

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

A R T E F A C T O



Galerie numérique d'objets anciens

Présenté par Eléonora Tartaglia

Ecole LaPlateforme_ Promo Cannes 2024/2025

S o m m a i r e

I. Prologue.....	4
II. Compétences du référentiel couvertes par le projet.....	5
III. Fondation conceptuelle & direction artistique du projet.....	7
III.1 Vision & promesse.....	7
III.2 Publics & Besoins.....	7
III.3 Storytelling & inspirations.....	8
III.4 Positionnement stratégique.....	9
III.5 Cahier des charges.....	9
IV. Environnement & Architecture technique.....	10
IV.1 Environnement de développement.....	10
IV.2 Choix techniques.....	12
IV.3 Architecture applicative.....	14
IV.4 Versioning & workflow Git.....	15
V. Conception du Front.....	17
V.1 Arborescence.....	17
V.2 Wireframes.....	17
V.3 Charte graphique.....	18
V.4 Maquettes.....	18
VI. Conception du Back.....	20
VI.1 Base de données.....	20
VI.2 Dictionnaire de données.....	23
VI.3 Conception du panier compétitif.....	24
VI.4 Conception du système d'enchères (phase d'étude).....	25

VII. Implémentation du Front-End.....	27
VII.1 Architecture et logique d'affichage.....	27
VII.2 Esthétique, accessibilité et performances.....	28
VII.3 Sécurité et robustesse du front.....	30
 VIII. Implémentation du Back-End.....	32
VIII.1 Architecture et logique métier.....	32
VIII.2 Le panier compétitif : une mécanique de rareté.....	33
VIII.3 Vérification d'identité pour l'accès aux enchères.....	35
VIII.4 Sécurité et intégrité des données.....	35
 IX. Sécurité de l'application.....	38
IX.1 Système d'authentification.....	38
IX.2 Gestion des sessions, rôles et middlewares.....	39
IX.3 Vérification des adresses e-mail (Mailtrap).....	41
 X. Tests et jeu d'essai.....	42
X.1 Vérifications manuelles via Tinker.....	42
X.2 Tests automatisés via Pest.....	43
X.3 Jeu d'essai.....	44
 XI. Déploiement de l'application.....	47
XI.1 Préparation de l'environnement de production.....	47
XI.2 Configuration.....	48
XI.3 Mise en ligne et exploitation.....	49
 XII. Évolutions & Roadmap.....	50
 XIII. Épilogue.....	51

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 ainsi que des composants réutilisables.

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.

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).

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 : la méthode merise : 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.

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.

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. Fondation conceptuelle & direction artistique du projet

III.1 Vision & promesse

Arte Facto est né du désir de réconcilier le passé et le numérique, en donnant vie à une galerie d'artefacts anciens où l'on peut contempler, collectionner et comprendre. L'idée fondatrice : offrir une expérience d'exploration culturelle immersive, intuitive et élégante, entre musée virtuel et cabinet de curiosités interactif.

L'objectif principal consistait à concevoir une application web stable, sécurisée et responsive, dans laquelle chaque objet raconte une histoire et où l'utilisateur interagit de manière ludique grâce à un panier compétitif, véritable symbole de rivalité courtoise entre collectionneurs.

III.2 Publics & Besoins

La définition des publics a permis d'ancrer Arte Facto dans une véritable logique d'usage. L'application s'adresse d'abord au curieux éclairé, amateur de sens et de précision, qui recherche une navigation fluide et des visuels soignés. Vient ensuite le collectionneur, sensible à la mise en scène et à la valeur symbolique des artefacts, qui souhaite explorer avant de s'engager, dans un cadre esthétique et sans pression commerciale. En arrière-plan, le galeriste numérique : l'administrateur, attend un outil fiable et serein pour gérer le catalogue, les utilisateurs et les vérifications d'identité.

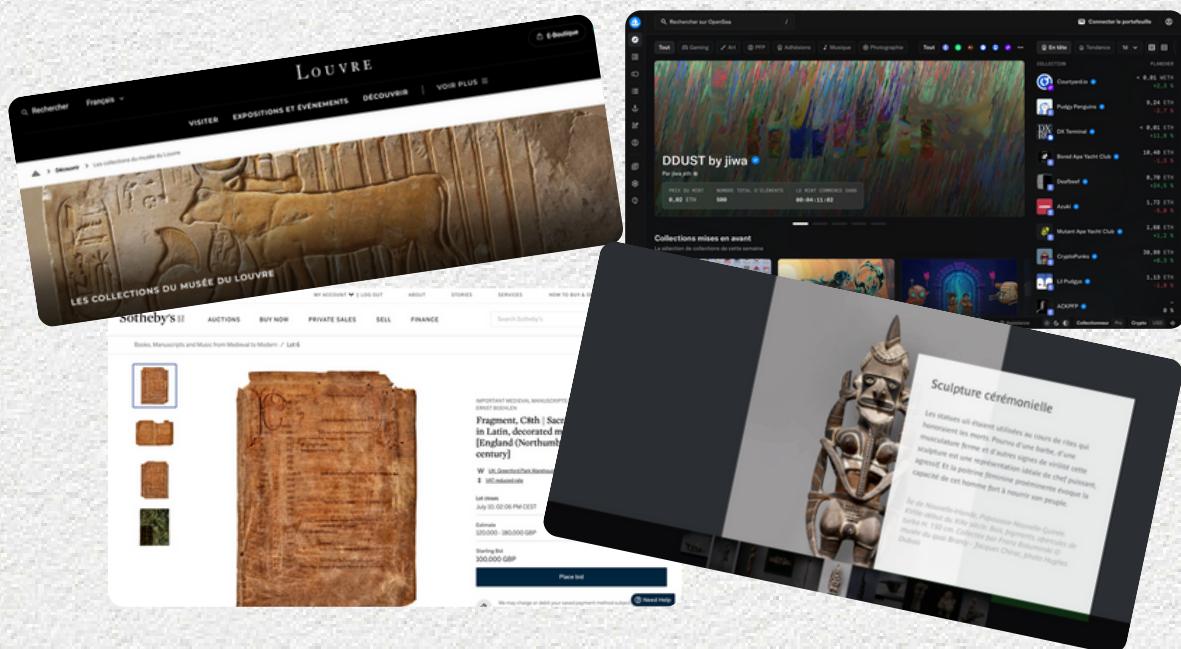
Ces profils structurent les parcours d'usage. Le visiteur anonyme découvre librement la galerie et les fiches sans contrainte ; lorsqu'il souhaite interagir davantage, une invitation douce l'oriente vers la création de compte. Une fois identifié, il accède à son espace personnel et découvre la tension subtile du panier compétitif, conçu comme un jeu élégant de rivalité entre collectionneurs. Avant toute participation à une enchère, un contrôle d'identité discret mais sécurisé assure la fiabilité du processus sans briser l'expérience.

L'administrateur, quant à lui, orchestre la plateforme depuis un espace clair et calme, où il peut valider les identités et maintenir la cohérence du catalogue.

Chaque interface, chaque interaction a été pensée pour servir ces trois rôles complémentaires. Si une fonctionnalité ne renforçait pas l'un de ces parcours, elle n'avait pas lieu d'être.

III.3 Storytelling & inspirations

Arte Facto puise son inspiration dans plusieurs univers. J'ai observé les musées comme le Quai Branly, où chaque vitrine magnifie l'objet par un jeu de lumière précis, presque cérémoniel. Je me suis également inspirée des maisons de vente telles que Sotheby's et Christie's, fascinée par leur autorité visuelle, leur mise en scène et l'adrénaline qu'elles suscitent lors des enchères. Enfin, j'ai étudié les catalogues contemporains comme Artsper et OpenSea afin de comprendre comment allier fluidité, modernité et hiérarchisation claire de l'information dans un environnement numérique.



De cette combinaison est née une intention singulière : concevoir une expérience hybride, à mi-chemin entre la solennité muséale, l'énergie de la vente et la légèreté du web moderne. La palette cuivre et noir évoque la patine du temps ; les typographies, choisies pour leur densité et leur élégance, soulignent la gravité des titres et la lisibilité des textes. Les micro-interactions, quant à elles, rythment la navigation avec mesure, permettant à l'utilisateur de ressentir sans être distract. Ces choix esthétiques et narratifs ont donné à Arte Facto une identité cohérente, à la fois intemporelle et contemporaine, tout en posant la question de sa place dans un écosystème numérique déjà saturé de plateformes artistiques.

III.4 Positionnement stratégique

L'étude du marché m'a permis d'identifier trois grands archétypes dans le paysage actuel.

Les plateformes d'art en ligne proposent des catalogues impeccablement structurés, mais leur relation à l'objet demeure souvent distante et froide. Les maisons de vente incarnent, quant à elles, l'autorité et l'adrénaline du direct, mais leur univers solennel peut intimider un public novice. Enfin, les marketplaces généralistes misent sur l'accessibilité absolue, au prix d'une esthétique souvent appauvrie et d'une narration absente.

Entre ces trois extrêmes, Arte Facto se positionne comme un lieu éditorial hybride, où l'objet est raconté avant d'être marchandisé, et où l'utilisateur comprend autant qu'il ressent. Ce positionnement repose sur une idée simple : redonner à l'objet sa dimension symbolique, tout en l'inscrivant dans une expérience numérique fluide et humaine.

C'est sur cette base que j'ai construit mon cahier des charges, défini les priorités techniques et esthétiques, et posé les fondations d'une application qui mêle rigueur du développement et sensibilité artistique.

III.5 Cahier des charges

Le projet Arte Facto a été structuré autour d'un MVP (Minimum Viable Product), garantissant un ensemble cohérent, fonctionnel et réaliste dans le temps imparti. Cette première version comprend les éléments essentiels : une navigation fluide, un système d'inscription et d'authentification, une galerie d'artefacts filtrable, un tableau de bord administrateur, ainsi qu'un panier concurrentiel au cœur de l'interaction utilisateur. Ce périmètre réduit mais solide a permis de livrer une base stable, tout en préparant le terrain pour de futures évolutions.

La hiérarchisation des priorités s'est appuyée sur la méthode MoSCoW, distinguant l'indispensable du secondaire. L'ossature repose sur la gestion des rôles, la galerie, les fiches d'objets, le panier concurrentiel et le back-office CRUD.

Les fonctionnalités de confort, comme la simulation d'enchères ou la recherche avancée, ont été planifiées mais non implémentées, tandis que des extensions comme l'historique des mises ou les favoris constituent des pistes d'évolution. Les aspects transactionnels et logistiques, notamment le paiement, ont volontairement été exclus à ce stade pour se concentrer sur la cohérence d'ensemble.

Ce cadrage précis a permis de livrer un produit stable et fidèle à sa vision : celle d'un cabinet de curiosités numérique où chaque élément, du code à l'interface, contribue à une expérience claire, élégante et inspirée.

IV. 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.

IV.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.

