TechReuse Market

Application web de revente d'appareil électronique

DOSSIER PROJET

Développeur Web et Web Mobile

Assi Melvin

2025 – Assofac DWWM6- Formateur SOUMARE

Assofac îledeFrance

Remerciements

Je tiens à exprimer ma sincère gratitude à toutes les personnes qui m'ont soutenu tout au long de cette formation.

Un grand merci à mon formateur, Monsieur Demba Soumare, pour la confiance qu'il m'a accordée. Grâce à lui, j'ai pu bénéficier d'une expérience enrichissante lors de mon année. Travailler avec des technologies telles que React, Node.js et Express m'a permis de renforcer mes compétences pratiques dans un environnement professionnel.

Je tiens également à remercier mes camarades de classe pour leur esprit d'entraide sans lesquels je n'aurais pas pu m'investir pleinement dans cette formation.

À vous tous, un immense merci pour votre contribution précieuse à cet accomplissement.

Table des matières

Introduction

Dans le cadre de la formation Développeur Web et Web Mobile (DWWM), il nous a été demandé de concevoir et de développer une application web complète répondant à un besoin réel, tout en appliquant les bonnes pratiques de conception, de développement et de sécurité vues au cours de la formation.

Le projet TechReuseMarket s'inscrit dans une démarche à la fois technologique et écologique. Il s'agit d'une marketplace web dédiée à la vente et à l'achat de matériel informatique d'occasion. L'objectif est de favoriser le réemploi des équipements numériques tout en offrant une plateforme sécurisée, intuitive et performante aux utilisateurs (acheteurs, vendeurs et administrateurs).

Cette application permet aux utilisateurs :

- D'acheter ou vendre des produits informatiques reconditionnés ou d'occasion,
- De consulter les fiches de produits détaillées,
- De gérer un panier d'achat et de réaliser des paiements en ligne sécurisés via Stripe,
- Et aux administrateurs de superviser les annonces, transactions et comptes utilisateurs.

Le développement de ce projet a permis de mobiliser l'ensemble des compétences techniques du référentiel DWWM :

- Front-end avec React, TypeScript et Vite pour l'interface utilisateur,
- Back-end avec Node.js, Express et PostgreSQL pour la gestion des données et des API,

- Sécurité avec JWT, Helmet, Csurf, et la mise en place d'un environnement HTTPS conforme aux exigences RGPD. L'environnement de développement a été conteneurisé avec Docker et le code géré via GitHub.

Ce projet a été réalisé selon une méthodologie Agile, permettant une progression itérative, une gestion efficace des priorités et une adaptation continue aux besoins. La solution finale se veut modulaire, évolutive et maintenable, afin de pouvoir accueillir de nouvelles fonctionnalités à long terme.

Ainsi, TechReuseMarket illustre parfaitement la capacité à concevoir, développer et sécuriser une application web complète, en respectant les standards professionnels du développement web moderne.

Contexte et Objectifs du Projet

1. Contexte général

Dans un contexte où la transition écologique devient une priorité, la revalorisation du matériel informatique s'impose comme une solution durable pour réduire les déchets électroniques.
Le projet TechReuse Market s'inscrit dans cette démarche en proposant une marketplace dédiée à l'achat et à la vente de matériel informatique d'occasion.

Contrairement aux plateformes généralistes telles qu'eBay ou Leboncoin, TechReuse Market met l'accent sur :

- la sécurité des transactions,
- la traçabilité des produits,
- et la simplicité d'utilisation pour les acheteurs comme pour les vendeurs.

Ce projet a été conçu dans le cadre de la formation Développeur Web et Web Mobile (DWWM), dans le but de mettre en pratique l'ensemble des compétences du référentiel : conception, développement front-end et back-end, sécurité, gestion de base de données et déploiement.

2. Objectifs du projet

Le principal objectif est de développer une application web complète et sécurisée permettant :

- à des vendeurs de publier leurs produits informatiques reconditionnés ou d'occasion,
- à des acheteurs d'accéder à un catalogue clair, filtrable et ergonomique,
- et à un administrateur de superviser les utilisateurs, transactions et avis.

L'application doit :

- offrir une expérience fluide et responsive sur tous les appareils ;
- garantir la sécurité des données et des paiements via l'intégration de Stripe ;
- et respecter les normes RGPD en vigueur.

3. Portée du projet

Le projet couvre la totalité du cycle de développement :

- Analyse et conception (MCD/MLD/MPD, user stories, charte graphique) ;
- Développement front-end avec React, TypeScript et Vite ;
- Développement back-end avec Node.js, Express et PostgreSQL ;
- Mise en place des paiements via Stripe Checkout ;
- Déploiement et tests de performance (Postman, Lighthouse, sécurité).

Cette approche full-stack permet de démontrer la maîtrise des compétences techniques et méthodologiques attendues d'un développeur web complet.

Cahier des Charges

1. Présentation générale du projet

TechReuseMarket est une application web full-stack développée dans le cadre de la formation Développeur Web et Web Mobile (DWWM).
Elle vise à créer une marketplace éco-responsable et sécurisée dédiée à la vente et à l'achat de matériel informatique d'occasion ou reconditionné.
L'objectif principal est de favoriser le réemploi des équipements numériques tout en garantissant :

- une expérience utilisateur fluide et responsive,
- la sécurité des paiements et des données (HTTPS, JWT, CSRF, RGPD),
- et une gestion efficace des annonces, utilisateurs et transactions.

Le projet repose sur une architecture React (frontend) et Node.js/Express (backend), avec une base PostgreSQL.
L'ensemble est conteneurisé avec Docker, et les paiements sont gérés via Stripe Checkout.

2. Objectifs fonctionnels et techniques

Les objectifs du projet sont à la fois fonctionnels, techniques et écologiques :

- Connecter acheteurs, vendeurs et administrateurs sur une plateforme intuitive.
- Promouvoir le réemploi du matériel électronique pour réduire les déchets.
- Offrir un parcours utilisateur fluide grâce à une interface moderne.
- Gérer centralement les annonces, utilisateurs, commandes et transactions.
- Garantir la sécurité complète des paiements et des données.

3. Besoins fonctionnels

1. Page d'accueil : Produits phares, catégories, promotions, avis.
2. Catalogue : Filtres (catégorie, prix, état), recherche.
3. Achat/Vente : Panier, commande, publication d'annonces.
4. Espace Admin : Gestion des comptes, annonces, transactions.
5. Connexion et Gestion de Compte.
6. Contact : Formulaire avec reCAPTCHA.

4. Acteurs du système

Acteur	Rôle
Acheteur	Recherche, achète et évalue les produits
Vendeur	Publie des annonces, gère ses ventes
Administrateur	Modère les contenus, gère les utilisateurs et suit les statistiques
Visiteur	Parcoure le catalogue sans inscription

5. Besoins fonctionnels

Les besoins fonctionnels sont traduits sous forme de User Stories (US) selon la méthode Agile.

- US1 : Présenter la marketplace avec produits phares, catégories et avis.
- US2 : Menu de navigation clair (accueil, catalogue, contact).
- US3 : Catalogue avec filtres et recherche.
- US4 : Fiche produit détaillée avec avis.
- US5 : Gestion du panier et paiement sécurisé Stripe.
- US6 : Espace vendeur pour publier ses annonces.
- US7 : Tableau de bord administrateur.
- US8 : Connexion/inscription avec validation et reCAPTCHA.
- US9 : Formulaire de contact.
- US10 : Statistiques des ventes et avis.

6. Contraintes techniques

- Front-end : React, Vite, TypeScript.
- Back-end : Node.js, Express.
- Bases de données : PostgreSQL (principale), Firestore (avis, contacts, photos).
- Outils : Docker, GitHub, Postman, Figma, Trello.
- Pas d'installation de packages externes sans vérification de sécurité.

7. Exigences de sécurité

- HTTPS obligatoire pour toutes les communications.
- Mots de passe hachés avec Argon2id et tokens JWT signés.
- Protection contre XSS

- Rate limiting pour éviter les attaques par force brute.
- Rôles et permissions (admin, vendeur, acheteur).
- Fichiers sensibles (.env) protégés et non versionnés.
- Conformité RGPD : consentement cookies, droit à l'oubli.

8. Livrables attendus

- Application web complète et fonctionnelle.
- Code source propre, versionné sur GitHub.
- Base de données PostgreSQL.
- Documentation technique et guide d'installation.
- Maquettes, diagrammes Merise et UML.
- Rapport de projet (40-60 pages).
- Captures d'écran.

Compétences du Référentiel Couvertes

Dans le projet TechReuseMarket, j'ai développé une application web full-stack en appliquant les compétences clés de ma formation :

1. Développer la partie front-end d'une application web ou web mobile sécurisée

- Installation et configuration de l'environnement de travail : J'ai configuré un environnement cohérent avec Docker, Docker Compose et Visual Studio Code, tout en automatisant l'intégration et le déploiement continu grâce à GitHub Actions.
- Maquettage des interfaces utilisateur : J'ai réalisé des maquettes interactives avec Figma pour valider rapidement les interfaces utilisateur et obtenir des retours itératifs.
- Création d'interfaces utilisateur statiques : Les pages statiques ont été développées en Styled css.
- Développement de la partie dynamique des interfaces utilisateur : J'ai ajouté des fonctionnalités interactives avec React et TypeScript.

2. Développer la partie back-end d'une application web ou web mobile sécurisée

- Mise en place d'une base de données relationnelle : J'ai modélisé la base de données avec l'approche Merise, produisant des diagrammes clairs et un dictionnaire de données détaillé.
- Développement des composants d'accès aux données SQL et NoSQL : J'ai utilisé PostgreSQL pour les données relationnelles et Firebase pour les données non structurées, assurant une gestion des données flexible et performante.
- Développement des composants métier côté serveur : Structuration du back-end avec Node.js/Express et Sequelize pour l'accès aux données.
- Documentation du déploiement : J'ai produit une documentation détaillée pour le déploiement via Docker, Docker Compose, garantissant un déploiement sûr.

3. Résumé

Le projet couvre 100 % des compétences DWWM : full-stack, sécurité, gestion de données, déploiement.

Outils et technologies utilisés

Outils de Gestion du Projet et de Conception

Trello a été utilisé pour suivre l'avancement des tâches et assurer une gestion claire des priorités et des échéances.

La gestion du code source s'est faite via Git et GitHub, permettant de centraliser les versions et de faciliter l'intégration continue.

Figma a servi à la création des maquettes et des prototypes de l'interface utilisateur, assurant ainsi une validation rapide des designs.

Looping est logiciel qui de structurer la db via la méthode Mérise.

Configuration et Développement Technique

Visual Studio Code comme IDE principal pour sa flexibilité et ses extensions compatibles avec les langages utilisés.

Docker et Docker Compose ont permis de conteneuriser et d'isoler les services, offrant un environnement portable pour la db , le backend et la frontend.

Les dépendances frontend ont été gérées avec npm, tandis que Vite a permis d'optimiser les fichiers CSS et JavaScript.

PostgreSQL avec PgAdmin pour la gestion des données relationnelles

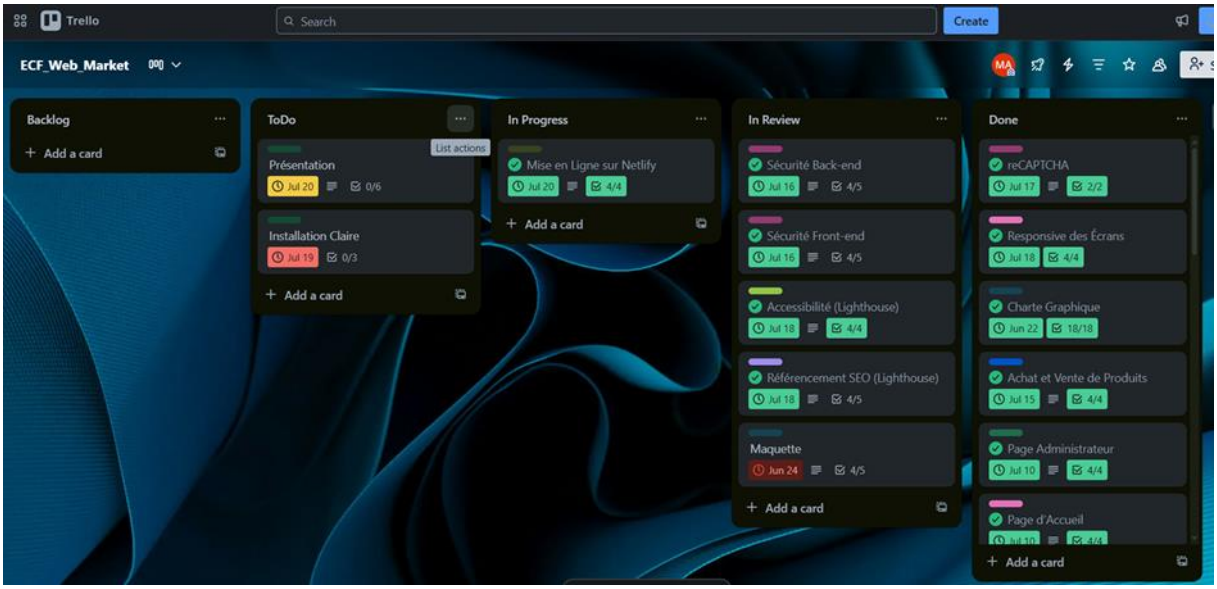
Firebase pour les données non structurées (avis, contacts et photos)

Postman pour tester et valider les API.

Gestion de Projet

1. Méthode Agile avec Trello

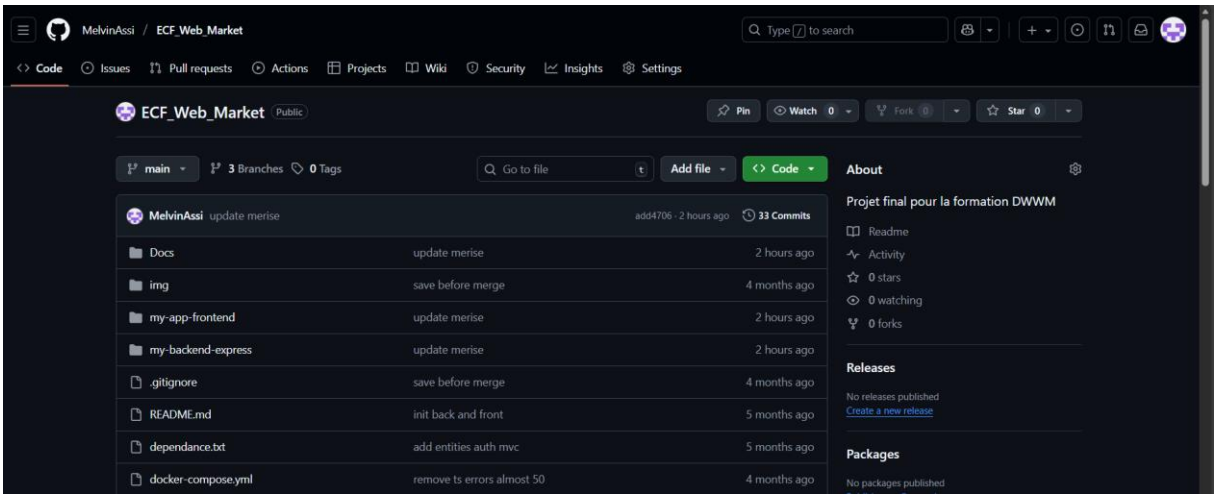
J'ai utilisé Trello pour organiser et suivre le projet en appliquant une méthode Agile. Les tâches étaient organisées en colonnes (« To Do », « In Progress », « In Review », « Done ») et étiquetées par type, avec des descriptions et une checklist. Cela a permis un suivi clair et structuré des sprints.



