ANTOINE DINH

RAPPORT DE PROJET

Avril 2025

AP4 GSB

Bts SIO OPtion SLAM

SOMMAIRE

01

CONTEXTE GSB

Contexte GSB

Galaxy Swiss Bourdin (GSB) a émergé d'une fusion stratégique entre deux grands noms de l'industrie pharmaceutique : le géant américain Galaxy, spécialisé dans le traitement des maladies virales comme le SIDA et les hépatites, et le conglomérat européen Swiss Bourdin, connu pour ses médicaments traditionnels. Cette collaboration en 2009 a donné naissance à un leader incontesté du secteur pharmaceutique.

GSB a établi son siège administratif à Paris pour son entité européenne, tandis que le siège social de la multinationale se situe à Bagneux, dans les Hauts-de-Seine. Ces choix stratégiques reflètent l'engagement de GSB envers l'excellence et l'innovation au cœur de l'Europe.

Avec une équipe de vente composée de 480 visiteurs médicaux en France métropolitaine et 60 dans les départements et territoires d'outre-mer, GSB maintient une présence significative et influente dans le domaine médical, assurant une couverture complète et efficace sur tout le territoire national.

GSB est reconnu comme un expert mondial en études cliniques, contribuant de manière significative à la progression de la recherche médicale. Au niveau national, GSB excelle dans la vente de médicaments en B2B avec les professionnels de santé et en C2B avec les clients particuliers, consolidant ainsi sa position d'expert en commercialisation de solutions thérapeutiques.

Le département IT de GSB a besoin d'une application mobile pour permettre à ses employés de signaler et suivre les problèmes techniques rencontrés.

02

BESOINS ET OBJECTIFS

Expression des besoins et objectifs

Ajout de ticket: L'utilisateur, en tant qu'employé, doit pouvoir ajouter de nouveaux tickets de manière simple et claire. Le formulaire de création doit permettre de saisir les informations essentielles liées à la demande (titre, description, priorité, localisation, etc.). Cela facilite la remontée rapide des incidents ou besoins, tout en garantissant une bonne qualité des informations enregistrées pour un traitement efficace.

Gestion des tickets : Les utilisateurs employés doivent pouvoir consulter la liste complète des tickets qu'ils ont créés. Chaque ticket représente une demande ou un incident à traiter. Les employés peuvent également supprimer leurs propres tickets si nécessaire, ou ajouter des commentaires afin de fournir des précisions ou de suivre l'évolution de la situation. Ces fonctionnalités permettent d'assurer un suivi efficace des demandes et de centraliser les échanges autour de chaque ticket.

Interface de liste de tickets: L'application doit proposer une interface permettant d'afficher une liste claire et organisée des tickets. Les utilisateurs doivent pouvoir filtrer les tickets par statut (ouvert, en cours, résolu, etc.) et par priorité (basse, moyenne, haute) afin de faciliter la navigation et la gestion des demandes.
Un système de pagination avec chargement progressif ainsi qu'une fonction de recherche permettent d'accéder rapidement à un ticket spécifique.
Des indicateurs visuels (couleurs, icônes) doivent être utilisés pour distinguer facilement les niveaux de priorité et les statuts, améliorant ainsi la lisibilité et l'efficacité de l'interface.

Assignation et suivi: Les administrateurs doivent pouvoir assigner des tickets à des employés spécifiques afin d'organiser la répartition des tâches et assurer un meilleur suivi.
Le système doit également notifier les utilisateurs concernés en cas de nouvelle assignation ou de mise à jour sur un ticket (commentaire, changement de statut, etc.), pour garantir une communication fluide et en temps réel.
Enfin, un tableau de bord doit présenter des statistiques de base, afin de fournir une vue d'ensemble de l'activité et d'aider à la prise de décision.

Ces objectifs sont essentiels pour développer une application robuste et sécurisée qui répond aux exigences des professionnels de santé. Ils s'inscrivent également dans les meilleures pratiques en matière de gestion des stocks et de sécurité des patients.

Fonctionnalités natives : L'application doit tirer parti des fonctionnalités natives des appareils mobiles pour enrichir l'expérience utilisateur et améliorer l'efficacité du système.

L'intégration de la géolocalisation permet aux employés de marquer automatiquement l'emplacement des problèmes lors de la création d'un ticket, facilitant ainsi l'intervention sur le terrain et la traçabilité des incidents.

De plus, la mise en place de notifications push garantit que les utilisateurs sont informés en temps réel des mises à jour importantes liées aux tickets (nouveaux commentaires, changements de statut, assignations), favorisant une réactivité optimale.

Ces fonctionnalités sont fondamentales pour concevoir une application fiable et efficace, adaptée aux besoins des professionnels de santé. Elles contribuent à une meilleure gestion des tickets et des ressources, tout en respectant les bonnes pratiques en matière d'organisation, de traçabilité et de sécurité des utilisateurs.

03

ANALYSE FONCTIONNELLE ET CHOIX TECHNIQUES

Analyse fonctionnelle et choix techniques

Listage des fonctionnalités

L'application créée pour GSB présente diverses fonctionnalités clés, divisées pour objectif finale la création de l'ordonnance

Authentification et gestion des utilisateurs :

- Écran de connexion pour les utilisateurs: Permet aux utilisateurs de se connecter de manière sécurisée ou de s'inscrire s'il n'a pas de compte.
- Écran de gestion des rôles : Offre la possibilité de visualiser et de monter le niveau d'autorisation des utilisateurs "employées".

Gestion des tickets:

- Liste des tickets: Affiche les tickets qui peuvent être affichés selon le rôle de l'utilisateur avec le titre, le statut, la priorité du ticket.
- Création d'un nouveau ticket: Permet de générer un nouveau ticket.
- Suppression d'une commande existante : Permet la suppression d'une commande qui a le statut : "en attente".

Gestion des tickets avec détails:

- Affichage des détails du ticket avec des informations supplémentaires (le nom du créateur, la date de création, la date de la dernier modification, la personne qui est assigné à ce ticket, et position), un bouton permet de plus de voir les commentaires associés.
- Suppression du ticket : Permet la suppression d'un ticket
- Modification du ticket: Permet de changer des informations du ticket

Gestion des tickets avec détails (support):

- Permet en plus de modifier le statut

Gestion des tickets avec détails (admin):

- Attribution de ticket : Permet en affichant la liste des supports, d’attribuer le ticket

Langage et technologies utilisées

React Native

Développement multiplateforme : React Native permet de créer des applications mobiles pour iOS et Android à partir d'un seul code source JavaScript, ce qui réduit considérablement le temps et les coûts de développement. Cette approche garantit une cohérence de l'expérience utilisateur sur toutes les plateformes.
Performances proches du natif : Grâce à son bridge natif et à l'utilisation de composants natifs, React Native offre des performances fluides et adaptées aux besoins des applications professionnelles.
Écosystème et communauté active : Tout comme React, React Native bénéficie d'un écosystème riche et d'une grande communauté. De nombreuses bibliothèques et outils sont disponibles pour intégrer des fonctionnalités courantes comme la navigation, la géolocalisation, ou les notifications push.

Expo

Expo est un ensemble d'outils et de services qui simplifie le développement mobile en environnement JavaScript. Il permet un démarrage rapide, une gestion simplifiée des permissions natives, et l'accès à des fonctionnalités avancées comme la géolocalisation, les notifications push ou encore le déploiement en OTA (Over-the-Air).
Grâce à Expo, le processus de développement, de test et de déploiement est plus rapide et plus fluide, ce qui en fait un atout précieux pour les équipes de développement mobile.

