DAT-X-Zone

Contents

1 X-Zone — Document d'Architecture Technique (DAT)

Date : 25 janvier 2026
Version du document : 1.0.0 (référence : `info.version` dans `Backend/xzone-openapi.yaml`)
Auteur : Non spécifié dans le dépôt (document produit par analyse statique du code)
Équipe projet : Global Glimpse SARL (référence : `frontend/README.md`)
Niveau de confidentialité : À renseigner (non spécifié dans le dépôt)

1.1 1. Historique et gestion du document

Version	Date	Auteur	Description
1.0.0	25/01/2026	Non spécifié (généré)	Première version basée sur le dépôt (Backend + frontend).

1.2 2. Résumé exécutif

1.2.1 2.1 Contexte du projet

Le dépôt contient une solution applicative structurée en deux ensembles :

- `Backend/` : application **Laravel 12** incluant à la fois une **UI backoffice** via **Inertia + React** (sources dans `Backend/resources/js`) et une **API HTTP** (routes dans `Backend/routes/api.php`).
- `frontend/` : application **React + Vite** (SPA avec React Router) consommant l'API via `/api` et accédant aux médias via `/storage`.

1.2.2 2.2 Objectifs métiers (observables dans les routes, modèles, migrations)

- Exposer un **catalogue produits** (produits, catégories, documents, avis, promotions, blog).
- Couvrir un **cycle de vente** (devis, commandes, factures) et des fonctions backoffice (stocks, finance, RH, helpdesk).
- Supporter une **vente web** (checkout "Cash On Delivery", suivi de commande public).

- Proposer un **portail client** (auth token via Sanctum, endpoints profil/devis/factures/commandes/tickets/documents/configurateur).

1.2.3 2.3 Enjeux techniques

- Séparer les responsabilités entre backoffice (Inertia) et site public (SPA).
- Définir des conventions API (public vs client vs v1 backoffice), et appliquer des protections (throttling, auth, RBAC).
- Garantir l'intégrité des opérations métiers via transactions et validation.
- Industrialiser le déploiement via scripts et procédures documentées.

1.2.4 2.4 Contraintes identifiées (issues du code et des docs)

- Auth API portail client : implémentée côté backend (Sanctum). La SPA `frontend/` ne montre pas, dans les fichiers inspectés, de gestion du token (stockage et injection `Authorization: Bearer`).
- Clé IA : `frontend/vite.config.ts` injecte `GEMINI_API_KEY` au code frontend via `define`, ce qui rend la clé accessible côté client.
- Durcissement CSP : le middleware `Backend/app/Http/Middleware/SecurityHeaders.php` indique éviter une CSP pour ne pas casser l'UI Inertia.
- Dans cette copie de workspace, `Backend/config/` ne contient que `web_orders.php` (les configs Laravel standards ne sont pas visibles ici). Les constats "configuration" s'appuient donc sur `Backend/.env.example`, `Backend/phpunit.xml` et `Backend/docs/*`.

1.2.5 2.5 Bénéfices de l'architecture retenue

- Architecture éprouvée (Laravel MVC, Eloquent, Inertia) pour le backoffice.
- API structurée par usages (public / portail client / v1 backoffice) et documentée (OpenAPI + `Backend/docs/api-v1.md`).
- Mesures de protection déjà présentes : rate limiting, RBAC, headers, validation, transactions.

1.3 3. Présentation générale de l'application

1.3.1 3.1 Finalité de l'application

Fournir une plateforme X-Zone couvrant :

- un site public (présentation + catalogue + parcours commande + suivi),
- un backoffice d'administration/gestion,
- une API supportant les interactions des différents canaux,
- un portail client (API token).

1.3.2 3.2 Périmètre fonctionnel (dérivé de `Backend/routes/*` et `Backend/database/migrations/*`)

- Catalogue : produits, catégories, attributs, images, variantes, compatibilités, documents.
- Ventes : clients, devis, factures, commandes, statuts/historiques.
- Commandes web : tables dédiées `web_orders*`, endpoint public `/api/orders`.
- Contenu : blog/actualités.
- Promotions.
- Stocks : mouvements, raisons, pièces jointes.
- Finance : transactions, catégories, rappels.
- RH / congés.
- Helpdesk/tickets.
- Logs/audit : activity log + login logs.

1.3.3 3.3 Hors périmètre

Non explicitement défini dans le dépôt. Les éléments suivants ne sont pas présents en tant que composants identifiables/industrialisés dans le code :

- Supervision/monitoring (agents, dashboards, alerting) : non trouvé.
- Backups et plan de restauration : non trouvé.
- Pipeline CI/CD : non trouvé.

1.3.4 3.4 Typologie des utilisateurs (observée via RBAC, routes et API)

- Utilisateurs backoffice authentifiés + email vérifié (`auth` + `verified` sur routes web).
- Rôles/permissions : gestion via Spatie Permission (middleware `permission:*` sur routes web). Bypass `SuperAdmin` via `Gate::before`.
- Clients portail : utilisateurs avec flag `portal_client` (middleware `portal_client`), auth Sanctum token.
- Public : consultation catalogue/blog, soumission avis, création commande web, suivi commande.

1.3.5 3.5 Environnements cibles

- Développement local : `.env` conforme à `Backend/.env.example` ; proxy Vite pour le frontend (`frontend/.env.example`).
- Test : `Backend/phpunit.xml` fixe `APP_ENV=testing` + sqlite in-memory.
- Recette : non explicitement définie dans le dépôt.
- Production : procédure dans `Backend/docs/production.md`.

1.4 4. Architecture globale

1.4.1 4.1 Vue d'ensemble Frontend / Backend

Le dépôt implémente deux applications frontend distinctes :

1) Backoffice : UI Inertia/React servie par `Backend/` (rendu via routes web Laravel).
2) Site public : SPA React dans `frontend/`.

Les deux consomment (directement ou indirectement) les services du backend : DB, storage, queue, mail.

