**Mémoire Projet**

*Concepteur développeur d’applications*

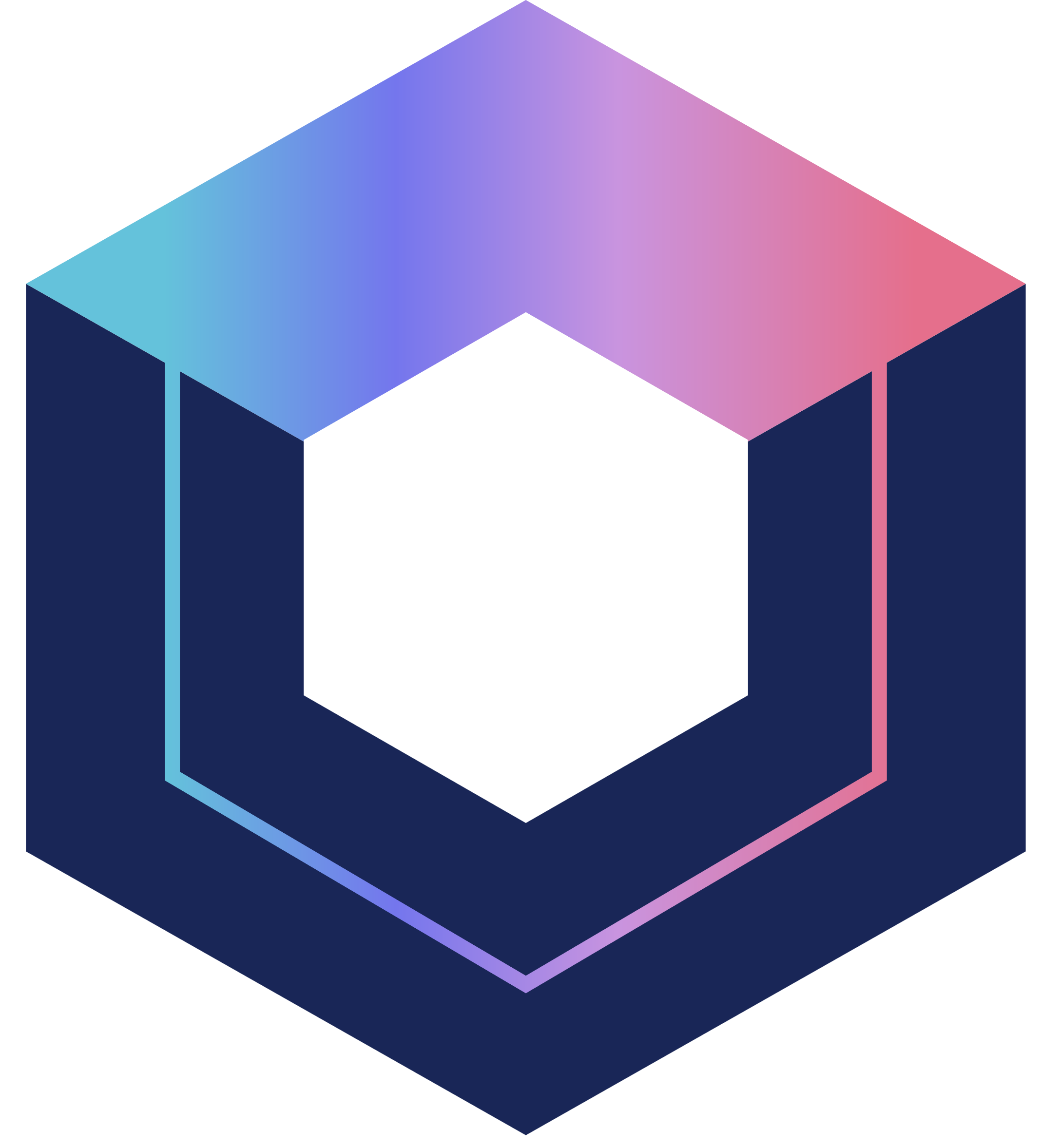
Application BlockLumen Plateforme de simulation de trading de cryptomonnaies.

Table des matières

[1. Présentation 4](#_Toc201187909)

[a. Introduction en français 4](#_Toc201187910)

[2. Cahier des charges 5](#_Toc201187911)

[a. Contexte 5](#_Toc201187912)

[b. Objectif principaux 5](#_Toc201187913)

[c. Fonctionnalités essentielles 5](#_Toc201187914)

[d.Technologies 7](#_Toc201187915)

[1. Back-end 7](#_Toc201187916)

[2. Front-end 7](#_Toc201187917)

[3. OS / IDE / Utilitaires 7](#_Toc201187918)

[3. Le Design 8](#_Toc201187919)

[a. Maquettage et Conception Graphique 8](#_Toc201187920)

[1. Logo 8](#_Toc201187921)

[2. Palette graphique 8](#_Toc201187922)

[3. La typographie 8](#_Toc201187923)

[4. Maquette & WireFrames 9](#_Toc201187924)

[4. Conception et modélisation des données 10](#_Toc201187925)

[a. Clarification des besoins 10](#_Toc201187926)

[b. Dictionnaire de données 11](#_Toc201187927)

[c. Modèle Conceptuel Des Données ( MCD ) 12](#_Toc201187928)

[d. Modèle Logique Des Données ( MLD ) 13](#_Toc201187929)

[e. Modèle Physique Des Données ( MPD ) 14](#_Toc201187930)

[f. Relations 15](#_Toc201187931)

[g. Diagrammes UML 17](#_Toc201187932)

[1. Diagramme de cas d’utilisation UML : 19](#_Toc201187933)

[2. Diagramme d’activités UML 20](#_Toc201187934)

[3. Diagramme d’entités 22](#_Toc201187935)

[5. Environnement de développement 23](#_Toc201187936)

[1. Créer la structure de base du Frontend avec Vite + React + TypeScript 25](#_Toc201187937)

[2. Installation des dépendances pour le Frontend et affichage page React 25](#_Toc201187938)

[3. Créer le BackEnd 27](#_Toc201187939)

[4. Mise en place de Docker - BackEnd 28](#_Toc201187940)

[5. GitHub 31](#_Toc201187941)

[6. Développement de l’application 32](#_Toc201187942)

[1. Création du Backend 32](#_Toc201187943)

[a. Création base de données et Migrations 32](#_Toc201187944)

[b . Création des entités - Backend 34](#_Toc201187945)

[c. Création des contrôleurs ( CRUD basique / Backend ) 35](#_Toc201187946)

[d. Création des routes et test routes - Backend 35](#_Toc201187947)

[2. Création du Frontend 36](#_Toc201187948)

[a. Contexte (Auth, Session) 36](#_Toc201187949)

[b. Router (Public vs .Privé) 37](#_Toc201187950)

[c. AppConstant et ApiConstant 38](#_Toc201187951)

[d. Redux store 39](#_Toc201187952)

[e. Redux/slices/tradeSlice.ts & services/api.ts 40](#_Toc201187953)

[f. Inscription et connexion des utilisateurs 41](#_Toc201187954)

[g. Charts et Consommation par API des cours actuel des cryptos 41](#_Toc201187955)

[h. RGPD et Cookies 41](#_Toc201187956)

[3. Securité 41](#_Toc201187957)

[7. Conclusion 42](#_Toc201187958)

[a. Remerciements 42](#_Toc201187959)

[b. Axes d’amélioration 42](#_Toc201187960)

[c. Retour d’expérience 42](#_Toc201187961)

[Annexes 43](#_Toc201187962)

# 1. Présentation

## a. Introduction en français

Je m’appelle Nicolas Lopez. En 2022, j’ai intégré la Prépa Concepteur Développeur d’Applications à l’Idem, où j’ai validé ma première année avec succès.

Dans ce cadre, j’ai effectué un stage de fin d’études chez REVIMPORT S.A.S., dont la mission était double : me familiariser avec les process en entreprise et débuter une migration de leur l’ERP Navision vers Microsoft Business Central.

