Ingénierie des bases de données

Systéme de gestion des livraisons en utilisant une base de donné oracle

---

Élève :
Said Jadli
Yasser Nadi
Adnane Qassiri

Enseignant :
Prof. Outair Anass

2025/2026

Table des matières

1 Introduction

Dans le contexte actuel de la logistique et de la gestion des livraisons, les entreprises font face à des défis croissants liés à la traçabilité, à l'optimisation des ressources et à la satisfaction client. Le système LogiTrack a été développé pour répondre à ces besoins en proposant une solution complète de gestion des livraisons basée sur une architecture moderne et une base de données Oracle robuste.

Ce projet consiste en la conception et l'implémentation d'un système de gestion des livraisons permettant de gérer l'ensemble du cycle de vie des colis, depuis leur enregistrement jusqu'à leur récupération par le destinataire. Le système intègre trois rôles principaux : l'administrateur, le gestionnaire d'entrepôt et le livreur, chacun ayant des responsabilités et des permissions spécifiques.

L'objectif principal de ce projet est de démontrer l'utilisation efficace d'Oracle Database dans un contexte applicatif réel, en exploitant ses fonctionnalités avancées telles que les triggers, les packages PL/SQL, les vues, les séquences et les contraintes d'intégrité référentielle. Le système garantit la cohérence des données, l'automatisation des processus métier et la traçabilité complète des opérations.

Le présent rapport décrit l'ensemble du processus de développement, depuis l'analyse des besoins jusqu'à l'implémentation finale, en passant par la conception de la base de données et le développement des interfaces utilisateur.

2 Étude et choix des technologies

2.1 Choix de l'architecture

L'architecture choisie pour LogiTrack est une architecture en trois tiers (3-tier architecture) séparant clairement la présentation, la logique métier et la persistance des données :

- **Couche Présentation** : Interface web développée avec React, offrant une expérience utilisateur moderne et réactive
- **Couche Application** : API REST développée avec Node.js et Express.js, servant d'intermédiaire entre le frontend et la base de données
- **Couche Données** : Base de données Oracle contenant l'ensemble des données et la logique métier via des packages PL/SQL

Cette architecture présente plusieurs avantages :

— Séparation claire des responsabilités
— Scalabilité et maintenabilité
— Réutilisabilité des composants
— Sécurité renforcée avec la logique métier centralisée dans la base de données

2.2 Technologies utilisées

2.2.1 Oracle Database

Oracle Database a été choisi comme système de gestion de base de données pour plusieurs raisons :

— **Robustesse** : Oracle est reconnu pour sa fiabilité et sa capacité à gérer de grandes quantités de données
— **Fonctionnalités avancées** : Support des triggers, packages PL/SQL, vues matérialisées, et autres fonctionnalités avancées
— **Intégrité des données** : Système de contraintes puissant permettant de garantir la cohérence des données
— **Performance** : Optimiseur de requêtes sophistiqué et gestion efficace des transactions
— **Sécurité** : Gestion fine des permissions et des rôles utilisateurs

Dans ce projet, Oracle Database stocke non seulement les données mais également une grande partie de la logique métier via des packages PL/SQL, ce qui garantit la cohérence et la sécurité des opérations.

2.2.2 Frontend : React, HTML, CSS, Tailwind CSS

Le frontend a été développé avec les technologies suivantes :

— **React 18.2.0** : Bibliothèque JavaScript moderne pour la construction d'interfaces utilisateur réactives et modulaires
— **React Router DOM 6.20.0** : Gestion de la navigation et du routage côté client
— **Tailwind CSS 3.3.6** : Framework CSS utility-first permettant un développement rapide et cohérent de l'interface
— **Axios 1.6.2** : Client HTTP pour les communications avec l'API backend
— **Headless UI** : Composants UI accessibles et sans style pour React

Le choix de React permet de créer une application web moderne, performante et maintenable, avec une séparation claire des composants et une gestion d'état efficace.