- Sprint 1 (18-24/06 2025) : Analyse
- Sprint 2 (25 /06-01/07 2025) : Initialisation
- Sprint 3 (2-8/07 2025) : Développement back-end
- Sprint 4 (9-15/07 2025): Développement front end/back-end
- Demi-Sprint 5 (16-20/07 2025) : Sécurité, optimisation, déploiement, présentation

2. Gestion du code source

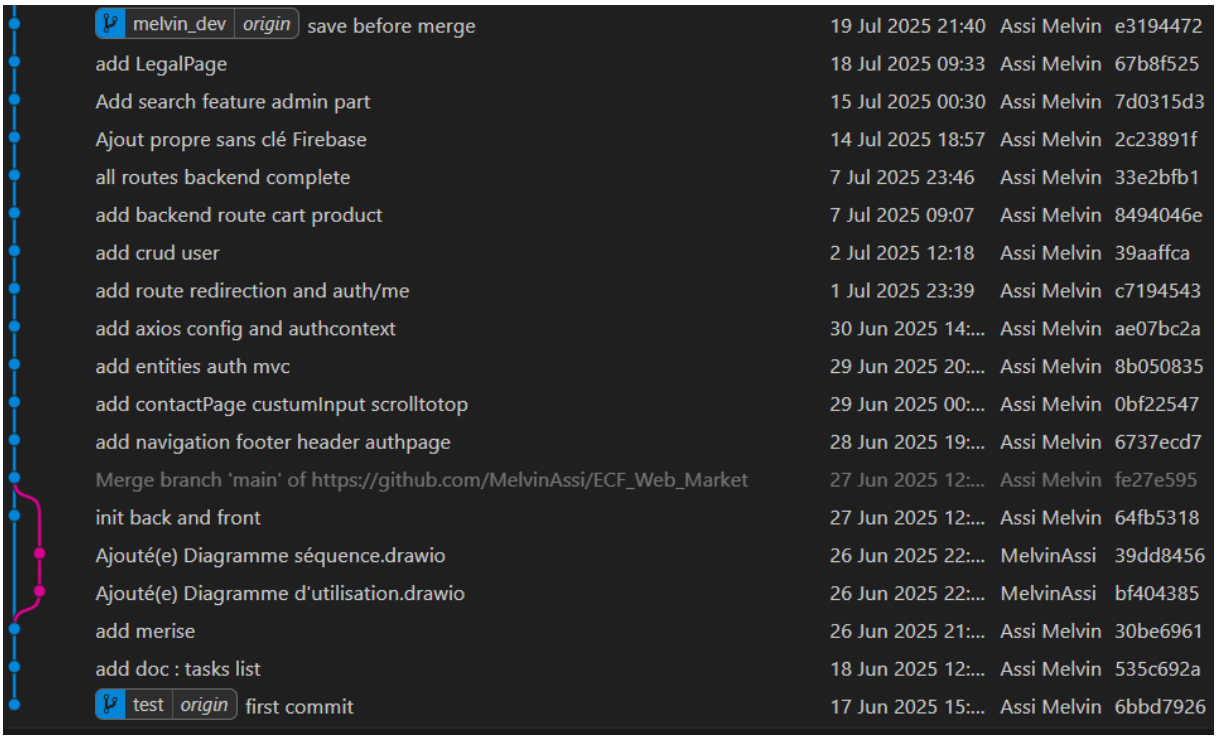
J'ai utilisé Git et GitHub pour gérer le code source, en créant plusieurs branches pour chaque fonctionnalité. Une fois les fonctionnalités testées et validées, je les fusionne avec la branche principale (main) pour garantir sa stabilité. J'ai également utilisé des tags pour marquer les versions stables. Un fichier README.md a été ajouté pour documenter l'utilisation du projet.



Commandes Git courantes :

- git checkout -b [nom_branche] : Créer et se déplacer vers une nouvelle branche.
- git merge [nom_branche] : Fusionner une branche dans la branche principale.
- git status : Vérifier l'état des modifications.
- git add . : Ajouter des fichiers modifiés à l'index.
- git commit -m "[message]" : Valider les modifications avec un message descriptif.
- git push : Pousser les modifications vers GitHub.
- git tag [nom_tag] : Marquer une version stable du projet.

Voici le Git graph qui montre quelques commits effectués lors du projet :



Conception de la base de données

1. Les règles de gestion

Pour la création d'une database il faut réfléchir à la structure pour avoir quelque chose de cohérent avec le cahier des charges . Voici les règles de gestion établis :

1. Un **User** est identifié par un identifiant unique (`id_user`) dans la base de données.

2. Chaque User possède une adresse e-mail unique, valide et obligatoire dans la table `user`.

3. Le mot de passe est "obligatoire" pour les connections avec email, haché avec Argon2id, et stocké dans la table `user`.

4. Chaque User doit fournir un nom (name), un prénom (firstname), une adresse (adress), et un numéro de téléphone (phone) obligatoires lors de l'inscription.

5. La date de création d'un compte est enregistrée automatiquement dans la table `user`.

6. Les comptes peuvent être désactivés par un Admin via le champ `is_active` dans la table `user`.

7. Les sessions sont gérées avec un champ `last_activity` pour suivre l'inactivité (expiration après 30 minutes).

8. Le rôle d'un User est défini comme `BUYER`, `SELLER` ou `ADMIN` dans la table `user`.

9. Un **User** peut se connecter avec un compte exterieur et sera lié a la table table `user`.

Gestion des Produits et Annonces

8. Un **Product** est identifié par un identifiant unique (`id_product`) et lié à une **Category**.

9. Chaque Product possède un nom, une description, un prix, un état (`NEW`, `GOOD`, `USED`), un statut de vérification (`UNDER_VERIFICATION`, `RECONDITIONED`, `READY_TO_SELL`, `REJECTED`), et des images.

10. Une **Listing** est créée par un Seller et liée à un seul Product dans la base.

11. Les Products soumis via une Listing passent par une vérification obligatoire par l'entreprise avant d'être mis en vente.

12. Les Products nécessitant un reconditionnement (ex. : changement de batterie pour un téléphone) sont marqués comme `RECONDITIONED` après traitement.

13. Les Products non conformes (ex. : chargeur cassé) sont marqués comme `REJECTED` et ne peuvent pas être mis en vente.

14. Seuls les Products avec le statut `READY_TO_SELL` sont visibles dans le catalogue public.

15. Les Listings ont un statut (`PENDING`, `ONLINE`, `DELETED`) géré dans la table `listing`.

Gestion des Achats et Commandes

16. Un **Buyer** peut créer un **Cart** lié à son `id_user` dans la base.

17. Une **Order** est identifiée par un `id_order`, liée à un Buyer, et contient plusieurs Products via une table d'association.

18. Une **Transaction** relie une Order, un Buyer et un Seller, avec un montant et un statut (`PENDING`, `VALIDATED`, `CANCELLED`).

19. La date et le montant total d'une Order sont enregistrés dans la table `order`.

Gestion des Avis

20. Un **Review** est laissé par un Buyer sur un Product ou un Seller, avec un commentaire, une note (1 à 5) et une date.

21. Les Reviews sont stockés dans la table `review` et peuvent être validés ou supprimés par un Admin.

Formulaire de Contact

22. Un message de contact est stocké dans la table `contact` avec un titre, une description, un e-mail et une date d'envoi.

23. Les messages ont un statut (`PENDING`, `READ`, `RESPONDED`) et peuvent être liés à un Admin qui les traite.

24. Les Statistics** sont stockées dans la table `statistic`, avec le nombre de produits vendus, le chiffre d'affaires et une période.

25. Les Statistics peuvent être filtrées par catégorie via un champ `category_id`.

2. Dictionnaire de Données

Suite aux règles établis, la création du dictionnaire peut se faire. Il a donc fallu recenser toutes les variables utiles pour chaque table avec le nom, le type, les contraintes et la description.

Voici un exemple avec la table Users :

Nom du champ	Type	Contraintes	Description
id_user	UUID / SERIAL	PK, unique, non nul	Identifiant unique de l'utilisateur
email	VARCHAR(100)	unique, non nul, format email	Adresse e-mail de l'utilisateur
password	TEXT	haché (bcrypt/Argon2id)	Mot de passe haché
name	VARCHAR(100)	non nul	Nom de l'utilisateur
firstname	VARCHAR(100)	non nul	Prénom de l'utilisateur
adress	VARCHAR(100)	non nul	Adresse de l'utilisateur
phone	VARCHAR(20)	non nul, format téléphone (10 chiffres)	Numéro de téléphone de l'utilisateur
role	ENUM	valeurs : 'BUYER', 'SELLER', 'ADMIN'	Rôle de l'utilisateur
created_at	TIMESTAMP	auto-généré	Date de création du compte
is_active	BOOLEAN	défaut true	Statut actif ou désactivé
last_activity	TIMESTAMP	optionnel	Dernière activité pour gestion des sessions
stripe_account_id	VARCHAR(100)	unique, optionnel	ID Stripe du compte vendeur

(Voir voir l'annexe pour les 13 tableaux complets)

3. Modélisation MERISE

- MCD : Entités + relations

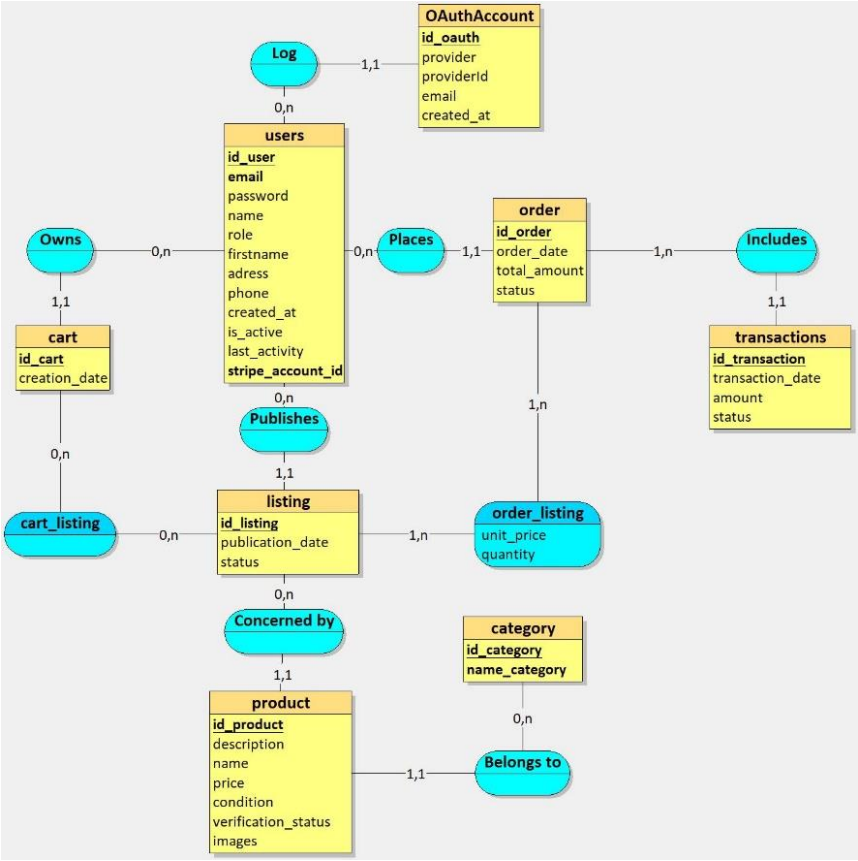
Le MCD représente les entités et leurs relations avec les cardinalités. Voici une description textuelle

Entités et Relations

- User (id_user, email, password, name, firstname, adress, phone, role, created_at, is_active, last_activity)
    - Publishes (0,N) -> Listing (1,1): Un Seller peut publier plusieurs Listings, une Listing est publiée par un seul Seller.
    - Places (0,N) -> Order (1,1): Un Buyer peut passer plusieurs Orders, une Order est passée par un seul Buyer.
    - Owns (0,N) -> Cart (1,1): Un Buyer peut posséder un Cart, un Cart appartient à un seul Buyer.
    - Leaves (0,N) -> Review (1,1): Un Buyer peut laisser plusieurs Reviews, un Review est laissé par un seul Buyer.
    - Handles (0,N) -> Contact (0,1): Un Admin peut traiter plusieurs messages, un message est traité par au plus un Admin.
    - log User (1,N) —> (1,1) OAuthAccount: Un user peut se connecter avec un compte exterieur
- Product (id_product, name, description, price, condition, verification_status, images, category_id)
    - Belongs to (1,1) -> Category (0,N): Un Product appartient à une seule Category, une Category peut contenir plusieurs Products.
    - Concerned by (1,1) -> Listing (1,1): Une Listing concerne un seul Product, un Product est lié à une seule Listing.
    - Evaluated by (0,N) -> Review (0,1): Un Product peut recevoir plusieurs Reviews, un Review peut concerner un Product.
- Listing (id_listing, publication_date, status, seller_id, product_id)
    - Contained in (0,N) -> Cart_Listing (1,N): Un Listing peut être dans plusieurs Carts, un Cart peut contenir plusieurs Listing.
- Category (id_category, name_category)
- Order (id_order, order_date, total_amount, status, buyer_id)
    - Includes (1,1) -> Transaction (1,N): Une Order est liée à une ou plusieurs Transactions.
- Cart (id_cart, buyer_id, creation_date)
    - Contains (1,1) -> Cart_Listing (0,N): Un Cart peut contenir plusieurs Listings via la table d'association.

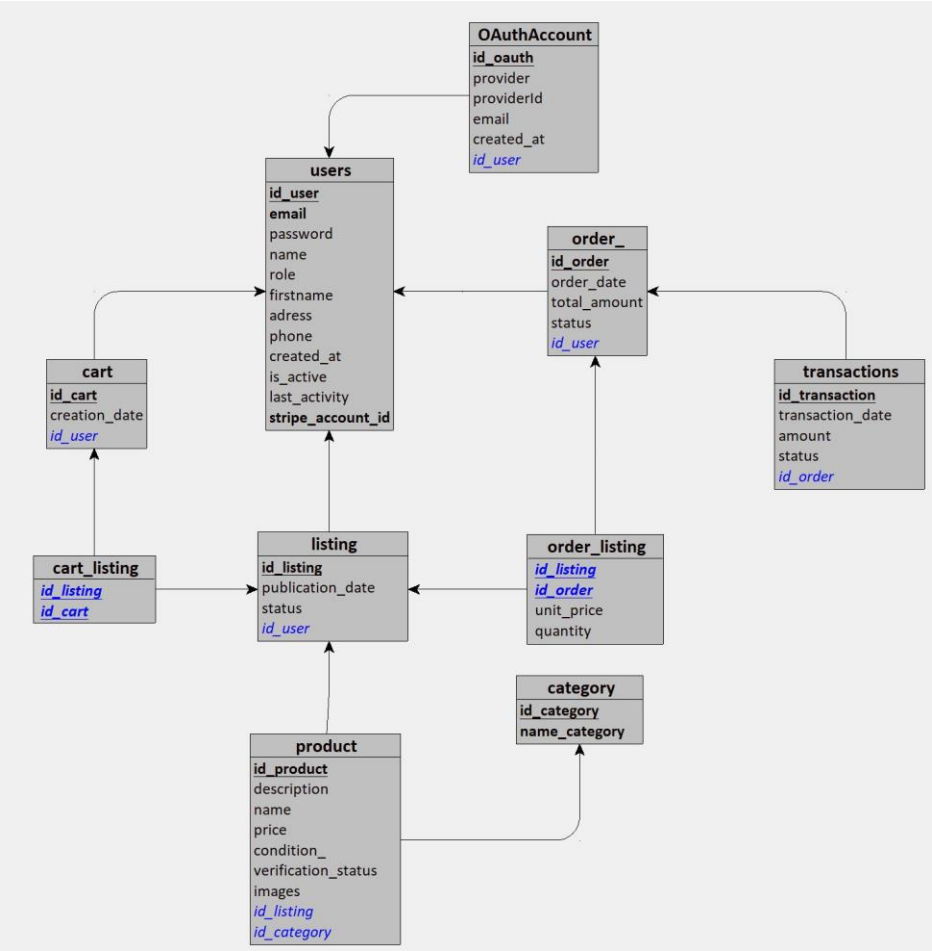
- Cart_Listing (id_cart, id_listing, quantity)
- Review (id_review, comment, rating, review_date, buyer_id, product_id, seller_id)
- Transaction (id_transaction, transaction_date, amount, status, buyer_id, , order_id)
- Contact (id_contact, title, description, email, sent_date, status, admin_id)
- Statistic (id_statistic, products_sold, revenue, period, category_id)
- Order_Listing (order_id, listing_id, quantity, unit_price)
    - Represents (1,N) -> Order (1,1): Une Order contient plusieurs listings, chaque ligne d'Order_Listing est liée à une seule commande.
    - Concerns (1,N) -> Listing (1,1): Un Listing peut apparaître dans plusieurs commandes, chaque ligne de Order_Listing référence un seul Listing.