Business central est le même ERP mais en version cloud et moderne avec la possibilité de créer des extensions plutôt que de modifier le code source. Mon implication m’a valu une gratification de stage et l’offre d’un contrat d’alternance d’un an.

Durant cette alternance, j’ai poursuivi l’analyse fonctionnelle, la conception de la base de données et le développement des premiers modules métiers. Malheureusement, REVIMPORT a été placée en liquidation judiciaire, entraînant un licenciement économique et l’interruption du projet de migration.

Le lendemain, je me suis rendu au job dating organisé par l’Idem afin de rencontrer des entreprises du secteur. J'ai eu l'opportunité de m'entretenir avec un membre de l'équipe d'**Alternea**, une entreprise de Perpignan (Saint-Charles) spécialisée dans le **commerce de gros (commerce interentreprises) de fruits et légumes**.

J’ai été retenu pour débuter une alternance de deux ans, durant laquelle je serai chargé de moderniser l’UI/UX de l’interface utilisées par l’ensemble des employés. J’ai à ce jour déjà signé le contrat et j’attends avec impatience de commencer ce projet en début septembre pour la rentrée.

Aujourd’hui, dans le cadre de ma deuxième année et pour valider mon titre RNCP 6, je consacre mon mémoire à BlockLumen, une plateforme de simulation de trading de cryptomonnaies. Ce projet représente l’aboutissement de ma formation et de mes expériences.

# 2. Cahier des charges

## a. Contexte

Le marché des cryptomonnaies, en forte croissance, reste très volatil et complexe pour les débutants. BlockLumen propose une simulation sécurisée du trading (achats, ventes, gestion de portefeuille) enrichie de modules. Ce nom, court et évocateur, associe “Block” (blockchain) et “Lumen” (lumière), pour symboliser l’éclaircissement et la compréhension du trading crypto. Afin de mieux appréhender la phase initiale je crée un cahier des charges.

## b. Objectif principaux

**Expérience utilisateur complète** :

* Authentification sécurisée (JWT + bcrypt), création de compte, gestion de profil et préférences.
* Multi-portefeuille (chaque utilisateur peut avoir plusieurs portefeuilles).
* Historique détaillé des transactions (achats, ventes, frais).

**Calcul de valeur précis** :

* Chaque détention (Wallet\_Holding) stocke quantité et prix moyen.
* Valeur de portefeuille calculée dynamiquement à partir des cours historiques (Price).

**Modules pédagogiques** :

* Séquences d’apprentissage (texte, vidéo, quiz).
* Suivi de progression (User\_Learn).

**Responsivité** : fonctionnement fluide sur desktop, tablette et mobile.

**Maintenabilité & scalabilité** :

* Architecture modulaire (React + Node.js/Express + MySQL).
* Containerisation (Docker + Docker Compose).

## c. Fonctionnalités essentielles

**Authentification & Profil**

* **Créer un compte** : formulaire (username, email, password), mots de passe hachés.
* **Se connecter / Se déconnecter** : via JWT, middleware de protection des routes.
* **Modifier profil / préférences** : mettre à jour mot de passe, email et options (thème, notifications).

**Gestion des Portefeuilles**

* **Créer un portefeuille** : initialiser avec solde virtuel (10 000 $).
* **Lister mes portefeuilles** : affichage de l’ensemble des portefeuilles d’un utilisateur.
* **Voir détail d’un portefeuille** :
  + Liste des détentions (Wallet\_Holding) pour chaque cryptomonnaie (quantité, prix moyen).
  + Calcul automatique de la valeur totale (intégration des cours).
  + Visualisation graphique des variations (Chart.js).

**Gestion des Holdings (via Trade)**

* **Créer / Mettre à jour un holding** :
  + À l’achat, si la ligne existe, mise à jour de la quantité et recalcul du prix moyen ; sinon création d’une nouvelle ligne.
  + À la vente, décrémenter la quantité, supprimer si zéro.
* Cette logique est déclenchée automatiquement lors de chaque transaction (Trade).

**Simulation de Trading (Trade)**

* **Créer une transaction** (Trade) :
  + Enregistrement des opérations d’achat/vente : holding\_id, crypto\_symbol, type, amount, price, fee, timestamp.
  + Vérification préalable du solde disponible (pour achat).
  + Calcul automatique des frais (pourcentage ou montant fixe).
  + Mise à jour du Wallet\_Holding associé.
* **Consulter historique des transactions** : filtre par portefeuille ou cryptomonnaie, pagination, tri par date.

**Suivi des Cours (Price)**

* **Insertion périodique des cours** : récupération via API tierce (CoinGecko) et insertion dans la table Price (crypto\_symbol, value, recorded\_at).
* **Graphiques en temps réel** : requêtes pour afficher les dernières valeurs (Chart.js).

**Modules Pédagogiques (Learn)**

* **Lister les modules** : affichage des titres et ordre de chaque module.
* **Consulter un module** : contenu pédagogique (texte, images, vidéos).
* **Marquer comme complété** : mise à jour de User\_Learn (is\_completed, completed\_at) pour suivre la progression.

## d.Technologies

### 1. Back-end

**Node.js & Express :** Cadre JavaScript léger et pour développement.

**MySQL :** Base de données relationnelle robuste, hébergeant utilisateurs, portefeuilles et historiques de transactions.

**Docker :** Containerisation du serveur et de la base de données pour garantir portabilité et isolation.

### 2. Front-end

**React 18  :** Bibliothèque JavaScript pour construire une interface dynamique et réactive.

**Vite :** Outil de bundling ultra-rapide, optimisé pour le développement moderne.

**Tailwind CSS :** Framework utilitaire facilitant la mise en forme et le design cohérent.

**React-Chart.js-2 / Chart.js :** Pour afficher graphiques de prix et indicateurs en temps réel.

**TypeScript :** Typage statique pour renforcer la fiabilité et la maintenabilité du code.

### 3. OS / IDE / Utilitaires

**Git & GitHub :** Gestion de versions et collaboration.

**VS Code :** Environnement de développement, avec extensions TypeScript et Docker.

**Postman :** Tests et appels d’API.

**Figma :** Prototypage des écrans et design UX/UI.

**app.diagrams.net / Draw.io :** Modélisation du MCD et des architectures applicatives.

# 3. Le Design

## a. Maquettage et Conception Graphique

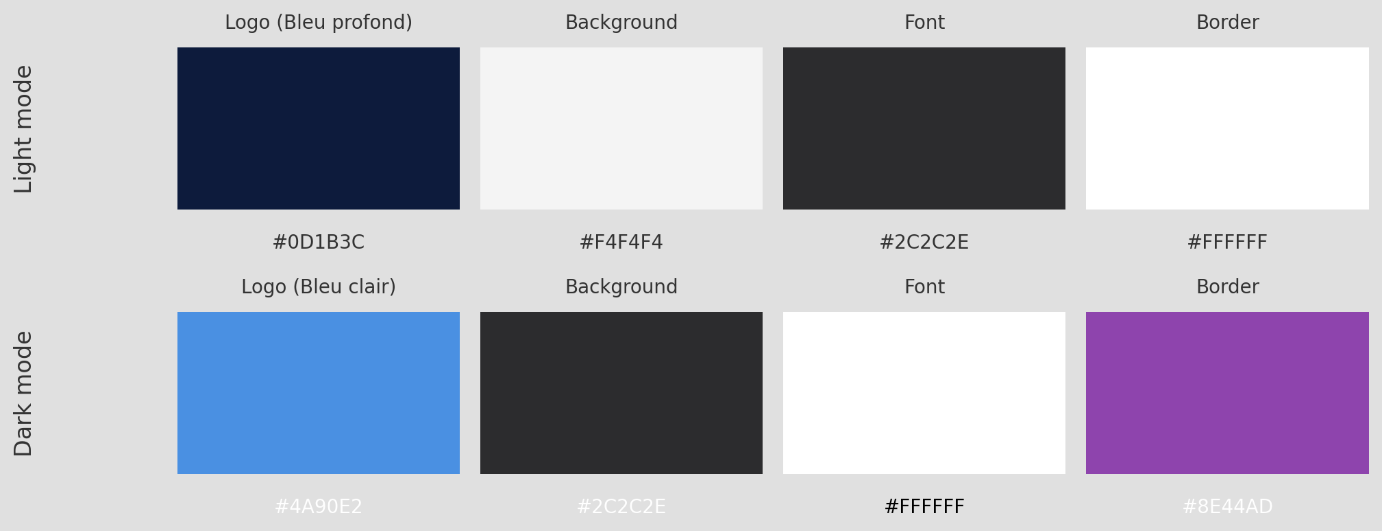
La phase de design a été réalisée à l’aide de **Figma**, outil principal pour l’élaboration des wireframes et des maquettes visuelles de l’application. Après définition de la charte graphique (logo, palette de couleurs, typographies), chaque écran a été prototypé dans Figma pour valider l’ergonomie et la cohérence visuelle avant le développement.

