PROJET ACCOMPAGNÉ

PROJET PERSONNEL ENCADRÉ

Projet : GSB Feedback

Fait par

Liogier Nolan

Sommaire

BTS SERVICES INFORMATIQUES AUX ORGANISATIONS SESSION 2025
ANNEXE 9-1-B : Fiche descriptive de réalisation professionnelle (recto)
Épreuve E5 - Conception et développement d'applications (option SLAM)

DESCRIPTION D'UNE RÉALISATION PROFESSIONNELLE	**N° réalisation :**
Nom, prénom : Liogier Nolan	**N° candidat :**
Épreuve ponctuelle ☒ **Contrôle en cours de formation** ☐	**Date :** 09 / 02 / 2026

Organisation support de la réalisation professionnelle

Ce projet s'inscrit dans le cadre du Projet Personnalisé Encadré se basant sur un contexte entrepreneurial fictif donné par le centre de formation. L'entreprise en question était le Laboratoire Galaxy Swiss Bourdin (GSB) et l'objectif fut de concevoir une application mobile permettant de consulter et d'effectuer un compte rendu de visites médicales

Intitulé de la réalisation professionnelle

Développement de l'application « GSB Feedback ».

Période de réalisation : dec 2025 – fev 2026 **Lieu :** AFIP FORMATIONS

Modalité : ☒ **Seul(e)** ☐ **En équipe**

Compétences travaillées

☒ Concevoir et développer une solution applicative
☒ Assurer la maintenance corrective ou évolutive d'une solution applicative
☒ Gérer les données

Conditions de réalisation (ressources fournies, résultats attendus)

Ressources fournies : Contexte GSB de l'éducation nationale, formations
Résultats attendus : Application mobile complète Frontend, Backend, Tests , Versionning, Outils de gestion de projet.

- Interface utilisateur
- Documentation utilisateurs
- Sécurité
- Tests unitaires

Description des ressources documentaires, matérielles et logicielles utilisées

- Serveur local : Node.js
- Gestion de version et collaboration : GitHub pour la sauvegarde et le Versionning
- Langage de programmation :
  - Frontend : Flutter / Dart
  - Backend: Node.js
- Base de donnée : PostgresSQL
- Gestionnaire de librairies : npm
- Librairies : Express, GoRouter, l10n, urlLuncher
- Outils : Postman, Cursor IDE, Docker
- Modélisation de données : Draw.io
- Gestion de projet : Notion

Modalités d'accès aux productions et à leur documentation

Github : https://github.com/NolanLiogier/gsb-feedback-mobile-app

Portfolio : https://nolanliogier.fr/

BTS SERVICES INFORMATIQUES AUX ORGANISATIONS SESSION 2025

ANNEXE 9-1-B : Fiche descriptive de réalisation professionnelle
(verso, éventuellement pages suivantes)
Épreuve E5 - Conception et développement d'applications (option SLAM)

Descriptif de la réalisation professionnelle, y compris les productions réalisées et schémas explicatifs

Le projet consistait à développer une application mobile sécurisée destinée à la gestion centralisée des visites médicales au sein de l'entreprise pharmaceutique fictive Galaxy Swiss Bourdin (GSB). L'application permet la visualisation des entreprises clientes, la gestion des visites médicales, de leur planification, de leur affectation ainsi que le suivi des feedbacks.

Front-end

Pour la conception de l'interface utilisateur, j'ai utilisé Flutter, un framework permettant de développer une application mobile performante et responsive. Il m'a permis de créer une interface claire, adaptée aux smartphones et organisée selon les différents rôles utilisateurs.

Back-end

Côté serveur, l'application repose sur Node.js avec Express. La communication avec la base de données Postgres est réalisée via une API REST utilisant le format JSON, garantissant une gestion structurée et sécurisée des données.

Authentification

L'application intègre un système d'authentification sécurisé, les mots de passe sont hachés avant leur enregistrement en base de données.

Rôles

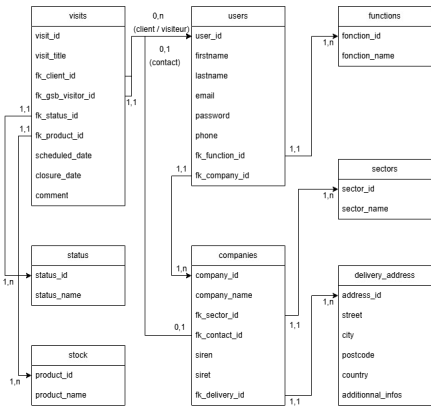
- Client : gestion de ses visites et visualisation de son entreprise
- Visiteur médical : consultation des visites et création de feedback
- Responsable : création des visites, affectation des visiteurs et suivi global

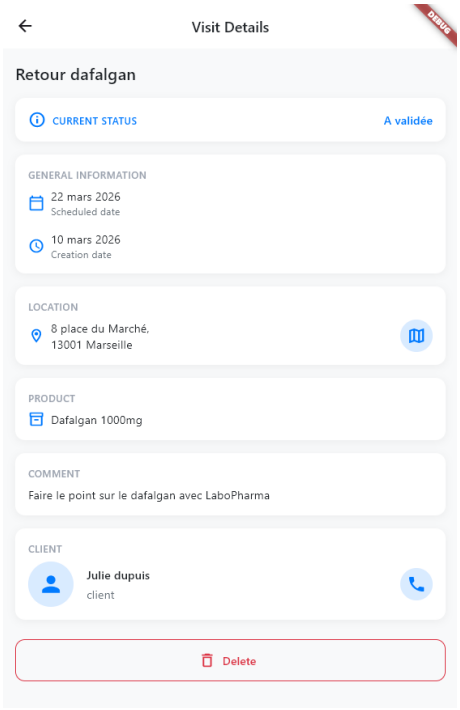
L'accès aux fonctionnalités est contrôlé selon le rôle et l'état de connexion.

Fonctionnalités principales

L'application permet :

- La visualisation des entreprises clientes
- La création, modification et suivi des visites médicales
- La gestion des statuts des visites
- L'affectation des visiteurs médicaux
- La création et consultation des feedbacks
- L'affichage des visites via planning





II. Cahier des charges

a) Présentation de Galaxy Swiss Bourdin (GSB)

Galaxy Swiss Bourdin (GSB) est un laboratoire pharmaceutique issu de la fusion, en 2009, de deux acteurs majeurs du secteur : l'américain Galaxy, spécialisé dans les traitements contre les maladies virales telles que le SIDA et les hépatites, et le conglomérat européen Swiss Bourdin, expert en médicaments d'usage courant.

Ce rapprochement a donné naissance à un leader de l'industrie pharmaceutique. Le siège administratif de Galaxy Swiss Bourdin Europe est basé à Paris, tandis que le siège mondial de la multinationale est situé à Philadelphie, aux États-Unis.

GSB s'appuie sur une force commerciale dédiée à l'information et au conseil auprès des professionnels de santé. En France métropolitaine (y compris la Corse), 480 visiteurs médicaux assurent ce rôle, auxquels s'ajoutent 60 représentants dans les départements et territoires d'outremer.

Leur mission consiste à informer médecins, pharmaciens, infirmiers et autres prescripteurs sur les produits du laboratoire. Bien qu'ils ne réalisent pas directement de ventes, leur travail influence significativement la prescription des médicaments développés par GSB.

L'organisation de l'entreprise repose sur une structure hiérarchique régionale, regroupée en plusieurs grands secteurs géographiques (Sud, Nord, Paris-Centre, Antilles-Guyane, etc.), garantissant ainsi une couverture eU'icace du territoire.

Les principaux concurrents du laboratoire en France sont les groupes Sanofi, Viatris et enfin le groupe Bayer.

b) Origine de la demande

Le secteur pharmaceutique est caractérisé par des exigences élevées en matière d'organisation, de traçabilité et de suivi des activités commerciales.

La gestion des visites médicales, des plannings des visiteurs médicaux ainsi que le suivi des retours clients (feedbacks) représentent un enjeu stratégique majeur afin de garantir une relation efficace avec les professionnels de santé et d'assurer la promotion optimale des produits du laboratoire.

Dans ce contexte, le laboratoire Galaxy Swiss Bourdin (GSB) souhaite optimiser et centraliser la gestion de ses visites médicales.

Pour répondre à ces besoins, l'entreprise a décidé de développer une application mobile nommée GSB Feedback, destinée à faciliter la gestion des demandes de visites, leur planification, leur affectation aux visiteurs médicaux ainsi que le suivi des comptes rendus et avis clients.

Sur décision de la direction, GSB fait appel à un développeur afin de concevoir une solution moderne, sécurisée et adaptée aux différents profils d'utilisateurs (responsables, visiteurs médicaux et clients).

Attentes :

- Centraliser la gestion des visites médicales au sein d'une seule application
- Faciliter la planification et le suivi des visites
- Permettre l'affectation des visiteurs médicaux par les responsables
- Offrir une visibilité globale des plannings
- Mettre en place un système structuré de feedback après chaque visite
- Sécuriser les données et gérer précisément les droits d'accès selon les rôles
- Améliorer la coordination entre responsables, visiteurs médicaux et clients
- Disposer d'une application mobile ergonomique et simple d'utilisation

Existant :

Avant la mise en place de l'application GSB Feedback, la gestion des visites médicales reposait sur des outils hétérogènes (tableurs, échanges d'e-mails, appels téléphoniques).

Cette organisation entraînait :

- Une dispersion des informations
- Un manque de visibilité sur les plannings
- Des difficultés dans le suivi des statuts des visites
- Une traçabilité limitée des comptes rendus et feedbacks
- Un risque d'erreurs dans l'affectation des visiteurs médicaux

Les responsables ne disposaient pas d'une vue centralisée des visites et des performances, tandis que les visiteurs médicaux ne bénéficiaient pas d'un outil unique leur permettant de consulter et gérer efficacement leurs rendez-vous.

Objectifs :

- Centraliser 100% de la gestion des visites médicales via l'application
- Réduire les erreurs liées à la dispersion des informations (emails, appels, fichiers externes)
- Améliorer la planification et le suivi des visites
- Optimiser la coordination entre responsables, visiteurs médicaux et clients
- Offrir aux clients une meilleure visibilité sur leurs visites et leur planning
- Mettre en place un suivi structuré des feedbacks après chaque visite
- Garantir la sécurité et la confidentialité des données
- Disposer d'une application mobile maintenable et évolutive

Cible :

Principales :

- Les visiteurs médicaux du laboratoire GSB
- Les clients (médecins / entreprises clientes)
- Les responsables commerciaux

Secondaires :

- Les administrateurs du système
- La direction de l'entreprise

c) Fonctionnalités principales

Gestion des visites médicales

Les responsables peuvent créer des demandes de visite via une interface dédiée en renseignant les informations nécessaires (client concerné, date souhaitée, objet de la visite).

Le client peut accepter, refuser ou proposer une nouvelle date tant que la visite n'est pas validée.

Une fois la date validée, le responsable affecte un visiteur médical.
Chaque visite possède un statut clair (en attente, validée, clôturée), permettant un suivi précis de son avancement.

Les visiteurs médicaux peuvent consulter les détails de leurs visites planifiées et accéder aux informations nécessaires avant leur réalisation.

Entreprises clientes

Le module permet la consultation des informations des entreprises clientes.

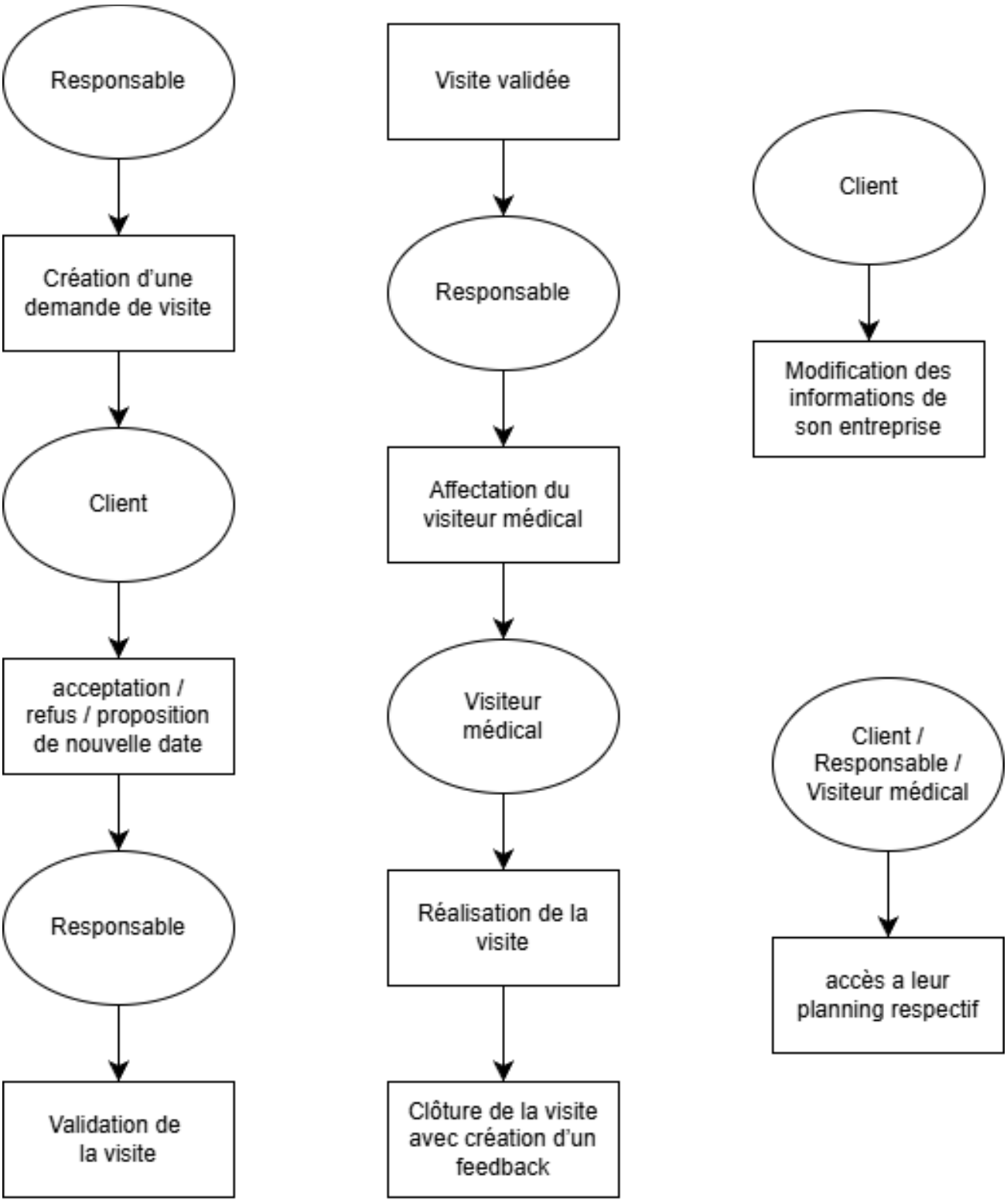
Les responsables disposent d'une vision globale de l'ensemble des entreprises.
Les clients ont un accès restreint aux seules données de leur propre entreprise et peuvent mettre à jour leurs informations si nécessaire.