Voici le résultat réalisé sur Looping montrant de manière schématique les entités et les relations qui les lies.



- MLD : Tables + FK

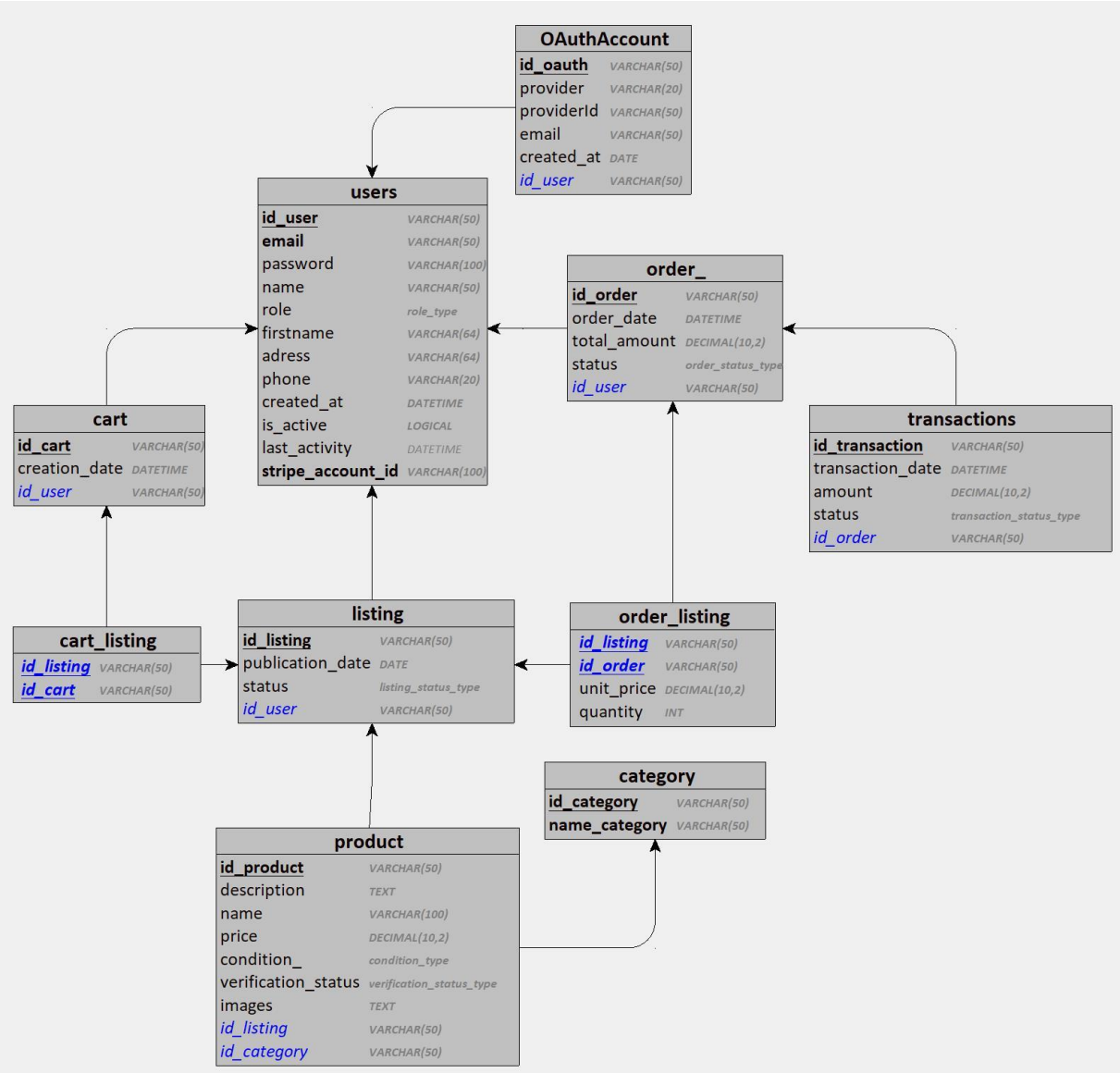
  Le MLD traduit le MCD en tables relationnelles, en respectant les contraintes et les relations.Sur looping les entités deviennent des tables et les clefs étrangères apparaissent.



- MPD : PostgreSQL + ENUM + CHECK

  Le MPD est une implémentation spécifique pour PostgreSQL, incluant les types de données, contraintes et index.

La version finale des tables avec les types donnés ajoutés



Donc il reste à traduire chaque en langage SQL pour pouvoir les crée dans la db Postgre. Looping a un outil pour le faire mais certaine variable sont aàchanger car la traduction est faite pour MySql et non pour Postgre.

Voici un exemple pour la table Users :

```sql
CREATE TABLE users (
    id_user UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(100) NOT NULL UNIQUE,
    password TEXT ,
    name VARCHAR(100) NOT NULL,
    firstname VARCHAR(100) NOT NULL,
    adress VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL CHECK (phone ~ '^[0-9]{10}$'),
    stripe_account_id VARCHAR(100) UNIQUE,
    role role_type NOT NULL DEFAULT 'BUYER',
    created_at TIMESTAMP DEFAULT NOW(),
    is_active BOOLEAN DEFAULT TRUE,
    last_activity TIMESTAMP
);
```

Le reste des tables est à retrouver dans l'annexe. Dans un premier j'ai utilisé cette méthode pour ma db. Mais j'y apporterai de changement car la db sera automatisé (et sécurisé) dans la suite du projet grâce à Sequilize.

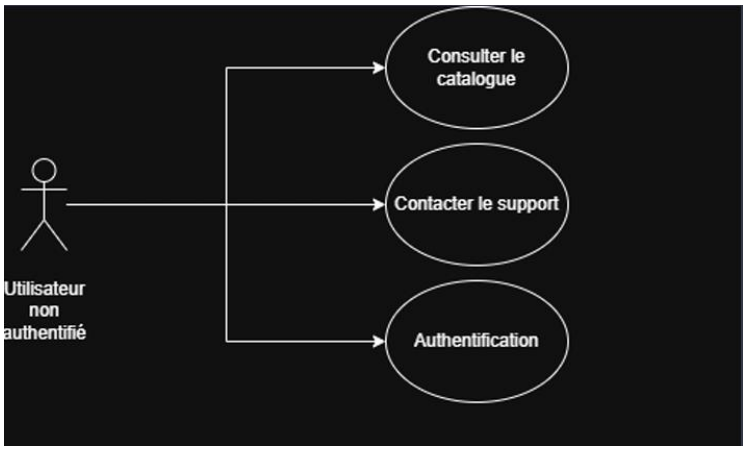
4. UML

- Cas d'utilisation : Visiteur, Acheteur, Vendeur, Admin

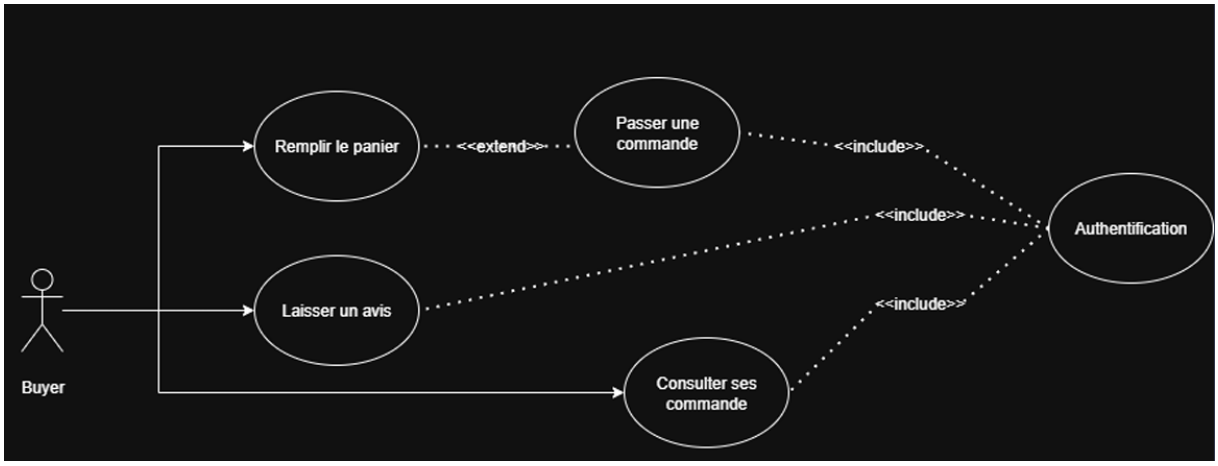
  Ces diagrammes permettent de visualiser les actions qu'un acteur peut réaliser sur le site selon son niveau d'autorisation. Bien sûr ils sont présentés par ordre de niveau donc qui peut le plus peut le moins.

  Visiteur simple :

  Il peut faire que des choses limités sur le site car il n'est pas connecté.

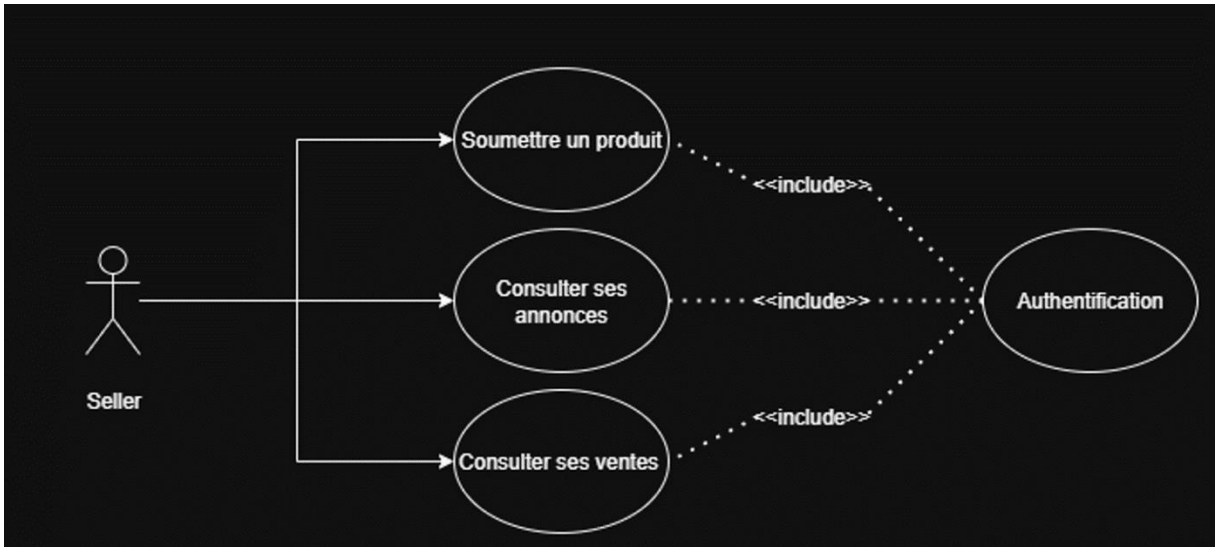


Acheteur :



Client simple qui peut faire des commandes.

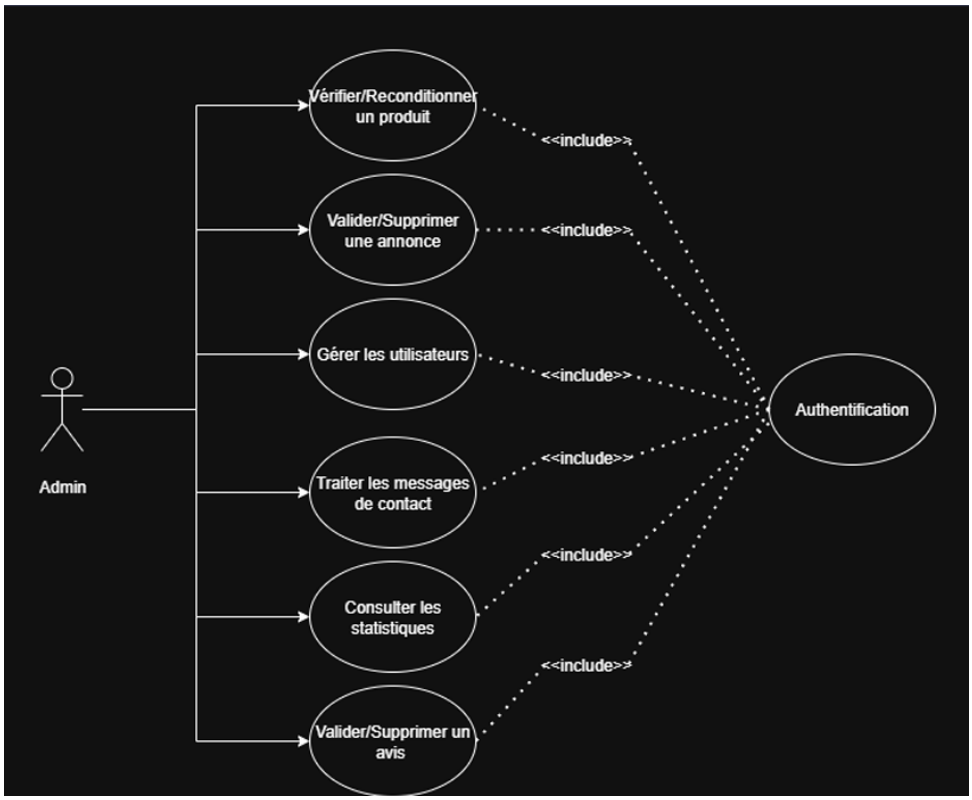
Le vendeur :



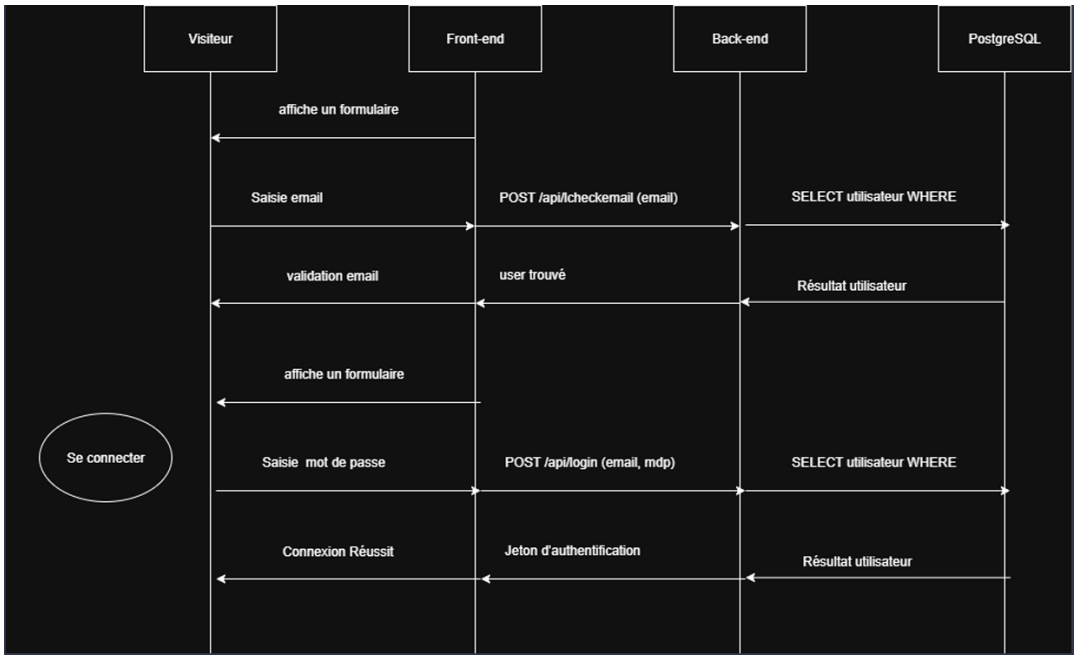
Cet utilisateur permet de soumettre des annonces et de récupérer le fruit de ses ventes.

