



Documentation fonctionnelle

Description du produit

Digital Wallet est une application web (Next.js) de portefeuille numérique, conçue pour gérer plusieurs wallets par utilisateur et opérer en multi-devises (EUR, USD, GBP). L'utilisateur peut consulter ses soldes, effectuer des transferts internes (entre wallets de la plateforme) et démarrer des opérations de paiement.

Le produit intègre un parcours de dépôt via Stripe (Checkout + webhook de confirmation), un parcours de retrait (cashout) vers IBAN, et un mode d'échange inter-systèmes via un protocole inter-wallet basé sur des requêtes signées en HMAC-SHA256. Une marge plateforme de 1% est appliquée selon les cas et centralisée dans un wallet système. Une détection de fraude par scoring (ACCEPTED / REVIEW / BLOCKED) encadre la création des transactions.

Références code (principales): [src/app/api/wallets/route.ts](#) , [src/app/api/transactions/route.ts](#) ,
[src/app/api/payments/*](#) , [src/lib/payments.ts](#) , [src/lib/interwallet.ts](#) , [src/lib/fraud.ts](#) , [src/lib/platform-fee.ts](#) .

Problème utilisateur

L'utilisateur a besoin d'un outil simple pour centraliser de l'argent et le séparer par "enveloppes" (plusieurs wallets) afin de mieux organiser ses dépenses. Il veut pouvoir envoyer et recevoir des fonds rapidement, y compris quand les devises diffèrent, sans devoir calculer manuellement les conversions.

Il attend aussi une expérience crédible quand il ajoute ou retire de l'argent: dépôts via un paiement en ligne (Stripe) et retraits vers un compte bancaire. Enfin, il doit pouvoir faire confiance au système (historique, statuts, traçabilité) et bénéficier d'une première couche de protection (anti-fraude). Dans le contexte du projet, l'interopérabilité avec d'autres "wallets" (autres groupes/systèmes) est un besoin clé, via un protocole signé plutôt qu'un simple appel non authentifié.

Parcours utilisateur

1) Onboarding / Auth

Depuis la page d'accueil (`/`), l'utilisateur choisit de créer un compte (`/register`) ou de se connecter (`/login`). À l'inscription, un wallet principal est créé automatiquement (EUR) et une session est établie via un cookie `auth_token` (JWT). Le dashboard est accessible uniquement si la session est valide.

Références: `src/app/api/auth/register/route.ts` , `src/app/api/auth/login/route.ts` ,
`src/app/api/auth/me/route.ts` , `src/lib/auth.ts` .

2) Dashboard

Une fois connecté, l'utilisateur arrive sur `/dashboard` et consulte une vue d'ensemble: la liste des wallets, un solde total converti en EUR, ainsi qu'un extrait des transactions récentes.

Références: `src/app/(dashboard)/dashboard/page.tsx` , `src/app/(dashboard)/layout.tsx` .

3) Gestion des wallets

Dans `/wallets` , l'utilisateur consulte l'ensemble de ses wallets et peut en créer de nouveaux (limité à 5). Chaque wallet a un nom, une devise, et un solde. Depuis

la page de détail `/wallets/[id]`, il retrouve les transactions associées et des actions rapides pour créditer ou retirer.

Références: `src/app/(dashboard)/wallets/page.tsx` , `src/app/(dashboard)/wallets/[id]/page.tsx` ,
`src/app/api/wallets/route.ts` .

4) Créditer un wallet (Deposit)

Depuis `/deposit`, l'utilisateur choisit le wallet à créditer, saisit un montant (min 5, max 1000) et voit un récapitulatif du total à payer (frais Stripe + marge plateforme 1%). Il est ensuite redirigé vers Stripe Checkout. Le crédit effectif du wallet est déclenché par le webhook Stripe après confirmation du paiement.

Références: `src/app/(dashboard)/deposit/page.tsx` , `src/app/api/payments/deposit/route.ts` ,
`src/app/api/payments/webhook/route.ts` , `src/lib/payments.ts` .

5) Envoyer de l'argent (Transfert local)

Depuis `/transactions`, l'utilisateur initie un transfert en sélectionnant un wallet source, puis en saisissant l'email du destinataire afin de sélectionner un wallet cible. Le montant saisi correspond au montant reçu sur le wallet de destination (dans sa devise). Si les devises diffèrent, le serveur calcule le montant à débiter via conversion, puis exécute débit/crédit.

