Compte-rendu hebdomadaire - stage SIO

BESNIER Alexandre

3e semaine

Description de mission :

Mission principale :

La mission principale de la semaine 3 a consisté à **fiabiliser et structurer l'application web développée lors de la semaine 2**, afin de la rendre exploitable dans un contexte multi-utilisateurs et plus proche d'un environnement de production.

Cette phase a notamment porté sur la **migration de la base de données d'un stockage local vers une solution externe PostgreSQL hébergée sur Supabase**, permettant une meilleure persistance des données, un accès distant sécurisé et une préparation à un déploiement réel.
En parallèle, un **système d'authentification via Google OAuth** a été intégré, avec gestion des sessions utilisateurs et persistance de la connexion à l'aide de cookies afin de répondre aux contraintes de rechargement propres à Streamlit.

La semaine a également été consacrée à la **poursuite du développement des modules métier**, en particulier la gestion des employés (absences, congés, calendrier) et les fonctionnalités logistiques (stocks, livraisons, inventaires), ainsi qu'à l'amélioration de l'ergonomie et de la lisibilité de l'application.

Travail réalisé

- Migration de la base de données locale vers PostgreSQL (Supabase).
- Mise en place d'une connexion sécurisée à la base via secrets Streamlit.
- Intégration de l'authentification Google avec gestion des sessions et cookies.
- Structuration de l'application selon une architecture modulaire (models, repositories, services, use cases).
- Développement et correction des modules de gestion des employés et de la logistique de calendrier et de gestion des stocks et de gestion du compte.
- Amélioration de l'interface utilisateur, des parcours de navigation et refonte de l'interface utilisateur de page.

Outils

Python / Streamlit : développement de l'application web. **PostgreSQL (Supabase)** : base de données externe. **SQLAlchemy / Alembic** : ORM et gestion des migrations. **Google OAuth** : authentification des utilisateurs. **Git / GitHub** : gestion de versions, branches et suivi des user stories.

Structure du Projet (Evolution):

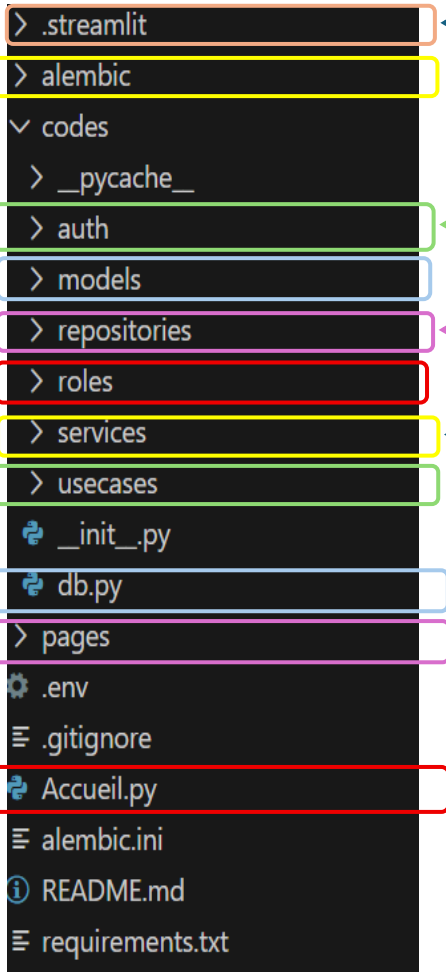
alembic/
Gestion des **migrations de la base de données**
→ Permet de versionner et faire évoluer le schéma PostgreSQL (Supabase) sans perte de données.

codes/models/
Modèles de données (ORM SQLAlchemy)
→ Définition des tables et relations (utilisateurs, employés, absences, stocks, livraisons, etc.).

codes/roles/
Gestion des rôles et permissions
→ Prépare la différenciation des droits utilisateurs (administrateur, employé, etc.).

