ColisExpress

Plateforme collaborative de livraison interurbaine

Documentation de conception & Plan de réalisation

Version 1.0 — Mai 2025

Projet	ColisExpress
Stack	NestJS · React · PostgreSQL
Équipe	3–4 développeurs
Méthode	Agile Scrum — Sprints 2 semaines
Déploiement	Vercel + Azure + CI/CD GitLab

1. Contexte & Objectifs du projet

1.1 Problématique

Les solutions traditionnelles de livraison de colis présentent des limites importantes : coûts élevés, délais longs, et manque de flexibilité pour les envois interurbains entre particuliers. Simultanément, des milliers de personnes effectuent quotidiennement des trajets interurbains avec une capacité de transport inutilisée.

1.2 Solution proposée

ColisExpress est une plateforme web collaborative qui met en relation des expéditeurs de colis avec des voyageurs effectuant déjà le trajet souhaité - sur le modèle du covoiturage appliqué à la livraison.

1.3 Objectifs

- Réduire les coûts de livraison pour les expéditeurs
- Accélérer les délais grâce à des trajets existants
- Permettre aux voyageurs de monétiser leur espace disponible
- Favoriser une économie collaborative basée sur la confiance
- Offrir un suivi en temps réel des expéditions

1.4 Acteurs du système

Acteur	Description	Accès
Visiteur	Utilisateur non authentifié	Consultation des trajets publics
Expéditeur	Rôle par défaut après inscription	Recherche, réservation, paiement, suivi
Voyageur	Expéditeur ayant validé son identité	Publication de trajets + rôle expéditeur
Administrateur	Équipe ColisExpress	Gestion complète de la plateforme

2. User Stories & Backlog

2.1 Authentification & Gestion des utilisateurs

ID	En tant que…	Je veux…	Priorité
US-01	Visiteur	M'inscrire via Google ou Facebook (OAuth)	MVP
US-02	Utilisateur	Me connecter avec JWT sécurisé	MVP
US-03	Utilisateur	Gérer mon profil (photo, nom, contact)	MVP
US-04	Expéditeur	Devenir voyageur en soumettant permis + immatriculation	MVP
US-05	Utilisateur	Me déconnecter et invalider mon token	MVP

2.2 Gestion des annonces de voyage

ID	En tant que…	Je veux…	Priorité
US-06	Voyageur	Publier un trajet (départ, arrivée, date, capacité)	MVP
US-07	Voyageur	Modifier ou supprimer mon annonce	MVP
US-08	Voyageur	Consulter mes trajets publiés	MVP
US-09	Voyageur	Voir les demandes d'expédition sur mon trajet	MVP
US-10	Voyageur	Accepter ou refuser une demande d'expédition	MVP

2.3 Gestion des expéditions

ID	En tant que…	Je veux…	Priorité
US-11	Expéditeur	Rechercher des trajets par ville et date	MVP
US-12	Expéditeur	Filtrer les trajets par poids/volume accepté	MVP
US-13	Expéditeur	Réserver un trajet et décrire mon colis	MVP
US-14	Expéditeur	Payer via crédits sur la plateforme (Stripe)	MVP
US-15	Expéditeur	Consulter l'historique de mes expéditions	MVP

2.4 Suivi & Évaluation

ID	En tant que…	Je veux…	Priorité
US-16	Voyageur	Mettre à jour le statut du colis manuellement	MVP

US-17	Expéditeur	Voir le statut de mon colis en temps réel	MVP
US-18	Expéditeur	Évaluer le voyageur après livraison	MVP
US-19	Voyageur	Évaluer l'expéditeur après livraison	MVP
US-20	Expéditeur	Voir la position GPS du voyageur (avancé)	Avancé

