Todo List

Etape #08 : API

Variables d'environnement

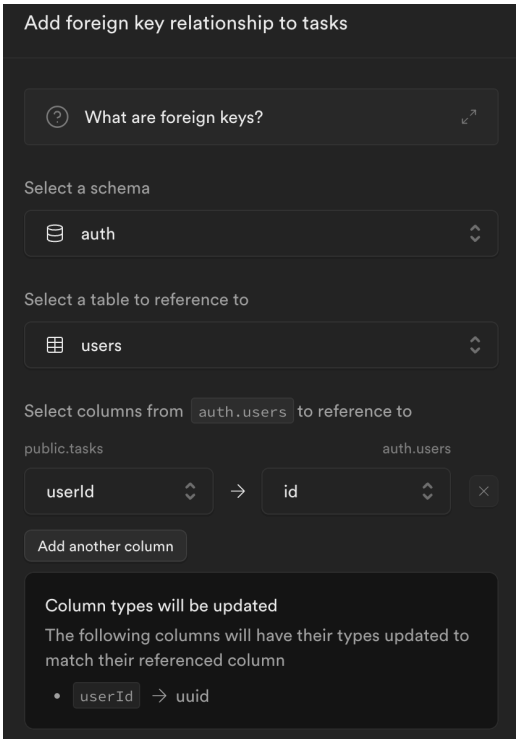
- Créez une branche ***etape08*** dans votre dépôt Git et passez sur cette branche.
- Ajoutez le package ***flutter_dotenv*** au projet (https://pub.dev/packages/flutter_dotenv).
- Créez un fichier ***.env*** à la racine du projet, ajoutez-le aux ***assets*** du fichier ***pubspec.yaml***
- Pour des raisons de confidentialité et de sécurité, faites en sorte que le fichier ***.env*** ne soit pas ajouté au dépôt Git de votre projet.

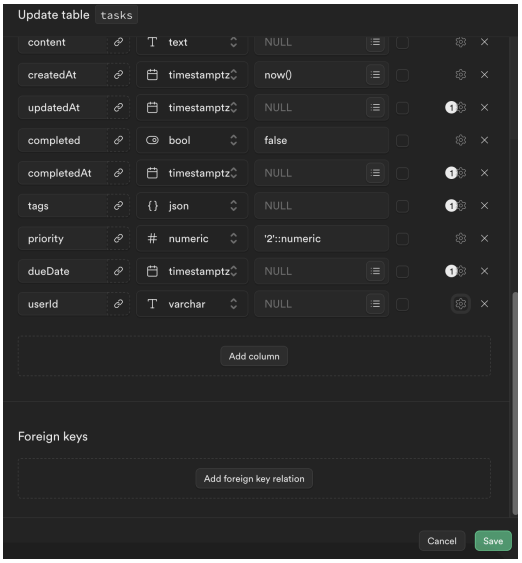
API REST

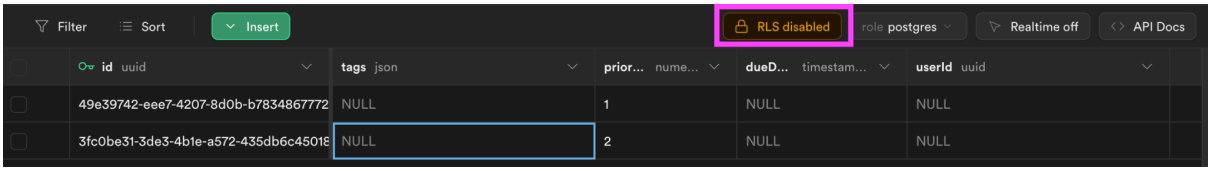
- Mettez en place une ***API REST*** distante avec via le service *Cloud* du *CMS Headless* ***Supabase*** (ou en alternative, le *CMS Headless* ***Directus***, en local via ***Docker*** https://docs.directus.io/self-hosted/docker-guide.html#example-docker-compose).
- L'*API* devra exposer les endpoints suivants :
    - ***tasks*** *(create, read, update, delete),*
    - ***tags*** (create, read, delete),
    - ***users*** *(sign up, sign in, read user's tasks, read user's tags).*
- Si ***Supabase*** est employé pour la réalisation de l'***API REST*** :
    - Consultez la documentation officielle mise à disposition par ***Supabase*** (https://supabase.com/docs/guides/api).
    - Créez un compte utilisateur sur ***Supabase*** (https://supabase.com/).