### 1. Logo

Ce logo associe une forme hexagonale inspirée de la blockchain à un dégradé bleu-pourpre-rose évoquant la lumière et la clarté pédagogique, tandis que la dominante de bleu profond installe une sensation de confiance et de sérieux ;

Le logo a été créé dans Figma au format vectoriel (SVG).

### 2. Palette graphique



### 3. La typographie

Pour les titres et intertitres, la police **Poppins** est utilisée en gras (700) pour les niveaux H1 et H2, et en semi-gras (600) pour H3 et H4 ; leur couleur est le bleu profond (#0D1B3C) sur fond clair ou le blanc (#FFFFFF) sur fond sombre.

Pour le corps de texte, on adopte **Roboto** en regular (400) pour les paragraphes et en medium (500) pour les labels et mentions importantes ; la couleur est le gris anthracite (#2C2C2E) sur fond clair, ou le blanc (#FFFFFF) sur fond bleu profond.

### 4. Maquette & WireFrames

Des wireframes ont été réalisées afin de valider la structure et l’enchaînement des écrans, d’optimiser le parcours utilisateur et de clarifier l’emplacement des éléments.

Une image contenant texte, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

# 4. Conception et modélisation des données

Pour structurer la base de données de BlockLumen, la méthode Merise a été utilisée, fondée sur la distinction entre modèles conceptuel, logique et physique. Après une phase d’étude préalable des besoins un MCD détaillé a été élaboré, puis traduit en MLD et enfin en MPD. Cette démarche assure la cohérence, la performance et l’évolutivité de la solution.

La structuration des données de BlockLumen est déclinée en quatre étapes :

1. Clarification des besoins & dictionnaire de données,
2. Modèle conceptuel de données (MCD),
3. Modèle logique de données (MLD),
4. Modèle physique de données (MPD).

Pour les parties Merise, Base de données et développement, une convention en anglais a été adopté.

## a. Clarification des besoins

La phase de clarification a pour objectif de recenser et formaliser l’ensemble des besoins fonctionnels et des flux d’information de BlockLumen, puis de définir un dictionnaire précisant chaque donnée manipulée par le système. Cette étape garantit une base solide pour la modélisation conceptuelle, en assurant cohérence terminologique et exhaustivité des entités.

L’ensemble des exigences s’articule autour de cinq processus clés :

1. **Gestion des comptes :** inscription, authentification sécurisée (JWT + bcrypt), récupération de mot de passe, et personnalisation des paramètres.
2. **Visualisation des cours :** affichage en temps réel (API CoinGecko ou dataset) de plusieurs cryptomonnaies avec sélection de périodes (1 h, 1 j, 1 sem., 1 mois).
3. **Simulation de trading :** crédit initial de 10 000 $ fictifs, module d’achat/vente avec calcul instantané des gains/pertes et mise à jour du solde.
4. **Modules pédagogiques :** séquence de leçons ordonnées pour découvrir les concepts du trading crypto.
5. **Portefeuille :** Gestion du portefeuille de cryptos.

La plateforme doit rester claire et responsive (desktop, tablette, mobile) et regrouper les sections **Accueil**, **Marchés**, **Trader**, **Apprendre**, **Comptes**. Ces processus et règles de gestion (API, sécurité, responsive) constituent le socle de la future modélisation Merise.

## b. Dictionnaire de données

Une image contenant texte, menu, capture d’écran

Le contenu généré par l’IA peut être incorrect.Afin de créer le dictionnaire de données, on relève chaque information à stocker.

## c. Modèle Conceptuel Des Données ( MCD )

Une image contenant texte, diagramme, Plan, Dessin technique

Le contenu généré par l’IA peut être incorrect.Après avoir établi le dictionnaire de données et en fonction des informations recueillies, j’ai identifié les entités, leurs attributs et les relations entre celles-ci. Cette étape m’a permis de créer un modèle conceptuel des données et ainsi définir les relations fonctionnelles, pour cela on utilise la notion de cardinalité au sein du mcd. Cela représente le nombre d’occurrences ou d’entités qui peuvent être associées à une autre entité au travers d’une relation. Elle permet ainsi de délimiter et de définir les contraintes liées aux associations entre les entités.

## d. Modèle Logique Des Données ( MLD )

Le Modèle Logique des Données (MLD) constitue une étape clé de la méthodologie Merise. Il représente une abstraction textuelle de la base de données à informatiser, servant de transition entre le Modèle Conceptuel de Données (MCD) et le Modèle Physique de Données (MPD). Son rôle est de traduire les entités et associations du MCD en une structure relationnelle concrète, permettant de définir le nombre et la composition des tables, ainsi que leurs contraintes d'intégrité, avant leur implémentation dans un Système de Gestion de Base de Données Relationnelle (SGBDR).

Le MLD est généralement présenté sous une forme textuelle linéaire, suivant des conventions spécifiques pour décrire chaque table et ses attributs :

* Chaque ligne correspond à la définition d'une table de la base de données.
* Le nom de la table est indiqué en premier.
* Les champs (attributs) de la table sont listés entre parenthèses et séparés par des virgules.
* Les clés primaires (PK) sont clairement identifiées, habituellement soulignées ou préfixées par un symbole distinctif (par exemple, #). Elles sont placées au début de la liste des champs pour en faciliter l'identification.
* Les clés étrangères (FK) sont également indiquées, souvent préfixées par un symbole spécifique (par exemple, \* ou un dièse # si elles font partie d'une clé primaire composite) pour signaler leur rôle de liaison avec d'autres tables.

MLD :

**User** ( #user\_id, username, email, password\_hash, created\_at, last\_login )

**Wallet** ( #wallet\_id, created\_at, balance, \*user\_id )

**Wallet\_Holding** ( #holding\_id, crypto\_symbol, quantity, average\_price, \*wallet\_id )

**Trade** ( #trade\_id, crypto\_symbol, type, amount, price, fee, timestamp, \*holding\_id )

**Preference** ( #preference\_id, key, value, \*user\_id )

**Price** ( #price\_id, crypto\_symbol, value, recorded\_at )

**Learn** ( #learn\_id, title, content, order\_index )

**User\_Learn** (#user\_learn\_id, is\_completed, completed\_at, \*user\_id, \*learn\_id )

## e. Modèle Physique Des Données ( MPD )

Une image contenant texte, diagramme, Plan, Dessin technique

Le contenu généré par l’IA peut être incorrect.Le MPD (Modèle Physique de Données) constitue la traduction concrète du MCD vers l'implémentation technique. Il spécifie précisément la structure des tables, les types de données de chaque attribut, l'identification des clés primaires et étrangères, ainsi que l'ensemble des contraintes d'intégrité nécessaires au bon fonctionnement du système de gestion de base de données SGBD (ici MySQL). J'ai réalisé le MPD avec draw.io, qui permet de visualiser clairement la structure de la base de données, les relations entre tables et les attributs de chaque entité.

