HAI811I – Programmation Mobile

RAPPORT TP3

SALHI Nina
22115492

Encadrant : Seriai Abdelhak Djamel

Master 1 Informatique - IASD

Université de Montpellier
Département d'Informatique

Table des figures

1 Introduction

Dans le cadre de ce TP, j'ai développé une application Android en Jetpack Compose permettant à un utilisateur de s'inscrire, se connecter, saisir son planning journalier et en consulter une synthèse.

J'ai fait le choix d'utiliser Jetpack Compose comme framework d'interface graphique moderne proposé par Google, pour sa simplicité, sa réactivité et sa bonne intégration avec les principes de Material Design 3. Pour la persistance des données, j'ai opté pour Room, une solution ORM officielle de Google, qui permet de gérer facilement les bases de données SQLite avec des entités, des DAO, et des migrations automatiques.

2 Création de base de données

L'objectif de cet exercice était de créer une interface permettant à un utilisateur de saisir ses informations personnelles (nom, prénom, email, etc.) via un formulaire, puis de les enregistrer localement pour les réafficher ensuite dans une page dédiée.

2.1 Structure de l'application

L'application repose sur l'architecture suivante :

- Modèle (Entity) : Représente la structure des données utilisateur à enregistrer.
- DAO (Data Access Object) : Contient les méthodes d'accès à la base de données (insertion, requête...).
- Base de données Room : Fournit un accès centralisé et sécurisé à la base de données locale.
- Interface Compose : Affiche le formulaire d'inscription et les données de manière dynamique.

L'utilisateur entre ses informations personnelles via un formulaire composé de champs dynamiques (Jetpack Compose). Avant l'enregistrement, les champs sont validés (longueur minimale du mot de passe, format de l'email, unicité du login...). Les informations saisies sont converties en un objet utilisateur, puis insérées dans la base Room via une méthode du DAO. Une fois l'utilisateur enregistré, l'interface redirige vers une page de confirmation qui affiche les données enregistrées. Les données affichées proviennent directement de la base de données. Cela garantit la persistance même en cas de fermeture de l'application.

2.2 Remarque

Lorsqu'un utilisateur interagit avec un formulaire (inscription, planning...), les données sont stockées dans des variables d'état (remember). À la validation, ces données sont transformées en objet de type User ou Planning, puis enregistrées dans la base grâce à insert via DAO. L'affichage (comme dans la page Synthèse) repose sur des requêtes Room, souvent dans un LaunchedEffect, pour récupérer dynamiquement les données liées à l'utilisateur ou à une date. Les navigations entre les écrans utilisent les routes déclarées avec

des paramètres (comme le login) pour assurer la continuité entre les actions de l'utilisateur et les données affichées.

Voici le rendu de l'application en image :