Gestion des feedbacks

À l'issue d'une visite, le visiteur médical rédige un compte rendu (feedback) décrivant le déroulement de la visite et les éléments abordés.

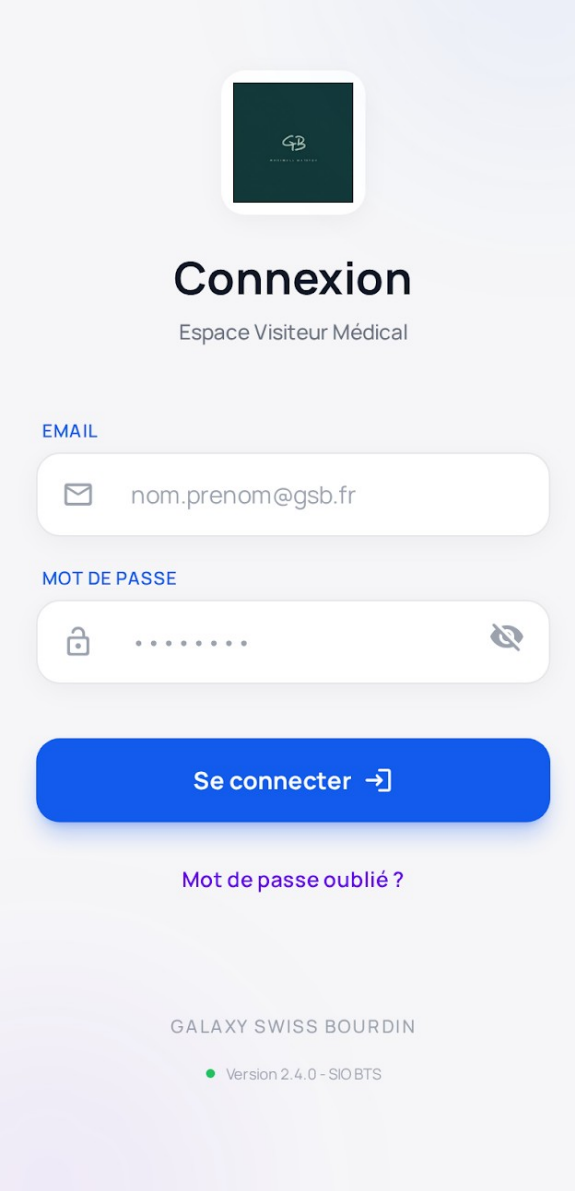
Le client peut consulter le feedback une fois la visite clôturée et donner son avis sur la prestation réalisée.Ce module permet d'assurer une traçabilité des échanges et d'améliorer la qualité des visites.

d) Workflow

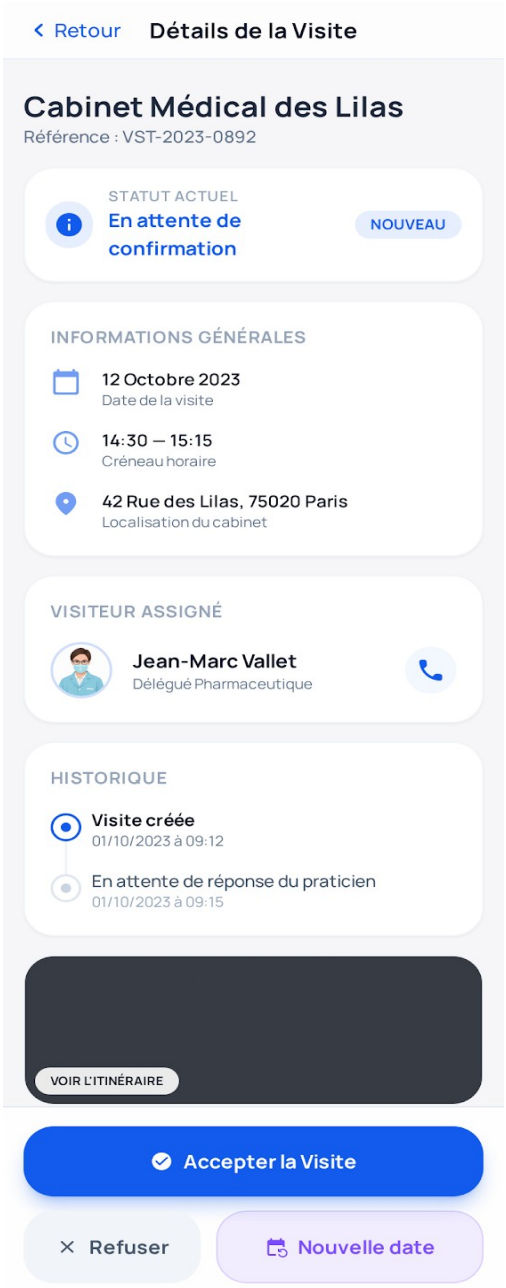

Responsable
Création d'une demande de visite
Client
acceptation / refus / proposition de nouvelle date
Responsable
Validation de la visite
Visite validée
Responsable
Affectation du visiteur médical
Visiteur médical
Réalisation de la visite
Clôture de la visite avec création d'un feedback
Client
Modification des informations de son entreprise
Client / Responsable / Visiteur médical
accès a leur planning respectif

e) Maquette

Login :



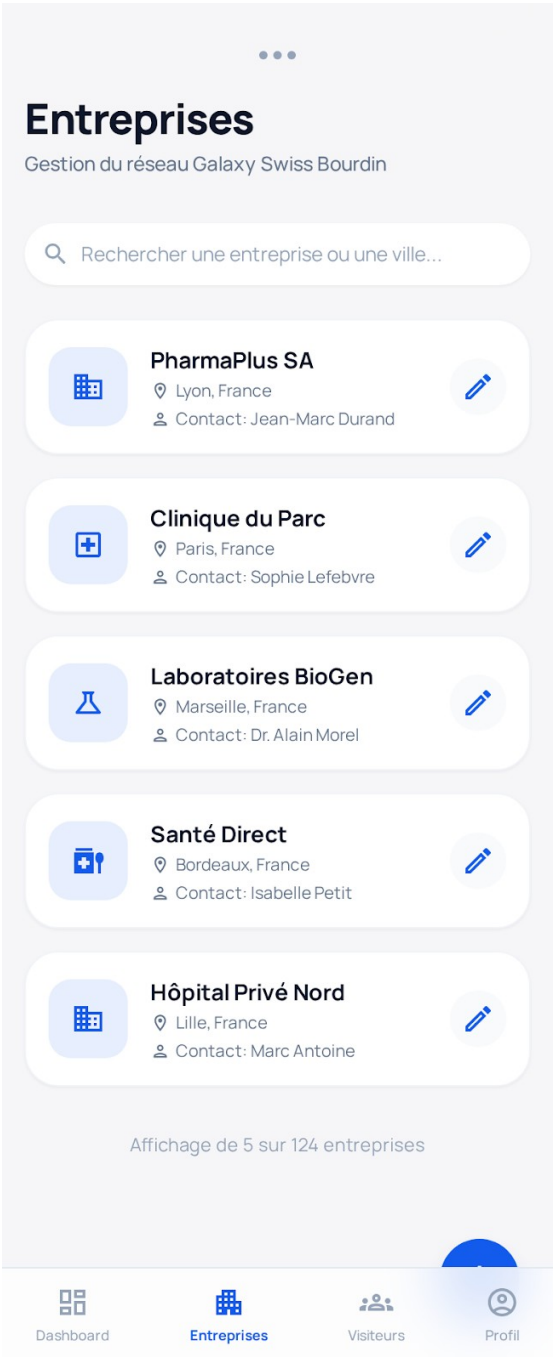
Liste de visites :



Feedback :


Retour
Compte-rendu de visite
Jean-Luc Dupont
VISITEUR MÉDICAL #GSB-8821
12 Octobre 2023 à 14:30
RAPPORT DE VISITE
Présentation de la nouvelle gamme G-Prozac Plus. Le praticien s'est montré intéressé par les études cliniques récentes. Échanges sur les effets secondaires potentiels chez les patients seniors. Un échantillon a été laissé pour test. Prochain rendez-vous prévu pour le suivi mensuel.
VOTRE AVIS
NOTEZ CETTE CONSULTATION
4.0 / 5.0
COMMENTAIRES ADDITIONNELS
Qu'avez-vous pensé de cette visite ?
Envoyer l'avis

Entreprises:



f) Technologies

Le choix des technologies vise à proposer une application mobile simple à utiliser, fiable et adaptée aux besoins du laboratoire Galaxy Swiss Bourdin (GSB) pour la gestion des visites médicales.

Architecture de l'application

L'application repose sur une architecture séparant le front-end mobile et le back-end API. Le backend Node.js est structuré selon une architecture MVC (Modèle – Vue – Contrôleur).

Cette organisation permet de séparer la logique métier, la gestion des routes et l'accès aux données, facilitant ainsi la maintenance et l'évolution de l'application.

Le frontend Flutter suit une structure classique organisée par écrans, widgets et services, permettant une bonne organisation du code et une séparation claire entre l'interface utilisateur et les appels API.

Interface utilisateur – Flutter

L'interface graphique est développée avec Flutter en utilisant les composants Material Design. Cela permet de créer une application mobile moderne, responsive et cohérente sur les appareils Android.

Langage backend – Node.js

Le backend est développé en Node.js avec le framework Express.
Il permet de gérer la logique métier, l'authentification des utilisateurs, la gestion des rôles ainsi que les échanges sécurisés avec la base de données via une API REST.

Base de données – PostgreSQL

Les données de l'application sont stockées dans une base PostgreSQL.
Elle centralise les informations relatives aux utilisateurs, aux entreprises clientes, aux visites médicales et aux feedbacks, tout en assurant la cohérence et l'intégrité des données.

g) Budget

Coûts de développement

Le développement de l'application a été réalisé par un développeur indépendant sur une durée totale de 4 mois.

Pour simuler un coût réaliste correspondant à une prestation professionnelle, on estime un tarif journalier de 150 € par jour (20 jours par mois).

Estimation :

- 4 mois × 20 jours/mois × 150 €/jour = 12 000 €

Ce montant couvre l'ensemble des phases du projet :

- Analyse des besoins
- Conception de l'application mobile
- Développement Flutter (front-end)
- Développement de l'API Node.js (back-end)
- Tests fonctionnels et de sécurité
- Déploiement et publication sur les stores

Les technologies utilisées (Flutter, Node.js, PostgreSQL, Express, Git, GitHub) sont gratuites et open source. Le matériel utilisé correspond à l'équipement personnel du développeur, sans coût supplémentaire.

Coûts d'infrastructure

L'application nécessite :

- Un serveur pour héberger l'API Node.js et la base de données
- Un nom de domaine
- Un certificat SSL

Estimation hébergement serveur : 10 € / mois, soit 120 € / an

Publication sur les stores :

- Google Play Store : frais unique de 25 €
- Apple App Store : 99 € / an (abonnement développeur)

Coûts de maintenance et de tests

La maintenance de l'application (corrections, mises à jour évolutives et support) est assurée par le développeur.

Les tests fonctionnels ont été réalisés manuellement sur émulateur et appareils physiques.

Les tests API ont été effectués à l'aide d'outils gratuits (Postman).

Coût estimé : 0 € supplémentaire (hors temps de travail inclus dans le développement).

Coûts de gestion de projet

La gestion du projet (planification, suivi des tâches, versioning et documentation) a été réalisée à l'aide d'outils gratuits tels que :

- Git et GitHub
- Notion

Ces outils n'ont généré aucun coût supplémentaire.

Estimation globale du budget

Poste	Coût (€)
Développement	12 000
Hébergement serveur (1 an)	120
Apple Developer (1 an)	99
Google Play (paiement unique)	25
Maintenance et tests	0
Gestion de projet	0
Total estimé	12 244 €

h) Planning

Phase 1 : Analyse et conception (2 semaines – Décembre)

Cette phase a permis de définir le périmètre du projet et d'identifier les besoins fonctionnels liés à la gestion des visites médicales.

Elle comprend :

- Rédaction du cahier des charges (objectifs, contraintes techniques, rôles)
- Analyse du processus de gestion des visites (demande, validation, affectation, feedback)
- Identification des différents profils utilisateurs (Responsable, Visiteur médical, Client)
- Réalisation de premières maquettes mobiles pour valider les flux (création visite, validation, clôture)
- Documentation des fonctionnalités prévues

Phase 2 : Conception technique (2 semaines – Décembre)

Cette phase a consisté à préparer la structure technique du projet avant le développement :

- Conception du schéma de la base de données PostgreSQL
- Définition des statuts de visite et des relations entre les tables
- Initialisation du projet Flutter et Node.js
- Mise en place de l'architecture MVC côté backend
- Configuration de l'environnement de développement (Git, organisation du projet)
- Tests initiaux de connexion entre Flutter et l'API

Phase 3 : Développement (5 semaines – Janvier)

Phase principale consacrée à l'implémentation des fonctionnalités :

- Développement de la gestion des visites (création, modification, statuts)
- Implémentation de la gestion des entreprises clientes
- Mise en place de l'authentification JWT et gestion des rôles
- Développement du module de feedback
- Création des tableaux de bord selon les profils
- Conception de l'interface mobile avec Flutter (Material Design)

Phase 4 : Tests et ajustements (2 semaines – Février)

Les tests sont réalisés tout au long du développement et renforcés en fin de projet :

- Tests fonctionnels de chaque module
- Vérification des rôles et permissions
- Tests des flux utilisateurs (validation, affectation, clôture)
- Correction des anomalies
- Optimisation de l'interface et amélioration de l'ergonomie

Phase 5 : Déploiement et documentation (1 semaine – Février)

Phase finale pour préparer la mise en production :

- Déploiement de l'API sur le serveur
- Configuration de la base de données en production
- Génération des versions mobiles
- Préparation à la publication sur Play Store et App Store
- Rédaction de la documentation utilisateur

Durée totale estimée du projet : 12 semaines (3 mois)

Cette planification permet un développement structuré, avec une répartition équilibrée entre conception, développement, tests et mise en production.

III. Rôles utilisateurs

L’application GSB Visit distingue trois types d’utilisateurs, chacun disposant de droits et d’accès spécifiques adaptés à ses responsabilités.

a) Visiteurs médicaux

Les visiteurs médicaux représentent les commerciaux chargés de réaliser les visites auprès des clients.

Ils disposent des fonctionnalités suivantes :

• Consultation de leur planning de visites (vue calendrier)
• Accès aux détails des visites qui leur sont affectées
• Rédaction d’un feedback à l’issue de la visite

Les visiteurs médicaux ont accès aux modules suivants : Planning, Visites et Feedback.
Limitations : un visiteur médical ne peut pas créer ou affecter une visite et ne peut consulter que ses propres visites.

b) Clients (Médecins / Entreprises)

Les clients représentent les professionnels de santé recevant les visites médicales.

Ils disposent des fonctionnalités suivantes :

• Consultation des informations de leur propre entreprise
• Consultation de leur planning de visites (vue calendrier)
• Acceptation, refus ou proposition de nouvelle date pour une visite
• Consultation de l’historique de leurs visites
• Consultation du feedback une fois la visite clôturée
• Possibilité de donner leur avis après la réalisation de la visite

Les clients ont accès aux modules suivants : leur entreprise, leurs visites et leur planning.

