1. Introduction

L'application de reconnaissance faciale que vous développez, nommée "Evaluatix", vise à permettre l'authentification des utilisateurs via la reconnaissance faciale. Elle utilise des technologies modernes de Flutter pour les Systèmes Android, intégrée avec **Firebase** pour la gestion des utilisateurs et des données, et des bibliothèques comme **google_ml_kit** pour la détection et la comparaison des visages. Ce rapport détaille les composants techniques, les fonctionnalités, les dépendances, et propose une méthodologie pour capturer l'UI.

2. Technologies et Outils Utilisés

2.1. Environnement de Développement

- **Flutter** : Framework multiplateforme pour la création d'interfaces utilisateur, utilisé avec la version stable la plus récente (par exemple, 3.22.x au 21 mai 2025).
- **Dart** : Langage de programmation principal pour Flutter, utilisé pour la logique métier et l'UI.
- **Android Studio** : Environnements de développement avec extensions Dart et Flutter pour le débogage et l'analyse.
- **Node.js :** Serveur backend pour la génération de tokens, hébergé sur localhost:3000.

2.2. Bibliothèques et Dépendances

Les dépendances listées dans pubspec.yaml incluent :

- **firebase_auth (v5.4.3 ou plus)** : Gestion de l'authentification des utilisateurs (inscription, connexion, déconnexion) avec e-mail et mot de passe.
- **cloud_firestore (v5.4.3 ou plus)** : Stockage des données utilisateur (e-mail, nom d'affichage, URL ou chemin de l'image faciale) dans une base de données NoSQL.
- **google_ml_kit (v0.16.0 ou plus)** : Fournit des capacités de détection faciale via ML Kit de Google, incluant la classification, les points de repère, et le suivi.
- **path_provider (v2.1.1)** : Accès aux répertoires temporaires et documents pour sauvegarder les images localement.
- **path (v1.8.3)** : Manipulation des chemins de fichiers.
- **http (v1.1.0)** : Requêtes HTTP pour télécharger des images depuis Cloudinary ou d'autres services cloud.
- **image_picker (v1.0.4)** : Sélection d'images depuis la galerie ou l'appareil photo.
- **audioplayers (v5.2.1)** : Lecture de sons pour les retours de succès ou d'échec.
- **shared_preferences (v2.2.2)** : Stockage local des chemins d'images.

2.3. Bibliothèques et Dépendances (Node.js)

- **Express.js** : Framework pour gérer les routes HTTP.
- **Firebase Admin SDK** : Génération de tokens personnalisés via admin.auth().createCustomToken.
- **Fichiers de Configuration** :
    - google-services.json : Configuration Firebase côté Android (non utilisé par le serveur, mais présent dans le projet).
    - firebasefile.json: Clé de service Firebase pour le serveur Node.js.
    - package.json : Définit les dépendances (ex. express, firebase-admin).

2.4. Services Externes

- **Firebase** : Utilisé pour l'authentification et la base de données.
- **Cloudinary** : Service de stockage d'images dans le cloud, avec un téléversement via API pour les images faciales (Cloud Name : ********, Upload Preset : **********).

3. Architecture et Composants

3.1. Structure des Fichiers

- **lib/main.dart** : Point d'entrée de l'application, initialisation de Firebase.
- **lib/bienvenue.dart** : Écran principal affichant les informations utilisateur et les options de gestion de la reconnaissance faciale.
- **lib/face_recognition_service.dart** : Service pour la détection et la comparaison des visages.
- **lib/image_picker_service.dart** : Service pour sélectionner des images depuis l'appareil.
- **lib/user_repository.dart** : Gestion des opérations sur les utilisateurs (inscription, mise à jour, récupération).
- **lib/user_model.dart** : Modèle de données pour les utilisateurs.
- **lib/signin.dart** : Écran de connexion (non détaillé dans le code partagé).

Node.js (Serveur)

- **server.js** : Point d'entrée du serveur, gère la route /generateCustomToken.
- **firebasefile.json:** Clé de service Firebase.
- **package.json** et **package-lock.json** : Gestion des dépendances.
- **node_modules** : Bibliothèques installées (ex. express, firebase-admin).

3.2. Fonctionnalités Principales

1. **Authentification Utilisateur** :
    - Inscription avec e-mail et mot de passe via Firebase Auth.
    - Connexion et déconnexion automatique.
2. **Gestion des Images Faciales** :
    - Sélection d'une image via ImagePickerService.
    - Sauvegarde locale ou sur Cloudinary, avec un chemin stocké dans SharedPreferences ou Firestore.
3. **Reconnaissance Faciale** :
    - Détection des visages avec google_ml_kit (points de repère comme yeux, nez, bouche).
    - Comparaison des visages basée sur les distances euclidiennes normalisées.
    - Retours sonores (succès/échec) avec audioplayers.
4. **Interface Utilisateur** :
    - Affichage du profil utilisateur (nom, e-mail, photo).
    - Boutons pour mettre à jour la photo de profil et tester la reconnaissance.
    - Message de feedback après chaque action.
    - Gestion des Paramètres Utilisateur(Modification du nom, mot de passe, e-mail, et photo de profil.)

3.3. Logique Métier

- **Détection et Comparaison** : Utilise les points de repère faciaux pour normaliser et comparer les images.
- **Stockage** : Choix entre stockage local (via path_provider) ou cloud (via Cloudinary et Firestore).
- **Gestion des Erreurs** : Utilisation de try-catch avec debugPrint pour le débogage.

4. Détails Techniques

4.1. Rôle et Fonctionnement du Serveur Node.js

4.1.1. Configuration et Initialisation

- Le serveur est construit avec Express.js et utilise Firebase Admin SDK pour interagir avec Firebase.
- La clé de service (firebasefile.json) est chargée pour initialiser l'Admin SDK

4.1.2. Route /generateCustomToken

- **Objectif** : Générer un token personnalisé pour un utilisateur après validation.
- **Requête** : POST avec un corps JSON contenant userId et email.
- **Logique** :
    - Vérifie que userId et email sont fournis.
    - Requête Firestore pour confirmer l'existence de l'utilisateur et la correspondance de l'e-mail.
    - Génère un token personnalisé avec admin.auth().createCustomToken(userId).
- **Réponse** :
    - Succès : { token: "<token_value>" } (code 200).
    - Échec : { error: "message" } (codes 400 ou 404).

4.1.3. Intégration avec la Reconnaissance Faciale

- **Flux** :
    1. L'utilisateur capture une image via ImagePickerService dans Bienvenue.dart.
    2. L'image est comparée localement avec FaceRecognitionService.compareFaces.
    3. Si la comparaison réussit, l'application envoie une requête POST à http://localhost:3000/generateCustomToken avec userId et email.
    4. Le serveur valide l'utilisateur et renvoie un token.
    5. L'application stocke le token (par exemple, dans SharedPreferences) et ouvre la session.