L'administrateur :

Il peut gérer tout le site



- Séquence : se connecter



Le but de ce schéma permet d'avoir toutes les étapes nécessaires pour effectuer une action

Le reste des diagrammes de séquence se trouve dans l'annexe.

Charte Graphique et Maquettes

1. Présentation générale

La charte graphique de TechReuse Market a été conçue afin d'assurer une identité visuelle moderne, écologique et cohérente à travers toutes les pages du site.
Elle définit les couleurs, les typographies, les styles d'interfaces et les règles d'adaptation sur les différents supports (mobile, tablette, desktop).

L'objectif principal est de proposer une expérience utilisateur claire et intuitive, tout en valorisant le message écologique et technologique du projet.

2. Palette de couleurs

La palette de couleurs repose sur cinq teintes principales, soigneusement sélectionnées pour créer une identité à la fois sobre, professionnelle et dynamique :

Couleur	Code	Utilisation
Blanc pur	#FFFFFF	Arrière-plans principaux (~50%)
Bleu-gris foncé	#34374C	Navigation, titres, texte principal (~20%)
Rouge vif	#EE2B47	Boutons, accents visuels (~10%)
Gris clair	#F6F6F6	Fonds secondaires (~15%)
Bleu-gris très foncé	#2C2E3E	Bordures, ombres, contrastes (~5%)

Ce choix permet de transmettre une impression de confiance, de modernité et de durabilité, en cohérence avec la philosophie de revalorisation du matériel informatique.

3. Typographies

Les polices ont été sélectionnées pour associer modernité et lisibilité :

- Merriweather → utilisée pour les titres (H1, H2) afin d'apporter un aspect sérieux et professionnel.
- Noto Sans → utilisée pour les paragraphes et les boutons, pour sa clarté sur tous les écrans.
- Cormorant → utilisée pour les sous-titres et citations, ajoutant une touche élégante.

Les tailles de texte ont été définies pour le responsive design :

- H1 : 36px (desktop) / 28px (mobile)
- H2 : 28px / 22px
- Paragraphe : 16px / 14px
- Bouton : 20px / 18px

4. Logo

Le logo est textuel et reprend les deux parties du nom :

- TechReuse : fond #EE2B47, texte #F6F6F6
- Market : fond #F6F6F6, texte #EE2B47

Le tout est encadré d'un border-radius de 6px pour arrondir les angles, avec une police Montserrat Bold de 22px (desktop) et 18px (mobile).
Ce logo symbolise la dualité entre technologie et revalorisation, tout en restant simple et lisible.

Logo :

NORMAL : TechReuse Market TechReuseMarket

ALTER : TechReuse Market TechReuseMarket

5. Icônes et éléments graphiques

Les icônes proviennent de Font Awesome (style Solid), assurant une cohérence entre les différents éléments d'interface :

- fa-home : accueil
- fa-search : recherche
- fa-shopping-cart : panier
- fa-user : profil
- fa-star : notation
- fa-envelope : contact
- fa-trash : suppression
- fa-bars : menu mobile

Les icônes sont dimensionnées à 24px (desktop) et 20px (mobile), avec des marges internes de 8px.

6. Boutons et formulaires

Les boutons sont définis selon trois styles cohérents :

Type	Couleur	Effet
Principal	fond #EE2B47, texte blanc	hover : agrandissement +0.05
Secondaire	fond #2C2E3E, texte #F6F6F6	hover : inversion des couleurs
Tertiaire	fond transparent, bordure colorée	hover : léger fond de contraste

Les formulaires suivent une ligne épurée :

- Labels : Noto Sans, couleur #F6F6F6
- Champs : fond #F6F6F6, bordure #D5D5D5, radius 6px
- Focus : bordure accentuée #EE2B47
- Erreur : message rouge vif + icône d'alerte

7. Responsive design et animations

Le site est totalement responsive, avec trois points de rupture :

- Mobile : ≤ 767px
- Tablette : 768–1023px
- Desktop : ≥ 1024px

Les animations sont légères mais dynamiques :

- Effet *scale(1.05)* au survol des boutons et images
- Transitions douces (0.2s) pour renforcer la fluidité

9. Accessibilité et cohérence

Chaque image est accompagnée d'un texte alternatif (alt).
Les contrastes de couleurs respectent les normes WCAG AA, garantissant une bonne lisibilité.
La charte graphique a été testée avec Lighthouse afin de vérifier la performance, l'accessibilité et la cohérence visuelle sur toutes les pages principales.

Développement du Projet

1. Mise en place de l'environnement de développement

Le développement de TechReuse Market s'est appuyé sur un environnement moderne et modulaire.
Le projet est divisé en deux parties principales :

- Frontend : développé avec React + TypeScript via Vite pour la rapidité de build et le Hot Reloading.
- Backend : développé avec Node.js / Express, communiquant avec une base de données PostgreSQL via Sequelize ORM.

L'environnement local a été configuré grâce à Docker Compose, permettant d'exécuter simultanément :

- Un conteneur backend Node.js
- Un conteneur frontend React
- Un conteneur PostgreSQL
- Un conteneur Stripe

```
FROM node:23-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

```
FROM node:23-alpine (last pushed 5 months a
WORKDIR /app
COPY package*.json ./
RUN npm install
RUN npm install -g serve
COPY . .
RUN npm run build
EXPOSE 5173
CMD ["serve", "-s", "dist", "-l", "5173"]
```

```
▷ Run Service
frontend:
  build:
    context: ./my-app-frontend
    dockerfile: Dockerfile
  ports:
    - "5173:5173"

  env_file:
    - ./my-app-frontend/.env
  networks:
    - app-network
```

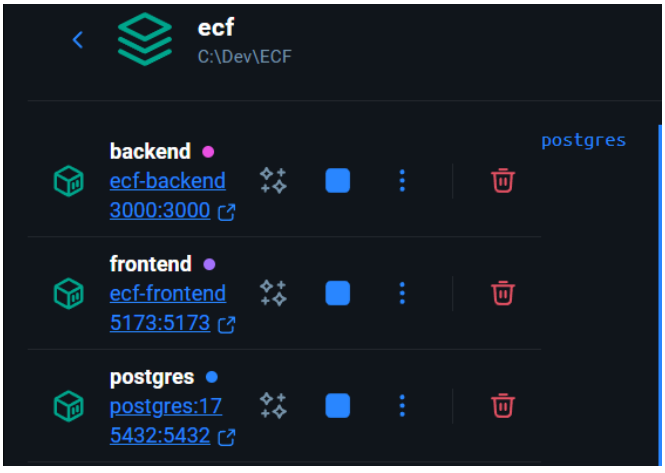
```
services:
  ▷ Run Service
  backend:
    build:
      context: ./my-backend-express
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - ./my-backend-express/certs:/app/certs
    env_file:
      - ./my-backend-express/.env.docker
    depends_on:
      - postgres
    networks:
      - app-network
```

```yaml
stripe-cli:
  image: stripe/stripe-cli:latest
  command: listen --forward-to https://backend:3000/payments/webhook --skip-verify
  volumes:
    - ~/.config/stripe:/root/.config/stripe
  depends_on:
    - backend
  networks:
    - app-network
```

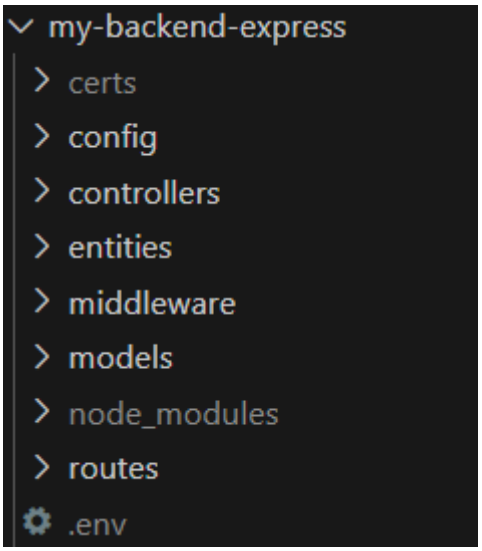
```yaml
postgres:
  image: postgres:17
  env_file:
    - ./postgres.env
  ports:
    - "5432:5432"
  volumes:
    - postgres_data:/var/lib/postgresql/data
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql
  networks:
    - app-network
```

```yaml
networks:
  app-network:
    driver: bridge

volumes:
  postgres_data:
```

L'utilisation de Docker a permis d'assurer une portabilité optimale entre les environnements de développement et de production, tout en facilitant la maintenance.



2. Développement du backend



Le backend repose sur Express.js, structuré selon une architecture MVC (Model – View – Controller) afin de séparer clairement :

- La logique métier (dans les contrôleurs),
- L'accès aux données (via les modèles Sequelize),
- Les routes HTTP (dans le dossier `routes`).

Fonctionnalités principales :

- Authentification sécurisée avec JWT et bcrypt (hash des mots de passe).
- Gestion des rôles : `BUYER`, `SELLER`, `ADMIN` pour définir les accès selon le profil utilisateur.
- Gestion des produits et annonces : publication, validation, suppression.
- Système de panier et commandes avec enregistrement automatique des transactions.
- Intégration du paiement via Stripe Checkout, relié au webhook pour valider automatiquement les commandes après paiement.
- Sécurité serveur : mise en place des middlewares `helmet`, `express-rate-limit` et `csurf` pour contrer les attaques XSS, CSRF et force brute.

```
"dependencies": {
  "argon2": "^0.43.0",
  "cookie-parser": "^1.4.7",
  "cors": "^2.8.5",
  "csrf": "^3.1.0",
  "dotenv": "^16.6.1",
  "express": "^5.1.0",
  "express-rate-limit": "^7.5.1",
  "express-validator": "^7.2.1",
  "firebase-admin": "^13.4.0",
  "google-auth-library": "^10.4.1",
  "helmet": "^8.1.0",
  "jsonwebtoken": "^9.0.2",
  "pg": "^8.16.2",
  "sequelize": "^6.37.7",
  "stripe": "^19.1.0"
}
```

La base de données relationnelle PostgreSQL est gérée avec Sequelize, ce qui permet :

- La création automatique des tables,
- La validation des relations (FK, contraintes, ENUM),
- Une gestion simplifiée des migrations.

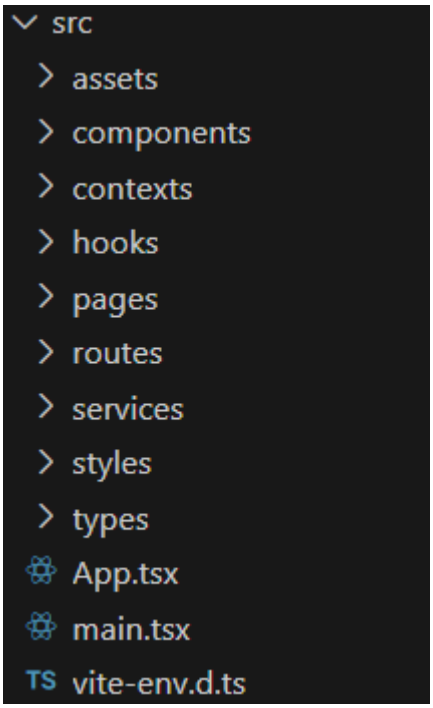
```
const { Sequelize } = require('sequelize');
require('dotenv').config();

const sequelize = new Sequelize(process.env.DB_NAME, process.env.DB_USER, process.env.DB_PASSWORD, {
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  dialect: 'postgres',
  logging: false,
});

module.exports = sequelize;
```

```
sequelize
  .sync({ alter: true })
  .then(() => {
    console.log("Base de données synchronisée avec succès.");
    https.createServer(credentials, app).listen(PORT, ()  => {
        console.log(` ✅ Serveur HTTPS démarré sur https://localhost:${PORT}`);
    });
  })
  .catch((err) => {
    console.error(
      "Erreur lors de la synchronisation de la base de données :",
      err
    );
  });
```

3. Développement du Frontend

Le frontend de l'application TechReuse Market a été développé avec React 19 et TypeScript, en utilisant le bundler moderne Vite pour sa rapidité et sa configuration minimale.

Architecture du projet



Le code est organisé en plusieurs dossiers :

components/ → composants réutilisables (Header, Footer, Cards, Modals…)

pages/ → pages principales de l'application (Home, Catalogue, Product, Cart, Checkout…)

contexts/ et hooks/ → gestion de l'état utilisateur et du panier

services/ → appels API via Axios

styles/ → fichiers pour le style global de l'app

Cette architecture permet une séparation claire entre la logique métier, la présentation et les appels réseau.

Interface utilisateur :

Le design repose sur Styled Components, une librairie de CSS-in-JS qui permet :

De définir des styles dynamiques directement dans les composants,

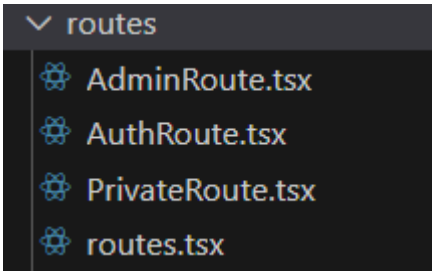
De gérer facilement les thèmes (couleurs, typographie, responsive),

D'éviter les conflits de classes CSS.

Les animations et transitions sont gérées avec GSAP (GreenSock) pour offrir une expérience fluide et moderne (effets de slide, apparition de cartes produits, transitions de pages, etc.).

Navigation :

La navigation est assurée par React Router DOM (v7) :



Routes publiques : Accueil, Catalogue, Détails produit, Contact, Connexion/Inscription.

Routes protégées : Panier, Commande, Tableau de bord utilisateur (Seller/Admin).

Les routes privées sont protégées à l'aide d'un système de JWT et d'un contexte d'authentification vérifiant la validité du token avant d'autoriser l'accès.

(voir Annexe pour la liste des routes détailé)

Gestion des formulaires :

Les formulaires (connexion, inscription, publication d'annonce, contact, paiement) sont développés avec :

Formik pour la gestion des champs et du state,

Yup pour la validation côté client (email, mot de passe, téléphone, prix…).