codes/usecases/
Cas d'usage métier
→ Implémente les scénarios fonctionnels complets (enchaînement de règles et d'actions).

pages/
Pages Streamlit (interface utilisateur)
→ Chaque fichier correspond à une fonctionnalité métier (RH, logistique,



streamlit/
Dossier de configuration Streamlit
→ Gestion des paramètres de l'application (thème, secrets, configuration serveur).

codes/auth/
Gestion de l'authentification
→ Connexion Google OAuth, gestion des sessions, cookies et sécurité des utilisateurs.

codes/repositories/
Accès aux données (CRUD)
→ Centralise les requêtes vers la base de données et isole la logique SQL.

codes/services/
Services transversaux de l'application
→ Navigation, helpers UI, règles métier communes, normalisation des données.

db.py
Gestion de la base de données
→ Connexion PostgreSQL (Supabase), configuration SQLAlchemy et initialisation

Accueil.py
Page d'accueil de l'application
→ Initialisation globale, restauration de session utilisateur et navigation principale.

Le passage à une base de données distante s'est imposé avec l'intégration de l'authentification Google. En effet, la gestion de comptes utilisateurs, de sessions persistantes et de données partagées entre plusieurs utilisateurs nécessite une base accessible à distance et indépendante de l'instance locale de l'application. L'utilisation d'une base PostgreSQL hébergée sur Supabase permet ainsi de centraliser les informations d'authentification, d'assurer la persistance des sessions entre les connexions et de garantir un accès sécurisé et cohérent aux données, condition indispensable au fonctionnement d'une application web multi-utilisateurs.

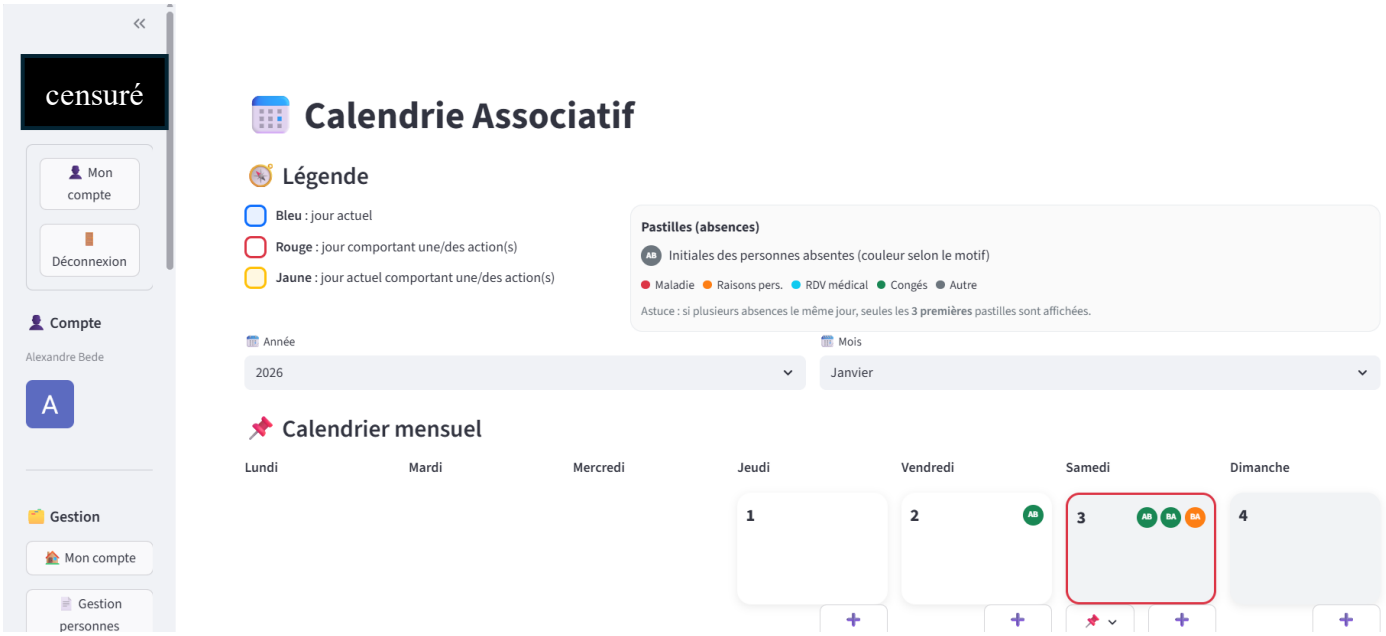
Un peu plus en détail :

