StormPass

Dans cet page nous détaillerons la manière dont j'ai créer l'application StormPass avec Flutter.

Résumé et Fonctionnalités

StormPass est une application de gestion de mots de passe sécurisée, conçue pour générer, stocker et gérer les mots de passe de manière fiable. Elle combine une interface conviviale avec des options de sécurité avancées pour assurer la protection des données sensibles. Voici les fonctionnalités clés et le principe de fonctionnement de l'application.

Fonctionnalités principales

1. **Génération de mots de passe sécurisés**
    - Génération de mots de passe aléatoires avec options de personnalisation (minuscules, majuscules, chiffres, symboles).
    - Utilisation de l'API de random.org stockée directement dans des variables d'environnement, pour un niveau d'aléatoire élevé, garantissant des mots de passe robustes.
2. **Coffre-fort sécurisé**

- Sauvegarde des mots de passe dans un coffre-fort chiffré, protégé par authentification biométrique.
- Accès rapide et sécurisé aux mots de passe enregistrés.

3. **Calcul de l'entropie des mots de passe**
   - Affichage d'un indicateur d'entropie pour chaque mot de passe, permettant de visualiser sa robustesse.
4. **Sauvegarde automatique locale**
   - Option pour sauvegarder automatiquement les mots de passe chiffrés dans le stockage local de l'application, assurant la récupération en cas de besoin.
5. **Synchronisation avec le cloud**
   - Option de synchronisation des mots de passe vers le cloud pour une protection supplémentaire.
   - Synchronisation sécurisée après authentification.
6. **Paramètres personnalisables**
   - Options pour activer ou désactiver la sauvegarde automatique locale et la synchronisation cloud, accessibles depuis la page des paramètres.

Principe de fonctionnement

StormPass est conçu autour de plusieurs principes de sécurité :

- **Chiffrement AES** : Chaque mot de passe est chiffré avec une clé AES avant d'être stocké.
- **Authentification biométrique** : Requise pour accéder au coffre-fort et effectuer des actions sensibles.
- **Sauvegardes sécurisées** : Les mots de passe sont régulièrement sauvegardés de manière chiffrée.
- **API random.org** : Assure la génération de mots de passe vraiment aléatoires pour une sécurité renforcée.