```
{!checkedEmail ? (
  <>
    <p>Saisissez votre e-mail pour vous connecter.</p>
    <Formik
      key="email-step"
      initialValues={{ email: "" }}
      validationSchema={validationSchema1}
      onSubmit={handleEmailCheck}
    >
      {() => (
        <FormikForm>
```

```
const validationSchema1 = Yup.object({
  email: Yup.string()
    .required("L'email est obligatoire")
    .email("Format d'email invalide")
    .max(64, "Email trop long !"),
});

const validationSchema2 = Yup.object({
  password: Yup.string()
    .required("Le mot de passe est obligatoire")
    .min(12, "12 caractères minimum")
    .max(64, "Mot de passe trop long !")
    .matches(/[A-Z]/, "Doit contenir au moins une majuscule")
    .matches(/[a-z]/, "Doit contenir au moins une minuscule")
    .matches(/\d/, "Doit contenir au moins un chiffre")
    .matches(/[!@#$%^&*]/, "Doit contenir au moins un caractère spécial"),
});
```

Cette combinaison garantit une validation fluide et des messages d'erreur clairs pour l'utilisateur.

Intégration du paiement Stripe

L'application utilise @stripe/react-stripe-js et @stripe/stripe-js pour intégrer le paiement sécurisé :

```
<PageContainer >
    <h1>Paiement sécurisé</h1>
    <p>Total : {total.toFixed(2)} €</p>
    <Button
      text="Procéder au paiement"
      onClick={async () => {
        const order = await axios.post("/orders");
        const res = await axios.post("/payments/create-checkout-session", {
          orderId: order.data.id_order,
        });
        window.location.href = res.data.url;
      }}
    />      You, 4 days ago • update merise …
</PageContainer>
```

Le frontend crée une session de paiement en appelant l'API backend.

Stripe gère la redirection vers la page de paiement sécurisée.

En cas de succès ou d'annulation, l'utilisateur est redirigé vers les pages PaymentSuccess ou PaymentCancel.

```
const session = await stripe.checkout.sessions.create({
  payment_method_types: ["card"],
  mode: "payment",
  line_items: [
    {
      price_data: {
        currency: "eur",
        product_data: {
          name: `Commande ${order.id_order}`,
        },
        unit_amount: Math.round(order.total_amount * 100),
      },
      quantity: 1,
    },
  ],
  customer_email: req.user.email,
  success_url: `${process.env.FRONT_URL}/payment-success?order=${order.id_order}`,
  cancel_url: `${process.env.FRONT_URL}/payment-cancel?order=${order.id_order}`,
  metadata: {
    order_id: order.id_order,
    buyer_id: req.user.id,
  },
});

res.json({ url: session.url });
```

L'intégration a été testée avec les cartes de test Stripe afin de vérifier le bon fonctionnement des flux de paiement et la synchronisation avec le webhook côté serveur.

Sécurité côté client :

Plusieurs mesures de sécurité ont été mises en œuvre :

Google reCAPTCHA pour la prévention des robots sur les formulaires.

Validation stricte des entrées via Yup.

Appels API sécurisés avec Axios et envoi automatique du JWT dans les en-têtes.

Gestion du logout automatique en cas d'expiration du token.

Expérience utilisateur :

L'interface est responsive et optimisée pour les écrans mobiles, tablettes et ordinateurs. Les icônes proviennent de Font Awesome, et les animations assurent une navigation fluide et moderne.

```
"dependencies": {
  "@fortawesome/fontawesome-svg-core": "^6.7.2",
  "@fortawesome/free-brands-svg-icons": "^6.7.2",
  "@fortawesome/free-regular-svg-icons": "^6.7.2",
  "@fortawesome/free-solid-svg-icons": "^6.7.2",
  "@fortawesome/react-fontawesome": "^0.2.2",
  "@react-oauth/google": "^0.12.2",
  "@stripe/react-stripe-js": "^5.2.0",
  "@stripe/stripe-js": "^8.1.0",
  "axios": "^1.10.0",
  "firebase": "^11.9.1",
  "formik": "^2.4.6",
  "gsap": "^3.13.0",
  "react": "^19.1.0",
  "react-dom": "^19.1.0",
  "react-google-recaptcha": "^3.1.0",
  "react-router-dom": "^7.6.2",
  "styled-components": "^6.1.19",
  "uuid": "^11.1.0",
  "yup": "^1.6.1"
},      Assi Melvin, 4 months ago • init back and
```

Tests et Validation

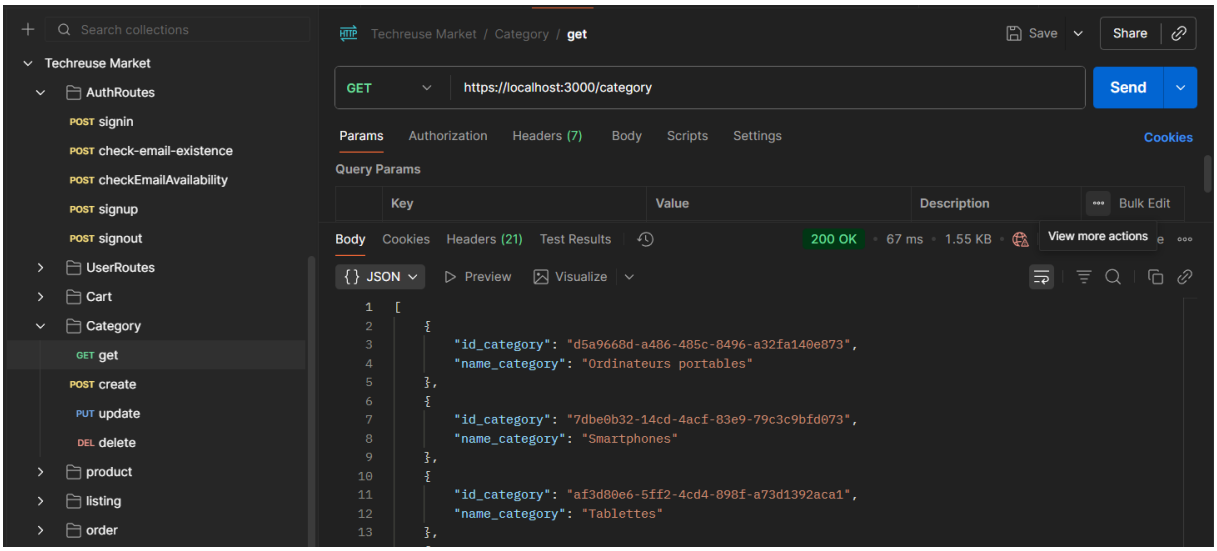
Afin d'assurer la fiabilité et la qualité du projet TechReuse Market, plusieurs types de tests ont été réalisés tout au long du développement :

1. Tests unitaires et de validation des routes API

Les principales routes du back-end (authentification, gestion des produits, commandes et catégories) ont été testées à l'aide de Postman.
Chaque requête HTTP (GET, POST, PUT, DELETE) a été vérifiée avec des cas de succès et d'échec :

- /auth/signup et /auth/signin : validation des règles de sécurité et du JWT.
- /category : création, modification et récupération des catégories.
- /orders et /transactions : contrôle du bon enchaînement des statuts (PENDING, VALIDATED, CANCELLED).

Des tests de cohérence SQL ont également été effectués via Sequelize pour s'assurer de l'intégrité des clés étrangères et des relations entre tables.



2. Tests d'intégration front-back

Les tests d'intégration ont porté sur :

- La communication entre React et Express via Axios.
- La gestion des tokens JWT dans les routes privées.

- Le maintien du panier et de la session utilisateur entre plusieurs appareils connectés.
- La validation des formulaires avec Formik et Yup.

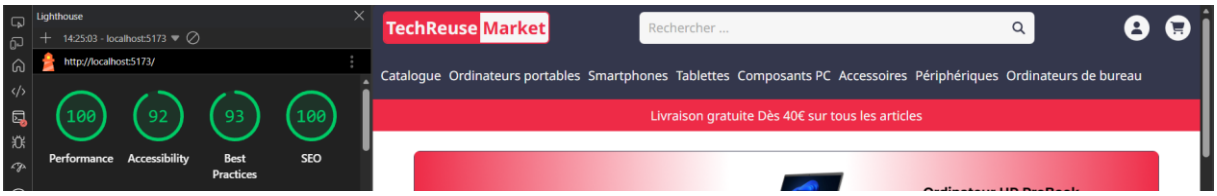
Ces vérifications ont permis d'assurer la cohérence du parcours utilisateur complet : inscription → navigation → ajout au panier → paiement → confirmation.

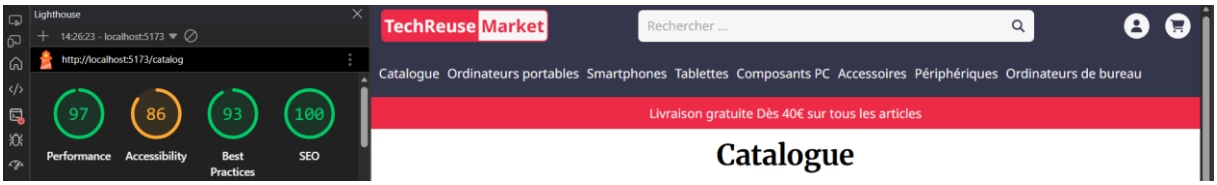
3. Tests de performance et accessibilité

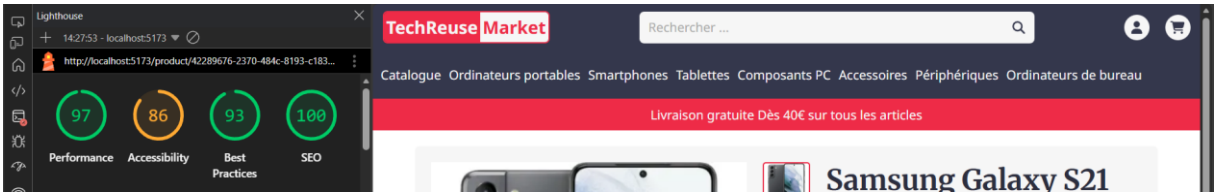
Les pages principales (Home, Catalog, Product, SignIn, SignUp) ont été testées avec Google Lighthouse en mode production (npm run build).
Les scores moyens observés sont :

- Performance : entre 90 et 98
- Accessibilité : entre 95 et 100
- Bonnes pratiques : entre 95 et 100
- SEO : entre 90 et 100

Ces résultats démontrent une application optimisée et conforme aux standards web actuels.







Résultats et Déploiement

1. Résultats obtenus

Le projet TechReuse Market est entièrement fonctionnel et répond aux exigences du cahier des charges :

- Authentification complète avec JWT et OAuth (Google)
- Gestion des panier, commandes, transactions et avis clients
- Paiement sécurisé via Stripe Checkout
- Interface réactive et ergonomique conçue avec React, TypeScript et Styled-components
- Sécurité renforcée côté serveur avec Helmet, Rate Limiter, validation des entrées et HTTPS
- Base de données relationnelle PostgreSQL normalisée selon le modèle MERISE
- Tableau de bord administrateur pour la gestion des utilisateurs, catégories et produits

2. Déploiement

Le projet a été préparé pour un déploiement sur un environnement de production Dockerisé :

- Back-end : conteneur Express avec configuration .env sécurisée.
- Front-end : build React (vite build)
- Base de données : instance PostgreSQL
- Paiements : intégration Stripe avec clés secrètes en variable d'environnement.
- Sécurité HTTPS :

Les scripts de déploiement et la documentation technique permettent une mise en production reproductible et sécurisée.

3. Notice Technique

Cette notice a pour objectif d'expliquer les étapes nécessaires pour lancer, développer et déployer le projet TechReuse. Elle couvre l'utilisation de Docker, Node.js ainsi que les outils tiers comme Stripe.

Structure du projet

```
/project-root
├── backend/ # API Express + Stripe + PostgreSQL
├── frontend/ # React + Vite
├── docker-compose.yml
└── postgres.env # Variables d'environnement pour PostgreSQL
```

Prérequis

- Node.js ≥ 18
- Docker Desktop ou moteur Docker + Docker Compose v2
- Stripe CLI (en développement uniquement)
- Navigateur Web moderne

Déploiement avec Docker Compose

Le projet utilise Docker Compose pour orchestrer l'ensemble des services :

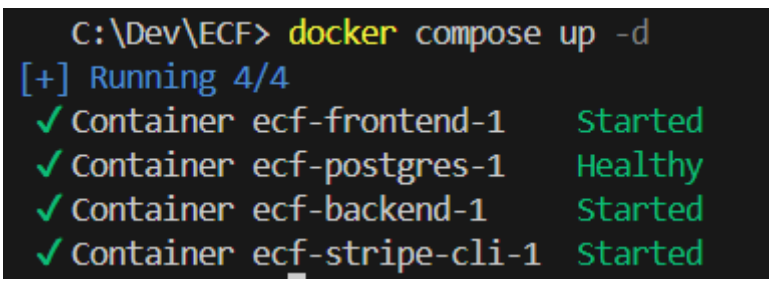
Service	Conteneur	Description
Backend	backend	API Node.js (port 3000)
Frontend	frontend	Interface React (port 5173)
Base de données	postgres	PostgreSQL 17 (port 5432)

Démarrer les services

Pour lancer l'application, il faut tout d'abord démarrer DockerDesktop. Puis dans le terminal à la racine du dossier vous pouvez écrire la commande suivante :

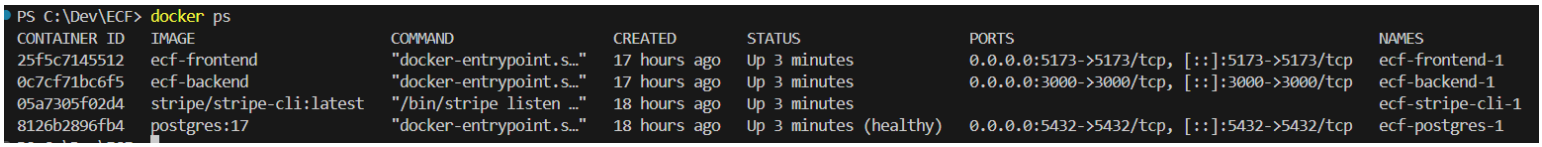
docker compose up -d

Tous les services sont automatiquement buildés et lancés en arrière-plan. Voici le résultat :



Pour vérifier que tout fonctionne :

docker ps