3. Modèle de données — Schéma PostgreSQL

3.1 Entités principales

Table	Champs clés	Relations
users	id, email, name, avatar_url, role[], credits, rating_avg, created_at	1:N trips, 1:N shipments, 1:N reviews
user_verifications	id, user_id, license_number, plate_number, status, submitted_at	N:1 users
trips	id, traveler_id, origin, destination, departure_at, capacity_kg, capacity_m3, price_per_kg, status	N:1 users, 1:N shipments
shipments	id, sender_id, trip_id, description, weight_kg, volume_m3, status, tracking_code, credits_paid	N:1 users, N:1 trips
tracking_events	id, shipment_id, status, note, lat, lng, created_at	N:1 shipments
reviews	id, reviewer_id, reviewee_id, shipment_id, rating, comment, created_at	N:1 users, N:1 shipments
credit_transactions	id, user_id, amount, type (buy/earn/spend), ref_id, created_at	N:1 users

3.2 Diagramme Entité-Relation (ERD) — Représentation textuelle

```
users ──< user_verifications users ──< trips          (traveler_id) users ──< shipments
(sender_id) trips ──< shipments shipments ──< tracking_events shipments ──< reviews
(reviewer_id, reviewee_id → users) users ──< credit_transactions
```

3.3 Énumérations (ENUM PostgreSQL)

Enum	Valeurs
user_role	SENDER, TRAVELER, ADMIN
verification_status	PENDING, APPROVED, REJECTED
trip_status	OPEN, FULL, IN_PROGRESS, COMPLETED, CANCELLED
shipment_status	PENDING, ACCEPTED, PICKED_UP, IN_TRANSIT, DELIVERED, CANCELLED
transaction_type	PURCHASE, EARNING, SPENDING, REFUND

4. Architecture technique

4.1 Vue d'ensemble — Architecture 3-tiers

Couche	Technologie	Hébergement	Rôle
Présentation (Frontend)	React 18 + Vite + Tailwind CSS	Vercel	UI/UX, routing, état global
Logique métier (Backend)	NestJS (TypeScript) + TypeORM	Azure App Service	API REST, auth, règles métier
Données	PostgreSQL 15	Azure Database for PostgreSQL	Persistance, relations, ACID
Cache / Temps réel	Redis	Azure Cache for Redis	Sessions, WebSocket events
Fichiers	AWS S3 / Azure Blob	Cloud	Permis, immatriculation, photos

4.2 Structure du projet NestJS

```
src/ ├── auth/          # Module auth (OAuth, JWT, guards) ├── users/         # Profils, vérifications, crédits ├── trips/         # Annonces de voyage ├── shipments/     # Expéditions, réservation ├── tracking/      # Mise à jour statut, WebSocket GPS ├── reviews/ # Système d'évaluation ├── payments/      # Intégration Stripe ├── notifications/ # Emails, push notifications ├── common/        # Guards, pipes, interceptors, DTOs └── database/ # Entités TypeORM, migrations
```

4.3 Structure du projet React

```
src/ ├── pages/         # Pages (Home, Login, Dashboard, Trip, Shipment...) ├── components/ # Composants réutilisables (Card, Map, Badge...) ├── hooks/         # Hooks personnalisés (useAuth, useTrips...) ├── stores/        # Zustand — état global ├── services/      # Appels API (axios instances) ├── types/         # Interfaces TypeScript partagées └── utils/ # Helpers, formatters
```

4.4 Catalogue des endpoints REST

Méthode	Endpoint	Auth	Description
POST	/auth/google	Non	Login OAuth Google → JWT
POST	/auth/facebook	Non	Login OAuth Facebook → JWT
POST	/auth/refresh	Non	Renouveler le JWT
POST	/auth/logout	JWT	Invalider le token

