Institut Supérieur de Management d'Administration et de Génie Informatique

Projet de Fin d'Études

En vue de l'obtention du diplôme de

Licence Professionnelle en Développement Web et Mobile

MONO AI Fashion Store : Plateforme e-commerce intelligente avec essayage virtuel assisté par l'IA

Réalisé par : Saad Bouhamou

Soutenu devant le jury composé de :

Président	Yassine Rayri
Encadrant	Nassim Kharmoum
Examinateur	Kamal Najem

Année universitaire **2025 - 2026**

Dédicace

Je dédie ce travail à mes chers parents, pour leur amour, leur soutien et leurs sacrifices tout au long de mon parcours.

À ma famille, pour ses encouragements constants et sa confiance.
À tous mes enseignants, qui ont contribué à ma formation et au développement de mes compétences.

Enfin, à toutes les personnes qui m'ont soutenu de près ou de loin dans la réalisation de ce projet de fin d'études.

Remerciements

Avant tout, je remercie Allah le Tout-Puissant de m'avoir accordé la santé, la patience et la persévérance nécessaires pour mener à bien ce projet de fin d'études.

J'exprime ma profonde gratitude à mon encadrant Nassim kharmoum pour sa disponibilité, ses conseils précieux et son accompagnement tout au long de la réalisation de ce travail. Ses remarques et orientations ont été d'une grande valeur pour l'aboutissement de ce projet.

Je remercie également l'ensemble des enseignants de l'ISMAGI pour la qualité de leur formation et les connaissances qu'ils nous ont transmises durant notre parcours académique.

Mes sincères remerciements s'adressent également aux membres du jury qui ont accepté d'évaluer ce travail et de consacrer une partie de leur temps à son examen.

Enfin, je tiens à remercier ma famille ainsi que toutes les personnes qui m'ont soutenu, encouragé et accompagné, de près ou de loin, durant la réalisation de ce projet.

Résumé

Ce projet de fin d'études présente la conception et le développement de MONO, une plateforme e-commerce moderne spécialisée dans la vente de vêtements, intégrant une fonctionnalité innovante d'essayage virtuel assisté par l'intelligence artificielle (AI Try-On). L'objectif principal est d'améliorer l'expérience utilisateur en proposant une interface élégante, performante et intuitive, tout en réduisant les incertitudes liées aux achats de vêtements en ligne.

L'application adopte une architecture découplée (Decoupled Architecture) séparant le Frontend du Backend. La partie Frontend est développée avec Next.js 15, React et TypeScript afin d'offrir une interface fluide, responsive et optimisée pour le référencement (SEO). Le Backend repose sur Express.js, Prisma ORM et MySQL pour assurer la gestion des données, la sécurisation des échanges, ainsi que l'implémentation des règles métier. La gestion de l'état global est réalisée à l'aide de Zustand, tandis que GSAP est utilisé pour offrir une expérience utilisateur immersive grâce à des animations performantes.

Le projet met également en œuvre une architecture modulaire facilitant la maintenance, l'évolutivité et l'intégration future d'un fournisseur d'intelligence artificielle réel grâce à un système de Provider interchangeable. Les résultats obtenus démontrent la faisabilité d'une solution e-commerce moderne, performante et évolutive répondant aux exigences techniques et ergonomiques des applications web actuelles.

Mots-clés :
E-commerce – Next.js – Express.js – Prisma ORM – Intelligence Artificielle – AI Try-On

Abstract

This final-year project presents the design and development of MONO, a modern e-commerce platform dedicated to fashion retail and enhanced with an innovative Artificial Intelligence-based virtual try-on (AI Try-On) feature. The main objective is to improve the online shopping experience by providing an elegant, intuitive, and high-performance web application while reducing customer uncertainty when purchasing clothing online.

The application follows a decoupled architecture, separating the Frontend from the Backend. The Frontend is developed using Next.js 15, React, and TypeScript to provide a responsive user interface optimized for performance and search engine optimization (SEO). The Backend is built with Express.js, Prisma ORM, and MySQL to manage business logic, data persistence, and secure communication. Global state management is implemented using Zustand, while GSAP is used to deliver smooth and immersive user interface animations.

