RÉSUMÉ DE PROJET BOUTIQUE EN LIGNE

Sommaire

I. INTRODUCTION

Ce rapport présente la conception et la réalisation d'une application mobile destinée à la gestion des abonnements personnels et familiaux. Le client accède à l'application depuis un smartphone via Expo (React Native), tandis que la logique métier et la persistance des données sont prises en charge par une API développée avec Symfony.

La solution a pour ambition de centraliser les informations essentielles liées aux abonnements — telles que le service concerné, le coût, la fréquence de facturation et les dates d'échéance — afin d'offrir une vision claire et exploitable du portefeuille d'abonnements de l'utilisateur.

Dès la première version, l'application permet de créer et de modifier des abonnements, de les associer à des membres d'un espace, de leur attribuer des tags et de consigner les paiements effectués. Un système de notifications est prévu pour informer l'utilisateur des renouvellements imminents ou des actions à entreprendre. L'objectif global est de simplifier le suivi des dépenses récurrentes et de réduire les oublis ou les doubles souscriptions.

II. PARCOURS UTILISATEUR

Le parcours utilisateur démarre par une phase d'authentification. Après avoir créé un compte ou s'être connecté, l'utilisateur est redirigé vers un tableau de bord qui lui présente la liste de ses abonnements actifs ainsi qu'une estimation du coût mensuel moyen.
À partir de cette vue, il peut ouvrir le détail d'un abonnement pour consulter ses caractéristiques et le modifier si nécessaire. La création d'un abonnement se fait en plusieurs étapes fluides : l'utilisateur choisit un service dans un catalogue ou saisit un service libre, définit le montant et la devise, précise le mode et la fréquence de facturation et indique la date de début ainsi que, le cas échéant, la date de fin. Il peut activer le renouvellement automatique et ajouter des notes.

L'abonnement peut ensuite être rattaché à un membre de l'espace (par exemple un conjoint ou un enfant) et recevoir un ou plusieurs tags, ce qui facilite les recherches et les regroupements thématiques. Les paiements relatifs à un abonnement sont enregistrés depuis l'écran de détail. L'utilisateur conserve ainsi un historique de ses règlements et peut rapidement calculer un total versé ou vérifier la cohérence entre la fréquence attendue et les paiements observés.

À mesure que les échéances approchent, l'application prépare des notifications afin d'attirer l'attention de l'utilisateur sur les actions à entreprendre.
Le périmètre de cette première version exclut volontairement l'encaissement réel via une passerelle de paiement et se concentre sur la tenue et la mise à jour d'informations fiables. Les fonctionnalités de collaboration, via les espaces et les membres, posent cependant les bases d'une gestion partagée à moyen terme.

III. CONCEPTION BASE DE DONNÉES

La base de données repose sur PostgreSQL et utilise des identifiants de type UUID pour toutes les clés primaires et étrangères.
Les entités sont normalisées et accompagnées d'index sur les clés étrangères afin de garantir des jointures efficaces. Les horodatages sont enregistrés sous forme de timestamps immuables, ce qui permet d'assurer une bonne traçabilité. Le noyau fonctionnel s'articule autour de plusieurs ensembles complémentaires.

Le premier regroupe la gestion des abonnements à travers les tables « users », « services » et « subscriptions », auxquelles s'ajoute la table « payments » pour consigner les règlements. La personnalisation est apportée par les « tags » et la table d'association « subscription_tags » qui autorise un étiquetage souple et évolutif. Le second ensemble vise la collaboration grâce aux « spaces », « members » et « permissions », qui structurent un foyer ou une équipe. Enfin, les « notifications » et leurs « notification_targets » constituent le mécanisme de messagerie interne et de suivi de lecture.

Sur le plan des relations, un utilisateur possède plusieurs abonnements, et chaque abonnement est rattaché à un service. Un abonnement peut comporter plusieurs paiements au fil du temps. Les tags sont associés aux abonnements via une table d'association, ce qui permet de réaliser des recherches croisées et des regroupements thématiques. Les espaces contiennent des membres et peuvent définir des permissions par utilisateur. Les notifications sont diffusées à un ou plusieurs destinataires, et l'application conserve l'état de lecture pour chaque cible. Les contraintes d'intégrité référentielle ont été choisies pour traduire fidèlement le cycle de vie des données. Ainsi, la suppression d'un utilisateur entraîne la suppression des abonnements qui en dépendent, tandis que la suppression d'une catégorie ne supprime pas les services mais met à jour leur référence afin d'éviter toute rupture fonctionnelle. Ce compromis permet de protéger les données critiques tout en facilitant l'évolution du catalogue.

