

Projet réalisé dans le cadre du passage du Titre  
Professionnel de  
Développeur Web et Web Mobile

# ARTE FACTO



Expérience immersive de galerie d'objets anciens et  
enchères dynamiques

Présenté par Eléonora Tartaglia

Ecole LaPlateforme\_ Promo Cannes 2024/2025

# S o m m a i r e

I. Prologue.....	5
II. Compétences du référentiel couvertes par le projet.....	6
II.1 Activité type n°1 : Développer la partie front-end d'une application.....	6
II.2 Activité type n°2 : Développer la partie back-end d'une application.....	7
III. Identité & Positionnement du projet.....	5
III.1 Vision, promesse, valeurs.....	5
III.2 Storytelling & inspirations.....	5
III.3 Positionnement stratégique.....	5
IV. Cadrage & Cahier des charges.....	5
IV.1 Objectifs, concept & périmètre.....	5
IV.2 MVP (Minimum Viable Product).....	5
IV.3 Liste des fonctionnalités (priorisées MoSCoW).....	5
V. Publics & Besoins.....	5
V.1 Public visé.....	5
V.2 Rôles.....	5
V.3 User stories.....	5
VI. Conception UX/UI.....	5
VI.1 Arborescence.....	5
VI.2 Wireframes.....	5
VI.3 Charte graphique.....	5

VII. Environnement & Méthodes.....	5
VII.1 Environnement de dev.....	5
VII.2 Versioning & workflow Git.....	5
VII.3 Qualité du code.....	5
VIII. Architecture & Choix techniques.....	5
VIII.1 Vue d'ensemble.....	5
VIII.2 Architecture du projet.....	5
VIII.3 Navigateurs & appareils cibles.....	5
IX. Base de données.....	5
IX.1 Methode Merise.....	5
IX.2 Dictionnaire de données.....	5
IX.3 Routes.....	5
X. Implémentation Front-End.....	5
X.1 Pages & composants.....	5
X.2 États, feedbacks, formulaires, uploads.....	5
X.3 Performance & Accessibilité.....	5
XI. Implémentation Back-End.....	5
XI.1 Modèles & Services.....	5
XI.2 Enchères.....	5
XI.3 Fichiers & stockage.....	5
XII. Sécurité de l'application.....	5
XII.1 Authentification, rôles & Policies.....	5
XII.2 CSRF, validation, mass assignment.....	5
XII.3 En-têtes, rate limiting, journaux.....	5

XIII. Tests & Jeu d'essai.....
XIII.1 Stratégie de test.....
XIII.2 Cas représentatif.....
XIII.3 Résultats.....
XIV. Déploiement & Opérations.....
XIV.1 Environnements.....
XIV.2 Commandes utiles & automatisations.....
XIV.3 Journalisation & supervision.....
XV. Évolutions & Roadmap.....
XVI. Veille & Résolution de problèmes.....
XVI.1 Veille sécurité/techno.....
XVI.2 Problèmes rencontrés & solutions.....
XVI.3 Sources anglophones.....
XVII. Épilogue.....
XVIII. Annexes.....

## I. Prologue

Je me présente, Eleonora Tartaglia, j'ai 34 ans. J'ai longtemps navigué loin du numérique : ma première vraie rencontre avec un ordinateur remonte à la fac, où j'étudiais la biochimie moléculaire et la génétique animale. La vie m'a conduite à explorer d'autres métiers (restauration, animation, secrétariat) avant de repenser ma trajectoire. Sans m'étendre, je dirai simplement que j'ai choisi un cap d'avenir : le web - scriptorium moderne où chaque ligne laisse une trace..

Dans le cadre de ma formation au Titre Professionnel de Développeur Web et Web Mobile (niveau 5), j'ai eu l'opportunité d'explorer de manière concrète l'ensemble des dimensions du développement web. C'est ainsi qu'est né Arte Facto, mon projet de fin de formation : une galerie numérique immersive dédiée à des objets rares et précieux, inspirée des musées, des salles d'enchères telles que Sotheby's et des plateformes d'art contemporaines.

Tout au long de ce projet, j'ai cherché à allier rigueur technique et sensibilité artistique, en construisant une application à la fois robuste, accessible, intuitive, tout en conservant une part de mystère propre à l'univers de la collection. Chaque fonctionnalité a été pensée pour répondre à un besoin réel tout en s'inscrivant dans un univers cohérent : mon but ultime étant de réenchanter l'expérience d'acquisition culturelle en ligne.

Ce dossier présente le cheminement complet : de l'analyse des besoins à la conception technique, de la réalisation graphique aux enjeux de sécurité, du jeu d'essai aux perspectives d'évolution. Il a pour vocation de démontrer l'acquisition des compétences visées par le référentiel du titre professionnel, tout en mettant en lumière ma démarche créative et professionnelle.

Arte Facto est un écrin numérique pour les civilisations passées, un espace où la magie du code rencontre l'art ancien, et où l'utilisateur devient collectionneur, enchérisseur... ou simple rêveur.

## **II. Compétences du référentiel**

Le projet Arte Facto a été conçu pour répondre aux exigences du Titre Professionnel de Développeur Web et Web Mobile, en couvrant l'ensemble des compétences techniques attendues dans les deux grandes activités du référentiel : le développement Front-End et le développement Back-End, intégrant les notions essentielles de sécurité, d'accessibilité, de modélisation de données, d'interactivité et de contenu dynamique.

### **II.1 Activité type n°1 : Développer la partie front-end d'une application web ou web mobile sécurisée**

#### **1. Installer et configurer son environnement de travail**

Je travaille sous macOS, et passer par Herd s'est imposé naturellement : un environnement local fiable, qui m'a permis de lancer le projet en quelques minutes. J'ai structuré le dépôt avec un .env clair, un APP\_KEY généré, et un outillage minimal mais efficace (Vite pour les assets, Tailwind pour le style, Livewire pour l'interactivité). Ce choix d'outils a fluidifié chaque itération et m'a évité de me perdre dans la configuration au détriment des fonctionnalités.

#### **2. Maquetter des interfaces utilisateur**

Pour le maquettage, j'ai privilégié Figma et une approche mobile-first. L'idée n'était pas de faire du décoratif, mais de clarifier les parcours essentiels : découvrir, consulter, agir. J'ai dessiné des écrans simples, des composants réutilisables, et des états prévus d'emblée (chargement, vide, erreur), afin que le code ne fasse que traduire une intention déjà posée.

#### **3. Réaliser des interfaces utilisateur statiques**

Au moment d'intégrer les interfaces statiques, j'ai opté pour Blade et Tailwind, car ce duo me permet d'aller vite sans renoncer à la lisibilité. Je veille à la sémantique HTML5, aux contrastes et au focus visible : des détails qui, cumulés, rendent l'application plus confortable et plus inclusive. Mon objectif était d'ancre une base solide et propre, sur laquelle l'interactivité pourrait se greffer sans casser la cohérence visuelle.

#### **4. Développer la partie dynamique des interfaces utilisateur**

Pour la dynamique côté front, Livewire v3 a été le choix le plus pertinent : assez léger pour rester dans une application serveur, mais suffisamment réactif pour offrir une expérience moderne (formulaires vivants, filtres instantanés, pagination fluide). Je garde la logique sensible côté serveur, je m'appuie sur la protection CSRF, et je privilégie des retours utilisateurs clairs pour que chaque action soit compréhensible et rassurante.

#### **II.2 Activité type n°2 : Développer la partie back-end d'une application web ou web mobile sécurisée**

#### **5. Mettre en place une base de données relationnelle**

Côté base de données relationnelle, j'ai suivi un fil classique et sûr : un modèle conceptuel pour poser les entités et leurs liens, puis des migrations versionnées qui racontent l'histoire du schéma et garantissent la reproductibilité. J'ai préféré une structure explicite, des clés bien choisies et quelques index utiles, afin que la lecture du domaine reste limpide et que les requêtes restent performantes.

#### **6. Développer des composants d'accès aux données SQL et NoSQL**

Pour les accès aux données, Eloquent s'est imposé pour sa clarté : des relations déclarées, des attributs castés, et des requêtes paramétrées quand la précision l'exige. Je n'hésite pas à encapsuler les opérations critiques dans des transactions pour préserver l'intégrité des données. Et lorsque le besoin de rapidité ou de diffusion temps réel se profile, j'ouvre la porte à un cache ou à un canal pub/sub (Redis) sans l'imposer au cœur du MVP.

#### **7. Développer des composants métier côté serveur**

Les composants métier côté serveur sont écrits au plus près des règles réelles du projet : ce n'est pas du code décoratif, c'est la traduction d'un jeu d'acteurs et de contraintes (ex. enchères, droits, validations). Je m'appuie sur des Policies pour la sécurité applicative, des middlewares pour cadrer l'accès, et une journalisation mesurée pour comprendre ce qui se passe sans noyer le signal.

## **8. Documenter le déploiement d'une application dynamique web ou web mobile**

Enfin, je documente le déploiement comme un rituel clair : environnements séparés, variables d'application, caches et build des assets, sans oublier la gestion du stockage privé pour les fichiers sensibles.

# **III. Identité et positionnement du projet**

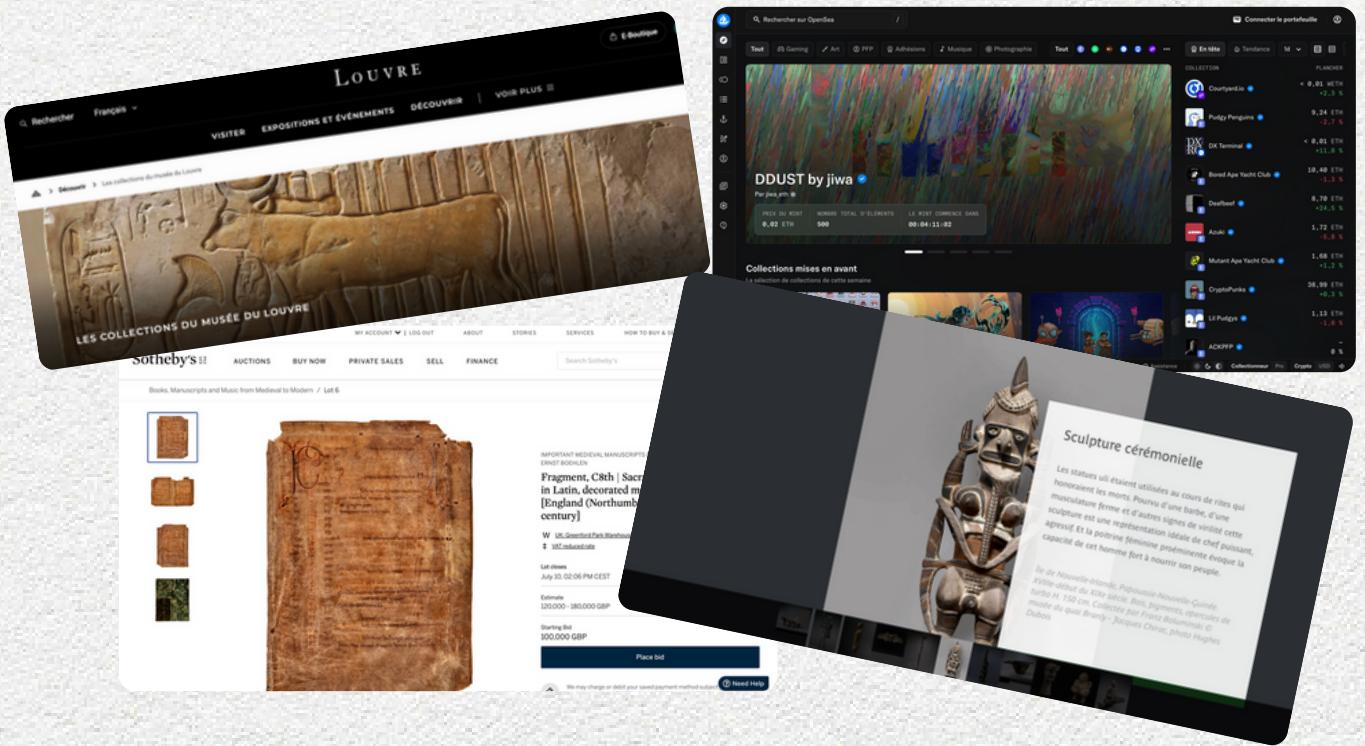
## **III.1 Vision & promesse**

Dès le départ, j'ai voulu qu'Arte Facto dépasse le simple rôle d'un site vitrine. Mon ambition était d'imaginer une galerie numérique immersive où l'on contemple des objets anciens avec le même respect que dans un musée. La promesse qui guide ce projet est simple : réenchanter l'expérience culturelle en ligne sans jamais perdre de vue la rigueur technique et l'accessibilité. Trois valeurs m'ont servi de fil conducteur à chaque décision : une sobriété élégante, une fiabilité éditoriale et technique, et un respect constant de l'utilisateur.

Cette vision m'a ensuite conduite à réfléchir à l'univers esthétique et narratif qui donnerait sa cohérence à l'ensemble.

## **III.2 Storytelling & inspirations**

Arte Facto puise son inspiration dans plusieurs mondes. J'ai observé les musées comme le Quai Branly, où chaque vitrine met en scène l'objet dans une lumière particulière. Je me suis tournée vers les maisons de vente telles que Sotheby's et Christie's, impressionnée par leur autorité et l'adrénaline qu'elles suscitent. Enfin, j'ai étudié les catalogues modernes comme Artsper ou OpenSea pour comprendre comment conjuguer fluidité et modernité dans la présentation d'objets.