Firestore

Firestore est une base de données NoSQL en temps réel proposée par Google via Firebase. Elle permet de stocker et synchroniser des données entre les utilisateurs en temps réel, tout en offrant une structure flexible adaptée aux applications mobiles.
Grâce à sa scalabilité automatique, sa gestion fine des règles de sécurité, et son intégration transparente avec les autres services Firebase, Firestore constitue une solution idéale pour gérer efficacement les tickets, les utilisateurs et les commentaires dans une application collaborative.

Ces technologies offrent une base solide pour le développement de l'application mobile. La solution multiplateforme permet un déploiement rapide et cohérent sur iOS et Android, tandis qu'Expo simplifie le cycle de développement et d'intégration des fonctionnalités natives. Enfin, Firestore assure une gestion efficace, sécurisée et en temps réel des données. Ensemble, elles permettent de créer une application moderne, performante et évolutive, parfaitement adaptée aux besoins métier et aux exigences des utilisateurs.

04

MA BDD

MCD, MLD, schéma

L'utilisation de Firestore, une base de données NoSQL orientée documents, modifie l'approche traditionnelle de modélisation des données. Contrairement aux bases relationnelles, Firestore ne repose pas sur des tables ni sur des relations explicites définies par des clés primaires et étrangères, ce qui rend difficile voire inadaptée la création d'un MCD (Modèle Conceptuel de Données), d'un MLD (Modèle Logique de Données) ou d'un schéma relationnel classique.

Dans Firestore, les données sont structurées en collections contenant des documents qui peuvent eux-mêmes contenir des sous-collections ou des attributs de types variés, comme des chaînes de caractères, des booléens, des tableaux... mais aussi des références à d'autres documents.
Ces références permettent de simuler une forme de relation entre entités, par exemple entre un ticket et son créateur (createdBy) ou entre un ticket et l'utilisateur à qui il est assigné (assignedTo). Elles n'impliquent toutefois aucune contrainte d'intégrité automatique, comme c'est le cas dans une base relationnelle : leur gestion doit être assurée explicitement au niveau de l'application.

Ainsi, bien que Firestore permette de représenter des liens logiques entre données grâce aux références, ces liens ne justifient pas à eux seuls la création d'un MCD ou d'un MLD complet. La souplesse et la flexibilité du modèle de données NoSQL sont en effet au cœur de son fonctionnement, rendant les schémas stricts moins pertinents dans ce contexte.

Exemple d’application des référencesId

```
// Collection produits
{
  _id: ObjectId("p1"),
         titre: "Smartphone XYZ",
        prix: 699,
        fabricant_id: ObjectId("m1")
}

// Collection fabricants
{
  _id: ObjectId("m1"),
         nom: "TechCorp",
        pays: "USA",
```

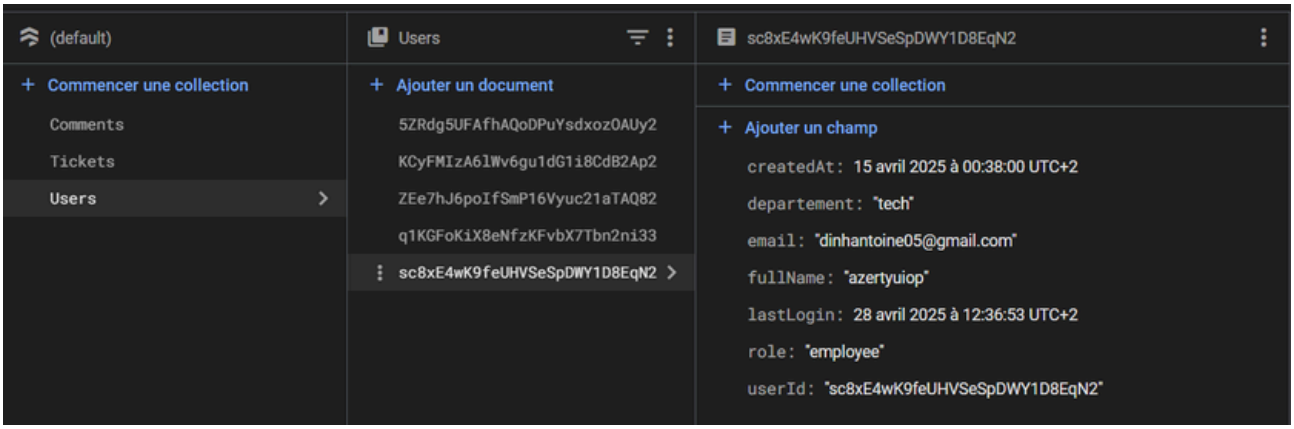
Interfaces dans le code pour gérer les relations

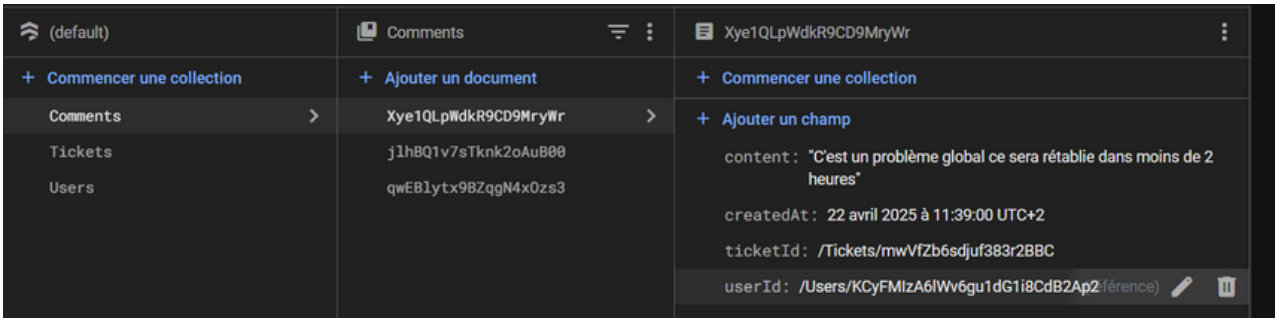
```
export interface comments {
    id?:string,
    ticketId: DocumentReference;
    userId: DocumentReference;
    content: string;
    createdAt: Timestamp;
    attachmentUrl?: string;
}
```

