ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Rapport projet :

application de gestion de mots de passe

Ayman LAROUR
5IIR16

Table des matières

Table de figures

Introduction

Dans le cadre de ce projet, j'ai eu l'opportunité de concevoir et de développer une application multiplateforme complète en m'appuyant sur des technologies modernes et des pratiques avancées. Ce travail m'a conduit à relever divers défis techniques, allant de l'intégration de mécanismes de sécurité robustes — tels que le chiffrement des données et l'utilisation de Firebase — à la création d'une interface utilisateur fluide et ergonomique grâce à Flutter. J'ai également consolidé mes compétences en gestion de version en utilisant Git pour structurer efficacement l'évolution du projet.

Ce projet m'a permis d'intégrer des fonctionnalités avancées tout en optimisant l'expérience utilisateur afin de la rendre plus intuitive. L'ensemble de ces réalisations illustre ma maîtrise des technologies multiplateformes ainsi que ma capacité à gérer une base de données et à intégrer des services cloud.

En somme, cette expérience m'a offert l'occasion de renforcer mon savoir-faire technique tout en développant des compétences essentielles comme l'autonomie, l'organisation et la rigueur. Elle me prépare à aborder des projets plus ambitieux et à relever de nouveaux défis technologiques dans l'avenir.

Présentation du projet

I. Contexte du projet

Pourquoi une application de gestion de mots de passe ?

Dans un contexte où chaque utilisateur doit gérer un nombre croissant de comptes en ligne — réseaux sociaux, plateformes professionnelles, services bancaires ou achats en ligne — la sécurité des mots de passe devient un enjeu majeur. Pourtant, beaucoup continuent d'utiliser des mots de passe simples ou identiques sur plusieurs plateformes, augmentant considérablement les risques de piratage et de fuite de données.

Face à ces problématiques, j'ai choisi de développer une application de gestion de mots de passe. L'objectif est de proposer un outil individuel, simple d'utilisation, sécurisé et fiable, permettant de centraliser, organiser et protéger ses mots de passe. Le projet s'appuie sur des techniques modernes de chiffrement afin de garantir la confidentialité et la sûreté des données personnelles.

II. Objectifs du projet

- ✓ **Sécurisation et gestion des données personnelles**

L'objectif principal de ce projet individuel est de concevoir une application capable de répondre aux besoins suivants :

1. **Fournir un espace sécurisé pour stocker et gérer les identifiants des utilisateurs.**
2. **Générer automatiquement des mots de passe robustes**, conformes aux standards actuels de sécurité.
3. **Mettre à disposition une interface intuitive et ergonomique**, permettant une utilisation fluide même pour les utilisateurs non technophiles.
4. **Garantir la confidentialité des données grâce à un chiffrement avancé**, protégeant ainsi les mots de passe contre tout accès non autorisé.

Besoins Fonctionnelles

III. Étude des besoins et fonctionnalités à intégrer

Avant de commencer le développement, j'ai réalisé une analyse approfondie des besoins des utilisateurs afin d'identifier les fonctionnalités indispensables à une application de gestion de mots de passe. Les besoins prioritaires retenus sont les suivants :

- **Authentification sécurisée** pour protéger l'accès aux données sensibles.
- **Gestion complète des comptes enregistrés**, incluant l'ajout, l'affichage, la modification et la suppression.
- **Génération automatique de mots de passe forts**, afin d'encourager de bonnes pratiques de sécurité et d'éviter la réutilisation de mots de passe.
- **Affichage sécurisé des mots de passe**, masqués par défaut pour prévenir toute consultation non autorisée.

A. Gestion des utilisateurs (authentification et comptes)

L'authentification constitue un élément central de l'application. J'ai mis en place un système permettant à l'utilisateur de :

- **Créer un compte** en utilisant une adresse email, un numéro de téléphone et un code confidentiel.
- **Se connecter** afin d'accéder aux comptes qu'il a enregistrés.

B. Gestion des mots de passe

Une fois connecté, l'utilisateur dispose d'un espace où il peut gérer l'ensemble de ses mots de passe :

- **Ajouter de nouveaux comptes**, en renseignant :
  - Le nom de l'application ou du service (ex. : *Gmail*).
  - L'identifiant de connexion (ex. : *user123@gmail.com*).
  - Le mot de passe, soit généré automatiquement, soit saisi manuellement.
- **Mettre à jour ou supprimer** les comptes déjà existants.