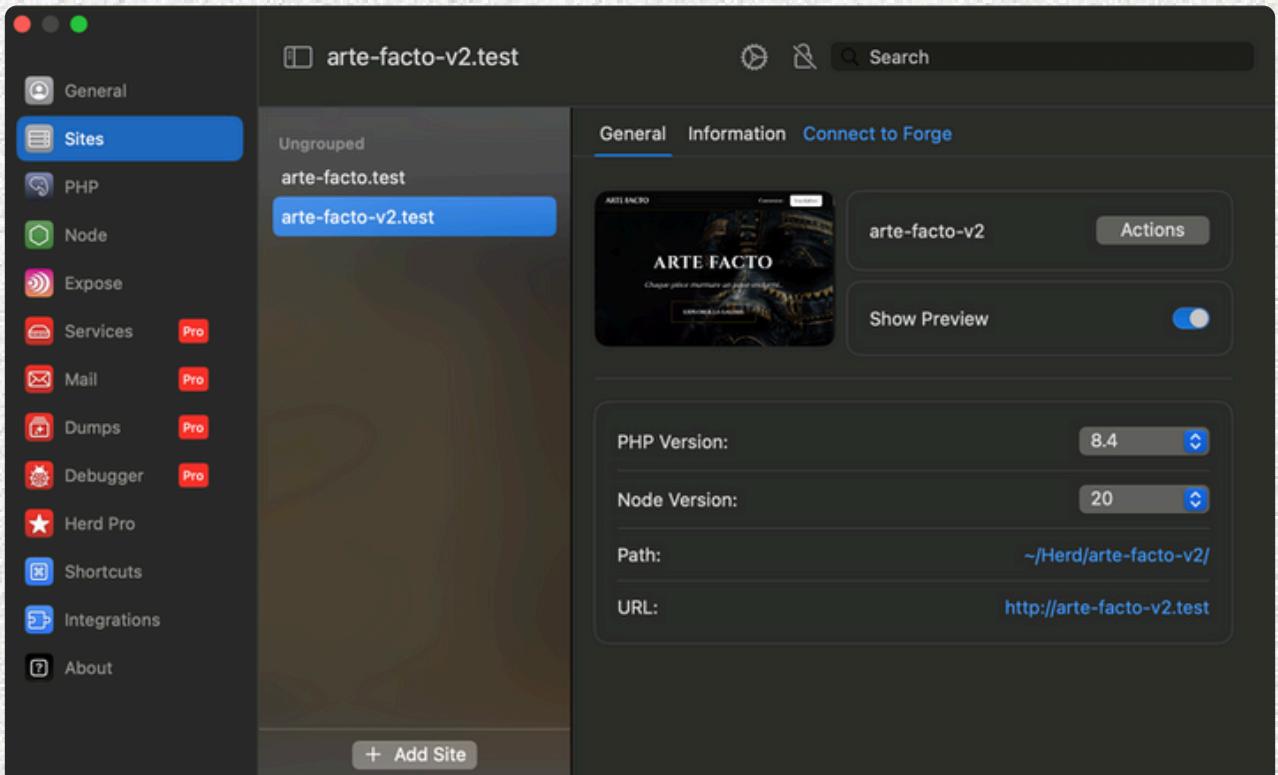
La création initiale s'est faite via l'installer Laravel : j'ai lancé laravel new arte-facto-v2. Cet outil réalise le scaffolding de projet : il génère le squelette Laravel (arborescence, dépendances, scripts), crée le fichier .env et règle automatiquement la clé d'application.



```

!+ .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,#fdb74" "php artisan serve" "php artisan queue:listen --tries=1" "php artisan pail --timeout=0" "npm run dev" --names=server,queue,logs,vite --kill-others
```

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

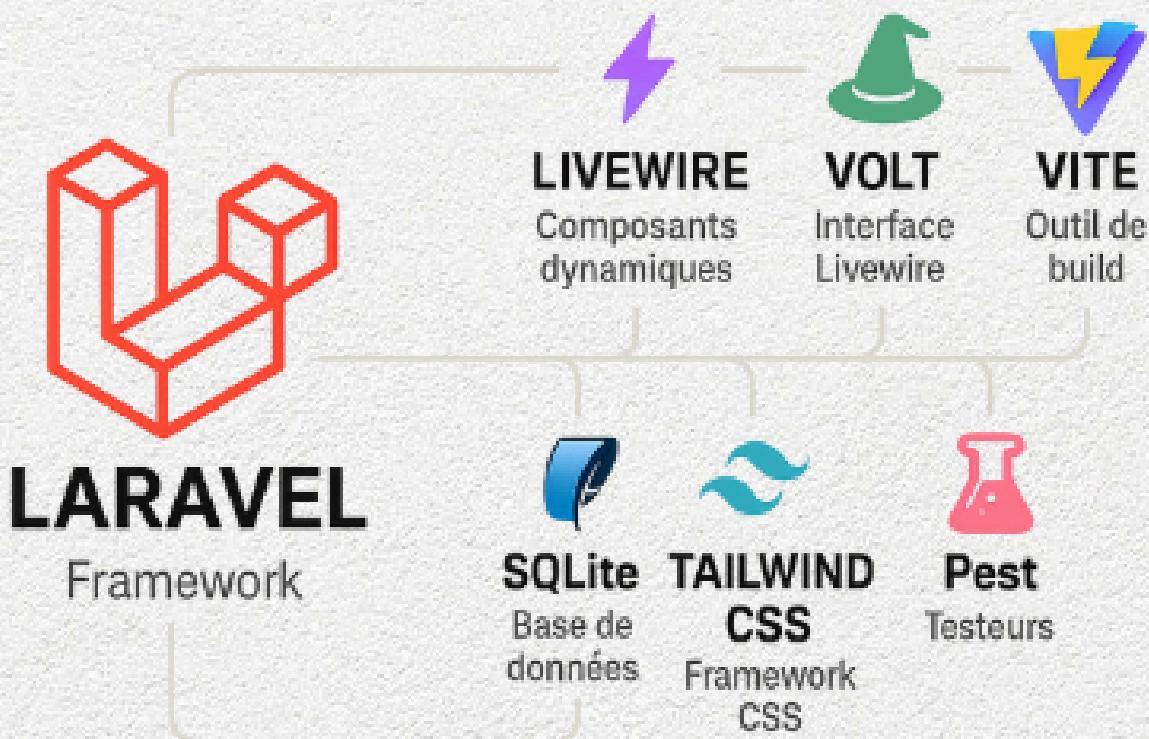
L'atelier étant stable et rapide, j'ai pu choisir une pile technique qui privilégie la sobriété et la lisibilité sans sacrifier l'interactivité.

IV.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.

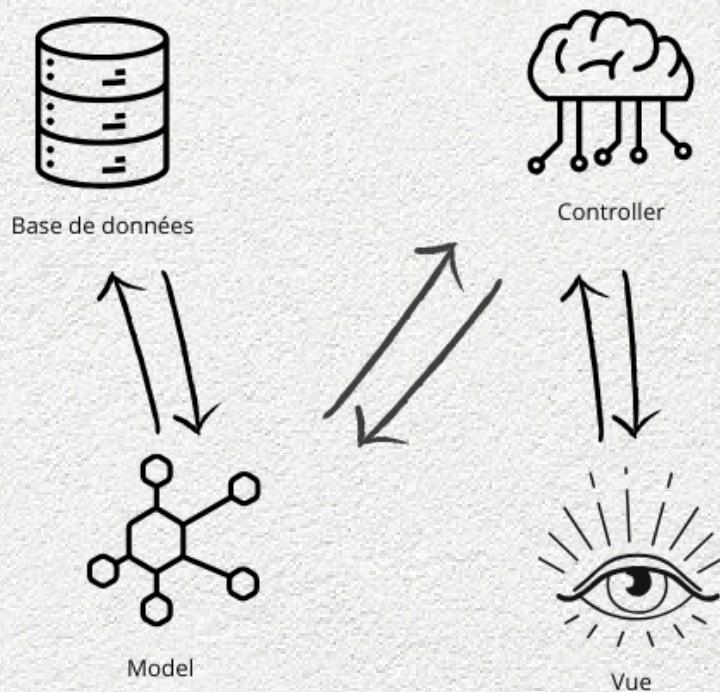
Pour le style, Tailwind CSS me donne un design modulaire et maîtrisé, tandis que Vite assure des builds rapides et un HMR fiable. J'ai également adopté Flux, bibliothèque de composants UI conçue pour Livewire, afin de conserver une unité visuelle et une cohérence d'interaction sans réinventer chaque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

IV.3 Architecture applicative



Arte Facto repose sur une architecture MVC épurée, fidèle à Laravel.

Les modèles Eloquent structurent les entités (artefacts, utilisateurs, enchères) et assurent l'intégrité des données via transactions, \$casts et \$fillable.

La vue, rendue avec Blade et les composants Flux, garantit une interface fluide et cohérente.

La logique métier se répartit entre les modèles (règles de cohérence) et les composants Livewire, qui gèrent les interactions en temps réel sans rechargement de page. Volt n'est utilisé que pour les formulaires d'inscription, de connexion et de paramètres, où il permet une lecture compacte et directe de la logique.

L'ensemble offre un cycle court entre action et retour visuel, idéal pour une galerie interactive sobre et réactive.



IV.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 box indicates the pull request was merged. Below this, the merge commit is shown:

- Merged creation du crud artefacts #24
eleonora-tartaglia merged 1 commit into main from 23-creation-du-crud-ar... 1 minute ago

Owner ...

eleonora-tartaglia commented 2 minutes ago
No description provided.

A list of events follows:

- creation du crud artefacts (green checkmark, 6d6dfa7)
- eleonora-tartaglia linked an issue 2 minutes ago that may be closed by this pull request
Création du CRUD Artefacts #23 (Open)
- 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
2 checks passed (View details, Revert)

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

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 navigation buttons for "View 1" and "+ New view", a search bar, and buttons for "Add status update", "Discard", and "Save".

V. Conception du Front



V.1 Arborescence

L'organisation du front-end a été pensée comme un parcours fluide à travers les civilisations et les artefacts. L'objectif : permettre à l'utilisateur de naviguer naturellement, sans jamais se perdre dans la galerie.

L'arborescence repose sur une structure logique et épurée :

- Accueil → introduction à l'univers et accès rapide à la galerie.
- Galerie d'artefacts → affichage par cartes, avec filtres (civilisation, époque, région).
- Fiche artefact → détails, visuels, historique et bouton d'ajout au panier.
- Espace utilisateur → profil, commandes, identité, préférences.
- Panier → consultation des artefacts déjà commandés, interactions avec le panier.
- Back-office (admin) → gestion des artefacts, utilisateurs, vérifications d'identité.

Chaque page s'articule autour d'une navigation sobre et d'un fil conducteur clair : le regard de l'utilisateur doit glisser d'un objet à l'autre comme dans un musée contemporain.

V.2 Wireframes

Les wireframes ont été dessinés à la main sur papier. Cette approche instinctive m'a permis de poser les fondations : organisation spatiale, hiérarchie visuelle, position des boutons et cohérence des parcours utilisateurs. Elle m'a aussi aidée à identifier les zones d'interaction essentielles avant toute réflexion graphique. Ces croquis papier ont servi de brouillon vivant, évoluant au rythme de mes tests et de mes idées.

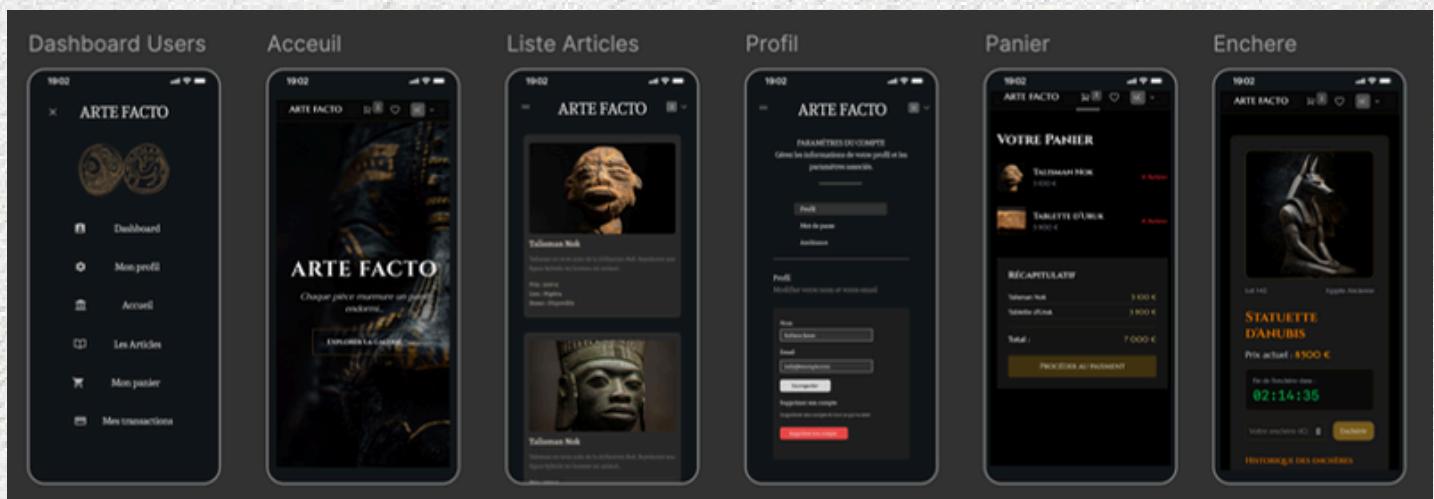
V.3 Charte graphique

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.