Limitations : un client ne peut pas accéder aux informations d’autres entreprises ni donner un avis avant la réalisation de la visite.

c) Responsables

Les responsables supervisent l'organisation des visites médicales.

Ils disposent des fonctionnalités suivantes :

• Création des demandes de visite
• Affectation des visiteurs médicaux aux visites
• Consultation de l'ensemble des visites
• Visualisation des plannings de tous les utilisateurs

Les responsables ont accès aux modules suivants : Visites, Entreprises, Planning global et Tableau de bord.

Limitations : le responsable ne rédige pas de feedback et n'intervient pas dans la validation du contenu des comptes rendus.

IV. Documentation Technique

a) Outils

- Notion : organisation du projet, suivi des tâches et documentation
- GitHub : gestion du versioning et sauvegarde du code
- Draw.io : modélisation de la base de données et des schémas
- Cursor IDE : développement et édition du code
- Postman : tests des requêtes et des échanges backend
- Docker : gestion des conteneurs pour l'environnement de développement
- FileZilla : transfert des fichiers vers le serveur de production

b) Environnement technique

Environnement de développement

- Serveur local : Docker
- Langage backend : Node.js
- Base de données : PostgresSQL
- Langages frontend : Flutter, dart

Environnement de production

- Hébergement : Vercel
- Stores : AppStore, PlayStore

Gestion des dépendances et configuration

- pubspec.yaml : gestion des dépendances Flutter / Dart
- flutter_dotenv : gestion des variables d'environnement via un fichier .env
- GoRouter : gestion de la navigation et du routage dans l'application

- l10n (Flutter Localization) : gestion de l'internationalisation et des traductions
- url_launcher : ouverture d'URLs externes (sites web, email, téléphone)

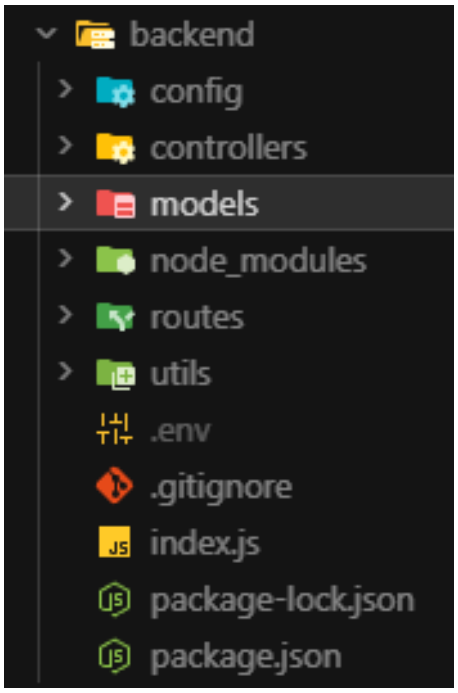
c) Architecture de l'application

Backend (Node.js / Express)

Routes
- Définition des routes HTTP de l'API (auth, companies, visits)
- Délégation des requêtes aux controllers

Controllers

- Réception et traitement des requêtes HTTP
- Appel des méthodes des models
- Renvoi des réponses JSON au client



Models

- Requêtes SQL et accès à la base de données via le client pg (Pool)
- Regroupement des opérations CRUD et de la logique métier liée aux données

Config

- db.js : création du pool de connexion PostgreSQL, paramètres issus du fichier .env

Utils

- Fonctions utilitaires réutilisables (ex. validation, regex)

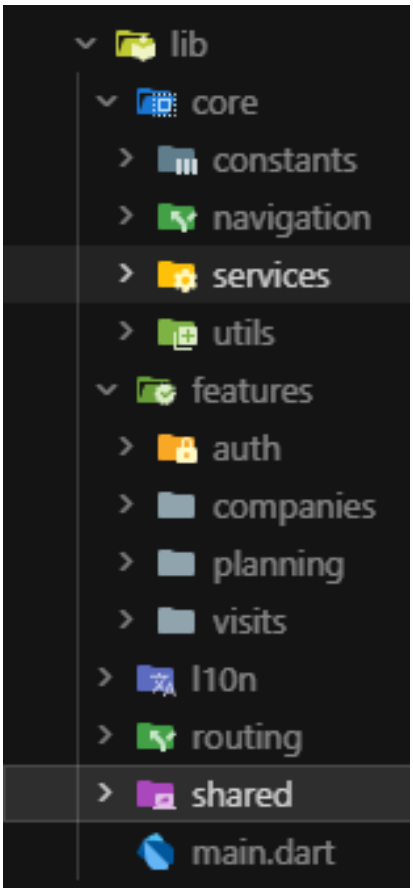
Frontend (Flutter)

Routing

- Définition des routes de l'application (GoRouter)
- Gestion de la navigation et des paramètres d'URL

Features

- Découpage par domaine fonctionnel (auth, companies, visits, planning)
- domain (modèles métier),
- data (services, accès données),
- presentation (écrans et widgets)



Core

- Constants : couleurs, espacements, styles, constantes applicatives (ex. URL API)
- Services : appel API (ApiService), préférences utilisateur (UserPreferences)
- Utils : fonctions utilitaires (dates, adresses, etc.)
- Navigation : observateur de routes

Shared

- Composants d'interface réutilisables (barre d'application, barre de navigation, notifications)

L10n

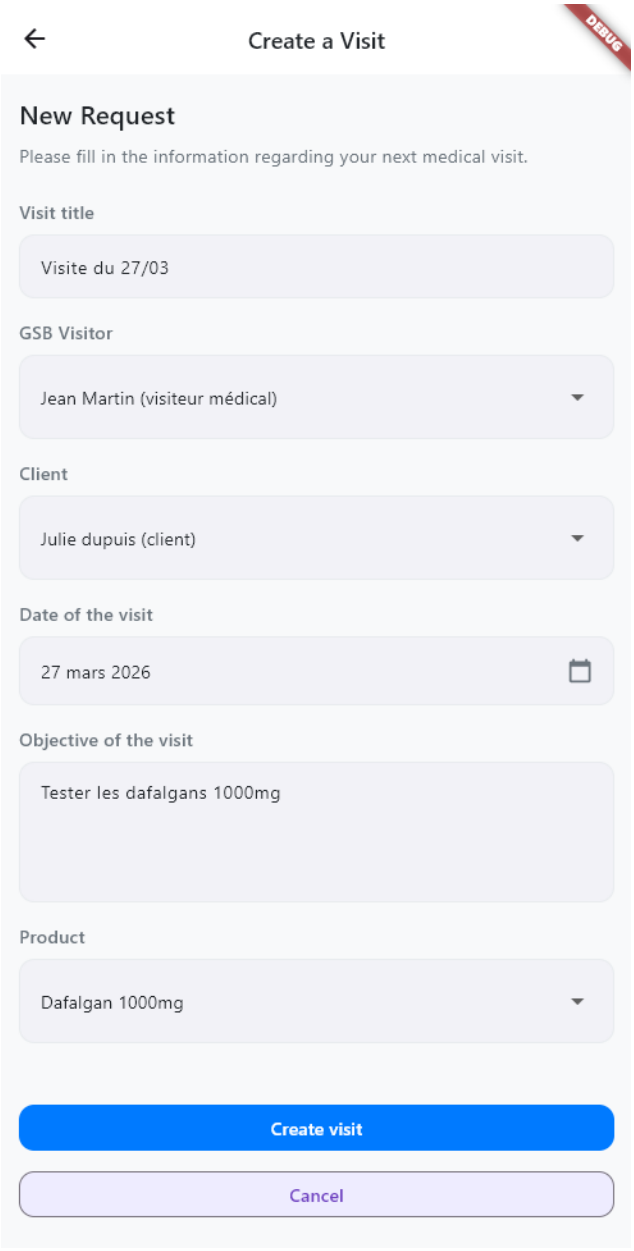
- Fichiers de traduction (.arb) et classes de localisation générées pour l'internationalisation

Synthèse

- Backend : Routes → Controllers → Models → PostgreSQL (config dans config/db.js).
- Frontend : Routing → Features (domain / data / presentation) + Core (constantes, services, utils) + Shared (composants) + L10n (traductions).

d) Exemple de fonctionnement : création d'une visite

L'utilisateur remplit le formulaire (titre, visiteur, employé client, produit, date prévue, commentaire) et valide.



Le fichier api_service est appelé avec les données du formulaire.

```dart
final result = await ApiService.createVisit(
  visitTitle: title,
  fkClientId: _selectedEmployeeId!,
  fkGsbVisitorId: _selectedVisitorId!,
  fkStatusId: _statusToValidate,
  fkProductId: _selectedProductId,
  scheduledDate: scheduledDateStr,
  comment: comment.isEmpty ? null : comment,
);
```

Une requête API est envoyée à la route POST /visits/createVisit.

```dart
static Future<VisitsResult> createVisit({
  required String visitTitle,
  required int fkClientId,
  required int fkGsbVisitorId,
  required int fkStatusId,
  int? fkProductId,
  String? scheduledDate,
  String? closureDate,
  String? comment,
}) async {
  final uri = Uri.parse('${AppConstants.apiBaseUrl}/visits/createVisit');
  final body = <String, dynamic>{
    'visit_title': visitTitle,
    'fk_client_id': fkClientId,
    'fk_gsb_visitor_id': fkGsbVisitorId,
    'fk_status_id': fkStatusId,
  };
  if (fkProductId != null) body['fk_product_id'] = fkProductId;
  if (scheduledDate != null) body['scheduled_date'] = scheduledDate;
  if (closureDate != null) body['closure_date'] = closureDate;
  if (comment != null && comment.isNotEmpty) body['comment'] = comment;
  final response = await http.post(
    uri,
    headers: {'Content-Type': 'application/json'},
    body: jsonEncode(body),
  );
  Map<String, dynamic>? decoded;
  final responseBody = response.body;
  if (responseBody.isNotEmpty) {
    try {
      decoded = jsonDecode(responseBody) as Map<String, dynamic>?;
    } catch (exception) {
      decoded = null;
    }
  }
  return VisitsResult(statusCode: response.statusCode, body: decoded);
}
```

Le serveur Node effectue un premier routage et redirige vers visitsRoutes.

```
"use strict";

const express = require("express");
const authRoutes = require("./routes/authRoutes");
const companiesRoutes = require("./routes/companiesRoutes");
const visitsRoutes = require("./routes/visitsRoutes");
const app = express();
const port = Number(process.env.SRV_PORT) || 3000;

app.use(express.json());
app.use("/", authRoutes);
app.use("/companies", companiesRoutes);
app.use("/visits", visitsRoutes);

app.get("/", (req, res) => {
  res.json({ ok: true });
});

app.listen(port, () => {
  //
});
```

Le routeur Express envoie la requête au visitsController et à la fonction createVisitHandler.

```
"use strict";

const express = require("express");
const {
  getVisitsByUserIdHandler,
  getVisitDatasByIdHandler,
  getNewVisitDatasHandler,
  createVisitHandler,
  submitFeedbackHandler,
  acceptVisitHandler,
  updateVisitScheduledDateHandler,
  refuseVisitHandler,
  deleteVisitHandler,
} = require("../controllers/visitsController");

const router = express.Router();
router.get("/getVisitsByUserId", getVisitsByUserIdHandler);
router.get("/getVisitDatasById", getVisitDatasByIdHandler);
router.get("/getNewVisitDatas", getNewVisitDatasHandler);
router.post("/createVisit", createVisitHandler);
router.post("/submitFeedback", submitFeedbackHandler);
router.post("/acceptVisit", acceptVisitHandler);
router.post("/updateVisitScheduledDate", updateVisitScheduledDateHandler);
router.post("/refuseVisit", refuseVisitHandler);
router.post("/deleteVisit", deleteVisitHandler);

module.exports = router;
```

La fonction createVisitHandler vérifie la méthode (POST), lit le corps de la requête et valide les champs obligatoires (visit_title, fk_client_id, fk_gsb_visitor_id, fk_status_id) et optionnels (fk_product_id, scheduled_date, closure_date, comment).

```
let clientExists;
let visitorExists;
let statusExists;
let productExists = true;

try {
  const clientUser = await findUserById(fkClientId);
  clientExists = !!clientUser;
  const visitorUser = await findUserById(fkGsbVisitorId);
  visitorExists = !!visitorUser;
  statusExists = await checkStatusExists(fkStatusId);
  if (fkProductId != null) {
    productExists = await checkProductExists(fkProductId);
  }
} catch (err) {
  res.status(500).json({ message: "Database error", data: null });
  return;
}
```

```
if (!visitorExists) {
  res.status(400).json({
    message: "fk_gsb_visitor_id does not exist in users",
    data: null,
  });
  return;
}
if (!statusExists) {
  res.status(400).json({
    message: "fk_status_id does not exist in status",
    data: null,
  });
  return;
}
if (fkProductId != null && !productExists) {
  res.status(400).json({
    message: "fk_product_id does not exist in stock",
    data: null,
  });
  return;
}

let created;
try {
  created = await createVisit({
    visit_title: visitTitle.trim(),
    fk_client_id: fkClientId,
    fk_gsb_visitor_id: fkGsbVisitorId,
    fk_status_id: fkStatusId,
    fk_product_id: fkProductId,
    scheduled_date: scheduledDate,
    closure_date: closureDate,
    comment,
  });
```

Le contrôleur vérifie en base que le client et le visiteur existent, que le statut existe, et éventuellement que le produit existe dans la table stock.

En cas d'erreur de validation ou de donnée inexistante, une réponse 400 est renvoyée avec un message ; en cas d'erreur base de données, une réponse 500.

Si les validations sont correctes, le contrôleur appelle la fonction createVisit du model visit.

```
async function createVisit(payload) {
  const {
    visit_title,
    fk_client_id,
    fk_gsb_visitor_id,
    fk_status_id,
    fk_product_id,
    scheduled_date,
    closure_date,
    comment,
  } = payload;

  const result = await pool.query(
    `INSERT INTO visits (
      visit_title, fk_client_id, fk_gsb_visitor_id, fk_status_id, fk_product_id,
      scheduled_date, closure_date, comment
    )
    VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
    RETURNING visit_id`,
    [
      visit_title,
      fk_client_id,
      fk_gsb_visitor_id,
      fk_status_id,
      fk_product_id ?? null,
      scheduled_date ?? null,
      closure_date ?? null,
      comment ?? null,
    ]
  );

  const row = result.rows?.[0];
  return row ? { visit_id: row.visit_id } : null;
}
```

Le model exécute une requête SQL INSERT INTO visits (...) avec les paramètres fournis, puis RETURNING visit_id.