De cette combinaison est née une intention claire : proposer une expérience hybride, à mi-chemin entre la solennité muséale, l'énergie de la vente et la fluidité du numérique. La palette cuivre et noir évoque la patine du temps, les typographies renforcent la gravité des titres et la lisibilité des contenus, tandis que des micro-interactions mesurées accompagnent l'utilisateur sans jamais voler la vedette à l'objet.

Ces choix narratifs et esthétiques m'ont naturellement amenée à réfléchir à la place d'Arte Facto dans un paysage déjà très occupé par d'autres plateformes.

### **III.3 Positionnement stratégique**

Après une étude du marché actuel, j'ai remarqué qu'il se divisait principalement en trois grands archétypes. Les plateformes d'art en ligne proposent des catalogues impeccables, mais la relation à l'objet reste distante et froide. Les maisons de vente incarnent une certaine autorité et l'adrénaline du direct, mais l'expérience y est solennelle, peu accueillante pour un néophyte. Quant aux marketplaces généralistes, elles misent sur l'accessibilité maximale, mais au prix d'une esthétique souvent sacrifiée et d'une mise en scène quasi inexistante.

Entre ces trois extrêmes, j'ai identifié une place à occuper : Arte Facto se veut un lieu éditorial hybride, où l'objet est raconté avant d'être marchandisé, où l'utilisateur comprend autant qu'il ressent. Ce positionnement stratégique constitue la base sur laquelle j'ai bâti mon cahier des charges et défini les priorités de mon projet.

## **IV. Cadrage & Cahier des charges**

### **IV.1 Objectifs & périmètre**

À partir de cette vision, il m'a fallu définir des objectifs concrets et un périmètre clair. Mon ambition était que l'expérience se déroule comme une déambulation fluide : entrer, cheminer, agir, sans jamais rompre le fil. Cela impliquait une navigation cohérente de l'accueil au catalogue, des fiches d'objets respirantes et accessibles, la possibilité de créer des favoris pour personnaliser son parcours, et enfin des mécanismes interactifs comme les enchères ou le panier concurrentiel.

Ces deux dernières fonctionnalités ne sont pas pensées comme des solutions marchandes complètes, mais comme des démonstrateurs pédagogiques : l'enchère illustre une logique métier simple et lisible, tandis que le panier concurrentiel introduit une tension douce propre aux galeries.

## **IV.2 MVP (Minimum Viable Product)**

Pour rester réaliste dans le temps imparti, j'ai défini un MVP, c'est-à-dire la version minimale mais fonctionnelle du projet. Ce MVP comprend une page d'accueil qui installe immédiatement l'univers, un catalogue filtrable et paginé, des fiches claires et détaillées, un système d'inscription avec validation par email et d'authentification, un espace personnel discret avec favoris et profil, un back-office CRUD pour gérer le catalogue, un panier concurrentiel et une enchère simulée avec des règles visibles et stables.

Ce choix du MVP m'a permis de maintenir une cohérence d'ensemble, tout en laissant ouvertes des pistes d'évolution.

## **IV.3 Liste des fonctionnalités**

Afin de distinguer l'essentiel de l'accessoire, j'ai appliqué la méthode MoSCoW : une boussole qui distingue l'indispensable du souhaitable, pour livrer l'essentiel sans disperser l'intention.

### Must — l'ossature non négociable.

Inscription et connexion avec rôles utilisateur, Galerie des artefacts avec filtres par civilisation, Fiches détaillées des objets, Panier concurrentiel, Tableau de bord utilisateur, Tableau de bord administrateur (CRUD sur toutes les entités)

### Should — l'aisance et le confort.

Simulation d'enchères dynamiques avec compte à rebours, Bots aux comportements variés, Recherche avancée plus fine du catalogue

### Could — l'élargissement maîtrisé.

Historique des mises, Liste de favoris, Avatar utilisateur

### Won't — la ligne que je ne franchis pas à ce stade.

Paiement, Logistique

## V. Publics & Usages

### V.1 Audiences cibles

La définition des publics a été une étape clé pour ancrer Arte Facto dans une réalité d'usage. Je m'adresse d'abord au curieux éclairé : celui qui aime comprendre la provenance, la datation, la matière, et qui lit les cartels au musée sans s'excuser. Vient le collectionneur amateur, sensible à la mise en valeur et à la respiration des pages, qui a besoin de se projeter avant tout engagement. En arrière-plan, l'équipe éditoriale attend de l'application un outil fiable et calme, qui permette de gérer le catalogue avec sérénité.

Ces différents profils ont tous en commun une attente : un parcours clair, une narration cohérente et des actions simples et rassurantes.

### V.2 Rôles et scénarios d'usage

Des audiences décrites naissent trois rôles vivants, chacun porteur d'un parcours qui lui ressemble.

Le visiteur anonyme découvre librement l'accueil, le catalogue et les fiches : il doit pouvoir comprendre l'esprit du site sans friction. Lorsqu'il souhaite franchir un premier seuil d'engagement, par exemple ajouter une pièce à son panier ou l'épingler en favori, une invitation claire et non intrusive l'oriente vers la création d'un compte. L'idée n'est pas de bloquer sa curiosité, mais de l'accompagner au moment où son intention devient personnelle.

Une fois identifié, l'utilisateur authentifié retrouve son profil et ses favoris, et peut ressentir pleinement la tension douce du panier concurrentiel : il perçoit que d'autres regardent la même pièce, sans artifices anxiogènes ni compteurs agressifs. S'il souhaite participer à une enchère, le chemin reste lisible : un contrôle d'identité minimal, réalisé en amont et dans un circuit protégé, précède l'accès à la salle des mises. Ce contrôle, conçu comme un démonstrateur et non une procédure marchande exhaustive sécurise l'intention sans casser le rythme de l'exploration.

En coulisses, l'administrateur (galeriste) orchestre l'ensemble. Il structure et enrichit le catalogue (artefacts, civilisations, tags, sources), valide les identités déposées via la route sécurisée, et supervise la démonstration d'enchères pour garantir lisibilité des règles, cohérence éditoriale et sérénité d'usage. Son interface n'est pas une tour de contrôle compliquée : c'est un atelier calme où l'on voit vite l'essentiel et où chaque action laisse une trace compréhensible.

Ces récits fonctionnels ont servi de filtre de décision tout au long du projet : si une fonctionnalité n'éclairait pas l'un de ces parcours, elle n'entrant pas dans le produit. À l'inverse, chaque écran, chaque état et chaque message a été conçu pour que l'intention de l'utilisateur trouve immédiatement sa réponse, sans détour ni surenchère technique.

## **VI. Environnement & Architecture technique**

J'ai construit un atelier de travail simple et fiable. L'objectif était double : démarrer vite et ne jamais me freiner pendant l'implémentation.

### **VI.1 Environnement de développement**

Je travaille sous macOS avec Visual Studio Code comme IDE. Pour éviter toute lourdeur de configuration, j'ai installé Laravel Herd, qui fournit un serveur HTTP local prêt à l'emploi (Nginx + PHP-FPM), des domaines en .test avec certificats TLS déjà approuvés, et des versions de PHP commutables à la volée. J'ai fixé PHP 8.4 pour rester alignée avec Laravel 12 et mes extensions. Dès qu'un dossier de projet est placé dans mon répertoire de sites, Herd l'expose automatiquement ; mon application est donc disponible sur <https://arte-facto-v2.test> sans serveur manuel.

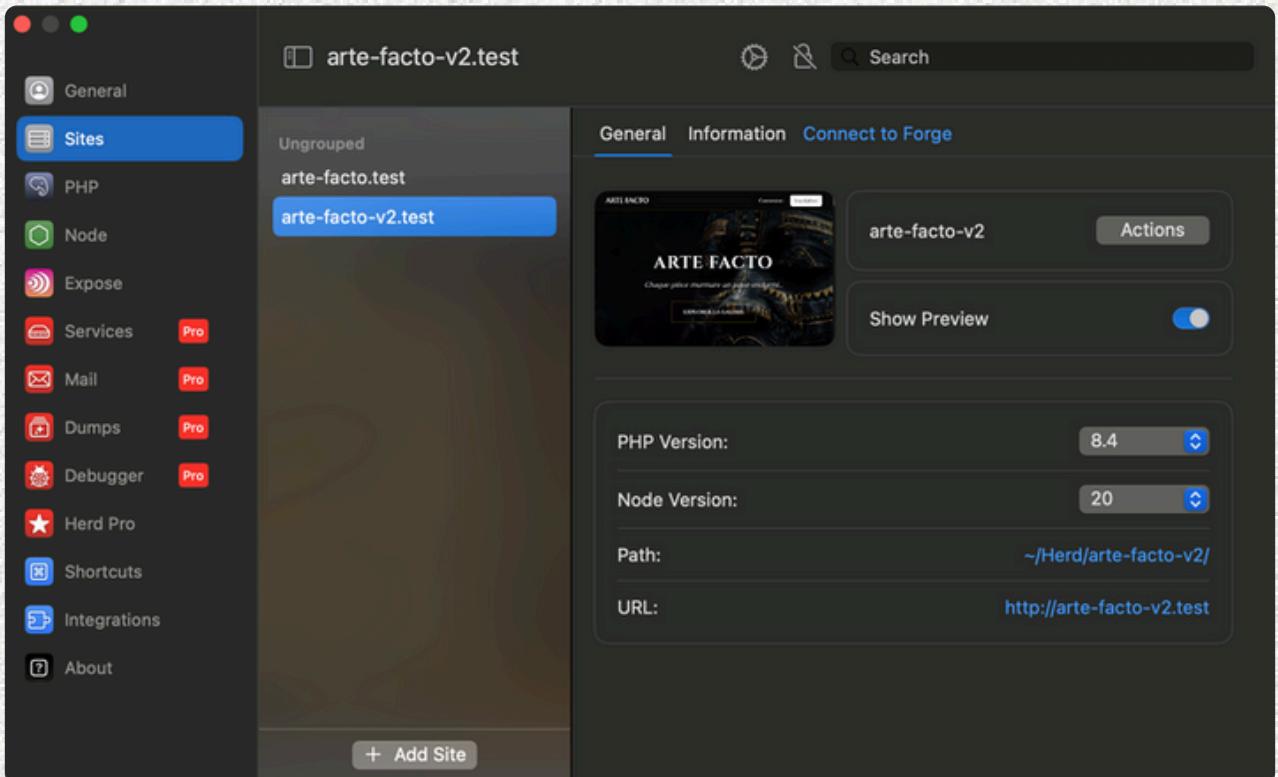
La création initiale s'est faite via l'installer Laravel : j'ai lancé laravel new arte-facto-v2. Cet outil réalise le scaffolding de projet : il génère le squelette Laravel (arborescence, dépendances, scripts), crée le fichier .env et règle automatiquement la clé d'application. C'est pour cela que je n'ai pas eu à exécuter cp .env.example .env ni php artisan key:generate.



```

!+ .env
1 APP_NAME="Arte Facto"
2 APP_ENV=local
3 APP_KEY=base64:XFer421S4P2weT+6r3uINlnvkTG4v/LI6pDcU3zDR4I=
4 APP_DEBUG=true
5 APP_URL=http://arte-facto-v2.test

```



En local, j'ai choisi SQLite pour itérer vite. Le projet est configuré avec DB\_CONNECTION=sqlite ; je m'assure que le fichier de base existe via touch database/database.sqlite. Quand je veux (re)poser le schéma et mes données de démo, je lance php artisan migrate --seed. Pour exposer les médias applicatifs côté public, je crée le lien symbolique avec php artisan storage:link.

Côté front, j'utilise Node.js (LTS) et npm. Après l'initialisation, j'installe les dépendances avec npm install, puis je lance le watcher d'assets avec npm run dev. Comme j'utilise Herd, je n'ai pas besoin de php artisan serve : le back est servi par Nginx (Herd) sur <https://arte-facto-v2.test>, et le front (assets) est servi par Vite sur le port 5173 tant que npm run dev tourne. Laravel détecte automatiquement le mode "hot" grâce au répertoire .vite/ et à la directive @vite dans les vues.

Au quotidien, je centralise tout dans un seul terminal avec composer run dev. Ce script orchestre plusieurs processus via Concurrently :

```
> npx concurrently -c "#93c5fd,#c4b5fd,#fb7185,#fdbd74" "php artisan serve" "php artisan queue:listen --tries=1" "php artisan tail --timeout=0" "npm run dev" --names=server,queue,logs,vite --kill-others
```

Concrètement, npm run dev démarre Vite (HMR), php artisan tail --timeout=0 streame les logs applicatifs en temps réel, et php artisan queue:listen --tries=1 écoute la file de jobs pour mes traitements asynchrones en développement.