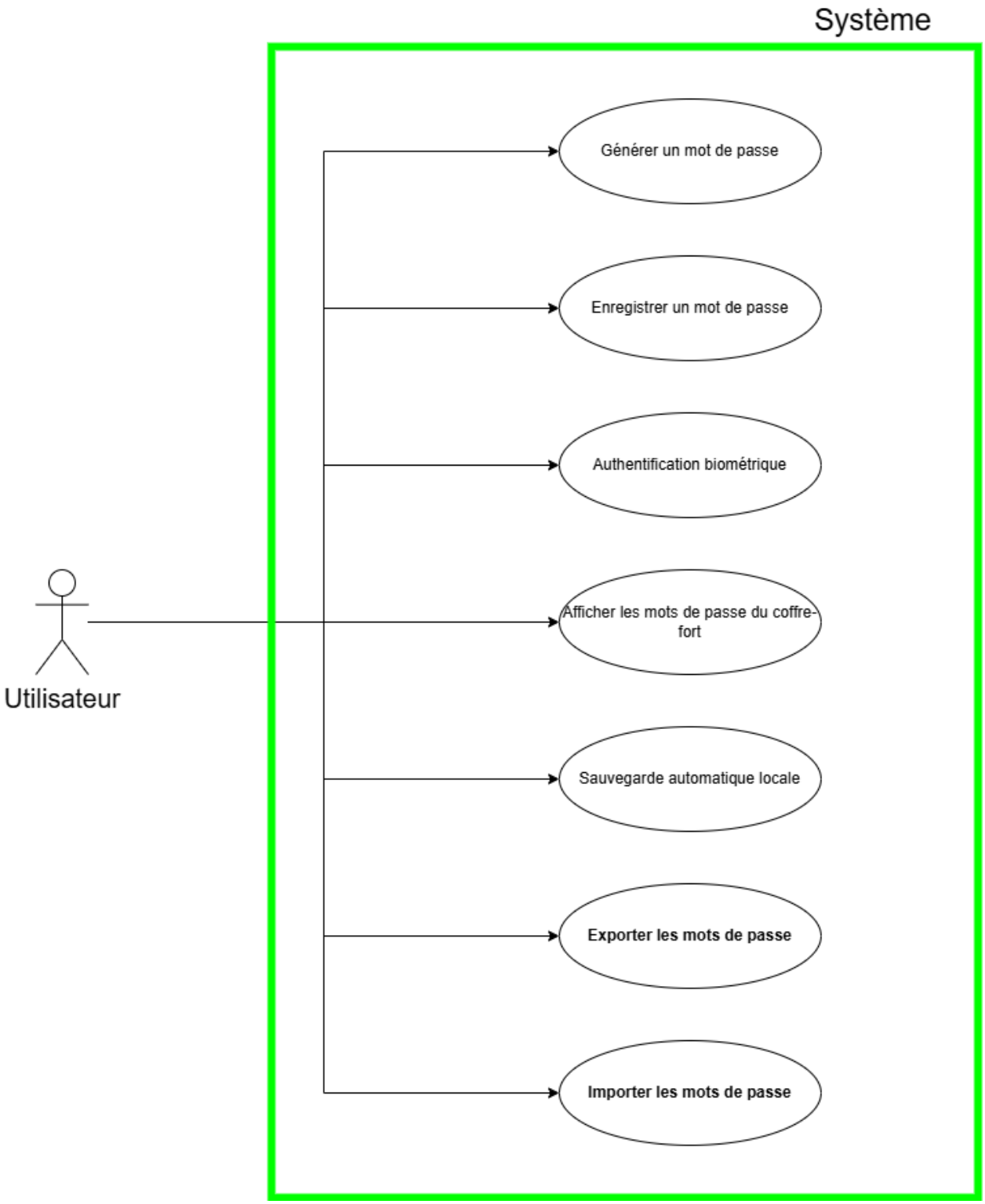
Conclusion

StormPass offre une solution complète pour gérer les mots de passe en toute sécurité. Avec une interface intuitive, des options de personnalisation et une sécurité de pointe, elle constitue un choix idéal pour ceux qui cherchent à protéger leurs données sensibles.