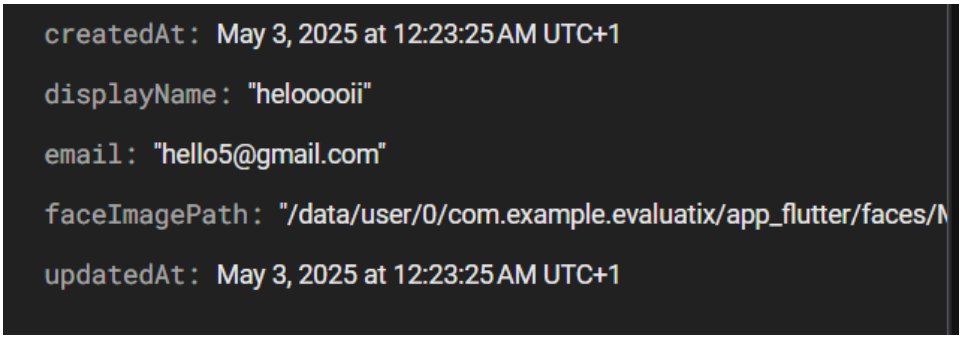
4.2. Processus de Login avec Traitement de l'Image

1. **Capture de l'Image** :
    - L'utilisateur clique sur "Tester la reconnaissance faciale".
    - Une image est sélectionnée via ImagePickerService.
2. **Comparaison Faciale Côté Client** :
    - FaceRecognitionService.compareFaces détecte les visages et calcule la distance normalisée.
    - Si la distance est < 0.5, la reconnaissance est considérée comme réussie.
3. **Requête au Serveur** :
    - Une fois la reconnaissance réussie, l'application envoie userId et email au serveur.
    - Le serveur valide les données dans Firestore et génère un token personnalisé.
4. **Ouverture de la Session** :
    - Le token est reçu et stocké localement.
    - L'utilisateur est redirigé vers Bienvenue.dart si valide, sinon un message d'erreur est affiché.

4.3. Configuration Firebase

Firebase est un composant central de l'application Evaluatix pour l'authentification des utilisateurs, le stockage des données, et la gestion des sessions via des tokens. Voici les aspects clés de son utilisation :

1. **Firebase Auth (v5.4.3 ou plus)** :
    - **Rôle** : Gestion de l'authentification utilisateur (inscription, connexion, déconnexion) avec e-mail et mot de passe.
    - **Intégration avec la reconnaissance faciale** : Lors d'une connexion via Face ID, une fois la reconnaissance faciale réussie côté client (via google_ml_kit), l'application envoie une requête au serveur Node.js pour générer un token personnalisé avec admin.auth().createCustomToken(userId). Ce token est utilisé pour authentifier l'utilisateur auprès de Firebase et ouvrir une session sécurisée.
    - **Processus** :
        - Inscription : Un utilisateur s'inscrit avec un e-mail (ex. "entreprise@gmail.com") et un mot de passe via UserRepository.signUp.
        - Connexion : L'utilisateur se connecte soit par e-mail/mot de passe, soit via Face ID, où le token personnalisé est validé.
        - Déconnexion : L'utilisateur peut se déconnecter depuis la section "Compte", ce qui révoque le token actif.
2. **Cloud Firestore (v5.4.3 ou plus)** :
    - **Rôle** : Stockage des données utilisateur dans une base de données NoSQL.
    - **Données stockées** : Chaque utilisateur a un document contenant des champs comme l'e-mail, le nom d'affichage (ex. "chai"), et l'URL ou le chemin de l'image faciale (selon si le stockage est local ou sur Cloudinary).
    - **Modèle de stockage** :
        - **Structure** : Firestore utilise une structure hiérarchique basée sur des collections et des documents. Dans Evaluatix, une collection probable comme users contient un document par utilisateur, identifié par un userId.



    - **Dynamisme des champs** : Selon la configuration (useCloud), le champ peut être faceImageUrl (pour Cloudinary) ou faceImagePath (pour stockage local via path_provider).

- **Accès aux données** : Le serveur Node.js interroge Firestore pour valider l'existence de l'utilisateur (via userId et e-mail) avant de générer un token personnalisé. L'application client utilise UserRepository.getUser pour récupérer les données utilisateur après connexion.

5. Explication du Code Source

Le code source de l'application "Evaluatix" est organisé en plusieurs fichiers principaux qui gèrent l'authentification, la reconnaissance faciale, et l'interface utilisateur. Voici une explication détaillée de chaque fichier et de son rôle dans le projet.

lib/main.dart

- **Rôle** : Ce fichier est le point d'entrée de l'application Flutter. Il est responsable de l'initialisation de Firebase et du lancement de l'application.
- **Détails** :
    - Initialise Firebase en utilisant les configurations définies dans firebase_options.dart via Firebase.initializeApp().
    - Définit la racine de l'application avec runApp, qui charge l'écran initial (probablement Signin pour la connexion ou un écran de démarrage).
    - Configure les services essentiels comme Firebase Auth et Firestore pour qu'ils soient disponibles dans toute l'application.
- **Logique** :
    - Vérifie si Firebase est correctement initialisé avant de démarrer l'application.
    - Gère les erreurs d'initialisation en affichant des messages de débogage via debugPrint.

lib/signin.dart

- **Rôle** : Ce fichier implémente l'écran de connexion, permettant aux utilisateurs de s'authentifier via e-mail/mot de passe ou Face ID.
- **Détails** :
    - **Classe Signin** : Un StatefulWidget qui gère l'interface de connexion et l'état des champs de saisie.
    - Utilise TextEditingController pour capturer les entrées e-mail (ex. "entreprise@gmail.com") et mot de passe.