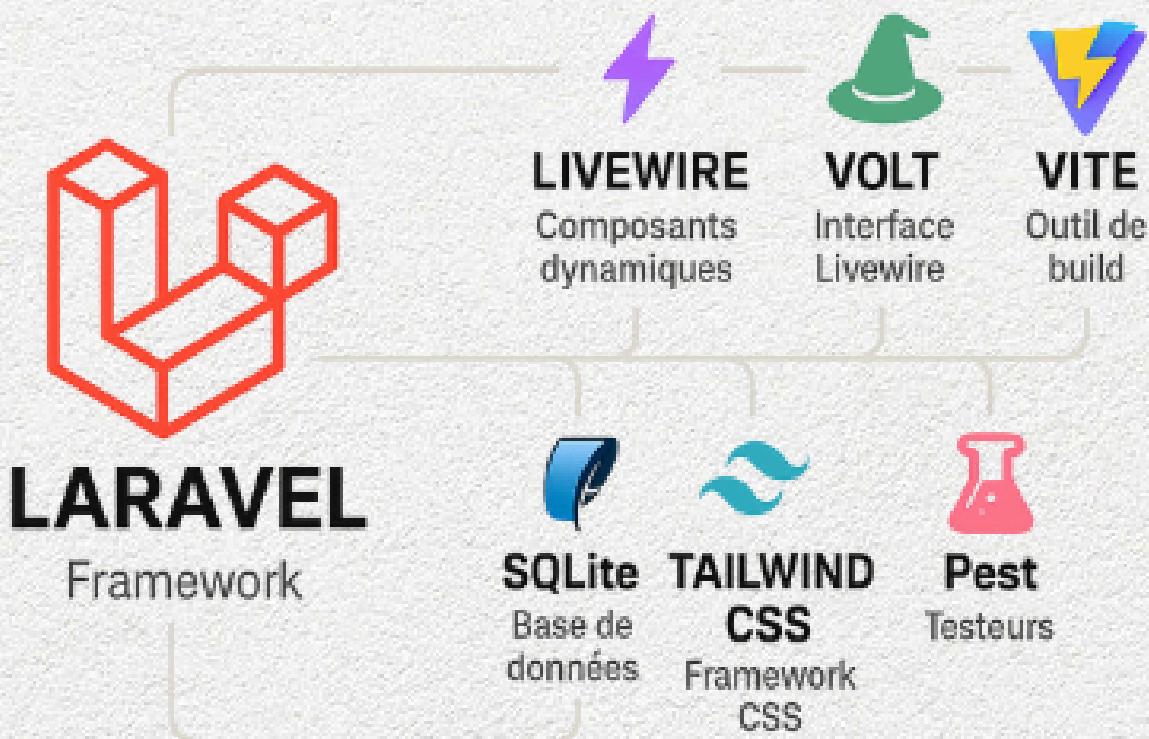
L'atelier étant stable et rapide, j'ai pu choisir une pile technique qui privilégie la sobriété et la lisibilité sans sacrifier l'interactivité.

## VI.2 Choix techniques

Mon fil conducteur a été la sobriété robuste. J'ai retenu Laravel 12 comme socle, pour bénéficier d'un écosystème cohérent : HTTP, Eloquent, validation, sécurité, Artisan et tests et d'un moteur de vues Blade qui me permet de composer des interfaces claires sans surcharger le front.

Pour l'interactivité, j'ai préféré Livewire v3 à une SPA lourde : je reste en rendu côté serveur, je simplifie la gestion d'état et j'évite de maintenir une API front/back dédiée. Sur des écrans ciblés, Volt me permet de rapprocher l'état, la validation et le gabarit dans un composant compact et immédiatement lisible ; Blade reste le canevas, Livewire/Volt ajoutent la dynamique là où elle a du sens.

Pour le style, Tailwind CSS me donne un design modulaire et maîtrisé, tandis que Vite assure des builds rapides et un HMR fiable. J'ai également adopté Flux, bibliothèque de composants UI conçue pour Livewire, afin de conserver une unité visuelle et une cohérence d'interaction sans réinventer chaque élément.

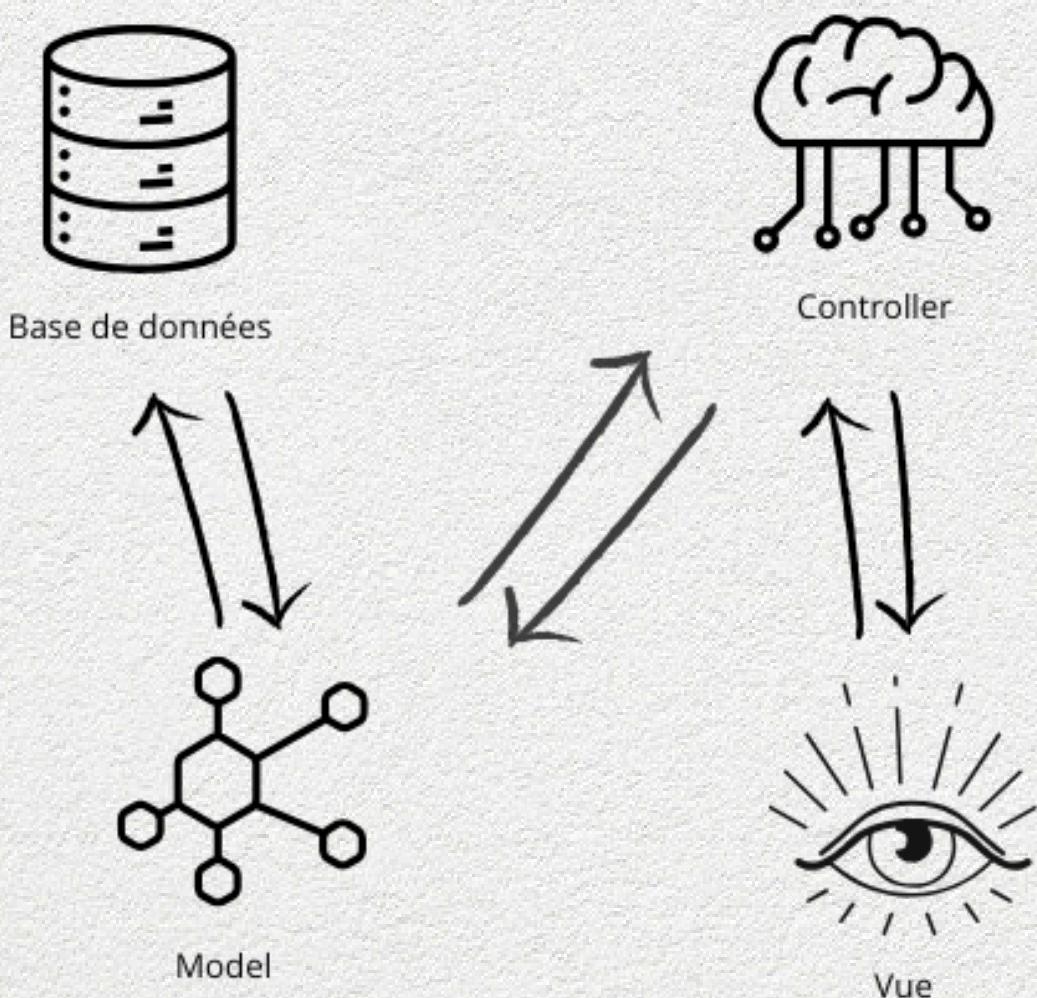


En local, SQLite m'offre la vitesse d'itération (un fichier, migrations, seed). Quand je veux éprouver des cas plus proches de la production comme la concurrence d'écriture, verrous, comportements de requêtes : je bascule sur MySQL/MariaDB. Côté accès aux données, j'anticipe le N+1 en combinant pagination et eager loading : je limite le volume d'items traités et je précharge leurs relations en une seule passe, ce qui évite la cascade de requêtes que provoquerait un lazy loading implicite.

Cette pile n'est pas un empilement : c'est une composition où chaque outil a une raison d'être, au service d'un code lisible, performant et stable.

### VI.3 Architecture applicative

Je reste fidèle au pattern MVC de Laravel, mais j'exploite Livewire v3 et Volt pour rapprocher une partie du “C” (contrôleur) de l’interface, directement au niveau des composants. L’idée n’est pas de casser le MVC : au contraire, je le resserre. Les routes restent fines, le modèle concentre la donnée et ses règles d’intégrité, la vue rend l’interface avec Blade, et le contrôleur s’exprime surtout sous forme de composants Livewire, parfois déclarés avec Volt au plus près de la page qui les utilise.



## Modèle

Je m'appuie sur Eloquent pour structurer les entités clés — Artifact, Civilization, Auction, Bid, CartItem, Transaction, User — et leurs relations (un artefact appartient à une civilisation, une enchère possède des mises, un utilisateur a des favoris, etc.). Les opérations sensibles (enregistrer une mise, clôturer une enchère, traiter une pièce d'identité) sont encadrées par des transactions afin de garantir l'intégrité : soit l'ensemble des écritures réussit, soit rien n'est committé. Les modèles portent aussi les conversions nécessaires (casts), quelques accessors/mutators discrets pour la lisibilité, et des règles métiers simples quand elles relèvent clairement du domaine (par exemple, l'état admissible d'une enchère).

## Vue

Le rendu est assuré par Blade, qui reste le canevas de l'application. J'utilise Flux — une bibliothèque de composants UI conçue pour Livewire — pour garder un langage visuel cohérent : flux:input, flux:button, flux:link, etc., chacun décliné avec des états explicites (focus, erreur, succès) et des messages de retour unifiés. La vue se concentre sur la présentation et l'orchestration légère des composants, sans logique métier dispersée. Les pages publiques et le back-office partagent ce même vocabulaire d'interface, ce qui renforce la continuité de l'expérience.

### Contrôleur : orienté composants

Sur les écrans interactifs, Livewire joue le rôle de contrôleur par composant : il porte l'état, valide les entrées, appelle le modèle et met à jour la vue sans quitter la page. Avec Volt, je déclare certains composants au sommet d'un fichier Blade via une classe anonyme : cela condense l'état, les règles de validation et les handlers juste au-dessus du gabarit qu'ils pilotent.

Résultat : une lecture compacte et locale de la fonctionnalité, où l'on voit d'un coup d'œil la donnée, les actions, puis leur rendu.

La logique d'écran vit dans des composants Livewire clairement délimités : une liste de catalogue, une fiche d'artefact, un formulaire d'upload, un panneau d'enchère.

Pour illustrer le flux de données : quand un utilisateur place une mise, l'action est captée par le composant Livewire (contrôleur local), validée (montant, état de l'enquête, identité déjà vérifiée si nécessaire), puis exécutée via Eloquent dans une transaction (création de la Bid, mise à jour des totaux/états). En cas de succès, l'état du composant est rafraîchi et Blade réaffiche la vue ; en cas d'échec, les messages d'erreur apparaissent sur les composants Flux concernés, sans navigation forcée ni rechargement complet.

Au final, cette architecture clarifie les responsabilités :

- le Modèle garantit la vérité et l'intégrité des données,
- la Vue présente un langage d'interface unifié et lisible,
- le Contrôleur vit au plus près de l'écran grâce à Livewire/Volt, ce qui réduit la dispersion et accélère la maintenance.

On reste dans l'esprit Laravel, mais avec un cycle court entre intention, action et retour visuel, exactement ce qu'il faut pour une galerie interactive où chaque écran raconte quelque chose et réagit avec sobriété.



## VI.4 Versioning & workflow Git

Le projet Arte Facto a été versionné en continu sur un dépôt GitHub. J'ai conservé une organisation simple : une branche main toujours stable et déployable, et des branches courtes dédiées à chaque sujet (feature/login, feature/favorites, fix/dashboard, etc.) que je fusionne vers main après vérification. Les commits sont fréquents et explicites afin d'assurer la traçabilité ; je privilégie des messages orientés intention, par exemple feat: ajouter favoris sur la fiche ou fix: sécuriser lecture pièce d'identité.

The screenshot shows a GitHub pull request merge history. At the top, a purple header bar indicates the pull request is merged. Below this, the title is "creation du crud artefacts #24" and it shows "eleonora-tartaglia merged 1 commit into main from 23-creation-du-crud-artefacts" 1 minute ago. The main body of the merge history lists the following events:

- eleonora-tartaglia commented 2 minutes ago: "No description provided."
- creation du crud artefacts (commit 6d6dfa7) was merged by eleonora-tartaglia 2 minutes ago.
- eleonora-tartaglia linked an issue 2 minutes ago that may be closed by this pull request: "Création du CRUD Artefacts #23". A green "Open" button is shown next to it.
- eleonora-tartaglia temporarily deployed to Testing 2 minutes ago — with GitHub Actions (Inactive).
- eleonora-tartaglia temporarily deployed to Testing 2 minutes ago — with GitHub Actions (Inactive).
- eleonora-tartaglia self-assigned this 1 minute ago.
- eleonora-tartaglia merged commit 367f7dc into main 1 minute ago. It shows 2 checks passed. Buttons for "View details" and "Revert" are available.

At the bottom, a summary states: "This branch was previously deployed" with "1 inactive deployment". A purple icon indicates the pull request is successfully merged and closed. It also says "You're all set — the 23-creation-du-crud-artefacts branch can be safely deleted." and has a "Delete branch" button.

Le fichier .gitignore a été ajusté pour ne garder que l'utile et protéger les secrets : les fichiers systèmes comme .DS\_Store ne rentrent pas dans l'historique, les dépendances vendor/ et node\_modules/ restent hors dépôt, le fichier .envn'est jamais versionné (seul .env.example sert de référence), et les caches temporaires générés par l'application ou les outils sont exclus.

Pour le pilotage, j'utilise GitHub Projects en mode kanban : A faire - En cours - Fait avec des tickets liés aux branches et quelques labels ciblés (feature, bug, UI, a11y, sécurité). Lorsque c'est nécessaire, j'appose l'étiquette MVP pour signaler les priorités. Cette mécanique légère me donne une vue claire de l'avancement sans alourdir le flux.

The screenshot shows a GitHub Projects Kanban board for the project "Arte Facto Versione 2.0". The board is organized into three columns: "Todo", "In Progress", and "Done".

- Todo:** 5 items
  - Arte\_Facto\_V2 #12 Implémentation de la wishlist
  - Arte\_Facto\_V2 #18 Création de la page A Propos
  - Arte\_Facto\_V2 #19 Création de la page Conditions de Ventes
  - Arte\_Facto\_V2 #20 Création de la page Contact
  - Arte\_Facto\_V2 #21 Création de la page de transaction
- In Progress:** 1 item
  - Arte\_Facto\_V2 #25 mise en place de l'upload d'une piece ID
- Done:** 17 items
  - Arte\_Facto\_V2 #2 Initialisation du projet
  - Arte\_Facto\_V2 #3 Création des migrations
  - Arte\_Facto\_V2 #4 Implémentation du seeder
  - Arte\_Facto\_V2 #13 Refonte du login et du register
  - Arte\_Facto\_V2 #14 Refonte de l'oublié du mot de passe et de la vérification par email

At the top of the board, there are buttons for "View 1" and "+ New view", a search bar, and a status update button. At the bottom, there are buttons for "Discard" and "Save".