- Créez un projet nommé ***TodoList***.
- Créez la base de données relationnelle (table ***tasks, tags***). La collection ***User*** existe déjà par défaut dans ***Supabase***.
- Consultez la documentation concernant les **règles de sécurisation des tables de données dans *Supabase*** (https://supabase.com/docs/guides/api/quickstart).
- Consultez la documentation concernant la sécurisation de l'***API REST*** (https://supabase.com/docs/guides/api/securing-your-api).

  Dans le dashboard de ***Supabase***, adaptez les règles (***auth policy***) afin d'autoriser l'accès en lecture et en écriture à chaque collection de données via l'***API REST*** (ou passez tout en public -> *"RLS Disabled"*).
  Table Editor / <table name> / auth policy / schema public
- Gérez les relations ***Task*** / ***User*** et ***Tag*** / ***User*** sachant que la collection ***Users*** appartient au *schéma* ***auth*** de la base de données ***PostgreSQL*** employée par ***SupaBase***),







- Consultez la documentation officielle mise à disposition par ***Supabase*** (https://supabase.com/docs/guides/api).
- Ajoutez 3 variables d'environnement dans le fichier *.env* de votre projet Flutter :
  - ***SUPABASE_URL***
  - ***SUPABASE_ANON_KEY***
  - ***API_KEY***
- Renseignez les valeurs respectives communiquées dans le *dashboard* de ***Supabase*** (***Project Settings / API***).

Client HTTP

- Ajoutez le package ***dio*** (https://pub.dev/packages/dio) aux dépendances de votre projet,
- Adaptez la méthode ***fetchTasks*** de la classe ***TaskService*** afin de retourner les données communiquées par l'***API REST*** de ***Supabase*** à la place des données fictives générées avec ***Faker*** :
  - procédez à une requête *HTTP* auprès du endpoint ***tasks*** de l'***API REST***,
  - convertissez les données *JSON* réceptionnées en objets *Dart* à l'aide d'une méthode *factory* ***fromJson*** dans la classe ***Task*** (cf. inspirez-vous de l'exemple suivant : https://docs.flutter.dev/cookbook/networking/background-parsing),

    A noter : avec *Dio*, le parsing des données ***JSON*** en données *Dart* est automatisé (ce qui n'est pas avec le package ***http*** fourni nativement par *Flutter*).
- Pour chaque modèle de données :
  - mettez en place une méthode ***fromJson*** pour convertir des données de type ***Map<String, dynamic>*** provenant de *l'API*, vers un objet typé (selon le cas :

Task, Tag ou ***User***),

cf. exemple.

- mettez en place une méthode ***toJson*** pour convertir les données du format ***Dart*** au format ***JSON*** en vue de les transmettre à l'***API*** pour les enregistrer (création / mise à jour).
  cf. exemple.

```
Map<String, dynamic> toJson() => {
        'id': id,
        'name': name,
        'createdAt': createdAt.toIso8601String(),
};
```

- Faites en sorte que la classe ***TasksProvider*** collabore avec la classe ***TaskService*** afin de récupérer et synchroniser les données via l'***API REST***.
- Sélectionnez l'une de ces approches :
  - Chargement des données distantes à chaque changement d'écran (nécessite un accès au réseau).
  - Chargement centralisé des données distantes au lancement de l'application, stockage local (*RAM* ou base de données locale) puis transmission des données locales aux différents écrans.
    - **Mise à jour des données distantes et locales**, à chaque opération d'écriture (modification / suppression)

      OU

    - **mise à jour des données locales puis synchronisation des données distantes en 1 seule opération** lorsque le réseau est accessible.

- **Emploi des données locales en priorité**, y compris après mise à jour des données distantes (cf. *"optimistic update"*) afin d'éviter les rechargements et la dépendance au réseau.