```
PS C:\Dev\ECF> docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED       STATUS                   PORTS                                       NAMES
25f5c7145512   ecf-frontend               "docker-entrypoint.s…"   17 hours ago  Up 3 minutes             0.0.0.0:5173->5173/tcp, [::]:5173->5173/tcp ecf-frontend-1
0c7cf71bc6f5   ecf-backend                "docker-entrypoint.s…"   17 hours ago  Up 3 minutes             0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp ecf-backend-1
05a7305f02d4   stripe/stripe-cli:latest   "/bin/stripe listen …"   18 hours ago  Up 3 minutes                                                         ecf-stripe-cli-1
8126b2896fb4   postgres:17                "docker-entrypoint.s…"   18 hours ago  Up 3 minutes (healthy)   0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp ecf-postgres-1
```

Puis accéder à :

- API : https://localhost:3000
- Site web : http://localhost:5173

Voici quelque autre ligne commande utile pour la gestion de docker et ses containers.

Mise à jour des services (après modification)

docker compose restart

Arrêter les services

docker compose down

Nettoyage complet (volumes + réseaux)

docker compose down --volumes --remove-orphans

Mode Développement (sans Docker)

Vous pouvez aussi travailler uniquement en local si nécessaire.

Pour le Backend :

cd backend

npm install

npm run dev

Pour le Frontend :

cd frontend

npm install

npm run dev

Le serveur frontend démarre par défaut sur http://localhost:5173

Pour Stripe CLI

Pour tester localement le webhook Stripe :

stripe.exe listen --forward-to localhost:3000/payments/webhook

Cela permet à Stripe d'envoyer les événements (paiement réussi, échec, etc.) directement à l' API.

Gestion des variables d'environnement

Chaque service utilise son propre fichier .env (non versionné) avec des informations sensibles comme par example :

- Clé Stripe
- JWT_SECRET
- URL de base de données
- URL du front
- Clé Google Recaptcha
- Clé GoogleAuth

Il y a des fichiers .env.example pour chaque container il suffit de crée les différents .env et de le remplir comme dans les exemple.

Génération de build (front uniquement)

Pour avoir des meilleurs performances :

cd frontend

npm run build

Conclusion

Le développement de TechReuse Market a permis de mettre en œuvre l'ensemble des compétences du référentiel Développeur Web et Web Mobile :

- Conception d'une base de données complète (MERISE + MLD + MPD)
- Développement Full Stack (React / Express / PostgreSQL)
- Mise en place d'un système d'authentification sécurisé (JWT + OAuth)
- Intégration de paiements en ligne avec Stripe
- Application des bonnes pratiques de sécurité et d'optimisation
- Gestion du projet en méthode Agile, avec versionnement Git/GitHub et suivi via Trello.

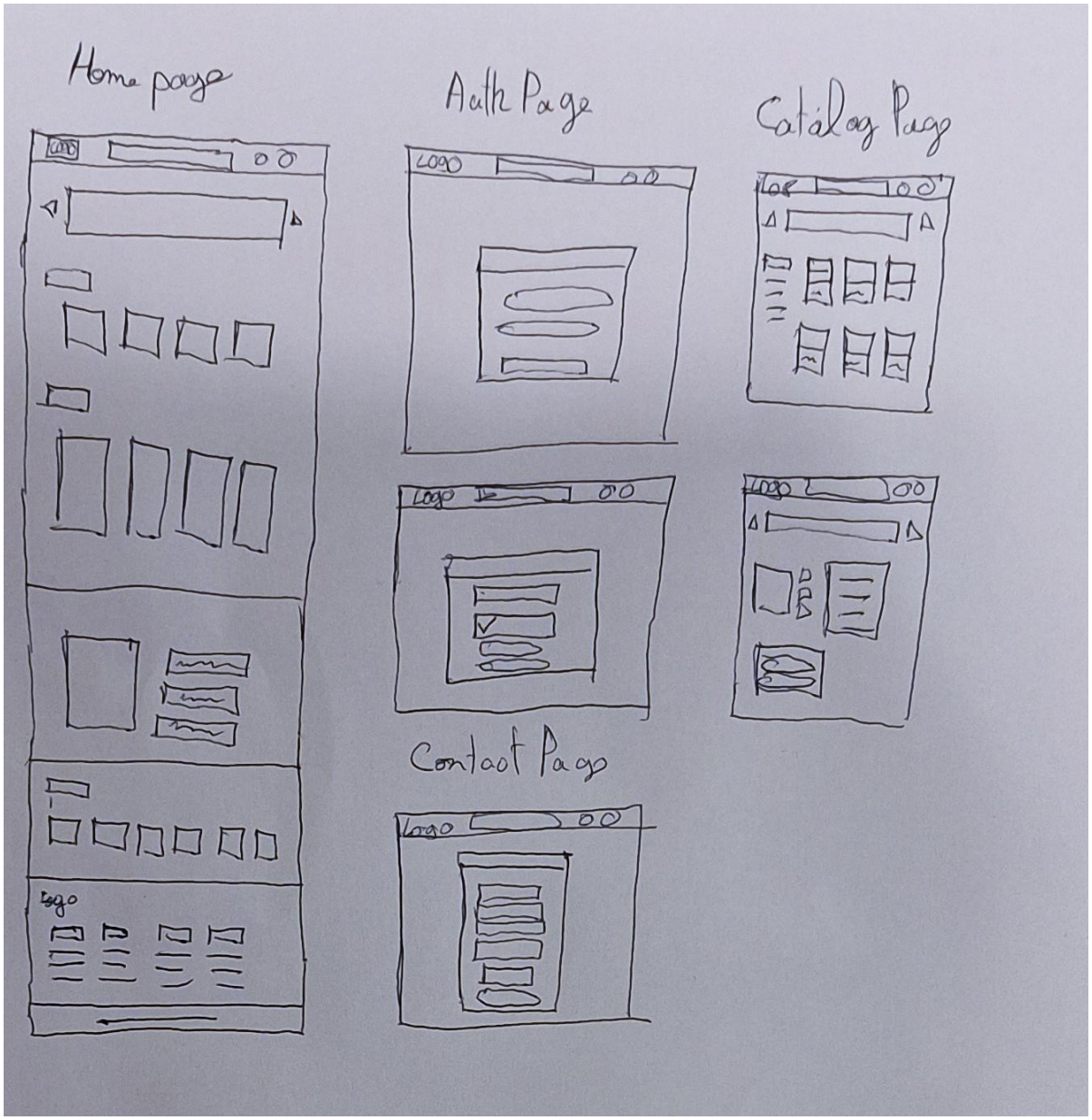
Ce projet représente une plateforme fonctionnelle, moderne et sécurisée, destinée à promouvoir la seconde vie du matériel informatique, tout en contribuant à l'économie circulaire numérique.

L'expérience acquise à travers ce projet a permis de renforcer les compétences techniques, méthodologiques et professionnelles nécessaires à la conception d'applications web complètes et sécurisées, prêtes pour un déploiement en production.

Annexes

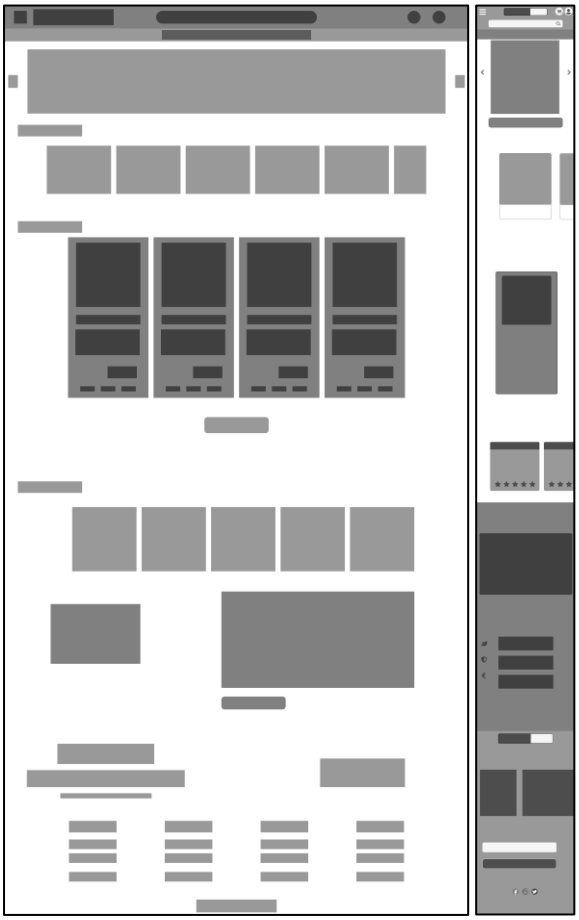
1. Maquettes Figma

   Zoonning :

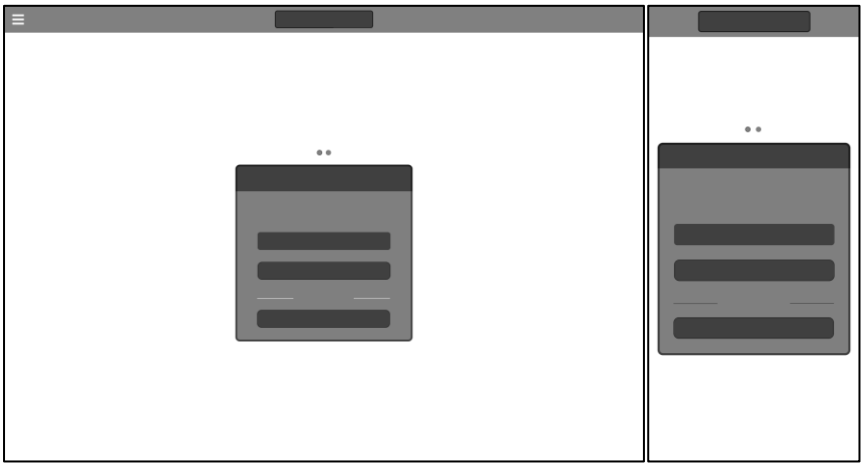


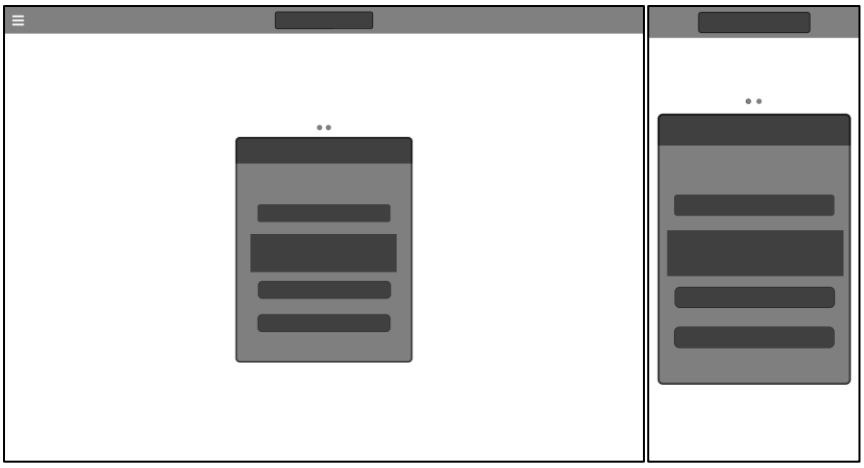
WireFrame :

HomePage :

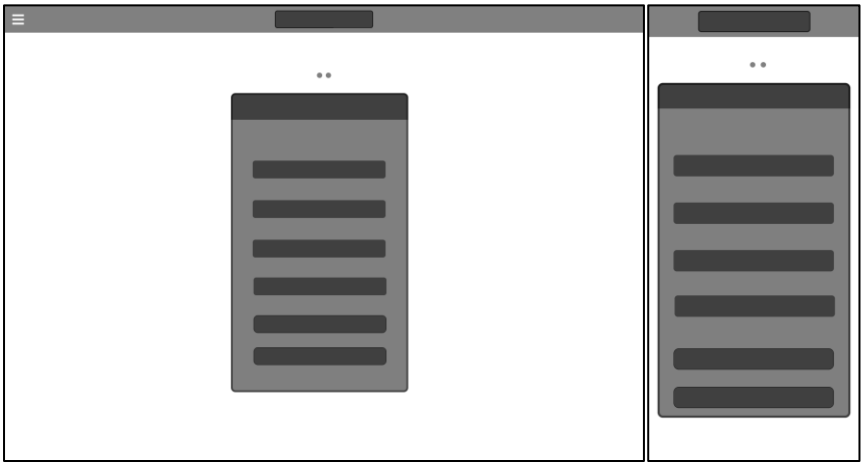


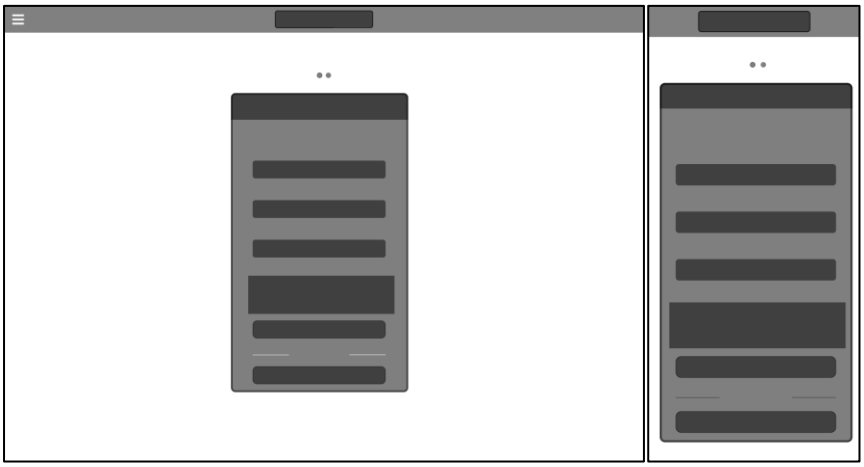
LoginPage :





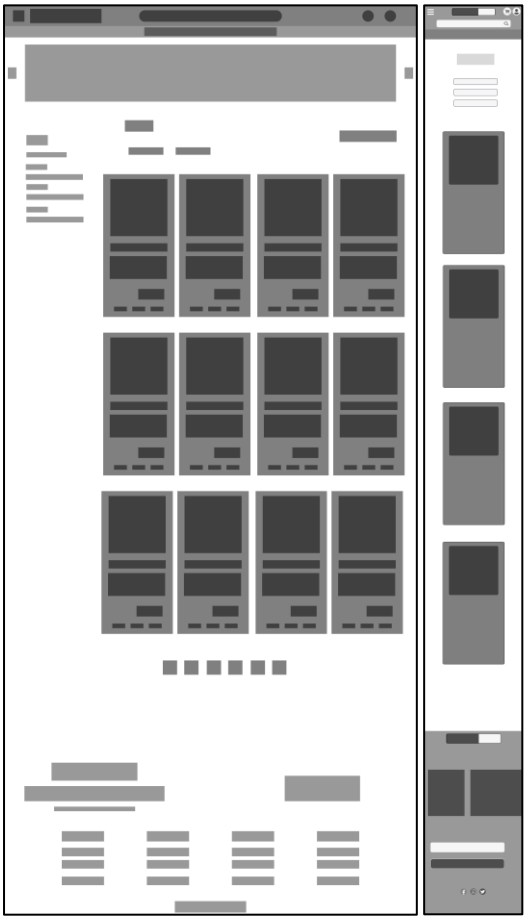
Singup Page :



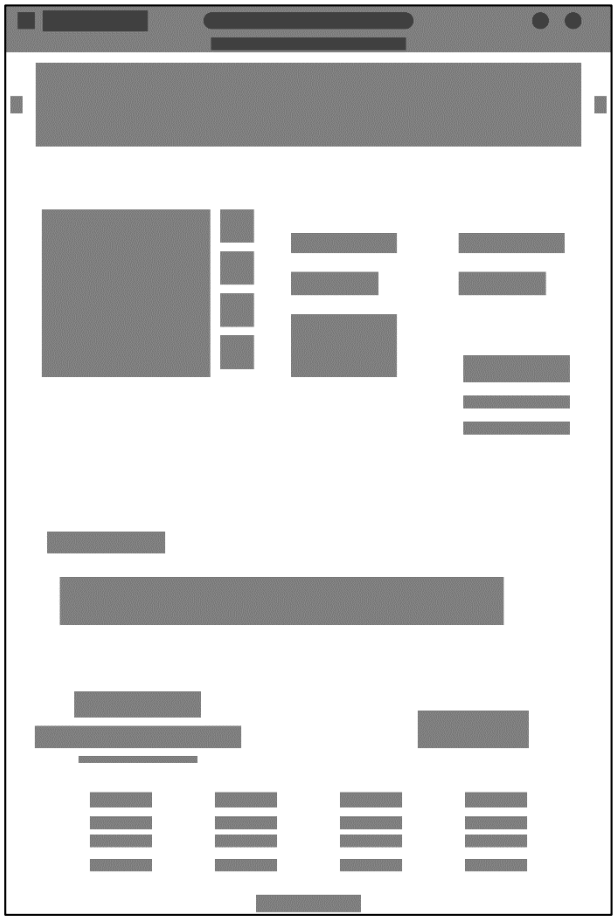


Contact Page :

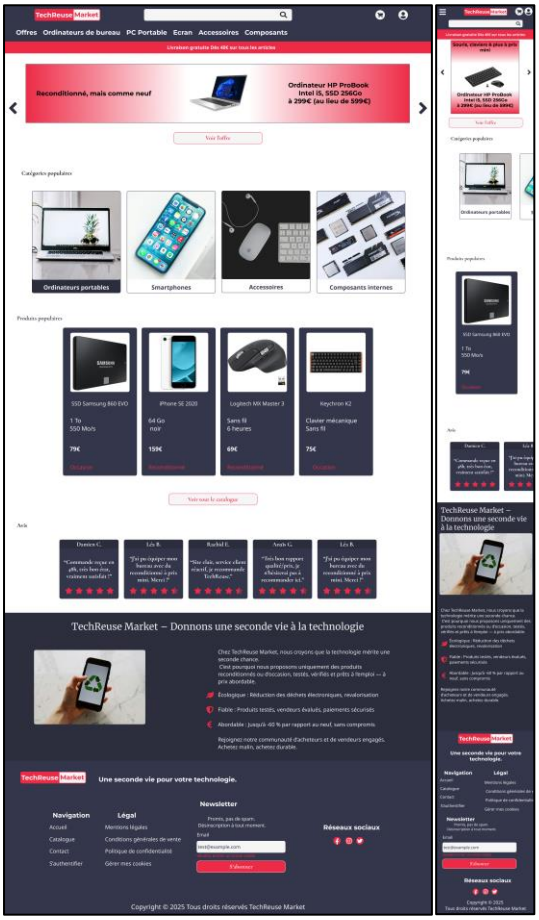
Catalog Page :

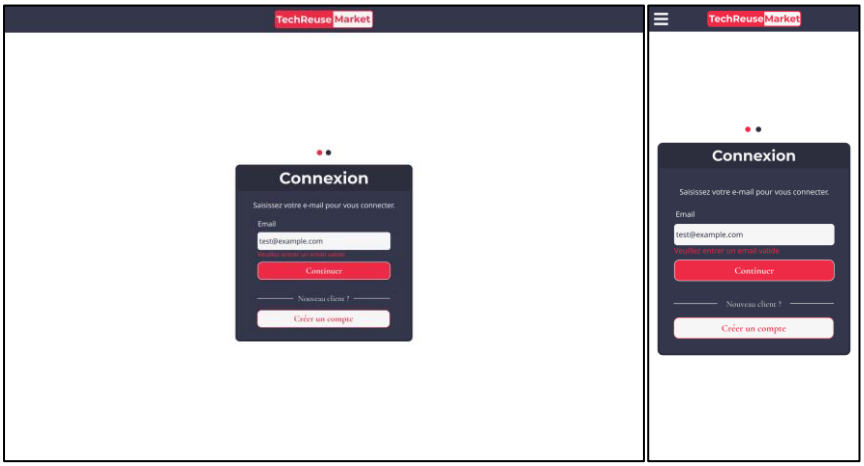


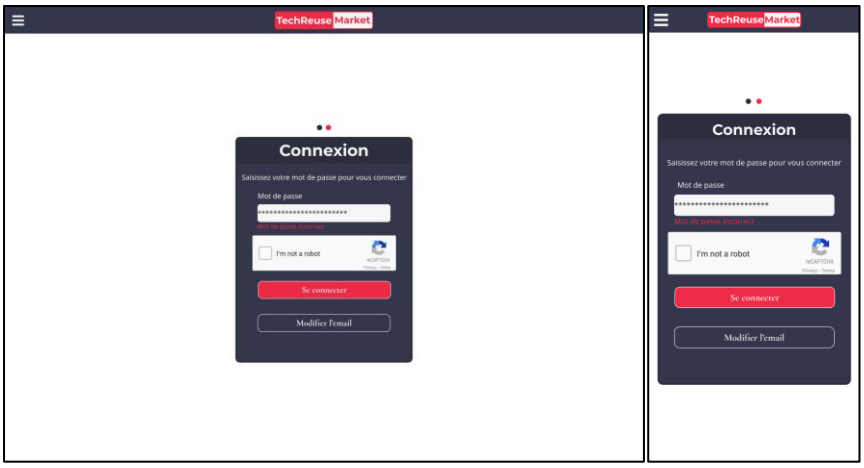
Product Page :

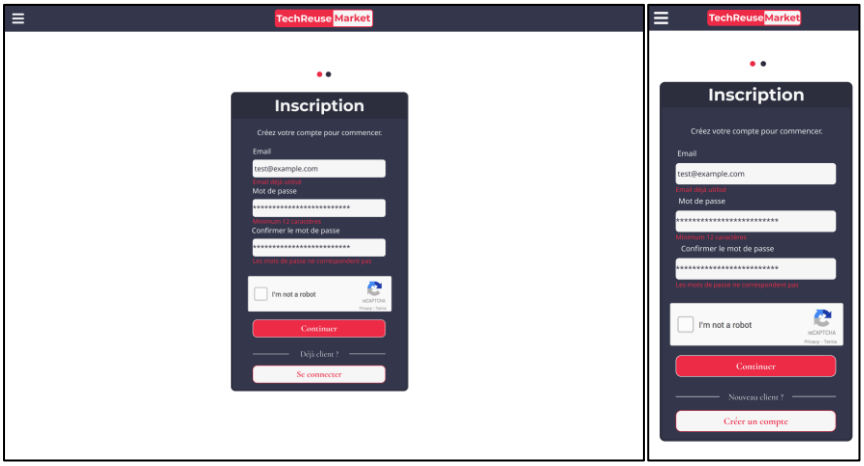


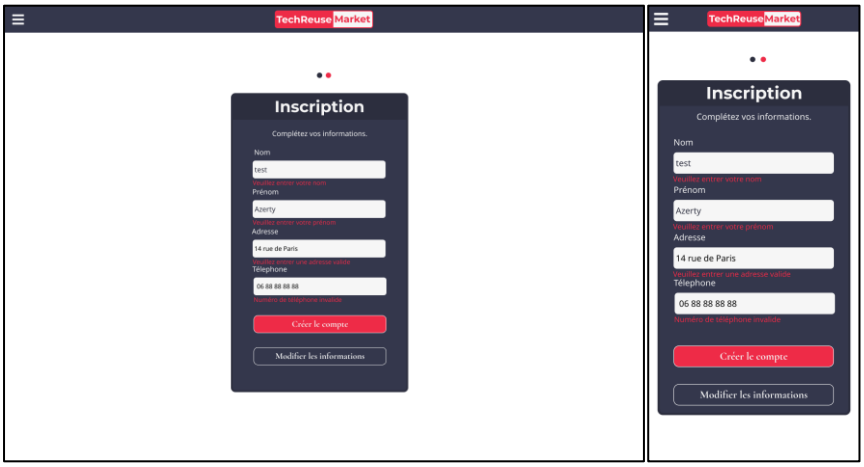
Mockup :

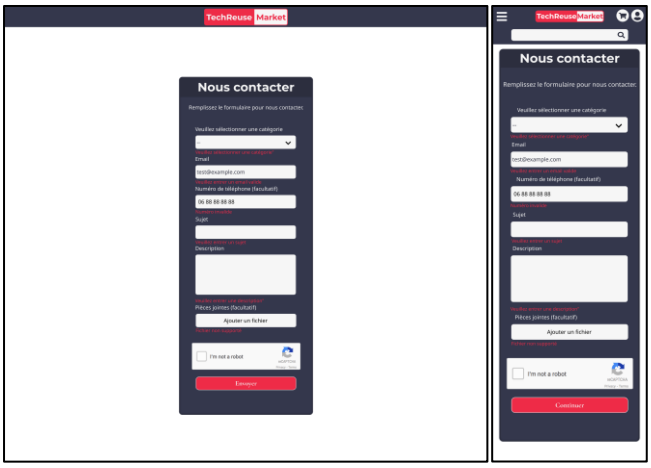

TechReuse Market
Offres Ordinateurs de bureau PC Portable Ecran Accessoires Composants
Reconditionné, mais comme neuf
Ordinateur HP ProBook
Intel i5, SSD 256Go
à 299€ (au lieu de 599€)
Catégories populaires
Ordinateurs portables
Smartphones
Accessoires
Composants internes
Produits populaires
SSD Samsung 860 EVO
1 To
550 Mo/s
79€
iPhone SE 2020
64 Go
noir
159€
Logitech MX Master 3
Sans fil
6 heures
69€
Keychron K2
Clavier mécanique
Sans fil
75€
Avis
TechReuse Market – Donnons une seconde vie à la technologie
Chez TechReuse Market, nous croyons que la technologie mérite une seconde chance.
C'est pourquoi nous proposons uniquement des produits reconditionnés au choix soigneux, testés, vérifiés et prêts à l'emploi — à prix abordable.
Écologique : Réduction des déchets électroniques, revalorisation
Fiable : Produits testés, vendeurs évalués, paiements sécurisés
Abordable : jusqu'à -40 % par rapport au neuf, sans compromis
Rejoignez notre communauté d'acheteurs et de vendeurs engagés.
Achetez malin, achetez durable.
TechReuse Market
Une seconde vie pour votre technologie.
Navigation
Accueil
Catalogue
Contact
Sauthentifier
Légal
Mentions légales
Conditions générales de vente
Politique de confidentialité
Gérer mes cookies
Newsletter
Réseaux sociaux
Copyright © 2025 Tous droits réservés TechReuse Market


TechReuse Market
Connexion
Saisissez votre e-mail pour vous connecter.
Email
test@example.com
Continuer
Nouveau client ?
Créer un compte
TechReuse Market
Connexion
Saisissez votre e-mail pour vous connecter.
Email
test@example.com
Continuer
Nouveau client ?
Créer un compte


TechReuseMarket
Connexion
Saisissez votre mot de passe pour vous connecter
Mot de passe
Mot de passe incorrect
I'm not a robot
Se connecter
Modifier l'email


TechReuseMarket
Inscription
Créez votre compte pour commencer.
Email
test@example.com
Email déjà utilisé
Mot de passe
Minimum 12 caractères
Confirmer le mot de passe
Les mots de passe ne correspondent pas
I'm not a robot
Continuer
Déjà client ?
Se connecter


TechReuseMarket
Inscription
Complétez vos informations.
Nom
test
Veuillez entrer votre nom
Prénom
Azerty
Veuillez entrer votre prénom
Adresse
14 rue de Paris
Veuillez entrer une adresse valide
Téléphone
06 88 88 88 88
Numéro de téléphone invalide
Créer le compte
Modifier les informations


TechReuse Market
Nous contacter
Remplissez le formulaire pour nous contacter.
Veuillez sélectionner une catégorie
Email
Numéro de téléphone (facultatif)
Sujet
Description
Pièces jointes (facultatif)
Ajouter un fichier
I'm not a robot
Envoyer
Continuer


Offres Ordinateurs de bureau PC Portable Ecran Accessoires Composants
Catalogue
Categories
Conditions
Verification
SSD Samsung 860 EVO
iPhone SE 2020
Logitech MX Master 3
Keychron K2
1 To
550 Mo/s
64 Go
noir
Sans fil
6 heures
Clavier mécanique
Sans fil
79€
159€
69€
75€
1 2 3 4 5 6
Une seconde vie pour votre technologie.
Navigation
Accueil
Catalogue
Contact
Vauthentifier
Légal
Mentions légales
Conditions générales de vente
Politique de confidentialité
Gérer mes cookies
Newsletter
Réseaux sociaux
Copyright © 2025 Tous droits réservés TechReuse Market
Reconditionné, mais comme neuf
Ordinateur HP ProBook
Intel i5, SSD 256Go
à 299€ (au lieu de 599€)
Voir l'offre
SAMSUNG
Bon état
Ajouter au panier