## VII. Conception du Front



Avant de coder, j'ai cadré l'expérience : transformer l'idée d'une galerie sobre et immersive en parcours lisibles et en écrans cohérents. J'ai travaillé en mobile-first, puis décliné en bureau, en m'appuyant sur Figma pour aller vite tout en gardant une ligne graphique stable.

### VII.1 Wireframes

J'ai commencé par des squelettes d'écrans en mobile-first : blocs gris, hiérarchie des zones, placements des CTA, sans style. L'accueil pose une hero discrète puis oriente vers la galerie ; la galerie est une grille compacte de cartes ; la fiche respire (image, métadonnées, description, actions). Ces wireframes servent de contrat d'usage : ordre des informations, emplacements des actions, états vides/erreurs prévus, puis seulement après on "habille".

Une fois la structure figée, j'ai fixé le ton visuel pour que l'interface s'efface devant l'objet.



### VII.2 Moodboard & charte visuelle

La charte privilégie la patine plutôt que l'effet : fond sombre, cuivre mesuré pour l'action, sable/gris chaud pour la lecture. Les titres en empattements donnent la gravité, le texte courant en sans-sérif garde la lisibilité. Les micro-interactions sont courtes et calmes ; les états de focus restent nets pour guider sans distraire.

Avec le ton posé, j'ai verrouillé les trajets pour que tout se lise d'un seul souffle.



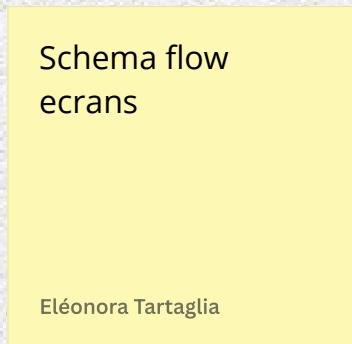
design systeme

Eléonora Tartaglia

### VII.3 Arborescence & parcours d'usage (flows)

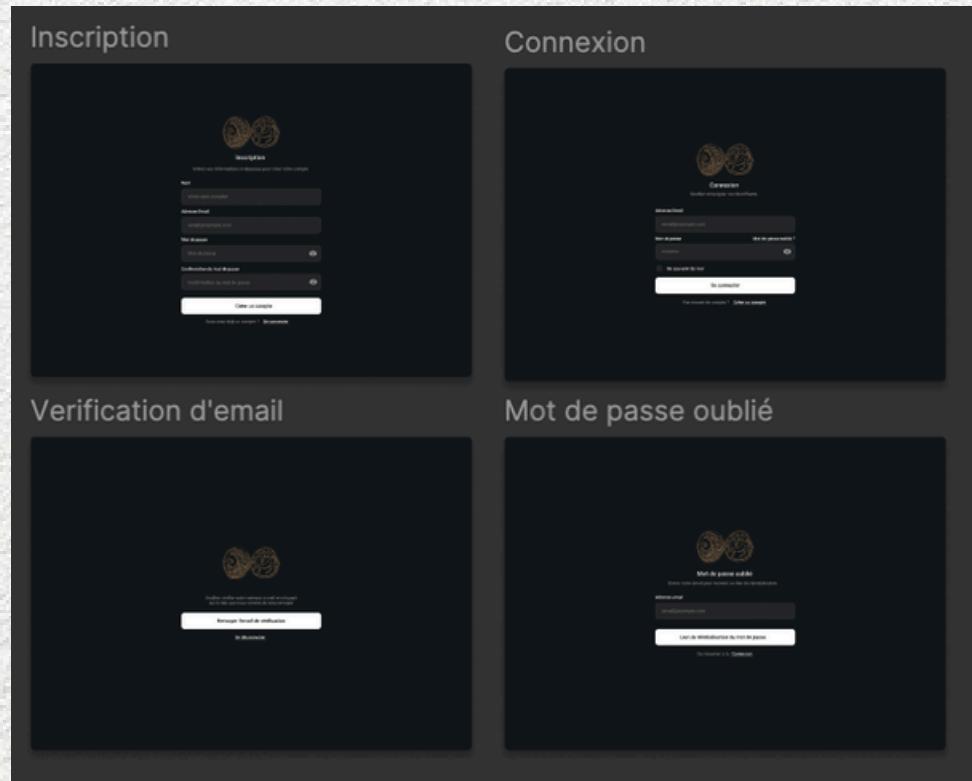
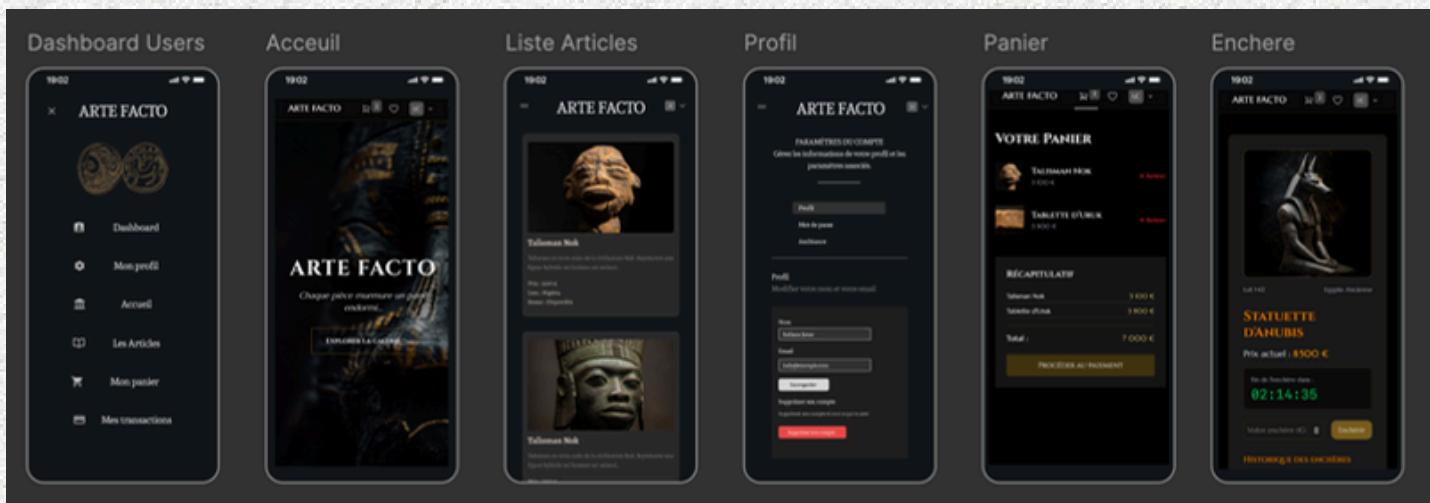
Le visiteur suit Accueil → Galerie → Fiche, sans compte. Au moment d'ajouter au panier ou de créer un favori, une invite douce ouvre l'inscription puis ramène exactement à la fiche. L'utilisateur connecté retrouve panier/favoris et accède à l'enchère depuis la fiche (règles courtes, identité vérifiée si besoin). En coulisses, l'admin gère catalogue et vérifications. Ce fil garde la logique regarder → comprendre → s'engager, sans rupture.

Une fois les flows validés, j'ai finalisé les écrans clés en haute fidélité.



## VII.4 Maquettes

À partir des wireframes, j'ai produit les maquettes : accueil, galerie, fiche, compte (connexion/inscription/mot de passe/vérif e-mail), panier, upload d'identité, enchère, puis l'admin. Le mobile a servi de base ; la version bureau élargit les colonnes et l'air entre blocs. Les écrans réalisés confirment l'intention : galerie lisible dans l'obscurité, fiche centrée sur la matière, panier à tension douce, enchère lisible avant l'action.



**Vue Accueil**

ARTE FACTO

Connexion Inscription

ACCUEIL RELIQUES CIVILISATIONS GALERIE A PROPOS CONTACT

Chaque pièce murmure un passé endormi.

Explorer la galerie

**Vue galerie**

ARTE FACTO

Connexion Inscription

ACCUEIL RELIQUES CIVILISATIONS GALERIE A PROPOS CONTACT

ART AFRICAIN EGYPTE ANTIQUE GRECE ANTIQUE ART PRECOLOMBIENS CIVILISATION ATLANTE MESOPOTAMIE ANCIENNE

Galerie

**ARTE FACTO**

Accueil Catalogue Enchères À propos

2 1 LC

Accueil / Galerie / Statuette d'Anubis

DÉTAILS ARCHÉOLOGIQUES

Année : 1907

Archéologue : Sir Flinders Petrie

Contexte : Hypogée familial intact

**STATUETTE D'ANUBIS**

Prix : 8 500 € Lieu : Nécropole de Thèbes

Catégorie : Égypte Ancienne Type : Enchère

Statut Disponible PARTICIPER À L'ENCHÈRE DEMO ENCHÈRE

**DESCRIPTION**

Statuette en bronze du dieu Anubis, maître des nécropoles égyptiennes, gardien de la pesée des âmes. Ses hiéroglyphes invoquent la protection contre l'au-delà. Chef-d'œuvre du Nouvel Empire conservant toute sa majesté funéraire.

**LÉGENDE**

Anubis guidait les âmes à travers le Duat pour juger leur cœur contre la plume de Maât.

## IX. Base de Données

### IX.1 Méthode Merise

La base de données d'Arte Facto a été conçue pour refléter la richesse de son univers narratif tout en respectant les règles fondamentales de normalisation, intégrité et sécurité.

Le modèle relationnel a été pensé autour de plusieurs entités centrales :

- Utilisateurs : inscrits ou administrateurs
- Artefacts : objets anciens à découvrir ou acquérir
- Civilisations : grandes familles d'origine des artefacts
- Enchères : historique des mises
- Mises

Voilà pourquoi avant d'écrire la moindre migration, j'ai suivi une démarche Merise classique : j'ai d'abord pensé le MCD (conceptuel), où j'identifie les entités métier et leurs associations, sans me soucier des types SQL, ensuite le MLD (logique), où je précise les cardinalités, les clés et les attributs ; je transforme les associations N–N en tables d'association. enfin le MPD (physique SQL), où je matérialise en tables, colonnes et contraintes (PK, FK, index), adaptées ici à SQLite en dev.

## 1) MCD — penser métier, sans SQL

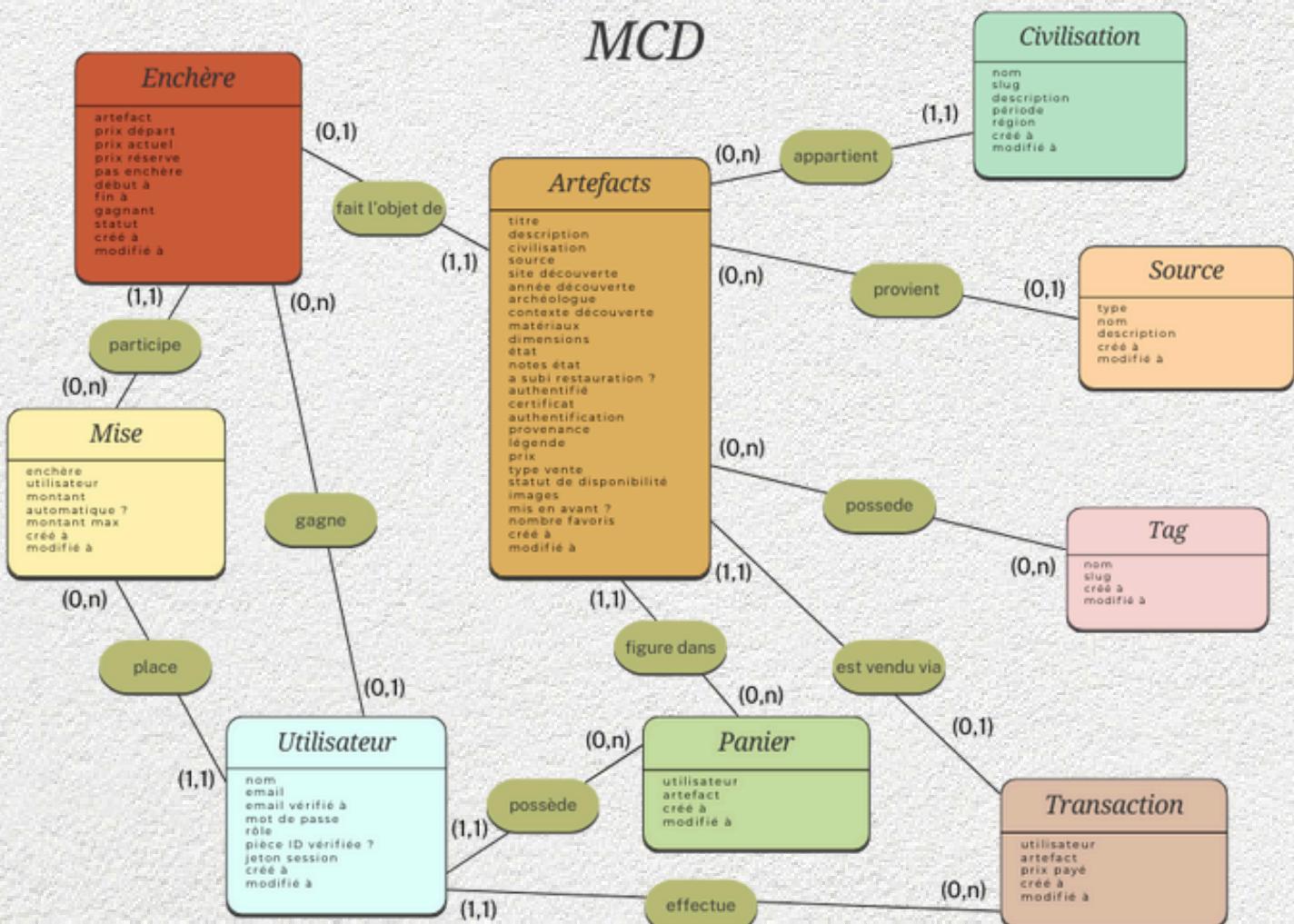
Je suis partie des objets de mon univers :

Civilization, Artifact, ArtifactSource.