The project also adopts a modular architecture that simplifies maintenance, scalability, and future integration with a real AI provider through an interchangeable Provider pattern. The obtained results demonstrate the feasibility of a modern, scalable, and high-performance e-commerce solution that meets both technical and user experience requirements.

Keywords :
E-commerce – Next.js – Express.js – Prisma ORM – Artificial Intelligence – AI Try-On

الملخص

يقدم مشروع التخرج هذا تصميم وتطوير منصة MONO، وهي منصة تجارة إلكترونية حديثة متخصصة في بيع الملابس، مع دمج ميزة التجربة الافتراضية للملابس بالاعتماد على الذكاء الاصطناعي (AI Try-On). يهدف المشروع إلى تحسين تجربة التسوق الإلكتروني من خلال توفير واجهة استخدام أنيقة وسريعة وسهلة الاستعمال، مع المساهمة في تقليل تردد العملاء عند شراء الملابس عبر الإنترنت.

يعتمد المشروع على معمارية مفصولة (Decoupled Architecture) تفصل بين الواجهة الأمامية (Frontend) والخادم الخلفي (Backend). تم تطوير الواجهة الأمامية باستخدام Next.js 15 وReact وTypeScript لتوفير أداء مرتفع وتجربة استخدام سلسة ومتجاوبة مع مختلف الأجهزة، بالإضافة إلى تحسين الظهور في محركات البحث (SEO). أما الخادم الخلفي فقد تم تطويره باستخدام Express.js و Prisma ORM وMySQL لإدارة قواعد البيانات، وتنفيذ منطق الأعمال، وتأمين تبادل البيانات بين مختلف مكونات النظام. كما تم اعتماد Zustand لإدارة الحالة العامة للتطبيق و GSAP لتقديم رسوم متحركة احترافية تعزز تجربة المستخدم.

كما يعتمد المشروع على بنية معيارية (Modular Architecture) تسهل صيانة النظام وتطويره مستقبلاً، مع إمكانية دمج مزود حقيقي للذكاء الاصطناعي بسهولة بفضل نمط Provider Pattern. وقد أثبتت النتائج إمكانية بناء منصة تجارة إلكترونية حديثة وقابلة للتوسع تجمع بين الأداء العالي وجودة تجربة المستخدم.

الكلمات المفتاحية

التجارة الإلكترونية، Next.js، Express.js، Prisma ORM، الذكاء الاصطناعي، AI Try-On

Liste des acronymes et des abréviations

Acronyme	Signification
AI	Artificial Intelligence
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
GSAP	GreenSock Animation Platform
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JWT	JSON Web Token
ORM	Object-Relational Mapping
REST	Representational State Transfer
SEO	Search Engine Optimization
SQL	Structured Query Language
SSR	Server-Side Rendering
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience

Liste des figures

Liste des tableaux

Table des matières

Introduction Générale

Au cours des dernières années, le commerce électronique a connu une évolution considérable, transformant profondément les habitudes de consommation dans de nombreux secteurs, notamment celui de la mode. Grâce à la démocratisation des technologies numériques et à l'amélioration des infrastructures web, les consommateurs peuvent désormais accéder à une grande variété de produits à tout moment et depuis n'importe quel appareil. Toutefois, malgré cette évolution, l'achat de vêtements en ligne demeure confronté à plusieurs difficultés, principalement liées à l'absence d'une expérience d'essayage réelle avant la validation de l'achat.

En effet, l'impossibilité de visualiser avec précision le rendu d'un vêtement sur soi constitue l'une des principales causes d'hésitation des clients et contribue à l'augmentation du taux de retour des articles. Cette problématique représente un enjeu important pour les plateformes e-commerce, aussi bien sur le plan économique que sur celui de la satisfaction des utilisateurs. Dans un contexte où l'expérience utilisateur est devenue un facteur essentiel de différenciation, l'intégration de nouvelles technologies apparaît comme une solution pertinente pour répondre à ces défis.