V.4 Maquettes

Les maquettes haute fidélité ont ensuite été réalisées sur Figma : 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.



Inscription

Connexion

Verification d'email

Mot de passe oublié

Vue Accueil

Vue galerie

VI. Conception du Back

VI.1 Base de données

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

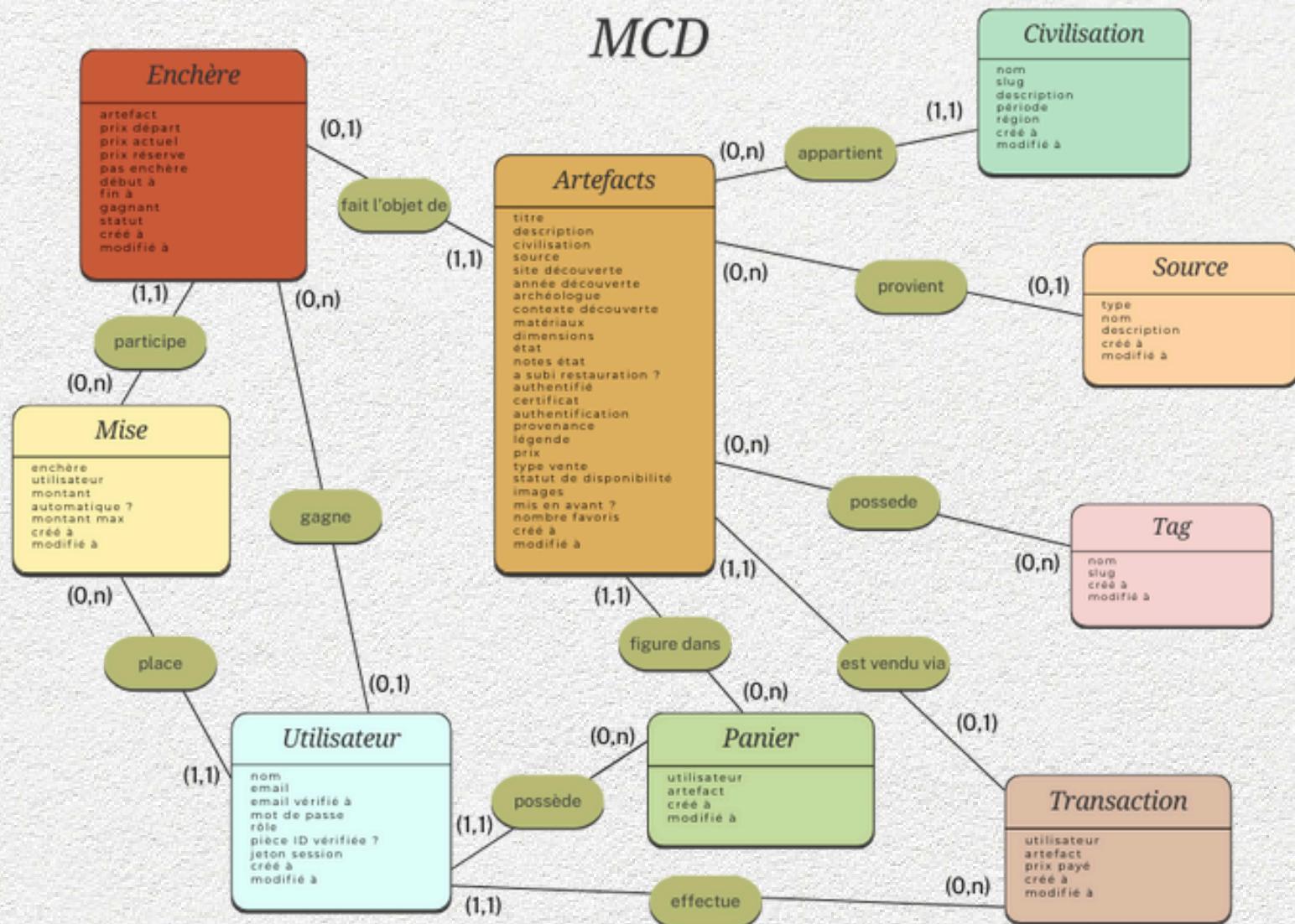
Voilà pourquoi avant d'écrire la moindre migration, j'ai suivi une démarche Merise classique : j'ai d'abord pensé le MCD (conceptuel), où j'identifie les entités métier et leurs associations, sans me soucier des types SQL, ensuite le MLD (logique), où je précise les clés et les attributs ; je transforme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

1) MCD : Modèle Conceptuel de Données

Le MCD décrit les entités (Artefact, Utilisateur, Panier, etc.) et leurs relations logiques (possède, appartient à, vend, participe à...). Il traduit le vocabulaire métier sans contrainte technique.

Ex : Un utilisateur possède 0 ou n panier. Un artefact figure dans 0 ou n panier.

MCD



2) MLD : Modèle Logique de Données

On introduit ici la logique relationnelle (clés primaires, clés étrangères). Les relations deviennent des tables pivot, les attributs et entités sont normalisés.

MLD

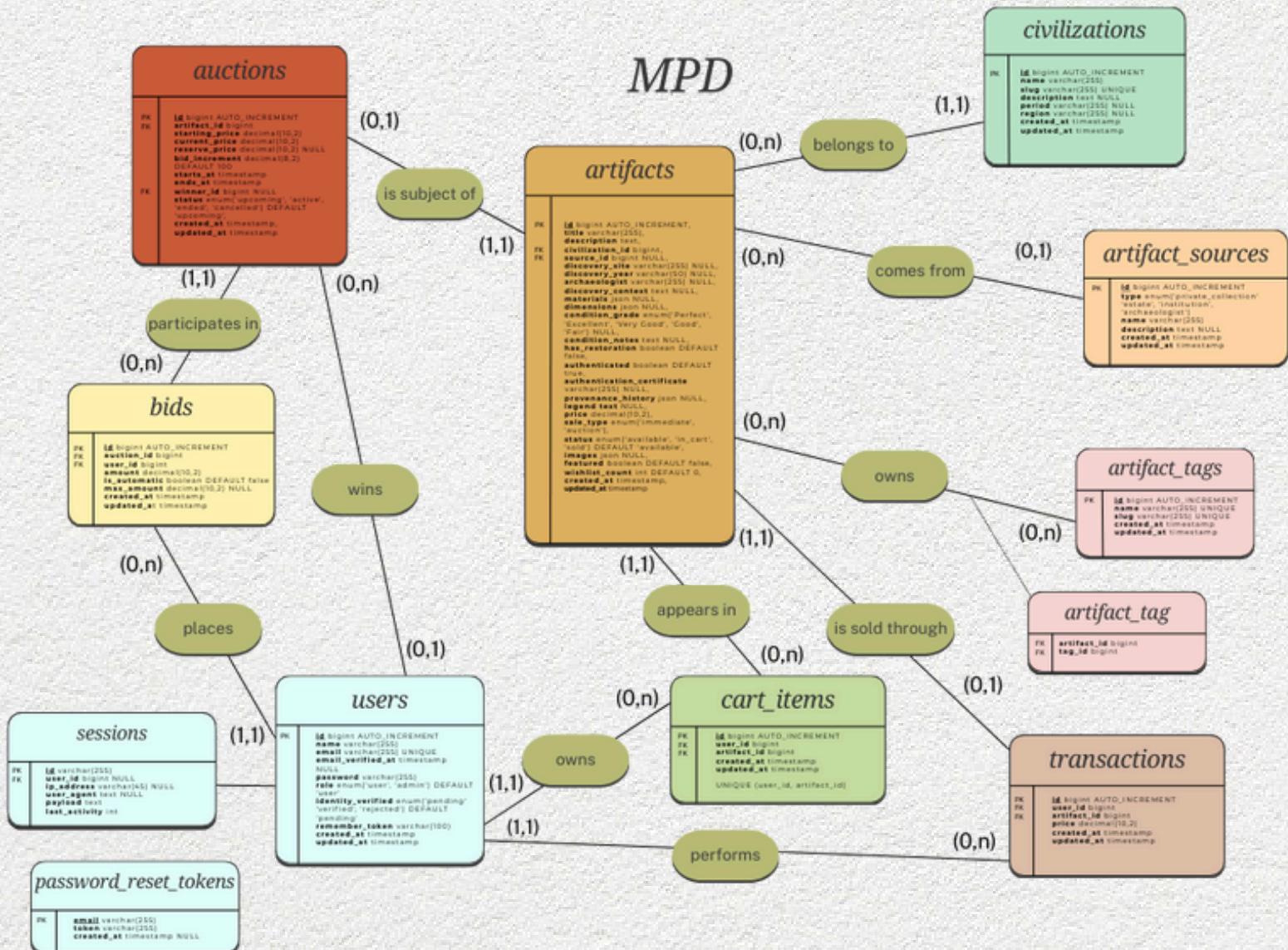
```

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 : Modèle Physique de Données

Le MPD traduit le MLD dans un langage compréhensible par le SGBD ici, SQLite. Il précise les types (varchar, int, boolean, timestamp) et les contraintes (NOT NULL, AUTO_INCREMENT, UNIQUE).

C'est ce modèle qui a servi à générer les migrations Laravel via php artisan make:migration.



VI.2 Dictionnaire de données

Table	Champs principaux	Type / Contraintes	Rôle fonctionnel
users	id, name, email, password, role, token	PK, varchar, unique	Gère les utilisateurs et leurs rôles
civilizations	id, name, description, period	PK, varchar, text	Répertorie les civilisations historiques
artifact_sources	id, type, name, location, description	PK, varchar	Identifie la provenance des artefacts
artifacts	id, title, description, discovery_date, price, civilization_id, source_id	FK vers civilizations / artifact_sources	Représente les objets historiques
artifact_tags	id, name	PK, varchar, unique	Définit les catégories de classement
artifact_tag (pivot)	artifact_id, tag_id	FK, composite	Lie les artefacts à leurs tags
auctions	id, artifact_id, starting_price, end_time	FK vers artifacts	Prépare la logique d'enchères
bids	id, user_id, auction_id, bid_amount	FK vers users / auctions	Stocke les offres faites sur les enchères
cart_items	id, user_id, artifact_id, status	FK vers users / artifacts	Gère les paniers compétitifs et leur état
orders	id, user_id, total_amount, status	FK vers users	Enregistre les commandes validées
order_items	id, order_id, artifact_id, quantity, price	FK vers orders / artifacts	Détaille les artefacts commandés
transactions	id, order_id, payment_method, amount, status	FK vers orders	Suit les paiements et leur état
sessions	id, user_id, last_activity	varchar, timestamp	Gère les sessions actives
password_reset_tokens	email, token, created_at	varchar, timestamp	Gère la réinitialisation de mot de passe

Cette structure relationnelle a été pensée pour garantir : la cohérence des données (clés étrangères, intégrité référentielle), la performance (normalisation des tables, requêtes indexées), l'évolutivité : ajout possible de nouvelles entités (enchères, paiements, évaluations) sans refonte majeure.

La base de données d'Arte Facto forme ainsi un écosystème complet, capable de soutenir le front, les interactions utilisateurs et la future extension vers des fonctionnalités avancées comme les enchères ou les paiements sécurisés.

Chaque artefact peut être proposé selon deux modes de vente distincts : l'achat immédiat ou la vente aux enchères. Lorsqu'un objet est disponible en achat direct, il entre dans le circuit du panier compétitif, un système pensé pour simuler une concurrence silencieuse entre collectionneurs.

VI.3 Conception du panier compétitif

Le panier compétitif constitue la fonctionnalité emblématique d'Arte Facto. Conçu comme une expérience de tension douce, il introduit une dimension inédite dans la navigation : la rareté et le désir deviennent des éléments de l'interface.

Concrètement, lorsqu'un utilisateur ajoute un artefact à son panier, celui-ci change d'état et devient convoité pour les autres utilisateurs. Cette information est visible en temps réel dans la galerie : un message signale que d'autres utilisateurs ont déjà cet artefact dans leur panier.

Ce comportement, illustré sur la capture ci-dessous, participe à créer un climat d'émulation sans frustration, inspiré du principe de la vitrine muséale où chaque pièce attire les regards mais reste fragilement accessible.

The screenshot shows a product page for a gold funerary mask from the XXIst Dynasty. The main image is a close-up of the mask's face. To the right, the product title is displayed in bold: "MASQUE FUNÉRAIRE EN OR - XXIE DYNASTIE". Below the title, the price is listed as "125 000 €". The product is categorized under "Egypte Ancienne" and is described as a "Vente directe". A status message indicates that "5 utilisateurs ont cet artefact dans leur panier". A prominent yellow button labeled "AJOUTER AU PANIER" with a shopping cart icon is located at the bottom right. At the very bottom of the page, there is a "DESCRIPTION" section containing a detailed paragraph about the mask's history and characteristics.

Au-delà de son aspect ludique, le panier compétitif porte un enjeu d'ergonomie et de réalisme. Il rend visible l'activité du site, crée de la valeur autour de l'objet convoité et renforce le sentiment de collection partagée. Sa conception ouvre également la voie à une extension naturelle : celle du système d'enchères, pensé comme une évolution du même principe d'interaction autour de la rareté.

VI.4 Conception du système d'enchères (phase d'étude)

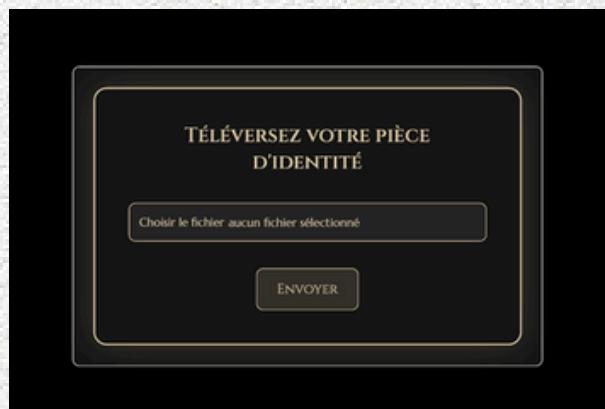
Les enchères constituent l'autre grand pilier du projet, bien que leur implémentation soit encore à l'état préparatoire. Elles prolongent le principe du panier compétitif en y ajoutant une dimension transactionnelle et temporelle, propre aux ventes d'art authentiques.

Chaque enchère est rattachée à un artefact et se déroule sur une période déterminée, avec un prix de départ, une réserve et un incrément minimal. Les utilisateurs pourront y participer en temps réel, plaçant des offres successives dont l'historique sera enregistré et horodaté.

Dès la conception de la base de données, deux tables principales ont été prévues pour cette fonctionnalité : auctions, qui définit les paramètres d'une vente (prix initial, réserve, dates de début et de fin, statut), et bids, qui enregistre chaque offre effectuée par un utilisateur, qu'elle soit manuelle ou automatique.

La conception intègre aussi un aspect sécuritaire et éthique : l'accès aux enchères n'est pas ouvert à tous les utilisateurs : pour y participer, il est nécessaire de téléverser une pièce d'identité. Ce document doit être validé par un administrateur avant de pouvoir participer. Ce contrôle préalable répond à une logique de confiance, essentielle dans tout système d'échange de biens culturels à forte valeur.

The screenshot shows a product page for a bronze statue of the Egyptian deity Anubis. The main image on the left is a dark, metallic statue of Anubis in a seated, recumbent pose, facing right. To the right of the image, the product title "STATUETTE D'ANUBIS" is displayed in bold capital letters. Below the title, the price "8 500 €" is shown in orange. Further down, categories "Égypte Ancienne" and "Enchère" are listed, along with a status indicator "Disponible" and a button "PARTICIPER À L'ENCHÈRE". At the bottom of the page, there is a link labeled "DESCRIPTION".



Bien que la fonctionnalité ne soit pas encore active dans la version actuelle du projet, une partie de sa mécanique a déjà été prototypée via les seeders. Ces derniers permettent de simuler des enchères en cours, terminées ou à venir, et d'y injecter des offres automatiques générées par différents utilisateurs.

Ce travail préparatoire constitue une base solide pour le développement ultérieur du module complet. Il illustre la démarche adoptée dans l'ensemble du projet : penser avant de coder, concevoir des structures durables et évolutives, capables d'accueillir de futures extensions sans refonte. Les enchères d'Arte Facto seront ainsi, à terme, le prolongement naturel du panier compétitif : une expérience immersive et crédible, où la valeur des artefacts se négocie dans le temps autant que dans le regard.

VII. Implémentation du Front-End

Le front-end d'Arte Facto repose sur une combinaison d'outils puissants et complémentaires, pensée pour offrir une expérience fluide, réactive et élégante.

Chaque couche technologique y trouve sa place avec précision : Blade structure le rendu serveur, Livewire assure la réactivité sans JavaScript manuel, Flux fournit une base d'interface accessible et modulaire, Tailwind CSS gère la cohérence visuelle et la responsivité, tandis que Volt, surplombant l'ensemble, orchestre la communication entre les composants avec une logique moderne et performante.

VII.1 Architecture et logique d'affichage

Le rendu des pages s'appuie sur un layout Blade global, au sein duquel le composant `<flux:header>` structure la navigation principale. Les éléments dynamiques : menu, panier, favoris, profil, utilisent la syntaxe déclarative de Flux, qui assure un rendu fluide et accessible sans surcharge de code.

Les directives `wire:navigate` permettent à Livewire de recharger uniquement les sections concernées lors d'une interaction, supprimant les rechargements de page complets. Ainsi, lorsqu'un utilisateur ajoute un artefact à son panier, le composant `cart-indicator` actualise en temps réel le badge d'icône grâce à une communication réactive entre front et back.