FIGURE 1 – Écran d'acceuil

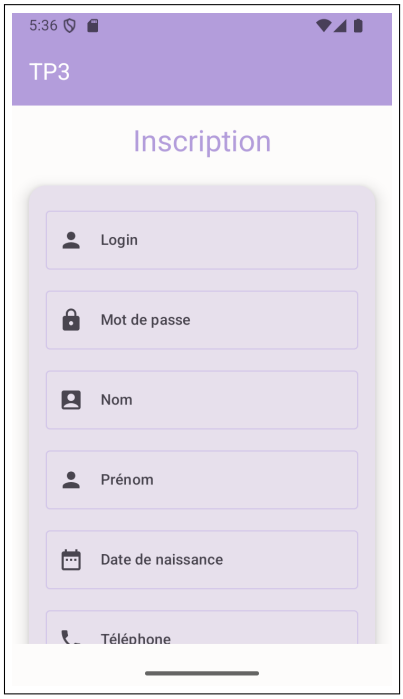

FIGURE 2 – Écran d'inscription

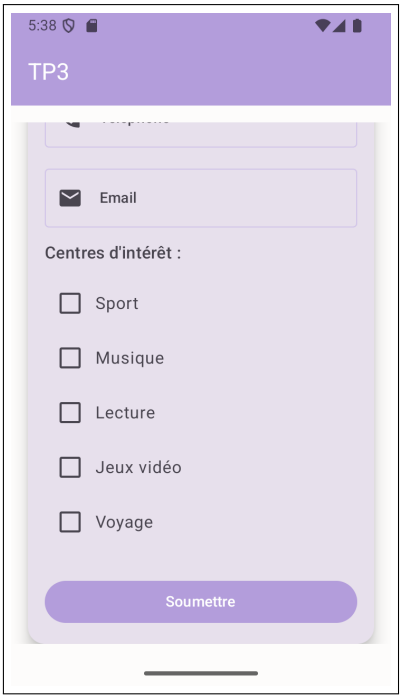

FIGURE 3 – Page d'inscription

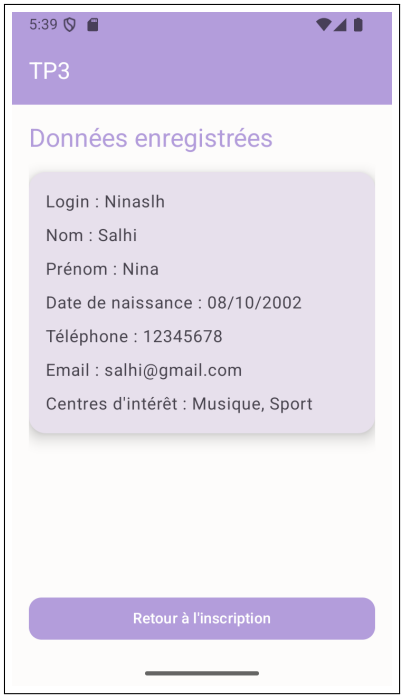

FIGURE 4 – Écran d'affichage

3 Utilisation de base de données

Dans cette seconde partie, l'objectif était de faire évoluer l'application initiale (créée à l'Exercice 1) en y intégrant une base de données locale à l'aide de Room, et en ajoutant plusieurs vérifications métier, ainsi que des écrans supplémentaires pour offrir une meilleure expérience utilisateur. Sachant que l'application suit une architecture modulaire avec des composables réutilisables, des navigations dynamiques, et une structure respectant les principes MVVM simplifiés.

L'écran de connexion propose à l'utilisateur de saisir son identifiant et son mot de passe. Ces informations sont vérifiées directement depuis la base avec Room :

Si les identifiants sont corrects, l'utilisateur est redirigé vers l'écran Planning, et son identifiant est passé dans la navigation. Sinon, un message d'erreur s'affiche à l'écran (via Toast).

Pour l'écran de planning journalier, il permet à l'utilisateur de remplir son emploi du temps pour la journée actuelle, en 4 créneaux fixes :

— 08h–10h

— 10h–12h

— 14h–16h

— 16h–18h

Lorsqu'un utilisateur enregistre son planning : La date du jour est automatiquement associée au planning. Si un planning existe déjà pour ce jour, il est mis à jour au lieu d'être recréé (grâce à la vérification dans le DAO). Le DAO PlanningDao contient notamment :

— insertPlanning(planning)

— getPlanningByLoginAndDate(login, date)

— updatePlanning(planning)

