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

HONORIS UNITED UNIVERSITIES

Rapport de projet flutter

Réalisé par :

AYA EL HAFIANE

CHAIMAE IHABI

Introduction :

Ce rapport présente le projet de développement d'une application mobile de gestion des tâches réalisée avec Flutter. L'objectif principal est de permettre à un utilisateur authentifié de créer, consulter, modifier et supprimer ses tâches, avec une persistance locale et une synchronisation vers une API REST.

Notre application intègre plusieurs notions importantes : authentification, gestion d'état avec BLoC/Cubit, persistance locale (Hive), communication HTTP, et support du mode sombre.

Objectifs du projet :

Les objectifs du projet sont les suivants :

- Authentifier les utilisateurs (inscription / connexion / déconnexion)
- Gérer une liste de tâches personnelle pour chaque utilisateur
- Ajouter, modifier, supprimer et marquer une tâche comme terminée
- Sauvegarder les tâches localement sur l'appareil
- Synchroniser les tâches avec une API REST (db.json)
- Offrir une interface moderne avec support du dark mode

Technologies utilisées :

- **Flutter :** c'est un framework open source développé par Google, permettant de créer des applications mobiles, web et desktop à partir d'une seule base de code. Il utilise le langage Dart et repose sur un système de widgets pour concevoir des interfaces rapides, personnalisables et performantes.

- **Dart :** c'est un langage de programmation optimisé pour les applications sur plusieurs plateformes. Il est développé par Google et est utilisé pour créer des applications mobiles, de bureau, de serveur et web.

- **FireBase Authentication :**
  Le SDK Firebase Authentication fournit des méthodes permettant de créer et d
  e
  gérer des utilisateurs se servant de leur adresse email et de leur mot de passe

  pour se connecter.
- **Hive :** Apache Hive est un système d'entrepôt des données distribué et résistant aux pannes qui permet d'effectuer des analyses à une échelle massive. Un entrepôt des données centralise le stockage des informations qui peuvent ainsi être facilement analysées pour prendre des décisions informées et orientées données. Hive permet aux utilisateurs de lire, d'écrire et de gérer des pétaoctets de données à l'aide de SQL.
- **BLoC / Cubit :** pour la gestion d'état.
- **HTTP / JSON Server :** communication avec l'API REST.

Architecture générale :

Notre projet est structuré selon une architecture, facilitant la maintenance et l'évolution de l'application . Sa Structure principale :

- models/ : définition des modèles de données (TaskModel)
- repositories/ : gestion de l'accès aux données (Hive + API)
- blocs/ : logique métier et gestion des états (AuthCubit, TaskCubit)
- pages/ : écrans de l'application (Login, Register, Home, Add/Edit Task)
- services/ : communication avec l'API REST
- ui/ : thèmes, couleurs et styles globaux
- widgets/ : composants réutilisables

✓ **Cette séparation garantit une bonne lisibilité du code, une réutilisabilité et une facilité de débogage.**

Fonctionnement de l’application :

a. Authentification

L’authentification est gérée par Firebase Authentication :

- Inscription avec email et mot de passe
- Connexion sécurisée
- Déconnexion
- Gestion des erreurs (mauvais mot de passe, problème réseau, etc.)

Le AuthCubit gère les différents états :

- AuthInitial
- AuthLoading
- AuthAuthenticated
- AuthUnauthenticated
- AuthFailure
- ✓ **Cela permet à l’interface de réagir automatiquement selon l’état de l’utilisateur.**

b. Gestion des tâches

Les tâches sont représentées par le modèle TaskModel contenant :

- identifiant unique
- titre
- description
- date de création
- état (faite / non faite)

La gestion des tâches repose sur TaskCubit, qui permet :

- Chargement des tâches
- Ajout immédiat d’une tâche
- Modification d’une tâche
- Suppression avec confirmation
- Mise à jour instantanée de l’interface

c. Persistance des données

- Localement (Hive) :
  Les tâches sont stockées par utilisateur dans une box Hive spécifique (tasks_userId).
- À distance (API REST) :
  Chaque tâche est également envoyée à une API REST (JSON Server) pour assurer une sauvegarde externe.

d. Synchronisation REST

Lors de l'ajout d'une tâche :