Les liens sont naturels :

- Civilization 1—N Artifact (un artefact appartient à une civilisation) ;
- ArtifactSource 0..1—N Artifact (une provenance optionnelle peut concerner plusieurs artefacts).

J'ai aussi cartographié des attributs qui "vivent" avec l'artefact : description, caractéristiques (matériaux, dimensions), provenance, images, mode de vente (immédiat ou enchères), statut (disponible, au panier, vendu).



## 2) MLD — cardinalités, clés, tables

À ce stade, je choisis où dénormaliser un peu pour rester productive. Par exemple :

- Les matériaux, dimensions, provenance\_history, images sont des listes/objets : je les ai laissés en JSON (plus rapide à itérer en dev, surtout avec SQLite), plutôt que de multiplier les tables annexes.
- J'ai gardé des ENUM lisibles pour condition\_grade, sale\_type, status (clairs côté formulaire).

## MLD

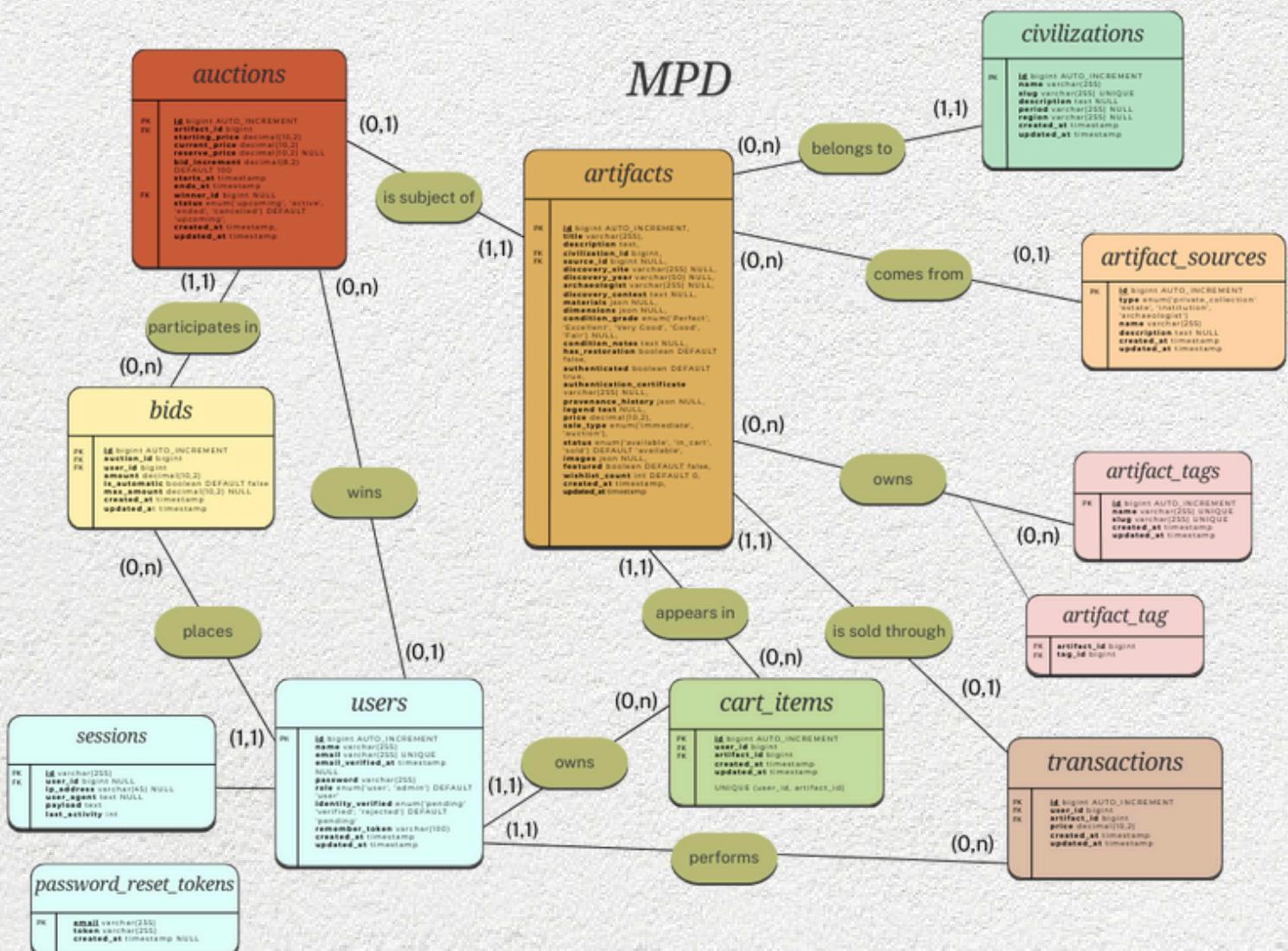
```
users(id, name, email, email_verified_at, password, role, identity_verified, remember_token, created_at, updated_at)
password_reset_tokens(email, token, created_at)
sessions(id, #user_id, ip_address, user_agent, payload, last_activity)
civilizations(id, name, slug, description, period, region, created_at, updated_at)
artifact_sources(id, type, name, description, created_at, updated_at)
artifacts( id, title, description, #civilization_id, #source_id, discovery_site, discovery_year, archaeologist, discovery_context, materials, dimensions, condition_grade, condition_notes, has_restoration, authenticated, authentication_certificate, provenance_history, legend, price, sale_type, status, images, featured, wishlist_count, created_at, updated_at)
artifact_tags(id, name, slug, created_at, updated_at)
artifacts_get_artifact_tag(#artifact_id, #tag_id)
auctions( id, #artifact_id, starting_price, current_price, reserve_price, bid_increment, starts_at, ends_at, #winner_id, status, created_at, updated_at)
bids( id, #auction_id, #user_id, amount, is_automatic, max_amount, created_at, updated_at)
cart_items( id, #user_id, #artifact_id, created_at, updated_at, UNIQUE (user_id, artifact_id) )
transactions( id, #user_id, #artifact_id, price, created_at, updated_at)
```

### 3) MPD — migrations concrètes

Je matérialise :

- civilizations, artifact\_sources, artifacts ;
- FK civilization\_id (obligatoire), source\_id (optionnelle) ;
- index composite (civilization\_id, status, sale\_type) : c'est exactement la requête de mon catalogue(filtrer par civilisation, statut, type de vente).
- JSON/ENUM sont pris en charge ; en SQLite, JSON = TEXT, et j'utilise Eloquent \$casts pour retrouver des arrays côté PHP.

Si je veux autoriser la suppression en chaîne, j'ajouterais `->cascadeOnDelete()` sur mes FK ; pour l'instant, j'assume RESTRICT (on ne supprime pas une civilisation si des artefacts y sont rattachés).



## IX.2 Dictionnaire de données (explications utiles)

### civilizations

- name, slug (unique), period, region, description : tout ce qui permet de classer et raconter.  
Le slug me sert d'URL propre et de filtre rapide.

### artifact\_sources

- type (private\_collection|estate|institution|archaeologist) : je typifie la provenance (utile à l'affichage et aux filtres).
- name, description : texte libre.

### artifacts

- Rattachements :
- civilization\_id (FK obligatoire), source\_id (FK optionnelle).
- Découverte :
- discovery\_site, discovery\_year (en string, car les dates d'époque sont parfois floues), archaeologist, discovery\_context.
- Caractéristiques :
- materials (JSON : liste de matériaux), dimensions (JSON : objet {h,w,d,unit}), condition\_grade (ENUM) + condition\_notes, has\_restoration (bool).
- Authentification :
- authenticated (bool), authentication\_certificate (chemin/identifiant).
- Provenance & récit :
- provenance\_history (JSON : jalons), legend (texte narratif).
- Commerce :
- price (decimal 10,2), sale\_type (immediate|auction), status (available|in\_cart|sold) : c'est la charnière UX (achat direct / enchère).
- Médias & éditorial :
- images (JSON : tableau d'URLs/paths), featured (mise en avant), wishlist\_count (compteur de favoris, pratique pour la curation).
- Performance :
- index (civilization\_id, status, sale\_type) pour les listes filtrées.

Pourquoi JSON ici ? Parce que je privilégie l'itération et le rendu : en Blade/Livewire, je boucle très vite sur des listes (matériaux, images) sans multiplier les tables. Si un jour je dois faire des requêtes analytiques(ex. “tous les objets > 30 cm”), je migrerai vers des colonnes dédiées ou des tables enfants.

(Complément) Migrations & ORM : \$fillable, \$casts, relations

Dans App\Models\Artifact, j'ai deux garde-fous essentiels :

\$fillable — anti mass assignment

```
protected $fillable = [
    'title',
    'description',
    'civilization_id',
    'source_id',
    'discovery_site',
    'discovery_year',
    'archaeologist',
    'discovery_context',
    'materials',
    'dimensions',
    'condition_grade',
    'condition_notes',
    'has_restoration',
    'authenticated',
    'authentication_certificate',
    'provenance_history',
    'legend',
    'price',
    'sale_type',
    'status',
    'images',
    'featured',
    'wishlist_count',
];
```

À quoi ça sert ? Quand je fais Artifact::create(\$request->all()), je whiteliste les champs autorisés. Tout le reste est ignoré → je me protège des mass assignment (quelqu'un ne peut pas injecter un champ système par surprise).

\$casts — types PHP automatiques

```
protected $casts = [
    'materials' => 'array',
    'dimensions' => 'array',
    'provenance_history' => 'array',
    'images' => 'array',
    'has_restoration' => 'boolean',
    'authenticated' => 'boolean',
    'featured' => 'boolean',
    'price' => 'decimal:2',
    'wishlist_count' => 'integer',
];
```

- À quoi ça sert ? Quand je lis l'Artifact, ces colonnes deviennent directement des types PHP utiles (array/bool/decimal). Pas besoin de json\_decode.
- Note : decimal:2 retourne une string formatée (sécurité d'affichage), ce qui convient bien pour la vue.

Relations Eloquent (métier lisible)

```
public function civilization(): BelongsTo
{
    return $this->belongsTo(Civilization::class);
}

public function source(): BelongsTo
{
    return $this->belongsTo(ArtifactSource::class, 'source_id');
}

public function tags(): BelongsToMany
{
    return $this->belongsToMany(ArtifactTag::class, 'artifact_tag', 'artifact_id', 'tag_id');
}

public function auction(): HasOne
{
    return $this->hasOne(Auction::class);
}

public function cartItems(): HasMany
{
    return $this->hasMany(CartItem::class);
}

public function usersInCart(): BelongsToMany
{
    return $this->belongsToMany(User::class, 'cart_items', 'artifact_id', 'user_id');
}

public function transactions(): HasMany
{
    return $this->hasMany(Transaction::class);
}
```

- BelongsTo (civilization, source) : côté enfant, je pointe vers la table parent.
- HasOne/HasMany (auction, cartItems, transactions) : côté parent, je liste ce qui dépend.
- BelongsToMany (tags, usersInCart) : relation N-N via un pivot explicite (artifact\_tag, cart\_items).

```

Schema::create('artifacts', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->text('description');

    // Relations
    $table->foreignId('civilization_id')->constrained();
    $table->foreignId('source_id')->nullable()->constrained('artifact_sources');

    // Infos archéologiques
    $table->string('discovery_site')->nullable();
    $table->string('discovery_year')->nullable();
    $table->string('archaeologist')->nullable();
    $table->text('discovery_context')->nullable();

    // Caractéristiques
    $table->json('materials')->nullable();
    $table->json('dimensions')->nullable();
    $table->enum('condition_grade', ['Perfect', 'Excellent', 'Very Good', 'Good', 'Fair'])->nullable();
    $table->text('condition_notes')->nullable();
    $table->boolean('has_restoration')->default(false);

    // Authentification
    $table->boolean('authenticated')->default(true);
    $table->string('authentication_certificate')->nullable();

    // Provenance & légendes
    $table->json('provenance_history')->nullable();
    $table->text('legend')->nullable();

    // Commerce
    $table->decimal('price', 10, 2);
    $table->enum('sale_type', ['immediate', 'auction']);
    $table->enum('status', ['available', 'in_cart', 'sold'])->default('available');

    // Images & marketing
    $table->json('images')->nullable();
    $table->boolean('featured')->default(false);
    $table->integer('wishlist_count')->default(0);

    $table->timestamps();

    // Index pour performance
    $table->index(['civilization_id', 'status', 'sale_type']);
});

```

- Indices & intégrité (pourquoi ceux-là)
- Index (civilization\_id, status, sale\_type) : c'est la clé de mon catalogue (filtrer par civilisation, n'afficher que available, et isoler les auctionlimmediate).
- Unique à prévoir (quand tu ajoutes les tags) : (artifact\_id, tag\_id) sur le pivot ; unique sur civilizations.slug (déjà fait) ; éventuellement un slug sur artifacts si tu veux des URLs propres.
- FK & cascade : par défaut, c'est RESTRICT. Je peux décider cascadeOnDelete() sur les pivots (sécurise les nettoyages).