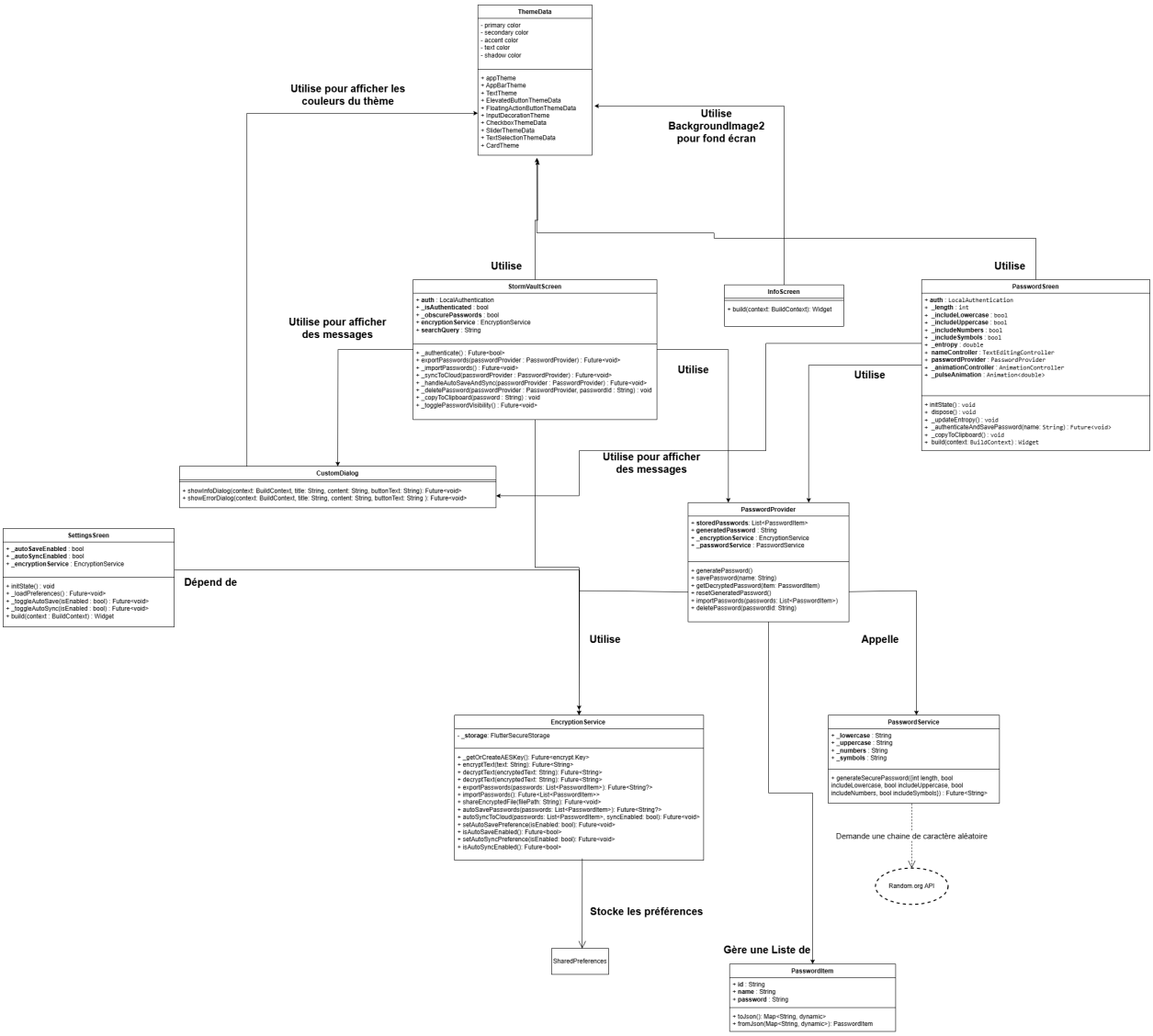
Organisation des fichiers du projet :

```
lib/
├── main.dart                    # Point d'entrée de l'application
├── screens/                     # Contient toutes les pages de l'application
│   ├── home_screen.dart         # Page d'accueil (menu principal)
│   ├── password_screen.dart     # Page pour générer des mots de passe
│   └── stormvault_screen.dart   # Page de stockage sécurisé des mots de passe
├── providers/                   # Gestion des états et logique de données
│   └── password_provider.dart   # Gère les mots de passe et le stockage sécurisé
├── models/                      # Définit les modèles de données
│   └── password_item.dart       # Modèle de données pour un mot de passe stocké
└── services/                    # Contient la logique de génération et de chiffrement
├── password_service.dart    # Génère des mots de passe sécurisés
└── encryption_service.dart # Chiffre, déchiffre, enregsitre et génère des fichiers avec les mots de passes chiffrés
```

Diagramme de cas d’utilisations :



Modélisation UML de l'application :



Descriptif Méthodes et Classes :

1. `PasswordProvider`

Gère la liste des mots de passe et les opérations de création, sauvegarde, et suppression.

- **Attributs :**
  - `storedPasswords` : Liste des mots de passe enregistrés (`List<PasswordItem>`).
  - `generatedPassword` : Mot de passe actuellement généré (`String`).

- `_encryptionService` : Instance pour le chiffrement (`EncryptionService`).
- `_passwordService` : Instance pour la génération de mots de passe (`PasswordService`).

- **Méthodes :**
  - `generatePassword()` : Génère un mot de passe avec les paramètres spécifiés.
  - `savePassword(String name)` : Sauvegarde le mot de passe généré avec un nom unique.
  - `getDecryptedPassword(PasswordItem item)` : Déchiffre un mot de passe.
  - `resetGeneratedPassword()` : Réinitialise le mot de passe généré.
  - `importPasswords(List<PasswordItem> passwords)` : Importe une liste de mots de passe.
  - `deletePassword(String passwordId)` : Supprime un mot de passe par ID.

---

2. `EncryptionService`

Gère le chiffrement, déchiffrement, et la sauvegarde automatique des mots de passe.

- **Attributs :**
  - `_storage` : Stockage sécurisé pour la clé de chiffrement (`FlutterSecureStorage`).

