



# DOSSIER PROFESSIONNEL (DP)

<i>Nom de naissance</i>	► TARTAGLIA
<i>Nom d'usage</i>	►
<i>Prénom</i>	► Eléonora
<i>Adresse</i>	► 3A Rue de la Petite Colline 06250 MOUGINS-LE-HAUT

## Titre professionnel visé

Développeur Web et Web Mobile

### MODALITÉ D'ACCÈS :

- Parcours de formation
- Validation des Acquis de l'Expérience (VAE)

## Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel.  
**Ce titre est délivré par le Ministère chargé de l'emploi.**

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen.**

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

### Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.
2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle.
3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation
4. de l'entretien final (dans le cadre de la session titre).

*[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]*

### Ce dossier comporte :

- ▶ pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- ▶ un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- ▶ une déclaration sur l'honneur à compléter et à signer ;
- ▶ des documents illustrant la pratique professionnelle du candidat (facultatif)

*Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.*

# DOSSIER PROFESSIONNEL (DP)

## Sommaire

---

### Exemples de pratique professionnelle

<b>Activité-Type N°1 : Développer la partie front-end d'une application web</b>	<b>p.</b>	<b>5</b>
▶ CP1 – Installer et configurer son environnement de travail	p.	5
▶ CP2 – Maquetter une application web ou web mobile	p.	8
▶ CP3 – Réaliser des interfaces utilisateur statiques	p.	10
▶ CP4 – Développer des interfaces dynamiques	p.	14
<b>Activité-Type N°2 : Développer la partie back-end d'une application web</b>	<b>p.</b>	<b>16</b>
▶ CP5 – Mettre en place une base de données relationnelle	p.	16
▶ CP6 – Développer des composants d'accès aux données	p.	19
▶ CP7 – Développer des composants métiers côté serveur	p.	23
▶ CP8 – Documenter le déploiement d'une application web	p.	27
<b>Titres, diplômes, CQP, attestations de formation (<i>facultatif</i>)</b>	<b>p.</b>	<b>30</b>
<b>Déclaration sur l'honneur</b>	<b>p.</b>	<b>31</b>
<b>Documents illustrant la pratique professionnelle (<i>facultatif</i>)</b>	<b>p.</b>	<b>32</b>
<b>Annexes (<i>Si le RC le prévoit</i>)</b>	<b>p.</b>	<b>33</b>

# **EXEMPLES DE PRATIQUE PROFESSIONNELLE**

# DOSSIER PROFESSIONNEL (DP)

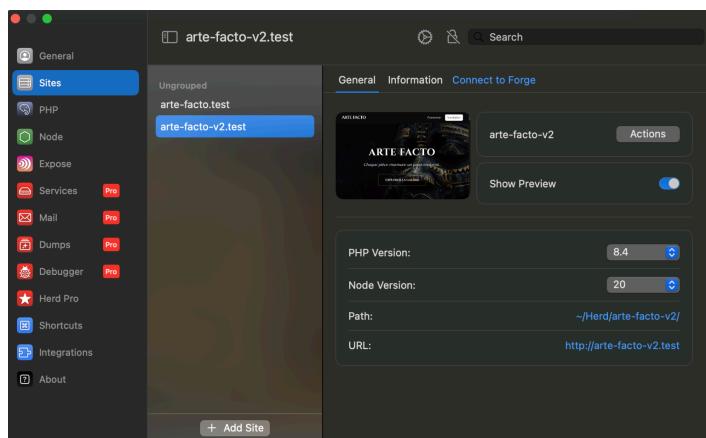
## Activité-type 1

### Développer la partie front-end d'une application web

Compétence n° 1 : *Installer et configurer son environnement de travail*

#### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Pour le projet Arte-Facto, j'ai installé et configuré mon environnement de développement complet grâce à Laravel Herd, un outil qui automatisé la mise en place de PHP, Composer, Node.js, NPM et Laravel dans leurs dernières versions stables. J'ai créé mon projet avec la commande `laravel new arte-facto-v2`, ce qui a généré le squelette de l'application, le dossier de travail et un serveur de développement accessible à l'adresse <http://arte-facto-v2.test>.



L'environnement s'est donc configuré automatiquement, sans manipulation manuelle de PHP ou MySQL. Le projet utilise SQLite comme base de données, ce qui simplifie les migrations et rend l'application légère et portable. Le fichier .env a été adapté pour définir les variables d'environnement, la clé d'application, la connexion SQLite, et le serveur de développement. J'ai ensuite testé le bon fonctionnement de la connexion en exécutant la commande `php artisan migrate`, qui a créé la base et les tables par défaut dans le fichier database/database.sqlite.

```
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
6
7 APP_LOCALE=en
8 APP_FALLBACK_LOCALE=en
9 APP_FAKE_LOCALE=en_US
10
11 APP_MAINTENANCE_DRIVER=file
12 # APP_MAINTENANCE_STORE=database
13
14 PHP_CLI_SERVER_WORKERS=4
15
16 BCRYPT_ROUNDS=12
17
18 LOG_CHANNEL=stack
19 LOG_STACK=single
20 LOG_DEPRECATIONS_CHANNEL=null
21 LOG_LEVEL=debug
22
23 DB_CONNECTION=sqlite
```

L'installation de Volt et Flux fait partie de la stack initiale du projet. Volt, surcouche de Livewire, gère l'authentification (inscription, connexion, profil, réinitialisation du mot de passe) et Flux offre une structure fluide et réactive pour la gestion des composants dynamiques. Même si je n'ai pas encore développé mes propres composants Volt, cette architecture me permet d'envisager la création future d'un système d'enchères en temps réel, où Volt pourra gérer les interactions utilisateurs et Flux la mise à jour asynchrone des données.

```

composer.json > ...
1
2   "$schema": "https://getcomposer.org/schema.json",
3   "name": "laravel/livewire-starter-kit",
4   "type": "project",
5   "description": "The official Laravel starter kit for Livewire.",
6   "keywords": [
7     "laravel",
8     "framework"
9   ],
10  "license": "MIT",
11  "require": {
12    "php": "^8.2",
13    "laravel/framework": "^12.0",
14    "laravel/tinker": "^2.10.1",
15    "livewire/flux": "^2.1.1",
16    "livewire/volt": "^1.7.0"
17  },

```

```

package-lock.json > {} packages
1  {
2    "name": "arte-facto-v2",
3    "lockfileVersion": 3,
4    "requires": true,
5    "packages": [
6      "": {
7        "dependencies": {
8          "@tailwindcss/vite": "^4.0.7",
9          "autoprefixer": "^10.4.20",
10         "axios": "^1.7.4",
11         "concurrently": "^9.0.1",
12         "laravel-vite-plugin": "^1.0",
13         "tailwindcss": "^4.0.7",
14         "vite": "^6.0"
15       }
16     }
17   }

```

Enfin, le système de build est assuré par Vite, associé à Tailwind CSS pour la mise en forme. Tout est orchestré à partir du fichier composer.json, dont le script composer run dev lance simultanément le serveur PHP, la file de jobs, la journalisation et la compilation des assets. Cette commande unique simplifie le lancement du projet : elle démarre à la fois le back-end et le front-end, garantissant une cohérence entre les processus. Grâce à cette configuration, j'obtiens instantanément un environnement fonctionnel, réactif et prêt à développer.

```

archimede@MacBookAir ~/Herd/arte-facto-v2 % composer run dev
> Composer\Config::disableProcessTimeout
> npx concurrently --c "#93c5fd,#4b5ffd,#fb7185,#fdb74" "php artisan serve" "php artisan queue:listen --tries=1" "php artisan tail --timeout=0" "npm run dev" --names=server,queue,logs,vite --kill-others
[vite]
[vite] > dev
[vite] > vite
[vite]
[queue]
[queue] [INFO] Processing jobs from the [default] queue.
[queue]
[logs]
[logs] [INFO] Tailing application logs. Press Ctrl+C to exit
[logs] Use -vLvv to show more details
[vite]
[vite] VITE v6.2.0 ready in 464 ms
[vite]
[vite] + Local: http://localhost:5173/
[vite] + Network: use --host to expose
[server]
[server] [INFO] Server running on [http://127.0.0.1:8000].
[server]
[server] Press Ctrl+C to stop the server
[server]
[vite]
[vite] LARAVEL v12.19.3 plugin v1.2.0
[vite]
[vite] + APP_URL: http://arte-facto-v2.test

```

## 2. Précisez les moyens utilisés :

L'ensemble du projet repose sur Laravel 12 et son écosystème intégré : Volt, Flux, Vite et Tailwind. Laravel Herd s'est chargé de gérer les versions et les dépendances, me permettant de me concentrer sur le développement plutôt que sur l'installation technique. Le projet est hébergé localement via l'URL générée par Herd, et la compilation des fichiers front est automatisée par Vite.