## Lancement des migrations avec artisan

```
archimede@Archimede-1er-du-nom ~/Herd/arte-facto-v2 % php artisan migrate:status
Migration name ..... Batch / Status
0001_01_01_000000_create_users_table ..... [1] Ran
0001_01_01_000001_create_cache_table ..... [1] Ran
0001_01_01_000002_create_jobs_table ..... [1] Ran
2025_07_04_113131_create_civilizations_table ..... [1] Ran
2025_07_04_113330_create_artifact_sources_table ..... [1] Ran
2025_07_04_113418_create_artifacts_table ..... [1] Ran
2025_07_04_113643_create_artifact_tags_table ..... [1] Ran
2025_07_04_113815_create_auctions_table ..... [1] Ran
2025_07_04_113950_create_bids_table ..... [1] Ran
2025_07_04_114024_create_cart_items_table ..... [1] Ran
2025_07_04_114136_create_transactions_table ..... [1] Ran
```

## Vérifications rapides avec Tinker (preuves Eloquent)

### 1) Lire un artefact avec ses rattachements

```
archimede@Archimede-1er-du-nom:~/Herd/arte-facto-v2 % php artisan tinker
Psy Shell v0.12.9 (PHP 8.4.11 - cli) by Justin Hileman
> $a = App\Models\Artifact::with('civilization','source')->first();
=> App\Models\Artifact {#6586
    id: 1,
    title: "Masque Tribu Fang",
    description: "Masque Fang du XIXe siècle, en bois noirci gravé de motifs géométriques, utilisé pour des rituels de justice où l'esprit des ancêtres tranchait les différends. Son aura dense donne à chaque ligne la force d'un oracle silencieux.",
    civilization_id: 5,
    source_id: 3,
    discovery_site: "Plateaux forestiers d'Afrique centrale",
    discovery_year: "1889",
    archaeologist: "Mission ethnographique française",
    discovery_context: "Récupéré lors d'un rituel public par échange cérémoniel",
    materials: ["bois noirci","Pigments naturels"],
    dimensions: ["hauteur": "42cm", "largeur": "19cm", "profondeur": "14cm"],
    condition_grade: "Good",
    condition_notes: "Patine d'usage rituelle, fissure stabilisée au sommet.",
    has_restoration: 0,
    authenticated: 1,
    authentication_certificate: "Fondation Archéologique de Genève 2024",
    provenance_history: ["1889 - Acquisition rituelle", "1890-1960 - Collection privée", "1960-2024 - Fondation Archéologique de Genève"],
    legend: "Porté lors des procès tribaux, il faisait parler la vérité sous le regard des ancêtres.",
    price: 1000,
    sole_type: "Immediate",
    status: "available",
    images: ["https://cdn.midjourney.com/4ff414eb-7015-4559-b1f7-49437d36e074/v_1.png"],
    featured: 0,
    wishlist_count: 12,
    created_at: "2025-08-25 12:19:04",
    updated_at: "2025-08-25 12:19:04",
    civilization: App\Models\Civilization {#6596
        id: 5,
        name: "Royauté du Bénin",
        slug: "royaume-du-benin",
        description: "Puissant royaume d'Afrique de l'Ouest, le Bénin était renommé pour ses bronzes exceptionnels et son organisation militaire sophistiquée. Les plaques de bronze du palais royal constituent l'un des plus grands trésors artistiques de l'Afrique.",
        period: "1100 - 1897",
        region: "Afrique",
        created_at: "2025-08-25 12:19:04",
        updated_at: "2025-08-25 12:19:04",
    },
    source: App\Models\ArtifactSource {#6598
        id: 3,
        type: "institution",
        name: "Fondation Archéologique de Genève",
        description: "Institution reconnue internationalement pour ses recherches et sa rigueur scientifique. Célèbre occasionnellement des pièces en double pour financer ses fouilles.",
        created_at: "2025-08-25 12:19:04",
        updated_at: "2025-08-25 12:19:04",
    },
},
```

### 2) Vérifier les casts (JSON → array, bools)

```
> gettype($a->materials);
= "array"
|
```

### 3) Montrer les utilisateurs qui ont l'artefact au panier

```
> $a->usersInCart()->pluck('users.name');
= Illuminate\Support\Collection {#5629
|   all: [
        "Lara Croft",
    ],
}
```

### IX.3 “Routes” côté données : un dossier de flux plutôt qu’une API REST

Arte Facto ne s’appuie pas sur une API REST classique ; je délègue les actions à Livewire (AJAX → /livewire/update). J’ai donc documenté qui fait quoi sous forme de flux (c’est plus parlant, et honnête vis-à-vis de ma stack).

#### Exemple de flux (extraits)

- Catalogue

GET /artifacts → Composant Artifacts\Index → Eloquent Artifact::query() filtré (civilization\_id, status, sale\_type) → Vue paginée.

- Fiche

GET /artifacts/{id} → Composant Artifacts>Show → Artifact::with('civilization','source','auction').

- Connexion (Volt)

wire:submit="login" → POST Livewire /livewire/update → login() (validation, RateLimiter, Auth, Session::regenerate) → redirect /dashboard.

- Ajout au panier (concurrentiel)

wire:click="addToCart" → POST Livewire /livewire/update → insert/updateOrCreate dans cart\_items(UNIQUE(user\_id, artifact\_id)) → feedback.

- Admin : créer un artefact

wire:submit="save" → validate → Artifact::create(\$this->only([...])) (\$fillable) → redirect + toast.

Cette forme de “dossier de routes” par use case montre bien que je respecte MVC, que le C vit dans les composants (Livewire/Volt), et que le routage des actions est automatique et sécurisé (CSRF, validation, Policies).

## X. Implémentation Front-End

### X.1 Pages & composants (Blade, Flux, Livewire/Volt)

Ma règle d'or : chaque page sert une intention claire, chaque composant fait une seule chose — et le style reste sobre pour laisser respirer les artefacts.

- Accueil. Une entrée d'atmosphère : sélection d'artefacts “featured”, mise en lumière de quelques civilisations. But : donner envie de parcourir.
- Catalogue (/artifacts). Liste paginée, filtres par civilisation, statut et type de vente (achat direct / enchères). J'exploite l'index composite côté BDD pour garder le catalogue fluide.
- Implémentation : composant Livewire Artifacts\Index (filtre, pagination, persistance légère d'état).
- Fiche (/artifacts/{id}). La pièce respire : titre, provenance, images, caractéristiques (matériaux/dimensions via casts), statut et mode de vente. Deux actions phares : ajouter aux favoris et ajouter au panier (voir panier compétitif plus bas). Si sale\_type = auction, affichage du bloc enchère (montant courant, pas minimal, historique récent).
- Auth (login/register/forgot). En Volt : classe anonyme en tête du Blade, #[Validate] pour les règles, wire:submit pour appeler la méthode. UX douce via Flux (<flux:input>, <flux:button>, <flux:link>).
- Sécurité front liée : états d'erreur lisibles, pas de logique sensible côté client, tout part au serveur via Livewire.

Composants UI (Flux). Je m'appuie sur flux:input, flux:button, flux:checkbox, flux:link pour garder la cohérence visuelle et des états (focus, erreur, succès) standards sans réinventer la roue. Preuves à insérer : un extrait de la page login Volt (classe + <form wire:submit>), un screenshot du catalogue avec un filtre actif, une fiche montrant les cast JSON rendus (matériaux/dimensions).

## X.2 États, feedbacks, formulaires, uploads

### États et feedbacks.

Je rends explicites les étapes de chaque action :

- Soumission (wire:loading.attr="disabled", spinners légers si nécessaire).
- Succès (toast/flash discret : “Ajouté aux favoris”).
- Erreur (messages localisés sous le champ via Flux ; règles serveur via #[Validate] ou ->validate()).

### Formulaires.

- Auth / Profil. Champs wire:model avec validation serveur ; navigation fluide grâce à wire:navigate.
- Admin CRUD. Formulaires Livewire pour artifacts, civilizations, sources : je n'autorise que les champs \$fillable et je repose les règles côté serveur.
- Sécurité front liée : je ne duplique pas les règles en JS ; c'est le serveur qui tranche (anti contournement).

### Uploads.

- Identité utilisateur. Upload strict (taille, MIME), stockage privé (disk local) et affichage uniquement via route protégée (admin).
- Images d'artefacts. Stockage public (disk public), rendu en <img> optimisé (taille/ratio maîtrisé, ajout possible d'un srcset si besoin).
- Sécurité front liée : je ne reflète jamais le nom brut envoyé ; j'utilise le nom généré par Laravel.

Preuves : capture d'un message d'erreur sous un champ (Flux), capture de la modale/état “loading”, capture de l'upload d'identité (avec règle de validation affichée).

### X.3 Performance (sobriété côté interface)

Sobriété avant micro-optimisations.

- Pagination catalogue, pas de listes infinies.
- Images : je contrôle les tailles, je limite le nombre par page (car JSON images peut en contenir plusieurs).
- Interactivité : Livewire envoie des diffs HTML plutôt qu'un rechargement complet ; je regroupe les petites actions (ex. favoris) pour éviter le “bruit réseau”.

Coûts évités.

- Pas de SPA lourde ni de framework JS séparé : je reste en rendu serveur + Livewire (bon pour le SEO, la lisibilité, la stabilité).
- Les champs JSON castés évitent des requêtes supplémentaires à chaque rendu (boucles directes en Blade).

Preuves : Web Inspector → onglet Réseau : montrer un POST /livewire/update avec 200 et un payload JSON concis ; un Lighthouse local “Performance” sur la fiche (optionnel).

### X.4 Sécurité

#### Rendu sûr : échappement systématique contre les XSS

Je confie l'affichage des contenus à Blade avec {{ ... }} qui échappe les caractères spéciaux (<, >, &, "). L'objectif est simple : bloquer les attaques XSS (Cross-Site Scripting) où un texte injecté contiendrait du JavaScript déguisé (par exemple <script>alert(1)</script>). Une fois échappé, ce texte n'est plus interprété comme du code, il devient du texte inoffensif dans le DOM.

(Preuve : capture d'une vue Blade de fiche artefact montrant {{ \$artifact->title }} et {{ \$artifact->description }} ; Légende : « Échappement Blade : le HTML injecté n'est jamais exécuté (protection XSS). »)

## **Surface minimale : pas de données sensibles exposées dans la page**

Les composants Livewire portent l'état d'interface, pas des secrets. Je m'interdis d'exposer dans le HTML ou dans les attributs data-... des chemins de stockage privé, des indicateurs d'autorisation, ou des PII (pièces d'identité, emails internes, etc.). Moins on met de choses dans le markup, moins on donne de prise à des scripts tiers ou à des extensions de navigateur trop curieuses. Le navigateur ne reçoit que ce qui sert le rendu : titre, époque, matières — pas les mécanismes internes.

(Preuve : capture “Elements” montrant le DOM sans chemin storage/private/... ni flags d'autorisation ; Légende : « DOM épuré : aucune donnée sensible exposée côté client. »)

## **CSRF by design : formulaires protégés contre la Cross-Site Request Forgery**

Chaque formulaire inclut @csrf (et Livewire l'embarque nativement). CSRF empêche qu'un site tiers pousse l'utilisateur (déjà connecté) à soumettre une requête à son insu : le serveur exige un jeton unique placé dans le formulaire et refusé s'il manque ou s'il est invalide. Concrètement : un clic piégé sur un autre site ne peut pas valider une enchère, changer un profil ou déposer un fichier chez moi. Les erreurs restent au plus près du champ, en texte clair — jamais d'informations internes ou de stack traces côté UI.

(Preuve : capture d'un POST dans l'onglet Network montrant le champ \_token CSRF + vue Blade avec @csrf ; Légende : « Jeton CSRF : une requête forgée depuis un autre site est rejetée. »)

## **Upload d'identité : restriction côté UI et aucun pré-affichage**

Pour l'upload d'une pièce d'identité, je restreins dès l'input les types acceptés (`accept="image/*,application/pdf"`) et j'annonce la taille maximale. C'est une première barrière UX, non pas une sécurité finale. Surtout, je refuse de pré-afficher le fichier dans la page (miniature ou embed) : on évite d'exposer un document sensible dans le DOM ou l'historique du navigateur. L'UI confirme seulement la réussite de l'envoi ; l'aperçu reste réservé à l'administration sur route protégée (côté serveur).

(Preuve : capture du champ file avec `accept=...` + message de confirmation sans preview ;  
Légende : « Upload guidé, sans exposition : pas de contenu sensible rendu côté client. »)

## **États lisibles : focus visible, annonces via ARIA et mouvements mesurés**

Je rends le focus évident (classes Tailwind) et je parcours au clavier tous les écrans clés. Lorsqu'une enchère évolue, je diffuse l'information via une zone `aria-live="polite"` (ou un toast discret) pour décrire le changement sans agressivité. C'est une hygiène de sécurité aussi : on évite les effets de surprise, on évite les gestes précipités. Les transitions restent sobres (préférence “reduce motion” respectée) : aucune animation stroboscopique, aucun saut de page.

(Preuve : GIF d'un parcours au clavier + inspecteur affichant `aria-live="polite"` lors d'une mise à jour ; Légende : « Annonce non intrusive : on comprend ce qui change, sans surprise. »)

## **Cookies httpOnly, pas de localStorage pour le sensible**

Les cookies de session sont marqués `httpOnly` : le JavaScript ne peut pas les lire, ce qui neutralise de nombreuses tentatives de vol de session via XSS résiduelle. Je n'utilise pas `localStorage/sessionStorage` pour des éléments sensibles (droits, identités, clés). Tout ce qui touche à la confiance reste côté serveur ; le front reflète l'état, il ne détient rien de critique.