1.4.2 4.2 Principe de séparation des couches

- Couche Présentation (Backend) : contrôleurs HTTP, middlewares, validation (FormRequest), ressources JSON.
- Couche Métier : `Backend/app/Services`, `Backend/app/Actions`, `Backend/app/Domain`.
- Couche Accès aux données : modèles Eloquent, relations, scopes, migrations.

1.4.3 4.3 Protocoles et formats d'échange

- HTTP/HTTPS.
- JSON pour l'API (`Accept: application/json` côté client `frontend/utils/apiClient.ts`).
- Inertia (JSON + navigation côté client) pour le backoffice.
- Stockage fichiers : exposition via `/storage/*` (symlink `public/storage`).

1.4.4 4.4 Schéma logique des flux applicatifs (description textuelle)

Flux "site public" (SPA) :

1. Le navigateur charge la SPA `frontend/`.
2. La SPA appelle `/api/...` (via `fetch` et base URL `VITE_API_BASE_URL`, par défaut `/api`).
3. En dev, Vite proxifie `/api` et `/storage` vers `VITE_API_PROXY_TARGET` (Laravel).
4. Laravel traite la requête (routes, middlewares, validation, contrôleur), interagit avec la DB et/ou le storage.
5. Laravel renvoie JSON ; la SPA gère loading/erreurs et rend les vues.

Flux "backoffice" (Inertia) :

1. Le navigateur accède aux routes web Laravel.
2. Middlewares web (dont `EnsureAppIsConfigured`, `HandleInertiaRequests`, `SecurityHeaders`) exécutent.
3. Le contrôleur renvoie une réponse Inertia ; la UI React `Backend/resources/js` rend les pages.

4. Les rôles/permissions et paramètres partagés sont injectés via `HandleInertiaRequests`.

1.4.5 4.5 Principes d'architecture retenus

- Backend : architecture **Laravel MVC** (contrôleurs, modèles, requests, resources), enrichie par des services/actions.
- UI backoffice : pattern **Inertia** (rendu SPA piloté par le serveur, pages React).
- API : segmentation par contexte (public / client / v1 backoffice) + documentation (OpenAPI + doc dédiée v1).

1.5 5. Architecture Backend (détaillée)

1.5.1 5.1 Stack technique complète (référence `Backend/composer.json`)

- PHP `^8.2`.
- Laravel `^12.0`.
- ORM : Eloquent.
- Auth API : Laravel Sanctum.
- RBAC : Spatie Permission.
- Audit : Spatie Activity Log.
- PDF : `barryvdh/laravel-dompdf`.
- Exports : `maatwebsite/excel`.
- Images : `intervention/image`.

1.5.2 5.2 Organisation des dossiers (structure observée)

- `Backend/app/Http/Controllers/` : contrôleurs web et API.
- `Backend/app/Http/Controllers/Api/` : API publique.
- `Backend/app/Http/Controllers/Api/Client/` : API portail client.
- `Backend/app/Http/Controllers/Api/V1/` : API v1 backoffice.
- `Backend/app/Http/Middleware/` : middlewares (onboarding, sécurité headers, Inertia, portail client).
- `Backend/app/Models/` : modèles (relations, scopes, casts, soft deletes, audit).
- `Backend/app/Services/` : services métiers (ex : congés, promotions, compatibilités).
- `Backend/app/Actions/` : actions unitaires (ex : conversion devis → facture).
- `Backend/database/migrations/` : schéma DB.
- `Backend/resources/js/` : UI backoffice (Inertia + React) + SSR.

1.5.3 5.3 Description des couches

Couche	Responsabilités	Exemples d'artefacts
Présentation	Routage, middlewares, validation, sérialisation, statuts HTTP	`routes/api.php`, `routes/web.php`, `app/Http/Middleware/*`, `app/Http/Requests/*`, `app/Http/Resources/*`, contrôleurs
Métier	Règles et cas d'usage, orchestration, intégrité	`app/Services/*`, `app/Actions/*`, `app/Domain/*`
Accès aux données	Modèles, relations, requêtes, migrations	`app/Models/*`, `database/migrations/*`

1.5.4 5.4 Flux d'exécution d'une requête (exemple "création commande web")

Endpoint : `POST /api/orders` (route dans `Backend/routes/api.php`, contrôleur `Backend/app/Http/Controllers/Api/WebOrderController.php`).

Chaîne d'exécution :

1. Router Laravel → contrôleur.
2. Validation via `StoreWebOrderRequest` (`$request->validated()`).
3. Chargement des produits + règles (refus des produits "prix sur devis").
4. Transaction `DB::transaction` : création `WebOrder` + `WebOrderItem`, calcul HT/TVA/TTC, snapshot des champs produit.
5. Envoi emails en "best effort" (try/catch) ; log warning si échec.
6. Réponse JSON : `{ success: true, order: {...} }`.

1.5.5 5.5 Gestion des transactions

L'usage de `DB::transaction` est présent dans plusieurs flux critiques, notamment :

- commandes web (API),
- changements de statuts (backoffice),
- tickets, catégories/attributs,
- promotions / devis / factures,
- congés (services Leave).

Objectif : garantir l'atomicité des écritures multi-tables et maintenir la cohérence des agrégats (totaux, historiques, pivots).

1.5.6 5.6 Accès aux données et ORM

- Eloquent Models avec :
  - `SoftDeletes` (ex : `Product`).

  - `HasUuids` sur certaines entités (ex : `Product`).
  - relations `belongsTo`, `hasMany`, `belongsToMany`.
  - scopes (ex : `Product::visible()`) pour centraliser les règles de visibilité.
- Schéma DB maintenu via migrations (source de vérité).

1.5.7 5.7 Gestion des erreurs

- Gestion des 404 de modèles : `ModelNotFoundException` rendu en JSON (dans `Backend/bootstrap/app.php`) lorsque la requête attend du JSON ou cible `api/*`.
- Gestion d'erreurs applicatives via statuts HTTP explicites sur certains endpoints :
  - 401/403 pour login portail client,
  - 404 pour suivi commande introuvable,
  - 422 pour erreurs de validation ou contraintes métier (ex : produit introuvable / sur devis dans commande web).