    - Implémente des widgets TextField stylisés pour l'e-mail et le mot de passe, avec des bordures arrondies et un design clair (probablement avec un fond gris clair comme 0xffF7F7F9).
    - **Méthode de connexion classique** : Appelle UserRepository.signUp ou une méthode similaire pour authentifier l'utilisateur via Firebase Auth avec e-mail/mot de passe.
    - **Connexion via Face ID** : Fournit un bouton "Passer Par Face ID" qui déclenche le processus de reconnaissance faciale (redirection vers un service ou un écran de capture d'image).
    - **Réinitialisation de mot de passe** : Inclut un lien "Reset" qui envoie un e-mail de réinitialisation via Firebase Auth.
    - **Navigation** : Fournit un lien cliquable "Vous n'avez pas de compte ?" pour naviguer vers un écran d'inscription.
- **Logique** :
    - Si la connexion par e-mail/mot de passe réussit, l'utilisateur est redirigé vers Bienvenue.dart.
    - Si l'utilisateur choisit Face ID, une requête est envoyée au serveur après reconnaissance faciale pour générer un token personnalisé.

lib/bienvenue.dart

- **Rôle** : Ce fichier implémente l'écran principal de l'application, affiché après une authentification réussie.
- **Détails** :
    - **Classe Bienvenue** : Un StatelessWidget ou StatefulWidget qui affiche les informations utilisateur (nom, e-mail, photo) et des options de gestion.
    - Affiche un message de bienvenue personnalisé (ex. "Chai, hello@gmail.com") et une photo de profil si disponible.
    - Inclut des boutons pour "Tester la reconnaissance faciale" et "Déconnexion".
    - **Méthode pour tester la reconnaissance** : Utilise ImagePickerService pour capturer une image, puis appelle FaceRecognitionService.compareFaces pour vérifier l'identité de l'utilisateur.
    - **Déconnexion** : Appelle une méthode de UserRepository pour révoquer le token actif et rediriger vers l'écran de connexion.
- **Logique** :
    - Si la reconnaissance faciale réussit (distance < 0.5), un son de succès est joué via audioplayers, et un message de feedback est affiché via ScaffoldMessenger.
    - Si elle échoue, un son d'échec est joué, et un message d'erreur est affiché.

lib/face_recognition_service.dart

- **Rôle** : Ce fichier contient la logique de détection et de comparaison des visages en utilisant google_ml_kit.
- **Détails** :
    - **Classe FaceRecognitionService** : Gère la détection des visages et la comparaison des données faciales.
    - **Détection des visages** : Utilise FaceDetector de google_ml_kit pour identifier les visages dans une image et extraire les points de repère (yeux, nez, bouche) ainsi que les boîtes englobantes.
    - **Comparaison des visages** : Implémente une méthode compareFaces qui calcule la distance euclidienne normalisée entre les points de repère des deux visages.
    - **Seuil de similarité** : Utilise un seuil (par exemple, 0.5) pour déterminer si les visages correspondent (distance < 0.5 signifie une correspondance réussie).
    - **Fonctions d'aide** :
        - _calculateDistance : Calcule la distance euclidienne entre les points de repère normalisés.
        - _toJson : Convertit les données faciales (points de repère, boîtes englobantes) en un format JSON compatible avec Firestore.
        - _toRect : Convertit les données de boîte englobante stockées en un objet Rect pour la comparaison.
- **Logique** :
    - Si la distance calculée est inférieure au seuil, la reconnaissance est réussie, et l'application envoie une requête au serveur pour générer un token personnalisé.
    - Les données faciales sont stockées dans Firestore pour une comparaison future (via faceImagePath ou faceImageUrl).

lib/image_picker_service.dart

- **Rôle** : Ce fichier fournit des fonctionnalités pour sélectionner et gérer les images utilisées dans la reconnaissance faciale.
- **Détails** :
    - **Classe ImagePickerService** : Gère la sélection d'images via image_picker.
    - **Sélection d'image** : Implémente une méthode pickImage qui permet de choisir une image depuis la caméra ou la galerie.
    - **Stockage local** : Utilise path_provider pour sauvegarder l'image localement et stocke le chemin dans SharedPreferences si useCloud est désactivé.
    - **Stockage cloud** : Si useCloud est activé, téléverse l'image sur Cloudinary (via http) et stocke l'URL dans Firestore.

- **Logique** :
  - Retourne un chemin ou une URL d'image qui est ensuite utilisé par FaceRecognitionService pour la détection faciale.
  - Gère les erreurs de sélection d'image (ex. permissions refusées) avec des messages de débogage.

lib/user_repository.dart

- **Rôle** : Ce fichier gère les opérations liées aux utilisateurs en interagissant avec Firebase Auth et Firestore.
- **Détails** :
  - **Classe UserRepository** : Contient des méthodes pour gérer les utilisateurs.
  - **Inscription (signUp)** : Crée un nouvel utilisateur via FirebaseAuth.instance.createUserWithEmailAndPassword, puis stocke les données (e-mail, nom, image) dans Firestore.
  - **Connexion** : Authentifie l'utilisateur via FirebaseAuth.instance.signInWithEmailAndPassword ou un token personnalisé généré par le serveur après reconnaissance faciale.
  - **Mise à jour (updateUser)** : Permet de modifier les informations utilisateur (nom, photo) dans Firestore.
  - **Récupération (getUser)** : Récupère les données utilisateur depuis Firestore après connexion.
  - **Déconnexion** : Révoque le token actif et déconnecte l'utilisateur via FirebaseAuth.instance.signOut.
- **Logique** :
  - Utilise try-catch pour gérer les erreurs (ex. e-mail déjà utilisé, mot de passe faible) et affiche des messages via debugPrint.
  - Stocke les informations d'authentification localement avec SharedPreferences pour maintenir la session.

lib/user_model.dart

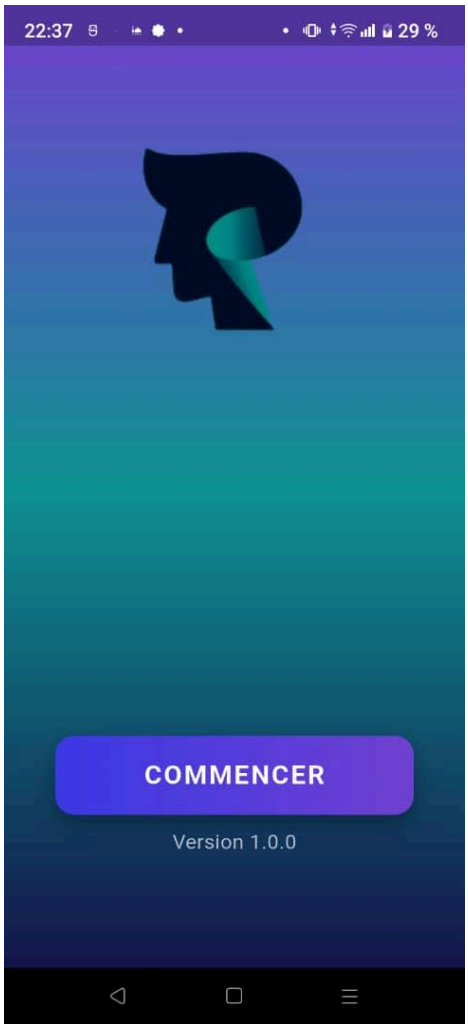
- **Rôle** : Ce fichier définit le modèle de données pour les utilisateurs.
- **Détails** :
  - **Classe UserModel** : Une classe Dart qui représente un utilisateur.
  - Contient des propriétés comme String userId, String email, String displayName, String faceImageUrl ou String faceImagePath, DateTime createdAt, et DateTime updatedAt.
  - Inclut des méthodes pour sérialiser (toJson) et désérialiser (fromJson) les données pour Firestore.
- **Logique** :
  - Utilisé par UserRepository pour structurer les données lors de la sauvegarde ou de la récupération depuis Firestore.