## f. Relations

**User → Wallet : One-to-Many**

* Un utilisateur peut posséder plusieurs portefeuilles, et chaque portefeuille appartient à un seul utilisateur.
* **Implémentation :** dans la table **Wallet**, user\_id est NOT NULL et référencé par une FK → User(user\_id).

**Wallet → Wallet\_Holding : One-to-Many**

* Un portefeuille peut enregistrer plusieurs lignes de détention, chaque ligne de détention appartenant à un unique portefeuille.
* **Implémentation :** dans la table **Wallet\_Holding**, wallet\_id est NOT NULL et référencé par une FK → Wallet(wallet\_id).

**Wallet\_Holding → Trade : One-to-Many**

* Une ligne de détention (pour une cryptomonnaie précise) peut enregistrer plusieurs transactions, chaque transaction appartenant à une seule détention.
* Implémentation : dans la table **Trade**, holding\_id est NOT NULL et référencé par une FK → Wallet\_Holding(holding\_id).

**User → Preference : One-to-Many**

* Un utilisateur peut avoir plusieurs préférences, chaque préférence rattachée à un seul utilisateur.
* **Implémentation :** dans la table Preference, user\_id est NOT NULL et référencé par une FK → User(user\_id).

**User → User\_Learn : One-to-Many**

* Un utilisateur peut suivre plusieurs modules, chaque suivi stocké dans une ligne **User\_Learn**.
* Implémentation : dans la table **User\_Learn**, user\_id est NOT NULL et référencé par une FK → **User**(user\_id).

**Learn → User\_Learn : One-to-Many**

* Un module pédagogique peut être suivi par plusieurs utilisateurs, chaque suivi matérialisé dans **User\_Learn**.
* Implémentation : dans la table **User\_Learn**, learn\_id est NOT NULL et référencé par une FK → **Learn**(learn\_id).

**User ↔ Learn : Many-to-Many**

* Un utilisateur peut suivre plusieurs modules, et un module peut être suivi par plusieurs utilisateurs.
* Implémentation : une clé primaire propre user\_learn\_id , deux clés étrangères user\_id (→ User) et learn\_id (→ Learn) et les attributs métiers is\_completed et completed\_at pour suivre l’état de complétion.

**Price : Entité autonome**

* Stocke l’historique des cours sans relation directe en V1.
* **Implémentation :** pas de FK ; un index (crypto\_symbol, recorded\_at) optimise les recherches temporelles.

**Résumé des cardinalités (notation Merise) :**

* **Utilisateur (0..n) – (1..1) Wallet**
* **Wallet (1..1) – (0..n) Wallet\_Holding**
* **Wallet\_Holding (1..1) – (0..n) Trade**
* **Utilisateur (1..1) – (0..n) Preference**
* **Utilisateur (1..1) – (0..n) User\_Learn**
* **User\_Learn** **(0..n) - (1..1) Learn**
* **Price :** entité autonome sans relation directe en V1.

## g. Diagrammes UML

Dans le diagramme de cas d’utilisation, l’utilisateur est l’acteur principal et interagit avec le système BlockLumen. Les cas d’utilisation représentant les principales fonctionnalités et interactions que les utilisateurs peuvent avoir avec l’application sont les suivants :

**Consulter accueil :**

Permet à l’utilisateur de consulter la page d’accueil.

**Créer un compte** :  
 Permet à l’utilisateur de créer un compte sur la plateforme BlockLumen en fournissant un nom d’utilisateur, une adresse e-mail et un mot de passe.

**Se connecter** :  
 Permet à l’utilisateur de s’authentifier sur BlockLumen pour accéder aux fonctionnalités protégées.

**Gérer profil et préférences** :  
 Permet à l’utilisateur authentifié de consulter et modifier ses informations de profil ainsi que ses préférences (thème, notifications, etc.).

**Créer un portefeuille** :  
 Permet à l’utilisateur de générer un nouveau portefeuille virtuel avec un solde initial (10 000 $ par défaut).

**Lister mes portefeuilles** :  
 Permet à l’utilisateur de visualiser la liste de tous ses portefeuilles existants.

**Voir détail d’un portefeuille** :  
Permet à l’utilisateur de sélectionner un portefeuille pour afficher ses retenues (chaque crypto détenue, quantité, prix moyen) ainsi que la valeur calculée.

➢ Voir mes holdings : Affiche toutes les lignes de détention (`Wallet\_Holding`) associées à un portefeuille donné.

➢ Consulter cours actuel : Récupère et affiche les derniers prix des cryptomonnaies concernées à partir de la table `Price`.

**Effectuer un trade (achat/vente)** :  
Permet à l’utilisateur d’acheter ou de vendre une cryptomonnaie pour un portefeuille donné. Lorsqu’un trade est validé :  
 – Si **achat**, créer ou mettre à jour la ligne correspondante dans Wallet\_Holding (quantité + average\_price).  
 – Si **vente**, diminuer la quantité dans Wallet\_Holding, et supprimer la ligne si la quantité devient nulle.

➢ Mettre à jour/inserer un holding : Après le trade, mettre à jour la `quantity` et le `average\_price` dans `Wallet\_Holding`.

➢ Vérifier solde disponible : S’assurer que le portefeuille dispose d’assez de fonds fictifs pour réaliser l’opération.

➢ Calculer frais : Déterminer et appliquer les frais simulés à la transaction.

**Consulter historique des transactions** :  
Permet à l’utilisateur de voir la liste de tous ses trades (filtrable par portefeuille ou par cryptomonnaie), avec date, type (achat/vente), quantité, prix et frais.

**Suivre les modules pédagogiques (Learn)** :  
Permet à l’utilisateur de naviguer dans les modules d’apprentissage disponibles. Pour chaque module :  
– **Voir le détail du module** (titre, contenu)  
– **Marquer comme complété** : créer ou mettre à jour la ligne dans User\_Learn (is\_completed, completed\_at).

**Se déconnecter** :  
Permet à l’utilisateur de terminer la session en invalidant son token JWT.

### 1. Diagramme de cas d’utilisation UML :

Comme l’illustre le diagramme, l’application limite l’accès à la majorité des fonctionnalités aux seuls utilisateurs authentifiés. Il est donc nécessaire de créer un compte et de se connecter pour pouvoir utiliser les cas d’utilisation de premier niveau.

Une image contenant texte, diagramme, ligne, Parallèle

Le contenu généré par l’IA peut être incorrect."Cas d’utilisation sécurisés"

Une image contenant texte, diagramme, ligne, cercle

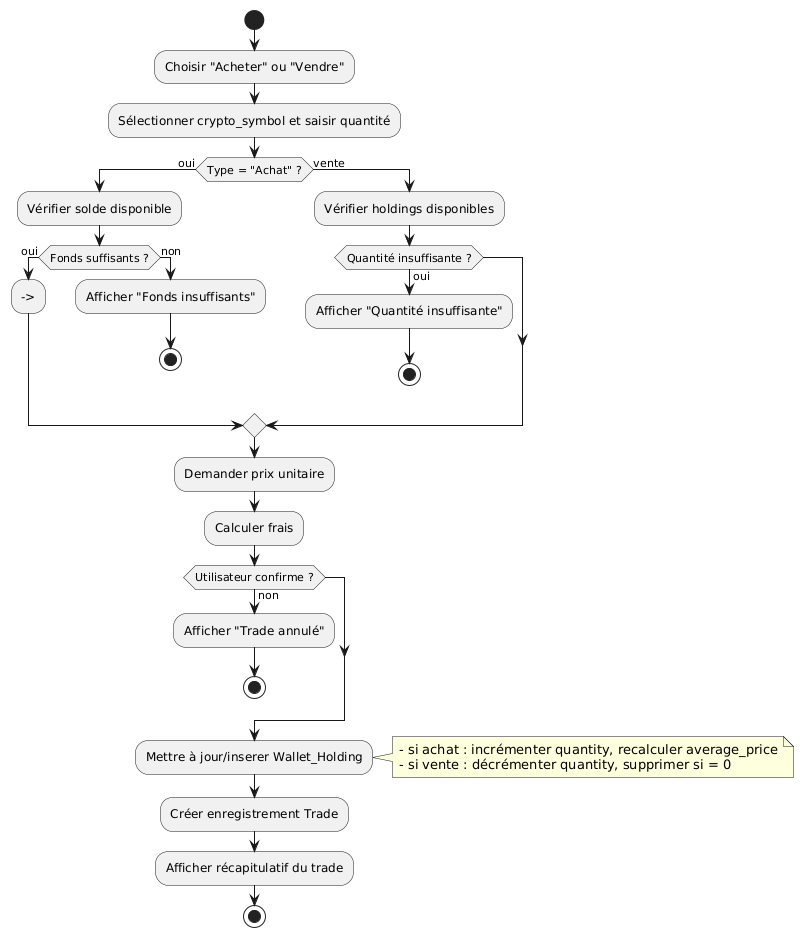
Le contenu généré par l’IA peut être incorrect."Cas d’utilisation publics"