```
resources > views > livewire > 🛒 cart-indicator.blade.php
1   <div>
2     <flux:navbar.item class="relative" icon="shopping-cart" :href="route('cart.index')" wire:navigate>
3       @if($count > 0)
4         <flux:badge size="xs" variant="primary" class="absolute -top-1 -right-1">
5           | {{ $count }}
6           | </flux:badge>
7         @endif
8       </flux:navbar.item>
9     </div>
```

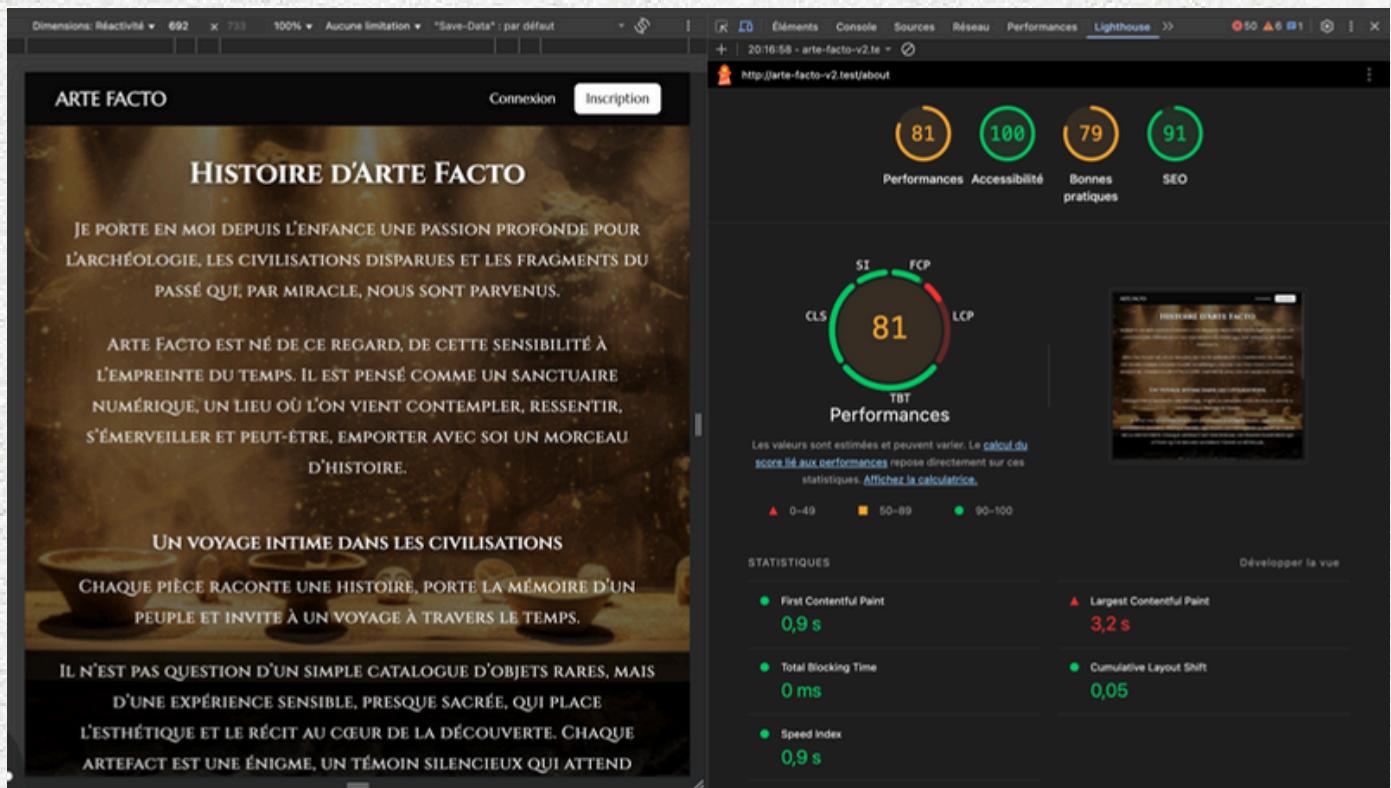
Ce mécanisme conserve la simplicité du rendu côté serveur tout en offrant une interactivité proche d'une application monopage. L'organisation du code reflète cette philosophie de clarté : chaque composant gère un rôle précis, les vues restent épurées, et la logique métier se concentre côté contrôleur. Le résultat est un affichage homogène, maintenable et parfaitement adapté à une montée en complexité future.

VII.2 Esthétique, accessibilité et performances

L'attention portée à l'accessibilité et à la sémantique se traduit directement dans la structure du code. La page "À propos" repose sur une hiérarchie claire : l'élément `<main role="main">` définit la zone principale de contenu, tandis que l'attribut `aria-label="Page À propos du site Arte Facto"` en fournit une description explicite pour les technologies d'assistance.