La Page ‘Calendrier qui ressence les absences des employés/les actions de l’association (vue de l’administrateur :

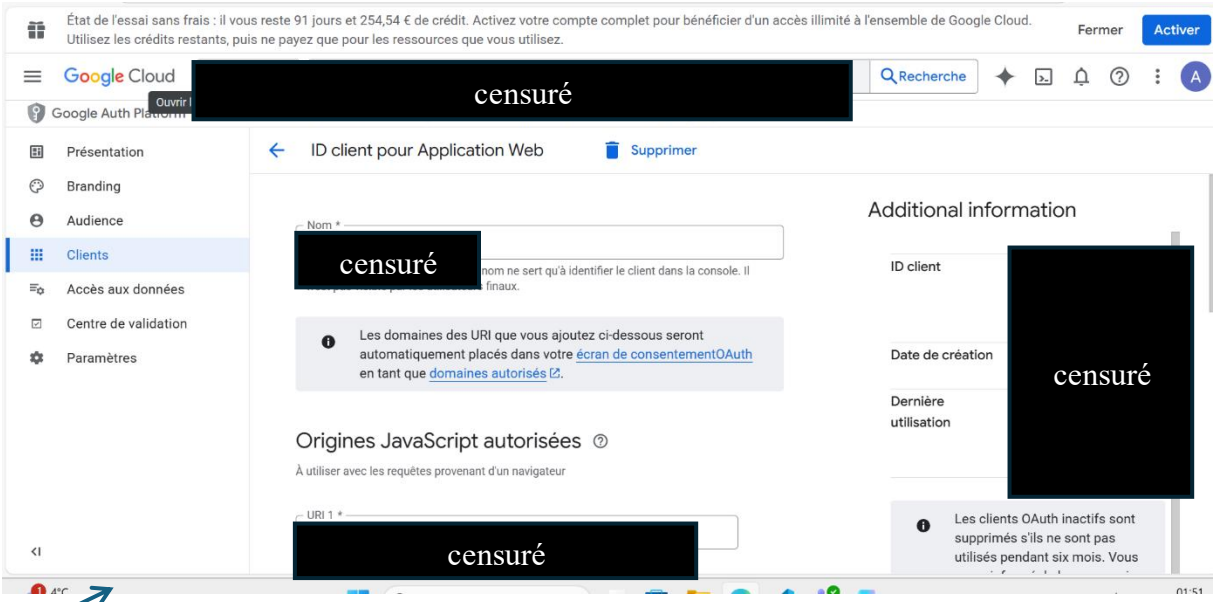
```python
target_date = date(year, month, day)
day_absences = absences_by_date.get(day, [])

has_action = day in action_days

classes = ["calendar-cell"]
if target_date == today and has_action:
    classes.append("today-action")
elif target_date == today:
    classes.append("today")
elif has_action:
    classes.append("has-action")

if weekday >= 5:
    classes.append("weekend")
```

target_date
→ Date complète correspondant à la cellule du calendrier (jour, mois, année).
day_absences
→ Liste des absences enregistrées pour ce jour, récupérées depuis la base de données.
has_action
→ Indique si une ou plusieurs actions associatives sont prévues ce jour-là.
classes
→ Liste dynamique de classes CSS appliquées à la cellule pour adapter l’affichage.
Jour actuel (today)
→ Bordure bleue pour repérer la date courante.
Jour avec action (has-action)
→ Bordure rouge pour signaler un jour important.
Jour actuel avec action (today-action)
→ Bordure jaune pour combiner les deux informations.
Week-end (weekend)
→ Fond grisé pour distinguer visuellement les samedis et dimanches.