GET	/users/me	JWT	Profil de l'utilisateur connecté
PATCH	/users/me	JWT	Mettre à jour le profil
POST	/users/me/verify	JWT	Soumettre permis + immatriculation
GET	/users/:id	JWT	Profil public d'un utilisateur
GET	/trips	Non	Lister les trajets (filtres: origin, dest, date)
POST	/trips	Voyageur	Créer un trajet
GET	/trips/:id	Non	Détail d'un trajet
PATCH	/trips/:id	Voyageur	Modifier son trajet
DELETE	/trips/:id	Voyageur	Supprimer son trajet
GET	/trips/me	Voyageur	Mes trajets publiés
GET	/shipments	JWT	Mes expéditions
POST	/shipments	JWT	Réserver une expédition
GET	/shipments/:id	JWT	Détail d'une expédition
PATCH	/shipments/:id/status	Voyageur	Mettre à jour le statut
POST	/shipments/:id/accept	Voyageur	Accepter une demande
POST	/shipments/:id/cancel	JWT	Annuler une expédition
GET	/shipments/:id/tracking	JWT	Historique de tracking
POST	/shipments/:id/tracking	Voyageur	Ajouter un événement
POST	/reviews	JWT	Créer une évaluation
GET	/users/:id/reviews	Non	Évaluations d'un utilisateur
POST	/payments/credits	JWT	Acheter des crédits (Stripe)
GET	/payments/history	JWT	Historique des transactions
GET	/admin/verifications	Admin	Vérifications en attente
PATCH	/admin/verifications/:id	Admin	Approuver / Rejeter
GET	/admin/users	Admin	Gestion des utilisateurs

5. Infrastructure & CI/CD

5.1 Pipeline GitHub Actions

```
Push / PR → main      │      ├─ [CI] Lint & Type Check (ESLint + tsc --noEmit)      ├─ [CI]
Tests unitaires (Jest)      ├─ [CI] Tests d'intégration (Supertest + BDD test)      ├─ [CI]
Build Frontend (Vite build)      ├─ [CI] Build Backend (nest build)      │      └─ [CD] si
branche main & tous les CI passent            ├─ Deploy Frontend → Vercel            └─
Deploy Backend  → Azure App Service
```

5.2 Environnements

Env	Branche Git	URL	DB
Development	feature/*	localhost	PostgreSQL local (Docker)
Staging	develop	staging.colisexpress.ca	Azure DB — staging
Production	main	colisexpress.ca	Azure DB — production

5.3 Docker Compose (développement local)

```
services:   postgres:     image: postgres:15-alpine     ports: ['5432:5432']     env:
POSTGRES_DB=colisexpress   redis:     image: redis:7-alpine     ports: ['6379:6379']
backend:     build: ./backend     ports: ['3000:3000']     depends_on: [postgres, redis]
frontend:     build: ./frontend     ports: ['5173:5173']
```

5.4 Variables d'environnement (.env)

Variable	Description
DATABASE_URL	URL de connexion PostgreSQL
JWT_SECRET	Clé secrète pour signer les JWT
JWT_EXPIRES_IN	Durée de vie du token (ex: 7d)
GOOGLE_CLIENT_ID / SECRET	OAuth Google
FACEBOOK_CLIENT_ID / SECRET	OAuth Facebook
STRIPE_SECRET_KEY	Clé API Stripe
STRIPE_WEBHOOK_SECRET	Signature des webhooks Stripe
REDIS_URL	URL de connexion Redis
GOOGLE_MAPS_API_KEY	Géocodage et carte interactive

AWS_S3_BUCKET / KEY / SECRET	Stockage fichiers (permis, photos)
FRONTEND_URL	URL du frontend (CORS)

6. Plan de réalisation — Sprints Agile

Chaque sprint dure 2 semaines maximum. Chaque feature est automatiquement testée (unitaire + intégration) et déployée via le pipeline CI/CD à la fin du sprint. La progression est incrémentale : chaque sprint construit sur le précédent.

6.0 Phase de mise en place (avant Sprint 1 — 1 semaine)

Tâche	Responsable	Détail
Création des dépôts Git	Tout le monde	Monorepo ou repos séparés frontend/backend
Init NestJS + TypeORM + PostgreSQL	Backend	Connexion DB, migrations, .env
Init React + Vite + Tailwind	Frontend	Router, Zustand, Axios config
Docker Compose local	Backend	postgres, redis, backend, frontend
GitHub Actions CI/CD skeleton	DevOps	Lint → Test → Build → Deploy (stubs)
Design system & palette	Frontend	Couleurs, typographie, composants de base

Sprint 1 — Authentification & Gestion des utilisateurs

Objectif : Un utilisateur peut s'inscrire, se connecter (OAuth) et gérer son profil.