C'est dans cette perspective qu'a été développé MONO, une plateforme e-commerce dédiée à la mode haut de gamme, combinant une identité visuelle minimaliste avec des technologies web modernes afin d'offrir une expérience d'achat fluide, performante et immersive. Au-delà des fonctionnalités classiques d'une boutique en ligne (catalogue, panier, authentification, gestion des commandes et tableau d'administration), le projet intègre un module d'essayage virtuel basé sur l'intelligence artificielle permettant aux utilisateurs de visualiser le rendu d'un vêtement sur leur propre photographie avant de finaliser leur achat.

Afin de répondre aux exigences de performance, de maintenabilité et d'évolutivité, une architecture logicielle découplée a été adoptée. L'application est composée d'un frontend développé avec Next.js, offrant une interface moderne et optimisée pour le référencement (SEO), et d'un backend indépendant développé avec Express.js, chargé de centraliser la logique métier et la gestion des données. La persistance des informations repose sur MySQL, tandis que Prisma ORM assure une interaction sécurisée et typée avec la base de données. L'état global de l'application est géré à l'aide de Zustand, et l'expérience utilisateur est enrichie grâce aux animations réalisées avec GSAP ainsi qu'à un système d'essayage virtuel conçu selon une architecture modulaire facilitant l'intégration de différents fournisseurs d'intelligence artificielle.

L'objectif principal de ce projet est de concevoir une solution e-commerce moderne, capable d'améliorer l'expérience d'achat en ligne tout en proposant une architecture robuste, modulaire et facilement extensible. Au-delà de la réalisation d'une application fonctionnelle, ce travail vise également à mettre en pratique les compétences acquises durant la formation en matière d'architecture logicielle, de développement Full-Stack, de conception de bases de données, de sécurité des applications web et d'intégration de services d'intelligence artificielle.

Le présent rapport est organisé en quatre chapitres. Le premier chapitre présente le contexte du projet, la problématique étudiée, les objectifs poursuivis ainsi que l'analyse des besoins. Le deuxième chapitre est consacré à la conception générale du système, notamment l'architecture logicielle, la modélisation de la base de données et les différents diagrammes UML. Le troisième chapitre décrit la réalisation technique de l'application en détaillant les technologies utilisées, l'architecture du frontend et du backend ainsi que les principales fonctionnalités développées. Enfin, le quatrième chapitre présente les tests réalisés, les résultats obtenus, une analyse critique du projet ainsi que les perspectives d'amélioration envisagées.

Les technologies utilisées reposent sur leurs documentations officielles **[1][2][3][4][5]**.

Chapitre 1 : Analyse et étude du projet

Introduction :

Ce premier chapitre présente le contexte général dans lequel s'inscrit le projet MONO. Il introduit les motivations ayant conduit à sa réalisation ainsi que les problématiques auxquelles il apporte une réponse. Ce chapitre expose également les objectifs du projet, l'étude des solutions existantes et l'analyse des besoins fonctionnels et non fonctionnels. Cette phase constitue une étape essentielle, car elle permet de définir les exigences du système avant d'entamer sa conception et son développement.

1.1 Présentation de l'organisme d'accueil

Institut Supérieur de Management, d'Administration et de Génie Informatique (ISMAGI)

L'Institut Supérieur de Management, d'Administration et de Génie Informatique (ISMAGI) est un établissement d'enseignement supérieur privé spécialisé dans la formation aux métiers du management, de l'informatique et des nouvelles technologies. L'institut a pour mission de former des profils capables de répondre aux besoins du marché de l'emploi grâce à une approche pédagogique alliant connaissances théoriques et compétences pratiques.

Dans le cadre de la Licence Professionnelle en Développement Web et Mobile, les étudiants sont amenés à réaliser un Projet de Fin d'Études (PFE) leur permettant de mettre en pratique les connaissances acquises tout au long de leur formation. Ce projet constitue une étape essentielle du cursus puisqu'il mobilise les compétences en analyse, conception, développement, gestion de bases de données, sécurité des applications web ainsi qu'en architecture logicielle.

Le projet MONO a été développé dans ce contexte académique avec pour objectif de concevoir une plateforme e-commerce moderne intégrant des technologies web récentes et un module d'essayage virtuel basé sur l'intelligence artificielle. Ce travail représente une synthèse des compétences techniques acquises durant la formation et illustre la capacité à concevoir une application Full-Stack répondant à des besoins réels.