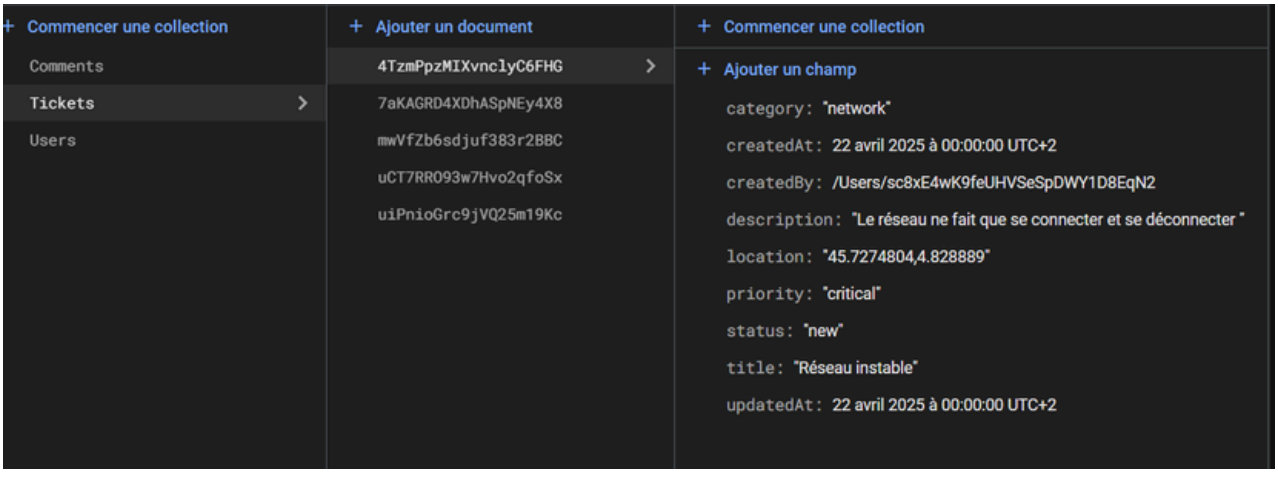
```
export interface User {
    userId:string,
    email: string;
    fullName: string;
    department: string;
    role: 'employee' | 'support' | 'admin'
    createdAt: Timestamp;
    lastLogin: Timestamp;
```

```typescript
export interface Ticket {
  id?: string;
  title: string;
  description: string;
  status: 'new' | 'assigned' | 'in-progress' | 'resolved' | 'closed';
  priority: 'low' | 'medium' | 'high' | 'critical';
  category: 'hardware' | 'software' | 'network' | 'access' | 'other';
  createdBy: DocumentReference;
  assignedTo?: DocumentReference;
  createdAt: Timestamp;
  updatedAt: Timestamp;
  dueDate?: Timestamp;
  location?: string;
```

Schéma dans la base firestore







05

ORGANISTAION DU CODE

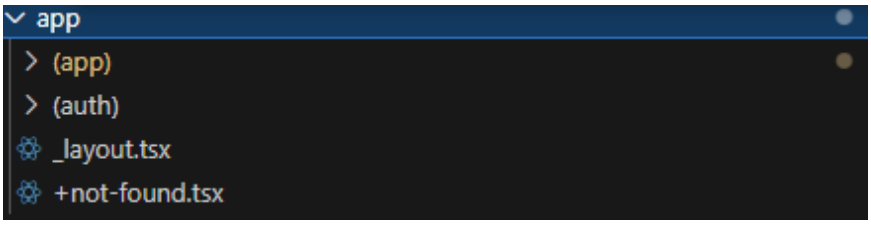
Organisation du code

React-native

L'utilisation de React-native permet une organsation du code avec division par module qui chacun a une utilité

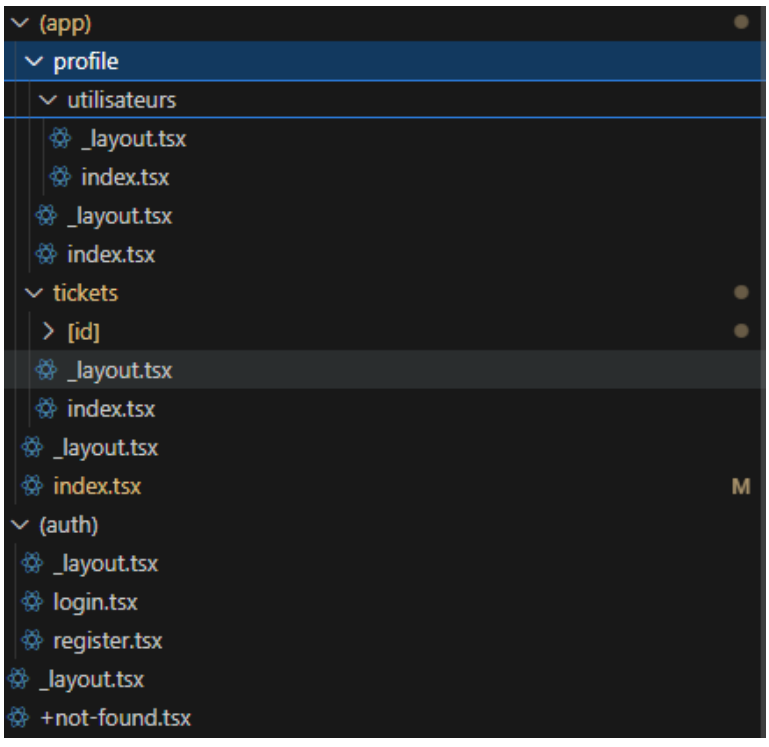
app/

Ce dossier est le cœur de la navigation avec Expo Router. Chaque fichier ou dossier ici correspond à une "route" dans ton application.



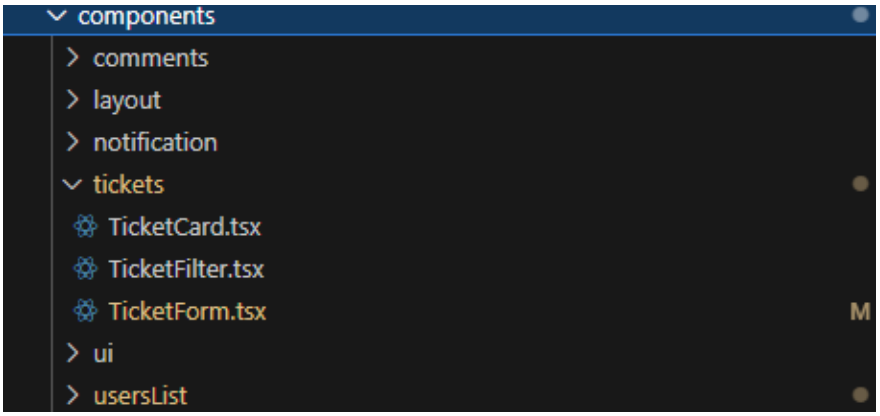
Le corps de l'application

Les différents pages sur lesquelles l'utilisateur va pouvoir naviguer



Components

Les composants seront ce qui sera le plus souvent les éléments affichés sur le frontend, il sera appelé sur le corps de l'application



Services

Les services servent à récupérer les données en backend de la base de données pour les envoyer ou de prendre des paramètres de frontend pour faire des modifications sur la base de donnée