Backend

- Entity User + migrations PostgreSQL
- Module Auth : Passport.js + GoogleStrategy + FacebookStrategy
- Génération et validation JWT (access + refresh token)
- Guard JwtAuthGuard global
- Endpoints: POST /auth/google, /auth/facebook, /auth/refresh, /auth/logout
- Endpoints: GET/PATCH /users/me, GET /users/:id
- Tests unitaires AuthService, UsersService
- Tests d'intégration e2e auth flow

Frontend

- Page Login avec boutons OAuth Google + Facebook
- Redirect post-login vers Dashboard
- Contexte Auth global (Zustand) — stockage JWT
- Axios interceptor : attache Bearer token + refresh auto
- Page Profil : affichage et édition (nom, avatar, bio)
- ProtectedRoute HOC — redirection si non authentifié
- Tests composants Login, Profil (Vitest + Testing Library)

✅ **Livrables & CI/CD :** Auth fonctionnelle end-to-end. Pipeline CI vert. Déploiement staging automatique.

Sprint 2 — Vérification Voyageur & Gestion des rôles

Objectif : Un expéditeur peut soumettre une demande de vérification pour devenir voyageur. Un admin peut l'approuver.

Backend

- Entity UserVerification + migrations
- Endpoint POST /users/me/verify (upload permis + plaque via S3)
- Service de stockage fichiers (AWS S3 ou Azure Blob)
- Guard RolesGuard + décorateur @Roles()
- Module Admin : GET /admin/verifications, PATCH /admin/verifications/:id
- Email notification (approbation / refus) via Nodemailer
- Tests unitaires VerificationService

Frontend

- Page 'Devenir Voyageur' : formulaire upload permis + immatriculation
- Indicateur statut vérification dans le profil (PENDING/APPROVED/REJECTED)
- Dashboard Admin : liste des vérifications en attente
- Actions admin : Approuver / Rejeter avec commentaire
- Badge 'Voyageur Vérifié' visible sur les profils
- Tests composants formulaire vérification

✓**Livrables & CI/CD :** Système de rôles opérationnel. Upload fichiers fonctionnel. Admin dashboard déployé.

Sprint 3 — Gestion des annonces de voyage

Objectif : Un voyageur vérifié peut publier, modifier et gérer ses annonces de trajet.

Backend

- Entity Trip + migrations (origin, destination, departure_at, capacity, price_per_kg)
- Intégration Google Places API pour autocomplétion villes
- CRUD Endpoints trips (POST, GET, PATCH, DELETE, GET /trips/me)
- Filtres de recherche : origin, destination, date, capacité
- Validation DTO (class-validator) : dates futures, capacités positives
- Enum TripStatus + logique de changement d'état
- Tests unitaires + intégration TripsModule

Frontend

- Page 'Publier un trajet' : formulaire avec autocomplétion Google Places
- Sélecteur date/heure, champs capacité kg/m³, prix par kg
- Page 'Mes trajets' : liste avec statuts colorés, actions modifier/supprimer
- Composant TripCard réutilisable
- Page de recherche de trajets : filtres et résultats en temps réel
- Pagination des résultats
- Tests Vitest TripForm, TripCard

✓**Livrables & CI/CD :** Annonces de voyage créées et consultables. Recherche fonctionnelle. Déployé en staging.

Sprint 4 — Gestion des expéditions & Paiement par crédits

Objectif : Un expéditeur peut réserver un trajet et payer via le système de crédits Stripe.

Backend

- Entities Shipment + CreditTransaction + migrations
- Intégration Stripe : achat de crédits (PaymentIntent), webhook confirmation

Frontend

- Page 'Acheter des crédits' : intégration Stripe Elements
- Affichage solde de crédits dans le header
- Page détail d'un trajet avec bouton 'Expédier un colis'

- Endpoint POST /payments/credits — création session Stripe
- Logique débit de crédits lors d'une réservation
- Endpoints shipments : POST /shipments, GET /shipments, GET /shipments/:id
- Endpoint POST /shipments/:id/accept (voyageur)
- Endpoint POST /shipments/:id/cancel + logique remboursement crédits
- Tests unitaires PaymentService, ShipmentService
- Tests intégration Stripe webhook (mock)

- Modal de réservation : description colis, poids, volume, confirmation coût
- Page 'Mes expéditions' : liste avec statuts
- Page 'Mes demandes' (voyageur) : accepter / refuser
- Historique des transactions de crédits
- Tests Vitest PaymentForm, ShipmentBooking

✅**Livrables & CI/CD :** Réservation et paiement end-to-end fonctionnels. Webhooks Stripe opérationnels. CI/CD vert.

