

Documentation Technique — Logistics Chatbot & Dashboard (Ultimate+)

Documentation Technique — Logistics Chatbot & Dashboard (Ultimate+)

****Version**** : Ultimate+ (JWT/Rôles, WebSocket, Excel & CSV, Alertes SMTP/Teams, Webhook transporteur, OIDC SSO scaffold, Postgres par défaut, Docker, Tests, Timeline Frontend)

****Références de téléchargements : **** - Starter : `logistics-chatbot-dashboard.zip` - Full : `logistics-chatbot-dashboard-full.zip` - Ultimate : `logistics-chatbot-dashboard-ultimate.zip` - ****Ultimate+ (recommandé)**** : `logistics-chatbot-dashboard-ultimate-plus.zip`

> Cette documentation couvre l'architecture complète, le modèle de données, les endpoints, la sécurité, la configuration (env), le déploiement Docker/Postgres, les rapports (CSV/Excel), le WebSocket live, le webhook transporteur, le scheduler d'alertes, les tests, l'interface Frontend (Dashboard + Chatbot + Timeline), et l'SSO OIDC (scaffold).

1. Architecture globale

****Objectif**** : piloter un flux logistique «production → chargement → scellé → export → transit → arrivée port NY/NJ → import → livraison», fournir un ****dashboard temps réel****, une ****interface Chatbot**** (« Où en est mon envoi ? »), des ****alertes proactives****, et des ****exports**** (CSV/Excel).

1.1 Pile technique

- ****Backend**** : FastAPI (Python 3.11), SQLAlchemy 2.x (ORM), Pydantic v2, APScheduler, JWT (PyJWT), Passlib (bcrypt) - ****Sécurité**** : JWT Bearer + rôles (`client`, `ops`, `admin`), clé API simple (`X-API-Key`) pour POST, secret webhook - ****Base de données**** : ****Postgres**** par défaut (SQLite possible en dev rapide) - ****Frontend**** : Next.js 14, React 18, Tailwind CSS - ****Temps réel**** : WebSocket `/ws/shipments` (push événements → maj dashboard) - ****Exports**** : CSV & ****Excel (openpyxl)**** - ****Alertes**** : SLA/étapes critiques ; stubs ****SMTP**** & ****Microsoft Teams**** - ****SSO OIDC**** : Authlib (scaffold login/callback) - ****Conteneurisation**** : Dockerfiles (front/back) + `docker-compose.yml` - ****Tests**** : pytest (sanity & flux de base)