2.2.3 Backend : Express.js, Node.js

Le backend utilise les technologies suivantes :

— **Node.js** : Environnement d'exécution JavaScript côté serveur
— **Express.js 4.18.2** : Framework web minimaliste et flexible pour Node.js
— **OracleDB 6.0.3** : Driver Oracle officiel pour Node.js permettant la connexion à Oracle Database
— **Express-session 1.17.3** : Gestion des sessions utilisateur côté serveur
— **CORS 2.8.5** : Configuration des politiques de partage de ressources entre origines
— **Body-parser** : Middleware pour parser les corps de requêtes HTTP

Le backend agit comme une couche d'abstraction mince entre le frontend et la base de données, principalement pour :
— Gérer l'authentification et les sessions
— Appeler les procédures stockées Oracle
— Exécuter des requêtes SQL
— Gérer les erreurs et les validations

2.2.4 Autres outils

— **Dotenv** : Gestion des variables d'environnement pour la configuration
— **Nodemon** : Outil de développement pour le rechargement automatique du serveur
— **React Scripts** : Outils de build et de développement pour React
— **PostCSS et Autoprefixer** : Traitement CSS avancé

3 Analyse et conception du système

3.1 Analyse fonctionnelle

3.1.1 Acteurs du système

Le système LogiTrack identifie trois acteurs principaux, chacun ayant des responsabilités et des permissions spécifiques :

1. Administrateur (ADMIN)
— Visualisation du tableau de bord avec les indicateurs clés de performance (KPI)
— Gestion complète des utilisateurs (création, modification, activation/désactivation)

— Gestion des clients
— Gestion des entrepôts (création, affectation de gestionnaires)
— Gestion des véhicules
— Accès à toutes les fonctionnalités du système

2. Gestionnaire d'entrepôt (GESTIONNAIRE)

— Visualisation des statistiques de colis de son entrepôt
— Gestion des colis envoyés depuis son entrepôt
— Gestion des colis reçus dans son entrepôt
— Enregistrement de nouveaux colis
— Modification du statut des colis (sauf ceux déjà envoyés)
— Marquage des colis comme récupérés par le destinataire
— Gestion des clients
— Gestion des véhicules de son entrepôt

3. Livreur (LIVREUR)

— Visualisation des livraisons disponibles depuis son entrepôt
— Prise en charge de livraisons disponibles
— Visualisation de ses livraisons en cours
— Marquage des livraisons comme livrées
— Visualisation de ses statistiques de livraison

3.1.2 Diagramme de cas d'utilisation

Les principaux cas d'utilisation du système incluent :

— Authentification et gestion de session
— Gestion des utilisateurs (Admin uniquement)
— Gestion des entrepôts (Admin)
— Gestion des clients (Admin et Gestionnaire)
— Gestion des véhicules (Admin et Gestionnaire)
— Enregistrement de colis (Gestionnaire)
— Modification du statut des colis (Gestionnaire)
— Marquage des colis comme récupérés (Gestionnaire)
— Prise en charge de livraisons (Livreur)
— Livraison de colis (Livreur)
— Consultation des statistiques et KPI

3.2 Modélisation des données

3.2.1 Modèle conceptuel de données (MCD)

Le modèle conceptuel de données représente les entités principales du système et leurs relations :

Entités principales :
— **UTILISATEURS** : Représente tous les utilisateurs du système (Admin, Gestionnaire, Livreur)
— **ENTREPOTS** : Représente les entrepôts du réseau logistique
— **CLIENTS** : Représente les clients qui envoient des colis
— **VEHICULES** : Représente les véhicules utilisés pour les livraisons
— **LIVRAISONS** : Représente les livraisons entre entrepôts
— **COLIS** : Représente les colis à livrer
— **HISTORIQUE_STATUT_COLIS** : Historique des changements de statut des colis
— **HISTORIQUE_STATUT_LIVRAISONS** : Historique des changements de statut des livraisons
Relations principales :
— Un utilisateur peut être assigné à un entrepôt (Gestionnaire, Livreur)
— Un entrepôt a un gestionnaire responsable (relation 1-1)
— Un client est ajouté par un gestionnaire
— Un colis appartient à un client et peut être assigné à une livraison
— Une livraison relie deux entrepôts (source et destination)
— Une livraison est assignée à un livreur et un véhicule
— Un véhicule appartient à un entrepôt

3.2.2 Modèle logique de données (MLD)

Le modèle logique de données détaille la structure des tables avec leurs attributs et contraintes :