Le contrôleur reçoit l'objet renvoyé (contenant visit_id). En cas d'échec ou d'absence de visit_id, une réponse 500 est envoyée.

```
let created;
try {
  created = await createVisit({
    visit_title: visitTitle.trim(),
    fk_client_id: fkClientId,
    fk_gsb_visitor_id: fkGsbVisitorId,
    fk_status_id: fkStatusId,
    fk_product_id: fkProductId,
    scheduled_date: scheduledDate,
    closure_date: closureDate,
    comment,
  });
} catch (err) {
  res.status(500).json({ message: "Database error", data: null });
  return;
}

if (!created || created.visit_id == null) {
  res.status(500).json({ message: "Database error", data: null });
  return;
}

res.status(201).json({ message: "OK", data: { visit_id: created.visit_id } });
```

En cas de succès, le contrôleur renvoie une réponse 201 avec le corps { message: "OK", data: { visit_id: ... } }.

Côté Flutter, ApiService.createVisit reçoit cette réponse. Si le code HTTP est 201, l'écran redirige vers la liste des visites (/visits) avec un indicateur de succès ; sinon un message d'erreur est affiché (SnackBar).

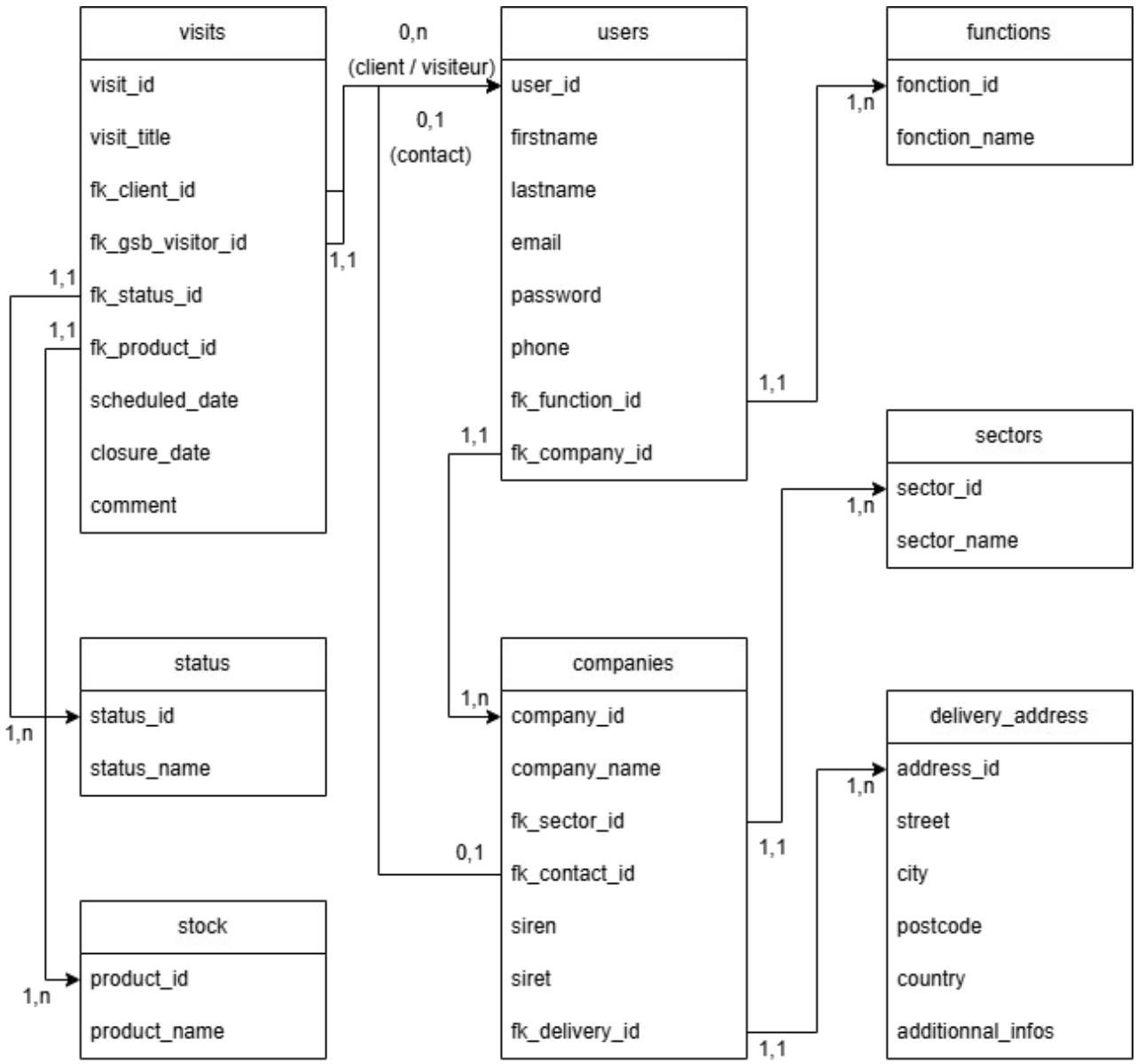
```
if (result.statusCode == 201) {
  WidgetsBinding.instance.addPostFrameCallback((_) {
    if (context.mounted) context.go('/visits', extra: true);
  });
} else {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text(
        result.body?['message'] as String? ?? l10n.newVisitCreateError,
      ), // Text
    ), // SnackBar
  );
}
```

Schéma global du flux

Action utilisateur (validation du formulaire)
→ Requête HTTP POST /visits/createVisit
→ Route Express /visits
→ visitsController.createVisitHandler
→ Validation des données et vérification des clés étrangères (models)
→ Model visit.createVisit (INSERT SQL)
→ Réponse 201 avec visit_id
→ Frontend : redirection vers la liste des visites avec message de succès

Ce cycle est repris pour les autres actions (acceptation, refus, modification de date, feedback, suppression de visite) avec leurs routes et handlers dédiés.

e) Base de données



V. Sécurité de l’application

La sécurité de l’application GSB Feedback repose sur plusieurs mécanismes mis en place dans le frontend Flutter et le backend Node.js afin de limiter les accès non autorisés, protéger les données et sécuriser l’authentification des utilisateurs.

a) Gestion des accès et des rôles

L’application utilise un système de rôles distinguant trois profils : responsable, visiteur médical et client.

Lors de la connexion, le backend renvoie les informations de l’utilisateur, notamment son identifiant, son entreprise de rattachement et son rôle. Ces informations sont utilisées par l’application pour adapter l’interface et afficher uniquement les fonctionnalités autorisées selon le profil connecté.

Par exemple, certaines actions comme la création d’une visite ou la validation d’une demande ne sont accessibles qu’aux rôles concernés.

Côté backend, des contrôles sont également présents sur certaines routes afin de limiter l’accès aux données, notamment pour empêcher un client d’accéder aux informations d’une autre entreprise. Cependant, la vérification des droits n’est pas encore centralisée dans un middleware global de l’API.

b) Sécurité des sessions

L’application ne repose pas encore sur un système de session serveur ou sur des jetons d’authentification de type JWT. Après l’authentification, les informations utiles à la navigation (identifiant utilisateur, identifiant d’entreprise et rôle) sont stockées localement dans l’application afin de conserver l’état de connexion et adapter l’interface entre les écrans.

Ce fonctionnement facilite la navigation mais ne constitue pas encore une validation forte de session côté serveur, puisque les requêtes suivantes ne sont pas associées à un mécanisme d’authentification dédié.

c) Protection des données

Des contrôles permettent de limiter l'accès aux données selon le profil utilisateur. Par exemple, un client ne peut consulter que les informations liées à sa propre entreprise.

Les écrans et listes affichés dans l'application sont également filtrés en fonction de l'utilisateur connecté et de son rôle, ce qui restreint l'accès aux visites, aux entreprises et aux actions disponibles.

Cependant, cette protection reste partielle car certaines routes backend utilisent encore des identifiants transmis par le client sans vérification centralisée de l'identité authentifiée.

d) Sécurisation de l'authentification

Les mots de passe sont stockés en base de données sous forme hachée avec l'algorithme Argon2, ce qui évite toute conservation en clair.

Lors de la connexion, les identifiants sont vérifiés côté backend après validation du format des données reçues. Les accès à la base PostgreSQL utilisent des requêtes paramétrées, ce qui protège contre les injections SQL.

Les champs saisis par l'utilisateur sont également validés en termes de type et de format. De plus, l'interface Flutter n'interprète pas directement du code HTML, ce qui limite certains risques de type XSS.

e) Validation et gestion des erreurs

Les données reçues par l'API sont contrôlées côté backend avant traitement. Les identifiants numériques, les champs obligatoires et certains formats comme l'adresse email sont vérifiés afin d'éviter l'envoi de données incohérentes ou malveillantes.

Les accès à la base de données sont encadrés par des blocs try/catch, permettant de gérer les erreurs sans exposer d'informations sensibles. En cas d'échec, l'API renvoie des messages génériques comme *Database error*.

f) Tests de sécurité réalisés

Des vérifications manuelles ont été effectuées pendant le développement pour contrôler les mécanismes de sécurité :

- test de connexion avec identifiants invalides ;
- vérification des restrictions d'affichage selon les rôles ;
- contrôle de l'accès aux données d'entreprise pour un client ;
- envoi de paramètres invalides ou incomplets à l'API ;
- vérification de la gestion des erreurs côté backend.

Ces tests ont permis de valider les premiers mécanismes de sécurité intégrés dans l'application.

g) Limites identifiées

Certaines protections avancées ne sont pas encore présentes dans cette version.

L'application ne dispose pas encore d'une authentification sécurisée par session serveur ou JWT, ni d'un middleware central chargé de vérifier systématiquement l'identité et le rôle de l'utilisateur à chaque requête.

De plus, certaines routes backend reposent encore sur des informations transmises par le client, ce qui devra être renforcé pour éviter toute élévation de privilèges.

Enfin, la généralisation du HTTPS et l'ajout d'une authentification centralisée constituent des pistes d'amélioration pour les futures évolutions du projet.

VI. Bilan

a) Difficultés rencontrées et solutions apportées

Apprentissage du langage Dart et du développement mobile

La première difficulté rencontrée lors de ce projet a été l'apprentissage du langage Dart et plus généralement du développement d'applications mobiles. N'ayant pas d'expérience préalable dans ce domaine, il a fallu prendre le temps de comprendre la logique de Flutter, le fonctionnement des widgets ainsi que la structure d'une application mobile.

Solution apportée :

Pour progresser rapidement, j'ai principalement appris par la pratique. J'ai réalisé plusieurs petits projets et exercices afin de me familiariser avec le langage et l'environnement Flutter. Je me suis également appuyé sur des vidéos de formation et différentes ressources en ligne pour mieux comprendre certains concepts. Cette approche m'a permis d'acquérir progressivement les bases nécessaires au développement de l'application.

Compréhension de l'architecture d'une application en Full API

La seconde difficulté a concerné la mise en place d'une architecture d'application reposant entièrement sur une API. L'application mobile agit uniquement comme un client qui communique avec un serveur, ce qui nécessite une bonne organisation du code et une gestion claire des échanges de données.

Solution apportée :

Pour mieux comprendre cette approche, j'ai réalisé plusieurs tests en développant de petites applications et en expérimentant différentes manières d'organiser le projet. Je me suis également appuyé sur la documentation officielle de Flutter et sur des exemples d'architecture utilisés dans des projets similaires. Ces essais m'ont permis de mettre en place une structure plus claire et adaptée au projet.

Gestion et attribution des rôles utilisateurs

La troisième difficulté a concerné la gestion des rôles utilisateurs. La principale complexité n'était pas seulement technique, mais aussi liée à la réflexion sur l'utilité des différents rôles et sur les fonctionnalités associées à chacun.

Solution apportée :

Pour résoudre ce problème, je me suis appuyé sur le cahier des charges afin d'identifier les différents profils d'utilisateurs et leurs besoins. Une réflexion sur les fonctionnalités de l'application m'a permis de définir des rôles cohérents et d'adapter les accès aux différentes fonctionnalités en fonction de ces rôles.

Enseignements tirés du projet

Ce projet m'a permis de découvrir le développement d'applications mobiles et de mieux comprendre l'organisation d'une application reposant sur une API. Il m'a également appris l'importance de l'expérimentation et de la pratique pour progresser rapidement dans l'apprentissage de nouvelles technologies.

b) Acquis et compétences développées

La réalisation du PPE GSB Feedback m’a permis de développer à la fois des compétences techniques et méthodologiques, tout en améliorant ma manière d’organiser et de conduire un projet informatique.

Sur le plan de la gestion de projet, ce PPE m’a amené à suivre une démarche complète : analyse du besoin, rédaction du cahier des charges, conception des maquettes et organisation des différentes étapes du développement.

Cette phase de préparation m’a permis de mieux comprendre l’importance de structurer un projet avant de commencer le développement.

Sur le plan technique, ce projet m’a permis de découvrir le développement d’applications mobiles avec Flutter et le langage Dart.

J’ai appris à concevoir des interfaces mobiles, à gérer la navigation entre les écrans et à structurer une application mobile de manière claire et maintenable.

Le projet m’a également permis de travailler avec une architecture basée sur une API, où l’application mobile communique avec un serveur pour récupérer et envoyer des données.

Cela m’a permis de mieux comprendre la gestion des requêtes, le traitement des réponses et l’organisation du code dans une application client.

J’ai aussi développé des compétences dans la gestion des rôles utilisateurs, en réfléchissant aux différents profils présents dans l’application et aux fonctionnalités qui leur sont associées.

Ce travail m’a permis de mieux comprendre l’importance de lier les fonctionnalités aux besoins réels des utilisateurs.

Enfin, ce projet m’a permis d’améliorer ma méthode d’apprentissage, notamment en progressant par la pratique, en réalisant des tests et en expérimentant différentes solutions techniques avant de choisir celle la plus adaptée au projet.

Dans l’ensemble, ce PPE m’a permis d’acquérir une première expérience concrète dans le développement d’applications mobiles et de mieux comprendre les différentes étapes nécessaires à la réalisation d’un projet informatique complet.

c) Axes d’amélioration

Plusieurs évolutions pourraient être envisagées afin d’améliorer et d’enrichir les fonctionnalités de l’application GSB Feedback.

Gestion des utilisateurs

La mise en place d’une interface dédiée à la gestion des utilisateurs permettrait d’administrer plus facilement les comptes présents dans l’application. Cette fonctionnalité pourrait inclure la création, la modification et la suppression des comptes ainsi qu’une gestion plus complète des rôles utilisateurs.

Gestion des entreprises

Une amélioration possible serait également l’ajout d’un système de gestion des entreprises. Cela permettrait d’ajouter, modifier ou supprimer les entreprises associées aux visites, afin de mieux structurer et organiser les données utilisées dans l’application.

Mise en place d’une barre de recherche

L’ajout d’une barre de recherche permettrait de retrouver plus rapidement certaines visites ou informations spécifiques dans l’application, notamment lorsque le volume de données devient plus important.