- **Méthodes :**
  - `_getOrCreateAESKey()` : Génère ou récupère la clé AES pour le chiffrement.
  - `encryptText(String text)` : Chiffre un texte.
  - `decryptText(String encryptedText)` : Déchiffre un texte chiffré.
  - `exportPasswords(List<PasswordItem> passwords)` : Exporte et chiffre les mots de passe dans un fichier local.
  - `importPasswords()` : Importe les mots de passe chiffrés depuis un fichier.
  - `shareEncryptedFile(String filePath)` : Partage le fichier chiffré.

- `autoSavePasswords(List<PasswordItem> passwords)` : Sauvegarde automatique des mots de passe localement.
- `autoSyncToCloud(List<PasswordItem> passwords, bool syncEnabled)` : Synchronise automatiquement les mots de passe avec le cloud si activé.
- `setAutoSavePreference(bool isEnabled)` : Configure la sauvegarde automatique.
- `isAutoSaveEnabled()` : Vérifie si la sauvegarde automatique est activée.
- `setAutoSyncPreference(bool isEnabled)` : Configure la synchronisation cloud.
- `isAutoSyncEnabled()` : Vérifie si la synchronisation cloud est activée.

---

3. `PasswordService`

Service pour générer des mots de passe sécurisés en utilisant l'API random.org.

- **Attributs :**
  - `_lowercase`, `_uppercase`, `_numbers`, `_symbols` : Ensemble de caractères pour la génération.
- **Méthodes :**
  - `generateSecurePassword(int length, bool includeLowercase, bool includeUppercase, bool includeNumbers, bool includeSymbols)` : Génère un mot de passe aléatoire selon les critères spécifiés en utilisant l'API random.org.

---

4. `SettingsScreen`

Écran pour gérer les préférences de sauvegarde et synchronisation automatiques.

- **Attributs :**
  - `_autoSaveEnabled` : Indique si la sauvegarde automatique est activée (`bool`).
  - `_autoSyncEnabled` : Indique si la synchronisation cloud est activée (`bool`).
  - `_encryptionService` : Service de chiffrement utilisé pour gérer les préférences.
- **Méthodes :**

- `_loadPreferences()` : Charge les préférences de sauvegarde et de synchronisation.
- `_toggleAutoSave(bool isEnabled)` : Active ou désactive la sauvegarde automatique.
- `_toggleAutoSync(bool isEnabled)` : Active ou désactive la synchronisation cloud.
- `build(BuildContext context)` : Construit l'interface utilisateur de l'écran des paramètres.

---

5. `StormVaultScreen`

Écran principal pour afficher les mots de passe stockés dans le coffre-fort.

- **Attributs :**
  - `auth` : Instance de `LocalAuthentication` pour la biométrie.
  - `_isAuthenticated` : Indique si l'utilisateur est authentifié (`bool`).
  - `_obscurePasswords` : Indique si les mots de passe sont masqués (`bool)`.
  - `encryptionService` : Instance pour le chiffrement.
  - `searchQuery` : Requête de recherche pour filtrer les mots de passe (`String`).
- **Méthodes :**
  - `_authenticate()` : Authentifie l'utilisateur avec la biométrie.
  - `exportPasswords(PasswordProvider passwordProvider)` : Exporte les mots de passe stockés.
  - `_importPasswords()` : Importe les mots de passe depuis un fichier.
  - `_syncToCloud(PasswordProvider passwordProvider)` : Synchronise les mots de passe vers le cloud.
  - `_handleAutoSaveAndSync(PasswordProvider passwordProvider)` : Sauvegarde et synchronise automatiquement après chaque modification.
  - `_deletePassword(PasswordProvider passwordProvider, String passwordId)` : Supprime un mot de passe après authentification.

- `_copyToClipboard(String password)` : Copie un mot de passe dans le presse-papier.
- `_togglePasswordVisibility()` : Basculer entre l'affichage et le masquage des mots de passe.

---

6. `PasswordItem`

Modèle représentant un mot de passe avec un identifiant unique, un nom, et le mot de passe chiffré.