3.1 Page de Connexion

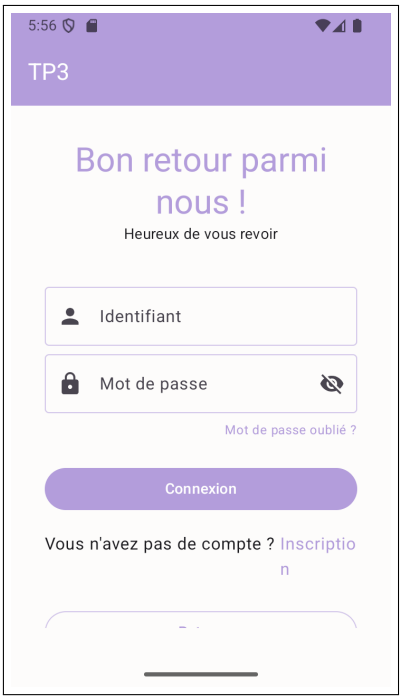

FIGURE 5 – Page de connexion

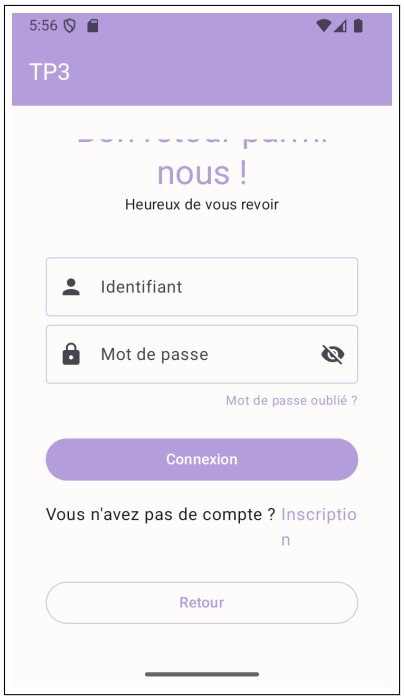

FIGURE 6 – Page de connexion

3.1.1 Remarque

La page de Connexion permet à l'utilisateur de s'authentifier en saisissant son identifiant (login) et son mot de passe. Une attention particulière a été portée à l'expérience utilisateur et à la sécurité :

- **Visibilité du mot de passe** : L'utilisateur peut choisir d'afficher ou de masquer son mot de passe à l'aide d'une icône d'œil (Visibility / VisibilityOff). Cela est implémenté en utilisant une variable d'état passwordVisible avec un composant IconButton pour basculer dynamiquement entre les deux icônes. Le champ de mot de passe utilise visualTransformation pour alterner entre le texte clair et masqué (PasswordVisualTransformation).
- **Vérification des identifiants** : Lors du clic sur le bouton *Connexion*, une requête Room est exécutée pour vérifier que le login existe en base de données et que le mot de passe correspond. Cela se fait dans une coroutine CoroutineScope(Dispatchers.IO) pour respecter les bonnes pratiques Android et éviter le blocage du thread principal.
- **Navigation conditionnelle** : Si la vérification est réussie, l'utilisateur est redirigé automatiquement vers la page de *Planning*, via navController.navigate("planning/login"), en passant son identifiant pour charger son planning personnalisé. En cas d'échec, un message d'erreur est affiché via Toast.
- **Lien vers l'inscription** : Un composant Row permet de proposer un lien cliquable "Inscription" si l'utilisateur n'a pas encore de compte. Cela redirige vers la page d'inscription via navController.navigate("inscription").

— **Retour à l'accueil** : Un bouton OutlinedButton permet de revenir à la page d'accueil (home) à tout moment, via une simple navigation.