### 2. Diagramme d’activités UML

Les diagrammes d’activités constituent un outil essentiel pour modéliser de manière claire et précise le fonctionnement interne et les différents flux métier de cette application.  
Ce diagramme illustre le processus complet lorsqu’un utilisateur réalise un échange : depuis le choix d’« achat » ou de « vente », jusqu’à la mise à jour du portefeuille et la création de l’enregistrement Trade.

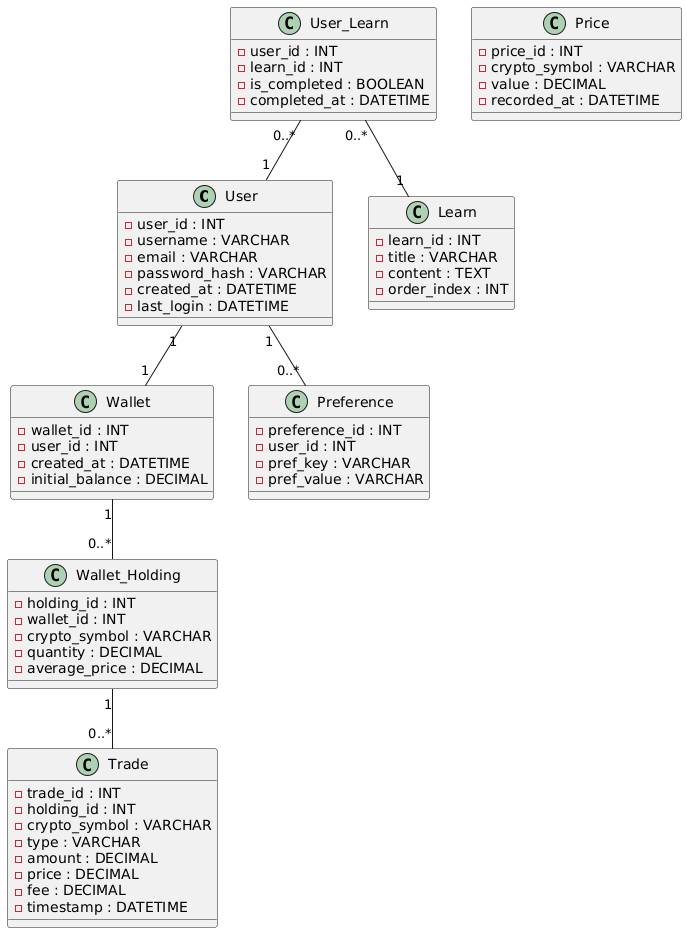
Chaque action est reliée par des transitions explicites, avec des nœuds de décision pour gérer les conditions d’entrée (vérification du solde pour un achat, disponibilité des holdings pour une vente) et des flux terminaux multi-sorties pour représenter les fins anticipées (affichage d’un message « fonds insuffisants » ou « quantité insuffisante »). Des nœuds de fusion garantissent que, quelles que soient les branches empruntées, le processus converge ensuite vers la saisie du prix unitaire, le calcul des frais et la confirmation utilisateur.

Enfin, l’usage de nœuds de jonction synchronise les sous-processus avant l’étape finale de mise à jour (insertion ou modification de Wallet\_Holding) et de création du Trade. Cette structuration permet de visualiser très rapidement les chemins critiques, d’identifier les points d’exception et de s’assurer que tous les scénarios sont pris en compte de façon normalisée.



### 3. Diagramme d’entités

A partir de la modélisation, j’ai développé un diagramme d’entités dans le cadre de la conception UML avec cardinalité et relations d’agrégation entre classes.



# 5. Environnement de développement

Avant de passer au développement, je valide le choix des technologies énoncées dans le Cahier des Charges. Je structure mon environnement autour de trois volets : Front-end, Back-end et Outils transverses.

**Front-end**

* **React 18 & React-Chart.js-2 / Chart.js**  
  J’utilise React 18 pour bénéficier de son rendu déclaratif et de ses Hooks, ce qui me permet de construire une interface modulable et réactive. React-Chart.js-2 s’intègre naturellement au flux React pour afficher des graphiques de cours temps réel, sans ralentir l’application.
* **Vite**  
  Vite assure un démarrage instantané et un hot-reload ultra-rapide grâce à ESBuild, réduisant significativement la latence entre le code et l’aperçu. Cette vitesse d’itération est essentielle pour affiner rapidement l’expérience utilisateur.
* **Tailwind CSS**  
  En misant sur un système de classes utilitaires, Tailwind CSS supprime l’écriture de styles « ad hoc » et assure une interface visuelle homogène. Chaque classe atomique correspond à une seule propriété CSS, ce qui accélère le prototypage et rend la maintenance des styles extrêmement simple.
* **TypeScript**  
  Le typage statique renforce la fiabilité du code en détectant les erreurs à la compilation, améliore l’autocomplétion et sert de documentation embarquée pour chaque composant.

**Back-end**

* **Node.js & Express**  
  Node.js me permet d’unifier le langage (TypeScript) côté serveur et client, tandis qu’Express, avec son système de middlewares, offre une base légère et extensible pour exposer notre API REST. Cette configuration est largement éprouvée pour les applications à microservices ou monolithiques.
* **MySQL**  
  Choisi pour sa robustesse transactionnelle et sa performance, MySQL répond aux besoins de cohérence du portefeuille et de l’historique des transactions. Ses capacités de réplication et de sauvegarde simplifient la montée en charge et la reprise d’activité.
* **Docker & Docker Compose**  
  La containerisation garantit un environnement isolé et identique pour l’API et la base de données, quel que soit le poste de développement ou de production. Docker Compose orchestre facilement l’ensemble de la stack (api, db) et assure un démarrage reproductible en un seul docker-compose up.

**Outils transverses**

* **Git & GitHub**  
  Git versionne le code avec précision, et GitHub facilite la collaboration via les pull requests, les revues de code et l’intégration continue. Ces pratiques sont indispensables pour garantir traçabilité et qualité.
* **Visual Studio Code**  
  VS Code, doté d’extensions TypeScript, Docker et ESLint, offre un débogueur intégré et un environnement unifié qui accélère le développement et la détection de bugs.
* **Postman**  
  Postman me permet de tester et documenter les endpoints REST de façon reproductible grâce aux collections partagées, préparant la base pour les tests automatisés (Jest /Supertest).
* **Figma**  
  Figma centralise la création de maquettes interactives et la librairie de composants UI, garantissant la cohérence entre le design et l’implémentation.
* **app.diagrams.net (Draw.io)**  
  Draw.io sert à modéliser rapidement le MCD, le MPD et l’architecture, et je stocke ces schémas dans le dépôt pour qu’ils restent synchronisés avec le code.