1.2 Structure des fichiers (principaux)

```
``` backend/ app/ __init__.py main.py # API, routes, WebSocket, rapports, webhook models.py # ORM: User, Shipment, Event, Enum EventType schemas.py # Schémas Pydantic (IO) database.py # Base, engine, SessionLocal auth.py # Hash, JWT (création/décodage) security.py # Dépendances: Bearer, rôles, X-API-Key alerts.py # Règles d'alertes + SMTP/Teams stubs scheduler.py # Démarrage APScheduler live.py # Gestion des connexions WebSocket & broadcast reports.py # Génération Excel (openpyxl) sso.py # OIDC (login/callback) via Authlib seed.py # Données d'exemple (optionnel) requirements.txt requirements-dev.txt tests/ test_core.py frontend/ pages/ _app.js index.js # Dashboard + Chatbot page shipment/[id].js # Timeline détaillée d'un envoi components/ ShipmentsTable.js # Table + lien → /shipment/[id] + WebSocket live Chatbot.js styles/, config Tailwind docker-compose.yml # Postgres par défaut + backend + frontend ```
```

---

### 2. Modèle de données & Entités

## 2.1 `User`

- `id`: int` (PK) - `email`: str` (unique, index) - `name`: str | None` - `role`: str` (valeurs : `client`, `ops`, `admin`) - `password\_hash`: str` (bcrypt) - `created\_at`: datetime`

## 2.2 `Shipment` (envoi)

- `id`: int` (PK) - `reference`: str` (PO/référence interne, index) - `customer`: str` - `origin`: str` / `destination`: str` - `incoterm`: str` (par défaut `FOB`) - `planned\_etd`: datetime | None` - `planned\_eta`: datetime | None` - `container\_number`: str | None` (numéro conteneur) - `seal\_number`: str | None` (scellé) - `sku`: str | None` (PO/SKU produit) - `quantity`: int | None` - `weight\_kg`: float | None` / `volume\_cbm`: float | None` - `status`: str` (statut courant = dernier `Event.type`) - `created\_at`: datetime` - **Relation** : `events`: List[Event]` (cascade delete)

## 2.3 `Event` (jalón)

- `id`: int` (PK) - `shipment\_id`: int` (FK → Shipment) - `type`: EventType` (enum) - `timestamp`: datetime` (par défaut `now()`) - `payload`: JSON | None` (données brutes : GPS, preuve photo, doc URLs...) - `note`: str | None` - `critical`: bool` (faciliter les alertes)

## 2.4 `EventType` (enum des jalons)

- **Production/Chargement/Scellé/Export/Transit/Arrivée/Import/Livraison** :  
`PRODUCTION\_READY`, `LOADING\_IN\_PROGRESS`, `SEAL\_NUMBER\_CUTOFF`,  
`EXPORT\_CLEARANCE\_CAMBODIA`, `TRANSIT\_OCEAN`, `ARRIVAL\_PORT\_NYNJ`,  
`IMPORT\_CLEARANCE\_CBP`, `FINAL\_DELIVERY`, ... - **Opérationnels** : `ORDER\_INFO`,  
`CONTAINER\_READY\_FOR\_DEPARTURE`, `PHOTOS\_CONTAINER\_WEIGHT`,  
`GPS\_POSITION\_ETA\_ETD`, `UNLOADING\_GATE\_OUT`, `CUSTOMS\_STATUS\_DECLARATION`,  
`UNLOADING\_TIME\_CHECKS` - **Système** : `LOGISTICS\_DB\_UPDATE`, `CHATBOT\_QUERY`,  
`REALTIME\_DASHBOARD`, `PROACTIVE\_ALERT`, `REPORTING\_ANALYTICS`, `USERS\_CLIENT`,  
`USERS\_LOGISTICS`

---

# 3. Sécurité & Authentification

## 3.1 JWT (Bearer)

- **Login** : `POST /auth/login` → retourne `{ "access\_token": "<JWT>", "token\_type": "bearer" }` - Stocker côté navigateur : `localStorage.setItem('token', '<JWT>')` - Les **routes GET** (lecture) exigent `Authorization: Bearer <JWT>`. - Les **routes POST** (écriture) exigent **ET** `Authorization: Bearer <JWT>` **ET** `X-API-Key: <clé>`.

## 3.2 Rôles

- `client` : lecture des ressources (par défaut) - `ops` : écriture opérationnelle (création envoi, push événements, rapports, etc.) - `admin` : peut créer des utilisateurs via `/auth/register` (limité aux admins)

## 3.3 Clé API & Webhook secret

- **Clé API** : header `X-API-Key` pour `POST /shipments` & `POST /events` - **Webhook** : `x-carrier-secret` sur `POST /webhooks/carrier` (normalisation d'événements)

## 3.4 OIDC (SSO) — Scaffold

- **Routes** : `/auth/oidc/login` → `/auth/oidc/callback` - Variables : `OIDC\_CLIENT\_ID`, `OIDC\_CLIENT\_SECRET`, `OIDC\_DISCOVERY\_URL`, `OIDC\_REDIRECT\_URL`,

`FRONTEND\_BASE\_URL` - Comportement : crée/MAJ un `User` local à partir de l'email OIDC puis **génère un JWT** et **redirige** vers le Front (`/#token=<JWT>`) - Si non configuré : `503 OIDC not configured`

---

## 4. API — Endpoints principaux

Base : `http://localhost:8000`

### 4.1 Santé

- `GET /health` → `{ "status": "ok" }`

### 4.2 Authentification

- `POST /auth/login` — **JWT** Body : ``json