Afin d’assurer une sécurité maximale, **tous les mots de passe sont chiffrés avant leur stockage dans Firebase**, ce qui garantit leur confidentialité même en cas de tentative d’accès non autorisé à la base de données.

IV. Présentation des Fonctionnalités Principales

• Connexion et authentification utilisateur

L’utilisateur accède à l’application en saisissant son email et son mot de passe. En cas d’erreur lors de la connexion, un message explicite apparaît afin d’indiquer le problème.

• Liste des comptes enregistrés

Les différents comptes sauvegardés sont affichés sous forme de liste.
Exemple :
– Nom du compte : *Gmail*
– Identifiant : *user123@gmail.com*
– Mot de passe : ******** (masqué par défaut)

• Ajout de compte avec génération automatique de mots de passe

L’application permet d’ajouter un nouveau compte et de générer automatiquement un mot de passe complexe et unique, par exemple : *Ht@93Xkz_*

• Chiffrement des mots de passe

Avant leur stockage dans Firebase, tous les mots de passe sont chiffrés afin de garantir leur confidentialité, même en cas d’accès non autorisé à la base de données.

V. Conception Générale

C. Diagramme de cas d'utilisation

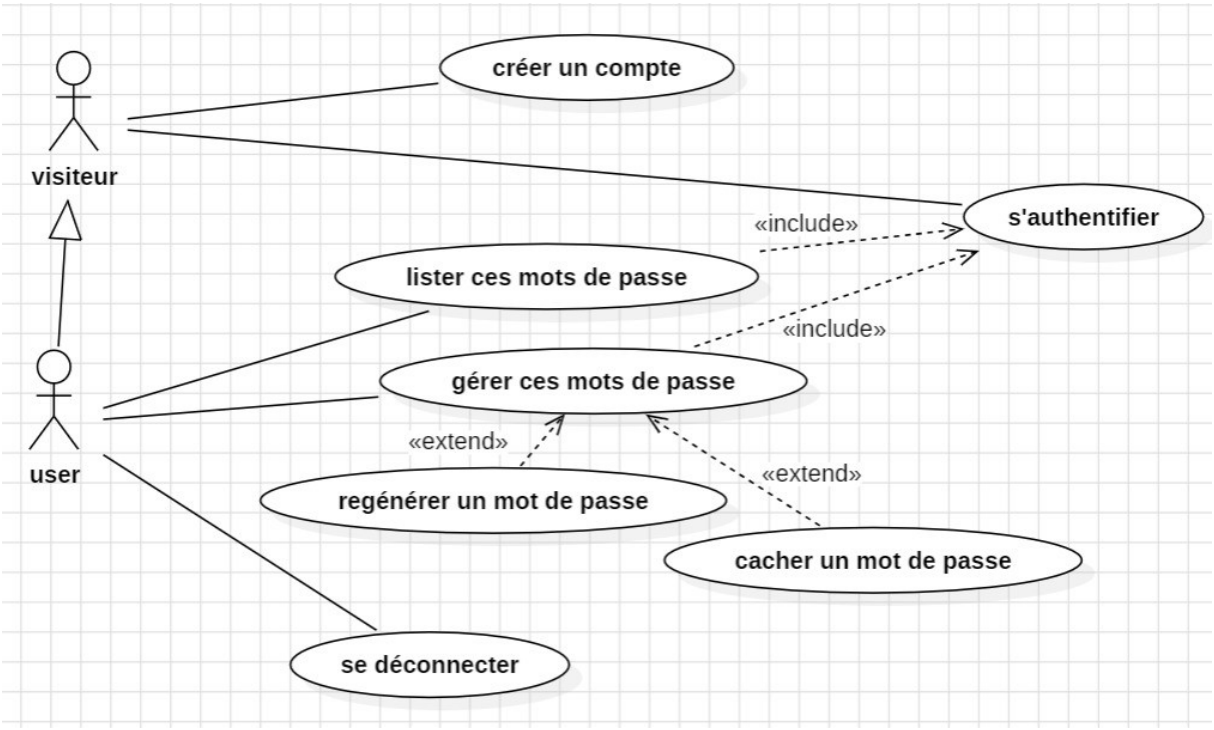


Figure 1 : Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation illustre les interactions entre deux types d'acteurs : **visiteur** et **utilisateur**. Il présente de manière claire les différentes fonctionnalités offertes par le système de gestion des mots de passe.