Le fichier .env contient les paramètres essentiels de l'environnement local, notamment la connexion SQLite, la configuration du serveur SMTP de test (Mailtrap) et les variables liées au cache, aux sessions et au filesystem. Les fichiers composer.json et package.json assurent la cohérence entre les dépendances PHP et JavaScript, garantissant que toutes les versions soient compatibles et stables.

Le système de versionnement est géré par GitHub, avec un dépôt unique et un usage de branches pour structurer les versions du projet. Cette organisation permet de revenir sur des itérations précises du développement et de maintenir un historique clair de l'évolution du code.

# DOSSIER PROFESSIONNEL (DP)

## 3. Avec qui avez-vous travaillé ?

J'ai travaillé seule sur ce projet, de l'installation initiale à la configuration complète de l'environnement. Tous les choix techniques et la mise en œuvre ont été réalisés en autonomie. J'ai toutefois utilisé GitHub Projects pour organiser mes tâches, définir les priorités et suivre l'avancement du développement, dans une logique de gestion de projet professionnelle et itérative.

## 4. Contexte

Nom de l'entreprise, organisme ou association **La Plateforme\_ École du numérique à Cannes**



Chantier, atelier, service ▶ *Projet personnel réalisé au cours de la formation : Arte-Facto*

Période d'exercice ▶ Du : **01/07/2025** au : **01/10/2025**

## 5. Informations complémentaires (facultatif)

Le choix de Volt et Flux s'inscrit dans une logique de modernité et d'anticipation. Ces bibliothèques, intégrées nativement à Laravel 12, permettent d'envisager la gestion d'événements en temps réel, essentielle pour un futur système d'enchères. L'utilisation de SQLite renforce la simplicité et la portabilité du projet, tandis que l'automatisation offerte par Herd et Vite optimise le temps de développement. Cet environnement de travail, à la fois minimaliste et puissant, garantit la stabilité du projet et facilite sa maintenance à long terme.

# Activité-type 1 Développer la partie front-end d'une application web

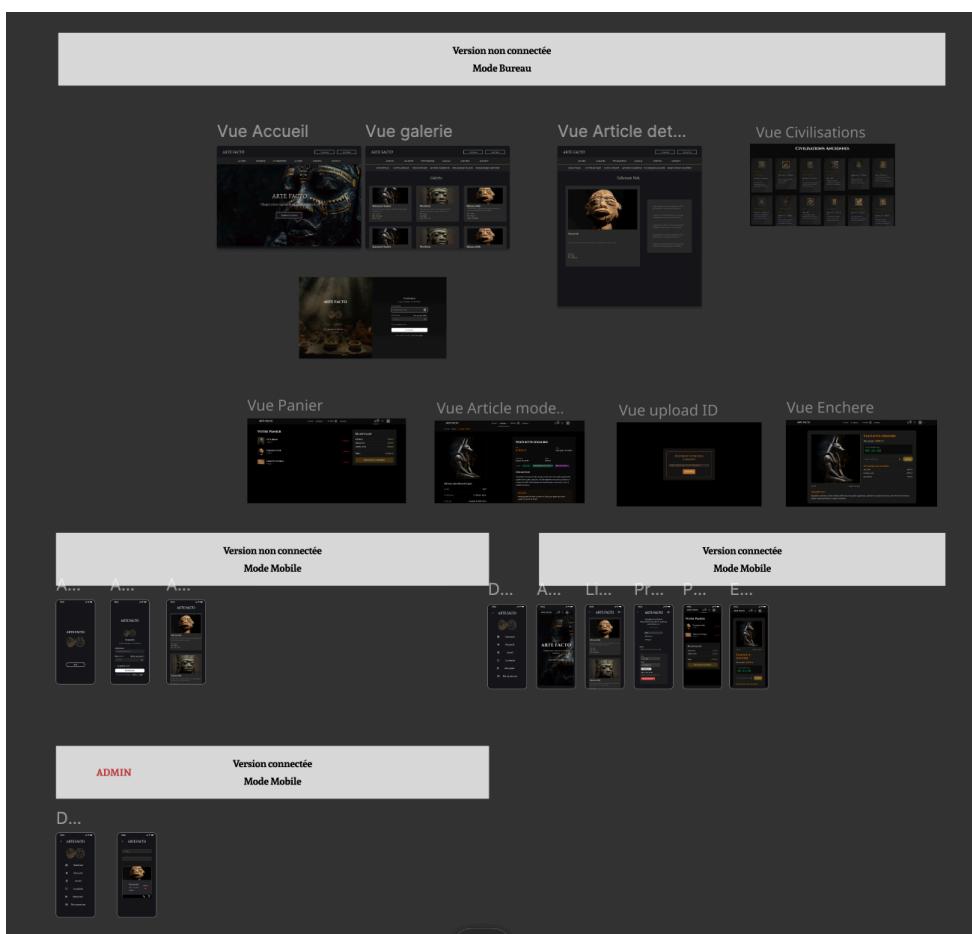
## Compétence n° 2 ▶ Maquetter une application web ou web mobile

### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Avant de concevoir l'interface finale d'Arte Facto, j'ai commencé par une phase d'inspiration structurée autour de références muséales et de maisons de vente afin d'identifier une direction artistique sobre et élégante adaptée à la mise en valeur d'artefacts.

A défaut de réaliser des wireframes formels j'ai préféré réaliser des croquis papier pour poser rapidement la hiérarchie d'information, les zones clé et les parcours cibles, sans figer le style. Ces esquisses m'ont permis de cadrer les écrans nécessaires et surtout d'organiser l'expérience en trois sections : visiteur anonyme (découverte libre du catalogue et des fiches), membre du site (accès au profil, favoris/panier et démarches de vérification), et admin (gestion de contenu et validations).

À partir de ces croquis, j'ai produit des maquettes haute fidélité dans Figma en déclinant systématiquement chaque vue en bureau et mobile. J'ai prévu les états importants (chargement, vide, erreur, validation d'email, identité en attente/validée) et j'ai clarifié un point central du projet : l'accès aux enchères n'est pas universel, il est conditionné par l'upload et la validation d'une pièce d'identité. Les écrans et messages associés encadrent donc la progression : membre non vérifié → téléchargement de la pièce → attente de validation → autorisation d'accès aux écrans. Les maquettes rendent explicites ces transitions pour éviter toute ambiguïté côté utilisateur.



# DOSSIER PROFESSIONNEL (DP)

## 2. Précisez les moyens utilisés :

J'ai utilisé Figma pour réaliser la planche de maquettes et pour documenter la palette, les polices et les composants réutilisables. La démarche s'est appuyée sur des croquis papier pour l'idéation rapide, puis sur un design system léger dans Figma (couleurs, styles de texte, composants d'inputs, boutons, cartes) afin de garantir la cohérence visuelle entre les trois sections d'utilisateurs. J'ai appliqué des principes d'accessibilité dès la maquette (contrastes suffisants, tailles de police lisibles, focus visibles prévus) et une grille responsive simple pour assurer la continuité mobile/bureau. La cartographie des rôles et droits a été décrite directement sur les écrans concernés via des notes Figma (visiteur → inscription ; membre → vérification d'email puis téléversement de la pièce ; admin → revue/validation). Enfin, j'ai veillé à la transposabilité vers l'implémentation en prévoyant des composants compatibles avec l'UI Flux et une structure de pages qui s'intègre naturellement aux gabarits Blade/Volt utilisés dans le projet.

## 3. Avec qui avez-vous travaillé ?

J'ai mené l'ensemble de la conception seule, de la recherche d'inspiration jusqu'aux maquettes finales, en m'auto-évaluant avec une grille de critères (lisibilité, accessibilité, cohérence des parcours selon les rôles) pour sécuriser les choix.