3.2 Page de Planning

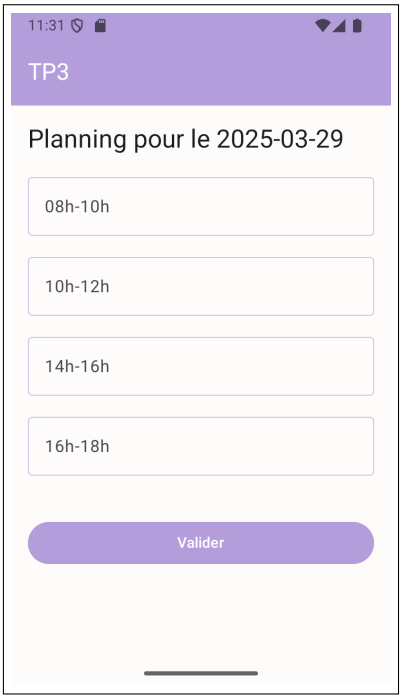

FIGURE 7 – Page de PLanning

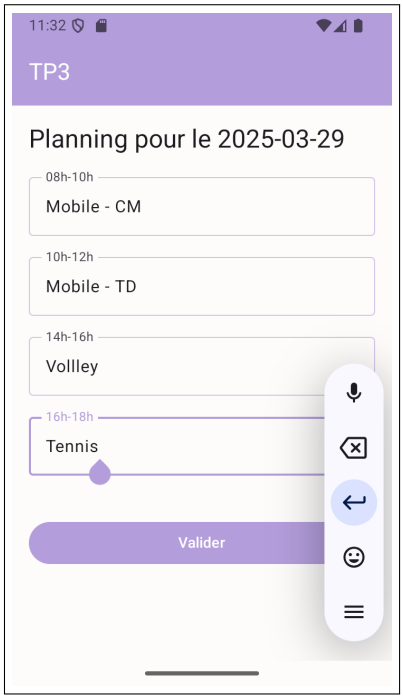

FIGURE 8 – Page de PLanning remplie

3.2.1 Remarque

La page de Planning permet à un utilisateur connecté de renseigner son emploi du temps journalier à l'aide de quatre créneaux horaires fixes :

— 08h – 10h
— 10h – 12h
— 14h – 16h
— 16h – 18h

Chaque champ est associé à un OutlinedTextField dans Jetpack Compose, et l'utilisateur peut saisir une activité pour chaque créneau (ex : Cours, TP, Réunion, Sport, etc.).

— **Transfert du login depuis la connexion** : Lorsqu'un utilisateur se connecte avec succès, son `login` est transmis automatiquement à la page de Planning via la navigation (par exemple : navController.navigate("planning/$login")). Ce paramètre est utilisé pour associer les données saisies au bon utilisateur, sans qu'il ait à ressaisir ses informations.

— **Gestion de la date** : À chaque accès à la page Planning, la date du jour est automatiquement détectée (SimpleDateFormat("yyyy-MM-dd")) et sert de clé pour sauvegarder ou mettre à jour les données en base. Cela permet de distinguer les plannings de différentes dates pour un même utilisateur.

— **Sauvegarde avec Room** : Lors de la validation, les données sont encapsulées dans un objet Planning contenant le login, la date, et les 4 créneaux. Cet objet est ensuite transmis à la base de données via Room (DAO).

Si un planning existe déjà pour cette date et ce login, il est **mis à jour**.

Sinon, un nouveau planning est **inséré**.

— **Persistance des données** : Grâce à l'utilisation de Room, les données du planning restent enregistrées localement sur l'appareil. Ainsi, l'utilisateur peut retrouver ses plannings à tout moment via la page *PlanningSynthese*.

— **Navigation vers la synthèse** : Après la validation, l'utilisateur est redirigé automatiquement vers la page `PlanningSynthese`, en lui passant également son login pour charger et afficher son planning du jour.