1. **Acteurs :**
    - **Utilisateur (User)** : Personne authentifiée disposant d'un accès complet aux fonctionnalités de gestion des mots de passe.
2. **Cas d'utilisation :**
    - **Créer un compte :**
      Permet aux visiteurs de s'inscrire pour devenir des utilisateurs du système.
    - **S'authentifier :**
      Cas d'utilisation central permettant aux visiteurs et aux utilisateurs de se connecter afin d'accéder aux fonctionnalités sécurisées.

- **Lister ses mots de passe :**
  Permet aux utilisateurs d'afficher l'ensemble des mots de passe stockés.
- **Gérer ses mots de passe :**
  Inclut la mise à jour, la suppression ou l'organisation des différents mots de passe enregistrés.
- **Cacher un mot de passe :**
  Fonctionnalité optionnelle permettant de masquer temporairement un mot de passe afin de protéger les données sensibles.
- **Régénérer un mot de passe :**
  Offre la possibilité de générer automatiquement un nouveau mot de passe sécurisé si nécessaire.
- **Se déconnecter :**
  Permet à l'utilisateur de quitter sa session en toute sécurité.

3. **Relations entre cas d'utilisation :**

- **Include (Inclure) :**
  Le cas d'utilisation **S'authentifier** est inclus dans **Créer un compte** et **Lister ses mots de passe**, car l'accès à ces fonctionnalités nécessite une authentification préalable.
- **Extend (Étendre) :**
  Le cas **Cacher un mot de passe** est une extension de **Gérer ses mots de passe**, car il ajoute un comportement supplémentaire destiné à renforcer la confidentialité.

D. Diagramme de classe

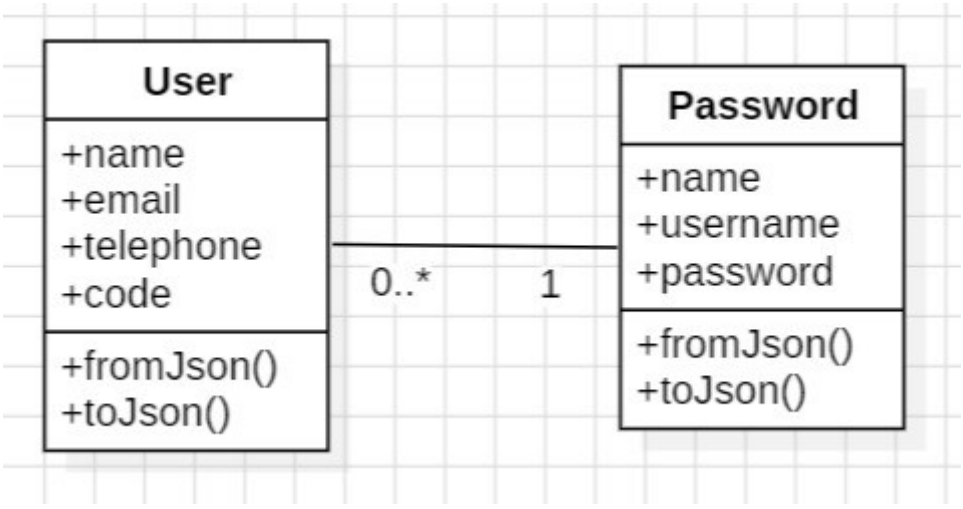
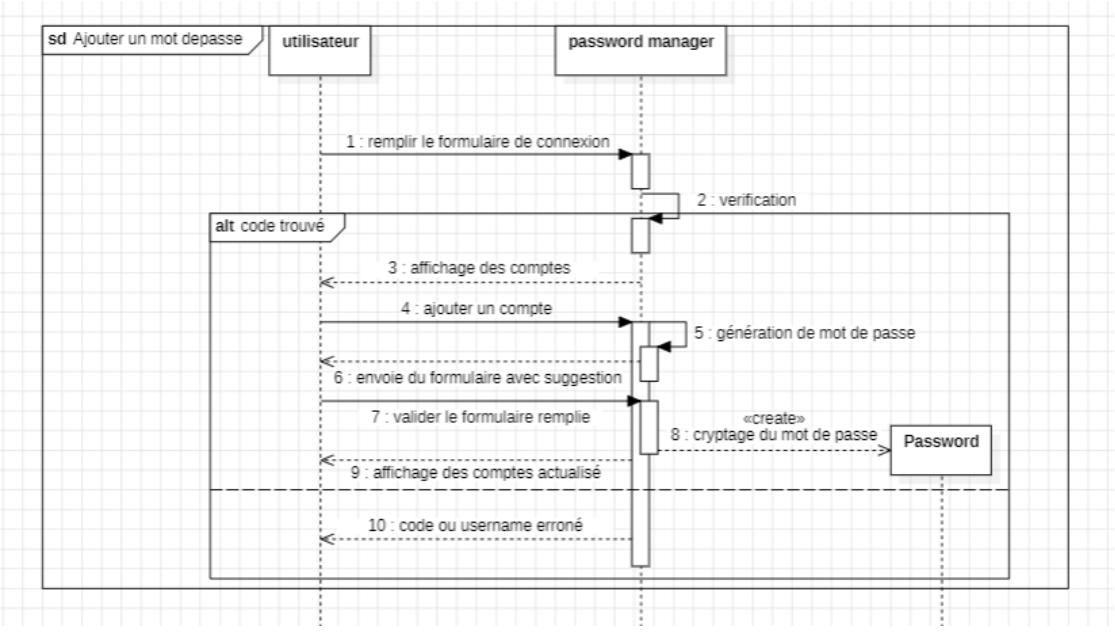