## 4. Contexte

Nom de l'entreprise, organisme ou association ► *La Plateforme\_ École du numérique à Cannes*

Chantier, atelier, service ► Projet personnel réalisé au cours de la formation : Arte-Facto

Période d'exercice ► Du : *01/07/2025* au : *01/10/2025*

## 5. Informations complémentaires (facultatif)

Les maquettes ont servi de contrat entre intention et développement : elles explicitent la séparation des trois profils, cadrent l'obtention du droit d'accès aux enchères via le téléversement de la pièce d'identité et limitent les risques d'ambiguïté pendant l'implémentation. Cette préparation visuelle a permis de conserver une identité sobre et muséale sans nuire à la clarté des actions.

# Activité-type 1 Développer la partie front-end d'une application web

Compétence n° 3 ▶ Réaliser des interfaces utilisateur statiques

## 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

J'ai intégré la page "À propos" du site Arte Facto, en veillant à conjuguer esthétique muséale, structure technique rigoureuse et accessibilité universelle. L'objectif était de créer une page immersive, évoquant une galerie d'art feutrée, tout en respectant les critères de performance et de lisibilité sur l'ensemble des supports.

La structure HTML repose sur Laravel 12 et son moteur de template Blade, à travers le composant <x-layouts.app.header>, garantissant la cohérence et la réutilisation du layout global du site. J'ai organisé le contenu à l'aide de balises sémantiques (<main>, <section>, <h1>, <h2>, <p>) et d'attributs ARIA (role="main", aria-label) pour que la lecture soit fluide et compréhensible par les lecteurs d'écran, notamment VoiceOver sur macOS.

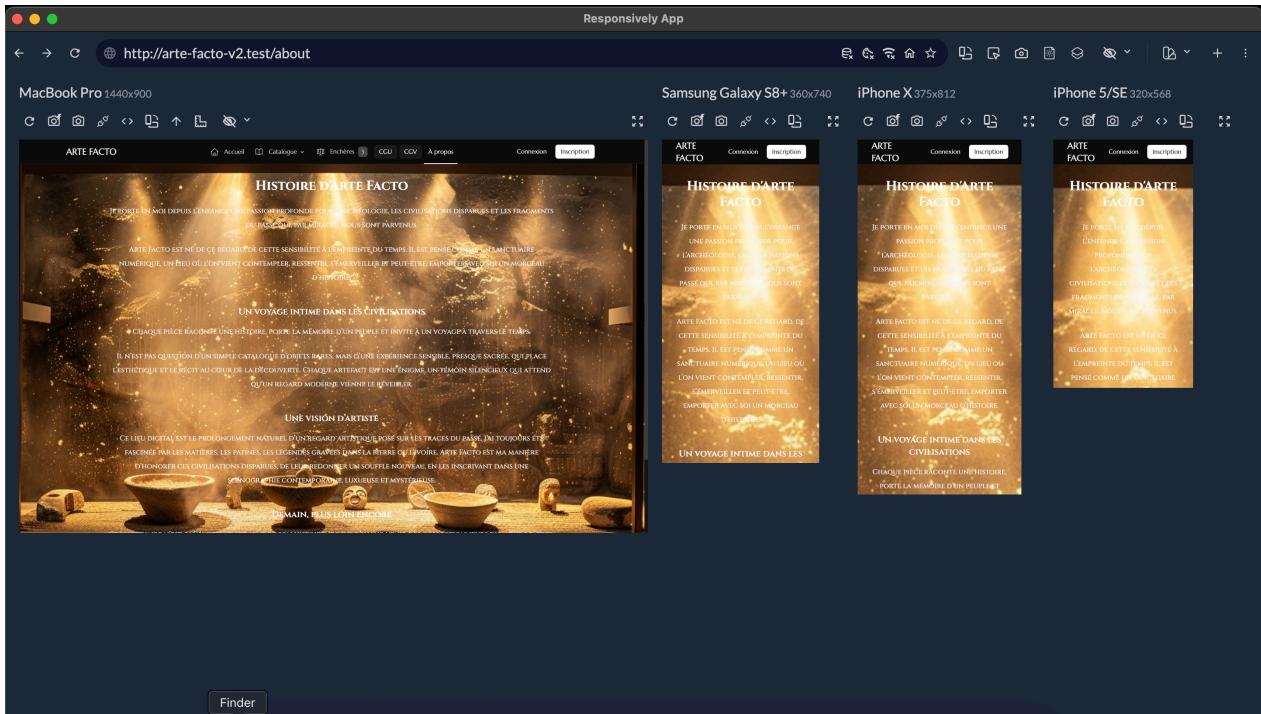
```
resources > views > about.blade.php
1  <x-layouts.app.header :title="'À propos'">
2
3  <main role="main" aria-label="Page À propos du site Arte Facto">
4      <section
5          class="relative p-8 text-center bg-cover bg-center bg-fixed min-h-screen flex flex-col justify-center items-center"
6          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 fragments du passé
26             qui, par miracle, nous sont parvenus.
27         </p>
```

La mise en forme a été assurée avec Tailwind CSS, selon une approche mobile-first. Les breakpoints par défaut (sm, md, lg, xl) ont permis d'ajuster la typographie, les marges et les espacements : par exemple, text-3xl md:text-4xl pour les titres et p-8 md:p-12 pour les espacements, assurant une lisibilité constante de 320 px (iPhone SE) à 1440 px (MacBook Pro). L'effet de profondeur visuelle a été obtenu par la combinaison d'un fond fixe bg-cover bg-fixed bg-center et d'un calque bg-[rgba(0,0,0,0.55)] backdrop-blur-[1px], simulant une vitrine légèrement floutée sans compromettre les performances.

Chaque paragraphe utilise une police Cinzel, des contrastes renforcés (text-zinc-100/95) et un interlignage généreux (leading-8) pour assurer un confort de lecture élevé. Les sections ont été pensées pour que le texte reste lisible quel que soit le niveau de zoom, sans perte de hiérarchie ni de cohérence visuelle.

Enfin, j'ai testé le rendu sur différents appareils grâce à Responsively App, un outil spécialisé dans le test multi-résolution en temps réel, permettant d'ajuster les comportements responsive simultanément sur plusieurs tailles d'écran.