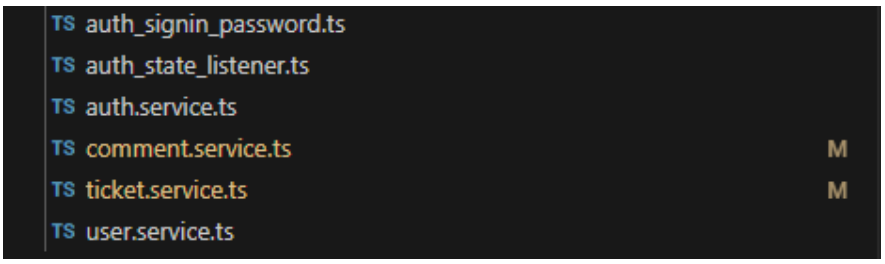
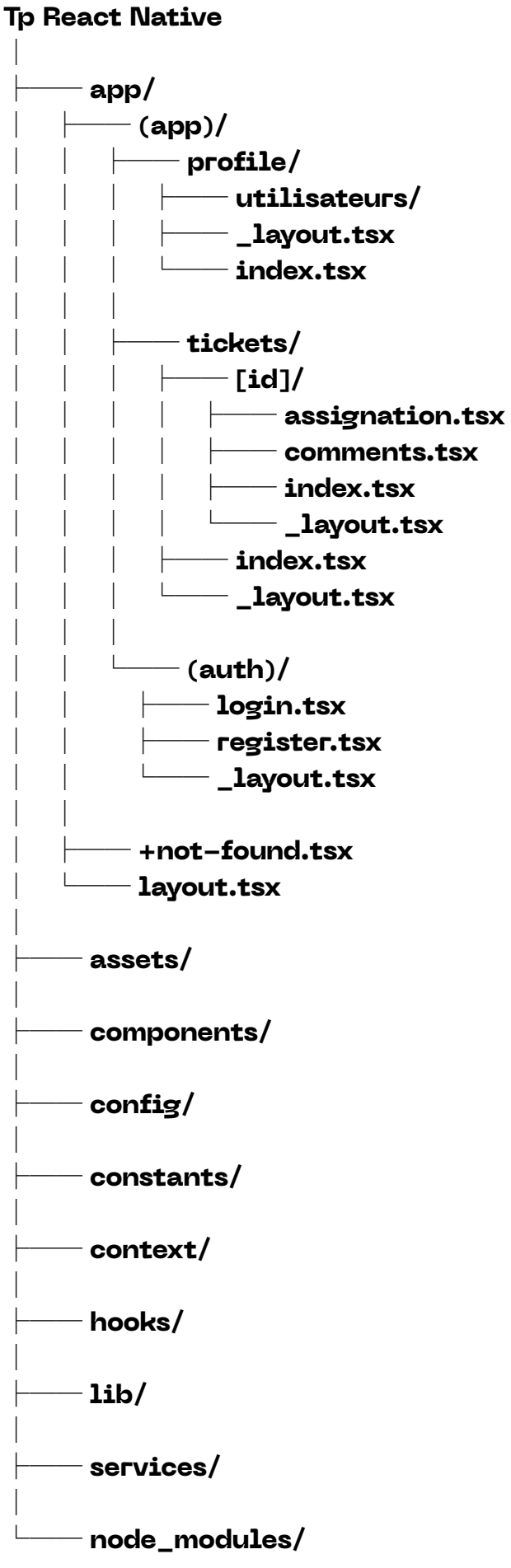


Schéma d'arborescence :



06

PRÉSENTATION DU CODE

Présentation du code

LOGIN / INSCRIPTION

La gestion de l'authentification pour la connexion utilise le service de firebase.

```
const handleLogin = async () => {
  if (!email || !password) {
    Alert.alert("Erreur", "Veuillez remplir tous les champs.");
    return;
  }

  setLoading(true);
  try {
    const userCredential= await signInWithEmailAndPassword(auth, email, password);
```

Concernant l'inscription, une utilisation d'un service de firebase et aussi utilisé mais il y a aussi une création dans la base de donnée de l'utilisateur

```
const userCredential = await createUserWithEmailAndPassword(auth, email, password);
const user = userCredential.user

await setDoc(doc(db, "Users", user.uid), {
  email: email,
  fullName: fullName,
  departement: departement,
  role: "employee",
  createdAt: new Date(),
  lastLogin: new Date(),
  userId: user.uid
});
```

L'affichage de liste de tickets

Fonctionnalité de base de l'application, permet de récupérer toutes les données dans ma table Tickets

```ts
const getAllTickets = async (): Promise<TicketTrue[]> => {
  //je ne le mets pas en abonnement en temps réel pour laisser l'utilité du pullToRefresh
  const ticketsCollection = collection(db, "Tickets");
  const snapshot = await getDocs(ticketsCollection);

  return snapshot.docs.map((doc) => ({
    id: doc.id,
    ...(doc.data() as TicketTrue),
  }));
};
```

Dans mon frontend

```ts
  const tickets = await getAllTickets();
  //filtrage selon le rôle
  let filtered = tickets;
  if (role === "employee") {
    filtered = tickets.filter(ticket => ticket.createdBy.id === user?.uid);
  } else if (role === "support") {
    filtered = tickets.filter(ticket => ticket.assignedTo?.id === user?.uid);
  }
  setYourTicketsData(filtered);
  setInitialTicketsData(filtered);
};
```

Triage des données en toggle

Fonctionnalité qui permet de trier les données selon le mode voulue, l'exemple ici sera sur la priorité

```ts
const priorityMap = new Map<string, number>([
  ["critical", 1],
  ["high", 2],
  ["medium", 3],
  ["low", 4],
]);
```

```ts
const sortByPriority = () => {
  if (!isPrioritySorted) {
  const sorted = [...yourTicketsData].sort((a, b) => {
    const aValue = priorityMap.get(a.priority.toLowerCase()) ?? 999;
    const bValue = priorityMap.get(b.priority.toLowerCase()) ?? 999;
    return aValue - bValue;
  });
  setYourTicketsData(sorted);
}else {
  setYourTicketsData(initialTicketsData)
}
setIsPrioritySorted(!isPrioritySorted)
setIsStatusSorted(false);
};
```

Barre de recherche

Fonctionnalité qui permet de changer les données affichés en temps réel selon la query donnée

```
useEffect(() => {
  handleSearch(searchQuery
}, [yourTicketsData]);
```

```
const handleSearch = (query: string) => {
  setSearchQuery(query);
  if (query === "") {
    setFilteredTickets(yourTicketsData);
  } else {
    const filtered = yourTicketsData.filter(ticket =>
      ticket.title.toLowerCase().includes(query.toLowerCase())
    );
    setFilteredTickets(filtered);
  }
};
```

```
export async function POST(req: NextRequest) {
  try {
    const body = await req.json();
    const newCommande = await CreateCommande({
      id_utilisateur: body.id_utilisateur,
      quantite: body.quantite,
      id_stock: body.id_stock,
    });
    return NextResponse.json(newCommande, { status: 201 });
  } catch (error) {
    console.error("Error creating commande:", error);
    return NextResponse.json(
      { error: "Failed to create commande" },
      { status: 500 }
    );
  }
}
```

L'affichage de liste

Dans la majorité des fonctionnalités il y la possibilité de voir la liste d'une table associé, prenons par exemple "Stocks", dans ces situations j'utilise une interface personnalisé car ma table est en int8 ce qui n'est pas bien pris avec les fonctions et plus difficile à utiliser

```
enum type {
  medicament,
  materiel,
}

export interface SerializedStocks {
  id_stock: number;
  nom: string;
  description: string;
  quantite_disponible: number;
  type: type;
}
```

```
export async function GetAllStocks(): Promise<SerializedStocks[]> {
  try {
    const stocks = await prisma.stocks.findMany();
    const serializedStocks: SerializedStocks[] = JSON.parse(
      JSONbig.stringify(stocks)
    );
    return serializedStocks;
  } catch (error) {
    console.error(error);
    throw new Error("Failed to fetch stocks");
  }
}
```

Insertion d'élément

Il s'agit d'une fonctionnalité utilisé pour les tickets ou encore les commentaires, dans l'exemple il s'agira de ticket

```
const handleAddTicket = async (ticket: TicketFirst) => {
  await createTicket({ title: ticket.title,
                       description: ticket.description,
                       status: ticket.status,
                       priority: ticket.priority,
                       category: ticket.category,
                       createdBy:user?.uid,
                       location: ticket.location,
                       dueDate: ticket.dueDate });
  setIsModalVisible(false)
};
```

```
const createTicket = async (ticket: TicketFirst): Promise<TicketTrue | null> => {
 try {
  const ticketsCollection = collection(db, "Tickets");
  if (!ticket.createdBy || typeof ticket.createdBy !== 'string') {
    throw new Error("une erreur sur l'utilisateur");
  }
  const userRef = doc(db, "Users", ticket.createdBy);
  const ticketData: TicketFirst = {
    title: ticket.title,
    description: ticket.description,
    status: ticket.status,
    priority: ticket.priority,
    category: ticket.category,
    createdBy: userRef,
    createdAt: Timestamp.fromDate(dateOnly),
    updatedAt: Timestamp.fromDate(dateOnly),
  };
  if (ticket.location) {
    ticketData.location = ticket.location;
  }
  if (ticket.dueDate) {
    ticketData.dueDate = ticket.dueDate;
  }
  await addDoc(ticketsCollection, ticketData);
  await notifyLocalTicket(ticketData.title)
```