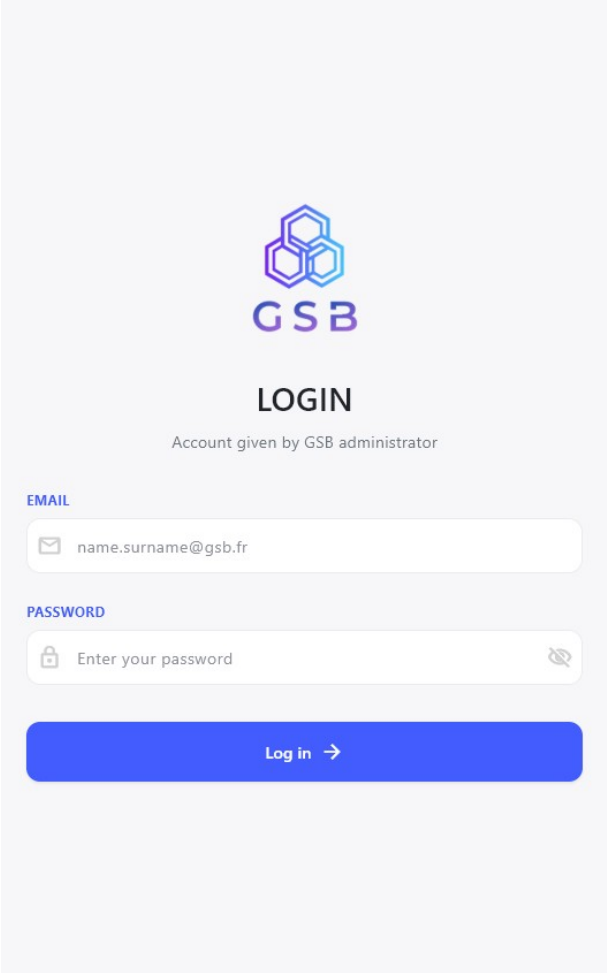
Ajout de filtres pour les visites
La mise en place de filtres (par date, entreprise ou utilisateur par exemple) permettrait de faciliter la consultation et l’analyse des visites enregistrées. Cette fonctionnalité améliorerait la navigation dans l’application et rendrait la recherche d’informations plus rapide et plus efficace.

VII. Documentation utilisateur

a) Client

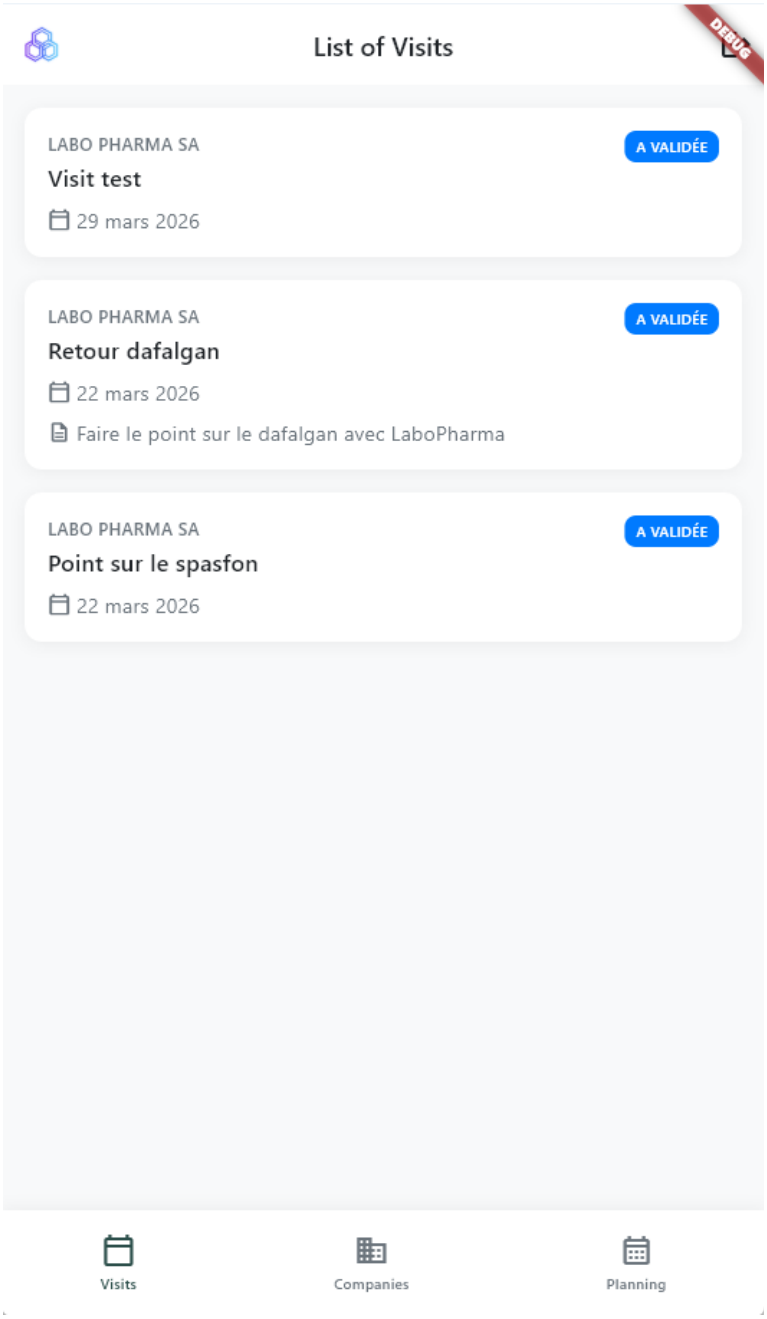
Connexion

L'utilisateur se connecte avec son adresse e-mail et son mot de passe.



Première page après connexion

Après la connexion, l'utilisateur arrive sur l'écran des visites.
Il y voit la liste des visites qui le concernent (titre, date prévue, statut, entreprise).



Un bouton Déconnexion est visible dans la barre d'application.
Un clic déconnecte l'utilisateur et termine la session.

Menu de navigation

Le menu principal permet d'accéder aux sections suivantes :

- Visites
- Entreprises
- Planning

Entreprise

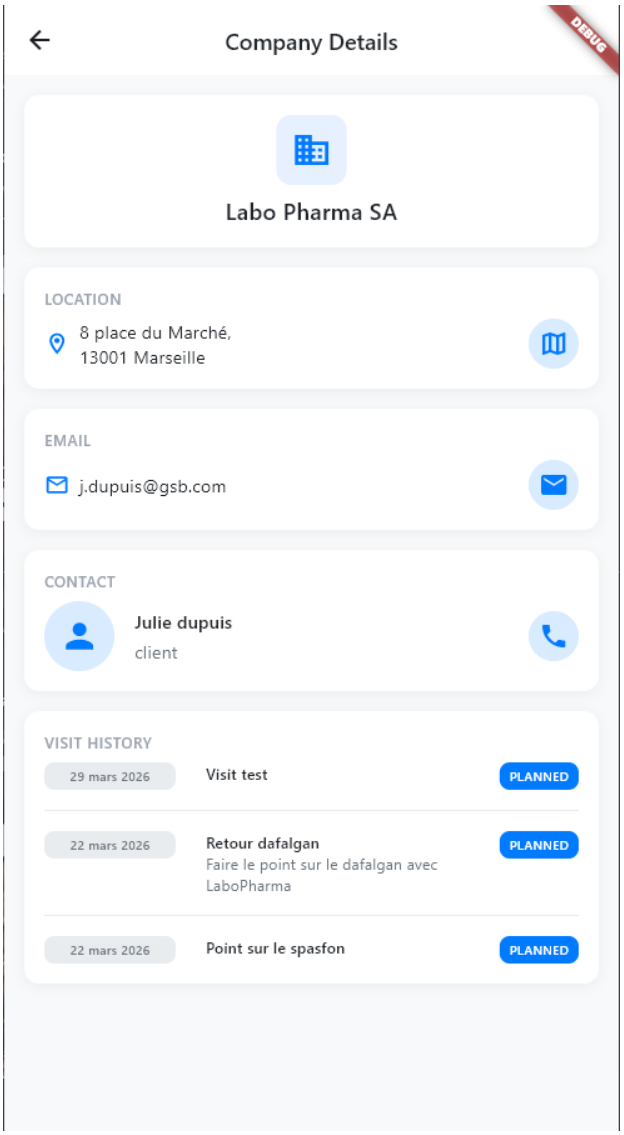
Dans la section Entreprises, l'utilisateur voit uniquement son entreprise (fiche entreprise).

Sont affichés :
- nom de l'entreprise,
- adresse,
- coordonnées du contact (client),
- historique des visites.

Un lien permet d'ouvrir l'adresse dans une application de cartographie.

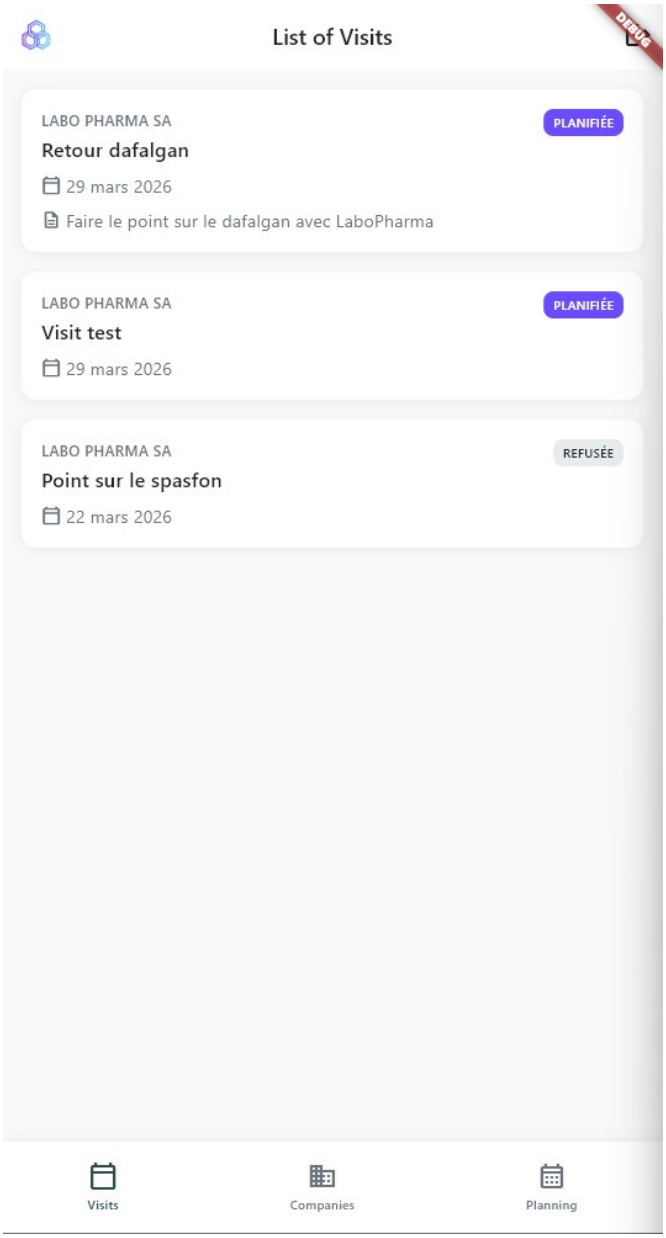
Les coordonnées (téléphone, e-mail) permettent d'appeler ou d'envoyer un e-mail.

Un bouton Retour ramène à l'écran des visites.

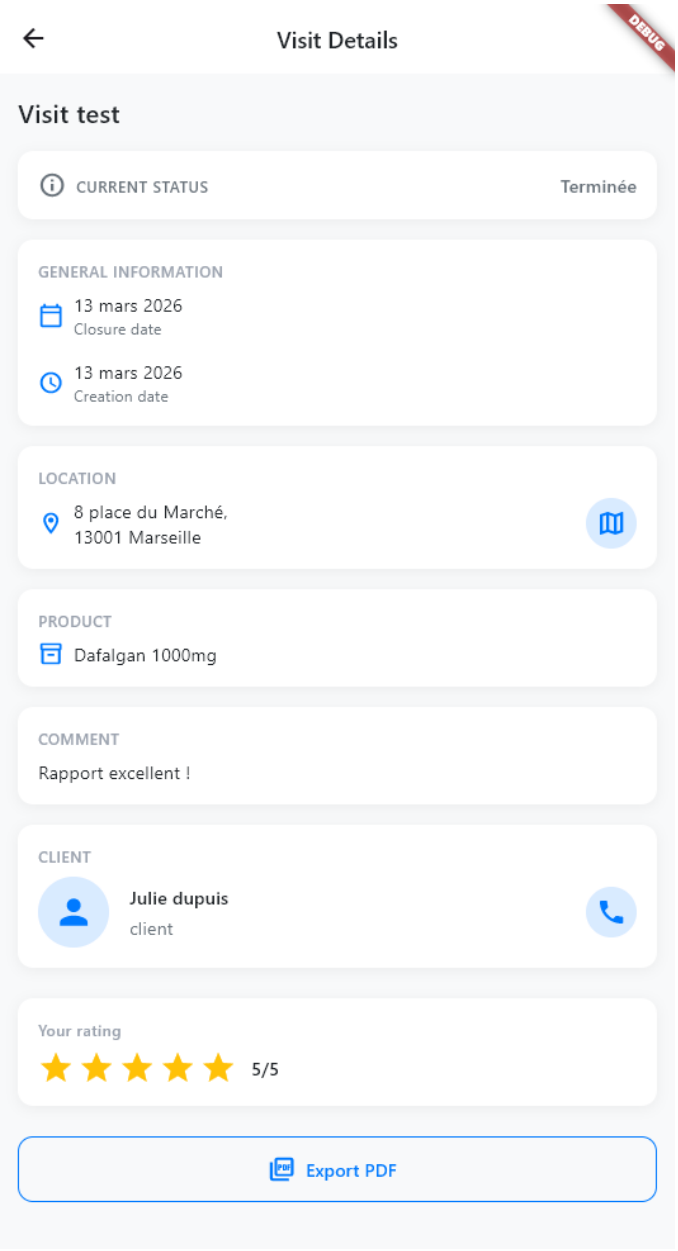


Visites

L’écran Visites affiche la liste des visites de l’utilisateur..



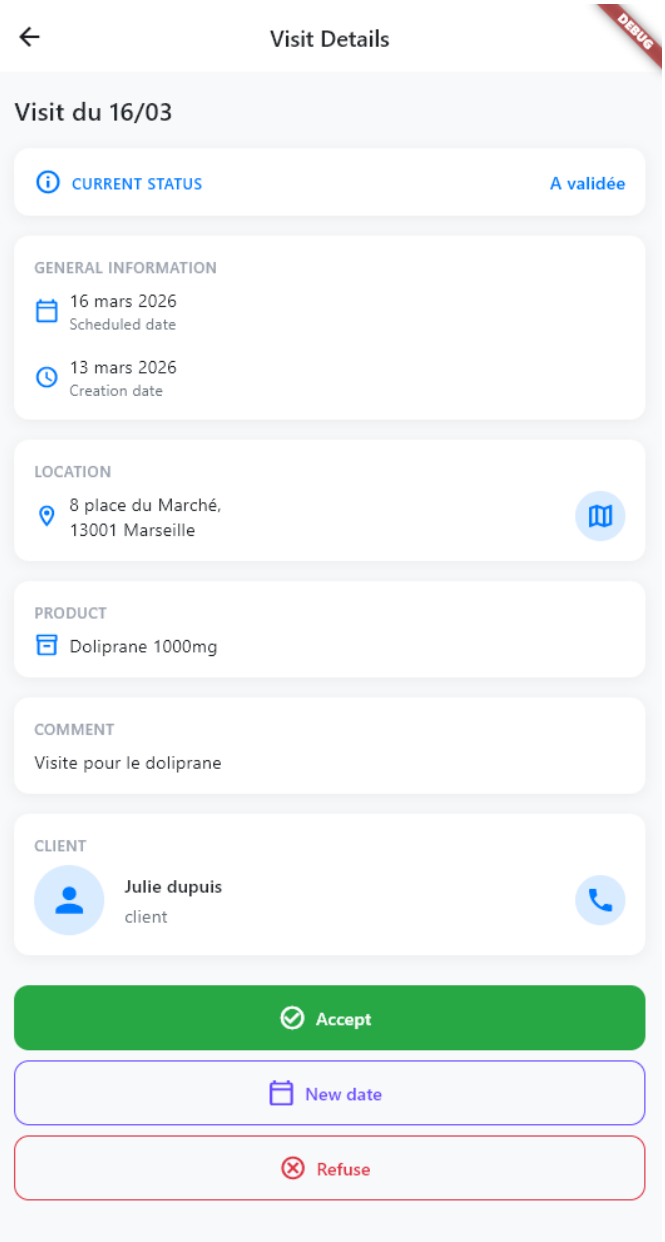
Un clic sur une visite ouvre la fiche détaillée de la visite (titre, statut, dates, lieu, produit, commentaire, contact client).

