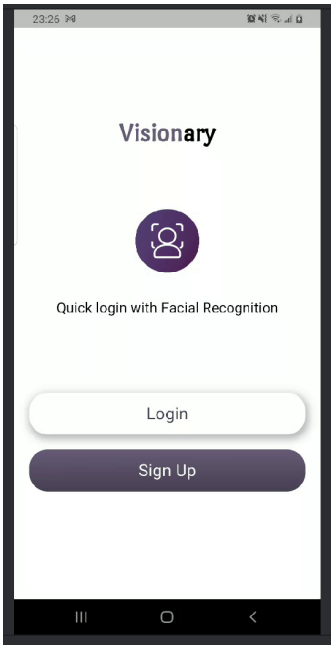

Figure 3.6: Écran d'Accueil de l'Application Visionary

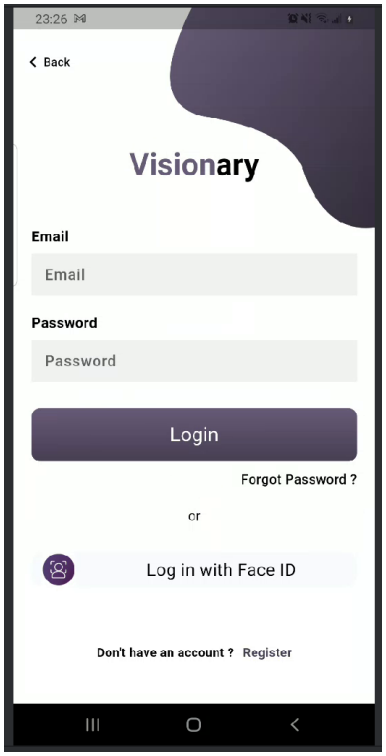

Figure 3.7: Page de Login par Email/Mot de Passe

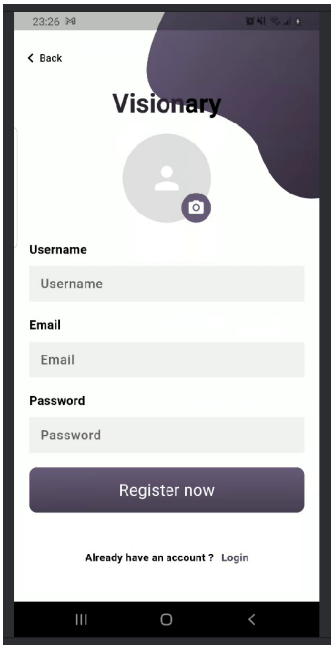

Figure 3.8: Page de Création de Compte

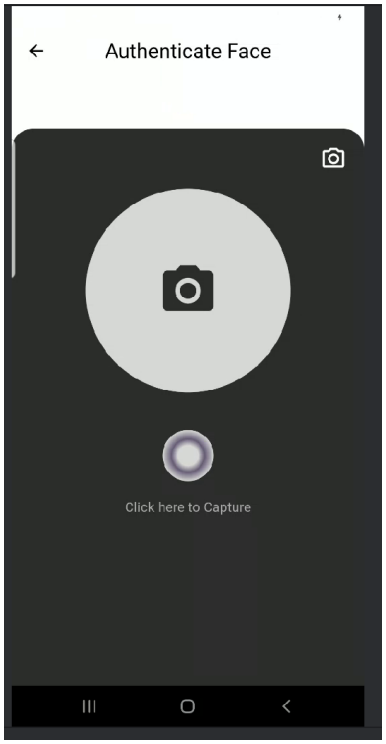

Figure 3.9: Interface de Prise de Photo pour Authentification

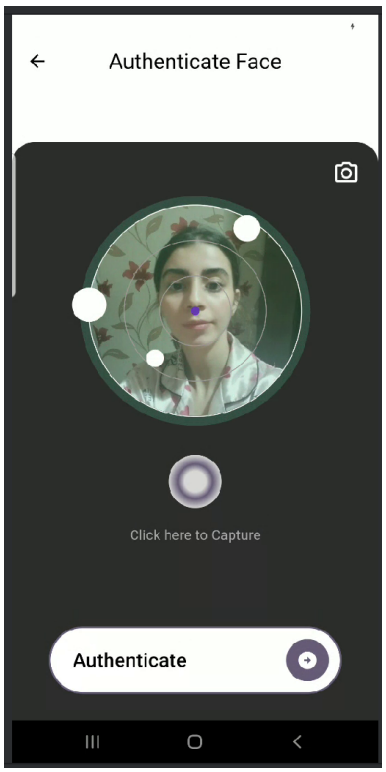

Figure 3.10: Processus d'Authentification par Détection Faciale

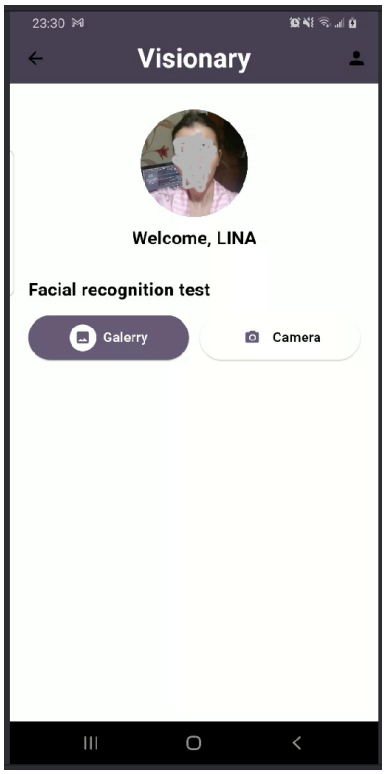

Figure 3.11: Page d'Accueil Utilisateur