- La tâche est enregistrée localement
- Elle est ensuite envoyée à l'API REST (db.json) via HTTP POST

e. Interface utilisateur (UI/UX)

L'interface respecte les principes de Material Design :

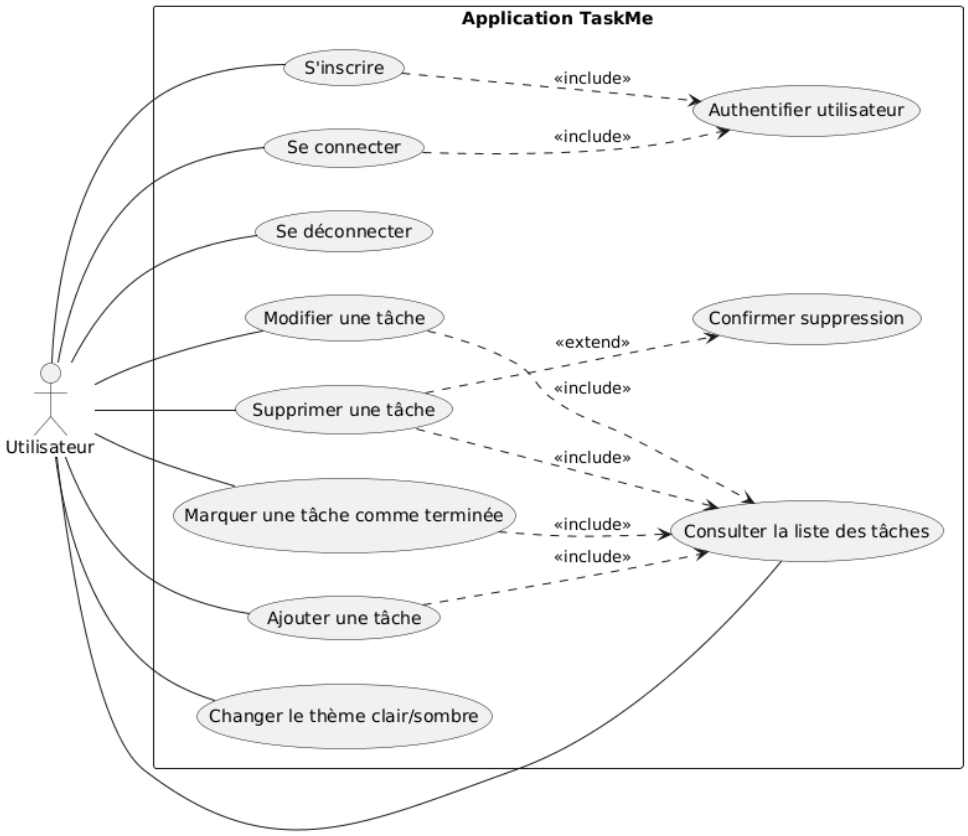
- Navigation fluide
- Boutons clairs et accessibles
- Feedback utilisateur (SnackBar, indicateurs de chargement)
- Design moderne et lisible

Fonctionnalités UI importantes :

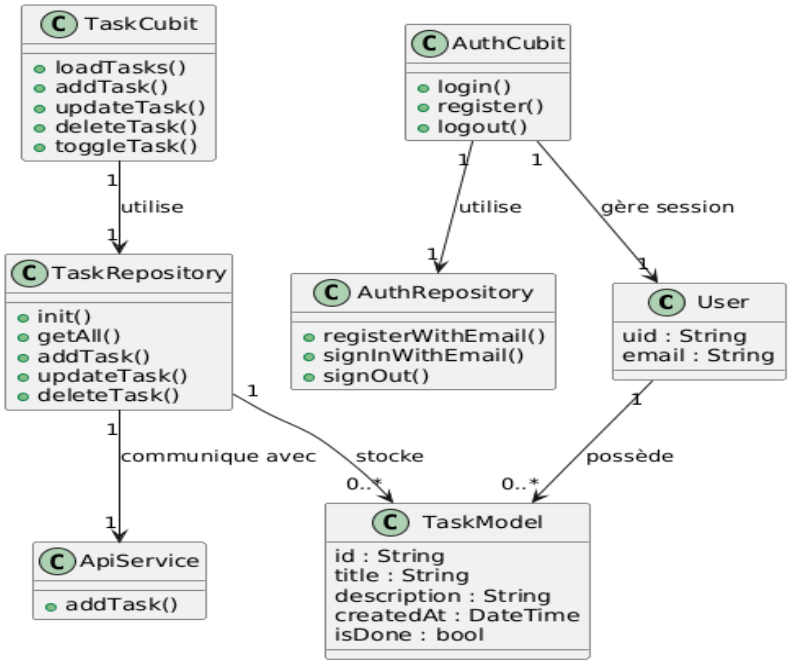
- Mode sombre / mode clair activable manuellement
- Swipe pour supprimer une tâche avec confirmation
- Progression visuelle des tâches complétées
- Interface responsive adaptée aux écrans mobiles