1.5.8 5.8 Journalisation (logs)

- Logs techniques/applicatifs : utilisation de `Log::info|warning|error` dans plusieurs contrôleurs/services (ex : promotions, catégories, commandes web).
- Logs d'activité (audit) : `spatie/laravel-activitylog` activé sur des modèles (ex : `User`, `Product`).
- Logs de connexion : listeners `LogSuccessfulLogin` / `LogFailedLogin` alimentant la table de login logs.

  Bonnes pratiques (implémentées)

  - "Best effort" sur l'emailing commande web : la commande n'est pas rollback si l'envoi mail échoue, l'échec est journalisé.
  - Ajout de rate limiting nommé centralisé (cf. `AppServiceProvider`).

1.5.9 5.9 Sécurité Backend

1.5.9.1 5.9.1 Authentification

- Backoffice (web) : routes d'auth standard (`Backend/routes/auth.php`) + middleware `auth` et `verified` sur `Backend/routes/web.php`.
- Portail client (API) : login `POST /api/client/login`, génération token Sanctum `createToken('client-portal')`.
- API v1 backoffice : `auth:sanctum`.

1.5.9.2 5.9.2 Autorisations

- RBAC : Spatie Permission.
  - Sur web : middleware `permission:*`.

  - Sur API v1 : middleware `role:Admin|SuperAdmin`.
- Bypass `SuperAdmin` via `Gate::before` (`Backend/app/Providers/AuthServiceProvider.php`).

1.5.9.3 5.9.3 Validation des entrées

- Validation Laravel via `FormRequest` (ex : `StoreWebOrderRequest`, `StoreProductReviewRequest`) et `Request::validate` (ex : suivi commande).

1.5.9.4 5.9.4 Protection des données sensibles

- Cookies "baseline" durcissement : recommandé dans `Backend/docs/security-audit-2026-01-18.md` (ex : `SESSION_SECURE_COOKIE=true` en prod).
- Headers de sécurité : middleware `SecurityHeaders` (nosniff, X-Frame-Options, HSTS si HTTPS, Permissions-Policy, etc.).

  Points de vigilance (issus du code)

  - CSP non activée (choix explicite dans `SecurityHeaders`) : à évaluer en fonction des exigences sécurité.

1.5.10 5.10 API

1.5.10.1 5.10.1 Convention de nommage

- Base : `/api`.
- Endpoints orientés ressources (produits, catégories, blog, etc.).
- Sous-espaces :
  - `client/*` pour le portail,
  - `v1/*` pour l'API backoffice.

1.5.10.2 5.10.2 Versioning

- Versioning explicite : préfixe `/api/v1`.
- Conventions détaillées : `Backend/docs/api-v1.md`.

1.5.10.3 5.10.3 Gestion des statuts HTTP

- 200 : réponses nominales (resources/JSON).
- 401/403 : authentification/autorisation (ex : login portail client).
- 404 : ressource introuvable (ex : suivi commande) + gestion centralisée `ModelNotFoundException`.
- 422 : validation/contraintes (ex : produit introuvable dans la commande, produit "sur devis").

1.5.10.4 5.10.4 Gestion des erreurs API

- Format minimal récurrent : `{ success: false, message: string }`.
- 404 "modèle introuvable" uniformisé pour `api/*`.

- OpenAPI : `Backend/xzone-openapi.yaml` décrit `ErrorResponse`.

1.6 6. Architecture Frontend (détaillée) — dossier `frontend/`

1.6.1 6.1 Stack technique (référence `frontend/package.json`)

- React 19.
- TypeScript.
- Vite.
- React Router.
- PDF : `jspdf`, `jspdf-autotable`.
- IA : `@google/genai`.
- Icons : `lucide-react`.

1.6.2 6.2 Organisation globale du code

- `frontend/index.tsx` : bootstrap React + providers.
- `frontend/App.tsx` : routing, lazy loading, layout.
- `frontend/pages/` : pages (Home, Shop, ProductDetail, Checkout, OrderTracking, etc.).
- `frontend/components/` : composants UI.
- `frontend/contexts/` : state global (chat, panier, langue).
- `frontend/utils/` : clients API, mappers, préfetching, génération PDF.

1.6.3 6.3 Architecture des composants

- Découpage par pages ; chargement différé via `React.lazy` et `Suspense`.
- Un layout conditionnel désactive header/footer sur la zone `/client-area/*`.

1.6.4 6.4 Gestion de l'état

- Context API :
  - `CartContext` persiste le panier en `localStorage`.
  - `ChatContext` stocke l'état d'ouverture et un "diagnosticContext".

1.6.5 6.5 Routing

- `BrowserRouter`.
- Routes principales :
  - catalogue : `/shop`, `/produits`, détail `/produits/:id` et `/product/:id`,
  - panier/checkout : `/cart`, `/checkout`,
  - commande : `/order-success/:orderNumber`, `/order-tracking`,
  - contenu : `/blog`, `/blog/:slug`,
  - "client area" : `/client-area/*`.

1.6.6 6.6 Communication avec le Backend

- Client HTTP : `frontend/utils/apiClient.ts` (wrapper `fetch`).
  - Base URL : `VITE_API_BASE_URL` (par défaut `/api`).
  - Erreurs : exception `ApiError` avec `status`, `url`, `body`.
- API catalogue : `frontend/utils/apiCatalog.ts` (produits, recherche, catégories, avis).
- API commandes : `frontend/utils/apiOrders.ts` (création commande + tracking).

1.6.7 6.7 Gestion des erreurs et des chargements

- Pages gèrent explicitement :
  - états `isLoading`/`isSubmitting`,
  - affichage d'erreurs via message utilisateur,
  - fallbacks UI via `Suspense`.

1.6.8 6.8 Sécurité Frontend

- Persistance : seul le panier est persistant (`localStorage`).
- Auth portail client : non observée dans les fichiers analysés (absence d'injection systématique `Authorization: Bearer` dans `apiClient`).
- IA : `frontend/vite.config.ts` injecte `GEMINI_API_KEY` au code client.