- **Attributs :**
  - `id` : Identifiant unique du mot de passe (`String`).
  - `name` : Nom du mot de passe (`String`).
  - `password` : Mot de passe chiffré (`String`).
- **Méthodes :**
  - `toJson()` : Convertit l'objet en JSON.
  - `fromJson(Map<String, dynamic> json)` : Crée un `PasswordItem` à partir d'un JSON.

---

7. `CustomDialog`

Affiche des pop-ups personnalisés pour les informations et les erreurs.

- **Méthodes :**
  - `showInfoDialog(BuildContext context, String title, String content, String buttonText)` : Affiche un pop-up d'information.
  - `showErrorDialog(BuildContext context, String title, String content, String buttonText)` : Affiche un pop-up d'erreur.

---

8. `InfoScreen`

Écran affichant des informations générales sur l'application.

- **Méthodes :**

- `build(BuildContext context)` : Construit l'interface utilisateur de l'écran d'informations.

---

9. `ThemeData` (dans `theme.dart`)

Définit le thème visuel de l'application avec des couleurs et des styles personnalisés.

- **Attributs :**
  - `primaryColor`, `secondaryColor`, `accentColor`, `textColor`, `shadowColor` : Palette de couleurs.
  - `appTheme` : Thème principal de l'application, incluant les styles des boutons, texte, appBar, etc.
- **Classes internes :**
  - `BackgroundImage` : Widget pour appliquer une image de fond globale.
  - `BackgroundImage2` : Variation avec un effet d'échelle.

---

10. `PasswordScreen`

Écran pour générer, personnaliser et enregistrer un nouveau mot de passe.

- **Attributs :**
  - `auth` : Instance de `LocalAuthentication` pour l'authentification biométrique.
  - `nameController` : Contrôleur pour le champ de texte du nom du mot de passe.
  - `length`, `includeLowercase`, `includeUppercase`, `includeNumbers`, `includeSymbols`, `_entropy` : Options de personnalisation du mot de passe.
  - `_animationController` et `_pulseAnimation` : Contrôleurs pour les animations de bouton.
- **Méthodes :**
  - `_updateEntropy()` : Calcule l'entropie du mot de passe.
  - `_authenticateAndSavePassword(String name)` : Authentifie et enregistre le mot de passe avec un nom unique.

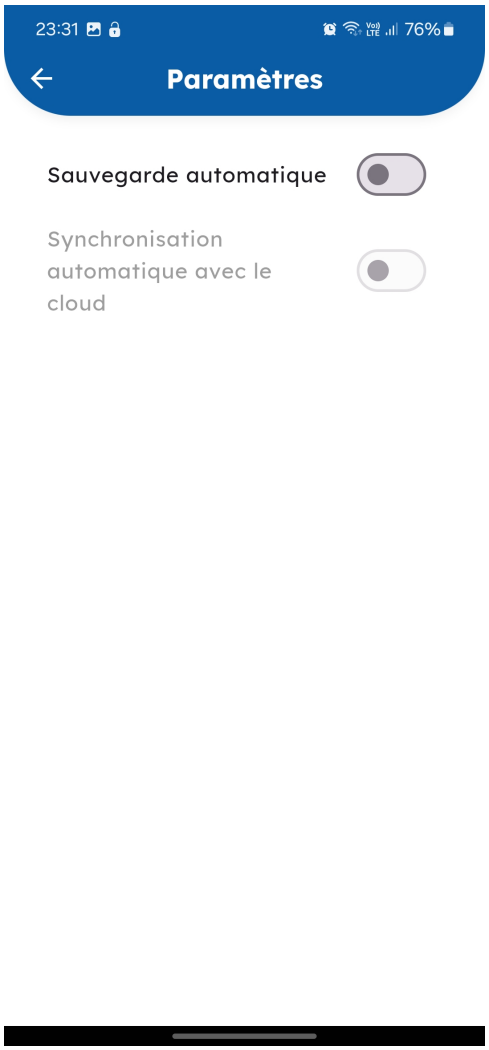
- `_copyToClipboard()` : Copie le mot de passe généré dans le presse-papier.
- `build(BuildContext context)` : Construit l'interface utilisateur de l'écran de génération de mot de passe.

Screenshot :

Accueil

Générateur Mots de passe

Coffre-Fort



Paramètres

Informations