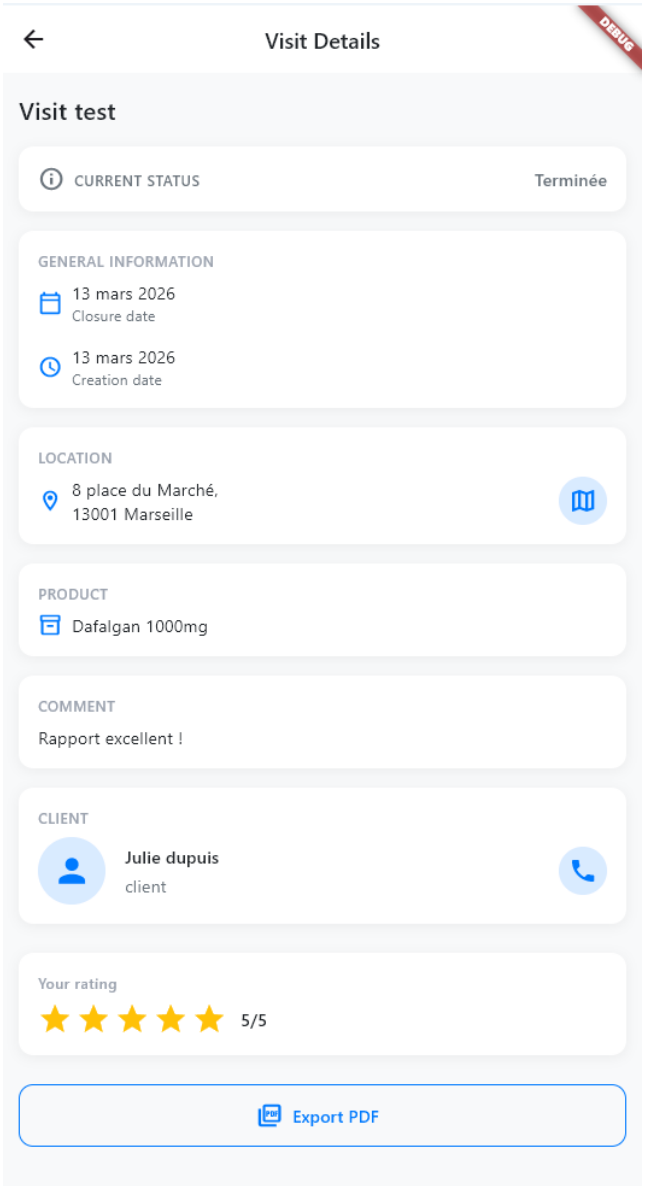


Tant que la visite est au statut « À valider » :

- l'utilisateur peut accepter la visite ;
- l'utilisateur peut proposer une nouvelle date ;
- l'utilisateur peut refuser la visite.

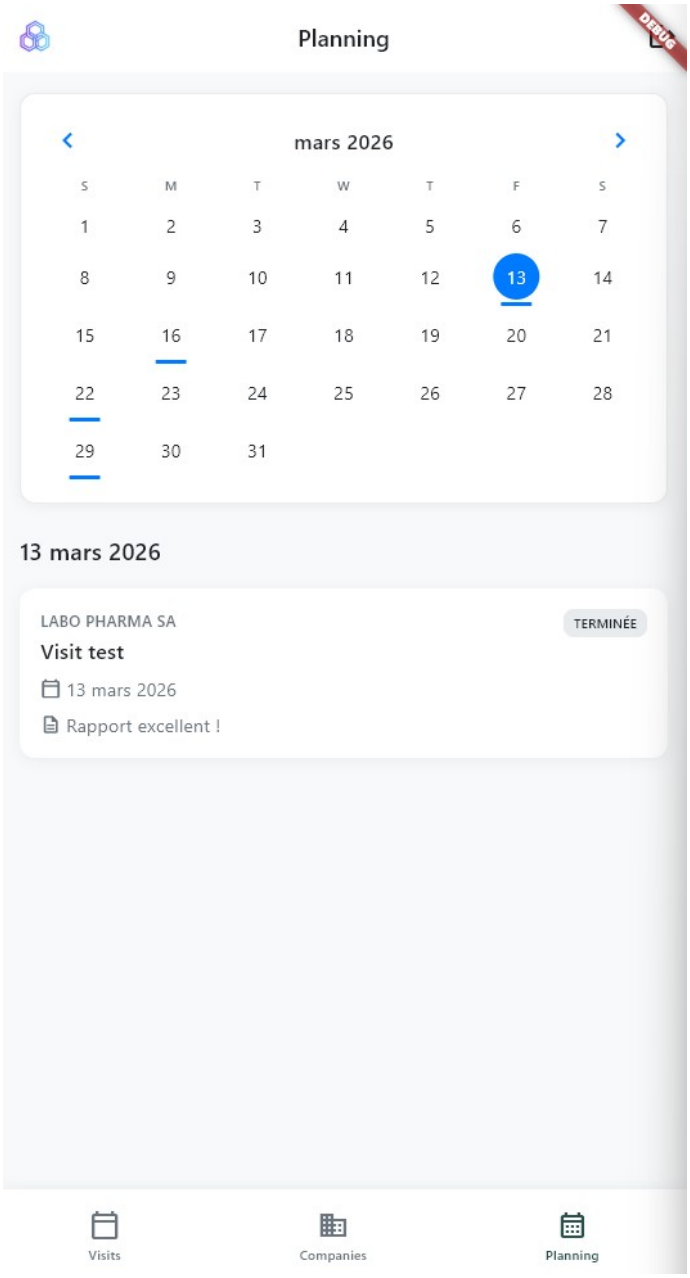
Une fois la visite effectuée (statut « Effectuée ») ou refusée, un bouton permet d'exporter la fiche en PDF.





Planning

La section Planning affiche un calendrier mensuel.
Les jours comportant au moins une visite sont repérés.

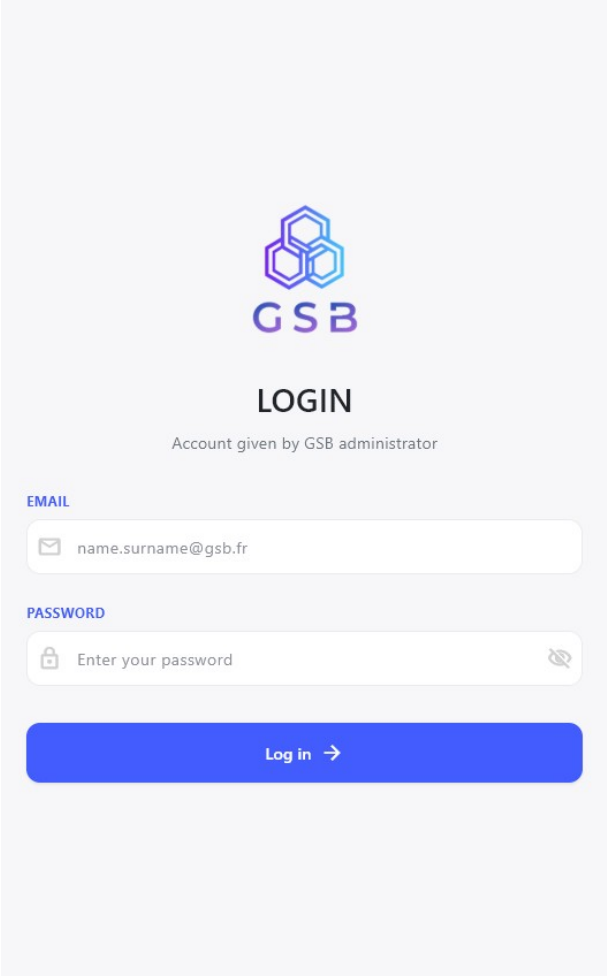


L’utilisateur sélectionne un jour pour afficher la liste des visites de ce jour.
Un clic sur une visite ouvre la fiche détail de la visite.

b) Responsable

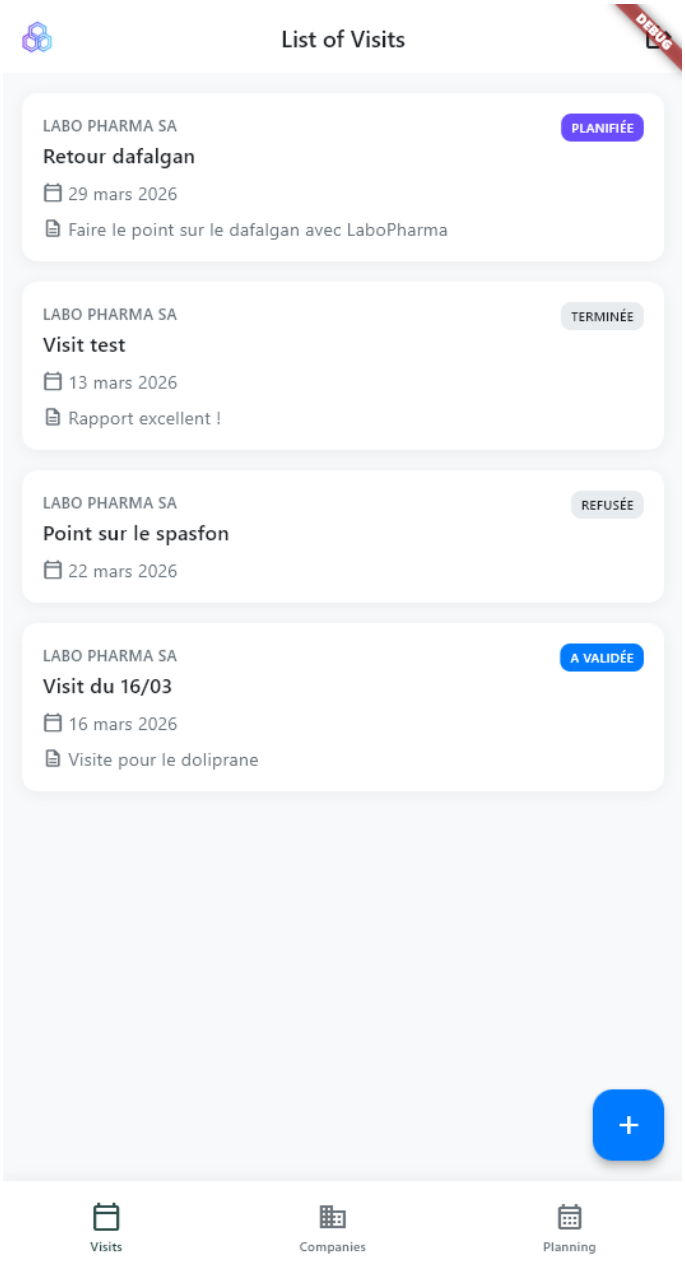
Connexion

Le responsable se connecte avec son adresse e-mail et son mot de passe.



Écran des visites

Après la connexion, le responsable arrive sur l'écran des visites.
Il y voit la liste des visites (titre, date, statut, entreprise).
Un bouton flottant « Créer une visite » est disponible.



Un bouton Déconnexion est visible.
Un clic déconnecte l'utilisateur et termine la session.

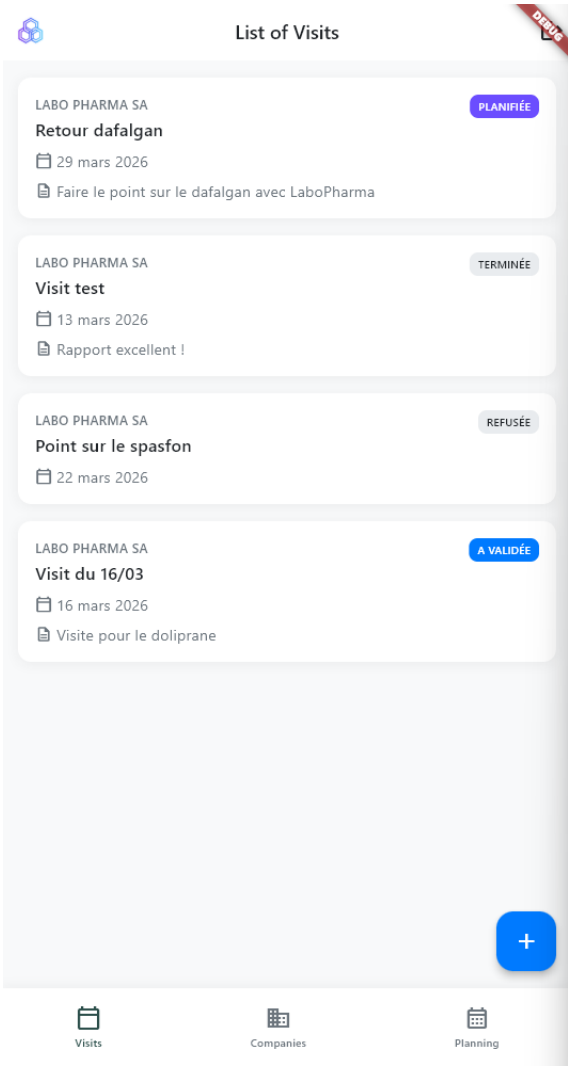
Menu de navigation

Le menu principal permet d'accéder aux sections suivantes :