Conception :

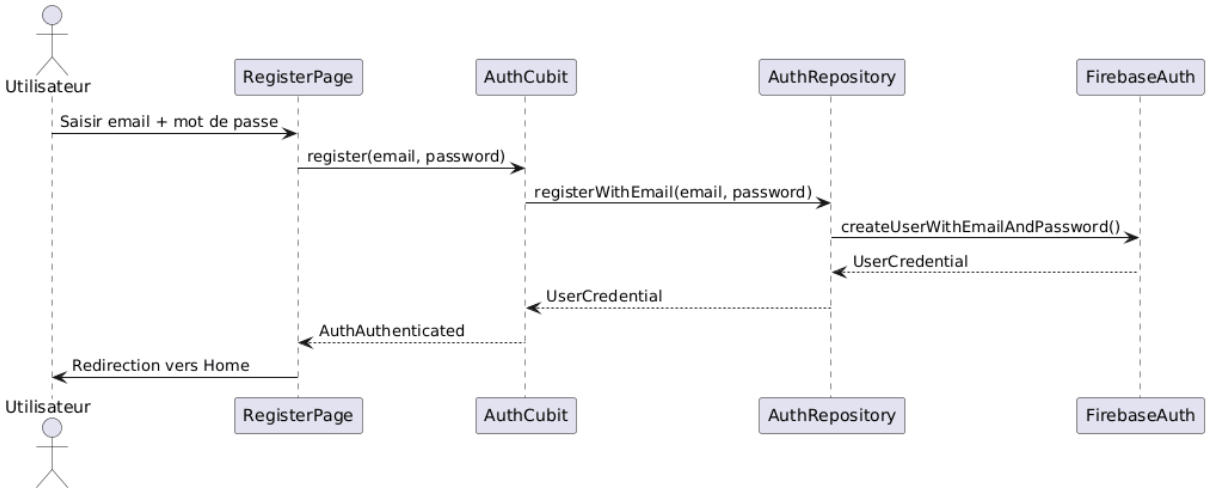
a. Diagramme de cas d’utilisation :



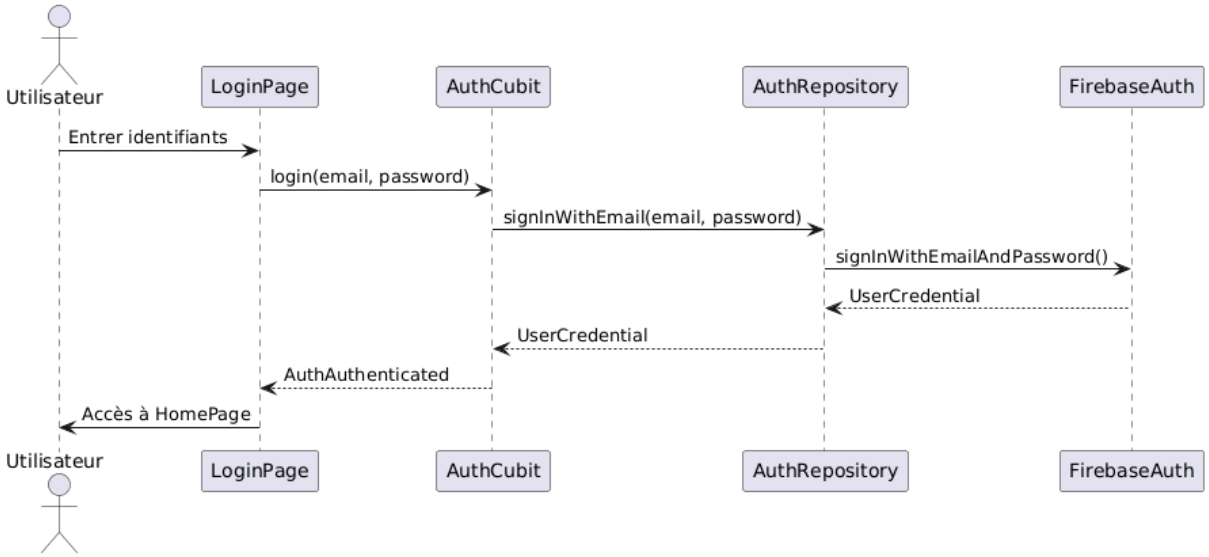
b. Diagramme de classes :