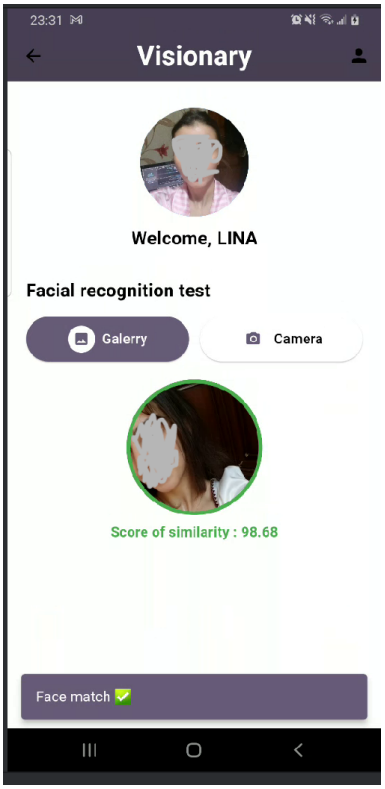

Figure 3.12: Authentification Faciale Validée (Score 98.68%)

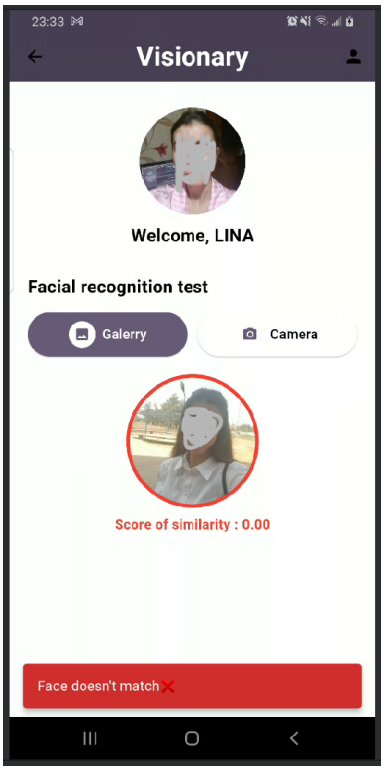

Figure 3.13: Échec de Reconnaissance Faciale

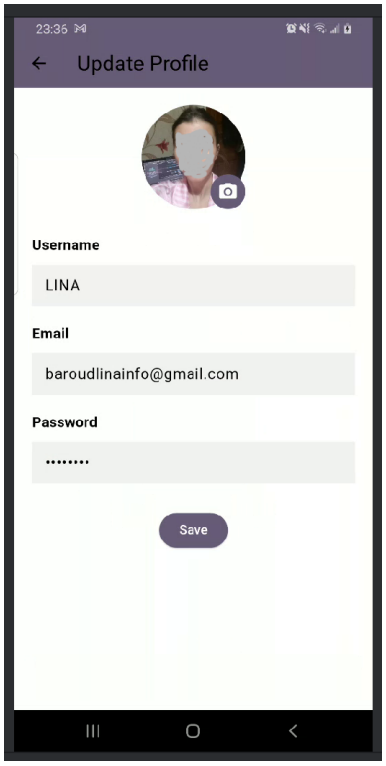

Figure 3.14: Interface de Mise à Jour des Informations Personnelles

CHAPTER 4

CONCLUSION ET PERSPECTIVE

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 :

- Renforcer la sécurité des données (cryptage).
- Ajouter le fonctionnement off-ligne.
- Ajouter l'option de la reconnaissance faciale en temps réel.