Figure 2 : Diagramme de classe

Le diagramme de classe met en évidence deux classes principales : **User** et **Password** .

1. **Classe User :**
    - **Attributs :**
        - *name* : Nom complet de l'utilisateur.
        - *email* : Adresse e-mail de l'utilisateur.
        - *telephone* : Numéro de téléphone.
        - *code* : Code confidentiel de l'utilisateur.
    - **Méthodes :**
        - *fromJson()* : Initialise un objet User à partir d'une structure JSON.
        - *toJson()* : Convertit l'objet User en un format JSON.
2. **Classe Password :**
    - **Attributs :**
        - *name* : Libellé du mot de passe (ex. : « Gmail », « E-mail pro »).
        - *username* : Identifiant ou nom d'utilisateur associé.
        - *password* : Le mot de passe proprement dit.
    - **Méthodes :**
        - *fromJson()* : Crée un objet Password depuis une structure JSON.
        - *toJson()* : Convertit l'objet Password en JSON.
3. **Relation :**
    - Une relation de type **0..*** lie la classe User à la classe Password.
    - Ainsi, un utilisateur peut posséder zéro, un ou plusieurs mots de passe, tandis qu'un mot de passe est toujours associé à un seul utilisateur.

E. Diagramme de séquence



1) *Acteurs et objets impliqués*

- **Utilisateur** : l'acteur qui interagit avec l'application.
- **Password Manager** : l'interface principale et la logique métier de l'application.
- **Password** : l'objet représentant le mot de passe à gérer.

2) *Étapes détaillées*

1. **Remplissage du formulaire de connexion**
    - L'utilisateur saisit son **email et mot de passe** dans le formulaire de connexion de l'application.
2. **Vérification des identifiants**
    - Le Password Manager envoie les informations à la logique de vérification.
    - S'il y a un problème (code ou username incorrect), un message d'erreur est renvoyé à l'utilisateur.
3. **Affichage des comptes existants**
    - Après une authentification réussie, le Password Manager récupère et affiche les comptes et mots de passe déjà enregistrés.
4. **Ajout d'un compte**
    - L'utilisateur clique sur le bouton pour ajouter un nouveau mot de passe.
    - Le formulaire d'ajout de compte s'affiche.
5. **Génération automatique du mot de passe**
    - Le Password Manager peut générer un mot de passe complexe et unique pour l'utilisateur.
6. **Envoi du formulaire avec suggestion**
    - L'utilisateur peut remplir le formulaire (nom du compte, identifiant, mot de passe généré ou saisi manuellement).
    - Le formulaire est envoyé au Password Manager pour traitement.
7. **Validation du formulaire**
    - Le Password Manager vérifie que toutes les informations sont correctes et complètes.
8. **Chiffrement du mot de passe**
    - Avant l'enregistrement, le mot de passe est chiffré pour assurer la sécurité des données.