  Point de vigilance (issu du code)

  - Une clé injectée au frontend doit être considérée comme exposée (risque de fuite/abus). Si le besoin est une clé secrète, le proxy doit être côté serveur.

1.6.9 6.9 Performance et bonnes pratiques

- Code splitting : lazy routes.
- Optimisation build : `frontend/vite.config.ts` définit `manualChunks` (react/ai/pdf/icons).
- Préfetching de routes : logique dédiée (ex : `prefetchCriticalRoutesOnIdle`).

1.7 7. Configuration et environnements

1.7.1 7.1 Variables d'environnement

Backend : `Backend/.env.example` (extraits structurants)

- Applicatif : `APP_ENV`, `APP_DEBUG`, `APP_URL`, `APP_LOCALE`.
- Logs : `LOG_CHANNEL`, `LOG_LEVEL`.
- DB : `DB_CONNECTION` (sqlite par défaut dans l'exemple).
- Session/cache/queue : `SESSION_DRIVER`, `CACHE_STORE`, `QUEUE_CONNECTION`.

Frontend : `frontend/.env.example`

- `VITE_API_BASE_URL` (par défaut `/api`).
- `VITE_API_PROXY_TARGET` (cible proxy dev pour `/api` et `/storage`).

1.7.2 7.2 Séparation dev / test / recette / production

- Dev : `.env` basé sur `.env.example`.
- Test : `Backend/phpunit.xml` impose `APP_ENV=testing`, sqlite in-memory, queue sync, session array.
- Recette : non défini dans le dépôt.
- Prod : `Backend/docs/production.md` décrit prérequis, caches, migrations `--force`, permissions storage.

1.7.3 7.3 Paramètres critiques

- `APP_KEY` (obligatoire).
- `APP_DEBUG=false` en production.
- `APP_URL` correct (génération URLs et liens).
- DB : `DB_*`.
- Mail : requis si emailing opérationnel attendu (sinon mailer log possible).

1.7.4 7.4 Gestion des secrets

Le dépôt ne contient pas de mécanisme de coffre-fort (Vault/KMS) ni de politique de rotation. Les secrets sont attendus via variables d'environnement (`.env`) conformément aux pratiques Laravel.

1.8 8. Installation et mise en service

1.8.1 8.1 Prérequis techniques

- PHP >= 8.2 (recommandé 8.3 selon `Backend/docs/production.md`).
- Composer.
- Node.js >= 18.18 (contrainte dans `Backend/package.json`).
- SGBD : sqlite (dev) ou autre (prod) selon configuration.

1.8.2 8.2 Procédure d'installation Backend

Depuis `Backend/` :

1. `composer install`
2. `copy .env.example .env` puis renseigner `APP_KEY` via `php artisan key:generate`.
3. `php artisan migrate`
4. `npm install`

1.8.3 8.3 Procédure d'installation Frontend

Depuis `frontend/` :

1. `npm install`
2. Copier `.env.example` vers `.env.local` (ou équivalent) et ajuster `VITE_API_PROXY_TARGET`.

1.8.4 8.4 Lancement de l'application

- Backend (dev intégré) : `composer run dev` (serve + queue:listen + vite).
- Frontend SPA : `npm run dev` (dans `frontend/`).

1.8.5 8.5 Vérifications post-installation

- Accès UI backoffice (route `/` redirige vers setup/login selon configuration).
- Endpoints API publics : `/api/products`, `/api/categories/tree`.
- Commande web : `/api/orders`.
- Suivi : `/api/orders/track`.
- Assets : `/storage/*` si `storage:link`.

1.9 9. Déploiement et exploitation

1.9.1 9.1 Stratégie de déploiement

Référence : `Backend/docs/production.md` et scripts :

- PowerShell : `Backend/scripts/deploy-prod.ps1`
- Bash : `Backend/scripts/deploy-prod.sh`

Les scripts couvrent : install dépendances, build Vite, caches Laravel ; options pour migrations et `storage:link` côté PowerShell.

1.9.2 9.2 Environnements

- Local : mode `local` des scripts.
- Production : mode `production` des scripts (`--no-dev`, `migrate --force`).

1.9.3 9.3 Rollback

Aucune stratégie de rollback automatisée n'est fournie dans le dépôt. Les scripts ne couvrent pas :

- rollback de migrations,
- versioning d'artefacts déployés,
- blue/green ou canary.

1.9.4 9.4 Supervision et monitoring

Non implémenté/documenté dans le dépôt.

1.9.5 9.5 Sauvegardes

Non implémenté/documenté dans le dépôt.

1.10 10. Sécurité et conformité

1.10.1 10.1 Bonnes pratiques appliquées (preuves dans le dépôt)

- Audit dépendances : `Backend/docs/security-audit-2026-01-18.md`.
- Rate limiting : `Backend/app/Providers/AppServiceProvider.php` + throttles dans `Backend/routes/api.php`.
- Headers de sécurité : `Backend/app/Http/Middleware/SecurityHeaders.php`.
- RBAC : Spatie Permission + bypass `SuperAdmin`.
- Validation : FormRequests + validation inline.

1.10.2 10.2 Risques identifiés (issus du code)

- Exposition potentielle de `GEMINI_API_KEY` au client (frontend build).
- CSP non activée (choix explicite).
- Frontend `frontend/` : absence observée d'intégration auth Sanctum (Bearer) alors que l'API portail client existe.

1.10.3 10.3 Mesures de mitigation (orientées mise en conformité, à valider)

- Déplacer les appels IA nécessitant secret côté backend, ou utiliser un proxy serveur.
- Évaluer une politique CSP compatible Inertia (mise en place progressive + reporting).
- Clarifier l'architecture "portail client" :
  - soit consommation directe de l'API Sanctum depuis un client dédié,
  - soit intégration au backoffice Inertia,
  - soit mise en place d'un BFF.