```
resources > views > about.blade.php
1  <x-layouts.app.header :title="'À propos'">
2
3  <main role="main" aria-label="Page À propos du site Arte Facto">
4      <section
5          class="relative p-8 text-center bg-cover bg-center bg-fixed min-h-screen flex flex-col justify-center items-center"
6          style="background-image: url('/images/fond.png'); font-family: 'Cinzel', serif;">
7
8      >
9
10     <div
11         class="absolute inset-0 bg-[rgba(0,0,0,0.55)] backdrop-blur-[1px] pointer-events-none"
12         aria-hidden="true">
13     </div>
14
15     <!-- Contenu principal -->
16     <div class="relative z-10 max-w-5xl text-zinc-100">
17
18         <!-- Titre -->
19         <h1 class="text-3xl md:text-4xl font-bold mb-6 text-white drop-shadow-[0_2px_4px_rgba(0,0,0,0.7)]">
20             | Histoire d'Arte Facto
21             </h1>
22
23         <!-- Intro-->
24         <p class="text-[1.1rem] leading-8 font-medium text-zinc-100/95 tracking-wide">
25             | Je porte en moi depuis l'enfance une passion profonde pour l'archéologie, les civilisations disparues et les
26             | qui, par miracle, nous sont parvenues.
27             </p>
```

Chaque sous-partie est encadrée par des titres hiérarchisés (`<h1>`, `<h2>`) qui guident la lecture vocale et structurent logiquement le texte. Les éléments purement décoratifs, comme le calque visuel d'arrière-plan, sont marqués avec `aria-hidden="true"` et `pointer-events-none` afin d'être ignorés par les lecteurs d'écran et ne pas interférer avec la navigation clavier. L'usage de `role`, `aria-label` et `aria-hidden` garantit que l'expérience reste aussi fluide pour un utilisateur voyant que pour un utilisateur de lecteur d'écran, conformément aux recommandations WCAG. Cette rigueur sémantique complète l'approche esthétique : le fond fixe, la typographie Cinzel et la palette sombre participent à l'immersion sans jamais nuire à la lisibilité.



Un audit Lighthouse a été réalisé pour évaluer la qualité du front-end d'Arte Facto selon quatre axes : performances, accessibilité, bonnes pratiques et SEO.

Le site obtient des résultats remarquablement équilibrés :

- Performances : 81/100, attestant d'un rendu fluide et d'un chargement rapide, malgré des visuels riches et un fond animé.
- Accessibilité : 100/100, preuve d'un soin particulier apporté à la structure sémantique, aux contrastes et à la compatibilité clavier.
- Bonnes pratiques : 79/100, reflétant une base saine et optimisable, notamment sur la compression des images de fond.
- SEO : 91/100, garantissant une indexation claire et un respect des standards de référencement.

Ces scores témoignent d'un équilibre entre esthétique immersive et rigueur technique, démontrant qu'un univers visuel fort peut coexister avec un code léger, accessible et optimisé.

VII.3 Sécurité et robustesse du front

La sécurité côté front a été traitée avec la même exigence que la couche back-end. Les routes utilisent les helpers Laravel (route(...)), empêchant toute injection d'URL.

```
<flux:navlist.item :href="route('cgu')" :current="request()>routeIs('about')" wire:navigate>
|   CGU
</flux:navlist.item>

<flux:navlist.item :href="route('cgv')" :current="request()>routeIs('about')" wire:navigate>
|   CGV
</flux:navlist.item>

<flux:navbar.item :href="route('about')" :current="request()>routeIs('about')" wire:navigate>
|   À propos
</flux:navbar.item>
```

Les formulaires sont protégés par des tokens @csrf, et l'échappement automatique des variables Blade neutralise les risques d'injection de script ex : {{ \$artifact->title }}.

```
<form method="POST" action="{{ route('logout') }}" class="w-full">
@csrf
<flux:menu.item as="button" type="submit" icon="arrow-right-start-on-rectangle" class="w-full">
|   Se déconnecter
</flux:menu.item>
</form>
```

```
-- Informations et description --
<div class="bg-zinc-950 border border-amber-900/30 p-8">
<h1 class="text-3xl font-bold text-white mb-6" style="font-family: 'Cinzel', serif;">
|   {{ $artifact->title }}
</h1>
```

Le contrôle conditionnel renforce la sécurité de l'affichage côté front.

L'instruction `@if(auth()->check() && auth()->user()->isAdmin())` vérifie à la fois que l'utilisateur est authentifié(`auth()->check()`) et qu'il possède le rôle administrateur (`auth()->user()->isAdmin()`).

Selon le résultat, le rendu bascule dynamiquement entre deux layouts distincts : `header_admin` pour les administrateurs et `header` pour les utilisateurs classiques.

Ainsi, aucune section du front réservée à l'administration n'est accessible sans validation préalable du rôle, garantissant une isolation stricte entre les espaces publics et les interfaces de gestion.

```
resources > views > components > layouts > 🗂 app.blade.php
1  @if(auth()->check() && auth()->user()->isAdmin())
2    <x-layouts.app.header_admin :title="$title ?? null">
3      {{ $slot }}
4    </x-layouts.app.header_admin>
5  @else
6    <x-layouts.app.header :title="$title ?? null">
7      {{ $slot }}
8    </x-layouts.app.header>
9  @endif
```

VIII. Implémentation du Back-End

Le back-end d'Arte Facto constitue la charpente logique du projet. C'est lui qui donne vie aux règles métier, assure la cohérence des interactions et veille à l'intégrité des données. L'enjeu n'était pas seulement de créer une structure fonctionnelle, mais de concevoir une mécanique fidèle à l'esprit du marché de l'art : rareté, authenticité et tension entre collectionneurs.

VIII.1 Architecture et logique métier

La couche back-end repose sur le framework Laravel 12, s'appuyant sur le moteur Eloquent ORM pour modéliser les entités du domaine : User, Artifact, Cart, Order, Auction et Bid.

Chaque modèle traduit une réalité précise du système, reliée aux autres par des relations Eloquent (hasMany, belongsToMany, morphOne) permettant de manipuler les données de manière fluide et cohérente.

```
app > Models > Artifact.php
13  {
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54     public function civilization(): BelongsTo
55     {
56         return $this->belongsTo(Civilization::class);
57     }
58
59     public function source(): BelongsTo
60     {
61         return $this->belongsTo(ArtifactSource::class, 'source_id');
62     }
63
64     public function tags(): BelongsToMany
65     {
66         return $this->belongsToMany(ArtifactTag::class, 'artifact_tag', 'artifact_id', 'tag_id');
67     }
68
69     public function auction(): HasOne
70     {
71         return $this->hasOne(Auction::class);
72     }
73
74     public function cartItems(): HasMany
75     {
76         return $this->hasMany(CartItem::class);
77     }
78
79
```

Le composant App\Livewire\Artifacts>Show illustre la logique métier au cœur du back-end.

Lors de l'initialisation (mount()), il charge l'artefact courant avec ses relations Eloquent (civilization, source, tags, usersInCart) afin de restituer un contexte complet de la pièce.

Cette approche démontre la puissance du pattern MVC appliqué à Livewire : le modèle centralise les données et leurs liens, tandis que le composant agit comme un contrôleur local orchestrant la cohérence métier.

La méthode checkCartStates() est ensuite appelée pour déterminer en temps réel les états de disponibilité et d'appartenance, garantissant une expérience fluide et synchronisée entre les utilisateurs.

Enfin, la recherche de trois artefacts apparentés par civilisation illustre une logique métier fine : les relations Eloquent deviennent un véritable langage du domaine.

```
public function mount($id)
{
    $this->artifact = Artifact::with(['civilization', 'source', 'tags', 'usersInCart'])
        ->findOrFail($id);

    $this->checkCartStates();

    $this->relatedArtifacts = Artifact::where('civilization_id', $this->artifact->civilization_id)
        ->where('id', '!=', $this->artifact->id)
        ->where('status', 'available')
        ->take(3)
        ->get();
}
```

VIII.2 Le panier compétitif : une mécanique de rareté

Le panier compétitif est la fonctionnalité emblématique du projet. Inspiré du comportement des galeries d'art, il reproduit la tension naturelle entre collectionneurs convoitant la même pièce rare.

Lorsqu'un utilisateur ajoute un artefact à son panier, celui-ci change immédiatement d'état en base et devient "convoité" pour les autres.

Le composant Livewire App\Livewire\Artifacts>Show assure la coordination de cette logique. Lorsqu'une œuvre est consultée, il interroge la base via la méthode checkCartStates() pour déterminer si l'artefact est déjà présent dans un ou plusieurs paniers via :

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

```
private function checkCartStates()
{
    if (Auth::check()) {
        $this->inCart = $this->artifact->usersInCart->contains(Auth::id());
        $this->otherCartsCount = $this->artifact->usersInCart->where('id', '!=', Auth::id())->count();
    } else {
        $this->otherCartsCount = $this->artifact->usersInCart->count();
    }
}
```

Les données renvoyées influencent directement l'affichage : une mention telle que “✓ Dans votre panier” ou “2 utilisateurs ont cet artefact dans leur panier” s'affiche en temps réel, créant une expérience à la fois interactive et émotionnelle.

L'ajout d'un artefact au panier déclenche un événement Livewire (cartUpdated), qui met à jour la barre de navigation et le compteur global sans recharge de page. Ce comportement, soutenu par Eloquent, assure la cohérence transactionnelle : un artefact ne peut être ajouté qu'une fois par utilisateur, et les états du panier sont automatiquement synchronisés entre tous les clients actifs.

```
$this->dispatch('cartUpdated');
```

La validation d'achat s'effectue via une transaction DB::transaction() combinée à lockForUpdate(), garantissant qu'un artefact ne puisse être réservé simultanément par plusieurs personnes : une approche indispensable pour simuler une dynamique de collection en temps réel.

```
try {
    DB::transaction(function () use ($order) {
        $remainingItems = [];

        // 1) Filtrer les items encore achetables
        foreach ($order->items as $item) {
            if (!$item->artifact_id) {
                // item sans artifact (sécurité) -> skip
                continue;
            }

            $a = Artifact::where('id', $item->artifact_id)->lockForUpdate()->first();
```

VIII.3 Vérification d'identité pour l'accès aux enchères

Les enchères, bien que toujours en phase préparatoire, introduisent une contrainte majeure : l'accès à ce mode de vente est conditionné à une vérification d'identité préalable. Cette exigence répond à une logique juridique et patrimoniale : seules les personnes identifiées peuvent participer à une vente d'objets de grande valeur. Sur le plan technique, cette vérification repose sur la propriété `identity_verified` du modèle User.

Lorsqu'un utilisateur tente de participer à une enchère, le système vérifie ce champ : si la vérification est valide, il accède au processus d'enchère, sinon, une modale Livewire s'ouvre, invitant au téléversement d'une pièce d'identité. Le document est alors envoyé via un formulaire sécurisé, stocké sur le serveur, puis validé manuellement par un administrateur.

```
public function participateAuction()
{
    if (!Auth::check()) {
        return redirect()->route('login');
    }

    if (Auth::user()->identity_verified !== 'verified') {
        $this->showIdentityModal = true;
        return;
    }

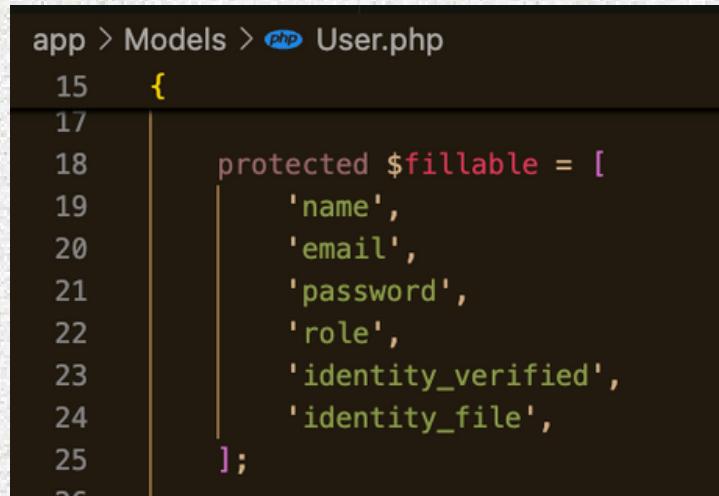
    return redirect()->route('auction.show', $this->artifact->id);
}
```

VIII.4 Sécurité et intégrité des données

La sécurité et l'intégrité des données s'appuient d'abord sur les mécanismes natifs d'Eloquent : chaque modèle est soigneusement configuré pour limiter les risques de corruption ou d'exploitation malveillante des données. Les formulaires sont protégés par des Form Requests dédiés, et la validation des champs se fait systématiquement avant toute persistance.

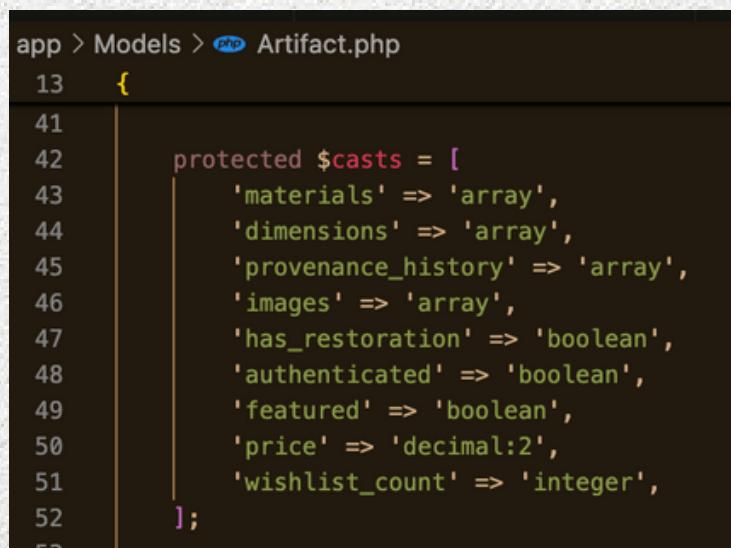
Les propriétés `$fillable` sont utilisées pour contrôler explicitement les champs qui peuvent être remplis en masse. Cette approche protège l'application contre les attaques de mass assignment, où un utilisateur tenterait d'injecter des champs non prévus (comme un rôle administrateur, un statut de vérification ou un montant de transaction) via un formulaire ou une requête forgée.

En ne déclarant que les attributs strictement nécessaires dans `$fillable`, les modèles User, Artifact, Order, Transaction ou CartItem deviennent des filtres qui rejettent silencieusement tout champ non autorisé.