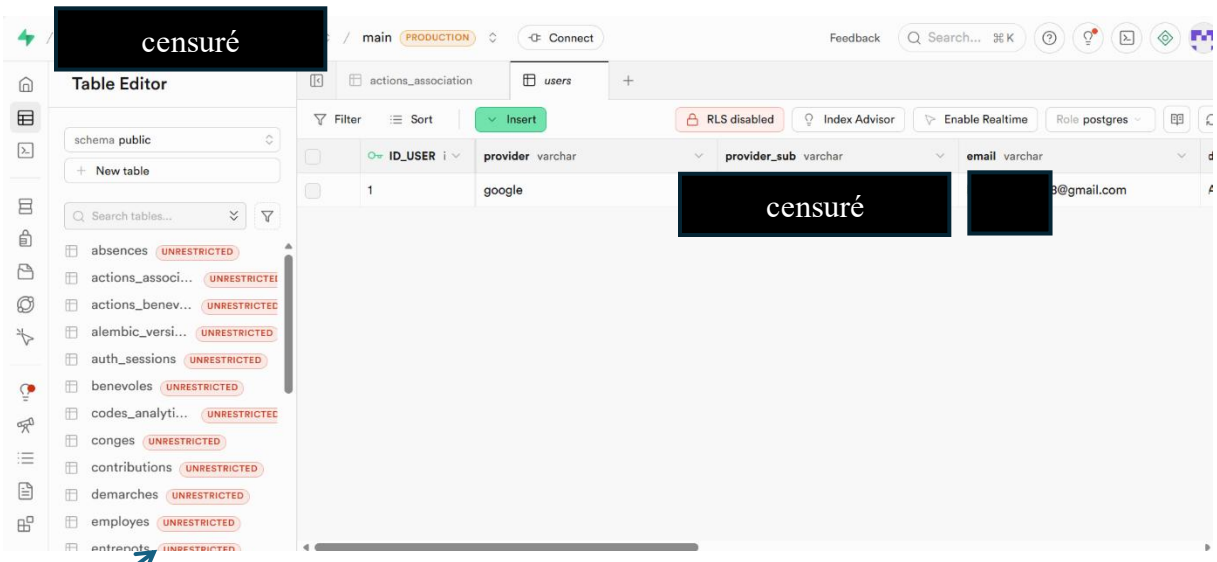
Visuel sur la page :



Fonctionnement de l'authentification via Google :