3.3 Page de Synthese de Planning

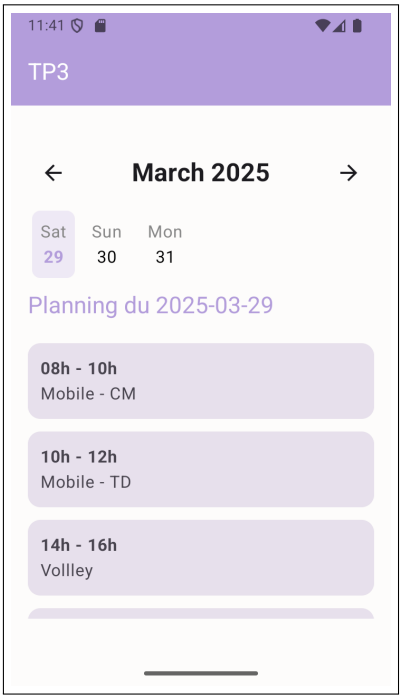

FIGURE 9 – Page de Synthese

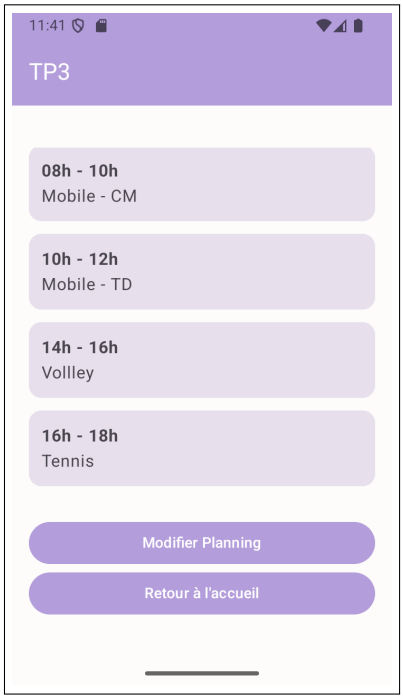

FIGURE 10 – Page de Synthese - suite

La page PlanningSynthese joue le rôle de tableau de bord des plannings enregistrés. Elle permet à l'utilisateur connecté de visualiser l'ensemble des plannings qu'il a saisis, jour par jour, de manière claire et interactive.

— **Chargement du planning du jour** : Lors de l'arrivée sur la page, la date du jour est automatiquement sélectionnée et le planning correspondant est chargé à partir de la base de données Room. Cela permet à l'utilisateur d'avoir une vue immédiate sur son emploi du temps actuel.

— **Affichage structuré des créneaux horaires** : Pour chaque jour sélectionné, 4 cartes sont affichées, correspondant aux créneaux suivants : `08h-10h`, `10h-12h`,

`14h-16h`, `16h-18h`. Si aucun planning n'a encore été défini pour cette date, un message d'information s'affiche.

— **Navigation dans les jours et les mois** : Une barre de calendrier horizontale en haut de l'écran permet de faire défiler les jours du mois en cours. Le jour en cours est automatiquement sélectionné au démarrage, mais l'utilisateur peut facilement naviguer dans le passé ou le futur. Le calendrier a été amélioré pour permettre également de faire défiler les **mois**, et ce de manière fluide. On commence l'affichage à partir de la date actuelle.

— **Consultation et modification du planning d'un jour spécifique** : En cliquant sur une date dans le calendrier, le planning associé est affiché en dessous. Si l'utilisateur clique sur le bouton `Modifier Planning`, il est redirigé vers la page `Planning`, avec le `login` et la `date sélectionnée` en paramètre. Ainsi, il peut :

Modifier un planning déjà existant pour cette date, ou en ajouter un nouveau si aucun n'avait été défini.

— **Persistance et mise à jour des données** : Lors de la validation dans la page `Planning`, la base Room est interrogée pour savoir si un planning pour cette date et cet utilisateur existe déjà :

Si oui, les nouvelles données sont **mises à jour**.

Sinon, un **nouvel enregistrement** est créé. Ces informations sont alors automatiquement disponibles dans `PlanningSynthese` après validation.

— **Navigation cohérente** : L'utilisateur peut revenir à la page d'accueil ou modifier un planning à tout moment grâce aux boutons `Retour à l'accueil` et `Modifier Planning` présents en bas de l'interface. Ces boutons ont été configurés avec des routes contenant les paramètres nécessaires (login et éventuellement date), assurant une transition fluide entre les écrans tout en maintenant l'identité de l'utilisateur pendant toute la session.

4 TP3 - Suite : Bonus : Manipulation des fichiers

Dans cet exercice, l'objectif était de concevoir une application Android utilisant Jetpack Compose et Material 3, permettant à un utilisateur de saisir des informations personnelles et professionnelles, et de déclencher en arrière-plan le téléchargement automatique des pages web liées aux liens fournis.