1.10.4 10.4 Conformité réglementaire

Non documentée dans le dépôt (RGPD, journaux légaux, conservation, etc.).

1.11 11. Performance et scalabilité

1.11.1 11.1 Points de charge

- Recherche produit (`/api/products/search`) identifiée comme coûteuse et protégée par un throttle dédié.
- Endpoints publics catalogue/catégories/blog protégés via throttling.

1.11.2 11.2 Limites identifiées

- Aucun mécanisme de cache applicatif explicite n'est décrit au niveau "feature" dans les fichiers analysés (en dehors des capacités Laravel via drivers).

1.11.3 11.3 Stratégies d'optimisation (existantes)

- Rate limiting centralisé.
- Queue disponible (script dev lance `queue:listen`).
- Build optimisé : chunks Vite côté `frontend/`.

1.11.4 11.4 Évolutivité

- Variables Redis présentes dans `.env.example` (capacité d'évolution vers cache/queue Redis).
- Structuration Services/Actions facilitant l'extraction de cas d'usage.

1.12 12. Maintenance et évolutions

1.12.1 12.1 Points sensibles du code

- Modèle "attributs" catalogue (approche EAV) : nécessite une discipline de migration/validation et une gestion de performance (relations et hydrations).
- Multiplicité de domaines (vente, stock, RH, helpdesk) : risque de couplage fonctionnel et complexité de tests.
- Synchronisation de champs snapshot (commande web) : cohérence des calculs HT/TVA/TTC et règles produits.

1.12.2 12.2 Bonnes pratiques de maintenance

- Maintenir la couverture tests Feature (Pest + sqlite in-memory).
- Consolider la normalisation des réponses d'erreur API si l'API devient un contrat inter-SI.
- Documenter explicitement la stratégie de monitoring, sauvegarde, et rollback.

1.12.3 12.3 Stratégie d'évolution

- Continuer à isoler les cas d'usage en Services/Actions.
- Étendre l'API v1 selon les conventions `Backend/docs/api-v1.md`.

1.12.4 12.4 Dette technique identifiée (factuelle)

- Absence de mécanisme de rollback/supervision/sauvegarde dans le dépôt.
- Gestion du secret IA côté frontend.
- Recette non définie.

1.13 13. Annexes

1.13.1 13.1 Glossaire

- **Backoffice** : interface d'administration servie par Laravel via Inertia.
- **Portail client** : API dédiée aux comptes "client" (`portal_client=true`) protégée par Sanctum.
- **COD** : Cash On Delivery (paiement à la livraison), utilisé pour les commandes web.

1.13.2 13.2 Acronymes

- **DAT** : Document d'Architecture Technique.
- **RBAC** : Role-Based Access Control.
- **ORM** : Object-Relational Mapping.
- **SSR** : Server-Side Rendering.

1.13.3 13.3 Références techniques (dépôt)

- OpenAPI : `Backend/xzone-openapi.yaml`
- Conventions API v1 : `Backend/docs/api-v1.md`
- Déploiement production : `Backend/docs/production.md`
- Audit sécurité : `Backend/docs/security-audit-2026-01-18.md`
- Scripts déploiement : `Backend/scripts/deploy-prod.ps1`, `Backend/scripts/deploy-prod.sh`

1.13.4 13.4 Notes complémentaires

1.13.4.1 Export du DAT Le dépôt inclut `scripts/export-dat.ps1` (à la racine) pour exporter ce fichier en HTML/DOCX/PDF via Pandoc. # Document d'Architecture Technique (DAT) — X-Zone (ecommerce)

Date : 25 janvier 2026
Périmètre analysé : dossiers `Backend/` et `frontend/` présents dans la workspace.

> Important : ce DAT est basé **uniquement** sur les artefacts visibles dans le code fourni (routes, modèles, middlewares, docs, manifests).

Dans cette copie de workspace, le répertoire `Backend/config/` ne contient que `web_orders.php` (les fichiers de config Laravel standards ne sont pas visibles). Les constats "configuration" s'appuient donc sur `Backend/.env.example` et les documents `Backend/docs/*`.

1.14 1. Présentation générale du projet

1.14.1 1.1 Objectif de l'application

Le projet met en œuvre :

- Un **backoffice** (application interne) pour gérer catalogue, ventes, stocks, RH, helpdesk, etc., via **Laravel + Inertia + React**.
- Une **API** HTTP (préfixe `/api`) exposant :
  - des endpoints **publics** (catalogue, catégories, blog, avis, promotions, commandes web),
  - un **portail client** protégé par **Laravel Sanctum** (login token + routes "client").
- Un **site frontend** séparé (`frontend/`) : site vitrine/e-commerce (catalogue, panier, checkout COD, suivi de commande), pages marketing, chatbot/diagnostic.

1.14.2 1.2 Périmètre fonctionnel (d'après routes + migrations)

Principales capacités couvertes :

- Catalogue produits : produits, images, variantes, compatibilités, attributs dynamiques (approche EAV), documents produits.
- Catégories : arborescence, filtres par catégorie.
- Promotions : promotions et codes.
- Ventes B2B : clients, devis, factures, commandes.
- E-commerce web (public) : création de commande (COD), suivi de commande, historique de statuts.
- Avis produits : listing + soumission (côté API publique).
- Blog / actualités : listing + détail.
- Stock : mouvements, raisons, pièces jointes.
- Finance : transactions financières, catégories de dépenses, rappels.
- RH : départements, employés.
- Congés : types, soldes, demandes, actions, jours fériés, calendrier.
- Helpdesk / tickets : refonte de tables (migration `ticketing_refactor_support_tables` + `create_helpdesk_ticketing_tables`).
- Portail client : authentification token, profil, devis, factures, commandes, tickets, documents produits, sauvegarde de configurateur.
- Journalisation : logs d'activité (Spatie Activity Log), logs de login.