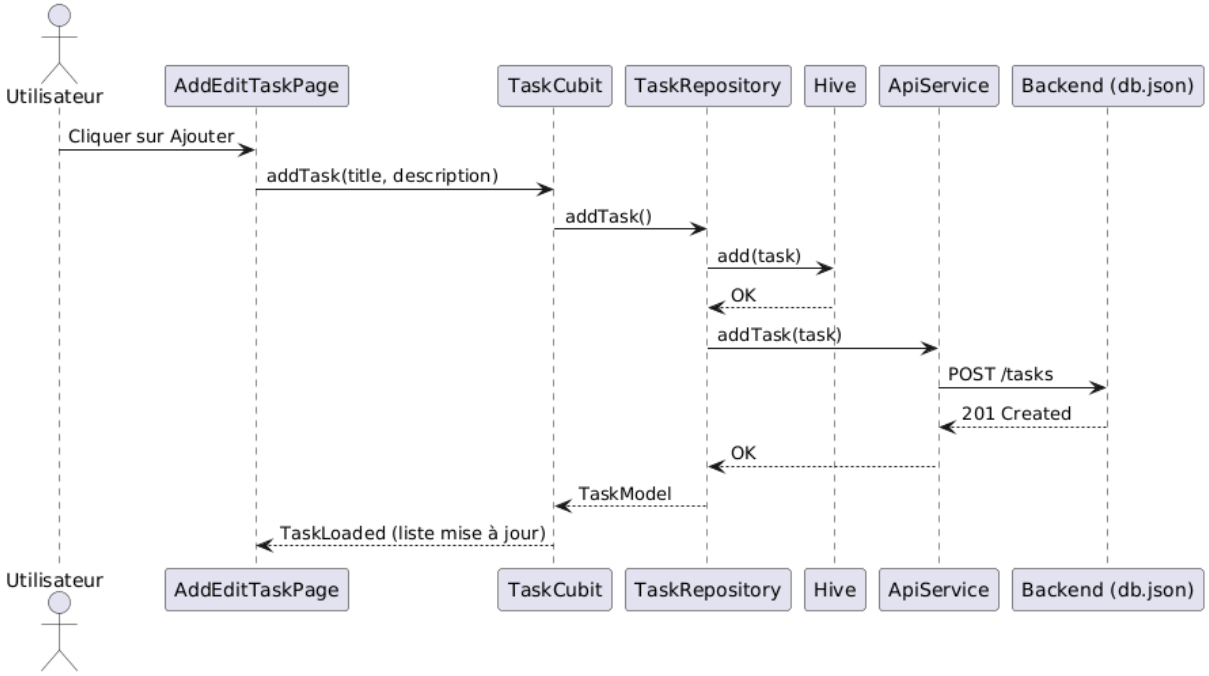
c. Diagramme de séquence – Inscription :