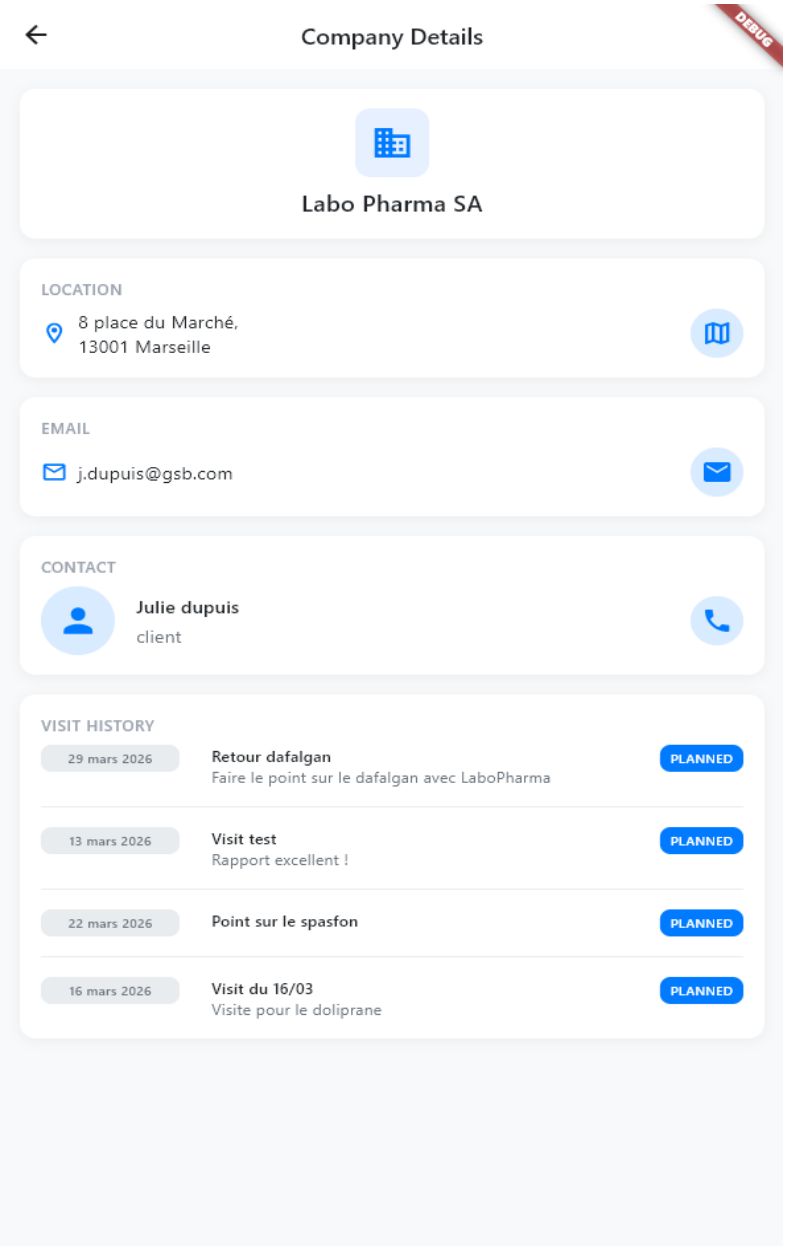
- Visites
- Entreprises
- Planning

Entreprises

La section Entreprises affiche la liste des entreprises auxquelles le responsable a accès.



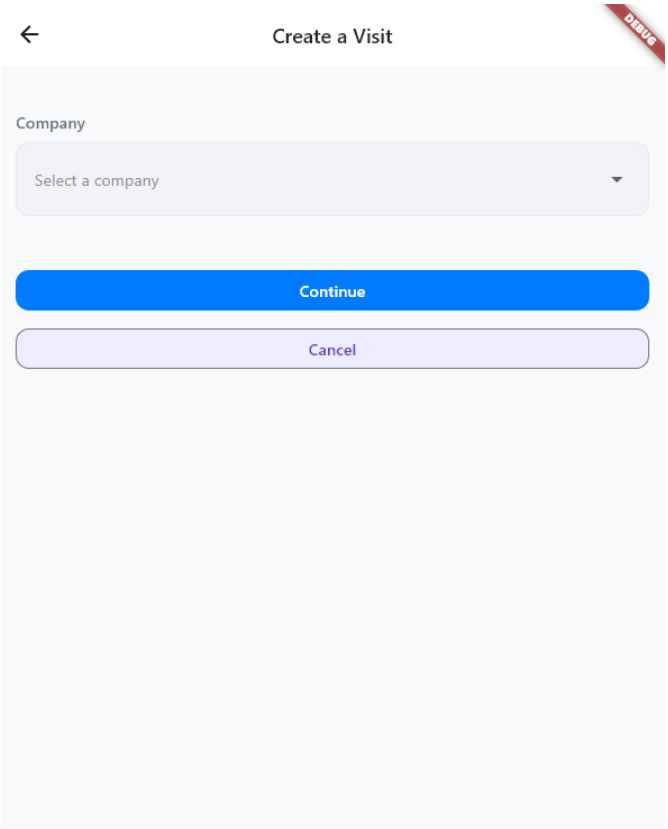
Un clic sur une ligne (ou sur une entreprise) ouvre la fiche entreprise.
La fiche affiche : nom, adresse, contact (client), historique des visites.



Création d’une visite

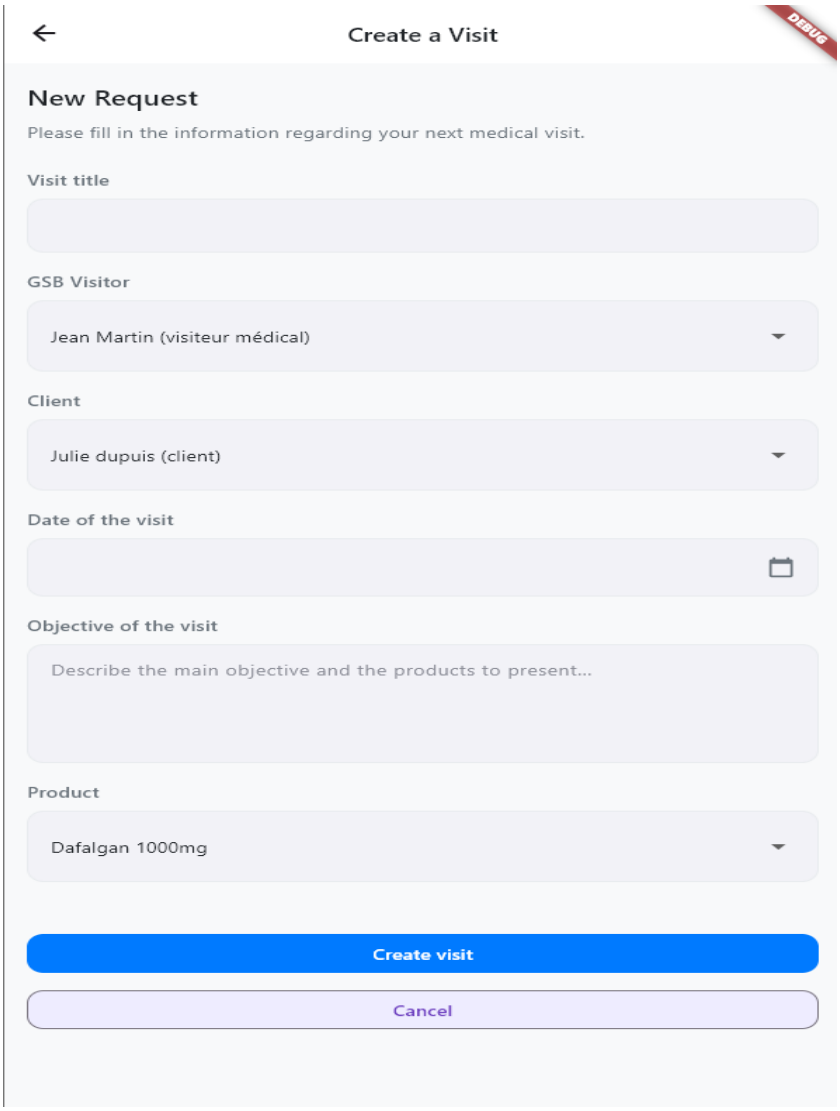
Un clic sur le bouton « Créer une visite » ouvre l’écran de choix de l’entreprise.

L’utilisateur sélectionne une entreprise puis poursuit.



Un formulaire permet de saisir :

- titre de la visite,
- date prévue,
- commentaire,
- visiteur médical,
- client (contact côté entreprise),
- produit.

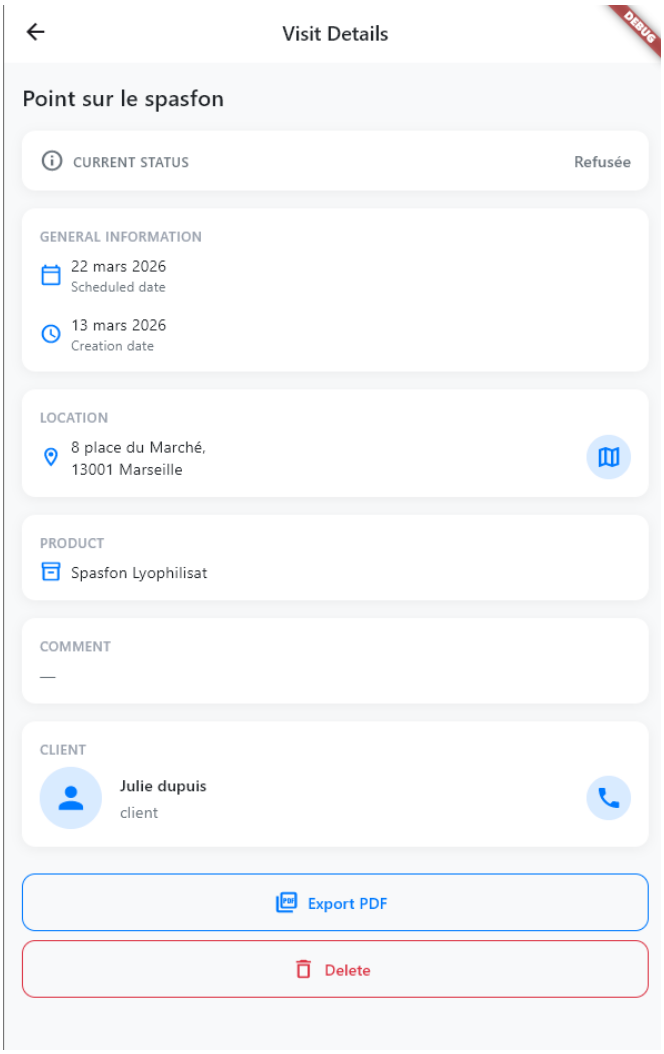


Un clic sur le bouton d'enregistrement crée la visite.
Un message de confirmation s'affiche et l'utilisateur est ramené à la liste des visites.

Visites – Fiche et actions

Un clic sur une visite ouvre la fiche détaillée :

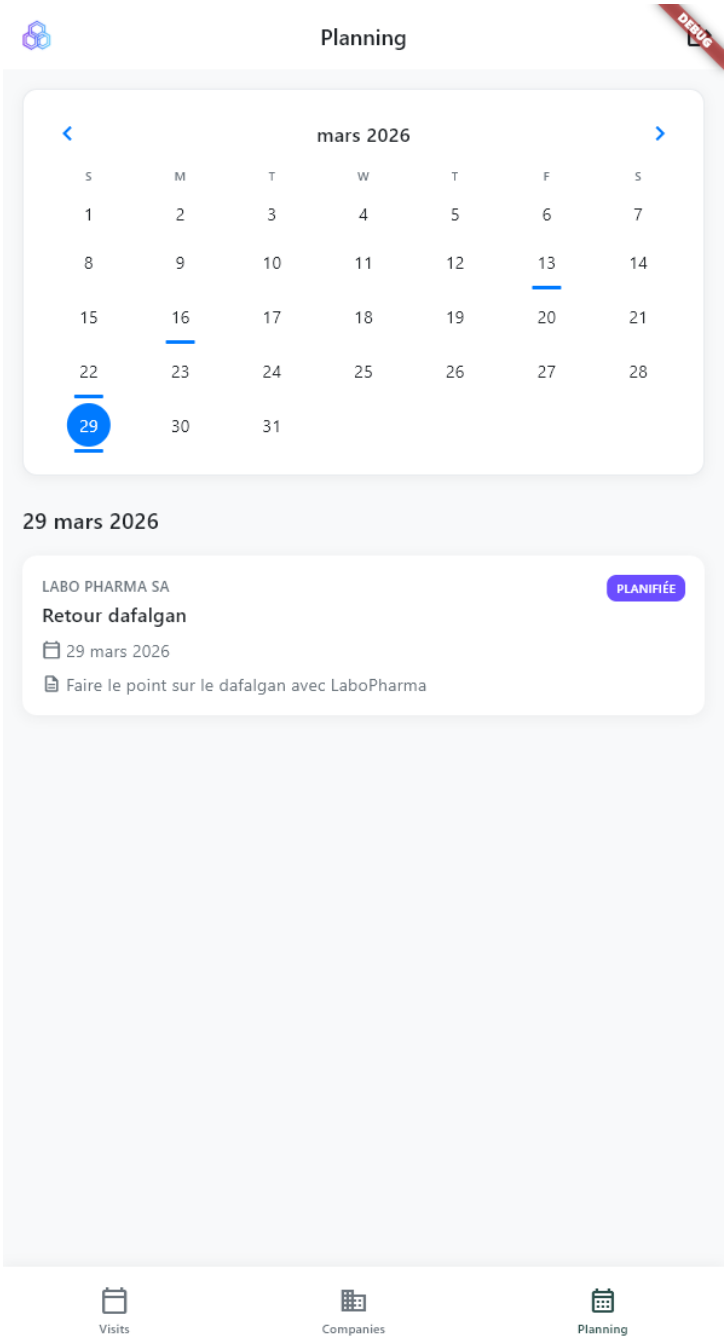
- titre,
- statut,
- dates,
- lieu,
- produit,
- commentaire,
- contact client,
- note de satisfaction



Le responsable peut supprimer une visite.
Pour les visites au statut « Effectuée » ou « Refusée », un bouton permet d’exporter la fiche en PDF.

Planning

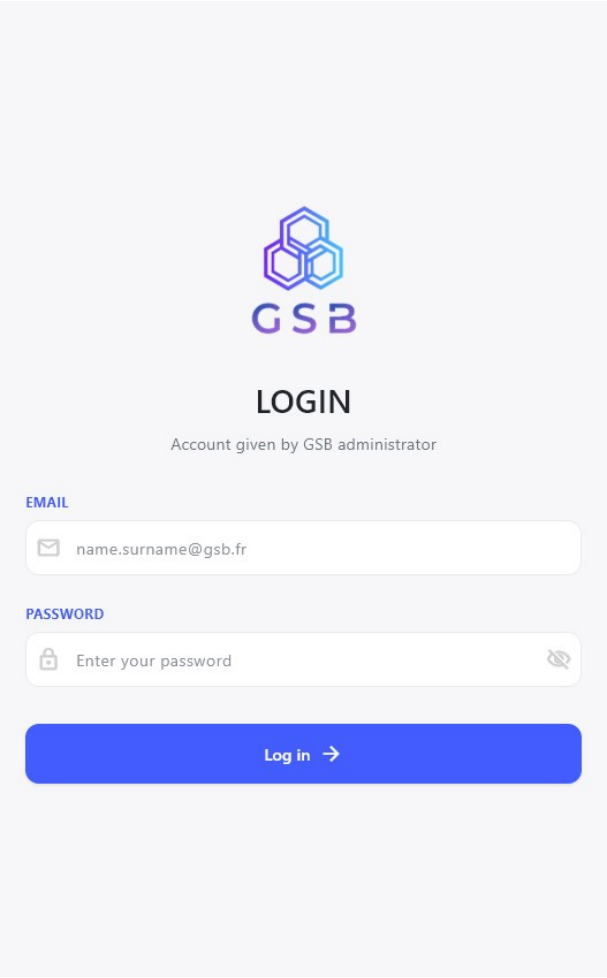
La section Planning affiche un calendrier mensuel.
Les jours avec au moins une visite sont indiqués.
La sélection d’un jour affiche la liste des visites de ce jour.
Un clic sur une visite ouvre la fiche détail de la visite.