1.14.3 1.3 Public cible

- **Équipes IT / Dev** : installation, exploitation, déploiement, maintenance.
- **Développeurs Backend** : Laravel, Eloquent, API Resources, middlewares, sécurité.
- **Développeurs Frontend** : React/Vite (site public) + React/Inertia (backoffice).
- **Ops / SRE** : variables d'environnement, storage, queue/scheduler, hardening.

1.15 2. Architecture globale

1.15.1 2.1 Vue d'ensemble

Le projet est composé de deux applications frontend distinctes :

1) **Backend (Laravel 12)**

- Rend une UI backoffice via **Inertia + React** (sources dans `Backend/resources/js`).
- Expose une **API** dans `Backend/routes/api.php`.

2) **Frontend (React + Vite)**

- Application SPA (React Router) dans `frontend/`.
- Consomme l'API Laravel via `fetch` (wrapper `frontend/utils/apiClient.ts`).
- En dev, utilise un **proxy Vite** vers Laravel pour `/api` et `/storage`.

1.15.2 2.2 Schéma logique de communication

```
flowchart LR
  U[Utilisateur Web] -->|HTTP(S)| FE[frontend/ (React Router + Vite build)]
  FE -->|GET/POST /api/*| API[Backend (Laravel) routes/api.php]
  FE -->|GET /storage/*| ST[Backend public/storage]

  A[Admin/IT] -->|HTTP(S)| BO[Backoffice Inertia (Laravel web routes)]
  BO -->|Inertia requests| LAR[Backend (Laravel)]

  API --> DB[(Base de données)]
  LAR --> DB
  LAR --> FS[(Storage: storage/app/public)]
  LAR --> Q[(Queue: queue:listen/work)]
```

1.15.3 2.3 Choix techniques et justifications (d'après manifests)

- **Laravel 12** (`Backend/composer.json`) : framework backend, routage, validation, sécurité (CSRF/session), Eloquent ORM.

- **Inertia.js + React** (backoffice) : UI moderne sans API "interne" dédiée, partage de props serveur→client, DX.
- **React + Vite** (site public séparé) : découplage marketing/e-commerce, routing côté client, optimisation build.
- **Laravel Sanctum** : authentification API via tokens personnels pour le portail client.
- **Spatie Permission** : RBAC (rôles/permissions) pour backoffice + endpoints v1.
- **Spatie Activity Log** : audit/traçabilité d'actions sur des modèles (ex: `User`, `Product`).
- **Rate limiting** dédié : protection des endpoints publics et sensibles (recherche, login client, tickets).

1.16 3. Backend

1.16.1 3.1 Stack technique

- Langage : PHP (contrainte `php: ^8.2`).
- Framework : Laravel `^12.0`.
- ORM : Eloquent.
- Authentification :
  - Backoffice : auth Laravel (routes `Backend/routes/auth.php`) + email verification (`verified`).
  - Portail client / API : Laravel Sanctum (tokens personnels).
- Autorisations : Spatie Permission (`spatie/laravel-permission`).
- Audit & logs d'activité : `spatie/laravel-activitylog`.
- Génération documents : `barryvdh/laravel-dompdf`.
- Exports : `maatwebsite/excel`.
- Media/Images : `intervention/image`.
- Front backoffice : Inertia (`inertiajs/inertia-laravel`) + React + Vite.

Base de données : - Visible via migrations (création tables métiers) et `Backend/.env.example` (par défaut `DB_CONNECTION=sqlite`). - Le document `Backend/docs/production.md` mentionne MySQL/MariaDB/PostgreSQL comme options supportées.

1.16.2 3.2 Structure des dossiers et responsabilités

- `Backend/app/Http/Controllers/` : contrôleurs Web (Inertia) + API.
  - `Backend/app/Http/Controllers/Api` : API publique.
  - `Backend/app/Http/Controllers/Api/Client` : API portail client (Sanctum).
  - `Backend/app/Http/Controllers/Api/V1` : API backoffice v1 (Sanctum + rôle).
- `Backend/app/Http/Middleware/` : middlewares custom (onboarding, sécurité headers, portail client, Inertia).

- `Backend/app/Models/` : modèles Eloquent (catalogue, vente, RH, tickets, etc.).
- `Backend/database/migrations/` : schéma DB (source de vérité).
- `Backend/resources/js/` : application React Inertia (pages, layouts, utils, types, SSR).
- `Backend/routes/` :
  - `web.php` : backoffice (protégé auth/verified + permissions)
  - `api.php` : API publique + portail client + API v1
  - `auth.php` : auth backoffice (login/register/reset/verify)
- `Backend/docs/` : docs internes (API v1, production, audit sécurité).
- `Backend/scripts/` : scripts de déploiement (PowerShell + bash).

1.16.3 3.3 Principaux modules et flux métier

1.16.3.1 3.3.1 Catalogue & contenu

- Catalogue public :
  - Listing produits, détail par UUID, détail par slug, recherche, nouveautés/best-sellers, recommandés.
  - Documents produit.
  - Avis produits (GET/POST) avec throttling.
- Catégories : arborescence, filtres par catégorie, produits par catégorie.
- Blog : listing + détail.

1.16.3.2 3.3.2 E-commerce web (commandes publiques)

- Création de commande "web" via endpoint public (checkout COD).
- Suivi de commande public via `order_number + email`.
- Tables dédiées : `web_orders`, `web_order_items`, `web_order_status_histories`.
- Configuration associée : `Backend/config/web_orders.php`.

1.16.3.3 3.3.3 Portail client (API protégée)

- Authentification : `POST /api/client/login`.
  - Validation email/password, `Auth::attempt`, vérification `portal_client`, émission token Sanctum.
- Autorisation : middleware `portal_client` (alias vers `EnsurePortalClient`).
- Fonctions exposées : profil, devis, factures, commandes, tickets, documents, configurateur.

1.16.3.4 3.3.4 Backoffice (UI Inertia)

- Routes `Backend/routes/web.php` : zone `auth` + `verified`.
- Autorisation fine par permissions : `->middleware('permission:xxx')` sur la majorité des routes.
- Partage global Inertia : user/roles/permissions + settings (branding/SEO/contacts) via `AppSetting`.