Edition de ticket

L'édition sera sur possible après avoir cliqué sur un ticket en particulier, il permettra de changer les données si des rectifications sont nécessaires

```
nPress: async () => {
 await updateTicket(idTicket, updatedTicket);
 const updated = await getDetailTicket(idTicket) as TicketFirst;
 if (updated) {
   setTicket(updated);
 }
 setIsEditModalVisible(false);
```

```
const ticketRef = doc(db, "Tickets", idTicket);
const now = new Date();
const dateOnly = new Date(now.getFullYear(), now.getMonth(), now.getDate());

const updatePayload: any = {
  title: updatedData.title,
  description: updatedData.description,
  status: updatedData.status,
  priority: updatedData.priority,
  category: updatedData.category,
  updatedAt: Timestamp.fromDate(dateOnly),
};

if (updatedData.assignedTo) {
  updatePayload.assignedTo =
    typeof updatedData.assignedTo === "string"
      ? doc(db, "Users", updatedData.assignedTo)
      : updatedData.assignedTo;
}

if (updatedData.dueDate) {
  updatePayload.dueDate = updatedData.dueDate;
}

await updateDoc(ticketRef, updatePayload);
```

Suppression de ticket

La suppression sera possible en tant qu'employé créateur du ticket ainsi que pour l'admin

```
  const checker = await deleteTicket(idTicket);
  if (checker) {
    setTicket(null);
    setLoading(true);
    goToTicketsIndex();
  }
```

```
const deleteTicket = async (idTicket:string) : Promise<boolean> => {
  try {
    await deleteDoc(doc(db, "Tickets", idTicket));
    return true;
  } catch (error) {
    console.error("Erreur lors de la suppression :", error);
    return false;
  }
};
```

L'assignation du ticket

En tant qu'admin, l'assignation est possible pour mettre un support sur le ticket

```
if (!idTicket || typeof idTicket !== "string") return;
try {
  await assignSupportToTicket(idTicket, user.userId);
```

```
const assignSupportToTicket = async (ticketId: string, supportUserId: string) => {
  try {
    const ticketRef = doc(db, "Tickets", ticketId);
    const supportRef = doc(db, "Users", supportUserId);

    await updateDoc(ticketRef, {
      assignedTo: supportRef,
      status: "assigned",
    });
```

Mettre les utilisateurs en support

En tant qu'admin, mettre des utilisateurs en tant que support est possible, il y avec cela un message d'erreur si il essaye de mettre de un admin en support.

```
if (selectedUser.role === "admin") {
    Alert.alert(
        "Action non autorisée",
        "Vous ne pouvez pas changer le rôle d'un administrateur.",
        [{ text: "OK" }]
    );
    setModalVisible(false);
    return;
}

try {
    await updateUser(selectedUser.userId);
    setModalVisible(false);
```

```
const updateUser = async (userId: string): Promise<void> => {
  if (!userId) throw new Error("ID d'utilisateur manquant.");

  const userRef = doc(db, "Users", userId);

  await updateDoc(userRef, { role: "support" });
```

Les notifications

Les notifications sont dans les components, il en existe par situation nécessaire à une notification et comme peut être vu dans les exemples est appelé dans les services

```
const notifyLocalAssignation = async (ticketTitle: string) => {
 await Notifications.scheduleNotificationAsync({
   content: {
     title: " Assignation réussie",
     body: `Le ticket "${ticketTitle}" a été assigné avec succès.`,
     sound: "default",
   },
   trigger: null,
 });
;
```

Des configurations sont toutefois nécessaires

```
 "android": {
   "permissions": ["ACCESS_FINE_LOCATION","NOTIFICATIONS"],
```

```
Notifications.setNotificationHandler({
  handleNotification: async () => ({
    shouldShowAlert: true,
    shouldPlaySound: true,
    shouldSetBadge: false,
  }),
});
```

Les layout

Ils permettent une présentation des vues et écrans données avec plus de cohérence et de personnalisation, ils donnent avec les Stack, un type de navigation par pile ou avec les tabs, une navigation à niveau égale pour une expérience utilisateur optimale

```
export default function RootLayout() {
  return (
    <AuthProvider>
      <Stack>
        <Stack.Screen name="(auth)" options={{ headerShown: false }} />
        <Stack.Screen name="(app)" options={{ headerShown: false }} />
        <Stack.Screen name="+not-found" options={{ headerShown: false }} />
      </Stack>
      <StatusBar style="light" />
    </AuthProvider>
  );
```

```
<Tabs
    screenOptions={{
        tabBarActiveTintColor: "#ffd33d",
        headerStyle: {
            backgroundColor: "#25292e",
        },
        headerShadowVisible: false,
        headerTintColor: "#fff",
        tabBarStyle: {
            backgroundColor: "#25292e",
        },
    }}
>
    <Tabs.Screen
        name="index"
        options={{
            title: "Dashboard",
            tabBarIcon: ({ color, focused }) => (
                <Ionicons
                    name={focused ? "home-sharp" : "home-outline"}
                    color={color}
                    size={24}
                />
            ),
        }}
    />
```

07

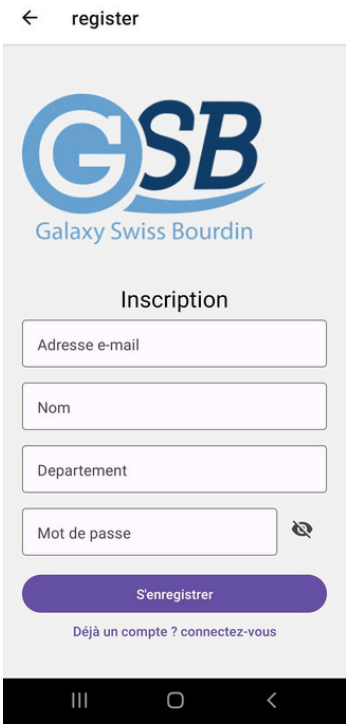
PRÉSENTATION APPLICATION

Présentation de l'application

Connexion

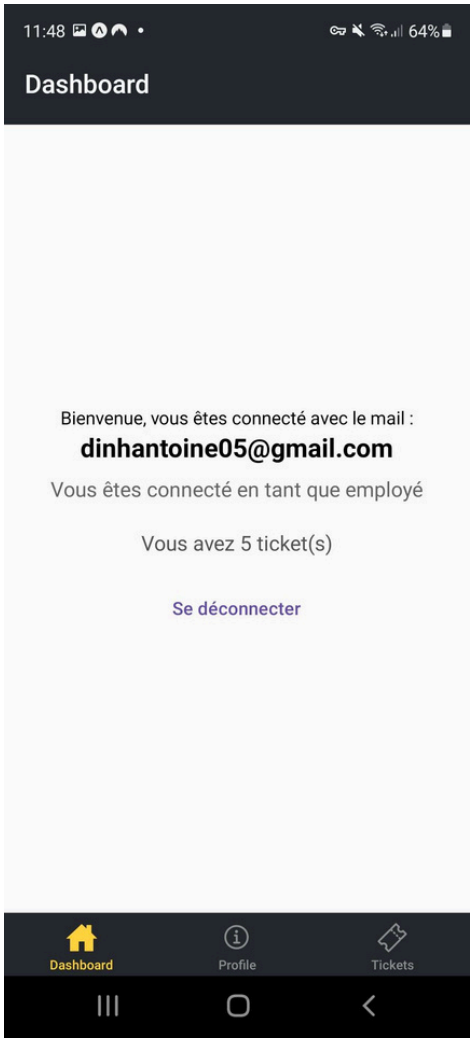
Au lancement de l'application, la page de Login s'affiche avec 2 zones de textes dont 1 en texte cachée et 2 boutons.