Une marge plateforme de 1% est appliquée uniquement si le transfert est effectué vers un autre utilisateur; les transferts entre wallets du même utilisateur n'entraînent pas de frais.

Références: `src/app/(dashboard)/transactions/page.tsx` , `src/app/api/transactions/route.ts` ,
`src/lib/currency.ts` .

6) Transfert inter-wallet (entre systèmes)

Depuis `/inter-wallet`, l'utilisateur saisit l'URL du système distant, l'identifiant du wallet distant, et le montant. Le système envoie alors une requête vers l'endpoint du système externe, avec une signature HMAC-SHA256 calculée sur

le payload. La transaction est suivie via un statut et des logs d'échanges inter-systèmes.

Références: `src/app/(dashboard)/inter-wallet/page.tsx`, `src/lib/interwallet.ts`, `src/app/api/inter-wallet/*`.

7) Retrait (Cashout)

Depuis `/cashout`, l'utilisateur sélectionne un wallet, saisit un montant ($>= 10$) et un IBAN. Le système débite le wallet du montant demandé, ajoute la marge (1%) et un frais de traitement fixe (0.25 pour virement bancaire), puis enregistre une transaction de retrait (WITHDRAWAL) initialement en PENDING.

Références: `src/app/(dashboard)/cashout/page.tsx`, `src/app/api/payments/cashout/route.ts`,
`src/lib/payments.ts`.

Fonctionnalités du MVP

Le MVP couvre l'authentification (inscription, connexion, déconnexion et récupération de l'utilisateur courant), la gestion des wallets (création et consultation), l'historique de transactions avec filtres, et l'exécution des transferts internes avec conversion de devise.

Il inclut également une couche anti-fraude par scoring, l'intégration Stripe pour les dépôts (Checkout + webhook), un cashout "initié" côté plateforme, et le protocole inter-wallet signé (transfer/validate/status) avec journalisation. La marge plateforme de 1% est prise en compte et accumulée dans un wallet système, et une documentation OpenAPI/Swagger est disponible.

Références: `src/app/api/*`, `src/lib/*`, `openapi.yaml`, `src/app/api-docs/page.tsx`, `tests/*`.

Limites actuelles

Transactions en REVIEW

Les transactions marquées REVIEW débiteront le wallet source, mais aucun workflow n'existe pour une approbation/rejet ultérieure permettant de créditer

effectivement le destinataire. Cela peut laisser des fonds "bloqués" en pratique.

Référence: [src/app/api/transactions/route.ts](#).

Cashout non finalisé

Le retrait ne déclenche pas un payout Stripe réel (la création Stripe est explicitement non implémentée). Le système enregistre l'intention et débite le wallet, mais ne réalise pas le virement effectif.

Référence: [src/lib/payments.ts](#).

Inter-wallet sortant: finalisation fragile

Le flux sortant ne vérifie pas la signature d'une réponse distante et ne met pas en place de mécanisme automatique de finalisation (validation/status polling) robuste. Des transactions peuvent rester en PROCESSING sans résolution automatique.

Références: [src/lib/interwallet.ts](#), [src/app/api/transactions/route.ts](#).

Inter-wallet sortant: vérification de solde manquante

Avant le débit d'un transfert inter-wallet sortant, le code ne vérifie pas le solde disponible sur le wallet source, ce qui peut conduire à un solde négatif.

Référence: [src/app/api/transactions/route.ts](#) (fonction `handleInterWalletTransfer`).

Gestion de session côté serveur limitée

La table `Session` est bien alimentée (login/register) et purgée au logout, mais la validation de session lors des requêtes repose surtout sur la vérification JWT. Il n'y a pas de contrôle systématique "token présent en base" ou "expiresAt" pour renforcer la révocation serveur.

Références: [src/lib/auth.ts](#), [src/app/api/auth/*](#).

Dépendances externes pour la conversion de devise

La conversion s'appuie sur une API publique de taux de change et un cache en mémoire (1h). En cas d'indisponibilité/quota, le service peut être dégradé.

Référence: [src/lib/currency.ts](#).