- Permet une gestion dynamique des champs d'image selon la configuration (faceImageUrl pour Cloudinary, faceImagePath pour stockage local).

6.Rapport UI

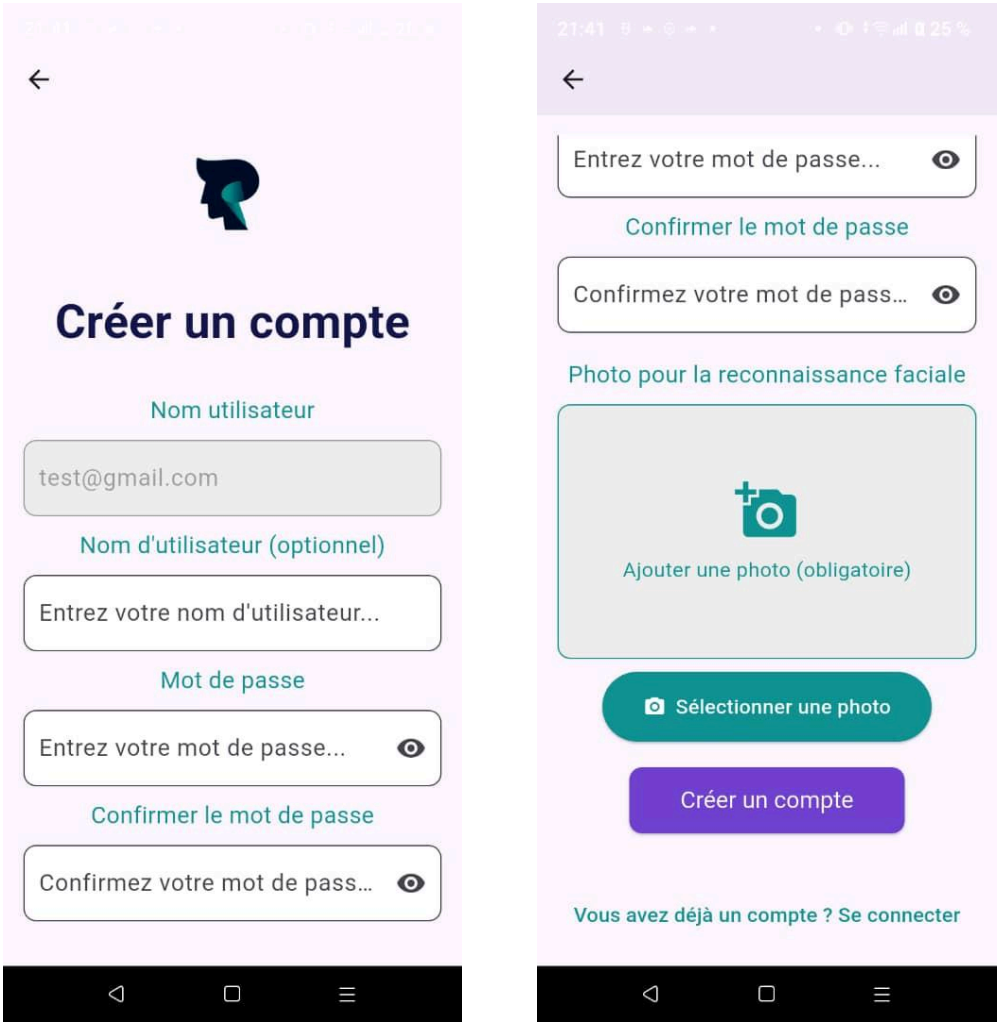
Etape 1 :

l'utilisateur lance l'application "Evaluatix" (version 1.0.0) et voit l'écran d'accueil avec le logo et le bouton "COMMENCER". En cliquant, il accède à l'écran de connexion, entre l'e-mail entreprise@gmail.com, et choisit entre "Next" ou "Passer Par Face ID", avec une option de réinitialisation si besoin, reflétant une authentification sérieuse.



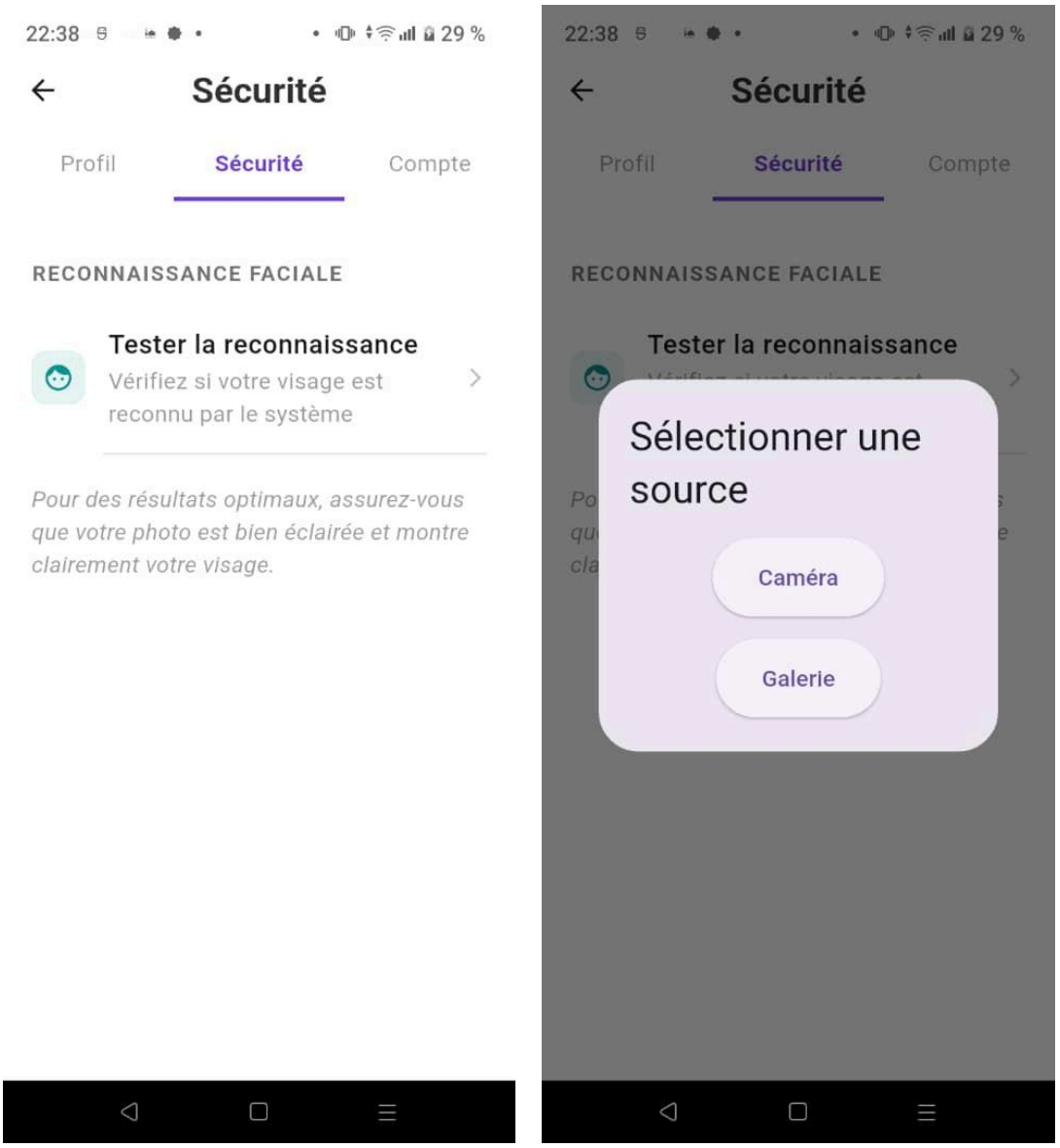
Etape 2 :