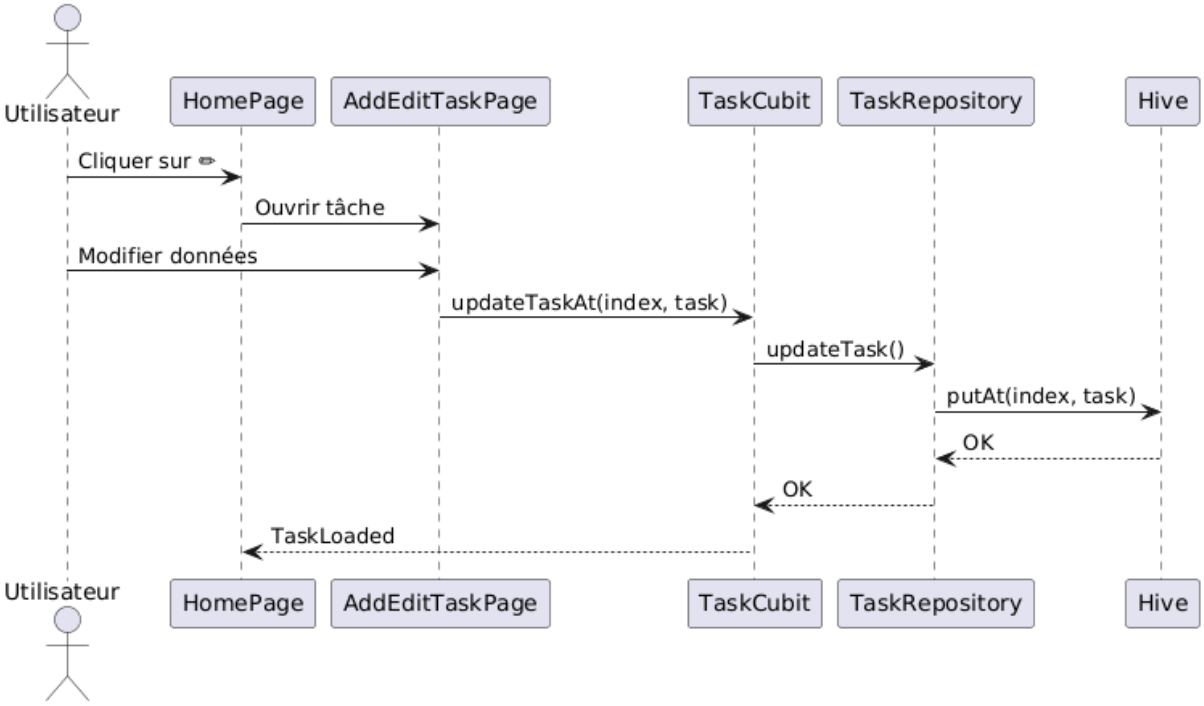
d. Diagramme de séquence – Connexion :



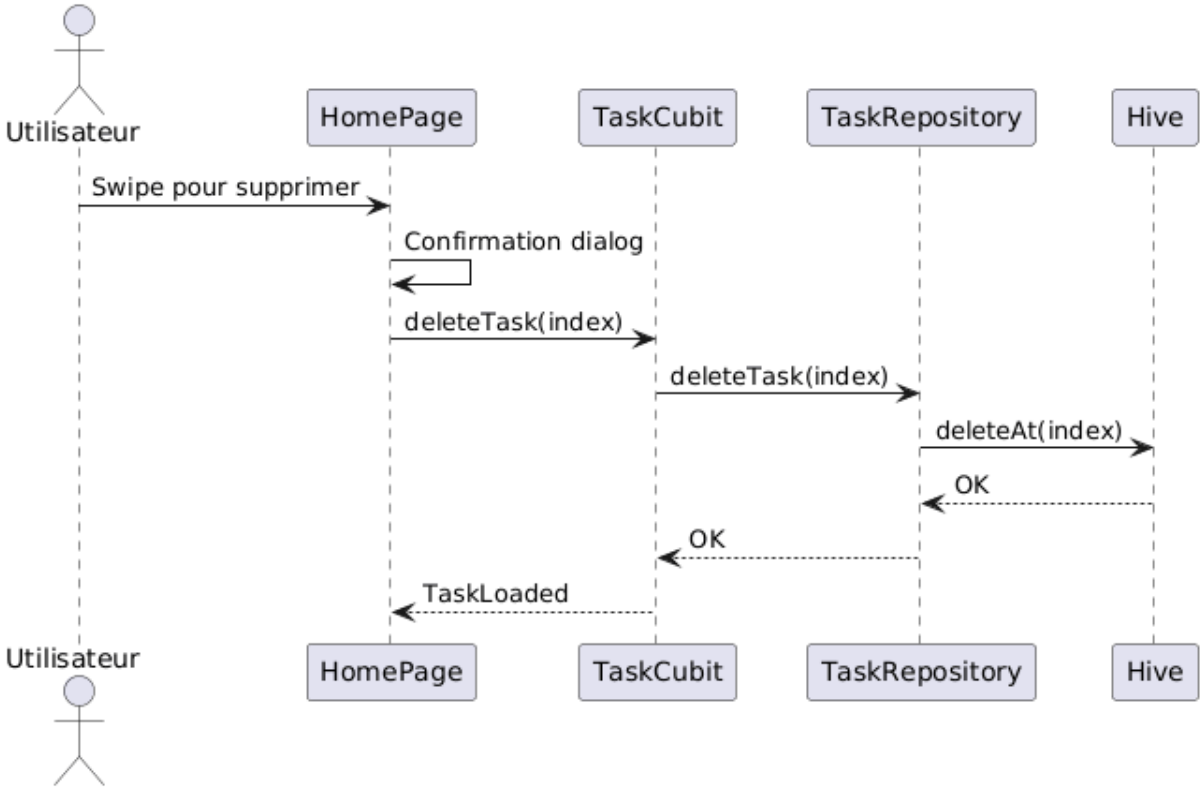
e. Diagramme de séquence – Ajouter une tâche :