9. **Affichage des comptes mis à jour**
    - Le Password Manager enregistre le nouveau mot de passe et met à jour l'affichage des comptes pour refléter l'ajout.
10. **Gestion des erreurs**

- Si le code ou le username fourni est incorrect, un message d'erreur est affiché à l'utilisateur (**alt branch**).

VI. Structure de l'application

L'application est structurée en trois couches principales :

- **Interface utilisateur** : regroupe les écrans de connexion, la liste des comptes et les formulaires.
- **Logique métier** : prend en charge les fonctionnalités principales, telles que la création, la lecture, la mise à jour et la suppression (CRUD) des comptes, ainsi que la génération automatique de mots de passe.
- **Firebase** : assure le stockage sécurisé des données et la gestion des utilisateurs.

VII. Structuration du projet en une architecture modulaire:

L'architecture du projet a été conçue pour assurer une organisation claire et faciliter la maintenance. Le répertoire principal **lib** est structuré en plusieurs sous-dossiers, chacun ayant un rôle précis :

1. **models** :
   Contient les modèles de données essentiels pour l'application.
   - **Password.dart** : Gère les informations liées aux mots de passe, incluant la logique de sécurité ou de validation.
   - **User.dart** : Représente les utilisateurs, avec leurs attributs et comportements associés.
2. **pages** :
   Regroupe les différentes pages constituant l’interface utilisateur.
   - **HomePage.dart** : Page principale, point d’entrée après l’authentification.
   - **LoginPage.dart** : Interface dédiée à la connexion des utilisateurs.
3. **services** :
   Contient les services et utilitaires nécessaires au fonctionnement de l'application.
   - **EncryptionHelper.dart** : Gère le chiffrement des données sensibles.
   - **firestore.dart** : Intègre la base de données Firebase pour les opérations CRUD.
   - **firebase_options.dart** : Fichier de configuration Firebase avec les paramètres spécifiques au projet.

Cette structure modulaire assure une séparation claire des responsabilités, rendant le code plus lisible, maintenable et évolutif. Elle facilite également la collaboration, permettant à chaque développeur de travailler sur un composant spécifique sans interférer avec les autres.

Conclusion

Ce projet m’a permis de concevoir et de développer une application complète et innovante, alliant sécurité et ergonomie. En combinant le chiffrement des données et les services Firebase pour assurer une protection optimale, et Flutter pour proposer une interface utilisateur intuitive, j’ai pu créer un produit à la fois robuste et facile à utiliser. Ce travail m’a permis de renforcer mes compétences en développement d’applications mobiles multiplateformes, en gestion de bases de données et en intégration de services cloud, tout en améliorant ma capacité à gérer un projet complexe de manière autonome. J’ai également enrichi l’application avec des fonctionnalités avancées telles que l’authentification biométrique, le partage sécurisé de mots de passe et l’optimisation de l’interface pour offrir une expérience utilisateur fluide et conviviale. En somme, cette expérience a été particulièrement formatrice sur les plans technique et professionnel, et m’a préparé à relever des défis technologiques plus ambitieux à l’avenir.