Un click sur le bouton Se connecter redirigera vers le bonne page si les champs mis sont bons, il y de plus une option pour créer un compte avec Inscription.



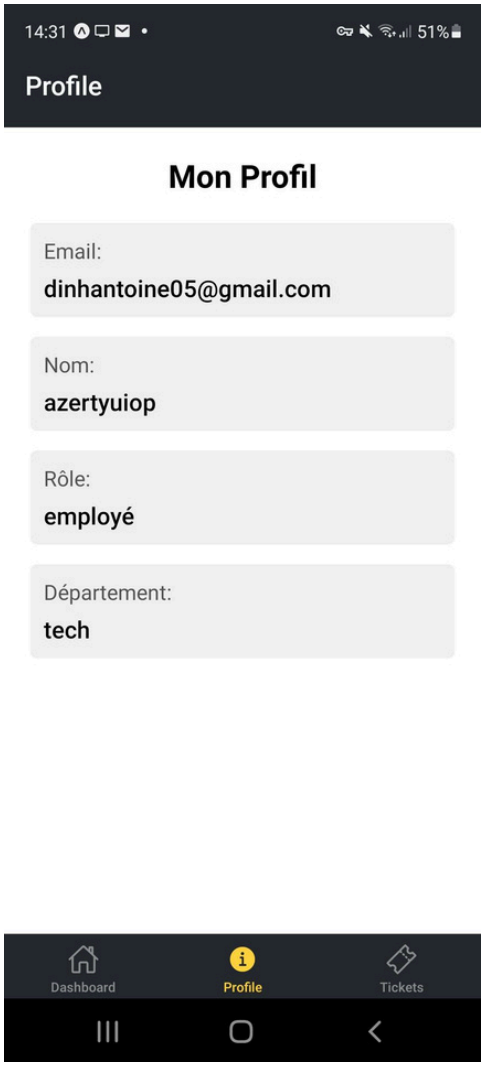
Dashboard

Dans les tabs disponibles, le premier possible est dashboard avec la possibilité de se déconnecter ainsi que que le nombre de tickets accessibles



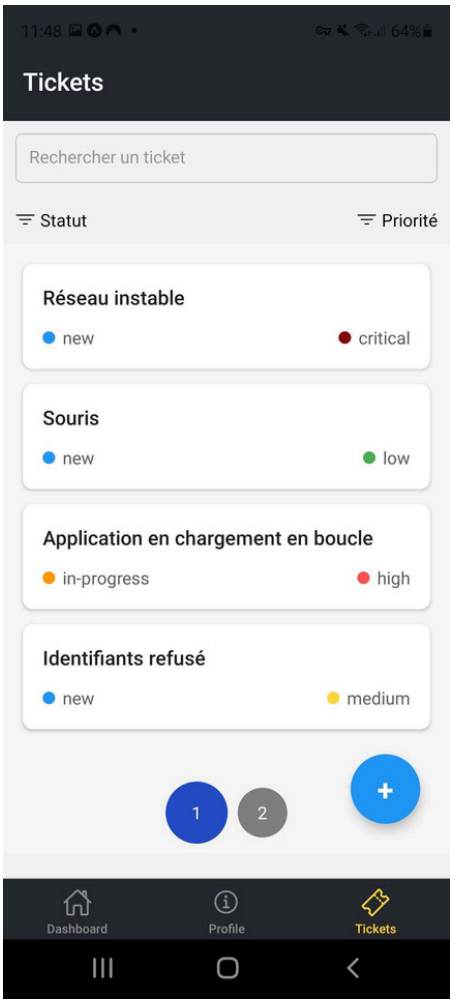
Profile

Une autre tabs pouvant être cliqué est profile, les détails de la personne est alors affichée



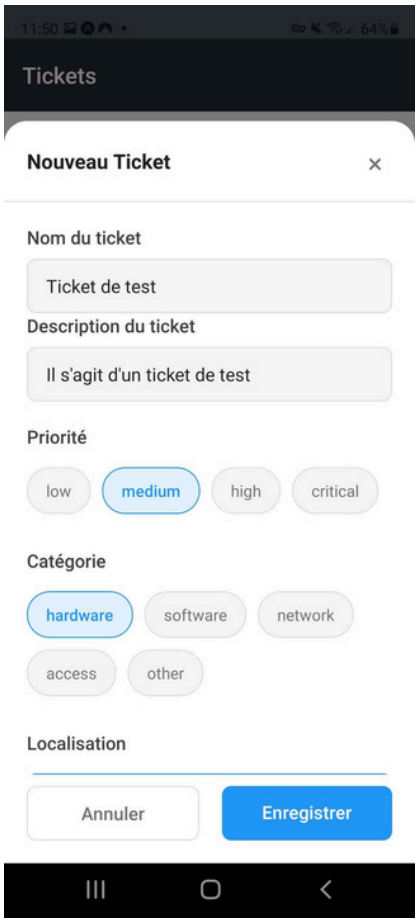
Liste des tickets

Cette fonctionnalité permet d'afficher tous les tickets avec une barre de recherche, de trier selon le statut ou la priorité ainsi que la possibilité d'ajouter un nouveau ticket si l'utilisateur est un employé.

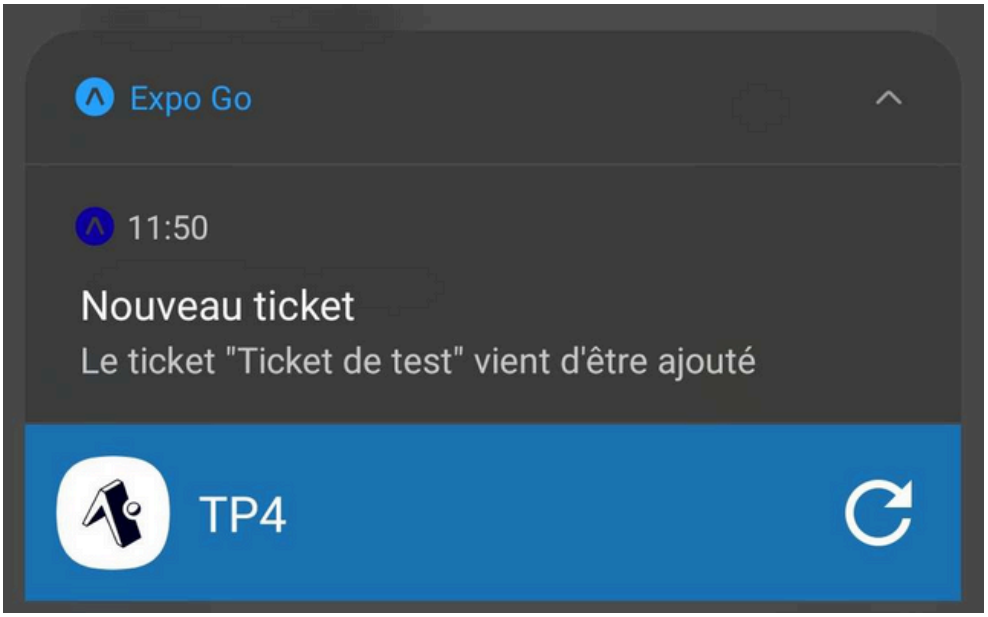


Ajouter un ticket

Quand l'utilisateur clique sur le + il peut alors ajouter grâce à un form un ticket selon les options disponibles