Acheter cette article
Fiche technique

Dictionnaire de Données :

1. Table User

Nom du champ	Type	Contraintes	Description
id_user	UUID / SERIAL	PK, unique, non nul	Identifiant unique de l'utilisateur
email	VARCHAR(100)	unique, non nul, format email	Adresse e-mail de l'utilisateur
password	TEXT	haché (bcrypt/Argon2id)	Mot de passe haché
name	VARCHAR(100)	non nul	Nom de l'utilisateur
firstname	VARCHAR(100)	non nul	Prénom de l'utilisateur
adress	VARCHAR(100)	non nul	Adresse de l'utilisateur
phone	VARCHAR(20)	non nul, format téléphone (10 chiffres)	Numéro de téléphone de l'utilisateur
role	ENUM	valeurs : 'BUYER', 'SELLER', 'ADMIN'	Rôle de l'utilisateur
created_at	TIMESTAMP	auto-généré	Date de création du compte
is_active	BOOLEAN	défaut true	Statut actif ou désactivé
last_activity	TIMESTAMP	optionnel	Dernière activité pour gestion des sessions

2. Table Product

Nom du champ	Type	Contraintes	Description
id_product	UUID / SERIAL	PK, unique, non nul	Identifiant unique du produit
name	VARCHAR(100)	non nul	Nom du produit

description	TEXT	optionnel	Description du produit
price	DECIMAL(10,2)	non nul, ≥ 0	Prix du produit
condition	ENUM	valeurs : 'NEW', 'GOOD', 'USED'	État initial du produit
verification_status	ENUM	valeurs : 'UNDER_VERIFICATION', 'RECONDITIONED', 'READY_TO_SELL', 'REJECTED'	Statut de vérification
images	TEXT[]	optionnel	URLs des images du produit
category_id	FOREIGN KEY	vers Category(id_category), non nul	Référence à la catégorie

3. Table Listing

Nom du champ	Type	Contraintes	Description
id_listing	UUID / SERIAL	PK, unique, non nul	Identifiant unique de l'annonce
publication_date	TIMESTAMP	auto-généré	Date de publication
status	ENUM	valeurs : 'PENDING', 'ONLINE', 'DELETED'	Statut de l'annonce
seller_id	FOREIGN KEY	vers User(id_user), non nul	Vendeur ayant publié l'annonce
product_id	FOREIGN KEY	vers Product(id_product), non nul	Produit lié à l'annonce

4. Table Category

Nom du champ	Type	Contraintes	Description
id_category	UUID / SERIAL	PK, unique, non nul	Identifiant unique de la catégorie
name_category	VARCHAR(100)	non nul, unique	Nom de la catégorie (ex. : ordinateurs, smartphones)

5. Table Order

Nom du champ	Type	Contraintes	Description
id_order	UUID / SERIAL	PK, unique, non nul	Identifiant unique de la commande
order_date	TIMESTAMP	auto-généré	Date de la commande
total_amount	DECIMAL(10,2)	non nul, ≥ 0	Montant total de la commande
status	ENUM	valeurs : 'PENDING', 'VALIDATED', 'DELIVERED'	Statut de la commande
buyer_id	FOREIGN KEY	vers User(id_user), non nul	Acheteur ayant passé la commande

---

6. Table Cart

Nom du champ	Type	Contraintes	Description
id_cart	UUID / SERIAL	PK, unique, non nul	Identifiant unique du panier
buyer_id	FOREIGN KEY	vers User(id_user), non nul	Acheteur associé au panier
creation_date	TIMESTAMP	auto-généré	Date de création du panier

7. Table Cart_Listing (Table d'association)

Nom du champ	Type	Contraintes	Description
id_cart	FOREIGN KEY	vers Cart(id_cart), non nul	Référence au panier
id_listing	FOREIGN KEY	vers Listing(id_listing), non nul	Référence au Listing
quantity	INTEGER	non nul, ≥ 1	Quantité du produit dans le panier

Clé primaire	(id_cart, id_listing)	Combinaison unique	Clé composite

8. Table Order_Listing (Table d'association)

Nom du champ	Type	Contraintes	Description
order_id	UUID	PK, FK vers Order(id_order)	Commande concernée
listing_id	UUID	PK, FK vers Listing(id_listing)	Listing commandé
quantity	INTEGER	≥ 1, non nul	Quantité achetée
unit_price	DECIMAL(10,2)	≥ 0, non nul	Prix unitaire au moment de la commande

9. Table Review

Nom du champ	Type	Contraintes	Description
id_review	UUID / SERIAL	PK, unique, non nul	Identifiant unique de l'avis
comment	TEXT	optionnel	Commentaire de l'avis
rating	INTEGER	non nul, 1 à 5	Note attribuée
review_date	TIMESTAMP	auto-généré	Date de l'avis
buyer_id	FOREIGN KEY	vers User(id_user), non nul	Acheteur ayant laissé l'avis
product_id	FOREIGN KEY	vers Product(id_product), optionnel	Produit concerné (optionnel si avis sur vendeur)
seller_id	FOREIGN KEY	vers User(id_user), optionnel	Vendeur concerné (optionnel si avis sur produit)

10. Table Transaction

Nom du champ	Type	Contraintes	Description
id_transaction	UUID / SERIAL	PK, unique, non nul	Identifiant unique de la transaction
transaction_date	TIMESTAMP	auto-généré	Date de la transaction

amount	DECIMAL(10,2)	non nul, ≥ 0	Montant de la transaction
status	ENUM	valeurs : 'PENDING', 'VALIDATED', 'CANCELLED'	Statut de la transaction
buyer_id	FOREIGN KEY	vers User(id_user), non nul	Acheteur
seller_id	FOREIGN KEY	vers User(id_user), non nul	Vendeur
order_id	FOREIGN KEY	vers Order(id_order), non nul	Commande associée

---

11. Table Contact

Nom du champ	Type	Contraintes	Description
id_contact	UUID / SERIAL	PK, unique, non nul	Identifiant unique du message
title	VARCHAR(255)	non nul	Titre du message
description	TEXT	non nul	Corps du message
email	VARCHAR(100)	non nul, format email	E-mail de l'expéditeur
sent_date	TIMESTAMP	auto-généré	Date d'envoi
status	ENUM	valeurs : 'PENDING', 'READ', 'RESPONDED'	Statut du message
admin_id	FOREIGN KEY	vers User(id_user), optionnel	Administrateur ayant traité

12. Table Statistic

Nom du champ	Type	Contraintes	Description
id_statistic	UUID / SERIAL	PK, unique, non nul	Identifiant unique de la statistique
products_sold	INTEGER	≥ 0	Nombre de produits vendus
revenue	DECIMAL(12,2)	≥ 0	Chiffre d'affaires généré
period	DATE	non nul	Période concernée
category_id	FOREIGN KEY	vers Category(id_category), optionnel	Catégorie concernée (optionnel)

13. Table OAuthAccount

Nom du champ	Type	Contraintes	Description
id_oauth	UUID / SERIAL	PK, unique, non nul	Identifiant du compte OAuth
provider	ENUM	valeurs : 'GOOGLE', 'GITHUB', 'FACEBOOK'…	Fournisseur OAuth
provider_user_id	VARCHAR(255)	unique par provider	Identifiant de l'utilisateur chez le fournisseur
access_token	TEXT	optionnel	Jeton d'accès temporaire
refresh_token	TEXT	optionnel	Jeton de rafraîchissement
created_at	TIMESTAMP	auto-généré	Date de création du lien
user_id	FOREIGN KEY	vers User(id_user), non nul	Utilisateur associé

Tableaux SQL

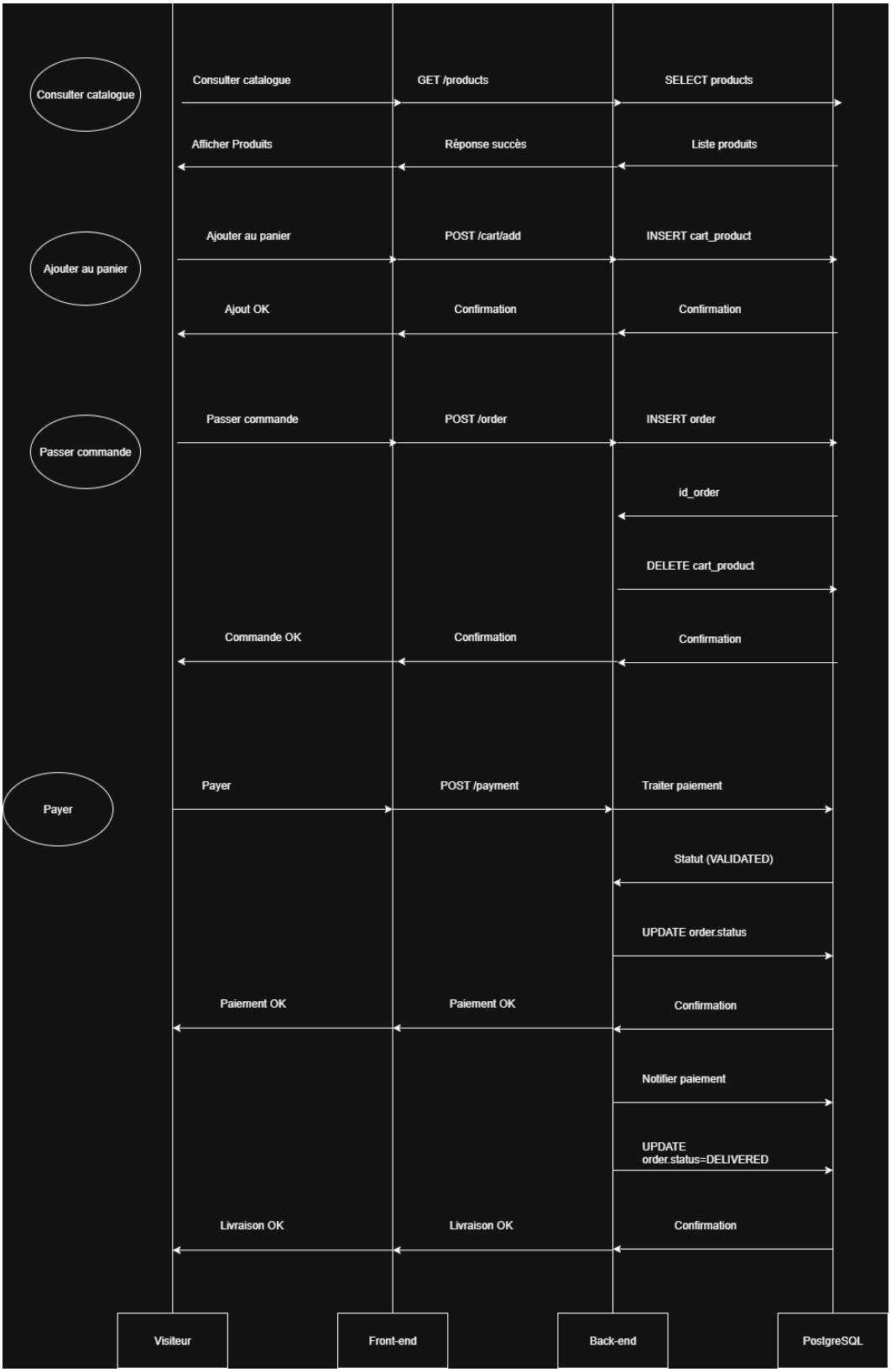
```
-- Création des types ENUM
CREATE TYPE role_type AS ENUM ('BUYER', 'SELLER', 'ADMIN');
CREATE TYPE condition_type AS ENUM ('NEW', 'GOOD', 'USED');
CREATE TYPE verification_status_type AS ENUM ('UNDER_VERIFICATION',
'RECONDITIONED', 'READY_TO_SELL', 'REJECTED');
CREATE TYPE listing_status_type AS ENUM ('PENDING', 'ONLINE', 'DELETED');
CREATE TYPE order_status_type AS ENUM ('PENDING','CANCELED', 'VALIDATED',
'DELIVERED');
CREATE TYPE transaction_status_type AS ENUM ('PENDING', 'VALIDATED',
'CANCELLED');
CREATE TYPE contact_status_type AS ENUM ('PENDING', 'READ', 'RESPONDED');
CREATE TYPE oauth_provider_type AS ENUM ('GOOGLE', 'GITHUB', 'FACEBOOK');
-- Table Users
CREATE TABLE users (
    id_user UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(100) NOT NULL UNIQUE,
    password TEXT ,
    name VARCHAR(100) NOT NULL,
    firstname VARCHAR(100) NOT NULL,
    adress VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL CHECK (phone ~ '^[0-9]{10}$'),
    role role_type NOT NULL DEFAULT 'BUYER',
    created_at TIMESTAMP DEFAULT NOW(),
    is_active BOOLEAN DEFAULT TRUE,
    last_activity TIMESTAMP
);