Sprint 5 — Suivi des expéditions (Tracking)

Objectif : Un voyageur peut mettre à jour le statut du colis. L'expéditeur suit en temps réel.

Backend

- Entity TrackingEvent + migrations
- Endpoints: GET /shipments/:id/tracking, POST /shipments/:id/tracking
- Intégration WebSocket (Socket.IO via NestJS Gateway) pour mises à jour temps réel
- Enum ShipmentStatus avec transitions validées (state machine)
- Intégration optionnelle GPS : endpoint PATCH /shipments/:id/location
- Tests unitaires TrackingService + Gateway

Frontend

- Page suivi colis : timeline des événements avec icônes et timestamps
- Connexion WebSocket client — mise à jour automatique sans refresh
- Carte Google Maps affichant les étapes du trajet
- Badge de statut animé (PENDING → PICKED_UP → IN_TRANSIT → DELIVERED)
- Notifications in-app lors d'un changement de statut
- Vue voyageur : boutons de mise à jour statut rapide
- Tests Vitest TrackingTimeline

✅**Livrables & CI/CD :** Tracking temps réel via WebSocket. Carte fonctionnelle. Feature complète déployée.

Sprint 6 — Système d'évaluation & Finalisation

Objectif : Les deux parties s'évaluent après livraison. Polissage UI, sécurité et performance.

Backend

- Entity Review + migrations
- Endpoints: POST /reviews, GET /users/:id/reviews

Frontend

- Modal d'évaluation post-livraison : étoiles 1-5 + commentaire
- Affichage des évaluations sur le profil public (note moyenne + liste)

- Calcul automatique rating_avg sur la table users (trigger ou hook TypeORM)
- Rate limiting (Throttler NestJS) sur les endpoints sensibles
- Helmet.js — headers de sécurité HTTP
- Audit des logs (Winston) — requêtes + erreurs
- Tests de charge basiques (k6 ou Artillery)
- Documentation API Swagger (OpenAPI) auto-générée

- Page Accueil publique : hero section, comment ça marche, stats
- Responsive mobile (breakpoints Tailwind)
- Gestion des erreurs globale : toasts, pages 404/500
- Optimisation des performances : lazy loading, code splitting
- Tests Vitest ReviewModal, ProfileRating
- Tests e2e Playwright : flow complet réservation → livraison → évaluation

✅**Livrables & CI/CD :** MVP complet. API Swagger disponible. Tests e2e passants. Déploiement production.

7. Conventions & Standards

7.1 Convention de nommage

Contexte	Convention	Exemple
Variables / fonctions	camelCase	getUserById()
Classes / Interfaces / DTOs	PascalCase	CreateShipmentDto
Fichiers TypeScript	kebab-case	create-shipment.dto.ts
Tables PostgreSQL	snake_case	credit_transactions
Colonnes PostgreSQL	snake_case	created_at, user_id
Endpoints REST	kebab-case pluriel	/shipments, /tracking-events
Branches Git	feature/nom-feature	feature/auth-oauth
Commits	Conventional Commits	feat(auth): add Google OAuth login
Variables d'env	SCREAMING_SNAKE_CASE	JWT_SECRET, DATABASE_URL

7.2 Standards de code

- Clean Code : fonctions courtes (< 20 lignes), noms explicites
- Principes SOLID respectés dans les services NestJS
- DTOs avec class-validator pour toute entrée utilisateur
- Pas de any TypeScript — typage strict activé
- Commentaires JSDoc sur les méthodes publiques des services
- Coverage de tests : objectif ≥ 70%

7.3 Workflow Git (GitHub Flow)

- main : branche de production — toujours déployable
- develop : branche d'intégration — déployée en staging
- feature/* : une branche par feature / US
- PR obligatoire avec revue de code avant merge dans develop
- Squash merge pour garder un historique propre