IV. ARCHITECTURES ET TECHNOLOGIES UTILISÉES

L’architecture suit une séparation stricte entre le client et le serveur. Le client, développé avec Expo et React Native, se charge de l’ergonomie, de la navigation et de l’interaction avec l’utilisateur.

Il communique avec une API HTTP sécurisée par jeton JSON Web Token. Le serveur, réalisé avec Symfony, expose des endpoints REST, applique les règles métier, contrôle la validation des données et persiste les informations dans PostgreSQL via Doctrine ORM.

La configuration CORS est ajustée pour autoriser exclusivement les origines pertinentes, et les rôles applicatifs sont stockés sous forme JSON afin de conserver une grande flexibilité.
Ce découplage permet d’itérer rapidement sur l’interface mobile tout en garantissant la cohérence des données côté serveur.
Il facilite également l’ajout de fonctionnalités transverses, comme la pagination systématique des réponses, le contrôle fin des statuts ou la mise en place d’un mécanisme de notifications extensible vers des pushs natifs.

V. GESTION DES ROUTES

Les routes de l’API suivent une convention lisible et prévisible.
L’authentification fournit un jeton qui doit être transmis avec chaque requête conséquente.
Des points d’entrée spécifiques existent pour récupérer le profil de l’utilisateur, interroger le catalogue des services, créer et modifier des abonnements, ou encore consigner des paiements.
Les ressources liées aux tags, aux espaces et aux membres restent disponibles afin d’enrichir progressivement l’usage.
Côté mobile, la navigation repose sur une combinaison de piles et d’onglets, ce qui permet d’accéder rapidement aux écrans les plus consultés tout en conservant un historique de navigation naturel.

VI. FONCTIONNALITÉS DU SITE

L’authentification repose sur un échange de jetons à durée de vie maîtrisée. Le client stocke ce jeton dans un espace sécurisé et le joint à chaque appel pour bénéficier des droits nécessaires.

Le cœur fonctionnel réside dans la gestion des abonnements : création, consultation et mise à jour des informations essentielles, rattachement à des membres, affectation de tags et suivi des paiements.

La présentation des listes s’accompagne de filtres et de tris qui exploitent les index de la base de données afin de fournir une réponse rapide même lorsque le volume de données augmente.
Les notifications constituent un axe d’engagement important. Dans un premier temps, elles sont consultables au sein de l’application avec un statut de lecture par destinataire. À moyen terme, l’intégration des notifications push permettra d’avertir l’utilisateur sans qu’il ait à ouvrir l’application. Cette progression graduelle permet d’installer la mécanique sans complexifier inutilement la première version.

VII. SÉCURITÉ ET OPTIMISATION

La sécurité est assurée à plusieurs niveaux. Les mots de passe sont “hashés” côté serveur et ne sont jamais exposés en clair. L’API limite l’accès aux ressources en fonction de rôles et de permissions, et les entrées utilisateur sont systématiquement validées. La communication entre le client et le serveur se fait par HTTPS, et la configuration CORS restreint les origines autorisées. Du point de vue de la performance, les index placés sur les clés étrangères et la pagination par défaut des listes évitent les requêtes coûteuses. Les journaux d’erreur et quelques informations d’audit permettent de suivre l’activité et de diagnostiquer les incidents.

VIII. TESTS

La qualité est maintenue par une combinaison d’outils et de bonnes pratiques. Des jeux de requêtes permettent de vérifier rapidement le bon fonctionnement des endpoints de l’API. Les tests unitaires ciblent les services et les validateurs les plus sensibles, tandis que des tests fonctionnels garantissent le comportement attendu des contrôleurs.

Côté mobile, des outils de formatage et d’analyse statique fluidifient la revue de code. Enfin, la séparation des environnements et la gestion rigoureuse des secrets contribuent à la stabilité de l’ensemble.