# DOSSIER PROFESSIONNEL (DP)

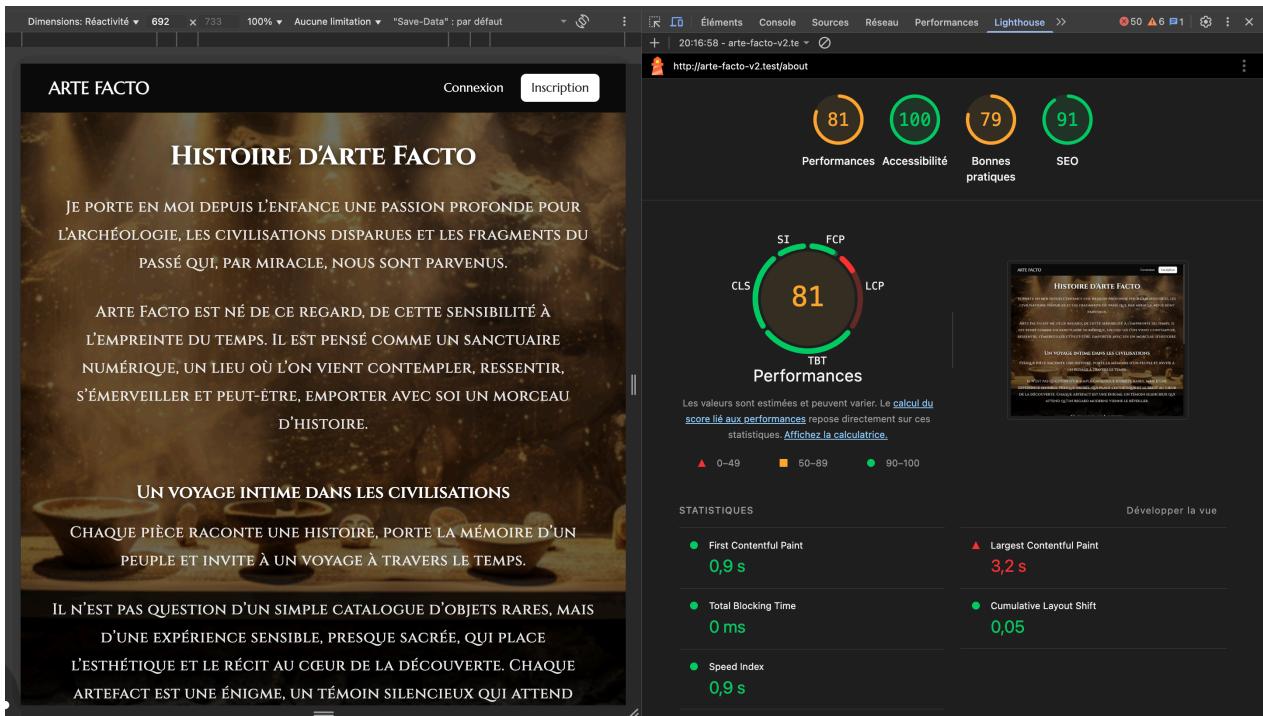


## 2. Précisez les moyens utilisés :

J'ai développé cette page avec VS Code et j'ai utilisé Responsively App pour simuler et contrôler le rendu adaptatif sur plusieurs résolutions (MacBook Pro 1440×900, Samsung S8+, iPhone X, iPhone SE..). Cet outil m'a permis de valider la cohérence du design et du confort de lecture sur tous les formats en plus de mon simulateur Xcode.

Concernant l'accessibilité, le code Blade et HTML est conçu pour être nativement compatible avec VoiceOver (macOS) grâce à une hiérarchie claire de titres (`<h1>` unique suivi de `<h2>` logiques), une balise `<main>` explicite, et des textes descriptifs sans abréviations. Cette approche garantit que la navigation par lecture vocale (tabulation et commande “VO + flèche”) restitue correctement le contenu sans confusion structurelle.

L'évaluation technique a été réalisée via Google Lighthouse, intégré à Chrome. Le rapport a confirmé la qualité du travail : 81 en performance, 100 en accessibilité, 79 en bonnes pratiques et 91 en SEO. Ces résultats attestent d'un code propre, bien structuré et conforme aux recommandations WCAG 2.1.



### 3. Avec qui avez-vous travaillé ?

J'ai mené cette intégration seule.

### 4. Contexte

Nom de l'entreprise, organisme ou association ► **La Plateforme\_ École du numérique à Cannes**

Chantier, atelier, service ► Projet personnel réalisé au cours de la formation : Arte-Facto

Période d'exercice ► Du : **01/07/2025** au : **01/10/2025**

### 5. Informations complémentaires (facultatif)

Cette page illustre ma capacité à allier esthétique visuelle, rigueur technique et optimisation du code. Les scores Lighthouse témoignent d'un travail approfondi sur l'accessibilité et le SEO, tout en conservant une atmosphère immersive. La logique mobile-first, l'utilisation maîtrisée des breakpoints et le contrôle de contraste font de cette intégration un modèle de responsive design performant et durable.

# DOSSIER PROFESSIONNEL (DP)

## Activité-type 1 Développer la partie front-end d'une application web

*Compétence n° 4 ► Développer des interfaces dynamiques*

---

### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

J'ai développé la page principale de la galerie des artefacts du site Arte Facto, dédiée à l'exploration et au tri d'objets historiques par région, civilisation et mots-clés. L'objectif était d'offrir une navigation fluide et réactive, sans recharge complet de la page, afin d'améliorer l'expérience utilisateur et la performance du site.

L'interface repose sur une communication asynchrone front-back, comparable au fonctionnement d'un appel AJAX, mais gérée nativement via le framework Livewire. Cette technologie permet de relier directement les interactions de l'utilisateur à la logique serveur, sans JavaScript manuel, en actualisant uniquement la partie du DOM concernée.

Concrètement, j'ai mis en place plusieurs fonctionnalités interactives :

- Une barre de recherche en direct, reliée à la propriété `wire:model.debounce.300ms="search"`, permettant de filtrer les artefacts dès la saisie d'un mot-clé (par exemple "masque" ou "vase"). Les résultats s'actualisent automatiquement en fonction de la saisie, sans recharge de page.
- Des filtres dynamiques par région et par civilisation, accessibles depuis un menu déroulant et le header principal. Lorsqu'un filtre est modifié, la grille d'artefacts se met à jour instantanément grâce aux liaisons Livewire (`wire:model.live="regionFilter"`), offrant une expérience fluide et réactive.
- Un système de pagination dynamique, entièrement intégré à Livewire, qui recharge uniquement le contenu de la grille et non l'ensemble de la page. Cela permet à l'utilisateur de continuer sa navigation sans rupture d'affichage ni retour en haut de page.
- Une grille d'affichage interactive des artefacts, avec effets visuels à l'aide de Tailwind CSS : transitions (transition-transform duration-700), effets de survol (group-hover:scale-105), et encadrements dynamiques à la couleur dorée (border-amber-900/30). Chaque carte affiche l'image, le titre, le prix et le statut (vendu ou disponible) en temps réel.
- Un affichage conditionnel du titre de la page, adapté automatiquement selon les filtres actifs : "Collection des Mayas" si un filtre de civilisation est sélectionné, "Collection des Amériques" si une région est filtrée, "Galerie des Artefacts" par défaut.

L'ensemble de ces fonctionnalités ont été intégrées dans le composant `resources/views/livewire/artifacts/index.blade.php`, associé au fichier `Index.php`, qui gère la logique Livewire côté front. Le tout a été encapsulé dans le layout global `components/layouts/app.blade.php` pour préserver la cohérence visuelle du site.

```

{{-- Filtres --}}
<div class="bg-zinc-900/50 border-b border-amber-900/30 py-6">
    <div class="container mx-auto px-4">
        <div class="flex flex-wrap gap-4 items-center justify-between">
            {{-- Recherche --}}
            <div class="flex-1 min-w-[300px]">
                <input type="text"
                    wire:model.live.debounce.300ms="search"
                    placeholder="Rechercher un artefact..."
                    class="w-full px-4 py-2 bg-black border border-amber-900/30 text-white">
            </div>

            {{-- Filtres --}}
            <div class="flex gap-4">
                {{-- Région --}}
                <select wire:model.live="regionFilter">
                    <option value="">Toutes les régions</option>
                    @foreach($regions as $region)
                        <option value="{{ $region }}>{{ $region }}</option>
                    @endforeach
                </select>
            </div>
        </div>
    </div>
</div>

```

## 2. Précisez les moyens utilisés :

Le développement a été réalisé sous Laravel 12, en s'appuyant sur : Livewire et Flux pour la communication asynchrone front-back. Blade pour le rendu des vues dynamiques et l'affichage conditionnel. Tailwind CSS pour la mise en forme réactive, fluide et épurée. Vite pour la gestion et la compilation des assets JavaScript et CSS.

L'interface a été testée sous Google Chrome avec l'outil Inspecteur / DevTools, permettant de suivre les requêtes Livewire en temps réel et de vérifier que les composants s'actualisaient bien sans recharge complet.