Après avoir saisi entreprise@gmail.com, il choisit "Vous n'avez pas de compte ? Reset", accédant au formulaire "Créer un compte" où il doit remplir : "Nom d'utilisateur" (ex. test@gmail.com), "Nom d'utilisateur (optionnel)", "Mot de passe", "Confirmer le mot de passe", et sélectionner une photo via "Sélectionner une photo" pour une authentification faciale sécurisée.



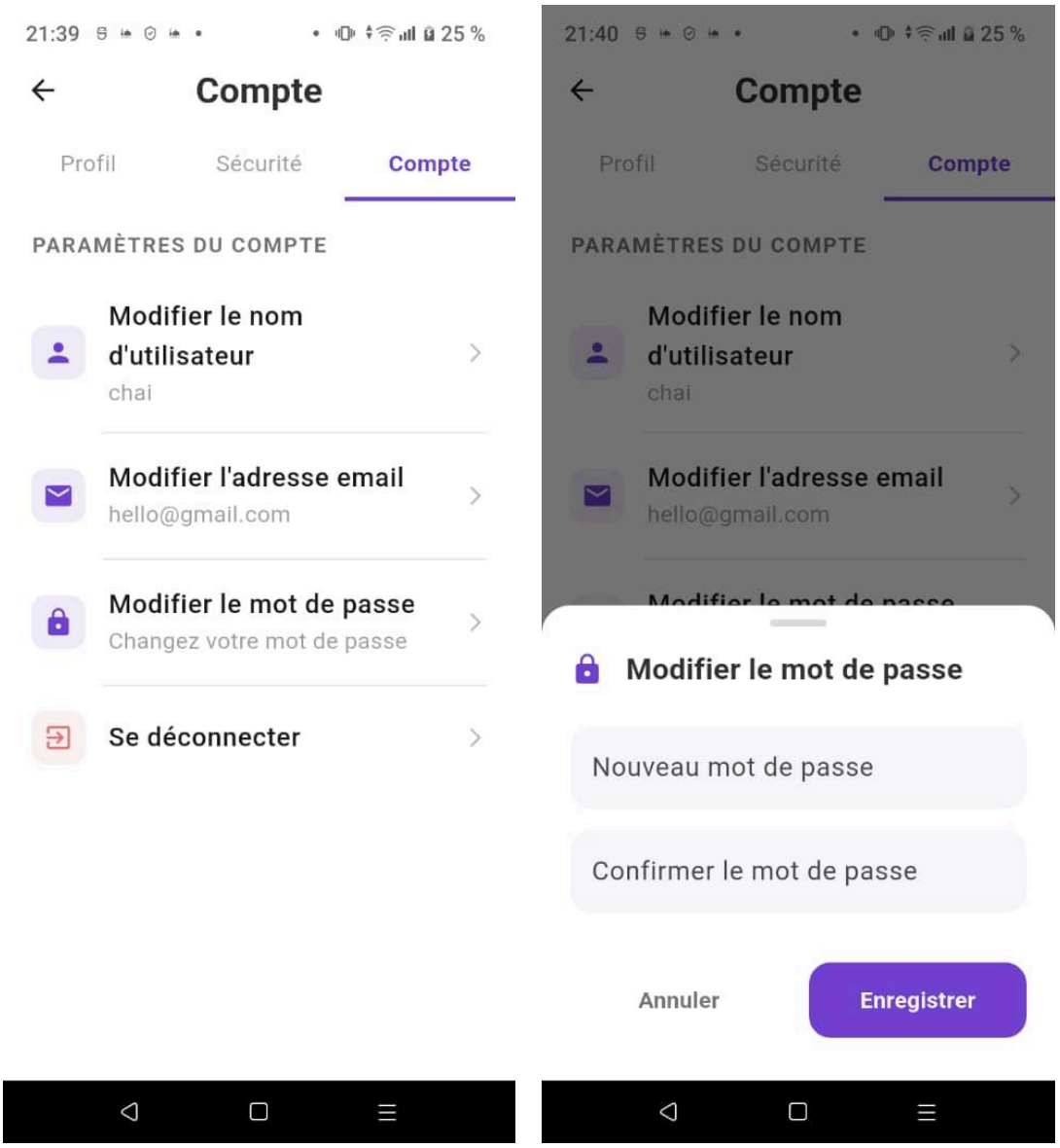
Etape 3 :