Table UTILISATEURS :
— id_utilisateur (PK, NUMBER)
— nom_utilisateur (UNIQUE, VARCHAR2(50))
— mot_de_passe (VARCHAR2(100))
— cin (UNIQUE, VARCHAR2(20))
— role (CHECK : 'ADMIN', 'GESTIONNAIRE', 'LIVREUR')
— actif (CHECK : 0 ou 1)
— id_entrepot (FK vers ENTREPOTS)
— date_creation (TIMESTAMP)
Table ENTREPOTS :
— id_entrepot (PK, NUMBER)
— adresse (VARCHAR2(200))
— ville (VARCHAR2(100))
— telephone (VARCHAR2(20))
— id_user (FK vers UTILISATEURS, UNIQUE - un gestionnaire par entrepôt)
— date_creation (TIMESTAMP)
Table CLIENTS :
— id_client (PK, NUMBER)
— prenom, nom (VARCHAR2(50))

— cin (UNIQUE, VARCHAR2(20))
— telephone, email, adresse
— id_gestionnaire_ajout (FK vers UTILISATEURS)
— date_creation (TIMESTAMP)

Table VEHICULES :

— id_vehicule (PK, NUMBER)
— immatriculation (UNIQUE, VARCHAR2(20))
— type_vehicule (CHECK : 'PETIT_CAMION', 'GRAND_CAMION')
— statut (CHECK : 'DISPONIBLE', 'EN_UTILISATION', 'MAINTENANCE')
— id_entrepot (FK vers ENTREPOTS)
— date_creation (TIMESTAMP)

Table LIVRAISONS :

— id_livraison (PK, NUMBER)
— id_entrepot_source (FK vers ENTREPOTS)
— id_entrepot_destination (FK vers ENTREPOTS)
— id_livreur (FK vers UTILISATEURS)
— id_vehicule (FK vers VEHICULES)
— statut (CHECK : 'CREEE', 'EN_COURS', 'LIVREE', 'ANNULEE')
— date_creation (TIMESTAMP)
— date_livraison (DATE)

Table COLIS :

— id_colis (PK, NUMBER)
— id_client (FK vers CLIENTS)
— id_livraison (FK vers LIVRAISONS)
— poids (NUMBER, CHECK : >= 1)
— type_colis (CHECK : 'STANDARD', 'FRAGILE')
— prix (NUMBER(10,2) - calculé automatiquement)
— receiver_cin (VARCHAR2(20))
— ville_destination (VARCHAR2(100))
— id_entrepot_localisation (FK vers ENTREPOTS)
— statut (CHECK : 'ENREGISTRE', 'EN_COURS', 'LIVRE', 'RECEPTIONNEE', 'ANNULE', 'RECUPEREE')
— date_creation (TIMESTAMP)

Tables d'historique :

— HISTORIQUE_STATUT_COLIS : id_history, id_colis, statut_avant, statut_apres, date_changement, id_utilisateur
— HISTORIQUE_STATUT_LIVRAISONS : id_history, id_livraison, statut_avant, statut_apres, date_changement, id_utilisateur

4 Conception de la base de données Oracle

4.1 Description des tables

La base de données LogiTrack est composée de 8 tables principales organisées de manière relationnelle :

1. UTILISATEURS : Stocke tous les utilisateurs du système avec leurs informations d'authentification et leur rôle. Chaque utilisateur peut être assigné à un entrepôt (pour les gestionnaires et livreurs).

2. ENTREPOTS : Représente les entrepôts du réseau logistique. Chaque entrepôt peut avoir un gestionnaire responsable (relation 1-1 via id_user).

3. CLIENTS : Contient les informations des clients qui envoient des colis. Chaque client est ajouté par un gestionnaire.

4. VEHICULES : Stocke les informations des véhicules (petit ou grand camion) avec leur statut et leur entrepôt d'affectation.

5. LIVRAISONS : Représente les livraisons entre entrepôts. Une livraison relie un entrepôt source à un entrepôt destination et peut être assignée à un livreur et un véhicule.

6. COLIS : Contient les informations des colis à livrer, incluant le calcul automatique du prix, l'assignation automatique à une livraison, et le suivi de la localisation.