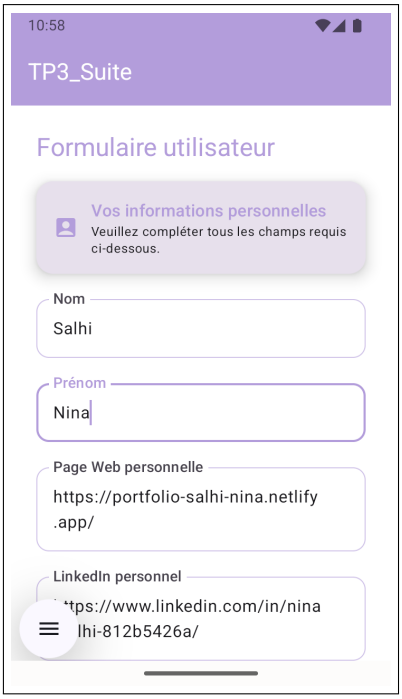

FIGURE 11 – Page de Formulaire

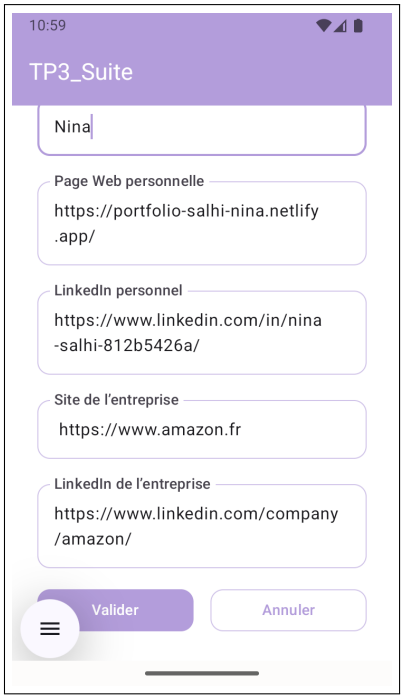

FIGURE 12 – Page de Formulaire - suite

4.1 Pourquoi un service ?

Dans Android, un Service est une composante destinée à exécuter des tâches en arrière-plan, sans interface graphique. Ici, on utilise un service pour éviter de bloquer le thread principal (UI thread) pendant que l'application télécharge les pages web à partir des liens fournis par l'utilisateur.

Le téléchargement HTML est potentiellement long ou instable (réseau, erreurs de lien), donc il doit être isolé du cycle de vie de l'interface utilisateur.

4.2 Enregistrement des données utilisateur en JSON

Lors de la validation du formulaire, les données saisies (nom, prénom, lien LinkedIn, lien entreprise, etc.) sont regroupées dans une classe UserLinks. Cette classe est sérialisée en JSON à l'aide de la bibliothèque kotlinx.serialization. Le fichier est ensuite enregistré localement dans le stockage interne de l'application (/data/data/.../files/user_data.json), garantissant sa confidentialité et sa persistance.

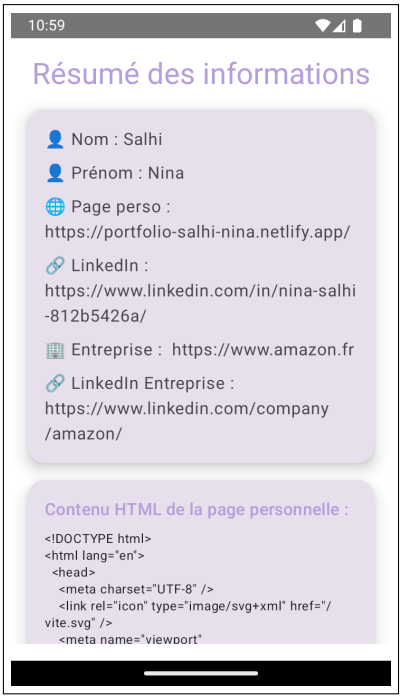

FIGURE 13 – Page de Resumé

FIGURE 14 – Page de Resumé - suite

4.3 Lancement d'un service pour le téléchargement des pages HTML

Afin d'éviter de bloquer l'interface principale, le téléchargement des pages HTML est effectué dans un service Android nommé HtmlDownloadService, basé sur IntentService.

Le service est déclenché après validation :Le JSON est passé au service via l'intention (Intent), ce qui permet d'accéder aux liens web dans le thread du service.