Annexes

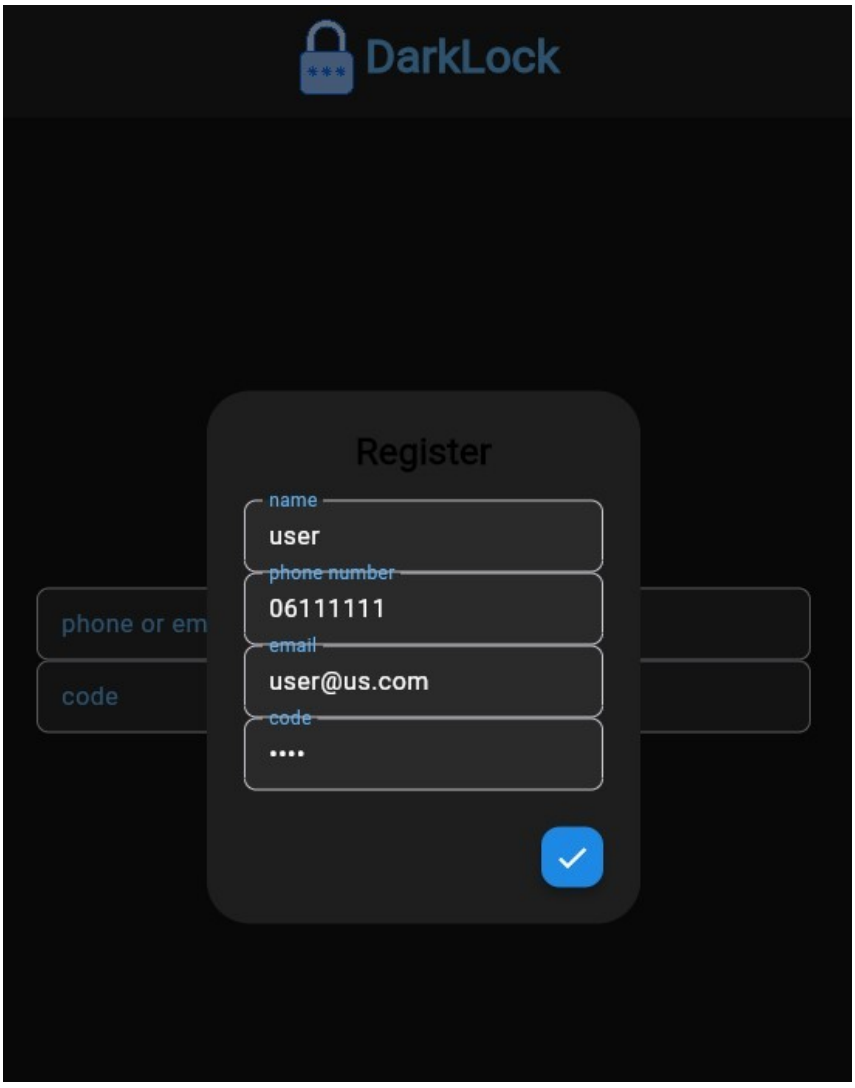


Figure 1 : Page de Register

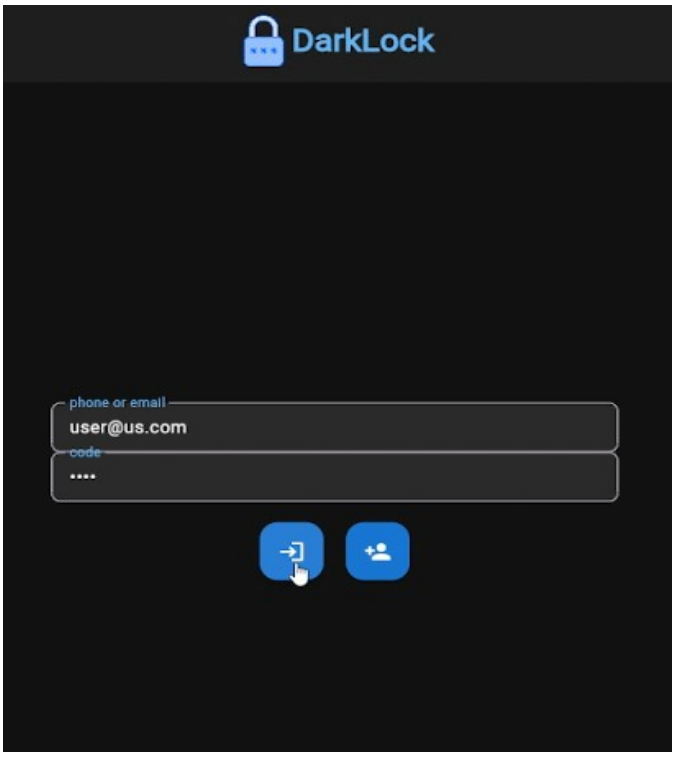


Figure 2 : Page de Connexion (Login)