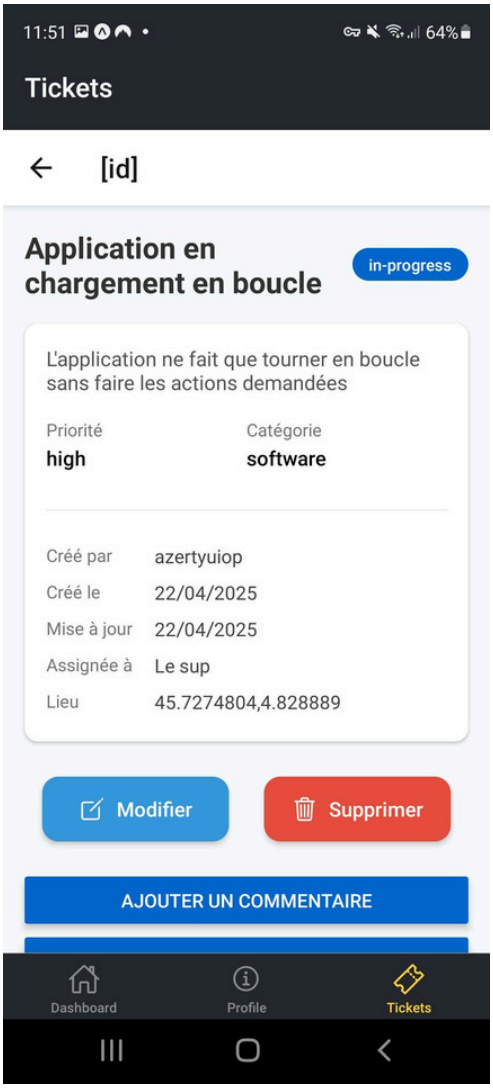
Après certaines actions une notification est envoyée qui permet de s'assurer que l'action a bien fonctionné



La liste affichée n'a pas forcément mise à jour alors faire un pull to refresh pour mettre à jour la liste

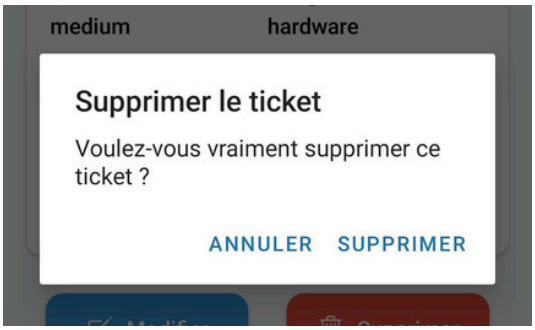
Détail du ticket

En cliquant sur un ticket, les détails attachés à celui-ci sera alors affiché avec des fonctionnalités en plus selon le rôle



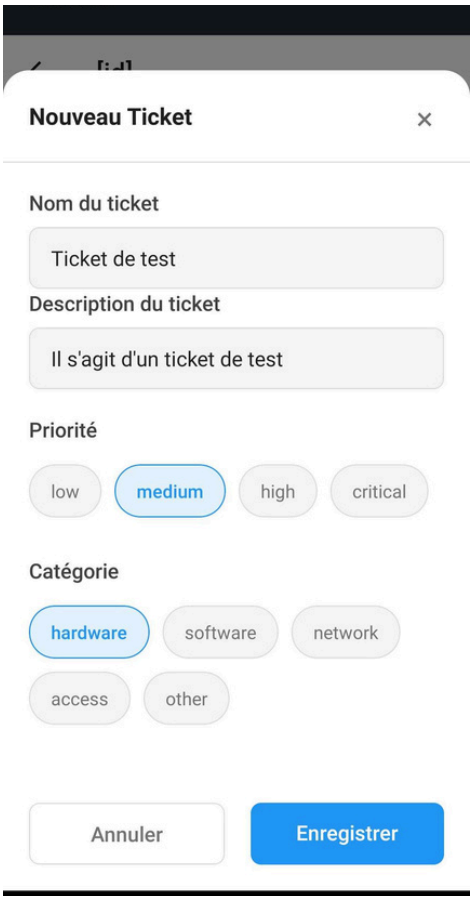
Suppression

Une suppression est possible pour les utilisateurs et l'admin



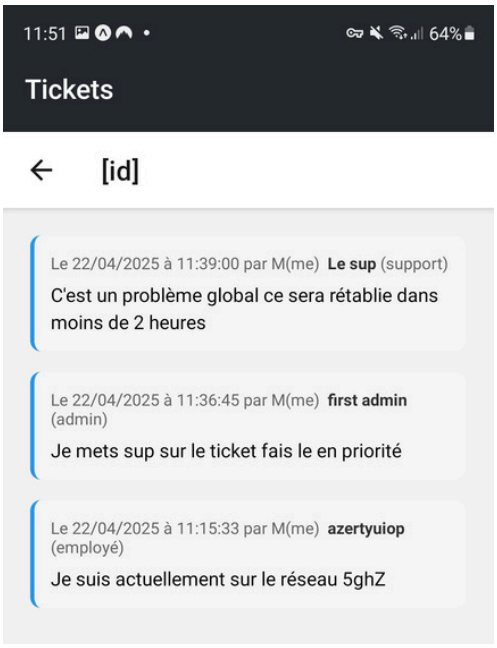
Edition

Une rectification du ticket est possible



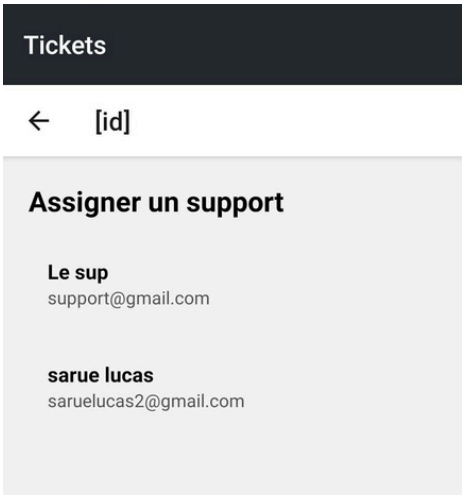
Commentaires

2 boutons sont associés à cette fonctionnalité, d'abord celle de création de commentaire qui ouvre un modal puis d'une qui apparaît uniquement si des commentaires existent déjà et qui affichent la liste des commentaires avec les personnes qui les ont ajouté