# DOSSIER PROFESSIONNEL (DP)

The screenshot shows a browser window with the title "Arte Facto". The page content is a gallery titled "GALERIE DES ARTEFACTS" featuring two artifacts: "MASQUE TRIBU FANG" and "MASQUE MAYA". The developer tools Network tab is open, showing a list of requests. One request, labeled "update", is selected. The details panel shows the following information:

- URL De Requête:** http://arte-facto-v2.test/livewire/update
- Méthode De Requête:** POST
- Code D'état:** 200 OK
- Adresse Distante:** 127.0.0.1:80
- Règlement Sur Les URL De Provenance:** strict-origin-when-cross-origin

The "En-têtes de réponse" section includes:

- Cache-Control: max-age=0, must-revalidate, no-cache, no-store, private
- Connection: keep-alive
- Content-Encoding: gzip
- Content-Type: application/json
- Date: Sat, 01 Nov 2025 14:44:43 GMT
- Expires: Fri, 01 Jan 1990 00:00:00 GMT
- Pragma: no-cache
- Server: nginx/1.25.4
- Set-Cookie: XSRF-TOKEN=eypJpdil6lmZnNTNRzRpC0tpMzlVxJjSHc9PSlsinZhbHVljoiaT4UFzKwm9tGwIkREkIvEdVzFHTEdYwVZkbkJwaXq4NHPStGKcUd0OUg3SlwL0d2bfWZE1LWWVVxYjISTNVc3dFeovVmab3RrkIB5aGtmOStJUEiT3FjamYSfpJYrBzdw5kUEzJmxaOnWTBkNmhsExZnZfpY1QlCJtYWMI0l0NWfINTTfymEY2VmOWRInm4NjNxZglyOTUwNzzM2JmODhKMWFJZJYIWY3ZT14MzbRzTRnOGQ2ODg4NDNliwiGfnjoln%3D; expires=Sat, 01 Nov 2025 16:44:43 GMT; Max-Age=7200; path=/; sameSite=lax
- artefact-session=eypJpdil6lm9H001Tr2RXRvdyb2NrJhOVZB2c9PSlsInZhbHVljoI0GHQmtFUnl2OzTV4amUemsnVKLwA3L2NOVj0by98aInRSRfL0d2bfWZE1LWWVVxYjISTNVc3dFeovVmab3RrkIB5aGtmOStJUEiT3FjamYSfpJYrBzdw5kUEzJmxaOnWTBkNmhsExZnZfpY1QlCJtYWMI0l0NWfINTTfymEY2VmOWRInm4NjNxZglyOTUwNzzM2JmODhKMWFJZJYIWY3ZT14MzbRzTRnOGQ2ODg4NDNliwiGfnjoln%3D; expires=Sat, 01 Nov 2025 16:44:43 GMT; Max-Age=7200; path=/; httpOnly; sameSite=lax
- Transfer-Encoding: chunked

## 3. Avec qui avez-vous travaillé ?

J'ai développé cette interface seule, dans la continuité de l'intégration graphique réalisée précédemment.

## 4. Contexte

Nom de l'entreprise, organisme ou association ► *La Plateforme\_ École du numérique à Cannes*

Chantier, atelier, service ► Projet personnel réalisé au cours de la formation : Arte-Facto

Période d'exercice ► Du : **01/07/2025** au : **01/10/2025**

## 5. Informations complémentaires (facultatif)

Cette partie du projet illustre la capacité à créer une interface dynamique, réactive et immersive sans recourir à des appels AJAX classiques ni à du JavaScript brut. Grâce à Livewire, la synchronisation entre le front et le back s'effectue automatiquement via des liaisons de propriétés (wire:model) et des mises à jour asynchrones du DOM, assurant une expérience fluide.

Cette approche modernise la galerie d'artefacts d'Arte Facto en transformant une page statique en une interface vivante et évolutive, proche des standards des sites professionnels de e-commerce ou de musées virtuels.

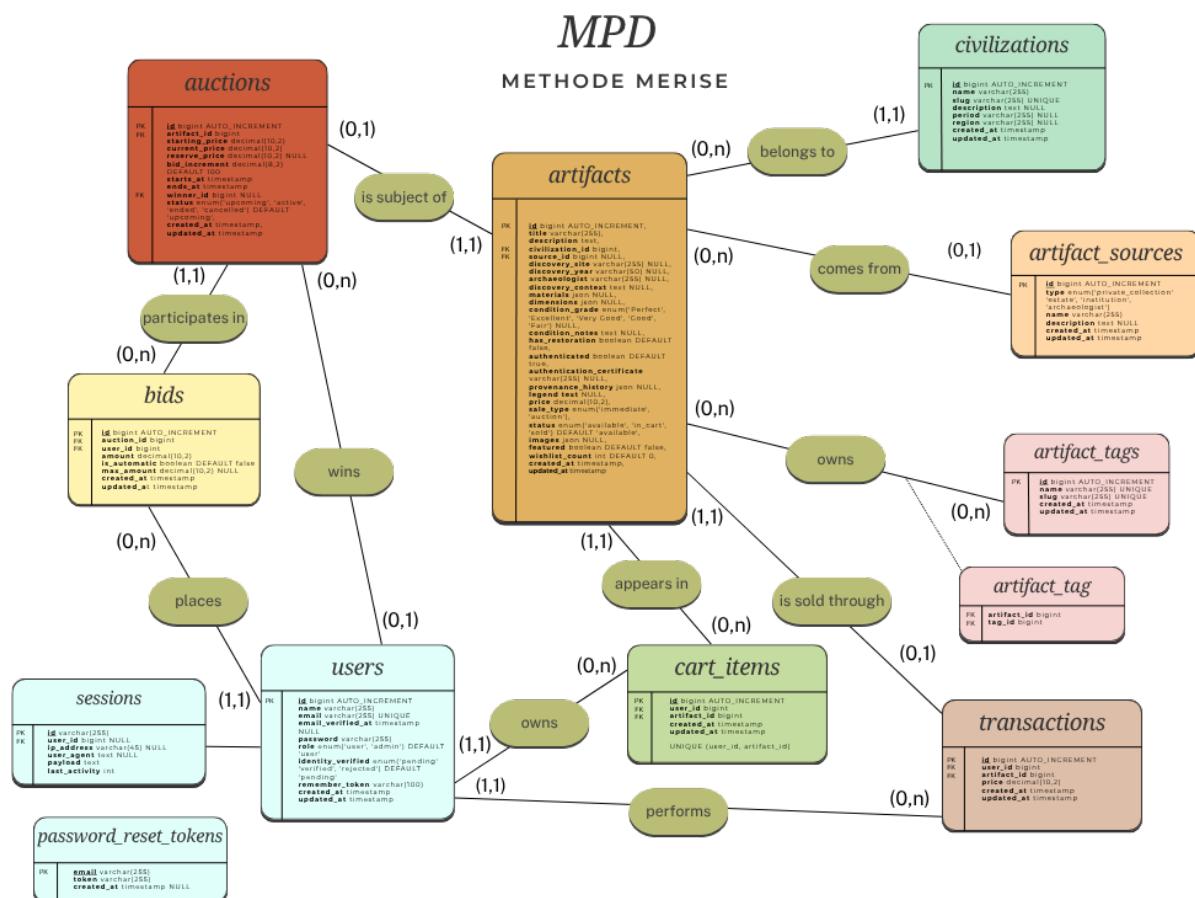
## Activité-type 2 Développer la partie back-end d'une application web

**Compétence n° 5 ► Mettre en place une base de données relationnelle**

#### **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :**

J'ai construit la base de données d'Arte Facto en suivant une démarche Merise classique. J'ai d'abord élaboré un MCD pour décrire le domaine : les artefacts constituent l'entité centrale, rattachés à des civilisations et, le cas échéant, à une source d'inventaire ; un artefact peut être proposé à la vente immédiate ou aux enchères ; il peut apparaître dans le panier d'utilisateurs et donner lieu à des transactions puis à des commandes et lignes de commande. À partir de ce MCD, j'ai décliné un MLD en précisant les identifiants techniques, les cardinalités, les clés étrangères et les associations, puis je l'ai transformé en MPD en choisissant les types concrets et les index utiles.

Cette trajectoire MCD → MLD → MPD m'a permis de garantir la normalisation (3FN), la cohérence référentielle et la maintenabilité future du schéma.



# DOSSIER PROFESSIONNEL (DP)

Une fois le MPD stabilisé, j'ai traduit le modèle en migrations Laravel et je l'ai exécuté en SQLite pour le développement. Le cœur du système est la table artifacts, qui porte l'identité, la description et les caractéristiques d'une pièce, ainsi que son positionnement commercial (type de vente, prix, statut).

Les relations structurantes sont matérialisées par des clés étrangères vers civilizations et artifact\_sources, tandis que les tables d'écriture métier (enchères, panier, transactions, commandes) organisent le cycle de vie commercial. L'un des points essentiels de cette conception a été l'équilibre entre la richesse des données et la performance des requêtes. C'est pour cette raison que j'ai intégré des index sur les colonnes les plus sollicitées, comme civilization\_id, status et sale\_type, afin d'optimiser les filtrages et recherches côté front.

Voici un extrait représentatif de la migration create\_artifacts\_table.php, qui illustre la logique des clés étrangères, des contraintes d'intégrité et des types adaptés à chaque donnée :

```
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);
});
```

Chaque champ a été pensé pour refléter une réalité précise du domaine archéologique : materials et dimensions sont stockés au format JSON pour leur flexibilité, tandis que les champs booléens comme authenticated ou has\_restoration permettent un traitement clair des états logiques. Cette structure rend la base aussi bien exploitable par le back-end que par les vues dynamiques gérées avec Livewire.