1.16.3.5 3.3.5 API backoffice v1

- Préfixe `/api/v1`.
- Protégée : `auth:sanctum` + `role:Admin|SuperAdmin`.
- Endpoints en lecture (GET index/show) sur référentiels et entités (produits, devis, factures, RH, etc.).
- Conventions documentées dans `Backend/docs/api-v1.md`.

1.16.4 3.4 Authentification & autorisations

1.16.4.1 3.4.1 Backoffice (web)

- Routes auth : `Backend/routes/auth.php`.
- Zone backoffice : group middleware `['auth', 'verified']` dans `Backend/routes/web.php`.
- Autorisations : Spatie Permission via middleware `permission:*`.
- Bypass SuperAdmin : `Gate::before` dans `Backend/app/Providers/AuthServiceProvider.php`.

1.16.4.2 3.4.2 API Portail client

- Login : `Backend/app/Http/Controllers/Api/Client/AuthController.php`.
- Token : `createToken('client-portal')`.
- Accès aux endpoints : `auth:sanctum` + `portal_client`.

1.16.4.3 3.4.3 API v1

- `auth:sanctum` + `role:Admin|SuperAdmin` (Spatie).

1.16.5 3.5 Gestion des erreurs et logs

- Gestion des 404 “modèle introuvable” côté API : rendu JSON standardisé dans `Backend/bootstrap/app.php`.
- Logs applicatifs : présence de `storage/logs` et mention dans `Backend/docs/production.md`.
- Logs d’activité (audit) : Spatie Activity Log (ex: `User` et `Product` utilisent `LogsActivity`).
- Logs de login : tables + contrôleurs dédiés (routes web contiennent `LoginLogController`, exports associés).

1.16.6 3.6 Configuration (environnements, variables, secrets)

- Variables principales (extrait) : `Backend/.env.example`.
  - `APP_ENV`, `APP_DEBUG`, `APP_URL`
  - DB : `DB_CONNECTION` (sqlite par défaut), etc.
  - `SESSION_DRIVER`, `CACHE_STORE`, `QUEUE_CONNECTION`
  - Logging : `LOG_CHANNEL`, `LOG_LEVEL`
  - Vite : `VITE_APP_NAME`

Configuration spécifique “web orders” : `Backend/config/web_orders.php`.

1.16.7 3.7 Sécurité (bonnes pratiques implémentées)

- Rate limiting :
  - `public-api` (120/min/IP) et `public-search` (30/min/IP) + throttles spécifiques (login client, tickets, configurateur) dans `Backend/app/Providers/AppServiceProvider.php`.
  - Application sur routes publiques : `Backend/routes/api.php`.
- Headers de sécurité : middleware `Backend/app/Http/Middleware/SecurityHeaders.php` (nosniff, frame options, referrer policy, permissions policy, HSTS si HTTPS).
- Protection portail client : `EnsurePortalClient` (403 JSON si non autorisé).
- RBAC : Spatie Permission + bypass SuperAdmin.
- Audit de dépendances : synthèse `Backend/docs/security-audit-2026-01-18.md`.

1.16.8 3.8 API : principes, conventions, versioning

- Préfixe général : `/api` (cf. OpenAPI `Backend/xzone-openapi.yaml`).
- Séparation logique :
  - API publique : catalogue/catégories/blog/promos/commandes web.
  - Portail client : `/client/*` protégé Sanctum.
  - API Backoffice v1 : `/v1/*` protégé Sanctum + rôle.
- Conventions v1 (pagination, tri, includes) : `Backend/docs/api-v1.md`.
- OpenAPI : description et schémas dans `Backend/xzone-openapi.yaml`.

1.17 4. Frontend (dossier `frontend/`)

1.17.1 4.1 Stack technique

- React 19 + TypeScript.
- Vite 6.
- Routing : React Router (`react-router-dom`).
- UI : composants internes + icônes `lucide-react`.
- PDF : `jspdf` + `jspdf-autotable`.
- IA (chatbot/diagnostic) : dépendance `@google/genai`.

Remarque : le README mentionne Tailwind CSS “via CDN”. Le `package.json` du `frontend/` ne déclare pas Tailwind en dépendance.

1.17.2 4.2 Architecture des composants

- Point d’entrée : `frontend/index.tsx` (mount React + `LanguageProvider`).
- App + routes : `frontend/App.tsx`.
  - Lazy-loading des pages (React.lazy) + `Suspense`.
  - Layout conditionnel : header/footer désactivés sur `'/client-area/*'`.
- Composants réutilisables : `frontend/components/`.

- Gestion multi-langue : `frontend/contexts/LanguageContext` + `frontend/translations.ts`.
- Panier : `frontend/contexts/CartContext.tsx`.

1.17.3 4.3 Gestion des états et formulaires

- State management : Context API + hooks.
- Panier persisté : `localStorage` (clé `xzone_cart_v1`).
- Checkout : formulaire contrôlé dans `frontend/pages/Checkout.tsx`.

1.17.4 4.4 Communication avec le Backend

- Client HTTP : wrapper `fetch` dans `frontend/utils/apiClient.ts`.
  - `VITE_API_BASE_URL` (par défaut `/api`).
  - Gestion d'erreurs via exception `ApiError` (message dérivé du body JSON si présent).
- APIs catalogue : `frontend/utils/apiCatalog.ts`.
- APIs commandes : `frontend/utils/apiOrders.ts`.
- En dev : proxy Vite
  - `/api` et `/storage` proxifiés vers `VITE_API_PROXY_TARGET` (voir `frontend/vite.config.ts`).

1.17.5 4.5 Sécurité côté Frontend

- Le client `apiClient` n'implémente pas d'auth token (pas d'`Authorization: Bearer` ajouté automatiquement).
- Le "Client Area" visible dans `frontend/pages/ClientArea.tsx` contient principalement des données mockées dans la portion lue ; la consommation effective de l'API portail client devra être confirmée/complétée si l'objectif est un portail client authentifié sur ce frontend.
- Les secrets : la config Vite mappe `GEMINI_API_KEY` en `process.env.*` au build (`frontend/vite.config.ts`). En pratique, cela rend la clé accessible au code frontend ; si la clé doit rester secrète, un proxy serveur est recommandé.