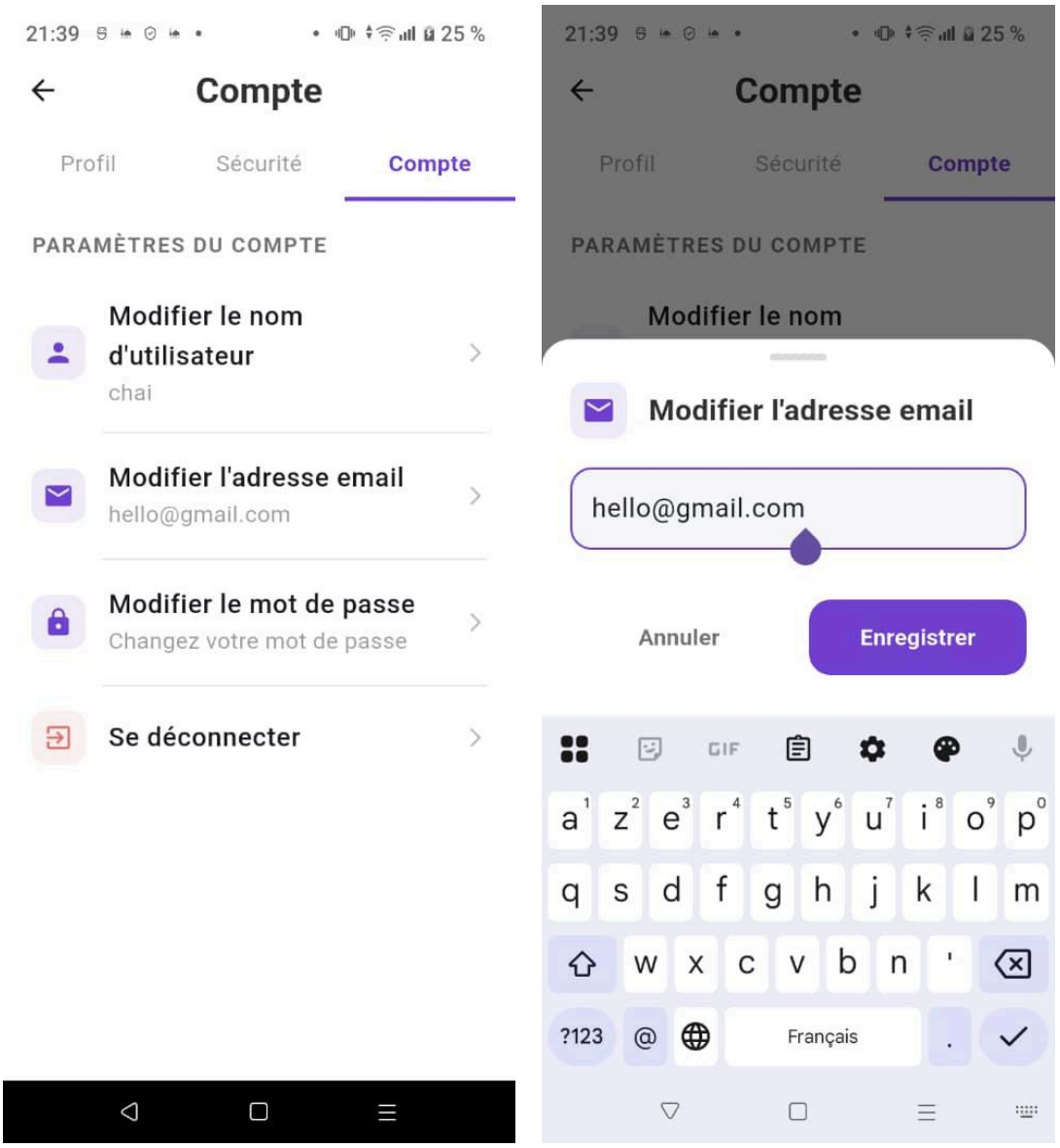
L'utaccède à la section "Sécurité" parmi les options "Profil", "Sécurité", et "Compte". Il clique sur "Tester la reconnaissance faciale", choisissant ensuite entre "Caméra" ou "Galerie" pour vérifier son visage, avec une consigne de bien éclairer son visage pour des résultats optimaux.



Etape 4 :

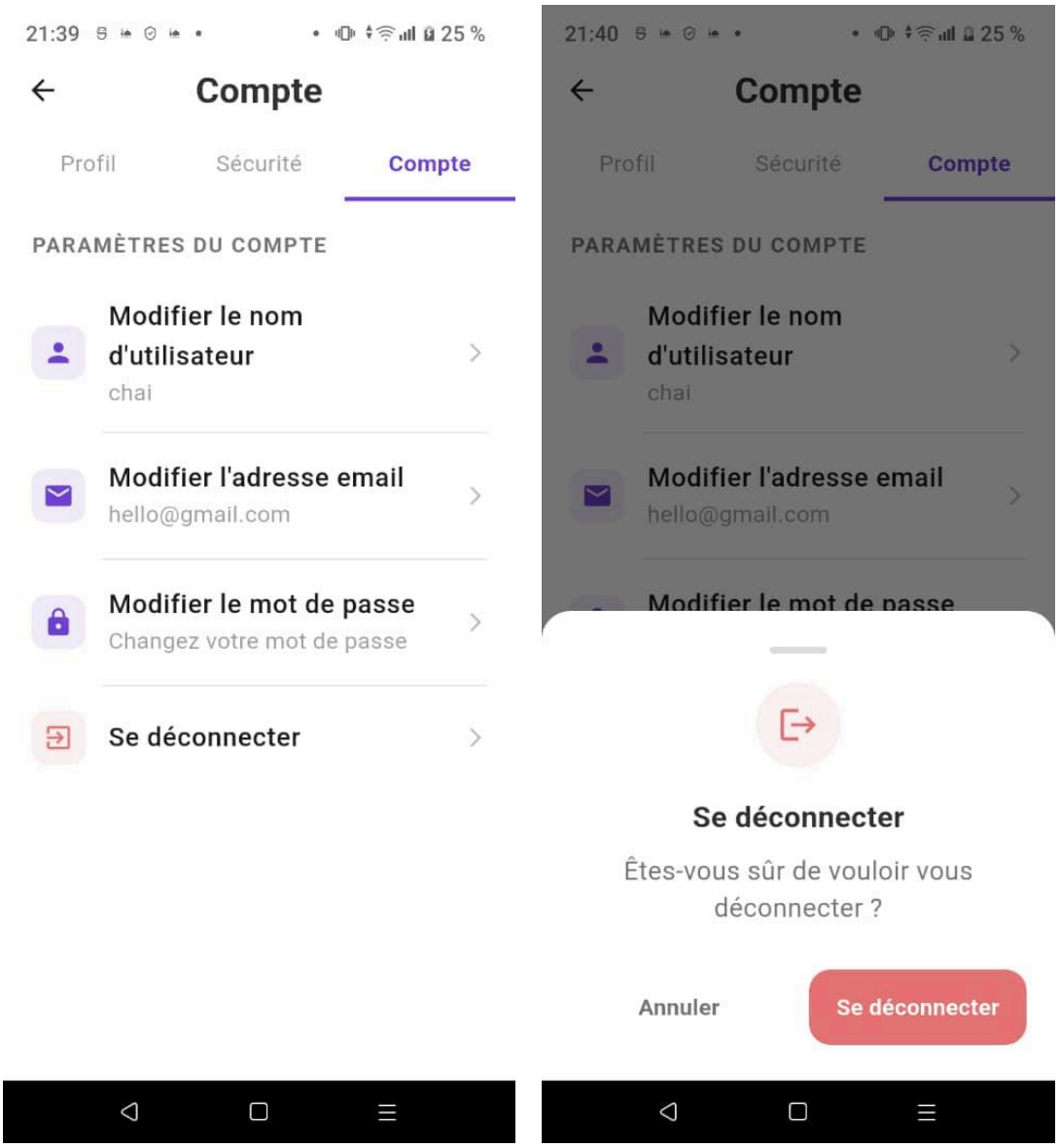
L'utilisateur accède à la section "Compte" parmi "Profil", "Sécurité", et "Compte". Dans "Paramètres du Compte", il peut modifier son nom d'utilisateur (ex. "chai"), son e-mail (ex. "hello@gmail.com"), ou son mot de passe via "Modifier le mot de passe", avec des champs pour le nouveau mot de passe et sa confirmation, puis clique sur "Enregistrer".