Figure 1.1 : Logo de l'ISMAGI

Site officiel de l'ISMAGI

1.2 Contexte du projet

Au cours des dernières années, le commerce électronique a connu une croissance considérable grâce à l'évolution des technologies web et à la démocratisation des achats en ligne. Le secteur de la mode figure parmi les domaines les plus concernés par cette transformation numérique, offrant aux consommateurs un accès rapide à un large choix de produits sans contrainte géographique.

Cependant, malgré cette évolution, les plateformes e-commerce rencontrent encore plusieurs défis. L'un des principaux problèmes réside dans l'impossibilité pour les clients d'essayer virtuellement les vêtements avant leur achat. Cette limitation engendre une incertitude concernant la taille, le style ou le rendu final des articles, ce qui augmente le taux de retour des commandes et impacte négativement l'expérience utilisateur.

Parallèlement, les progrès récents de l'intelligence artificielle ont permis l'émergence de nouvelles solutions capables d'améliorer l'expérience d'achat en ligne. Les technologies de Virtual Try-On offrent désormais la possibilité de générer une prévisualisation réaliste d'un vêtement porté par un utilisateur à partir d'une simple photographie.

C'est dans ce contexte qu'a été conçu le projet MONO, une plateforme e-commerce orientée mode premium intégrant un module d'essayage virtuel basé sur l'intelligence artificielle. L'objectif est de proposer une expérience utilisateur moderne, immersive et performante tout en s'appuyant sur une architecture Full-Stack robuste permettant une évolution future de la plateforme.

1.3 Problématique

Le développement rapide du commerce électronique a considérablement facilité l'achat de vêtements en ligne. Toutefois, malgré cette évolution, plusieurs difficultés continuent d'affecter l'expérience des consommateurs ainsi que la rentabilité des plateformes e-commerce.

L'un des principaux problèmes réside dans l'impossibilité pour les clients de visualiser avec précision le rendu d'un vêtement avant son achat. Les photographies classiques ne permettent pas toujours d'apprécier l'apparence réelle d'un article lorsqu'il est porté, ce qui crée une incertitude importante au moment de la décision d'achat.

Cette situation entraîne plusieurs conséquences, notamment une augmentation des retours produits, une diminution de la satisfaction client ainsi qu'une perte financière pour les entreprises. De plus, de nombreuses plateformes e-commerce proposent encore une expérience utilisateur standardisée, sans personnalisation ni fonctionnalités interactives permettant d'accompagner efficacement le client.

Face à ces constats, il apparaît nécessaire de développer une solution capable d'améliorer l'expérience d'achat tout en réduisant les limites des plateformes traditionnelles.
Le projet **MONO** répond à cette problématique en proposant une plateforme e-commerce moderne combinant une architecture Full-Stack performante avec un système d'essayage virtuel basé sur l'intelligence artificielle. Cette approche permet aux utilisateurs d'obtenir une prévisualisation plus réaliste des vêtements avant leur achat, améliorant ainsi la confiance, la satisfaction et la qualité de l'expérience utilisateur.

1.4 Objectifs du projet

L'objectif principal du projet **MONO** est de concevoir et de développer une plateforme e-commerce moderne dédiée au secteur de la mode, offrant une expérience utilisateur fluide, immersive et sécurisée grâce à l'intégration des technologies web récentes et de l'intelligence artificielle.
Pour atteindre cet objectif, plusieurs objectifs spécifiques ont été définis :

- Concevoir une architecture Full-Stack modulaire et évolutive.
- Développer une interface utilisateur moderne inspirée des plateformes premium.
- Mettre en place une authentification sécurisée basée sur JWT.
- Gérer le catalogue des produits, des collections, des variantes et des commandes.
- Développer un tableau de bord d'administration permettant la gestion complète de la plateforme.
- Intégrer un module d'essayage virtuel reposant sur l'intelligence artificielle.
- Concevoir une base de données relationnelle optimisée avec Prisma ORM et MySQL.
- Garantir la sécurité, les performances et la maintenabilité de l'application.