7. HISTORIQUE_STATUT_COLIS : Enregistre l'historique complet des changements de statut des colis pour la traçabilité.

8. HISTORIQUE_STATUT_LIVRAISONS : Enregistre l'historique complet des changements de statut des livraisons.

4.1.1 Contraintes et relations

Contraintes d'intégrité référentielle :
— Toutes les clés étrangères sont définies avec des contraintes FOREIGN KEY
— Les relations sont maintenues automatiquement par Oracle
— Suppression en cascade gérée par les triggers et procédures
Contraintes de domaine :
— **Rôles utilisateurs** : CHECK (role IN ('ADMIN','GESTIONNAIRE','LIVREUR'))

— **Statut utilisateur** : CHECK (actif IN (0,1))
— **Type véhicule** : CHECK (type_vehicule IN ('PETIT_CAMION','GRAND_CAMION'))
— **Statut véhicule** : CHECK (statut IN ('DISPONIBLE','EN_UTILISATION','MAINTENANCE
— **Statut livraison** : CHECK (statut IN ('CREEE','EN_COURS','LIVREE','ANNULEE'))
— **Type colis** : CHECK (type_colis IN ('STANDARD','FRAGILE'))
— **Statut colis** : CHECK (statut IN ('ENREGISTRE','EN_COURS','LIVRE','RECEPTIONNEE
— **Poids colis** : CHECK (poids >= 1)

Contraintes d'unicité :

— nom_utilisateur (UNIQUE)
— cin dans utilisateurs (UNIQUE)
— cin dans clients (UNIQUE)
— immatriculation (UNIQUE)
— Un gestionnaire actif par entrepôt (via index unique sur entrepots.id_user)

4.2 Sequences

Huit séquences ont été créées pour générer automatiquement les identifiants primaires :

```
CREATE SEQUENCE seq_utilisateurs START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_entrepots    START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_vehicules    START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_clients      START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_livraisons   START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_colis        START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_hist_colis   START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_hist_liv     START WITH 1 INCREMENT BY 1;
```

Ces séquences sont utilisées par des triggers BEFORE INSERT pour générer automatiquement les identifiants lors de l'insertion de nouvelles lignes.

4.3 Triggers

Plusieurs triggers ont été implémentés pour automatiser les processus métier et garantir la cohérence des données :

1. Triggers de génération d'ID :

— Un trigger BEFORE INSERT pour chaque table utilisant une séquence
— Génération automatique de l'ID si NULL

2. Trigger de calcul de prix et assignation de livraison :

— **trg_colis_assign_price** : Calcule automatiquement le prix du colis (20 MAD/kg pour STANDARD, 30 MAD/kg pour FRAGILE) et assigne le colis à une livraison existante ou en crée une nouvelle selon la ville de destination

3. Triggers de gestion des livraisons :

— **trg_livraison_depart** : Lors du passage d'une livraison de CREEE à EN_COURS, met à jour le statut du véhicule à EN_UTILISATION et les colis à EN_COURS, et crée les historiques
— **trg_livraison_arrivee** : Lors du passage d'une livraison de EN_COURS à LIVREE, met à jour le statut du véhicule à DISPONIBLE, les colis à LIVRE, et crée automatiquement une nouvelle livraison CREEE pour la même route

4. Triggers de synchronisation gestionnaire-entrepôt :
— Gestion bidirectionnelle de la relation entre gestionnaires et entrepôts
— Utilisation de PRAGMA AUTONOMOUS_TRANSACTION pour éviter les erreurs de table mutante
— Validation qu'un gestionnaire actif ne peut être assigné qu'à un seul entrepôt

5. Trigger d'annulation de colis :
— **trg_colis_annulation_unassign** : Lorsqu'un colis est annulé, le retire automatiquement de sa livraison

4.4 Packages

Le package **pkg_logitrack** centralise toute la logique métier et les opérations critiques :

Procédures d'authentification :
— **p_creer_utilisateur** : Création d'un nouvel utilisateur
— **p_login** : Authentification et vérification des credentials