L'implémentation s'appuie sur l'ensemble des migrations présentes dans database/migrations. Après génération, j'ai initialisé un jeu de données de démonstration cohérents, simulant un environnement de galerie archéologique réaliste via la commande php artisan migrate:fresh --seed, ce qui m'a permis de vérifier l'alignement entre le modèle conceptuel et les tables concrètes. Les tests de cohérence ont été réalisés via Thunder Client dans VS Code.

## 2. Précisez les moyens utilisés :

J'ai modélisé avec Merise (MCD pour les entités et leurs associations, MLD pour les attributs et les clés, MPD pour les types et index), puis j'ai implémenté l'ensemble via migrations Laravel et Artisan.

Le moteur SQLite a servi de support léger et fiable pour exécuter rapidement les cycles migration/seed, tandis que Thunder Client dans VS Code m'a permis de valider les lectures et jointures exposées par l'application, en vérifiant que les collections d'artefacts remontaient bien avec leur civilisation associée et les attributs attendus. Cette boucle MCD → MLD → MPD → migrations → seeds → requêtes applicatives garantit que la base est correctement structurée, ré-exécutable à l'identique, et contrôlée par des appels concrets.

### 3. Avec qui avez-vous travaillé ?

J'ai réalisé l'intégralité du travail seule, de la modélisation Merise jusqu'aux migrations et au peuplement de la base, en m'astreignant à une vérification systématique par requêtes applicatives sous Thunder Client.

### 4. Contexte

Nom de l'entreprise, organisme ou association **La Plateforme\_ École du numérique à Cannes**



Chantier, atelier, service▶ Projet personnel réalisé au cours de la formation : Arte-Facto

Période d'exercice ▶ Du : **01/07/2025** au : **01/10/2025**

### 5. Informations complémentaires (facultatif)

Le schéma a été pensé pour servir directement les besoins : filtrage par civilisation, facettes par région et statut de vente, récupération d'images multiples, et stockage de métadonnées souples via JSON pour les matériaux et dimensions. L'ajout d'index ciblés répond aux contraintes de tri et de pagination. Les aspects d'accès aux données (contrats Eloquent, typages, mass assignment) seront détaillés dans la compétence suivante afin de bien distinguer la structure relationnelle de la logique d'accès.

# DOSSIER PROFESSIONNEL (DP)

## Activité-type 2 Développer la partie back-end d'une application web

Compétence n° 6 ► Développer des composants d'accès aux données

### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

J'ai développé la couche d'accès aux données du projet Arte Facto en m'appuyant sur le moteur Eloquent de Laravel, afin de permettre à l'application de manipuler les artefacts, civilisations, transactions et utilisateurs de manière fluide, sans écrire directement de requêtes SQL.

L'ensemble repose sur une base de données relationnelle SQLite en environnement de développement, traduisant la structure normalisée issue du MLD, tandis que les modèles Eloquent assurent la correspondance entre les tables et les objets manipulés par le code.

Le modèle Artifact constitue l'un des composants centraux de cette architecture. Il regroupe toutes les informations relatives aux artefacts archéologiques : identité, description, contexte de découverte, caractéristiques physiques, statut commercial et relations avec les civilisations, les enchères ou les utilisateurs.

J'ai défini les propriétés \$fillable afin de préciser les champs autorisés à être remplis lors des opérations de création ou de mise à jour. Cette mesure renforce la sécurité du modèle en empêchant toute insertion non désirée dans la base.

Les propriétés \$casts assurent la conversion automatique des types de données, notamment pour les champs stockés au format JSON tels que materials, dimensions, provenance\_history ou images, ainsi que pour les booléens (has\_restoration, authenticated, featured).

Cette approche assure une parfaite symbiose entre la logique objet et la persistance des données, tout en rendant la manipulation des informations beaucoup plus naturelle côté code.

```
class Artifact extends Model
{
    use HasFactory;

    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',
    ];

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

Les relations entre modèles sont déclarées de manière explicite : belongsTo pour relier un artefact à sa civilisation ou à sa source, hasOne pour les enchères, hasMany pour les transactions, et belongsToMany pour les associations multiples comme les tags ou les utilisateurs ayant ajouté un artefact à leur panier.

Ainsi, une simple requête comme Artifat::with('civilization')->first() permet de récupérer l'ensemble des données liées, prêtes à être exploitées dans les vues front-end.

Même si le projet repose sur une base SQL, certaines structures imitent la souplesse du NoSQL : les champs JSON permettent d'enregistrer des informations semi-structurées telles que les matériaux, dimensions ou historique de provenance. Cette hybridation rend le modèle plus évolutif, sans nécessiter de modification du schéma global à chaque ajout de propriété.

L'ensemble de ces opérations a permis d'assurer un accès rapide, typé et cohérent aux données, tout en garantissant une architecture claire et extensible.

```
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);
}
```

# DOSSIER PROFESSIONNEL (DP)

```
= App\Models\Artifact {#6581
    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", "paisseur": "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: 1800,
    sale_type: "immediate",
    status: "available",
    images: ["https://cdn.midjourney.com/4ff414eb-7015-4559-b1f7-49437d36e074/0_1.png"],
    featured: 0,
    wishlist_count: 12,
    created_at: "2025-10-15 07:51:31",
    updated_at: "2025-10-15 07:51:31",
    civilization: App\Models\Civilization {#6591
        id: 5,
        name: "Royaume 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."}]
```

## 2. Précisez les moyens utilisés :

Le développement des composants d'accès aux données s'est effectué dans Visual Studio Code avec l'environnement Laravel 12 sous Herd et SQLite. Les modèles ont été générés et ajustés à l'aide des commandes artisan (make:model, migrate, tinker), puis testés dans Tinker, la console interactive de Laravel, pour valider les relations et la structure des objets.

Les tests réalisés avec la commande `Artifact::with('civilization')->first()` ont confirmé la conformité des relations Eloquent et la bonne interprétation des champs convertis par `$casts`. La combinaison de ces outils a offert une approche complète : manipulation objet côté code, validation structurelle dans Tinker.

---

## 3. Avec qui avez-vous travaillé ?

J'ai développé et testé cette couche d'accès aux données seule, en cohérence avec le schéma conceptuel validé lors de la phase précédente.

#### 4. Contexte

Nom de l'entreprise, organisme ou association **La Plateforme\_ École du numérique à Cannes**

▶

Chantier, atelier, service▶ Projet personnel réalisé au cours de la formation : Arte-Facto

Période d'exercice ▶ Du : **01/07/2025** au : **01/10/2025**

#### 5. Informations complémentaires (facultatif)

Ce travail m'a permis d'approfondir la logique objet-relation propre à Eloquent et de comprendre la différence entre une base relationnelle SQL et une approche documentaire NoSQL.

En combinant les deux logiques : structure fixe et champs JSON flexibles, j'ai conçu un modèle hybride, robuste et extensible, capable d'évoluer sans refonte majeure.

# DOSSIER PROFESSIONNEL (DP)

## Activité-type 2 Développer la partie back-end d'une application web

Compétence n° 7 ► Développer des composants métiers côté serveur

### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le cadre du développement du projet Arte Facto, j'ai conçu et implémenté la logique métier centrale de la plateforme : le système de panier concurrentiel et d'interaction avec les artefacts.

Cette mécanique a pour but de simuler les comportements d'une galerie d'art en ligne, où plusieurs utilisateurs peuvent s'intéresser à une même pièce rare, sans pour autant en fausser la disponibilité ou les règles d'acquisition.

Cette logique métier s'articule principalement autour de deux composants :

- Le composant Livewire App\LiveWire\Artifacts>Show, responsable de la consultation détaillée d'un artefact et de l'ajout au panier.

- Le contrôleur App\Http\Controllers\CartController, garant des règles de gestion du panier, de la validation des commandes et de la sécurité transactionnelle.

Le composant Livewire gère à la fois la présentation des informations archéologiques (civilisation, matériaux, contexte de découverte) et les interactions dynamiques entre utilisateur et artefact.

Lorsqu'un utilisateur consulte une œuvre, le composant vérifie via la méthode checkCartStates() si elle est déjà dans son panier (\$inCart) ou dans celui d'autres utilisateurs (\$otherCartsCount).

Cette donnée métier influence directement l'affichage : la mention "✓ Dans votre panier" ou "2 utilisateurs ont cet artefact dans leur panier" est générée en temps réel, traduisant la rareté et la tension commerciale du marché de l'art.