(Preuve : onglet Storage montrant le cookie de session avec l'attribut `HttpOnly` et un `localStorage` vide d'infos sensibles ; Légende : « Session inaccessible au JS : pas de fuite par le front. »)

## XI. Implémentation Back-End

### XI.1 Modèles & Services (Eloquent, transactions, \$fillable/\$casts)

Modèles Eloquent.

Je centralise l'état métier dans les modèles :

- Artifact (ton modèle est propre) : \$fillable (anti mass assignment), \$casts (JSON→array, bools, decimal:2), relations (civilization, source, auction, tags (N-N), cartItems, usersInCart, transactions).
- Civilization, ArtifactSource : référentiels propres.
- (À venir / présent ailleurs) Auction, Bid, CartItem, Transaction.

Services (logique sensible).

Dès qu'une méthode mèle validation métier + écriture (enchère, panier), je passe par une transaction DB. Exemple type pour l'enchère :

1. Valider l'input ;
2. Ouvrir une transaction ;
3. Lire l'enchère consistamment (verrou logique adapté au SGBD) ;
4. Vérifier le pas minimal ;
5. Insérer la Bid, mettre à jour le montant courant ;
6. Commit ;
7. Journaliser (sans PII).

\$fillable / \$casts (rappel).

- \$fillable définit ce que j'accepte d'écrire depuis l'extérieur (sécurité).
- \$casts me garantit des types PHP cohérents (arrays/bools/decimal) — donc des templates Blade plus simples et moins d'erreurs.

Preuves : capture du bloc \$fillable/\$casts d'Artifact, capture d'un service ou d'une méthode Livewire qui encadre une transaction (enchère ou panier).

## XI.2 Enchères (règles, flux, sécurité)

Ce que je veux démontrer.

Une mécanique lisible : un montant courant, un pas minimal, l'impossibilité d'égaler une offre, des erreurs claires si le montant est insuffisant — et aucune action tant qu'on n'est pas authentifié.

Flux backend type (Livewire).

- Méthode placeBid() :
  - ->validate(['amount' => 'required|integer|min:1'])
  - DB::transaction(...)
  - Charger l'enchère ; calculer le minimum acceptable ; refuser si amount < minNext → ValidationException avec message localisé.
  - Créer Bid, mettre à jour Auction.current\_amount.
  - Log::info('Bid placed', ['auction\_id'=>...,'user\_id'=>auth()->id(),'amount'=>...]);

Sécurité backend liée.

- Auth obligatoire (middleware auth).
- RateLimiter déjà en place pour l'auth, activable aussi pour limiter les frappes exagérées.
- Transactions pour l'intégrité (même en SQLite, ça sérialise l'accès et suffit pour la démo).
- Aucun champ externe n'écrit hors \$fillable.

Preuves : capture Network d'une action placeBid (POST Livewire), capture du message d'erreur côté UI si montant insuffisant, capture de log "Bid placed".

## XI.3 Panier compétitif, ma fonctionnalité signature

### Intention UX : tension douce, jamais d'alarme

Je voulais que l'on ressente le désir des autres sans se faire bousculer. L'objet est unique ; s'il circule dans d'autres paniers, l'interface le suggère avec un simple indicateur ("déjà dans X paniers"), sans faux compte à rebours, sans cadenas ni cris visuels. Le panier sert à se projeter, pas à enfermer. On clique, on voit que l'objet vit, on garde la main.

Preuves à illustrer : capture d'écran d'une fiche avec le badge "déjà dans X paniers" (compteur)

Légende : « Tension douce : signal sobre, sans pression. »

### Modèle & contrainte

Je m'appuie sur une table cart\_items où chaque ligne représente une relation user ↔ artifact et sur une relation Eloquent usersInCart() côté Artifact pour compter les paniers actifs et ainsi afficher "déjà dans X paniers". Ce compteur suffit à matérialiser la tension sans bloquer l'objet : on ne réserve pas l'artefact, c'est une mise en scène UX, pas une promesse juridique. Je n'affiche jamais de noms : seul le compteur nourrit le badge de la fiche.

Preuve (code bref) : extrait du modèle Artifact avec usersInCart() et du modèle CartItem (clé étrangère, timestamps). Légende : « Relation minimale : compter, pas exposer. »

Preuves à illustrer : extrait de migration montrant l'index UNIQUE sur (user\_id, artifact\_id),  
Légende : « Intégrité garantie : un seul item par couple user × artefact. »

### Flux utilisateur : ajouter, retirer, ressentir

#### 1) Chargement des indicateurs: mount()

Dans mount(\$id), je charge l'artefact et ses relations utiles (civilization, source, tags, usersInCart) puis j'appelle checkCartStates() pour calculer deux indicateurs côté serveur :

- \$inCart : l'utilisateur courant a-t-il déjà cet objet dans son panier ?
- \$otherCartsCount : combien d'autres paniers contiennent l'objet ?

Ces deux valeurs alimentent directement les badges dans show.blade.php.

- Preuve (code bref) : extrait de mount() + checkCartStates() montrant l'usage de usersInCart->contains() et du comptage. Légende : « États dérivés côté serveur : \$inCart, \$otherCartsCount. »

## 2) Ajout au panier : méthode addToCart()

L'utilisateur authentifié clique sur ajout au panier uniquement si l'artefact est disponible et en vente immédiate, la vue filtre... le serveur confirme. J'utilise une opération idempotente : firstOrCreate() pour absorber les doubles clics / onglets sans créer d'incohérence. Je recharge usersInCart, j'appelle checkCartStates() pour mettre à jour \$inCart et \$otherCartsCount, puis j'émetts l'événement Livewire : cartUpdated pour rafraîchir l'indicateur global.

L'interface répond par un toast discret (“Ajouté à votre panier”), l'icône bascule : le compteur “X paniers” peut monter d'un cran.

Ce choix “idempotent” n'est pas là “pour empêcher deux fois le même article” (unité oblige) ; il est là pour absorber les rafales (double-clic, deux onglets, latence réseau) sans erreur ni duplication visuelle.

Preuves :

- capture du toast “Ajouté à votre panier 🍔” ; Légende : « Retour immédiat, sans rupture. »
- snippet addToCart() montrant l'auth, le check d'état, un firstOrCreate([...]) et l'événement cartUpdated; Légende : « Idempotence et rafraîchissement ciblé. »

## 3) Retrait : méthode removeFromCart()

Lors du retrait d'un article du panier, le recalcul des états se fait via checkCartStates() et event cartUpdated, et un toast sobre “Retiré du panier” apparaît.

Preuve : petit extrait removeFromCart() et capture du toast. Légende : « Geste symétrique, état propre. »

#### 4) Mise en vente conclue : passage à sold et purge douce

Quand l'objet se vend, je bascule artifacts.status = 'sold'. Côté back, j'enchaîne une purge douce : je supprime en tâche courte toutes les entrées cart\_items correspondant à cet artifact\_id, pour désencombrer les paniers des autres utilisateurs.

Un artefact en sold affiche le badge “Vendu”, désactive le bouton, et remonte un message simple si l'utilisateur était en train d'interagir.

Preuves :

- capture de la fiche montrant “Vendu” (ton bloc existe déjà) ; Légende : « État final net, boutons inactifs. »
- extrait d'observer Listener/Job (ou méthode de service) qui fait la purge cart\_items par artifact\_id; Légende :« Purge douce : paniers mis à jour après vente. »

### Entrées propres : validation serveur systématique

Je ne laisse rien franchir la porte sans contrôle. Chaque requête passe par une validation côté serveur (Form Request ou règles Livewire), qui impose types, formats, bornes et cohérences avant toute persistance. L'objectif est double : assainir les données et éviter les états inattendus. Les messages restent sobres (pas d'informations internes) et collent aux champs concernés.

(Preuve : extrait d'une FormRequest ou d'un composant Livewire avec rules incluant required|mimes:jpg,jpeg,png,pdf|max:4096 pour l'upload identité ; capture d'une réponse 422 propre) — Légende : « Validation en amont : données nettes ou pas de données. »)

### Écriture maîtrisée : Mass Assignment verrouillé & casts explicites

Je n'ouvre que ce que je veux voir écrit. Les modèles exposent un \$fillable positif (jamais \$guarded=[]) pour interdire le remplissage massif de colonnes imprévues. J'ajoute des \$casts (dates, booléens, JSON) pour garantir des types stables à l'hydratation et à la sérialisation. Résultat : même une charge utile imaginative ne peut pas déborder le modèle.

(Preuve : capture du modèle Identity/Artifact montrant \$fillable ciblé et \$casts — Légende : « Écriture autorisée explicitement : pas d'assignation sauvage. »)

### Droit réel : Policies + rôle à l'entrée

Le droit ne se joue pas à l'affichage : il se décide au point d'action. Je m'appuie sur des Policies pour les gestes sensibles (lire une pièce d'identité, éditer un artefact), et j'emploie un gate pour l'espace admin. Côté routes, j'exige auth + verified avant d'entrer ; dans les contrôleurs, j'appelle \$this->authorize(...) avant de toucher à la ressource. Si l'interface se trompe, le serveur refuse net.

(Preuve : extrait IdentityPolicy@view, route admin avec middleware auth,verified + gate admin-area, contrôleur avec authorize('view', \$identity) ) — Légende : « Autorisation au bon endroit : impossible d'accéder sans droit. »)

## Injections SQL : comment je ferme la porte (Eloquent only)

Je ne concatène jamais de morceaux de SQL avec des entrées utilisateur. Tout passe par Eloquent, qui génère des requêtes paramétrées (bindings PDO) : les valeurs sont liées au statement et traitées comme données, jamais comme code. Même si un champ contient des caractères “agressifs”, ils restent inoffensifs parce qu’ils ne sont pas injectés dans la chaîne SQL. Deuxième filet, je valide toujours les entrées avant de m’en servir (type, format, longueur, existence en base). Ainsi, ce qui parvient jusqu’au moteur SQL est attendu et raisonnable.

## Documents sensibles : stockage privé & stream binaire

Les pièces d'identité restent hors du répertoire public. Je les dépose sur un disque privé (storage/app/private) et je ne stocke en base que chemin, MIME et taille. Pour consulter, je stremme le binaire après autorisation, avec des en-têtes qui empêchent le cache et ne divulguent rien du chemin interne :

- Content-Disposition: inline (affichage contrôlé),
- Cache-Control: private, no-store (pas de persistance navigateur),
- Content-Type issu du stockage.
- Chaque accès laisse une trace minimale (qui, quoi, quand) pour reconstituer les faits sans dupliquer la donnée.
- (Preuve : diff config/filesystems.php (disque private), capture onglet Network d'une consultation montrant les en-têtes, extrait du contrôleur de stream + authorize() ) —  
Légende : « Fichiers sensibles hors public, servis à la demande et sans cache. »)

## Règles d'enchère : transactions et états cohérents

L'enchère n'admet pas les états bancals. Je encapsule la mise dans une transaction : je lis l'offre courante, j'applique les règles (pas minimal, refus d'égalité), j'enregistre ; si quelque chose cloche, je rollback. On évite ainsi les offres fantômes et les courses de variables. Les messages côté utilisateur restent clairs sans exposer l'arrière-boutique.

(Preuve : extrait de méthode placeBid() avec DB::transaction(...) et contrôle amount > current ; test vert associé) — Légende : « Tout passe ou rien ne passe : cohérence garantie. »)

## Sessions fiables : anti-fixation, cookies httpOnly/secure/SameSite

La confiance vit côté serveur. Je range les sessions en base (driver database) pour pouvoir les invalider proprement et je régénère l'ID au login (request()->session()->regenerate()), ce qui bloque la fixation de session. Les cookies sont chiffrés, marqués HttpOnly (inaccessibles au JS), secure en HTTPS, et SameSite=Lax (ou Strict si possible). J'utilise des noms distincts par environnement pour éviter les collisions.

(Preuve : config/session.php (http\_only, secure, same\_site), .env avec SESSION\_DRIVER=database, capture table sessions + diff montrant la régénération à la connexion) — Légende : « Session serveur, cookie protégé : isolement garanti. »)

### Bruit contenu : Rate Limiting sur les chemins sensibles

Je mets un frein là où les rafales sont nuisibles : login, upload d'identité, tentatives répétées. Le rate limiting ralentit les essais rapprochés (réponse 429 + Retry-After) sans gêner l'usage normal. Cela réduit à la fois le bruit dans les logs et le risque d'abus.

(Preuve : extrait de définition RateLimiter (ex. for('login', ...)) et routes avec ->middleware('throttle:login'), capture d'une 429 en Network) — Légende : « Rafales freinées, usage légitime fluide. »)

### Secrets au chaud : .env non versionné & .env.example exhaustif

Les secrets et la configuration ne quittent pas l'environnement. Le fichier .env n'est jamais versionné ; le .env.example est exhaustif (sans valeurs sensibles) pour rejouer l'installation partout à l'identique. Je sais que changer APP\_KEY invalide toutes les sessions : c'est un acte rare, préparé et annoncé.

(Preuve : capture côté à côté .env.example (complet) et .env (flouté), commit montrant .env dans .gitignore) — Légende : « Reproductibilité sans divulgation : secrets hors dépôt. »)

### Traces utiles : journaux mesurés, jamais bavards

Je garde des logs ciblés : échecs répétés d'authentification, consultation d'un document d'identité, refus d'enchère pour règle violée. Je n'y mets aucune donnée personnelle en clair. Le but n'est pas d'accumuler, mais de comprendre ce qui s'est passé, quand et par qui, pour agir vite si nécessaire.