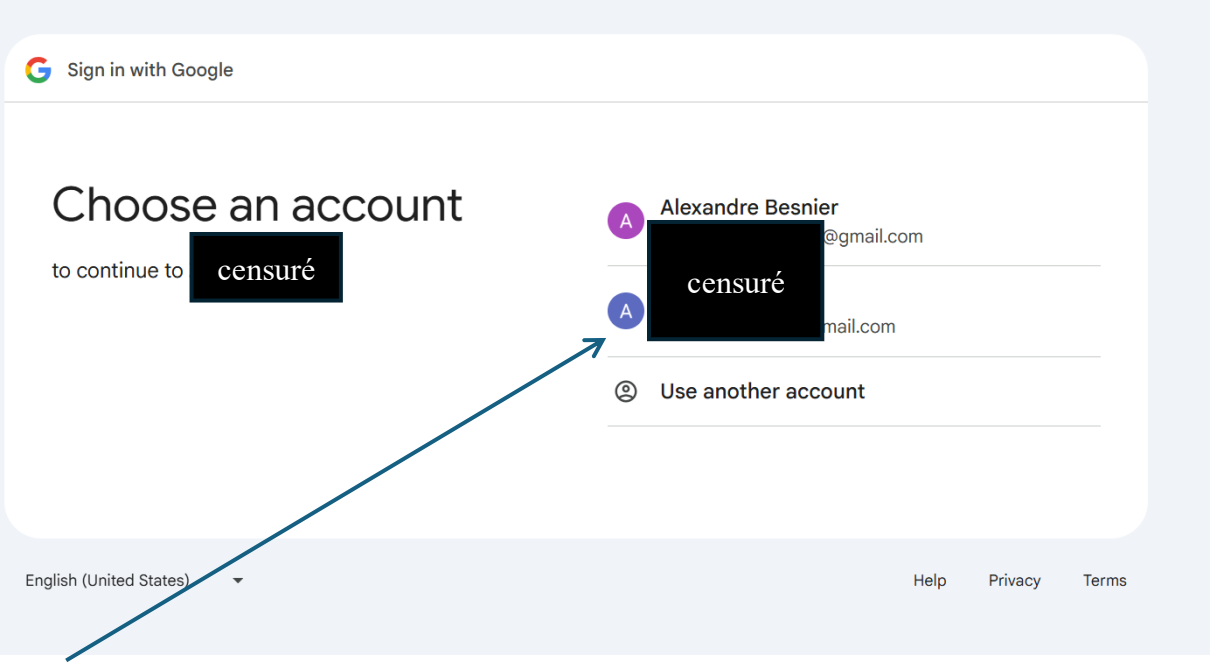


Cette capture présente la configuration Google Cloud utilisée pour l'authentification de l'application (OAuth 2.0). Elle permet d'identifier l'application auprès de Google et d'autoriser la connexion des utilisateurs via leur compte Google. Dans le code, la page de connexion Streamlit déclenche le flux OAuth (génération de l'URL d'authentification, gestion du paramètre *state* et redirection), puis récupère les informations d'identité (*claims*) afin de créer ou mettre à jour l'utilisateur côté application.