Pour garantir la reproductibilité, la cohérence et la portabilité de notre application **BlockLumen**, j’ai mis en place un environnement de développement structuré autour de deux axes principaux :

* + Initialisation des dossiers et des dépendances (frontend + backend)
  + Conteneurisation du Backend via Docker (Dockerfile et Docker Compose)

**En résumé** :

* **Docker** gère MySQL + le backend.
* **Vite** (via npm run dev) gère le frontend React/TypeScript localement.

Cette configuration garantit un hot-reload rapide pour le front, tout en isolant le backend et la base de données dans Docker.

### 1. Créer la structure de base du Frontend avec Vite + React + TypeScript

BlockLumen/

├── client/ # Contiendra le code de l’application React/TypeScript (Frontend)

└── server/ # Destiné au backend ( Node.js)

J’installe **npm** et **node** puis je lance la commande vite pour créer un projet React/Typescript dans le dossier client qui sera notre FrontEnd.

**Node.js** est la plateforme d’exécution JavaScript côté serveur, et **npm** est le gestionnaire de paquets associé. Sans **Node** ni **npm**, il est impossible d’installer les dépendances (React, Express, TypeORM, etc.) ou d’exécuter les scripts de démarrage.

**Vite** est un outil rapide pour créer des projets front-end modernes (React, Vue, Svelte…). Il génère une configuration optimisée pour le développement (hot-reload ultrarapide)

-$ npm create vite@latest . -- --template react-ts

Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.-$ npm install

-$ npm run dev

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

### 2. Installation des dépendances pour le Frontend et affichage page React

Redux Toolkit + React-Redux (gestion de l’état global)

* + npm i @reduxjs/toolkit react-redux

Axios (pour les requêtes HTTP) et react icons (iconographie)

* + npm i axios
  + npm i react-icons

React Router DOM (routage côté client) et Loader Spinner (chargement)

* + npm i react-router-dom
  + npm i react-loader-spinner

Tailwind CSS + PostCSS + Autoprefixer (styles utilitaires)

* + npm install -D tailwindcss postcss autoprefixer
  + npx tailwindcss init -p

Pour utiliser Tailwind, il faut l’initialiser dans le CSS global et indiquer à Vite où chercher.

Dans main.tsx j’importe App.css et dans App.css j’ajoute les 03 directives tailwind Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, Police, capture d’écran, Graphique

Le contenu généré par l’IA peut être incorrect.

Les dépendances sont installées et j’ai créé la structure de fichiers pour le Frontend :

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.assets : ressources statiques ( images, polices,… )

components : petits éléments d’interface réutilisables

constants : valeurs fixes ( couleurs, textes )

contexts : providers React pour partager les données

redux : store et slice pour état global

routers : configure des routes de navigation

screens : pages ou vues

services : appels API et logique externe

tools : fonctions utilitaires ( formatage, helpers )

On se rend sur **http://localhost:5173/** pour tester le bon fonctionnement du front :

La page React s’affiche sans erreur.

Une image contenant texte, capture d’écran, logiciel, multimédia

Le contenu généré par l’IA peut être incorrect.

### 3. Créer le BackEnd

#### a. Créer la structure des fichiers :

Je crée un dossier server/ a la racine du projet, et j’initialise un package.json :

* + Une image contenant texte, capture d’écran, logiciel

    Le contenu généré par l’IA peut être incorrect.npm init -y

controllers : héberge la logique métier

entities : contient les classes (ex : user.ts )

middlewares : Express ( JWT, gestion erreurs, … )

routes : définit les routes Express pour chaque entité

utils : tools et utilitaires ( logger, hashing)

index.ts : point d’entrée de l’Api

#### b. Ajouter les dépendances :

**Dépendances de production :**

Express : Framework web qui permet de créer des routes, intégrer middleware ( JSON, CORS, JWT ) et structurer l’API.

Dotenv : Gestion de variables d’environnement (.env).

mysql2: pilote MySQL pour Node.js

ORM TypeScript : mapping entre objets et base de donnée avec **sûreté des types**

JWT : création et vérification de token JWT

bcrypt : hash les mots de passe

* + npm install express dotenv mysql2 typeorm reflect-metadata jsonwebtoken bcrypt

**Dépendances de développement :**

TypeScript : compilateur

Ts-node : exécuter .ts sans compilation

Nodemon : redémarrage automatique du serveur en mode dev

@types/… : définitions de types pour Node, JWT, bcrypt

* + npm install -D typescript ts-node nodemon @types/node @types/express @types/jsonwebtoken @types/bcrypt

**Ajouter nos variables de scripts :**

Package.json :

"scripts": {

"dev": "nodemon", // démarre ts-node via nodemon

"build": "tsc", // compile le TypeScript en JavaScript

"start": "node dist/index.js" // Démarre l’API compilée (en production après "build")

},

"dev": "nodemon" : Nodemon démarre l'application et redémarre le serveur automatiquement dès qu'on sauvegarde des modifications dans un fichiers .ts

**Création des fichiers de configuration :**

* + tsconfig.json (compilation TypeScript )
  + nodemon.json (rechargement automatique en développement)
  + .env.example (modèle de variable d’environnement)

Chaque développeur copiera (cp .env.example .env) puis remplira .env avec ses vraies valeurs (jamais commitées)

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

**Vérification rapide du nodemon sans Docker :**

* + npm run dev

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

### 4. Mise en place de Docker - BackEnd

Pour l’environnement de développement côté backend j’utilise **Docker**.

Docker permet une isolation des dépendances, un montage de volumes et hot reload instantané ainsi qu’une sécurité renforcée étant donné que la base de données tourne dans son propre conteneur.

Tout d’abord on installe docker sur notra machine, on développe sur une OS Windows donc il faut installer Docker Desktop et WSL afin de simuler en environnement linux.

Enquite je crée et paramètre un **DockerFile** dans le dossier /server et un docker-compose.yml dans le dossier racine /BlockLumen.

#### a. DockerFile

Le **DockerFile** est un fichier de configuration qui me permet de construire une image. Son rôle principal est de garantir que, quelle que soit la machine hôte, le conteneur résultant contiendra exactement les mêmes composants et fonctionnera de manière identique.

Dans le cadre de **BlockLumen**, j’ai structuré le DockerFile en deux étapes distinctes : une étape de build (compilation) et une étape de production (exécution optimisée).

**Séparation des préoccupations build / production** : regrouper les dépendances de développement uniquement dans l’étape de build permet de créer une image finale légère, sans outils superflus. En production, on n’a pas besoin de compiler ou de redémarrer automatiquement le serveur à chaque modification de code.

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

#### b. DockerCompose - Backend

Docker Compose est un outil qui permet de définir et de lancer plusieurs conteneurs Docker à la fois. Il simplifie la configuration de l’ensemble de la stack (base de données, backend, etc.) en décrivant chaque service, ses volumes et ses réseaux en un seul endroit.

Pour l’utiliser, il suffit d’écrire un docker-compose.yml décrivant les services, puis de lancer docker-compose up --build, ce qui démarre automatiquement tous les conteneurs et gère leur interconnexion. Grâce à Docker Compose, on obtient un environnement reproductible et cohérent pour le développement comme pour la mise en production.

**Annexe 01 : Docker-compose.yml**

#### c. Création .env et test route

Je crée le vrai fichier .env avec la commande suivante : - cp .env.example .env

Copier .env.example en .env permet de disposer des variables d’environnement nécessaires sans versionner les valeurs sensibles dans Git.

Je modifie le fichier **index.ts** afin de tester une route et confirmer le bon fonctionnement

Ici on crée une route /health qui nous renvoie un message si tout fonctionne. Cette route se trouve dans notre index.Ts

**Annexe 02 : index.ts**

J’exécute la commande : docker-compose up –build et je teste la route de santé avec POSTMAN. On peut aussi aller directement sur l’url et le message apparaitra.