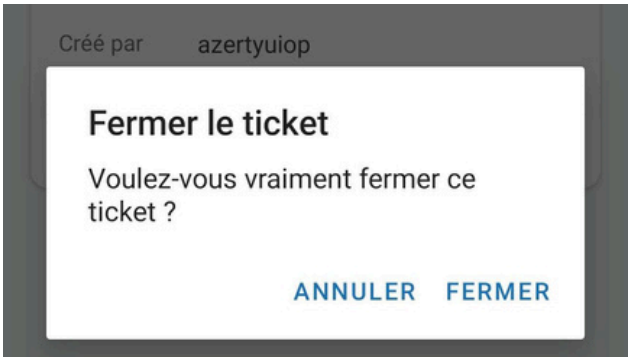
Assignation

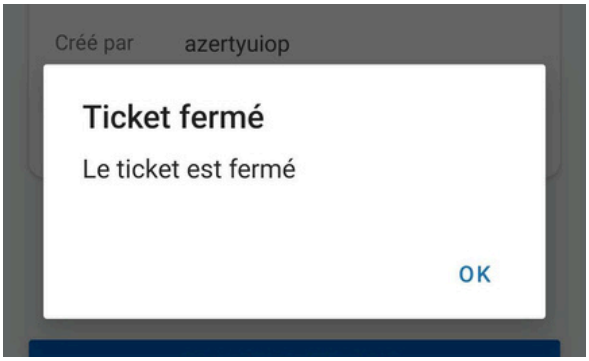
Une fonction disponible des admins pour assigner un ticket à un support



Fermeture

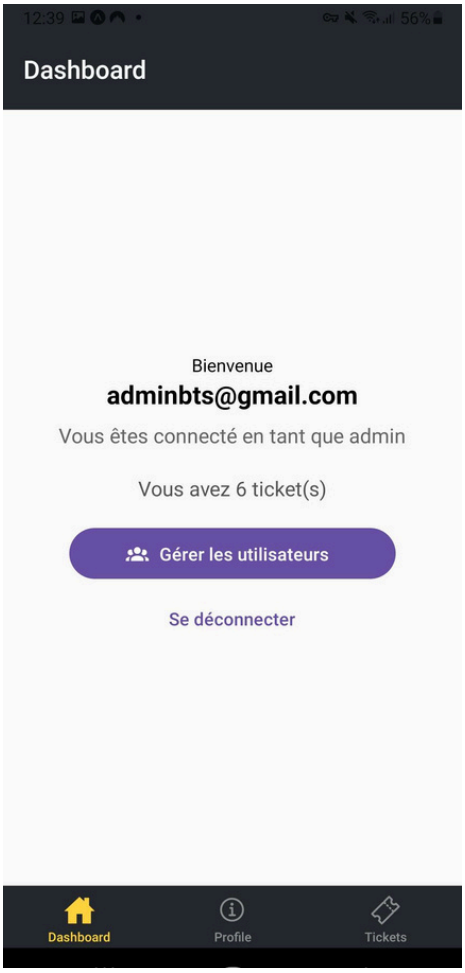
Une fonction disponible des admins pour fermer un ticket, un ticket fermé ne pourra plus être modifié mais peut être supprimé

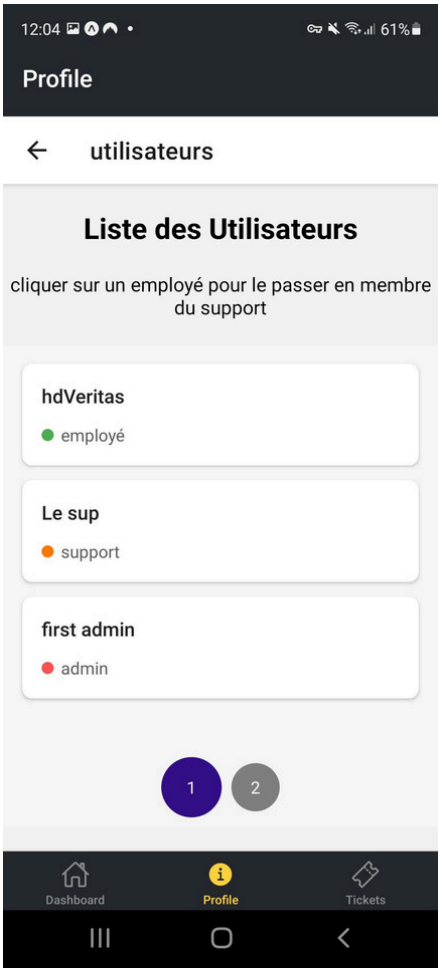


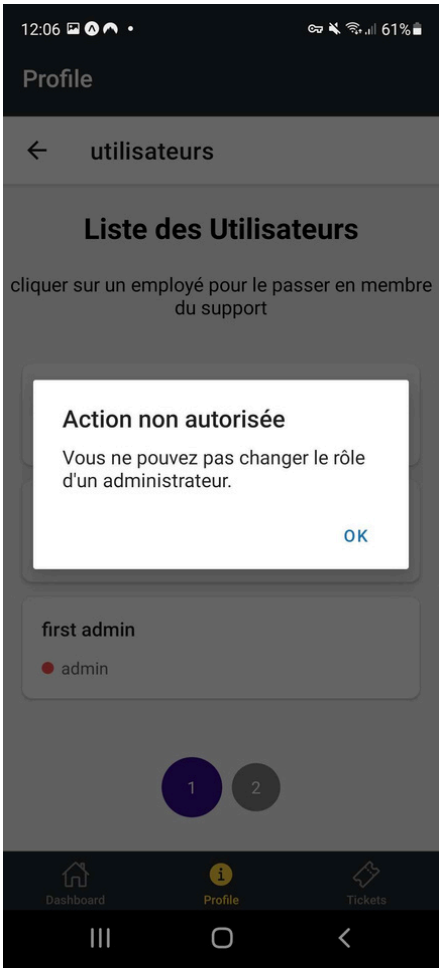


Changement de statut

Cette fonctionnalité est accessible dans le dashboard de l'admin, elle permet de mettre l'utilisateur choisie en support, cela est sécurisée en ce qui concerna le mise en support de l'admin.







Lancement de l'application

Allez sur ce lien github :

https://github.com/Goyef/MedSupport

Téléchargez le zip et l'extraire

Dans l'invite de commande de l'application faire npm i

Le lancement se fait avec cd '.\Tp React Native\'

npx expo start --tunnel
puis avec le QR code prendre avec expo go

créer un .env à la racine du projet avec à l'intérieur :

EXPO_PUBLIC_FIREBASE_API_KEY =
AIzaSyDSxMM5L0_BVFnqB4Rms2fpkcLZwKO15rU
EXPO_PUBLIC_FIREBASE_AUTH_DOMAIN = gsb-support.firebaseapp.com
EXPO_PUBLIC_FIREBASE_PROJECT_ID = gsb-support
EXPO_PUBLIC_FIREBASE_STORAGE_BUCKET = gsb-support.firebasestorage.app
EXPO_PUBLIC_FIREBASE_MESSAGING_SENDER_ID = 469372660263
EXPO_PUBLIC_FIREBASE_APP_ID =
1:469372660263:web:102e51eb3bea7c2ebffc02
EXPO_PUBLIC_FIREBASE_MEASUREMENT_ID = G-281X44XRBP

lien apk : https://expo.dev/artifacts/eas/2V5GYLaPDP6SANmPK3dMZL.apk
(pour l'apk Assurez vous bien d'avoir un android en mode développement, allez dans les paramètres et appuyez sur "informations sur le logiciel" puis cliquez 7 fois sur numéro de version)

Lancez avec l'application expo go

Les identifiants sont
pour l'employé: dinhantoine05@gmail.com
mot de passe : azertyuiop

pour le support: support@gmail.com et en mot de passe : azertyuiop

pour l'administrateur: adminbts@gmail.com et en mot de passe : azertyuiop

08

ANNEXES

GITHUB

Lien github dans l'éventualité où il y a un problème avec le code envoyé :

https://github.com/Goyef/MedSupport

Lien de l'apk :
https://expo.dev/artifacts/eas/2V5GYLaPDP6SANmPK3dMZL.apk

ANTOINE DINH

Avril 2025

Isitech

BTS SIO SLAM