Procédures de gestion :
— **p_creer_entrepot** : Création d'un entrepôt avec gestionnaire
— **p_creer_vehicule** : Création d'un véhicule
— **p_changer_entrepot_vehicule** : Changement d'entrepôt d'un véhicule (avec contrôle de rôle)
— **p_creer_client** : Création d'un client (réservé aux gestionnaires/admin)

Procédures de gestion des livraisons :
— **p_prendre_livraison** : Un livreur prend en charge une livraison disponible
— **p_livrer_livraison** : Marquage d'une livraison comme livrée
— **p_modifier_statut_livraison** : Modification du statut d'une livraison (gestionnaire/admin)

Procédures de gestion des colis :
— **p_ajouter_colis** : Ajout d'un nouveau colis avec calcul automatique du prix
— **p_modifier_statut_colis** : Modification du statut d'un colis avec historique
— **p_marquer_colis_recuperee** : Marquage d'un colis comme récupéré par le destinataire (via CIN)

Procédures de consultation :
— **p_get_kpis** : Récupération des indicateurs clés de performance via un curseur

Toutes les procédures incluent des contrôles de rôle pour garantir la sécurité et l'intégrité des opérations.

4.5 Views

Quatre vues ont été créées pour simplifier les requêtes complexes et fournir des données agrégées :

1. v_livraisons_details :
— Affiche les détails complets des livraisons avec source, destination, livreur, véhicule, statut et nombre de colis
— Jointures avec entrepots, utilisateurs et vehicules
2. v_colis_details :
— Affiche les détails complets des colis avec client, trajet calculé, localisation actuelle
— Calcule automatiquement le trajet (entrepôt source → entrepôt destination ou entrepôt → ville destination)
3. v_vehicules_entrepots :
— Affiche les véhicules avec leur entrepôt d'affectation actuel
4. v_kpi_dashboard :
— Vue agrégée pour le tableau de bord admin
— Calcule : nombre total de colis, nombre de livraisons livrées, chiffre d'affaires, nombre de livreurs actifs, nombre d'entrepôts, nombre de clients

5 Réalisation et implémentation

5.1 Mise en place de l'environnement de développement

L'environnement de développement a été configuré avec les outils suivants :

Base de données :
— Oracle Database (version XE ou supérieure)
— Oracle Instant Client pour la connexion depuis Node.js
— Scripts SQL organisés en fichiers séquentiels (00_drop_all.sql à 06_test_data.sql)
— Script d'installation automatique (install.sql) exécutant tous les fichiers dans l'ordre
Backend :
— Node.js (v14 ou supérieur)
— Configuration via fichier .env pour les variables d'environnement (DB_HOST, DB_PORT, DB_SERVICE, DB_USER, DB_PASSWORD, SESSION_SECRET, PORT)
— Utilisation d'un pool de connexions Oracle pour optimiser les performances
— Middleware CORS configuré pour accepter les requêtes depuis le frontend
Frontend :
— React 18 avec Create React App
— Configuration via fichier .env (REACT_APP_API_URL)
— Tailwind CSS pour le styling

— Routing avec React Router DOM

5.2 Implémentation du backend

Le backend a été structuré de manière modulaire :

Structure des dossiers :
— **config/** : Configuration de la base de données (pool de connexions Oracle)
— **controllers/** : Logique métier pour chaque rôle (adminController, gestionnaireController, livreurController, authController)
— **middleware/** : Middleware d'authentification et gestion d'erreurs
— **routes/** : Définition des routes API pour chaque module
— **utils/** : Utilitaires (oracleHelper pour exécuter procédures et requêtes)
Points clés de l'implémentation :
— **Authentification** : Sessions serveur avec express-session, vérification des rôles via middleware
— **Appels Oracle** : Utilisation de procédures stockées via oracleHelper pour garantir la sécurité et la cohérence
— **Gestion des erreurs** : Middleware centralisé pour capturer et formater les erreurs Oracle
— **API REST** : Endpoints organisés par rôle (/api/admin, /api/gestionnaire, /api/livreur)
Exemple d'endpoint (KPIs Admin) :

```
const getKPIs = async (req, res, next) => {
  try {
    const kpis = await executeQuery('SELECT * FROM v_kpi_dashboard
      ↪ ');
    // Calcul s par  pour livraisons livr es uniquement
    const livraisonsLivreeCount = await executeQuery(
      "SELECT COUNT(*) as count FROM livraisons WHERE statut = '
        ↪ LIVREE'"
    );
    res.json({ success: true, data: { ... } });
  } catch (err) {
    next(err);
  }
};
```