f. Diagramme de séquence – Modifier une tâche :



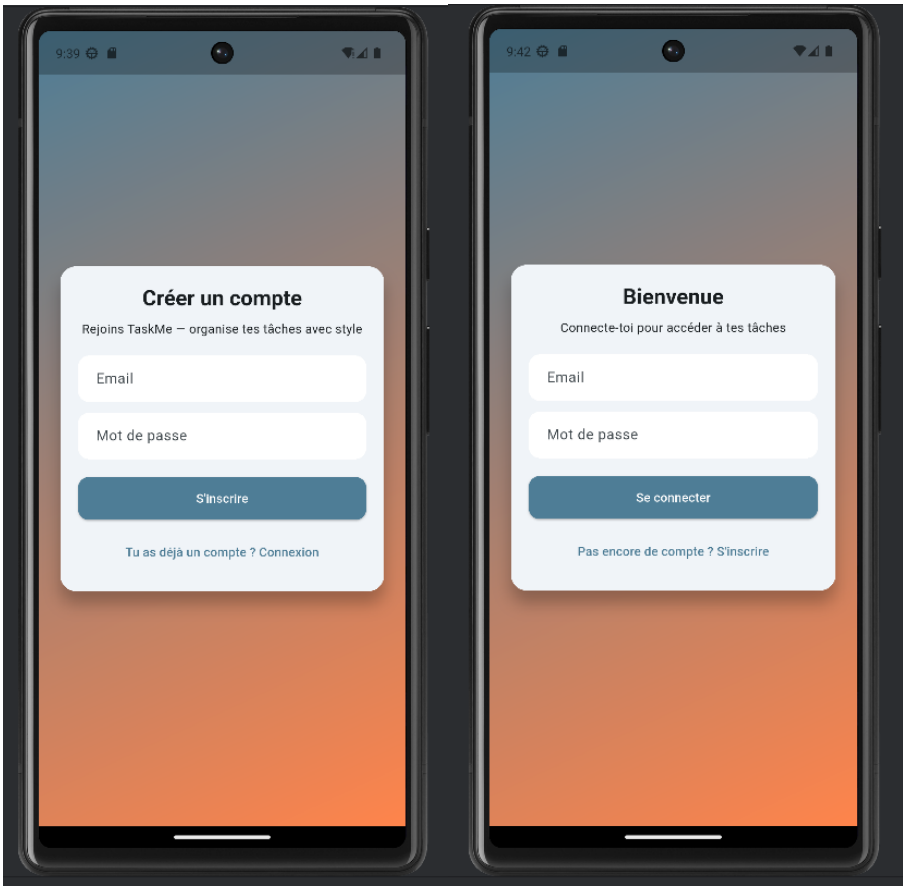
g. Diagramme de séquence – Supprimer une tâche :