(Preuve : extrait d'une entrée “identity.accessed” (user\_id, identity\_id, timestamp) ; niveau INFO) — Légende : « Tracer sans exposer : juste ce qu'il faut pour comprendre. »)

## XII. Sécurité de l'application

### XII.1 Authentification, rôles & Policies

Mon intention : je veux une authentification sobre, des rôles clairs, et des accès stricts sur les zones sensibles (admin, fichiers d'identité, logique d'enchaînement). L'UX reste douce, mais c'est le serveur qui tranche.

#### Authentification (Volt + sessions)

Les écrans Auth sont des composants Volt : la classe anonyme en tête de la vue porte l'état, les règles et la méthode login() (validation, RateLimiter, Auth::attempt, Session::regenerate).

- Session::regenerate() coupe net les attaques de fixation de session.
- Si remember est coché, Laravel émet le cookie remember\_web\_... HttpOnly/SameSite.

### Routes Volt (extrait de routes/auth.php)

```

Volt::route('login', 'auth.login')
    ->name('login');

Volt::route('register', 'auth.register')
    ->name('register');

```

Dans Safari DevTools → Network, au submit, je vois POST /livewire/update (payload JSON), 200 OK, et les Set-Cookie (nouveau XSRF-TOKEN, session régénérée, et remember).

```

Résumé
URL: http://arte-facto-v2.test/livewire/update
État: 200 OK
Source: Réseau
Adresse: 127.0.0.1:80
Initiateur: [www.rete-test.d29d (8)]
Content-Type: application/json
Cookie: XSRF-TOKEN=eyJpdiI6InZldkIjMmJhc2g3MmIQRWp5TDA4dnctPSInInZhbHVlciVmdKVKdqlkodS2MxLQOOGVYInBjIc2My73NVhsd2N4U1ZMZG9TcEgyJslpC9MRUR0TUhnInBjSEIGWOfQZNNZWhRjhpCVFRtXRMRWZPUgw2MzJcl3hTHRpSiFUmZ3NvF0b0llWSGUINHVP5X0209w2NZDjY2FeK3l6ElCjYXVmM0lyMjx0Gm2MfR0T1k1MwVMhDnD0Cq2zT1wMfRM4NvMfIM04NwElyNg5tYAz2ODVNgzT1fRNzMuT0ESTsZGJhwiGfnpjlnh%3D; arte_facto_session=eyJpdiI6IpOkpaz25WV2Yx5Ex3jRl04jMU9PSplnZhnhvJlojvn3ksJqdh0k3hcdn2864rwlp50RnTowdNzNGV4fzbmNsZmfls194mzfYzTbUlrs3h1NkZ1hRdC96Z3pXTHnsz09h53hPwlfWfM2vMY2l0bV1u4eRkrKN0tWwN0JiahxYytlUTz050UcLcJYwMj0lJzmqzTgqY3N2f0MfY4M0j9YEx722NGR9ZDc4Mf1g4M0nN0t5Mw1sNT21yM2Df0fZG0U1Mj1NwEhGfnpjlnh%3D
Origin: http://arte-facto-v2.test
Priority: url;
Referer: http://arte-facto-v2.test/login
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_16_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/10.6 Safari/605.1.15
X-Client

Requête
POST /livewire/update HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR;fr;q=0.9
Connection: keep-alive
Content-Length: 540
Content-Type: application/json
Cookie: XSRF-TOKEN=eyJpdiI6InZldkIjMmJhc2g3MmIQRWp5TDA4dnctPSInInZhbHVlciVmdKVKdqlkodS2MxLQOOGVYInBjIc2My73NVhsd2N4U1ZMZG9TcEgyJslpC9MRUR0TUhnInBjSEIGWOfQZNNZWhRjhpCVFRtXRMRWZPUgw2MzJcl3hTHRpSiFUmZ3NvF0b0llWSGUINHVP5X0209w2NZDjY2FeK3l6ElCjYXVmM0lyMjx0Gm2MfR0T1k1MwVMhDnD0Cq2zT1wMfRM4NvMfIM04NwElyNg5tYAz2ODVNgzT1fRNzMuT0ESTsZGJhwiGfnpjlnh%3D; arte_facto_session=eyJpdiI6IpOkpaz25WV2Yx5Ex3jRl04jMU9PSplnZhnhvJlojvn3ksJqdh0k3hcdn2864rwlp50RnTowdNzNGV4fzbmNsZmfls194mzfYzTbUlrs3h1NkZ1hRdC96Z3pXTHnsz09h53hPwlfWfM2vMY2l0bV1u4eRkrKN0tWwN0JiahxYytlUTz050UcLcJYwMj0lJzmqzTgqY3N2f0MfY4M0j9YEx722NGR9ZDc4Mf1g4M0nN0t5Mw1sNT21yM2Df0fZG0U1Mj1NwEhGfnpjlnh%3D
Origin: http://arte-facto-v2.test
Priority: url;
Referer: http://arte-facto-v2.test/login
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_16_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/10.6 Safari/605.1.15
X-Client

Réponse
HTTP/1.1 200 OK
Cache-Control: max-age=0, must-revalidate, no-cache, no-store, private
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json
Date: Fri, 19 Sep 2023 08:22:58 GMT
Expires: Fri, 01 Jan 1990 00:00:00 GMT
Pragma: no-cache
Server: nginx/1.4
Set-Cookie: XSRF-TOKEN=eyJpdiI6InZldkIjMmJhc2g3MmIQRWp5TDA4dnctPSInInZhbHVlciVmdKVKdqlkodS2MxLQOOGVYInBjIc2My73NVhsd2N4U1ZMZG9TcEgyJslpC9MRUR0TUhnInBjSEIGWOfQZNNZWhRjhpCVFRtXRMRWZPUgw2MzJcl3hTHRpSiFUmZ3NvF0b0llWSGUINHVP5X0209w2NZDjY2FeK3l6ElCjYXVmM0lyMjx0Gm2MfR0T1k1MwVMhDnD0Cq2zT1wMfRM4NvMfIM04NwElyNg5tYAz2ODVNgzT1fRNzMuT0ESTsZGJhwiGfnpjlnh%3D; arte_facto_session=eyJpdiI6IpOkpaz25WV2Yx5Ex3jRl04jMU9PSplnZhnhvJlojvn3ksJqdh0k3hcdn2864rwlp50RnTowdNzNGV4fzbmNsZmfls194mzfYzTbUlrs3h1NkZ1hRdC96Z3pXTHnsz09h53hPwlfWfM2vMY2l0bV1u4eRkrKN0tWwN0JiahxYytlUTz050UcLcJYwMj0lJzmqzTgqY3N2f0MfY4M0j9YEx722NGR9ZDc4Mf1g4M0nN0t5Mw1sNT21yM2Df0fZG0U1Mj1NwEhGfnpjlnh%3D; express=fr, 19 Sep 2023 10:22:59 GMT; Max-Age=7200; path=/; sameSite=strict
Set-Cookie: arte_facto_session=eyJpdiI6InZldkIjMmJhc2g3MmIQRWp5TDA4dnctPSInInZhbHVlciVmdKVKdqlkodS2MxLQOOGVYInBjIc2My73NVhsd2N4U1ZMZG9TcEgyJslpC9MRUR0TUhnInBjSEIGWOfQZNNZWhRjhpCVFRtXRMRWZPUgw2MzJcl3hTHRpSiFUmZ3NvF0b0llWSGUINHVP5X0209w2NZDjY2FeK3l6ElCjYXVmM0lyMjx0Gm2MfR0T1k1MwVMhDnD0Cq2zT1wMfRM4NvMfIM04NwElyNg5tYAz2ODVNgzT1fRNzMuT0ESTsZGJhwiGfnpjlnh%3D; express=fr, 19 Sep 2023 10:22:58 GMT; Max-Age=7200; path=/; sameSite=strict
Set-Cookie: XSRF-TOKEN=eyJpdiI6InZldkIjMmJhc2g3MmIQRWp5TDA4dnctPSInInZhbHVlciVmdKVKdqlkodS2MxLQOOGVYInBjIc2My73NVhsd2N4U1ZMZG9TcEgyJslpC9MRUR0TUhnInBjSEIGWOfQZNNZWhRjhpCVFRtXRMRWZPUgw2MzJcl3hTHRpSiFUmZ3NvF0b0llWSGUINHVP5X0209w2NZDjY2FeK3l6ElCjYXVmM0lyMjx0Gm2MfR0T1k1MwVMhDnD0Cq2zT1wMfRM4NvMfIM04NwElyNg5tYAz2ODVNgzT1fRNzMuT0ESTsZGJhwiGfnpjlnh%3D; express=fr, 19 Sep 2023 10:22:58 GMT; Max-Age=7200; path=/; sameSite=strict
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Powered-By: PHP/8.4.11

```

## Rôles & middleware

Je garde trois rôles : visiteur (non connecté), utilisateur, admin. Les routes admin sont groupées avec ['auth','verified','admin'] :

- auth : session valide
- verified : email confirmé pour les zones critiques
- admin : middleware maison IsAdmin (ex. auth()->user()->isAdmin()).

```
Route::prefix('admin')->middleware(['auth', 'verified', 'admin'])->group(function () {  
    // Dashboard admin  
    Route::get('/dashboard', function () {  
        return view('admin.dashboard');  
    })->name('admin.dashboard');  
  
    Route::get('/stats', function () {  
        return view('admin.stats');  
    })->name('admin.stats');  
});
```

routes/web.php sur le groupe admin

## Policies (droits fins, ex. fichiers d'identité)

Même si le middleware admin suffit pour ta démo, je pose une Policy pour documenter l'intention : “qui peut lire la pièce d'identité d'un utilisateur ?”.

```
app/Policies/IdentityPolicy.php  
php Copier le code  
  
class IdentityPolicy  
{  
    public function view(User $viewer, User $subject): bool  
    {  
        return $viewer->isAdmin(); // Optionnel: ou $viewer->id === $subject->id  
    }  
}  
  
AuthServiceProvider  
php Copier le code  
  
protected $policies = [ User::class => IdentityPolicy::class ];
```

Preuve : capture d'un extrait de Policy à faire ou du middleware IsAdmin.

## XIII. Conclusion

Le projet Arte Facto a été bien plus qu'un simple exercice de développement. Il a été une aventure complète, une plongée dans la création d'un univers technique et artistique cohérent, pensé de bout en bout.

J'ai conçu ce projet seule, en partant d'une idée originale jusqu'à sa réalisation technique, en passant par la modélisation, le design, la programmation, la logique métier, la sécurité, et le déploiement.

J'ai dû faire preuve d'autonomie, de rigueur, de curiosité, mais aussi de créativité, pour donner vie à un projet qui soit à la fois fonctionnel, immersif, et professionnel.

Grâce à Arte Facto, j'ai pu :

- Mettre en œuvre l'ensemble des compétences techniques du référentiel,
- Approfondir mon usage de Laravel, Livewire, Tailwind et Git,
- Apprendre à surmonter des problèmes concrets, parfois complexes,
- Et surtout, donner naissance à une application qui me ressemble : poétique, structurée, interactive.

Je retiens que développer, c'est créer du lien entre l'idée et l'expérience utilisateur. C'est transformer une pensée abstraite en interface tangible. C'est aussi un processus parfois long, exigeant, mais profondément gratifiant.

Ce projet m'a permis de gagner en :

- Confiance technique, par l'affirmation de mon autonomie,
- Vision métier, en pensant toujours à l'utilisateur final,
- Capacité à expliquer mon travail, ce que ce dossier reflète.

Je suis fière de ce que j'ai accompli seule, et impatiente de poursuivre cette aventure professionnelle avec de nouveaux projets, de nouvelles équipes, et une passion toujours aussi vive pour le développement web et la narration numérique.

"Arte Facto est un monde codé, une passerelle entre les civilisations imaginées et les technologies d'aujourd'hui. Il n'est pas simplement une application... mais une œuvre numérique vivante."

## XIV. Remerciements

Avant de refermer ce dossier, je tiens à exprimer ma gratitude profonde envers celles et ceux qui ont, de près ou de loin, accompagné la création d'Arte Facto, dans la lumière comme dans l'ombre.

Je remercie tout d'abord l'équipe pédagogique de la formation, et en particulier Nicolas, pour sa bienveillance, ses retours, sa patience et ses regards professionnels qui m'ont souvent permis de reprendre confiance et de recentrer mon travail.

Je remercie les intervenants techniques et leurs partages de savoir, qui m'ont poussée à dépasser mes limites, à comprendre les rouages plus profonds du web, et à devenir peu à peu une développeuse plus complète, plus autonome, plus consciente.

Je remercie également mes proches, pour leur soutien silencieux mais constant : leur écoute, leurs encouragements, leur présence lorsque le doute s'invitait ou que le code semblait trop capricieux.

Je pense aussi à mes camarades de formation, avec qui j'ai parfois partagé des moments de rires, parfois de frustration, mais toujours une aventure humaine commune autour d'un rêve numérique.

Et puis, je me remercie moi-même, humblement, pour avoir tenu bon.

Pour avoir appris à coder comme on apprend une langue, avec ses grammaires, ses silences, ses exceptions. Pour avoir transformé une idée floue en monde visible. Pour avoir persisté là où d'autres auraient lâché.

Enfin, un merci tout particulier à Merlin, mon mentor imaginaire et compagnon de route virtuel, dont la sagesse, les encouragements et les réponses magiques ont éclairé ma route à chaque ligne de ce grimoire.

"Il y a dans chaque ligne de code, un peu de nous-même.

Et dans chaque projet abouti, un éclat de victoire silencieuse."