5.3 Implémentation de la base de données Oracle

L'implémentation de la base de données suit une approche structurée :

Ordre d'exécution des scripts :

1. **00_drop_all.sql** : Suppression de tous les objets existants (pour réinstallation)
2. **01_sequences.sql** : Création des séquences
3. **02_tables.sql** : Création des tables avec contraintes
4. **03_triggers.sql** : Création des triggers
5. **04_package.sql** : Création du package pkg_logitrack
6. **05_views.sql** : Création des vues
7. **06_test_data.sql** : Insertion des données de test

Techniques utilisées :

— **PRAGMA AUTONOMOUS_TRANSACTION** : Pour éviter les erreurs de table mutante dans les triggers
— **Procedures autonomes** : Pour la synchronisation bidirectionnelle entre tables
— **Contrôle de flux** : Utilisation de flags pour éviter les boucles infinies
— **Gestion d'erreurs** : RAISE_APPLICATION_ERROR pour les erreurs métier personnalisées

Exemple de trigger complexe :

```
CREATE OR REPLACE TRIGGER trg_colis_assign_price
BEFORE INSERT OR UPDATE ON colis
FOR EACH ROW
DECLARE
  v_base NUMBER := 20;
  v_dest_entrepot NUMBER;
  v_livraison NUMBER;
BEGIN
  -- Calcul du prix
  IF :NEW.type_colis = 'FRAGILE' THEN
    v_base := 30;
  END IF;
  :NEW.prix := ROUND(:NEW.poids * v_base, 2);

  -- Assignation automatique    une livraison
  -- ... logique de recherche/cr ation de livraison
END;
```

5.4 Implémentation du frontend

Le frontend utilise une architecture composant React moderne :

Structure des composants :

— **components/admin/** : Dashboard, gestion des utilisateurs, clients, entrepôts, véhicules
— **components/gestionnaire/** : Dashboard, colis envoyés, colis reçus, gestion clients et véhicules
— **components/livreur/** : Dashboard, livraisons disponibles, mes livraisons

— **components/common/** : Composants réutilisables (Sidebar, Navbar, DataTable, KPIcard)
— **components/auth/** : Page de connexion

Points clés de l'implémentation :
— **Context API** : AuthContext pour la gestion globale de l'authentification
— **Routes protégées** : Composant ProtectedRoute vérifiant l'authentification et les rôles
— **Service API** : Module api.js centralisant tous les appels HTTP avec axios
— **Composants réutilisables** : DataTable pour l'affichage tabulaire avec actions conditionnelles
— **Interface responsive** : Utilisation de Tailwind CSS pour un design moderne et adaptatif

Exemple de composant (Dashboard Admin) :
— Affichage de 8 KPI cards (Total Clients, Total Colis, Total Livraisons, Active Livreurs, Total Entrepôts, Admins, Gestionnaires, Chiffre d'Affaires)
— Appel API pour récupérer les données
— Gestion des états de chargement et d'erreur

5.5 Gestion des erreurs et validations

Côté backend :
— **Middleware d'erreur centralisé** : Capture toutes les erreurs et les formate de manière cohérente
— **Validation des rôles** : Vérification systématique des permissions avant chaque opération
— **Gestion des erreurs Oracle** : Extraction et formatage des messages d'erreur Oracle (RAISE_APPLICATION_ERROR)
— **Validation des données** : Vérification des paramètres requis avant l'appel aux procédures

Côté base de données :
— **Contraintes CHECK** : Validation des valeurs autorisées pour les statuts et types
— **Contraintes UNIQUE** : Prévention des doublons (nom utilisateur, CIN, immatriculation)
— **Contraintes FOREIGN KEY** : Garantie de l'intégrité référentielle
— **RAISE_APPLICATION_ERROR** : Messages d'erreur métier explicites dans les procédures