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

### 5. GitHub

Pour sauvegarder mon projet et chaque modification, j’ai installé l’extension Git dans VS Code et GitHub Desktop. J’ai créé un nouveau repository sur GitHub, puis dans VS Code j’ai exécuté git init et fait un “initial commit”. Ensuite, j’ai lié le dépôt local à celui sur GitHub. Cette démarche me permet de conserver l’historique des changements, de travailler en équipe grâce aux branches et aux pull requests, et de revenir à un état antérieur si nécessaire.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.J’ajoute un README.md afin de fournir une vue d'ensemble du projet, des instructions d'installation et de démarrage, ainsi que des informations sur les technologies utilisées.

Enfin, pour accéder à l’application complète en dev, il faut cloner le repository et ouvrir deux onglets/terminaux et exécuter les commandes suivantes :

* + cd server && docker compose up --build (lance MySQL + backend)
  + cd client && npm run dev (lance le front Vite).

# 6. Développement de l’application

## 1. Création du Backend

### a. Création base de données et Migrations

Avant de démarrer le développement, une configuration rigoureuse de la base de données est impérative. À cette fin, le fichier docker-compose.yml a été orchestré pour lancer le conteneur MySQL, lequel est configuré pour importer automatiquement deux scripts SQL distincts : l'un (BlockLumenBDD.sql) définissant la structure de notre Modèle Physique de Données (MPD) issu de la méthode Merise, l'autre (fixtures.sql) contenant les jeux de données initiaux.

Ces scripts sont montés via des volumes dans le répertoire d'initialisation /docker-entrypoint-initdb.d/ du conteneur, assurant leur exécution automatisée au démarrage du service MySQL. De surcroît, une condition de dépendance stricte (condition: service\_healthy) est implémentée dans docker-compose.yml pour garantir que le service backend ne démarre qu'une fois la base de données pleinement opérationnelle et initialisée. Cette approche garantit une base de données fonctionnelle et pré-remplie, indispensable pour les phases d'exploitation et de validation.

**Annexe 03 : Fixtures.sql – Jeux de données**

**Annexe 04 : BlockLumen.sql – Scrip de création bdd**

Afin de voir visuellement notre bdd on installe l’extension suivante dans VSCode :

* + MySQL - Database Management for MySQL/MariaDB, PostgreSQL, Redis and ElasticSearch.

Une fois l’extension configurée, il suffit de se connecter à la base de donnée en renseignant nos identifiants, puis nos tables apparaissent immédiatement dans l’explorateur de bases.

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

On en profite pour gérer les migrations de la base de données de manière cohérente, tant en développement qu’en production.

**En développement**, nous avons besoin d’installer ts-node afin de pouvoir exécuter la CLI TypeORM directement sur nos fichiers TypeScript. Concrètement, ts-node est un utilitaire qui s’intercale entre Node et TypeScript :

* + Il compile le code TypeScript en JavaScript « à la volée » (sans générer de .js sur le disque).
  + Il exécute immédiatement le résultat dans un environnement Node.

Cela permet à TypeORM de charger src/index.ts (qui exporte AppDataSource) sans erreur de typage, puis de comparer le schéma actuel avec les entités TS pour produire un fichier de migration ou l’appliquer.

**En production**, on préfère compiler d’abord tout le projet en JavaScript (avec npm run build), ce qui génère un dossier dist/. On pointe alors la CLI TypeORM vers les fichiers compilés :

* + npm run build
  + npx typeorm migration:generate dist/migrations/NouvelleMigration -d dist/index.js
  + npx typeorm migration:run -d dist/index.js

De cette façon, on ne dépend plus de ts-node en production et on s’assure que la base de données suit précisément les entités compilées.

### b . Création des entités - Backend

Pour structurer la partie back-end, j’ai d’abord modélisé chaque table de la base de données sous forme d’**entités** TypeORM en TypeScript, en utilisant les décorateurs @Entity, @Column, @PrimaryGeneratedColumn, etc., pour décrire précisément le schéma relationnel et les types de chaque champ.

Par exemple, l’entité Trade définit sa clé primaire auto-incrémentée, sa relation Many-To-One vers WalletHolding (avec suppression en cascade), ainsi que les colonnes crypto\_symbol, type, amount, price et fee avec leurs contraintes (longueur, précision, valeurs par défaut). La colonne timestamp est automatiquement peuplée à la création de l’enregistrement grâce au décorateur @CreateDateColumn.

Entité Trade :

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Le contenu généré par l’IA peut être incorrect.

### c. Création des contrôleurs ( CRUD basique / Backend )

Pour chaque entité de notre API, je définis un **contrôleur** TypeScript dans server/src/controllers/, organisé en classe. J’injecte le Repository TypeORM correspondant via le constructeur, ce qui me garantit d’accéder aux méthodes de manipulation de la base de données une fois la DataSource initialisée. Chacune des opérations standards—récupérer tous les enregistrements (getAll), récupérer un enregistrement par son identifiant (getOne), créer (create), mettre à jour (update) ou supprimer (remove)—est exposée sous forme de **propriété fléchée** (arrow function).

Cette approche me permet de préserver automatiquement le contexte this sans appel à bind(). Dans chaque méthode, j’appelle le repository adapté, je gère les cas d’absence de donnée (renvoi d’un 404) et les erreurs serveur (renvoi d’un 500), et j’écris directement la réponse avec res.json() ou res.status(), sans préfixer d’un return puisque l’envoi de la réponse termine le cycle HTTP.

Une fois les méthodes implémentées, je crée un **router Express** associé à ce contrôleur, dans lequel je mappe chaque route HTTP (GET, POST, PUT, DELETE) à la méthode correspondante. Les **middlewares** (authentification, validation des données d’entrée, gestion centralisée des erreurs) seront ajoutés sur ces routes à mesure que la complexité métier et les exigences de sécurité augmentent.

**Annexe 05 : UserController.ts**

### d. Création des routes et test routes - Backend

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.Chaque route se place dans **server/src/routes/\*.ts**. Pour chaque entité, on crée un fichier de routes qui importe le controller correspondant, initialise un Router(), puis associe chaque chemin HTTP à la méthode d’instance du controller. Pour l’instant, on n’applique pas encore de middlewares (authentification, validation des inputs, etc.) ; on les ajoutera ultérieurement lorsque la logique métier et la sécurité seront en place.

Je teste la route de création d’un user via postman, cela fonctionne. J’en profite pour tester les autres routes CRUD et aucun problème. **Annexe 06 : Postman**

## 2. Création du Frontend

### a. Contexte (Auth, Session)

J’ai structuré l’authentification en deux contextes distincts : **AuthContext** gère la connexion/déconnexion, stocke le token JWT et les informations utilisateur (et restaure la session au démarrage), tandis que **SessionContext** se charge uniquement de vérifier dès le chargement de l’application si une session est active (présence d’un token dans localStorage) et de mettre à jour l’état isAuthenticated.

Une image contenant texte, capture d’écran, logiciel, affichage

Le contenu généré par l’IA peut être incorrect.Les modifications consistent à centraliser la lecture/écriture du token dans AuthContext (avec injection automatique dans Axios) et à simplifier l’effet de restauration de session (utilisation d’un useEffect sans dépendances pour éviter les répétitions). L’objectif global est de garantir qu’à chaque redémarrage du navigateur, l’utilisateur reste connecté s’il dispose d’un token valide, et que l’application puisse rapidement savoir si la session est active pour protéger ou afficher les routes.

### b. Router (Public vs .Privé)

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.**AppRouter** décide dynamiquement quel routeur utiliser (en ligne ou hors-ligne) en se basant sur l’état d’authentification de l’utilisateur.  
**OfflineRouter** regroupe les écrans accessibles sans être connecté (page d’accueil hors-ligne, formulaire de connexion et d’inscription).  
**OnlineRouter** définit les routes protégées une fois l’utilisateur authentifié (tableau de bord, liste/détail des portefeuilles, historique des trades, modules pédagogiques, profil, etc.).  
Chaque routeur inclut également un composant PageError pour gérer les erreurs de navigation.