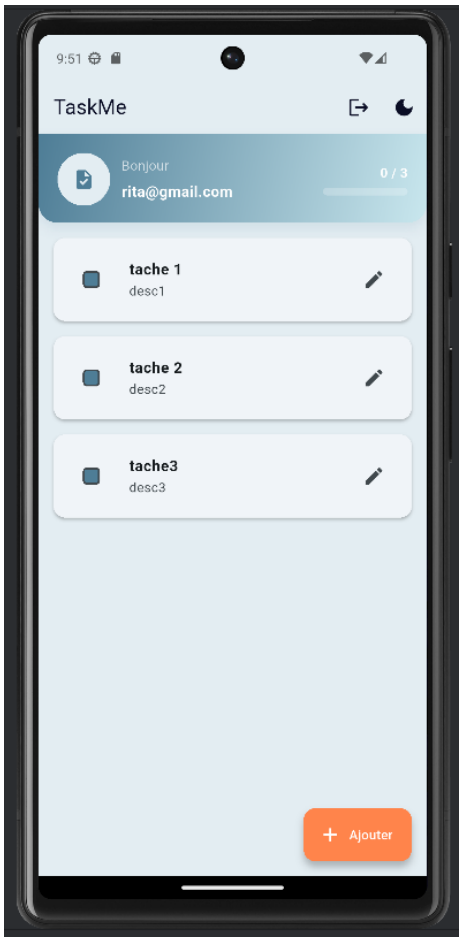
Résultats obtenus :

Page d’inscription :

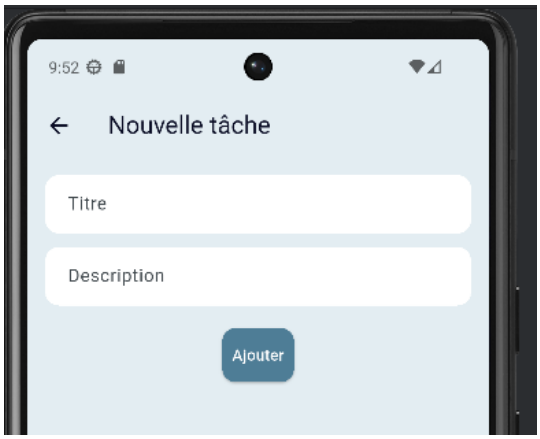
Page de connexion :



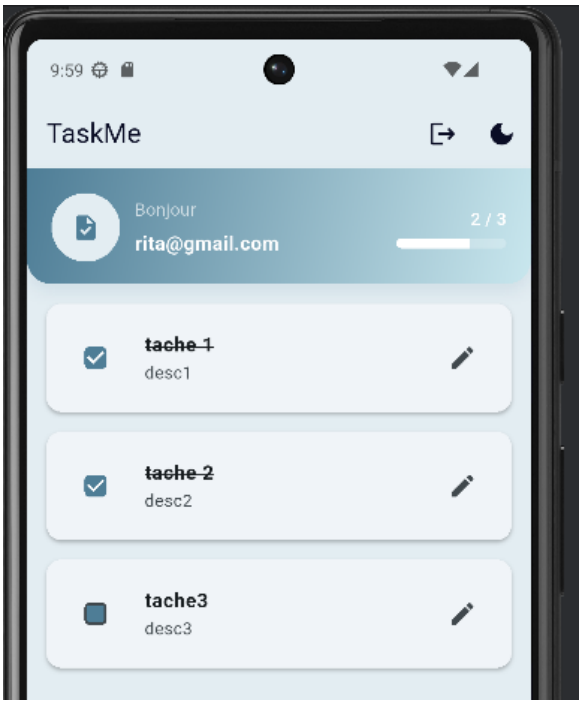
Home page :