1.18 5. Installation et démarrage

1.18.1 5.1 Prérequis

- PHP >= 8.2 (recommandé 8.3 selon `Backend/docs/production.md`).
- Composer.
- Node.js >= 18.18 (contrainte `Backend/package.json`).
- Base de données (sqlite par défaut en dev selon `.env.example`, autres possibles selon doc prod).

1.18.2 5.2 Installation Backend

Depuis `Backend/` :

- `composer install`
- `copy .env.example .env` (ou équivalent)
- `php artisan key:generate`
- `php artisan migrate`
- `npm install`

Script de dev intégré : `composer run dev` (serve + queue:listen + vite).

1.18.3 5.3 Installation Frontend

Depuis `frontend/` :

- `npm install`
- Copier/adapter `.env.example` (au minimum `VITE_API_PROXY_TARGET` en dev si besoin)

1.18.4 5.4 Lancement en local

Option A (Backoffice + API Laravel + assets Inertia) : - Dans `Backend/` : `composer run dev`

Option B (site frontend séparé + API Laravel) : - Lancer Laravel (ex: `php artisan serve` dans `Backend/`). - Dans `frontend/` : `npm run dev`.

1.18.5 5.5 Variables d'environnement

Backend : voir `Backend/.env.example`.
Frontend : voir `frontend/.env.example`.

1.19 6. Bonnes pratiques et conventions

1.19.1 6.1 Nommage & structure

- Backend : séparation claire `Controllers` / `Models` / `Middleware` / `Resources` / `Migrations`.
- API : regroupement par contexte (`Api`, `Api/Client`, `Api/V1`).
- Frontend : `pages/`, `components/`, `contexts/`, `utils/`.

1.19.2 6.2 Tests

- Framework : Pest (`Backend/require-dev` + `Backend/tests/Pest.php`).
- Tests Feature : utilisation de `RefreshDatabase` et DB sqlite in-memory (cf. `Backend/phpunit.xml`).

1.19.3 6.3 Linting / formatage

- Backend JS/TS (Inertia) : ESLint (flat config) + Prettier (voir `Backend/eslint.config.js`, scripts dans `Backend/package.json`).
- Backend PHP : `laravel/pint` disponible en dev (`Backend/composer.json`).

1.20 7. Déploiement

1.20.1 7.1 Stratégie

- Backend : procédure décrite dans `Backend/docs/production.md`.
  - `composer install --no-dev --optimize-autoloader`
  - migrations `php artisan migrate --force`
  - build Vite pour le backoffice (dans `Backend/`) ou via CI
  - caches Laravel (`config:cache`, `route:cache`, etc.)

1.20.2 7.2 Environnements

- Dev : `APP_ENV=local`, `APP_DEBUG=true` (cf. `.env.example`).
- Prod : recommandations dans `Backend/docs/production.md` et `Backend/docs/security-audit-2026-01-18.md` (debug off, cookies secure, caches).

1.20.3 7.3 Points de vigilance

- Storage : `php artisan storage:link` (serving `/storage/*`).
- Droits d'écriture : `storage/` et `bootstrap/cache/`.
- Secrets : permissions strictes sur `.env`.
- Queue & scheduler : config cron + superviseur (cf. doc production).

Scripts fournis : - PowerShell : `Backend/scripts/deploy-prod.ps1` - Bash : `Backend/scripts/deploy-prod.sh`

1.21 8. Maintenance et évolutivité

1.21.1 8.1 Points clés pour la maintenance

- La DB est fortement "métier" (beaucoup de modules) : migrations sont la référence.
- L'API publique est throttlée : ajuster centralement via `RateLimiter` (pas route par route).
- RBAC : permissions nombreuses côté backoffice ; maintenir une matrice rôles/permissions cohérente.
- Portail client : flag `portal_client` sur `users` + middleware dédié.

1.21.2 8.2 Axes d'amélioration possibles (basés sur constats code)

- Unifier/clarifier la stratégie "portail client" :
  - Le backend expose une API Sanctum complète.
  - Le `frontend/` n'embarque pas (dans les portions observées) de gestion de token (login + stockage + `Authorization`), et `ClientArea` semble maquetter des données.
- Clés IA : `GEMINI_API_KEY` exposée côté frontend par `define` Vite ; si besoin de confidentialité, déplacer l'appel IA côté backend.
- Standardisation erreurs : étendre le rendu JSON (au-delà du `ModelNotFoundException`) si l'objectif est une API homogène (ex: validation errors, exceptions applicatives).

1.21.3 8.3 Scalabilité

- Le backend est prêt pour :
  - exécution asynchrone via queue (`queue:listen` en dev script composer),
  - caching (variables `CACHE_STORE`, `REDIS_*` présentes dans `.env.example`).
- Les endpoints "coûteux" sont déjà throttlés (recherche).

1.22 Annexes

1.22.1 A. Références internes

- OpenAPI : `Backend/xzone-openapi.yaml`
- Conventions API v1 : `Backend/docs/api-v1.md`
- Procédure prod : `Backend/docs/production.md`
- Audit sécurité : `Backend/docs/security-audit-2026-01-18.md`

1.22.2 B. Export du document (HTML / DOCX / PDF)

Le projet inclut un script PowerShell d'export : `scripts/export-dat.ps1`.

1) Depuis la racine du projet, lancer :

- `powershell -ExecutionPolicy Bypass -File .\scripts\export-dat.ps1`

2) Sorties générées (par défaut) dans `dist-docs/` :

- `DAT-X-Zone.html`
- `DAT-X-Zone.docx`
- `DAT-X-Zone.pdf` (si moteur LaTeX disponible)

Pré-requis :

- Pandoc (obligatoire)
- Pour PDF : moteur LaTeX (ex: MiKTeX)