21:39
25 %
Compte
Profil
Sécurité
Compte
PARAMÈTRES DU COMPTE
Modifier le nom d'utilisateur
chai
Modifier l'adresse email
hello@gmail.com
Modifier le mot de passe
Changez votre mot de passe
Se déconnecter
21:40
25 %
Compte
Profil
Sécurité
Compte
PARAMÈTRES DU COMPTE
Modifier le nom d'utilisateur
chai
Modifier l'adresse email
hello@gmail.com
Modifier le mot de passe
Nouveau mot de passe
Confirmer le mot de passe
Annuler
Enregistrer



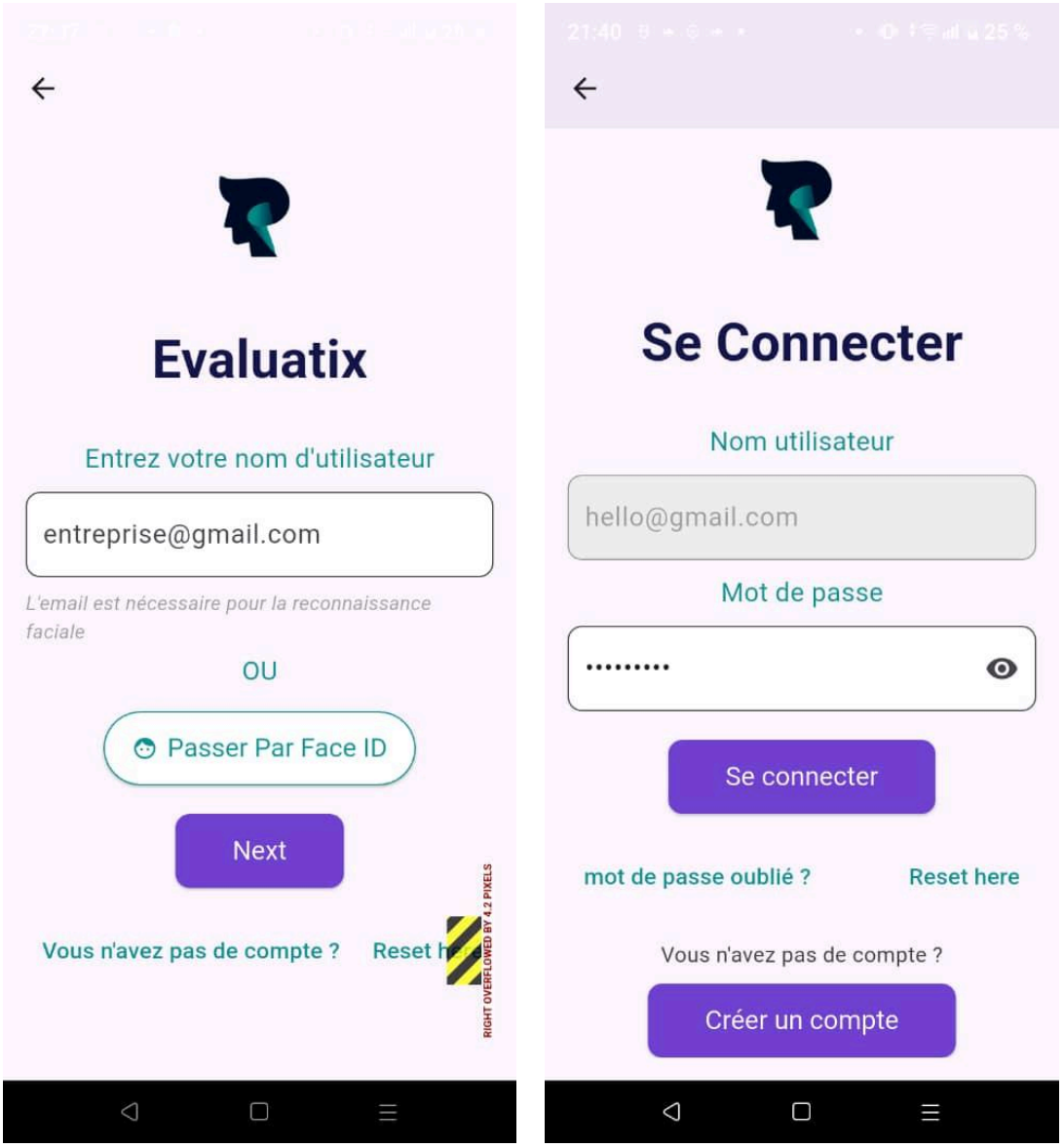
Etape 5 :

L'utilisateur accède à la section "Compte" parmi "Profil", "Sécurité", et "Compte". Dans "Paramètres du Compte", après avoir vérifié son nom (ex. "chai"), e-mail (ex. "hello@gmail.com"), et options de modification, il clique sur "Se déconnecter", confirmant via une popup avec "Annuler" ou "Se déconnecter" pour terminer sa session.