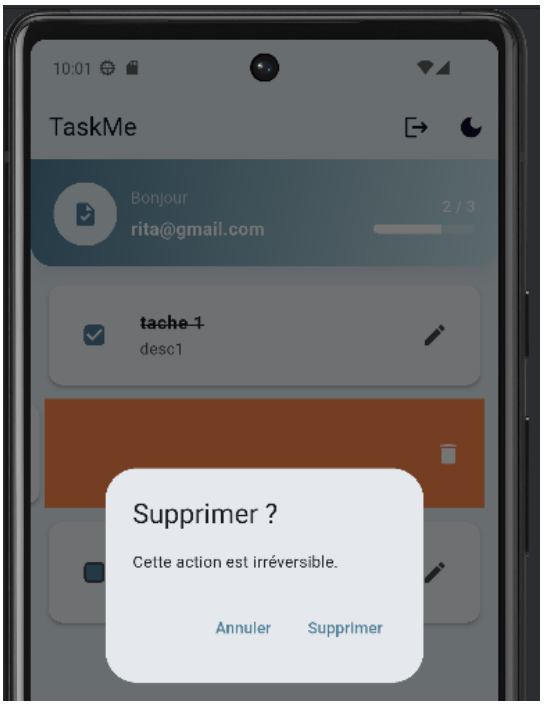
formulaires d'ajout et de modification :



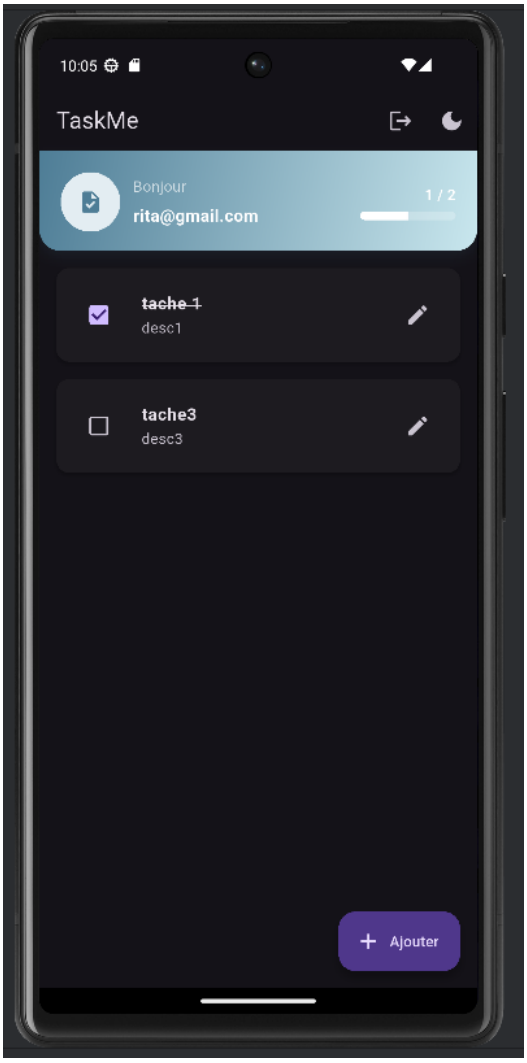
Taches barrées :



Suppression d'une tache :



Mode sombre :


10:05
TaskMe
Bonjour
rita@gmail.com
1 / 2
tache 1
desc1
tache3
desc3
Ajouter

Conclusion :

Ce projet a permis de mettre en pratique plusieurs concepts fondamentaux du développement mobile moderne à travers la conception et la réalisation de l'application TaskMe. La gestion d'état a été assurée de manière efficace grâce à l'architecture BLoC, garantissant une séparation claire entre la logique métier et l'interface utilisateur. La persistance locale des données, réalisée à l'aide de Hive, a permis d'assurer la disponibilité des tâches même en l'absence de connexion réseau, tandis que la communication avec une API REST a assuré la synchronisation des données vers le backend. L'ensemble du projet repose sur une architecture propre et modulaire, facilitant la maintenance, l'évolution et la compréhension du code. Cette application constitue ainsi une base solide pouvant être enrichie ultérieurement par des fonctionnalités avancées telles que la synchronisation bidirectionnelle en temps réel, la gestion des priorités, l'envoi de notifications ou encore l'amélioration des mécanismes de sécurité et de performance.

Bibliographie

Flutter documentation : https://docs.flutter.dev
Dart language : https://dart.dev
Firebase Authentication : https://firebase.google.com/docs/auth
Hive database : https://docs.hivedb.dev
BLoC / Cubit : https://bloclibrary.dev
JSON Server API : https://github.com/typicode/json-server
Material Design : https://m3.material.io