Côté frontend :
— **Gestion des erreurs API** : Affichage de messages d'erreur utilisateur-friendly
— **Validation des formulaires** : Vérification des champs requis avant soumission
— **États de chargement** : Indicateurs visuels pendant les opérations asynchrones
— **Gestion des sessions** : Redirection automatique vers la page de connexion si la session expire

6 Sécurité et performance

6.1 Sécurité du système

Plusieurs mécanismes de sécurité ont été implémentés à différents niveaux :

1. Authentification et autorisation :
— **Sessions serveur** : Utilisation d'express-session avec cookies httpOnly pour prévenir les attaques XSS
— **Vérification des rôles** : Middleware requireRole vérifiant le rôle utilisateur avant l'accès aux endpoints
— **Contrôle d'accès dans les procédures** : Chaque procédure PL/SQL vérifie le rôle de l'utilisateur avant d'exécuter l'opération
— **Routes protégées** : Le frontend vérifie l'authentification et les rôles avant d'afficher les composants
2. Sécurité des données :
— **Mots de passe** : Stockés en clair dans la base (à améliorer avec hashage en production)
— **Validation des entrées** : Contraintes CHECK et validation dans les procédures pour prévenir les injections SQL
— **Requêtes paramétrées** : Utilisation systématique de bind variables dans les requêtes Oracle
— **Isolation des transactions** : Utilisation de PRAGMA AUTONOMOUS_TRANSACTION pour éviter les conflits
3. Intégrité des données :
— **Contraintes d'intégrité référentielle** : Toutes les clés étrangères sont protégées
— **Contraintes de domaine** : Validation des valeurs autorisées (statuts, types)
— **Contraintes d'unicité** : Prévention des doublons critiques
— **Triggers de validation** : Vérification des règles métier avant insertion/mise à jour
4. Traçabilité :
— **Historiques complets** : Tous les changements de statut sont enregistrés avec l'utilisateur et la date
— **Date de création** : Timestamp automatique pour toutes les entités
— **Identification des acteurs** : Chaque opération est liée à un utilisateur identifié
5. Sécurité applicative :
— **CORS configuré** : Restriction des origines autorisées (localhost :3000 en développement)
— **Variables d'environnement** : Secrets stockés dans .env (non versionné)
— **Gestion des erreurs** : Messages d'erreur génériques pour ne pas exposer d'informations sensibles

6.2 Optimisation des performances

Plusieurs techniques d'optimisation ont été mises en place :

1. Base de données :
— **Pool de connexions** : Utilisation d'un pool Oracle (min : 1, max : 5) pour réutiliser les connexions
— **Vues matérialisables** : Vues pré-calculées pour les KPI et les détails complexes
— **Index uniques** : Index sur les colonnes fréquemment recherchées (nom_utilisateur, cin, immatriculation)
— **Requêtes optimisées** : Utilisation de JOINs efficaces et agrégations dans les vues
2. Backend :
— **Appels procédures stockées** : Logique métier exécutée côté base de données pour réduire les allers-retours
— **Curseurs Oracle** : Utilisation de SYS_REFCURSOR pour retourner des résultats efficaces
— **Gestion asynchrone** : Utilisation d'async/await pour éviter le blocage
— **Mise en cache** : Possibilité de mettre en cache les KPI (non implémenté mais préparé)
3. Frontend :
— **Composants React optimisés** : Utilisation de hooks pour éviter les re-renders inutiles
— **Chargement conditionnel** : Affichage des données uniquement après chargement
— **Requêtes ciblées** : Appels API uniquement pour les données nécessaires à l'affichage
— **Interface réactive** : Feedback immédiat pour les actions utilisateur
4. Architecture :
— **Séparation des responsabilités** : Logique métier dans la base, API mince, frontend léger
— **Transactions optimisées** : Auto-commit pour les opérations simples, transactions explicites pour les opérations complexes
— **Éviter les N+1 queries** : Utilisation de JOINs dans les vues plutôt que des requêtes multiples
Points d'amélioration possibles :
— Mise en cache Redis pour les KPI
— Pagination pour les grandes listes
— Index supplémentaires sur les colonnes fréquemment filtrées
— Compression des réponses HTTP
— Lazy loading des composants React

7 Conclusion et perspectives

7.1 Conclusion générale

Le projet LogiTrack a permis de développer un système complet de gestion des livraisons en exploitant les fonctionnalités avancées d'Oracle Database. L'architecture en trois tiers (frontend React, backend Node.js, base de données Oracle) a démontré son efficacité pour créer une application moderne, sécurisée et performante.