```
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();
    }
}
```

La méthode addToCart() applique ensuite les règles de validation :

- Seuls les utilisateurs authentifiés peuvent ajouter un artefact.
- Un artefact ne peut être ajouté qu'une fois par utilisateur.
- L'ajout déclenche un événement Livewire (cartUpdated) pour mettre à jour la barre de navigation, créant une synchronisation front-back instantanée sans recharge de page.

```

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

    if (! $this->artifact->usersInCart->contains(Auth::id())) {
        CartItem::create([
            'user_id' => Auth::id(),
            'artifact_id' => $this->artifact->id,
        ]);
    }

    $this->artifact->load('usersInCart');
    $this->checkCartStates();

    // Émet l'événement pour CartIndicator
    $this->dispatch('cartUpdated');
}

```

Enfin, la méthode `participateAuction()` illustre la notion de contrainte métier spécifique : l'accès aux enchères est conditionné à la vérification d'identité (`identity_verified`).

Cette logique, issue du droit du patrimoine, a été traduite dans le code par l'ouverture conditionnelle d'une modale Livewire invitant l'utilisateur à déposer un justificatif.

Le contrôleur `CartController` a été au centre de la couche transactionnelle. La méthode `checkout()` utilise une transaction `DB::transaction()` combinée à un verrouillage optimiste (`lockForUpdate()`), garantissant qu'un artefact ne puisse être acheté simultanément par plusieurs utilisateurs.

Chaque commande est générée sous la forme d'un snapshot immuable (`OrderItem`) contenant les données archéologiques exactes au moment de l'achat : provenance, dimensions, matériaux, légende et certification.

Cette approche répond à un impératif de fiabilité historique : dans Arte Facto, même après suppression ou mise à jour d'un artefact, les données d'origine associées à une commande demeurent archivées telles qu'elles étaient au moment de la transaction.

# DOSSIER PROFESSIONNEL (DP)

```
app > Http > Controllers > CartController.php
31 }
32
33
34     // POST /cart/checkout
35     public function checkout(Request $request)
36     {
37         $user = Auth::user();
38
39         $items = $user->cartItems()->with('artifact.civilization', 'artifact.source')->get();
40         if ($items->isEmpty()) {
41             return back()->with('status', 'Votre panier est vide.');
42         }
43
44         try {
45             DB::transaction(function () use ($user, $items) {
46                 // 1) Revalider disponibilité + verrouiller
47                 foreach ($items as $ci) {
48                     /** @var Artifact|null $a */
49                     $a = Artifact::where('id', $ci->artifact_id)->lockForUpdate()->first();
50                     if (!$a || $a->status !== 'available' || $a->sale_type !== 'immediate') {
51                         throw new \RuntimeException("Un article n'est plus disponible.");
52                     }
53                 }
54             });
55         }
56     }
57 }
```

L'affichage des statuts et compteurs dynamiques dans la vue Blade (resources/views/livewire/artifacts/show.blade.php) permet de traduire visuellement la logique métier. Chaque changement dans la base se répercute immédiatement sur l'interface, grâce à Livewire et aux événements serveur émis après chaque action utilisateur.

```
<!-- Statut + bouton panier -->
<div class="mb-8 flex items-center gap-4">
    <p class="text-sm text-gray-400">Statut</p>

    @if($artifact->status === 'sold')
        <span class="px-3 py-1 text-sm bg-red-900/30 text-red-500">
            Vendu
        </span>
    @elseif($inCart)
        <span class="px-3 py-1 text-sm bg-[#rgb(67,54,17)]">
            ✓ Dans votre panier
        </span>
    @elseif($otherCartsCount > 0)
        <span class="px-3 py-1 text-sm bg-yellow-900/30 text-yellow-400">
            {{ $otherCartsCount }} {{ Str::plural('utilisateur', $otherCartsCount) }} {{ $otherCartsCount > 1 ? 'ont' : 'a' }} cet artefact dans {{ $otherCartsCount > 1 ? 'leur' : 'son' }} panier
        </span>
    @else
        <span class="px-3 py-1 text-sm bg-green-900/30 text-green-500">
            Disponible
        </span>
    @endif