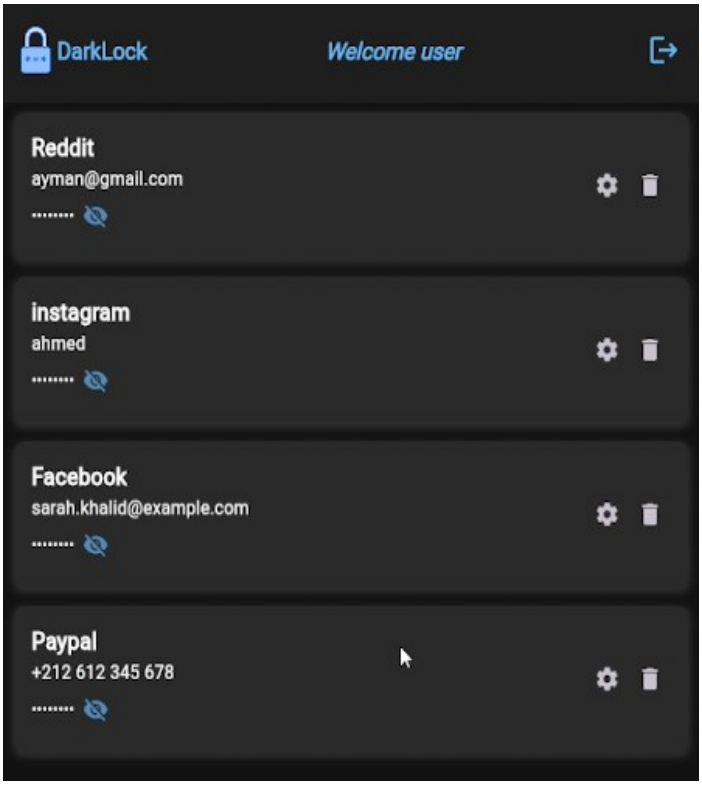


Figure 3 : Dashboard

Figure 4 : add button

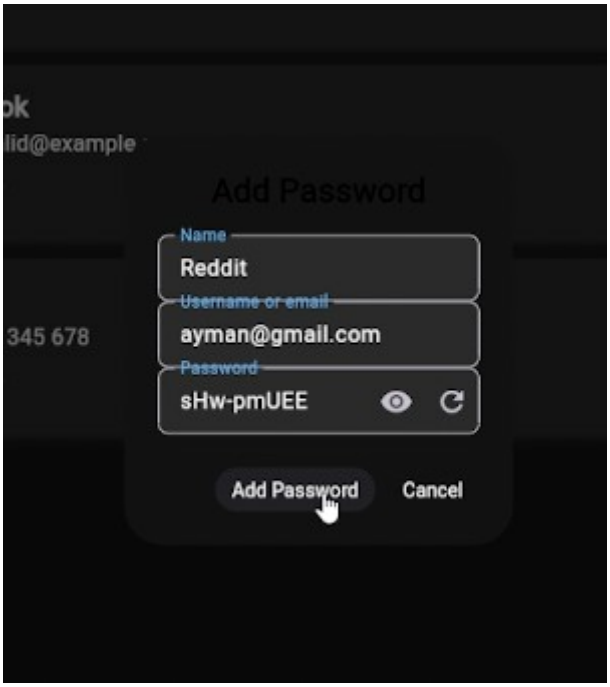


Figure 5 : Adding password form

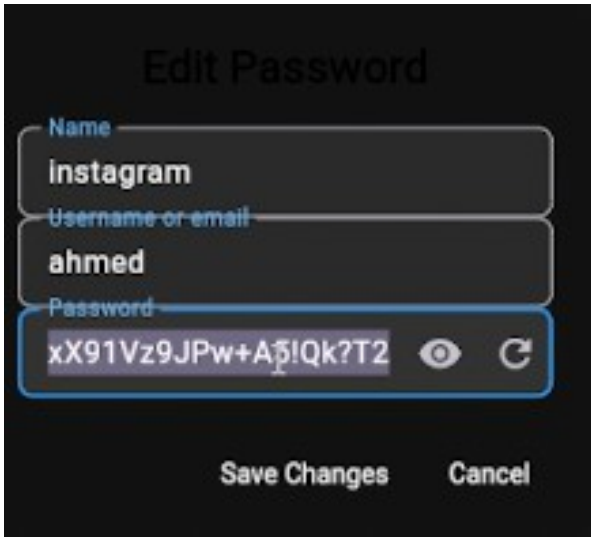


Figure 6 : Editing password form