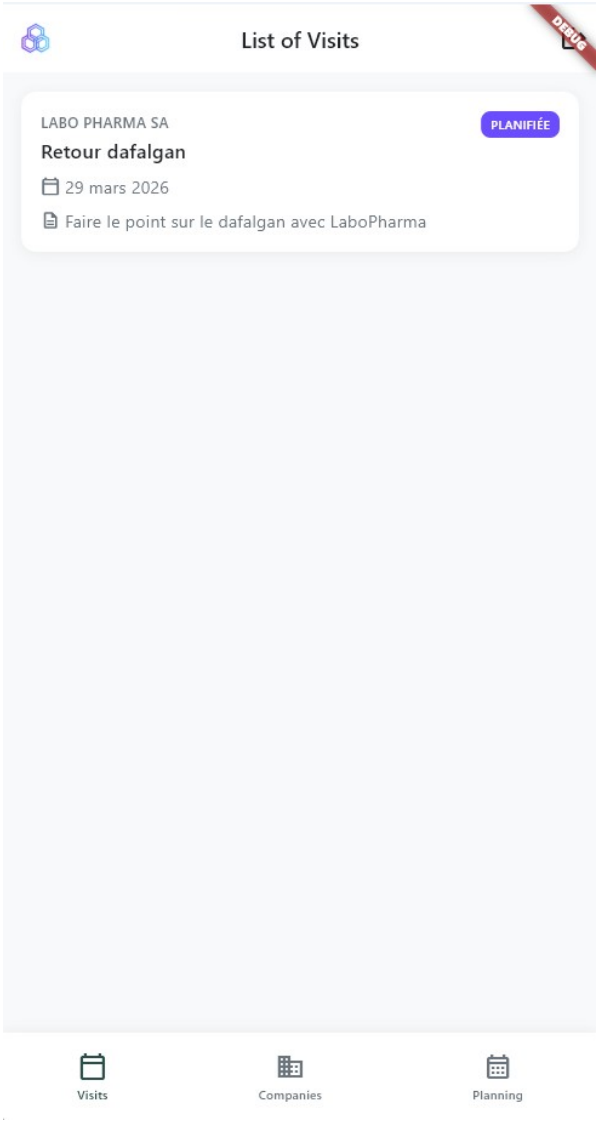
c) Visiteur médical

Connexion

Le visiteur médical se connecte avec son adresse e-mail et son mot de passe.
Écran des visites



Après la connexion, le visiteur médical arrive sur l'écran des visites.
Il y voit la liste des visites qui le concernent.



Un bouton Déconnexion est visible.
Un clic déconnecte l'utilisateur et termine la session.

Le visiteur médical ne peut pas créer de visite (pas de bouton « Créer une visite »).

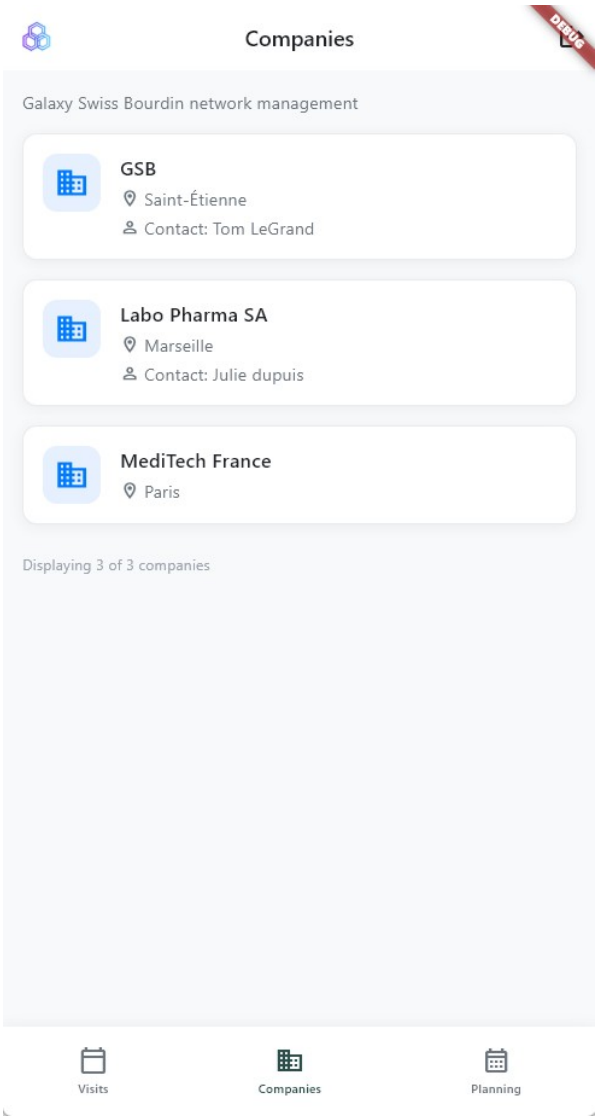
Menu de navigation

Le menu principal permet d’accéder aux sections suivantes :

- Visites
- Entreprises
- Planning

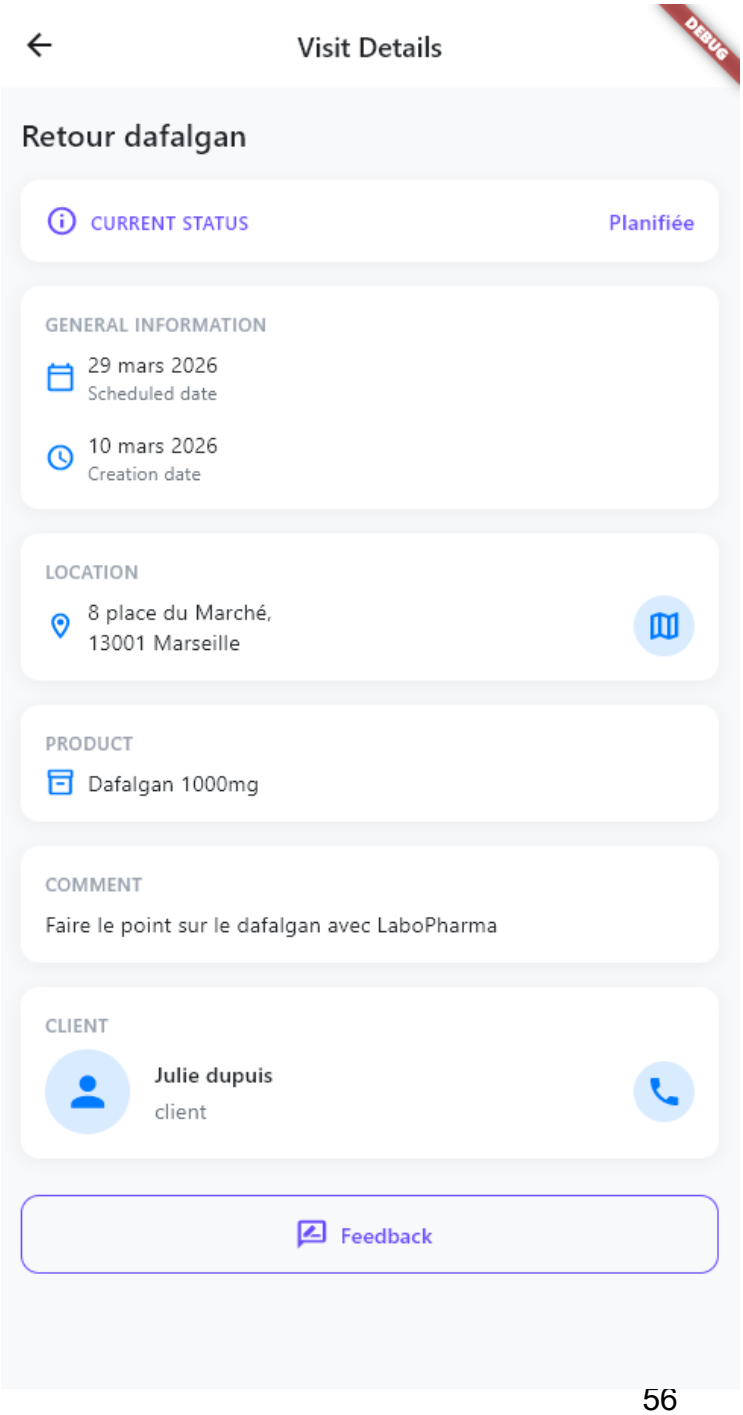
Entreprises

La section Entreprises affiche la liste des entreprises accessibles au visiteur.
Un clic sur une entreprise ouvre la fiche entreprise.

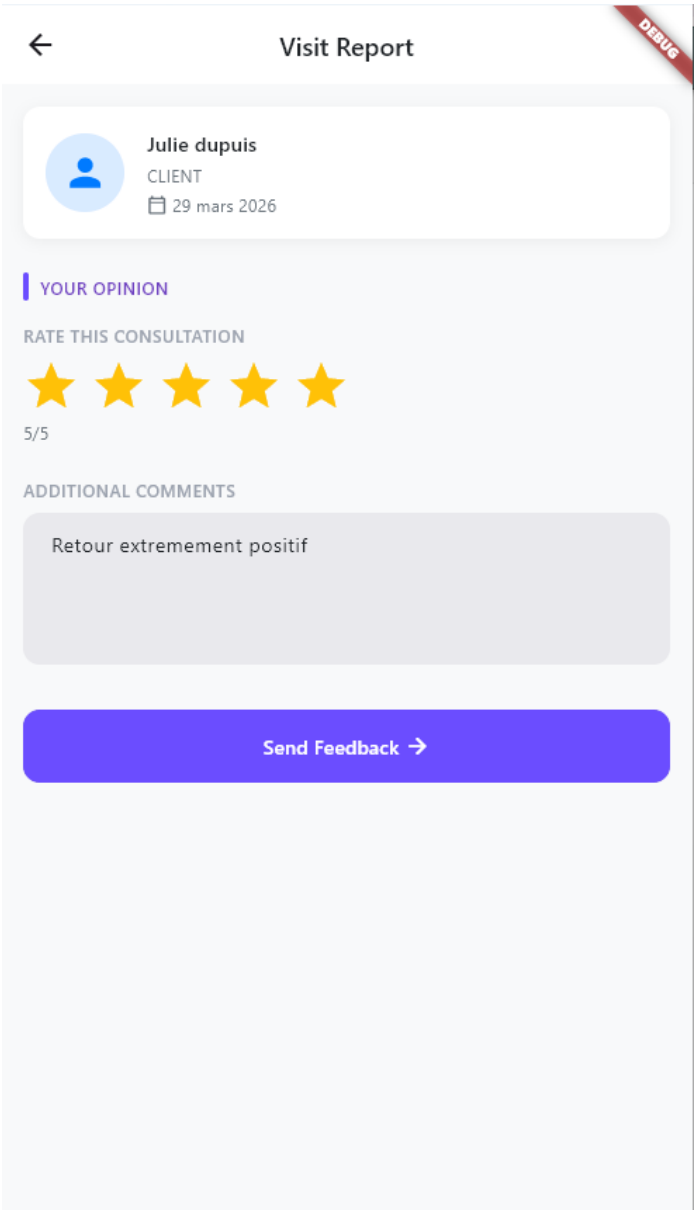


Visites – Fiche et retour de visite

Un clic sur une visite ouvre la fiche détaillée.
Pour les visites au statut « Planifiée », un bouton « Feedback » (ou équivalent) est disponible.



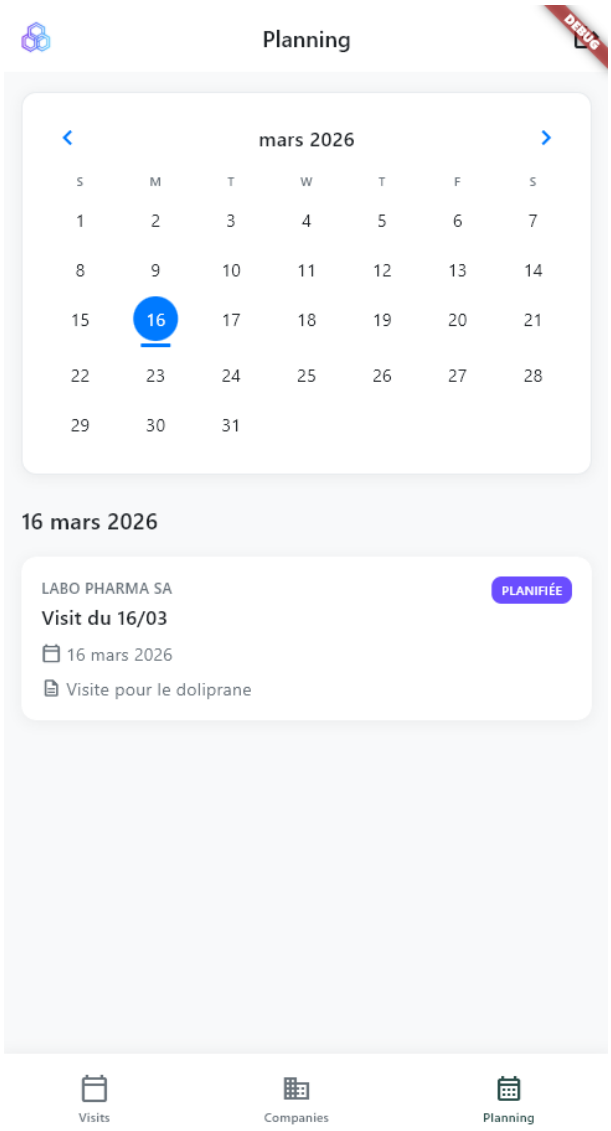
Un clic ouvre l’écran de saisie du retour : notation (par exemple de 1 à 5) et commentaire.
Un clic sur le bouton d’envoi enregistre le retour.



Un message de confirmation s’affiche et l’utilisateur est ramené à la liste des visites.

Planning

La section Planning affiche un calendrier mensuel.
Les jours avec au moins une visite sont repérés.



La sélection d'un jour affiche la liste des visites de ce jour.
Un clic sur une visite ouvre la fiche détail de la visite.

VIII. Accès

URL : https://gsb-nolan-liogier.fr/

Pour vous connecter en tant que client :

• email : j.dupuis@gsb.com
• password : ++PWDclientGSB2026

Pour vous connecter en tant que visiteur médical :

• email : j.martin@gsb.com
• password : ++PWDvmGSB2026

Pour vous connecter en tant que responsable

• email : t.legrand@gsb.com
• password : ++PWDresGSB2026