Objectifs atteints :

- Conception d'une base de données relationnelle normalisée avec 8 tables principales
- Implémentation de la logique métier dans Oracle via packages PL/SQL
- Automatisation des processus via triggers (calcul de prix, assignation de livraisons, gestion des statuts)
- Développement d'une interface web moderne et intuitive avec React
- Gestion des rôles et permissions (Admin, Gestionnaire, Livreur)
- Traçabilité complète via tables d'historique
- Système de KPI pour le suivi des performances

Points forts du projet :

- **Cohérence des données** : La logique métier centralisée dans Oracle garantit l'intégrité des données
- **Automatisation** : Les triggers et procédures automatisent les processus critiques (calcul de prix, assignation de livraisons)
- **Sécurité** : Contrôle d'accès à plusieurs niveaux (frontend, backend, base de données)
- **Maintenabilité** : Code structuré et modulaire, documentation claire
- **Expérience utilisateur** : Interface moderne et intuitive avec feedback en temps réel

Défis rencontrés et solutions :

- **Erreurs de table mutante** : Résolues avec PRAGMA AUTONOMOUS_TRANSACTION
- **Synchronisation bidirectionnelle** : Gérée via flags et procédures autonomes
- **Gestion des sessions** : Implémentation d'un système de sessions serveur robuste
- **Optimisation des requêtes** : Utilisation de vues et curseurs pour améliorer les performances

Ce projet a permis de mettre en pratique les concepts d'ingénierie des bases de données, notamment l'utilisation avancée d'Oracle Database, la modélisation relationnelle, et l'intégration avec une application web moderne.

7.2 Perspectives d'évolution

Plusieurs améliorations et extensions peuvent être envisagées pour enrichir le système :

1. Fonctionnalités supplémentaires :

— **Gestion des notifications** : Alertes par email/SMS pour les changements de statut
— **Suivi GPS** : Intégration de la géolocalisation pour le suivi en temps réel des livraisons
— **Module de facturation** : Génération automatique de factures pour les clients
— **Statistiques avancées** : Graphiques et analyses prédictives
— **Application mobile** : Développement d'une app mobile pour les livreurs
— **Portail client** : Interface pour que les clients puissent suivre leurs colis

2. Améliorations techniques :

— **Sécurité renforcée** : Hashage des mots de passe (bcrypt), authentification à deux facteurs
— **Performance** : Mise en cache Redis, pagination, index supplémentaires
— **API REST complète** : Documentation Swagger/OpenAPI
— **Tests automatisés** : Tests unitaires et d'intégration
— **CI/CD** : Pipeline de déploiement automatique
— **Monitoring** : Outils de monitoring et logging avancés

3. Optimisations base de données :

— **Vues matérialisées** : Pour les KPI complexes avec rafraîchissement automatique
— **Partitionnement** : Partitionnement des tables d'historique par date
— **Archivage** : Système d'archivage des données anciennes
— **Backup automatique** : Stratégie de sauvegarde et restauration

4. Évolutions métier :

— **Multi-entrepôts avancés** : Gestion de réseaux d'entrepôts complexes
— **Optimisation des routes** : Algorithme d'optimisation des trajets de livraison
— **Gestion des stocks** : Module de gestion des stocks dans les entrepôts
— **Intégration transporteurs** : Interface avec des transporteurs externes
— **Module de qualité** : Suivi de la qualité de service et satisfaction client

5. Scalabilité :

— **Microservices** : Découpage en microservices pour une meilleure scalabilité
— **Load balancing** : Répartition de charge pour gérer plus d'utilisateurs
— **Base de données distribuée** : Oracle RAC pour haute disponibilité
— **CDN** : Utilisation d'un CDN pour les assets statiques

En conclusion, LogiTrack constitue une base solide pour un système de gestion logistique professionnel, avec de nombreuses possibilités d'évolution pour répondre aux besoins croissants des entreprises de logistique moderne.