{ "email": "admin@example.com", "password": "ChangeMe123!" } `` - `POST /auth/register` — **admin uniquement** (créer un user avec rôle) - `GET /auth/oidc/login` — redirection OIDC (si configuré) - `GET /auth/oidc/callback` — callback OIDC → redirection Front

### 4.3 Envois (Shipments)

- `POST /shipments` — **protégé** (rôle `ops` + `X-API-Key`) Exemple : ``json { "reference": "25COA003", "customer": "L'Oréal", "origin": "Sihanoukville, KH", "destination": "NY/NJ, US", "incoterm": "FOB", "container\_number": "MSKU1234567", "planned\_etd": "2025-10-12T08:00:00Z", "planned\_eta": "2025-11-16T08:00:00Z" } `` - `GET /shipments` — **protégé** (JWT) : liste des envois - `GET /shipments/{id}` — **protégé** (JWT) : détail

### 4.4 Événements (Events)

- `POST /events` — **protégé** (rôle `ops` + `X-API-Key`) - Met à jour `Shipment.status` = `type` - Récupère `payload.seal` si présent pour `seal\_number` - **Broadcast WebSocket** d'un message `event\_created` - `GET /shipments/{id}/events` — **protégé** (JWT) : liste des events (desc)

### 4.5 Chatbot

- `POST /chat` — **protégé** (JWT) - Cherche par **conteneur**, sinon **reference**, sinon **dernier envoi** - Répond : `Dernier statut pour <ref>: <EventType> @ <timestamp>`

### 4.6 Rapports

- `GET /reports/shipments.csv` — **protégé** (rôle `ops`) - `GET /reports/shipments.xlsx` — **protégé** (rôle `ops`) (Excel via openpyxl)

### 4.7 Webhook Transporteur

- `POST /webhooks/carrier` — **protégé** (secret `x-carrier-secret`) Normalise `{container, milestone, timestamp, ...}` → `EventType` via un mapping : - `PRODUCTION` → `PRODUCTION\_READY` - `LOADING` → `LOADING\_IN\_PROGRESS` - `GATE\_OUT` → `UNLOADING\_GATE\_OUT` - `ARRIVAL\_NYNJ` → `ARRIVAL\_PORT\_NYNJ` - `IN\_TRANSIT` → `TRANSIT\_OCEAN` - `EXPORT\_CLEAR` → `EXPORT\_CLEARANCE\_CAMBODIA` - défaut → `LOGISTICS\_DB\_UPDATE`

---

## 5. Temps réel (WebSocket)

- **Endpoint** : `GET /ws/shipments` - Frontend ouvre la socket au chargement du dashboard.  
À chaque `POST /events`, le backend **broadcast** un message : ``json  
{ "type": "event\_created", "shipment\_id": <id>, "event\_id": <id>, "status": "<EventType>" } `` - Le  
front appelle `GET /shipments` pour relancer la table.

---

## 6. Alertes & Scheduler

- **APScheduler** démarre au lancement du backend. - **Règles** (exemples ; faciles à enrichir) : 1) **Transit > SLA** (`planned\_eta < now - ALERT\_SLA\_HOURS`) → alerte 2)  
**Gate-out sans scellé** (événement `UNLOADING\_GATE\_OUT` existe mais `seal\_number`  
manquant) → alerte - **Sorties** : - **Log** (toujours) + stubs **SMTP** (TLS) + **Teams  
webhook** (env) - Variables : `ALERT\_EMAILS`, `ALERT\_SLA\_HOURS`, `SMTP\_\*`,  
`TEAMS\_WEBHOOK`

---

## 7. Frontend (Next.js)

### 7.1 Dashboard

- `components/ShipmentsTable.js` — lecture via `/shipments` (JWT), **abonné WebSocket** -  
Lien sur la **Réf.** → `/shipment/[id]` (Timeline)

### 7.2 Chatbot

- `components/Chatbot.js` — `POST /chat` (JWT), résultat affiché en JSON

### 7.3 Timeline

- `pages/shipment/[id].js` — charge `GET /shipments/{id}` et `GET /shipments/{id}/events`,  
affiche les jalons en frise verticale.

---

## 8. Exports (CSV / Excel)

- **CSV** : `GET /reports/shipments.csv` - **Excel** : `GET /reports/shipments.xlsx` (openpyxl) -  
Colonnes : `id, reference, customer, origin, destination, incoterm, planned\_etd, planned\_eta,  
container, seal, sku, qty, weight\_kg, volume\_cbm, status, created\_at`

---

## 9. Configuration (ENV)

Backend (exemples) : ``

### BDD

SQLALCHEMY\_DATABASE\_URL=postgresql+psycopg://app:app@postgres:5432/logistics

### JWT

JWT\_SECRET=change-me JWT\_EXP\_MIN=60

### Alertes

ALERT\_EMAILS=ops@example.com,client@example.com ALERT\_SLA\_HOURS=240 SMTP\_HOST=

SMTP\_PORT= SMTP\_USER= SMTP\_PASS= TEAMS\_WEBHOOK=

## Sécurité intégrations

CARRIER\_SHARED\_SECRET=devsecret

## OIDC SSO

OIDC\_CLIENT\_ID= OIDC\_CLIENT\_SECRET= OIDC\_DISCOVERY\_URL=

OIDC\_REDIRECT\_URL=http://localhost:8000/auth/oidc/callback

FRONTEND\_BASE\_URL=http://localhost:3000 ```

Frontend : ``` NEXT\_PUBLIC\_API\_BASE=http://backend:8000 ```

---

## 10. Déploiement (Docker compose)

`docker-compose.yml` (résumé) : - **postgres** (image `postgres:16`) - **backend** (expose 8000, dépend de postgres) - **frontend** (expose 3000, dépend de backend)

Lancement : ```bash docker compose up --build

Backend: http://localhost:8000/docs

Frontend: http://localhost:3000