```
app > Models > User.php
15  {
17
18      protected $fillable = [
19          'name',
20          'email',
21          'password',
22          'role',
23          'identity_verified',
24          'identity_file',
25      ];
26
```

La propriété `$casts` joue également un rôle important dans la robustesse du système. Elle permet de forcer le typage d'attributs sensibles : par exemple, `identity_verified` est casté en booléen, les montants en decimal.. Cette conversion systématique garantit que les valeurs stockées et manipulées en mémoire sont cohérentes avec ce que le domaine métier attend, limitant les comportements imprévus et les interprétations ambiguës des données.



```
app > Models > Artifact.php
13  {
14
15      protected $casts = [
16          'materials' => 'array',
17          'dimensions' => 'array',
18          'provenance_history' => 'array',
19          'images' => 'array',
20          'has_restoration' => 'boolean',
21          'authenticated' => 'boolean',
22          'featured' => 'boolean',
23          'price' => 'decimal:2',
24          'wishlist_count' => 'integer',
25      ];
26
```

Sur le plan des échanges avec la base, Eloquent et le Query Builder de Laravel reposent sur les requêtes préparées PDO et l'injection de paramètres bindés : concrètement, les valeurs issues des formulaires utilisateurs ne sont jamais concaténées directement dans des chaînes SQL, mais transmises comme paramètres liés (bindings), ce qui neutralise les tentatives d'injection SQL.

L'utilisation exclusive des méthodes `where()`, `first()`, `create()`, `update()` ou `delete()` plutôt que de requêtes brutes permet de bénéficier pleinement de cette couche de protection. Les opérations critiques, comme la validation de commande ou la création de transaction, sont encapsulées dans des blocs `DB::transaction()`. En cas d'erreur, une exception force le rollback de l'ensemble des écritures, préservant l'intégrité des données.

```
app > Http > Controllers > OrdersController.php
13  {
14
15      /**
16       * POST /orders/{order}/pay
17       */
18      public function pay(Order $order): RedirectResponse
19      {
20          abort_if($order->user_id !== Auth::id(), 403);
21
22          if (!in_array($order->status, ['draft', 'pending_payment'])) {
23              return back()->withErrors(['order' => "Cette commande n'est pas payable (statut: {$order->status})."]);
24          }
25
26          $order->load('items');
27
28          try {
29              DB::transaction(function () use ($order) {
30                  $remainingItems = [];
31
32                  // 1) Filtrer les items encore achetables
33                  foreach ($order->items as $item) {
34                      if (!$item->artifact_id) {
35                          // item sans artifact (sécurité) -> skip
36                          continue;
37
38                      }
39
40                      $a = Artifact::where('id', $item->artifact_id)->lockForUpdate()->first();
41
42                      if ($a) {
43                          $remainingItems[] = $item;
44
45                      }
46
47                  }
48
49                  $order->items = $remainingItems;
50
51                  $order->status = 'paid';
52
53                  $order->save();
54
55              });
56          } catch (\Exception $e) {
57              // ...
58          }
59
60          return redirect('/orders');
61      }
62
63  }
```

Cette approche, combinée à l'usage des clés étrangères et des contraintes de migration, garantit qu'aucune commande ne puisse être validée sans artefact associé, ni aucune enchère exister sans lien avec un utilisateur et un objet clairement identifié.

L'ensemble de ces mécanismes : typage via `$casts`, contrôle des champs via `$fillable`, requêtes préparées PDO et transactions atomiques constitue une ligne de défense cohérente, intégrée au cœur même du back-end.

IX. Sécurité de l'application

La sécurité d'Arte Facto repose sur plusieurs couches cohérentes et interdépendantes, alliant robustesse du framework Laravel, contrôle des accès par rôles, vérification d'identité et validation des utilisateurs par e-mail. L'objectif n'a jamais été de complexifier le système, mais de garantir la confiance, la confidentialité et l'intégrité des échanges entre les utilisateurs et la plateforme.

IX.1 Système d'authentification

Le site repose sur le système d'authentification natif de Laravel, basé sur les sessions et cookies sécurisés. Cette approche, éprouvée et performante, permet d'assurer une gestion fiable des connexions et des rôles utilisateurs.

Lorsqu'un utilisateur se connecte, Laravel vérifie les identifiants via le modèle User et la méthode Auth::attempt(). Si l'authentification réussit, une session unique est créée et stockée dans la table sessions. Le navigateur reçoit alors un cookie nommé arte_facto_session, contenant uniquement l'identifiant de cette session. À chaque requête suivante, Laravel recharge automatiquement l'objet Auth::user() à partir de cet identifiant, garantissant la continuité et la sécurité de la navigation.

Le cookie remember_web_..., facultatif, gère l'option “Se souvenir de moi” en associant un token chiffré au champ remember_token du modèle User. Les cookies ainsi générés sont marqués comme HttpOnly et SameSite=Lax, ce qui empêche leur lecture par JavaScript et limite les fuites interdomaines. Les sessions sont stockées exclusivement côté serveur et ne transitent jamais dans le navigateur.

Chaque requête sensible (POST, PUT, DELETE) est protégée par un token CSRF (XSRF-TOKEN), intégré automatiquement aux formulaires par Laravel, empêchant ainsi les attaques de type Cross-Site Request Forgery. Les mots de passe, quant à eux, sont hachés via l'algorithme bcrypt, rendant impossible toute récupération en clair même en cas de compromission de la base de données.

La capture d'écran issue de l'inspecteur réseau illustre ce fonctionnement : la présence du cookie `arte_facto_session`, la génération du token CSRF et la réponse sécurisée du serveur avec le cookie `remember_web`...uniquement accessible côté HTTP.

Nom	Valeur
arte_facto_session	eyJpdiI6IjM5dWnrdTFQcIWpocThVbIAvUUJ6dUE9PSisInZhbHVljoiaGx...
XSRF-TOKEN	eyJpdiI6ImJQWVmT0hOTndEb0xwZndalWmZqYmcBPSisInZhbHVljoiaGx...

Nom	Valeur	D...	P...	E...	M...	S...	Htt...	Sa...
arte_facto_session	eyJpdiI6IjWfH2cyL...	-	/	0...	7...	✓	Lax	
remember_web	59ba36addc2b2f...	-	/	1...	3...	✓	Lax	
XSRF-TOKEN	eyJpdiI6IjR7ZY...	-	/	0...	7...		Lax	

IX.2 Gestion des sessions, rôles et middlewares

Au-delà de la simple authentification, la sécurité d'Arte Facto s'étend à une gestion rigoureuse des rôles et des autorisations via les middlewares. Chaque route sensible du site est protégée par un ensemble de filtres déclarés dans le fichier `bootstrap/app.php`. Le middleware personnalisé `IsAdmin` assure la séparation entre les utilisateurs standards et les administrateurs.

Sa logique est la suivante :

```
app > Http > Middleware > IsAdmin.php
1  <?php
2
3  namespace App\Http\Middleware;
4
5  use Closure;
6  use Illuminate\Http\Request;
7
8  class IsAdmin
9  {
10      public function handle(Request $request, Closure $next)
11      {
12          if (! auth()->check() || ! auth()->user()->isAdmin()) {
13              return redirect()->route('home');
14          }
15
16          return $next($request);
17      }
18  }
```

Ce middleware vérifie d'abord que l'utilisateur est bien authentifié, puis appelle la méthode `isAdmin()` du modèle `User`. Si l'utilisateur ne dispose pas du rôle administrateur, il est automatiquement redirigé vers la page d'accueil. Cette vérification empêche tout accès non autorisé à l'espace d'administration et renforce la hiérarchie des permissions.

Dans le fichier routes/web.php, les routes publiques telles que l'accueil ou la page des artefacts restent accessibles à tous, tandis que les routes encapsulées dans le middleware auth exigent une session active. Les routes plus sensibles, notamment celles regroupées sous le préfixe admin, cumulent les protections auth et admin. Cette organisation logique permet un cloisonnement clair et évolutif du site, où chaque utilisateur n'a accès qu'aux fonctionnalités correspondant à son rôle.

Ce découpage, à la fois simple et structuré, garantit qu'aucune donnée critique ne soit exposée et qu'aucune action administrative ne puisse être exécutée sans autorisation préalable.

Vocii un extrait de mon fichier web.php :

IX.3 Vérification des adresses e-mail (Mailtrap)

Afin d'assurer la validité des inscriptions et d'éviter la création de comptes automatisés, Arte Facto intègre un système complet de vérification d'adresses e-mail. Après l'inscription, Laravel génère un lien de confirmation signé et temporaire envoyé par e-mail grâce au service Mailtrap. Ce dernier simule un serveur SMTP sécurisé, idéal pour les environnements de développement, sans risquer d'envoyer de messages réels à des tiers.

La configuration de Mailtrap s'effectue directement dans le fichier .env du projet en renseignant les paramètres fournis par la plateforme : hôte SMTP, port, identifiants et protocole TLS. Lors de la réception du mail, l'utilisateur clique sur le lien de validation, ce qui met à jour automatiquement la colonne email_verified_at du modèle User. Tant que cette vérification n'est pas effectuée, l'accès aux sections protégées de la plateforme, notamment le panier et les fonctionnalités transactionnelles, reste bloqué. Le middleware verified contrôle cette condition à chaque requête et redirige systématiquement les comptes non confirmés.