### c. AppConstant et ApiConstant

J’ai centralisé les adresses de mon API dans le fichier ApiConstant.ts pour pouvoir modifier une seule fois l’URL de base (grâce à API\_ROOT).

J’utilise ensuite API\_URL comme point d’entrée pour toutes mes requêtes, puis j’ai listé les différents endpoints (/users, /trades, /wallets, etc.) sous la forme de constantes réutilisables.

De cette façon, je n’ai plus besoin de retaper manuellement chaque URL dans mes services ou slices Redux. Par ailleurs, j’ai défini STORAGE\_KEY dans AppConstant.ts pour stocker et récupérer de manière cohérente la session utilisateur (token + profil) dans le localStorage.

AppConstant.tsx

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.ApiConstant.tsx

### d. Redux store

J’ai structuré le store Redux en deux slices principales (« wallets » et « trades ») pour séparer clairement la logique métier liée aux portefeuilles et aux transactions. Chaque slice contient un état de chargement (boolean), un tableau d’entités (wallets ou trades) et un éventuel message d’erreur, ainsi que des thunks asynchrones (fetchWallets et fetchTrades) qui se chargent de récupérer les données depuis l’API.

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.Les reducers gèrent automatiquement les états « pending », « fulfilled » et « rejected » pour afficher un loader ou stocker les données retournées par le serveur.

Les selectors permettent d’extraire facilement, dans un composant React, l’état global (liste, loading, error) de chaque slice sans avoir à connaître la structure interne du state. Ainsi, dans useSelector, on peut simplement faire selectAllWallets(state) ou selectTradesLoading(state) pour récupérer les informations nécessaires.

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Le contenu généré par l’IA peut être incorrect.

### e. Redux/slices/tradeSlice.ts & services/api.ts

Dans ce code, on n’utilise pas directement axios.get à chaque thunk : j’ai fait le choix de passer par un fichier services/api.ts qui exporte une instance Axios déjà configurée. Cela évite de dupliquer la configuration de l’URL de base (via API\_ROOT) et l’injection automatique du token dans l’en-tête Authorization.

**tradeSlice.ts :**

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

**Api.ts :**

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.

### f. Inscription et connexion des utilisateurs

Je crée les vues et les méthodes de la landing page, le login et register, et je teste leur fonctionnement

**Annexe 07 : Landing page, Login / Register**

### g. Charts et Consommation par API des cours actuel des cryptos

Utilisation de chart.js pour afficher les charts avec consommation API pour récupérer les cours actuels

Backend

* Mettre en place JWT + middlewares d’authentification
* Validation des inputs (express-validator ou Joi)
* Gestion des erreurs centralisée

### h. RGPD et Cookies

Je mets en place … RGPD … pour la protection des données te droit rétractation des données

## 3. Securité

Middleware jwt

Protection des routes : Permettre uniquement aux utilisateurs authentifiés d’appeler /wallets, /trades, etc.

Pour l’intégrité et la confidentialité des données, j'ai intégré un middleware d’authentification/autorisation JWT et des guards d’autorisation pour chaque route.

Par ailleurs, au-delà du cast des paramètres URL avec parseInt(), j’ai défini et appliqué des schémas de validation et de sanitisation des corps de requête (via class-validator/express-validator), et mis en place un rate-limiter pour prévenir les envois malveillants ou excessifs. Enfin, j'ai mis en place des audits de sécurité automatisés en utilisant les règles de sécurité ESLint.

# 7. Conclusion

## a. Remerciements

Je souhaite avant tout remercier l’IDEM de m’avoir offert cette opportunité d’apprentissage continue et stimulante : les professeurs et intervenants ont été des piliers indispensables dans mon évolution, m’apportant conseils et éclairages au moment où j’en avais le plus besoin.

Je tiens également à souligner la qualité de l’encadrement pédagogique, toujours attentif et disponible tout au long de ce projet. Un grand merci à l’équipe de REVIMPORT pour son accueil et son soutien durant mon alternance.

Enfin, je remercie ma famille et mes amis pour leur patience et leurs encouragements constants, qui m’ont permis de mener à bien ce mémoire et de franchir chaque étape avec sérénité.

b. Axes d’amélioration  
– **Renforcement de la sécurité** : intégrer un flow complet de rafraîchissement de token (refresh tokens), renforcer la validation des données avec express-validator ou Joi, et mettre en place une politique de CORS stricte en production.  
– **Tests et CI/CD** : écrire des tests unitaires et d’intégration (Jest, Supertest) pour chaque couche (entités, services, contrôleurs) et configurer un pipeline d’intégration continue (GitHub Actions, GitLab CI) pour automatiser la linting, le build, les migrations et les déploiements.  
– **Scalabilité et optimisation** : mettre en cache les requêtes de prix (Redis), découpler le service de collecte de données (microservice dédié), et envisager l’utilisation de WebSockets (Socket.IO) pour des mises à jour en temps réel du portefeuille.  
– **Expérience utilisateur** : enrichir l’interface avec des notifications push pour les changements de cours, améliorer l’accessibilité et proposer un mode sombre/thème dynamique.

c. Retour d’expérience  
 L’année écoulée a marqué une véritable étape dans ma montée en compétences techniques et organisationnelles. Mon immersion chez REVIMPORT, puis la réalisation du projet BlockLumen en alternance, m’ont confronté à des exigences élevées et m’ont permis de grandir professionnellement à chaque nouvelle fonctionnalité implémentée. La construction de cette plateforme de simulation de trading, de la modélisation Merise jusqu’à l’intégration continue via Docke m’a démontré que je suis capable de mener à bien des projets complexes. Fort de ce succès, je suis désormais résolu à poursuivre mon parcours académique en master, avec l’ambition de devenir chef de projet.

# Annexes

Une image contenant texte, capture d’écran, menu

Le contenu généré par l’IA peut être incorrect.Annexe 01 : docker-compose.yml

Annexe 02 : index.ts

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Le contenu généré par l’IA peut être incorrect.

Annexe 03 : Fixture.sql

Une image contenant texte, capture d’écran

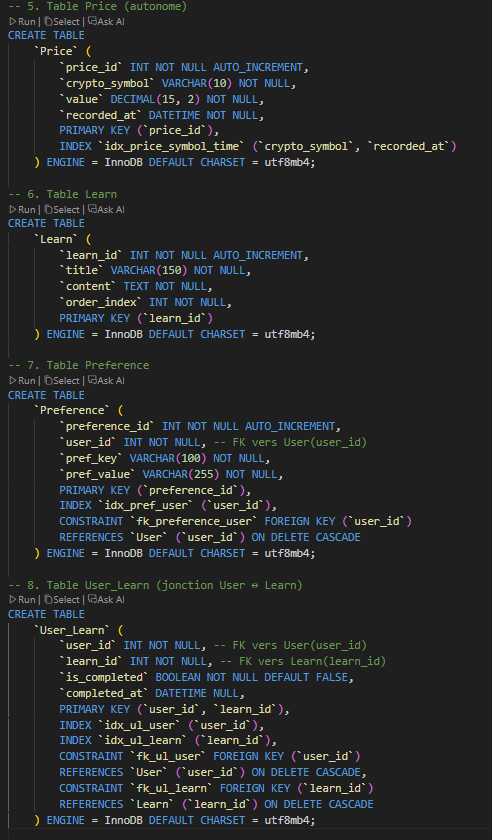
Le contenu généré par l’IA peut être incorrect.

Annexe 04 : BlockLumen.sql – Script Base de donnée

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.

Annexe 04 : BlockLumen.sql – Script Base de donnée



Annexe 05 : UserController.ts

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Le contenu généré par l’IA peut être incorrect.

Annexe 06 : Test route créer User avec Postman

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Annexe 07 : Landing page, Login, Register

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.