-- Table Category
CREATE TABLE category (
    id_category UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name_category VARCHAR(100) NOT NULL UNIQUE
);

-- Table Product
CREATE TABLE product (
    id_product UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
    condition condition_type NOT NULL,
```

```
    verification_status verification_status_type NOT NULL,
    images TEXT[],
    category_id UUID NOT NULL,
    FOREIGN KEY (category_id) REFERENCES category(id_category) ON DELETE
RESTRICT
);

-- Table Listing
CREATE TABLE listing (
    id_listing UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    publication_date TIMESTAMP DEFAULT NOW(),
    status listing_status_type NOT NULL,
    seller_id UUID NOT NULL,
    product_id UUID NOT NULL,
    FOREIGN KEY (seller_id) REFERENCES users(id_user) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES product(id_product) ON DELETE CASCADE
);

-- Table Orders (renommée de "order" à "orders")
CREATE TABLE orders (
    id_order UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_date TIMESTAMP DEFAULT NOW(),
    total_amount DECIMAL(10,2) NOT NULL CHECK (total_amount >= 0),
    status order_status_type NOT NULL,
    buyer_id UUID NOT NULL,
    FOREIGN KEY (buyer_id) REFERENCES users(id_user) ON DELETE CASCADE
);

-- Table Cart
CREATE TABLE cart (
    id_cart UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    buyer_id UUID NOT NULL,
    creation_date TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY (buyer_id) REFERENCES users(id_user) ON DELETE CASCADE
);

-- Table Cart_Listing (association)
CREATE TABLE cart_listing (
    id_cart UUID NOT NULL,
    id_listing UUID NOT NULL,
    quantity INTEGER NOT NULL CHECK (quantity >= 1),
    PRIMARY KEY (id_cart, id_listing),
```

```
    FOREIGN KEY (id_cart) REFERENCES cart(id_cart) ON DELETE CASCADE,
    FOREIGN KEY (id_listing) REFERENCES listing(id_listing) ON DELETE CASCADE
);

-- Table Transactions (renommée de "transaction" à "transactions")
CREATE TABLE transactions (
    id_transaction UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    transaction_date TIMESTAMP DEFAULT NOW(),
    amount DECIMAL(10,2) NOT NULL CHECK (amount >= 0),
    status transaction_status_type NOT NULL,
    buyer_id UUID NOT NULL,
    order_id UUID NOT NULL,
    FOREIGN KEY (buyer_id) REFERENCES users(id_user) ON DELETE CASCADE,
    FOREIGN KEY (order_id) REFERENCES orders(id_order) ON DELETE CASCADE
);
-- Table Order_Listing
CREATE TABLE order_Listing (
  order_id UUID NOT NULL,
  listing_id UUID NOT NULL,
  quantity INTEGER NOT NULL CHECK (quantity > 0),
  unit_price DECIMAL(10,2) NOT NULL CHECK (unit_price >= 0),

  PRIMARY KEY (order_id, listing_id),

  FOREIGN KEY (order_id) REFERENCES "orders"(id_order) ON DELETE CASCADE,
  FOREIGN KEY (listing_id) REFERENCES listing(id_listing) ON DELETE RESTRICT
);

-- Table Review
CREATE TABLE review (
    id_review UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    comment TEXT,
    rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
    review_date TIMESTAMP DEFAULT NOW(),
    buyer_id UUID NOT NULL,
    product_id UUID,
    seller_id UUID,
    FOREIGN KEY (buyer_id) REFERENCES users(id_user) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES product(id_product) ON DELETE SET NULL,
    FOREIGN KEY (seller_id) REFERENCES users(id_user) ON DELETE SET NULL
);
```

```
-- Table Contact
CREATE TABLE contact (
    id_contact UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    email VARCHAR(100) NOT NULL,
    sent_date TIMESTAMP DEFAULT NOW(),
    status contact_status_type NOT NULL,
    admin_id UUID,
    FOREIGN KEY (admin_id) REFERENCES users(id_user) ON DELETE SET NULL
);

-- Table OAuthAccount
CREATE TABLE oauth_account (
    id_oauth UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    provider oauth_provider_type NOT NULL,
    provider_user_id VARCHAR(255) NOT NULL,
    access_token TEXT,
    refresh_token TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    user_id UUID NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(id_user) ON DELETE CASCADE,
    UNIQUE (provider, provider_user_id)
);

-- Table Statistic
CREATE TABLE statistic (
    id_statistic UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    products_sold INTEGER NOT NULL CHECK (products_sold >= 0),
    revenue DECIMAL(12,2) NOT NULL CHECK (revenue >= 0),
    period DATE NOT NULL,
    category_id UUID,
    FOREIGN KEY (category_id) REFERENCES category(id_category) ON DELETE SET
NULL
);
```

Diagramme de Séquence :



Code source (extraits)

```
const AppRoutes = () => {      Assi Melvin, 4 months ago • Ajout propre sans clé Firebase
        {/* Public */}
        <Route path="/" element={<HomePage />} />
        <Route path="/signin" element={<AuthRoute><SignInPage /></AuthRoute>} />
        <Route path="/signup" element={<AuthRoute><SignUpPage /></AuthRoute>} />
        <Route path="/contact" element={<ContactPage />} />
        <Route path="/catalog" element={<CatalogPage />} />
        <Route path="/product/:id" element={<ProductPage />} />

        {/* Authenticated */}
        <Route path="/cart" element={<PrivateRoute><CartPage /></PrivateRoute>} />
        <Route path="/checkout" element={<PrivateRoute><CheckoutPage /></PrivateRoute>} />
        <Route path="/payment-success" element={<PaymentSuccessPage />} />
        <Route path="/payment-cancel" element={<PaymentCancelPage />} />

        <Route path="/dashbord" element={<PrivateRoute><DashboardPage /></PrivateRoute>} />
        <Route path="/user/info" element={<PrivateRoute><UserInfoPage /></PrivateRoute>} />
        <Route path="/user/listings" element={<PrivateRoute><UserListingsPage /></PrivateRoute>} />
        <Route path="/user/orders" element={<PrivateRoute><UserOrdersPage /></PrivateRoute>} />
        <Route path="/user/transactions" element={<PrivateRoute><UserTransactionsPage /></PrivateRoute>} />
        <Route path="/user/reviews" element={<PrivateRoute><UserReviewsPage /></PrivateRoute>} />

        <Route path="/user/products/:id" element={<PrivateRoute><ProductDetailsPage /></PrivateRoute>} />
        {/* Admin */}
        <Route path="/admin/users" element={<AdminRoute><UsersPage /></AdminRoute>} />
        <Route path="/admin/users/:id" element={<AdminRoute><UserDetailsPage /></AdminRoute>} />
        <Route path="/admin/products" element={<AdminRoute><ProductsPage /></AdminRoute>} />

        <Route path="/admin/listings" element={<AdminRoute><ListingsPage /></AdminRoute>} />
        <Route path="/admin/categories" element={<AdminRoute><CategoriesPage /></AdminRoute>} />
        <Route path="/admin/orders" element={<AdminRoute><OrdersPage /></AdminRoute>} />
        <Route path="/admin/transactions" element={<AdminRoute><TransactionsPage /></AdminRoute>} />
        <Route path="/admin/reviews" element={<AdminRoute><ReviewsPage /></AdminRoute>} />
        <Route path="/admin/contacts" element={<AdminRoute><ContactsPage /></AdminRoute>} />
        <Route path="/admin/stats" element={<AdminRoute><StatsPage /></AdminRoute>} />

        <Route path="/mentions-legales" element={<LegalNotice />} />
        <Route path="/cgv" element={<TermsAndConditions />} />
        <Route path="/confidentialite" element={<PrivacyPolicy />} />
    </Routes>
```