```
MAIL_MAILER=smtp
MAIL_HOST=sandbox.smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=d054f4b371a8bc
MAIL_PASSWORD=60556d2fa1f5cb
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=artefacto@example.com
MAIL_FROM_NAME="Arte-Facto"
```

The screenshot shows the Mailtrap web interface. On the left, a sidebar navigation includes Home, API/SMTP, Sandbox (selected), Inboxes, For marketers, General, and MT MCP Server. The main area displays an inbox with several verification emails from 'Arte-Facto <artefacto@example.com>' to various addresses like 'fred@gmail.com', 'hay@gmail.com', etc., sent 3 minutes ago, 3 hours ago, and 3 days ago. To the right, a detailed view of a specific email titled 'Verify Email Address' is shown. The email header shows 'From: Arte-Facto <artefacto@example.com>', 'To: <fred@gmail.com>', and a timestamp of '2025-06-10 11:57, 13 KB'. Below the header, tabs for HTML, HTML Source, Text, Raw, Spam Analysis, HTML Check, and Tech Info are visible. The main content of the email is a template with a 'Verify Email Address' button. At the bottom, there is a note about trouble clicking the button and a URL: <http://arte-facto.test/verify-email/6/b6ff9206b96725fc4f23772233ba8f3c7c3428?Expires=174956025&Signature=co>.

X. Tests et jeu d'essai

Afin de garantir la fiabilité et la sécurité du système de vérification d'identité, plusieurs niveaux de tests ont été réalisés sur l'application Arte Facto. Les vérifications se sont articulées autour de tests manuels via Tinker, de tests automatisés Pest intégrés à Laravel, puis d'un jeu d'essai complet simulant une situation réelle d'utilisateur et d'administrateur.

X.1 Vérifications manuelles via Tinker

Les premiers contrôles ont été effectués avec Tinker, l'interface interactive de Laravel, permettant d'interagir directement avec la base de données et les modèles Eloquent.

Cette méthode a servi à confirmer le bon fonctionnement des mises à jour automatiques sur les champs identity_file et identity_verified, ainsi que la cohérence des rôles (user ou admin).

Depuis le terminal, l'environnement a été ouvert avec la commande : php artisan tinker.

Quelques exemples de commandes exécutées :

```
[archimede@MacBookAir ~/Herd/arte-facto-v2 % php artisan tinker
Psy Shell v0.12.9 (PHP 8.4.11 - cli) by Justin Hileman
> App\Models\User::where('email', 'jack@example.com')->first()->identity_verified;
= "verified"
```

```
[> App\Models\User::where('role', 'admin')->pluck('email');
= Illuminate\Support\Collection {#6598
  all: [
    "indy@example.com",
  ],
}
```

Ces manipulations directes constituent une première assurance de la cohérence du système, avant toute automatisation.

X.2 Tests automatisés via Pest

En complément des tests manuels, l'ensemble des tests automatisés Pest fournis par Laravel a été exécuté afin de vérifier la stabilité de l'environnement applicatif.

Ces tests concernent principalement les fonctionnalités d'authentification, de réinitialisation de mot de passe et de vérification d'e-mail.

L'exécution s'effectue via la commande suivante : ./vendor/bin/pest

```
archimede@MacBookAir ~$/herd/arte-facto-v2 % ./vendor/bin/pest
PASS Tests\Unit\ExampleTest
✓ that true is true                                         0.01s

PASS Tests\Feature\Auth\AuthenticationTest
✓ login screen can be rendered                            0.37s
✓ users can authenticate using the login screen          0.55s
✓ users can not authenticate with invalid password      0.26s
✓ users can logout                                       0.01s

PASS Tests\Feature\Auth\EmailVerificationTest
✓ email verification screen can be rendered              0.02s
✓ email can be verified                                  0.02s
✓ email is not verified with invalid hash                0.01s

PASS Tests\Feature\Auth>PasswordConfirmationTest
✓ confirm password screen can be rendered                0.02s
✓ password can be confirmed                            0.02s
✓ password is not confirmed with invalid password      0.24s

PASS Tests\Feature\Auth>PasswordResetTest
✓ reset password link screen can be rendered            0.02s
✓ reset password link can be requested                  0.25s
✓ reset password screen can be rendered                0.27s
✓ password can be reset with valid token                0.30s
```

Le résultat visible dans la console indique que l'intégralité des tests par défaut sont passés avec succès.

Chaque test affiche un statut PASS en vert, indiquant que toutes les fonctionnalités d'authentification et de sécurité de base sont opérationnelles.

Même sans tests personnalisés supplémentaires, ce passage complet valide la stabilité du socle Laravel et confirme qu'aucune régression n'a été introduite dans les fonctionnalités principales d'accès et de sessions.

X.3 Jeu d'essai

Afin d'illustrer concrètement le fonctionnement du système, un jeu d'essai complet a été mis en place autour de deux profils :

- Indiana Jones : administrateur (indy@example.com)
- Jack Sparrow : utilisateur membre (jack@example.com)

Le scénario suivant reproduit le processus complet de vérification d'identité :

1. Connexion de l'utilisateur

Jack Sparrow se connecte à son compte membre. Son statut d'identité est pending et aucun fichier n'est encore associé à son profil.

#	name	email	email_verified_at	password	role	identity_file
1	Indiana Jones	indy@example.com	2025-10-15 07:51:26	\$2y\$12\$cLV4ZUFFpShl/DeRECFS.BRuAP6nHh/Z43...	admin	verified
2	Lara Croft	lara@example.com	2025-10-15 07:51:27	\$2y\$12\$bBNYynFeQflfWXvhC9sBzJ.F/B06.Wa1AwpE...	user	verified
3	Benjamin Gates	gates@example.com	2025-10-15 07:51:27	\$2y\$12\$57jgmpNbhSxFFWyqJo1FuCo/vf5oQK7pd6...	user	pending
4	Jack Sparrow	jack@example.com	2025-10-15 07:51:28	\$2y\$12\$AI70BLU.slkIScxzRYz.utnqfigaxVigNtVNs2sg...	user	verified
5	Bilbo Baggins	bilbo@example.com	2025-10-15 07:51:28	\$2y\$12\$4WlRTDak7OMC8GT04v46AC/eHy5j/qGJ3K...	user	verified
6	Allan Quatermain	allan@example.com	2025-10-15 07:51:28	\$2y\$12\$frMsuxxDQH3EY9arr2WKgj8OTaU12WkR8fW...	user	verified
7	Rick O'Connell	rick@example.com	2025-10-15 07:51:29	\$2y\$12\$Us4Qk9tnMj5G5frFs9XuMs1kk5yFtdx9CnL...	user	pending
8	Sarah Connor	connor@example.com	2025-10-15 07:51:29	\$2y\$12\$2Enlgg4QLox5WWlb.HUwsOhRIMcZXADsa...	user	pending

2. Tentative de participation à une enchère

En cliquant sur « Participer à l'enchère », l'application détecte que son identité n'est pas encore vérifiée et affiche une modale Livewire d'envoi de pièce d'identité.

The screenshot shows a product listing for a "STATUETTE D'ANUBIS". The price is 8 500 €. The item is categorized as "Egypte Ancienne" and its type is "Enchère". The status is "Disponible". A button labeled "PARTICIPER À L'ENCHÈRE" is visible. Below the listing, there is a "DESCRIPTION" section with a detailed text about the statue.

The screenshot shows a modal window titled "TÉLÉVERSEZ VOTRE PIÈCE D'IDENTITÉ". It contains a button "Choisir le fichier" with the message "aucun fichier sélectionné" and an "ENVOYER" button.

3. Téléversement du document

Jack téléverse une image de sa carte. Le fichier est validé (taille, format, sécurité) puis enregistré dans le répertoire privé storage/app/private/identities. Son profil est mis à jour : identity_file → nom du fichier généré automatiquement.

4. Vérification par l'administrateur

Indiana Jones, dans le tableau de bord d'administration, ouvre la section Vérification des identités.

VÉRIFICATION DES IDENTITÉS				
NOM	EMAIL	FICHIER	STATUT	ACTIONS
Lara Croft	lara@example.com		VÉRIFIÉ	
Jack Sparrow	jack@example.com		EN ATTENTE ✎	<button>VÉRIFIER</button> <button>REJETER</button>
Rick O'Connell	rick@example.com		EN ATTENTE ✎	<button>VÉRIFIER</button> <button>REJETER</button>
Sarah Connor	connor@example.com		EN ATTENTE ✎	<button>VÉRIFIER</button> <button>REJETER</button>

En cliquant sur « Vérifier », le champ identity_verified passe de pending à verified, tandis qu'un message de confirmation s'affiche.

VÉRIFICATION DES IDENTITÉS				
NOM	EMAIL	FICHIER	STATUT	ACTIONS
Lara Croft	lara@example.com		VÉRIFIÉ	
Jack Sparrow	jack@example.com		VÉRIFIÉ	
Rick O'Connell	rick@example.com		EN ATTENTE ✎	<button>VÉRIFIER</button> <button>REJETER</button>
Sarah Connor	connor@example.com		EN ATTENTE ✎	<button>VÉRIFIER</button> <button>REJETER</button>

5. Accès autorisé

Dès cette validation, Jack peut accéder à la salle des ventes et participer aux enchères (qui je le rappelle ne sont pas encore misent en place)

The screenshot shows a bronze Anubis statuette from ancient Egypt. The interface includes a header with navigation links: Accueil, Catalogue, Enchères, CGU, CGV, and À propos. A shopping cart icon with '1' is also present. The main content area displays the item's name, current price (8500 €), and the time until bidding ends (02:14:35). It also shows a history of bids and a field for placing a bid.

Bidder	Price (€)
Lara Croft	8500 €
Benjamin Gates	8200 €
Jack Sparrow	7800 €

DESCRIPTION
Statuette en bronze du dieu Anubis, maître des nécropoles égyptiennes, gardien de la pesée des âmes. Chef-d'œuvre du Nouvel Empire conservant toute sa majesté funéraire.

6. Suivi dans la base

Dans SQLite Viewer, les colonnes identity_file et identity_verified témoignent du chemin complet de la pièce d'identité et de son statut, assurant la traçabilité du processus.

ID	Name	Email	Email Verified At	Password	Role	Identity Verified	Identity File
1	Indiana Jones	indy@example.com	2025-10-15 07:51:26	\$2y\$12\$cLV4ZUFFpShlJDeRE	admin	verified	(NULL)
2	Lara Croft	lara@example.com	2025-10-15 07:51:27	\$2y\$12\$b8NyfnFeQftWXvhC	user	verified	le4p8WNfq5S42Ahx05zpU-
3	Benjamin Gates	gates@example.com	2025-10-15 07:51:27	\$2y\$12\$57ljgmpNbhsxFWYq	user	pending	(NULL)
4	Jack Sparrow	jack@example.com	2025-10-15 07:51:28	\$2y\$12\$AI7GBLj.slkisCxzRYz	user	verified	H7u5TVcU5qrkXLfQwNiaUp-
5	Bilbo Baggins	bilbo@example.com	2025-10-15 07:51:28	\$2y\$12\$4WitTDak70MO8GTI	user	verified	(NULL)
6	Allan Quatermain	allan@example.com	2025-10-15 07:51:28	\$2y\$12\$rMsuXDQH3EY9arr2'	user	verified	(NULL)
7	Rick O'Connell	rick@example.com	2025-10-15 07:51:29	\$2y\$12\$Us4Qk9tnRnj5G5frF	user	pending	2vu2HAoFzaPhk6hESTkb3
8	Sarah Connor	connor@example.com	2025-10-15 07:51:29	\$2y\$12\$2Enlgg4QLQx5WWlb	user	pending	U7SOQQnRJMIBURi65xtP-

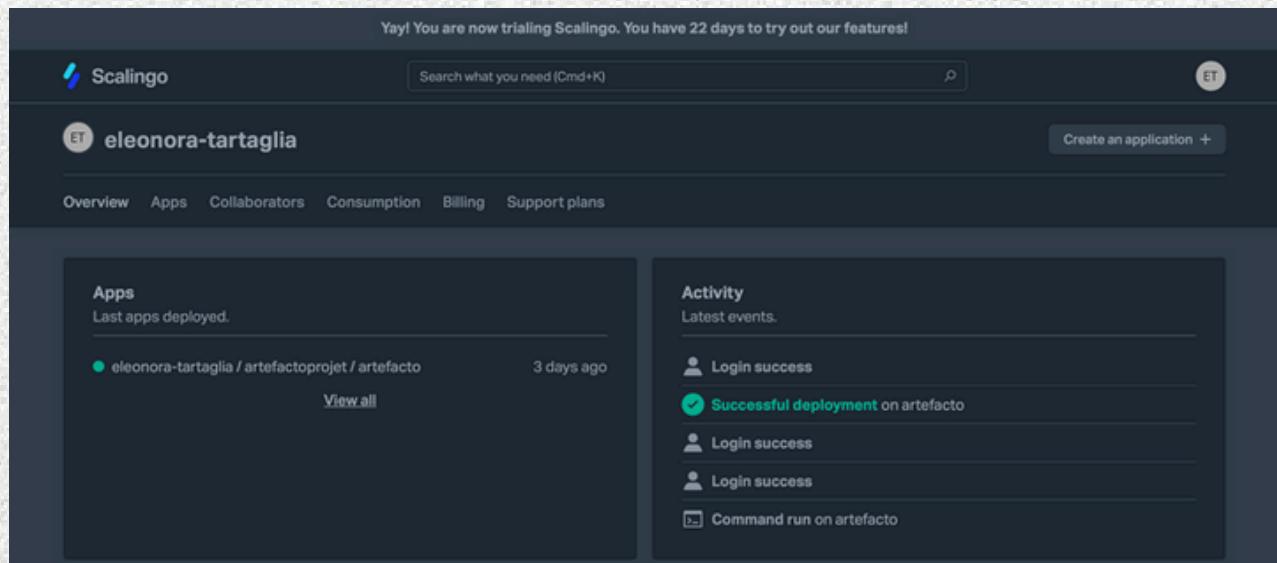
XI. Déploiement de l'application

Le déploiement de l'application Arte Facto a été réalisé sur la plateforme française Scalingo, choisie pour sa simplicité d'intégration avec Laravel et pour la fiabilité de son infrastructure cloud. L'environnement de production repose sur Laravel 12 et PHP 8.4, garantissant une exécution moderne, stable et sécurisée de l'application.

XI.1 Préparation de l'environnement de production

La première étape a consisté à préparer l'environnement de déploiement sur le poste local : L'outil en ligne de commande Scalingo CLI a été installé via Homebrew, puis relié au compte développeur grâce à la génération d'un token d'accès API et une authentification directe depuis le terminal.

Une fois connecté, le projet distant Arte Facto a été créé sur Scalingo et associé au dépôt GitHub du projet local. Cette liaison permet de bénéficier d'un déploiement continu automatisé : chaque git push sur la branche main déclenche automatiquement une nouvelle mise à jour sur l'application hébergée. Ce mécanisme assure la cohérence parfaite entre le code source et la version en ligne, tout en facilitant la maintenance et le suivi des évolutions.



The screenshot shows the Scalingo dashboard for the user 'eleonora-tartaglia'. At the top, a message says 'Yay! You are now trialing Scalingo. You have 22 days to try out our features!' Below the header, there's a search bar and a 'Create an application +' button. The main navigation menu includes 'Overview', 'Apps', 'Collaborators', 'Consumption', 'Billing', and 'Support plans'. On the left, under 'Apps', it says 'Last apps deployed.' and lists one entry: 'eleonora-tartaglia / artefactoprojet / artefacto' deployed '3 days ago'. There's a 'View all' link. On the right, under 'Activity', it says 'Latest events.' and lists several log entries: 'Login success', 'Successful deployment on artefacto', 'Login success', 'Login success', and 'Command run on artefacto'. Each log entry has a small icon next to it.

XI.2 Configuration

L'application utilise une base MySQL hébergée via un add-on Scalingo MySQL Sandbox.

Lors de sa création, Scalingo génère automatiquement une URL unique DATABASE_URL contenant l'ensemble des informations de connexion (hôte, port, utilisateur, mot de passe et base). Laravel sait interpréter cette URL directement, sans nécessiter la définition manuelle de variables comme DB_HOST, DB_PORT ou DB_PASSWORD.

Le fichier config/database.php a donc été configuré pour exploiter cette URL unique selon les bonnes pratiques du framework. Cette approche simplifie la gestion de la configuration et renforce la portabilité du projet entre environnements (local, préproduction, production).

Plusieurs variables d'environnement essentielles ont été configuré dans le tableau de bord Scalingo, garantissant la sécurité et le bon fonctionnement du projet en production :

APP_URL : définit l'URL publique de l'application en production, SESSION_DOMAIN..

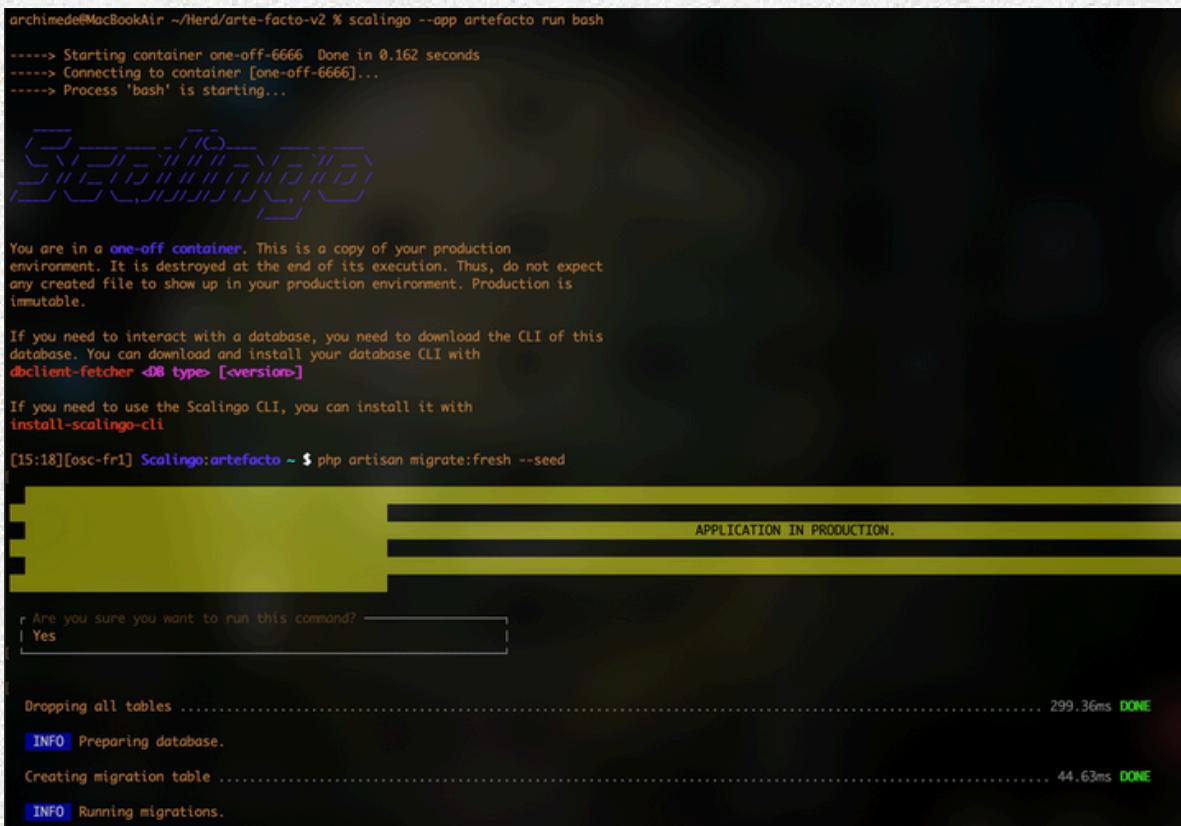
Pour le front-end, Arte Facto utilise Vite pour la compilation des fichiers CSS et JavaScript.

Afin de garantir une compatibilité optimale avec Scalingo, le build a été effectué en local avant le déploiement : **npm run build**.

Le dossier généré public/build a ensuite été ajouté manuellement au dépôt GitHub. Cette méthode permet d'intégrer directement les fichiers optimisés dans la version déployée, sans recourir à des multi-buildpacks ou à des étapes de build en ligne.

XI.3 Mise en ligne et exploitation

Une fois le déploiement automatique activé, un shell distant a été ouvert sur le conteneur Scalingo via la commande : **scalingo --app artefacto run bash**.