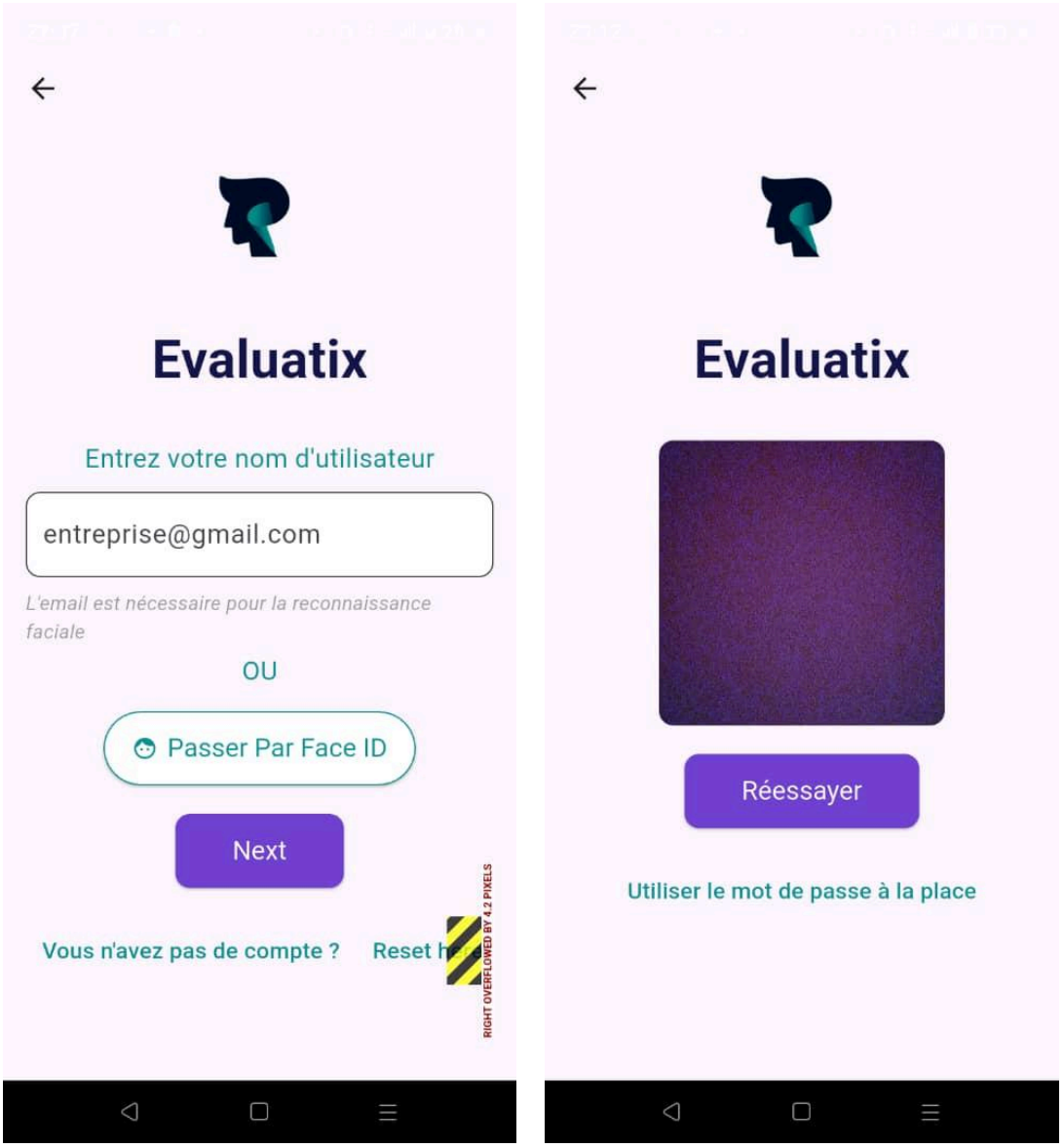
Etape 6 :

Après s'être déconnecté et revient à la page initiale. Il saisit son e-mail (ex. "entreprise@gmail.com") et choisit "Next" pour se connecter avec mot de passe, ignorant l'option "Passer Par Face ID", puis entre son mot de passe (ex. celui associé à "chai") pour accéder à son compte.



Etape 7 :

l'utilisateur revient a la page initiale, entre son e-mail (ex. "entreprise@gmail.com"), et choisit "Passer Par Face ID". L'application scanne son visage via la caméra ; un message s'affiche ensuite : "Reconnaissance réussie" en vert si le Face ID fonctionne, ou "Échec de la reconnaissance" en rouge si elle échoue, informant l'utilisateur du résultat.

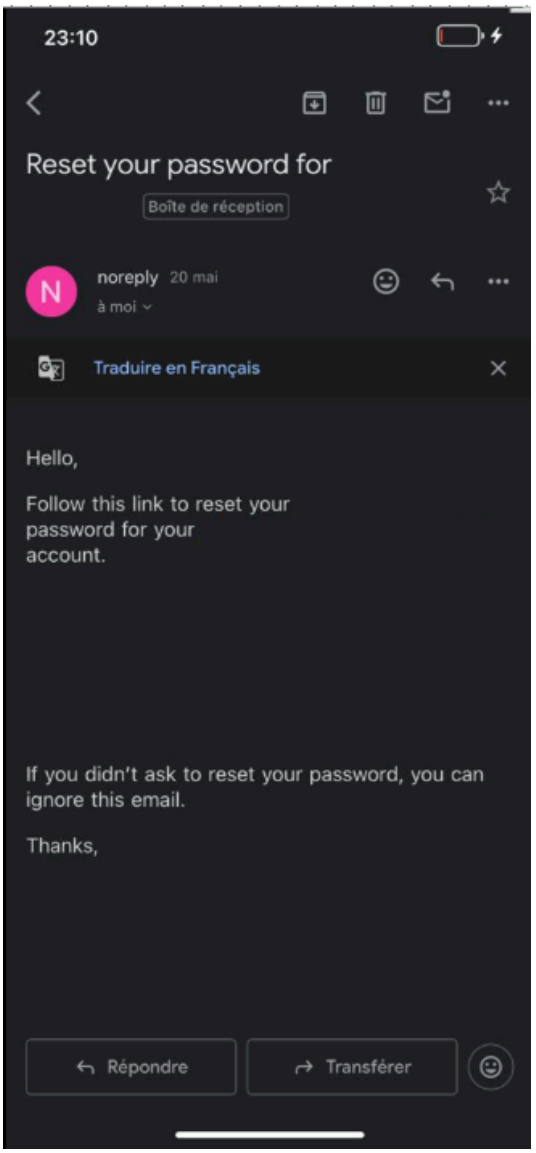


Etape 8 :

Après une connexion réussie. La section "Profil" s'affiche avec son nom (ex. "chai") et sa photo actuelle ; il peut soit tester son Face ID via "Tester la reconnaissance faciale", recevant un résultat ("Réussie" ou "Échec"), soit modifier sa photo de profil en cliquant sur "Mettre à jour ma photo de profil" pour en charger une nouvelle.

Etape 9 :

Dans le cas où l'utilisateur a oublié son mot de passe, clique sur "Reset here" sur la page initiale après entrer son e-mail (ex. "entreprise@gmail.com"). Un e-mail est envoyé avec un lien de réinitialisation, qu'il consulte dans sa boîte de réception pour créer un nouveau mot de passe.



7. Conclusion

Le projet **Evaluatix** met en œuvre une solution d'authentification par reconnaissance faciale, alliant efficacité, sécurité et simplicité d'utilisation. Grâce à l'intégration de technologies telles que Flutter, Firebase et Google ML Kit, l'application assure une identification fiable et rapide des utilisateurs. L'ajout d'un serveur Node.js pour la génération de tokens renforce la sécurité côté backend. Ce travail nous a permis de développer des compétences concrètes en développement mobile sécurisé. Evaluatix constitue ainsi une base solide pour des solutions biométriques futures.