    @if($artifact->status !== 'sold' && !$inCart)
        @if($artifact->sale_type === 'immediate')
            <button wire:click="addToCart"
                    class="px-1 py-1 text-sm"
                    style="background: #rgba(118, 94, 30, 0.58); color: #rgb(252, 252, 252); font-family: 'Cinzel', serif;">
                Ajouter au panier 🛍
            </button>
        @elseif($artifact->sale_type === 'auction')
            <button wire:click="participateAuction"
                    class="px-1 py-1 text-sm"
                    style="background: #rgba(24, 180, 160, 0.3); color: #rgb(252, 252, 252); font-family: 'Cinzel', serif;">
                Participer à l'enchère 🎰
            </button>
        @endif
    @endif
</div>
```

## 2. Précisez les moyens utilisés :

L'ensemble du module repose sur Laravel 12, Livewire 3 et la base SQLite, gérés dans l'environnement Herd. Le code métier a été développé dans Visual Studio Code, et les flux Livewire observés dans l'onglet Réseau > XHR de l'inspecteur navigateur pour analyser les échanges serveur en temps réel.

## 3. Avec qui avez-vous travaillé ?

Cette fonctionnalité a été développée individuellement, mais en cohérence avec la structure de données définie lors de la mise en place des migrations et modèles Eloquent.

## 4. Contexte

Nom de l'entreprise, organisme ou association ► *La Plateforme\_ Ecole du numérique à Cannes*

Chantier, atelier, service ► Projet personnel réalisé au cours de la formation : Arte-Facto

Période d'exercice ► Du : *01/07/2025* au : *01/10/2025*

## 5. Informations complémentaires (facultatif)

L'implémentation de cette logique métier démontre la puissance du couple Laravel + Livewire, qui me permet de conserver une architecture MVC claire et traditionnelle tout en introduisant une réactivité front-end native, sans recours à JavaScript complexe.

Grâce à Livewire, chaque composant agit comme un pont entre le modèle et la vue : les données sont manipulées en PHP côté serveur, puis synchronisées automatiquement dans l'interface sans rechargement complet de la page.

Ainsi, la méthode `addToCart()` modifie directement la base de données via le modèle `CartItem`, et le changement est immédiatement reflété dans la vue — le tout sans quitter la logique du contrôleur ni écrire de requête AJAX manuelle.

Cette approche m'a permis de garder la pureté du modèle MVC de Laravel :

- Modèles Eloquent : centralisent les relations et les règles de données (ex. `Artifact`, `Order`, `CartItem`).
- Composants Livewire : incarnent la logique métier (contrôle des paniers, vérification d'identité, filtrage dynamique).
- Vues Blade : traduisent ces comportements en éléments visuels cohérents et élégants.

# DOSSIER PROFESSIONNEL (DP)

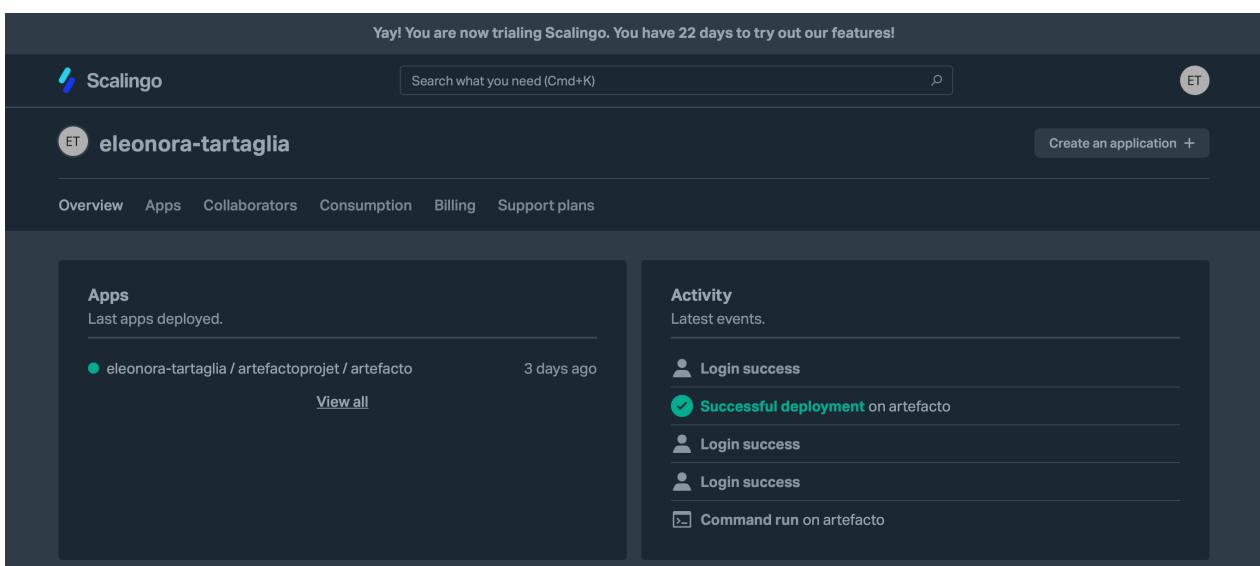
## Activité-type 2 Développer la partie back-end d'une application web

Compétence n° 8 ▶ Documenter le déploiement d'une application web

### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Le déploiement de l'application Artefacto a été réalisé sur la plateforme française Scalingo, dans un environnement Laravel 12 / PHP 8.4, afin de garantir une mise en ligne stable, sécurisée et facilement maintenable.

J'ai commencé par préparer l'environnement de production : installation de la CLI Scalingo via Homebrew, création du projet en ligne, génération du token d'accès API et connexion via le terminal.



The screenshot shows the Scalingo dashboard for the application 'eleonora-tartaglia'. At the top, a message says 'Yay! You are now trialing Scalingo. You have 22 days to try out our features!'. The dashboard has a search bar and a 'Create an application +' button. Below the header, there are tabs for Overview, Apps, Collaborators, Consumption, Billing, and Support plans. The 'Apps' section displays a list of deployed apps, with one entry for 'eleonora-tartaglia / artefactoprojet / artefacto' listed 3 days ago. The 'Activity' section shows a timeline of events, including 'Login success', 'Successful deployment' on 'artefacto', and 'Command run on artefacto'.

La première étape consistait à lier l'application Scalingo au dépôt GitHub du projet, afin de bénéficier d'un déploiement automatique à chaque git push sur la branche main. Cette approche garantit la cohérence entre le code source et la version déployée, tout en facilitant le suivi des mises à jour.

Une fois le lien établi, j'ai ajouté un add-on MySQL sandbox pour héberger la base de données en ligne. L'URL générée automatiquement (SCALINGO\_MYSQL\_URL et DATABASE\_URL) a été utilisée directement dans le fichier config/database.php, conformément aux bonnes pratiques Laravel. Ce choix évite d'avoir à déclarer manuellement les paramètres DB\_HOST, DB\_PORT ou DB\_PASSWORD, puisque Laravel sait interpréter cette URL unique.

J'ai ensuite configuré toutes les variables d'environnement indispensables à la production : APP\_URL, SESSION\_DOMAIN, FILESYSTEM\_DISK, ASSET\_URL, APP\_ENV, APP\_DEBUG=false, ainsi que les variables liées à la sécurité (SESSION\_SECURE\_COOKIE, CORS\_ALLOWED\_ORIGINS, SANCTUM\_STATEFUL\_DOMAINS).

Concernant les assets front-end, j'ai opté pour un build local de Vite : après exécution de npm run build, le dossier public/build a été ajouté manuellement au dépôt et poussé sur GitHub, garantissant ainsi une intégration directe des fichiers optimisés dans la version déployée. Cette méthode simplifie la compatibilité avec Scalingo, sans nécessiter de multi-buildpack.

Une fois le déploiement automatique déclenché, j'ai ouvert un shell distant sur le container Scalingo via la commande scalingo --app artefacto run bash. Depuis cet environnement isolé, j'ai exécuté les commandes d'exploitation suivantes : php artisan migrate:fresh --seed pour créer la base et injecter les données initiales, puis php artisan storage:link pour relier le stockage public aux fichiers uploadés.

```

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...

A small decorative graphic consisting of a grid of blue and white lines forming a stylized, abstract shape.

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
[REDACTED]
[REDACTED] APPLICATION IN PRODUCTION.
[REDACTED]

Are you sure you want to run this command?
| Yes | [REDACTED]

Dropping all tables ..... 299.36ms DONE
INFO Preparing database.
Creating migration table ..... 44.63ms DONE
INFO Running migrations.

```

Des commandes de maintenance (php artisan cache:clear, route:clear, config:cache) ont ensuite été lancées pour optimiser les performances.

Enfin, un redémarrage complet de l'application a été effectué (scalingo --app artefacto restart), suivi de vérifications fonctionnelles : connexion, navigation, création d'artefacts..

## 2. Précisez les moyens utilisés :

Ce déploiement s'appuie sur un ensemble d'outils et de services cohérents :

- Scalingo CLI pour la gestion du cycle de vie de l'application, la configuration des variables et le déclenchement des builds.
- GitHub pour la synchronisation et l'automatisation du déploiement.
- MySQL (addon Scalingo) pour la base de données relationnelle.
- Vite pour la compilation des assets front-end.
- Laravel Artisan pour les migrations, les seeders, la gestion du cache et la configuration du stockage.

L'ensemble du processus s'est déroulé sous macOS, directement depuis Visual Studio Code, avec des vérifications réseau via l'inspecteur du navigateur et des requêtes manuelles exécutées depuis le shell distant Scalingo.

Ce déploiement a permis de reproduire fidèlement l'environnement local, en assurant une compatibilité parfaite avec Livewire, la gestion des sessions sécurisées et la persistance des fichiers dans storage/app/public.

# DOSSIER PROFESSIONNEL (DP)

## 3. Avec qui avez-vous travaillé ?

Cette phase de déploiement a été réalisée individuellement.

## 4. Contexte

Nom de l'entreprise, organisme ou association **La Plateforme\_ École du numérique à Cannes**



Chantier, atelier, service▶ Projet personnel réalisé au cours de la formation : Arte-Facto

Période d'exercice ▶ Du : **01/07/2025** au : **01/10/2025**

## 5. Informations complémentaires (facultatif)

Le déploiement sur Scalingo illustre parfaitement la capacité de Laravel à s'intégrer dans un environnement Platform-as-a-Service (PaaS) tout en conservant sa philosophie artisanale et modulaire.

Grâce à la compatibilité native avec la variable DATABASE\_URL, Laravel s'adapte sans effort à la structure réseau dynamique du cloud, tout en maintenant la cohérence de son architecture MVC.

Le recours à Livewire n'a pas modifié la logique serveur : les composants continuent de communiquer avec le modèle et les contrôleurs Laravel, tandis que les mises à jour côté client s'effectuent en temps réel via des requêtes XHR automatiques. Ainsi, même en production, le modèle MVC reste intégralement respecté : la réactivité de l'application repose sur des échanges PHP natifs, et non sur une dépendance à un framework JavaScript externe.

Cette architecture confère à Artefacto un déploiement stable, reproductible et sécurisé, tout en offrant la fluidité d'une application moderne.

## **Titres, diplômes, CQP, attestations de formation**

*(facultatif)*

<b>Intitulé</b>	<b>Autorité ou organisme</b>	<b>Date</b>
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.

# DOSSIER PROFESSIONNEL (DP)

## Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] **Eléonora Tartaglia**,

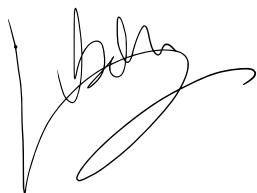
déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l'auteur(e) des réalisations jointes.

Fait à **Cannes**

le **01 Octobre 2025**

pour faire valoir ce que de droit.

Signature :



## Documents illustrant la pratique professionnelle

(facultatif)

# DOSSIER PROFESSIONNEL (DP)

## ANNEXES

*(Si le RC le prévoit)*