```
archimede@MacBookAir ~/Herd/arte-facto-v2 % scalingo --app artefacto run bash
----> Starting container one-off-6666 Done in 0.162 seconds
----> Connecting to container [one-off-6666]...
----> Process 'bash' is starting...

You are in a one-off container. This is a copy of your production environment. It is destroyed at the end of its execution. Thus, do not expect any created file to show up in your production environment. Production is immutable.

If you need to interact with a database, you need to download the CLI of this database. You can download and install your database CLI with dbclient-fetcher <DB type> [<version>]

If you need to use the Scalingo CLI, you can install it with install-scalingo-cli

[15:18][osc-fr1] Scalingo:artefacto ~ $ php artisan migrate:fresh --seed
[Progress Bar]
APPLICATION IN PRODUCTION.

r Are you sure you want to run this command? [Y/n]
| Yes

Dropping all tables ..... 299.36ms DONE
INFO Preparing database.

Creating migration table ..... 44.63ms DONE
INFO Running migrations.
```

Depuis cet environnement isolé, plusieurs commandes artisan ont été exécutées pour finaliser la mise en ligne : établir le lien symbolique entre le stockage privé et le répertoire public pour les fichiers uploadés, recréer la base et injecter les données initiales nécessaires aux tests et nettoyer et optimiser les caches pour garantir des performances maximales :

```
php artisan migrate:fresh --seed
php artisan storage:link
php artisan cache:clear
php artisan route:clear
php artisan config:cache
```

Enfin, un redémarrage complet de l'application a été effectué : **scalingo --app artefacto restart**. Cette étape permet de relancer tous les processus et d'appliquer immédiatement les modifications récentes.

XII. Évolutions & Roadmap

Les évolutions d'Arte Facto visent à renforcer la fiabilité, la sécurité et le réalisme du marché de l'art numérique.

Cinq axes guident cette roadmap : paiement sécurisé, enchères temps réel, sécurité renforcée, tableau de bord enrichi et messagerie interne.

L'intégration d'un système de paiement comme Stripe permettra de gérer le cycle complet des transactions : autorisation, capture, remboursement et génération automatique de factures. Chaque opération sera historisée pour garantir traçabilité et conformité.

Les enchères en temps réel s'appuieront sur des WebSockets (Laravel Echo, Pusher), offrant une communication bidirectionnelle instantanée, plus fluide et économique que le polling. Des bots simulateurs permettront de tester la robustesse et la cohérence transactionnelle du système.

Le renforcement de la sécurité passera par la mise en place d'une double authentification (2FA) et une gestion plus stricte des fichiers sensibles, notamment les pièces d'identité.

Le tableau de bord administrateur évoluera vers un outil d'analyse complet : statistiques dynamiques, suivi des ventes, enchères, utilisateurs vérifiés et taux de conversion.

Enfin, une messagerie interne sécurisée permettra aux acheteurs et vendeurs d'échanger directement, tout en assurant la modération et la traçabilité des échanges.

Ces évolutions prolongent la philosophie d'Arte Facto : une plateforme d'enchères à la fois humaine, fiable et exemplaire sur le plan technique.

XIII. Épilogue

Le projet Arte Facto a été bien plus qu'un simple exercice technique. Il a représenté une aventure complète, une immersion dans la création d'un univers à la croisée de la technologie et de la poésie, où chaque choix, de la base de données au design, participait à une vision cohérente et sensible.

J'ai mené seule ce projet, du concept initial à la mise en production, apprenant à structurer, à corriger, à affiner, jusqu'à trouver un équilibre entre logique et émotion. Au fil du développement, j'ai découvert ce que signifiait vraiment construire une application vivante : penser pour l'utilisateur, anticiper ses gestes, traduire une idée abstraite en expérience concrète.

J'y ai consolidé mes connaissances en Laravel, Livewire, Tailwind.. approfondi mes réflexes en matière de sécurité, de déploiement, et de gestion de projet. Mais surtout, j'y ai appris la patience du code, la rigueur de la pensée et la liberté créative qu'offre la programmation.

Arte Facto m'a permis de prendre confiance, de devenir plus autonome, et de mieux comprendre la place du développeur dans le tissu invisible du web : un artisan du lien, entre l'humain et la machine.

Avant de refermer ce dossier, je tiens à exprimer ma profonde gratitude envers toutes celles et ceux qui ont accompagné ce cheminement : notamment mes formateurs : Sambeau pour ses connaissances, sa patience, son encouragement à aller toujours plus loin, Aïcha pour sa bienveillance, son soutien et qui a su me redonner confiance même dans les moments de doutes, sans oublié mon compagnon qui a été ma boussole tous au long de ce projet , même si ce n'était pas facile tout les jours..

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

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