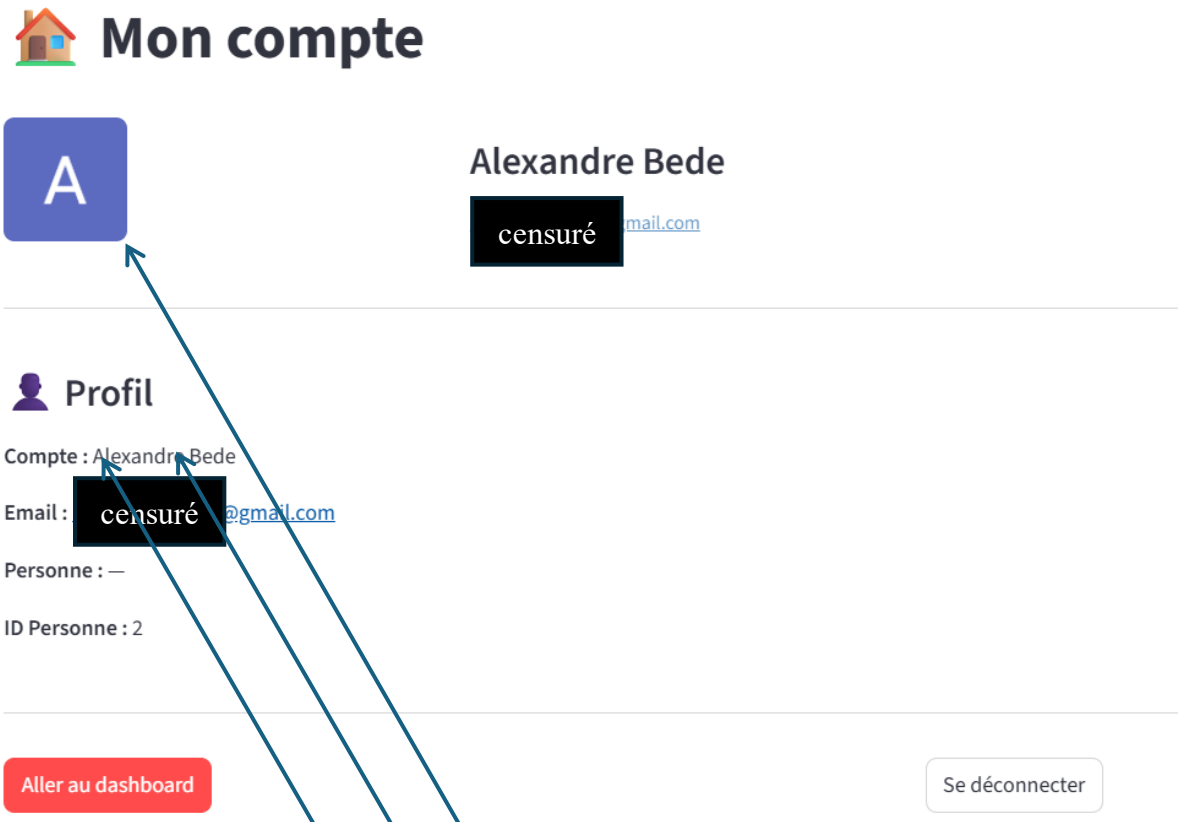


Cette capture montre la table des utilisateurs stockée dans la base PostgreSQL Supabase. Elle centralise les comptes applicatifs liés à l'authentification Google (email/identifiant, informations de profil) et sert de référence pour reconnaître un utilisateur à chaque connexion. Le code associe ces données aux *claims* récupérées via Google OAuth afin de créer ou mettre à jour automatiquement un utilisateur (mécanisme d'upsert), puis de relier ce compte à une entité métier interne (ex. Personne/Employé) pour piloter les

Login.py : Cette page déclenche le flux d'authentification Google OAuth/OpenID Connect. Les identifiants (Client ID, Client Secret, Redirect URI) sont chargés depuis st.secrets, ce qui évite d'exposer des informations sensibles dans le code. L'URL d'authentification est construite à partir du endpoint OIDC (authorization_endpoint) avec les paramètres requis (scope openid email profile). Un mécanisme de sécurité state signé (HMAC) avec durée de vie est généré pour protéger contre les attaques de type CSRF et garantir la validité du retour de Google.



Cette étape correspond à l'interface Google standard du flux OAuth. L'utilisateur sélectionne un compte Google, puis Google renvoie l'utilisateur vers l'application via la redirect_uri configurée. À ce moment-là, Google transmet un **code temporaire** (code) et le paramètre state initial, qui seront ensuite vérifiés côté application avant tout traitement. Cela garantit que la connexion est bien initiée par l'application et non par une redirection malveillante.



Cette page affiche le profil de l'utilisateur authentifié et sert de preuve de connexion fonctionnelle. L'accès est protégé par require_auth(), ce qui force la présence d'une session active avant d'afficher les données. Les informations affichées proviennent de st.session_state["auth_user"] (nom, email, photo), puis sont complétées par un lien avec les données métier internes : si l'utilisateur est rattaché à une entité Personne, le nom/prénom est récupéré depuis la base (Supabase PostgreSQL) pour assurer la cohérence entre l'identité Google et le référentiel interne. (= présent dans le code du projet)
L'utilisateur se voit ensuite attribué des rôles selon sa fonction (admin, gestionnaire, employé, bénévole etc..) qui lui donnent un certain nombre de limite/vues différentes sur les modules du site.

Compétences mobilisées :

Mission 1 :

1.6 : Organisation de son développement professionnel

Mission 2 :

1.2 : Réponse aux incidents et aux demandes d'assistance et d'évolution
1.4 : Travail en mode projet
1.5 : Mise à disposition des utilisateurs d'un service informatique