4.4 Fonctionnement du service HtmlDownloadService

Le service utilise les objets URL et readText() pour récupérer le contenu HTML brut des pages web, chaque lien est associé à un fichier (perso.html, linkedin.html, etc.), et le contenu est enregistré dans le répertoire interne de l'application. Les erreurs de téléchargement (ex. : lien LinkedIn bloqué ou inexistant) sont interceptées avec try-catch, et un fichier d'erreur est tout de même créé pour garder une trace. Une fois les liens récupérés, le service tente de télécharger le contenu HTML brut de chaque lien :

Les fichiers créés sont nommés selon le type de lien :

— perso.html

— linkedin.html

— entreprise.html

— entrepriseLinkedIn.html

Ils sont stockés dans le dossier privé de l'application, accessible uniquement par l'app, comme ceci :

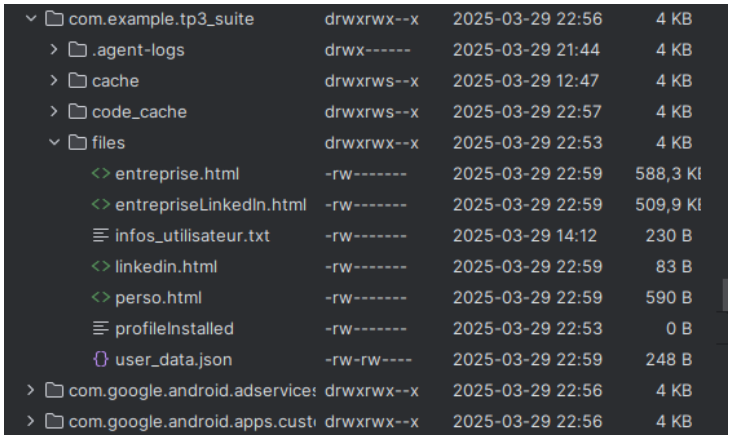

FIGURE 15 – Les fichiers créés

4.5 Remarque

Les fichiers générés par l'application, comme :

— user_data.json (données utilisateur sérialisées)

— perso.html, linkedin.html, entreprise.html, etc. (contenu HTML des pages web)

sont enregistrés dans le stockage interne privé de l'application Android. Ces fichiers ne sont pas accessibles directement via l'explorateur de fichiers du téléphone pour des raisons de sécurité.

Pour les visualiser en développement : On peut utiliser Android Studio avec l'outil intégré : Ouvrir Device File Explorer (menu View > Tool Windows > Device File Explorer) et naviguer vers :

/data/data/com.example.tp3_suite/files/

Vous y trouverez tous les fichiers HTML et JSON créés automatiquement par l'application.

5 Conclusion

Grâce à ce TP Android, j'ai réussi à élaborer une application complète pour la gestion des comptes utilisateurs et l'organisation quotidienne, en utilisant des technologies actuelles telles que Jetpack Compose pour l'interface utilisateur et Room pour la sauvegarde des données.

J'ai conçu une navigation fluide, des contrôles de saisie rigoureux et un système de gestion des données cohérent à travers plusieurs interfaces — inscription, connexion, saisie et récapitulatif du planning — qui respecte le contexte de l'utilisateur durant toute sa session. Jetpack Compose démontre sa capacité à concevoir des interfaces modernes et réactives à travers des fonctionnalités majeures telles que la vérification de l'unicité des identifiants, l'affichage du mot de passe, la modification quotidienne des plannings par date ou encore la synchronisation entre les différents écrans.

En complément, j'ai également réalisé le bonus du TP3, portant sur la manipulation des fichiers et l'utilisation d'un service Android. Cette extension m'a permis d'approfondir la gestion du cycle de vie des composants Android, en déclenchant un téléchargement

automatique de fichiers HTML en arrière-plan via un *IntentService*. Les données utilisateur sont enregistrées localement en JSON, puis utilisées par le service pour télécharger les pages web liées aux liens fournis. Les fichiers sont créés dans le stockage interne de l'application, garantissant leur sécurité et leur persistance.

Cette partie a illustré le rôle clé des services pour exécuter des tâches potentiellement longues sans bloquer l'interface, et a renforcé ma compréhension de la communication entre les composants (activité-service) dans une architecture Android propre.

Finalement, ce TP a fourni une expérience pratique dans la création d'une application mobile, de la structuration des données à la gestion des interactions utilisateur, au sein d'une architecture claire, robuste et maintenable.

6 Lien github

Le TP est disponible sur GitHub : Programmation-Mobile Repository