```

11. Tests & Postman

- **Pytest** : `backend/tests/test_core.py` (health → create → event → chat) ```bash cd backend
pip install -r requirements-dev.txt pytest ``` - **Collection Postman** :

`backend/postman_collection.json`

12. Exemples cURL

12.1 Login (JWT)

```bash curl -X POST http://localhost:8000/auth/login \ -H "Content-Type: application/json" \ -d  
'{"email":"admin@example.com","password":"ChangeMe123!"}' ```

### 12.2 Créer un envoi (ops + X-API-Key)

```bash curl -X POST http://localhost:8000/shipments \ -H "Authorization: Bearer <JWT>" \ -H  
"X-API-Key: dev" -H "Content-Type: application/json" \ -d
'{"reference":"25COA003","customer":"L'Oréal","origin":"Sihanoukville,
KH","destination":"NY/NJ, US","incoterm":"FOB","container_number":"MSKU1234567"}' ```

12.3 Pousser un event (ops + X-API-Key)

```bash curl -X POST http://localhost:8000/events \ -H "Authorization: Bearer <JWT>" \ -H  
"X-API-Key: dev" -H "Content-Type: application/json" \ -d  
'{"shipment\_id":1,"type":"SEAL\_NUMBER\_CUTOFF","payload":{"seal":"KH123456"}}' ```

## 12.4 Webhook transporteur

```
```bash curl -X POST http://localhost:8000/webhooks/carrier \ -H "x-carrier-secret: devsecret" -H
"Content-Type: application/json" \ -d
'{"container":"MSKU1234567","milestone":"IN_TRANSIT","timestamp":"2025-10-12T12:00:00Z"}'
```
```

## 12.5 Rapports

```
```bash
```

CSV

```
curl -L -o shipments.csv http://localhost:8000/reports/shipments.csv -H "Authorization: Bearer
<JWT>"
```

Excel

```
curl -L -o shipments.xlsx http://localhost:8000/reports/shipments.xlsx -H "Authorization: Bearer
<JWT>" ```
```

13. Bonnes pratiques & extensions

- **Idempotence** webhook : envoyer un identifiant d'événement unique (hash) dans `payload` ; côté API, ignorer si déjà existant (ajout facile). - **Pièces jointes** : stocker sur S3/Blob ; mettre les **URLs** dans `payload`. - **Migrations** : utiliser **Alembic** (guide dans README) pour versions prod. - **RBAC avancé** : filtrage par `customer`, `lane`, `brand` côté backend. - **Monitoring** : ajouter Prometheus/Grafana + logs JSON structurés. - **ETA prédictive** : intégrer AIS/carrier API et recalculer ETA dynamique.

14. Roadmap suggérée

1) Ajout **filtres UI** (client, lane, statut) + export CSV filtré 2) **Timeline enrichie** (icônes jalons, pièces jointes, code couleur SLA) 3) **Dashboard client** (lecture seule, périmètre restreint) 4) **SSO** complet (OIDC) + gestion de rôles depuis IdP 5) **Alertes** par canal : Teams/Email/SMS et règles par **client/produit**

15. FAQ rapide

- **Pourquoi une clé `X-API-Key` et un JWT ?** → Le JWT gère **qui** (utilisateur/role). La clé API contrôle **quelles intégrations** peuvent écrire (sécurité opérationnelle). - **Postgres obligatoire ?** → Par défaut oui dans le compose. SQLite reste possible en dev local rapide. - **Comment créer l'admin au départ ?** → via script Python dans le conteneur backend (voir messages précédents) ou exposer `/auth/register` temporairement.

Fin du document — Ultimate+