1.5 Étude des solutions existantes

Avant de concevoir la plateforme MONO, une étude des principales solutions e-commerce spécialisées dans le domaine de la mode a été réalisée afin d'identifier leurs fonctionnalités, leurs points forts ainsi que leurs limites.

Parmi les plateformes étudiées figurent **Zara**, **H&M**, **ASOS** et **Farfetch**, reconnues comme des références du commerce électronique dans l'industrie de la mode. Ces plateformes proposent généralement des fonctionnalités avancées telles que :

- un catalogue de produits organisé par catégories ;
- des filtres de recherche performants ;
- la gestion des favoris (Wishlist) ;
- un panier d'achat ;
- un système de paiement sécurisé ;
- le suivi des commandes.

Cependant, malgré la qualité de leurs interfaces et la richesse de leurs services, plusieurs limites demeurent. Dans la majorité des cas, le client ne peut toujours pas visualiser précisément le rendu réel d'un vêtement avant son achat. Cette limitation augmente le risque d'insatisfaction et favorise les retours de produits.

Par ailleurs, certaines grandes plateformes proposent des solutions d'essayage virtuel. Toutefois, ces fonctionnalités restent souvent limitées à certaines catégories de produits, nécessitent des équipements spécifiques ou reposent sur des services propriétaires difficilement accessibles dans un contexte académique.

Le projet **MONO** s'inspire des bonnes pratiques observées sur ces plateformes tout en proposant une architecture moderne et modulaire intégrant un module d'essayage virtuel basé sur l'intelligence artificielle. Cette approche permet d'améliorer l'expérience utilisateur tout en conservant une architecture évolutive facilitant l'intégration de nouveaux services à l'avenir.

Tableau 1.1 : Comparaison des solutions existantes

Tableau 1: Auteur

Critère	Zara	H&M	ASOS	Farfetch	MONO
Catalogue produits	✓	✓	✓	✓	✓
Authentification	✓	✓	✓	✓	✓
Gestion du panier	✓	✓	✓	✓	✓
Wishlist	✓	✓	✓	✓	✓
Dashboard administrateur	✗	✗	✗	✗	✓
Essayage virtuel IA	Partiel	✗	Partiel	Partiel	✓
Architecture modulaire	Non communiqué	Non communiqué	Non communiqué	Non communiqué	✓

1.6 Analyse des besoins

L'analyse des besoins constitue une étape essentielle dans le développement de toute application informatique. Elle permet d'identifier les fonctionnalités attendues par les utilisateurs ainsi que les exigences techniques et qualitatives que le système doit respecter.
Dans le cadre du projet MONO, les besoins ont été répartis en deux catégories : les besoins fonctionnels et les besoins non fonctionnels.

1.6.1 Besoins fonctionnels

Les besoins fonctionnels représentent les services que la plateforme doit offrir afin de répondre aux attentes des différents utilisateurs.
La plateforme MONO doit permettre de :

- Créer un compte utilisateur et s'authentifier de manière sécurisée.
- Consulter le catalogue des produits par catégories et collections.
- Rechercher un produit à l'aide d'un moteur de recherche.
- Filtrer les produits selon différents critères (catégorie, prix, taille, couleur, etc.).
- Consulter les détails complets d'un produit.
- Ajouter ou supprimer des produits du panier.
- Gérer une liste de favoris (Wishlist).
- Passer une commande.
- Consulter l'historique des commandes.
- Télécharger une photographie afin d'effectuer un essayage virtuel assisté par l'intelligence artificielle.
- Visualiser le résultat généré par le module Virtual Try-On.
- Gérer son profil utilisateur.

Pour l'administrateur, la plateforme doit également permettre :

- Gérer les utilisateurs.
- Gérer les produits.
- Gérer les catégories et les collections.
- Gérer les variantes (taille, couleur).
- Suivre les commandes.
- Consulter les statistiques générales de la plateforme.

1.6.2 Besoins non fonctionnels

En complément des fonctionnalités proposées, plusieurs exigences non fonctionnelles ont été définies afin de garantir la qualité globale de la plateforme.
Le système doit assurer :

- Une interface utilisateur moderne, intuitive et responsive.
- Une architecture logicielle modulaire facilitant la maintenance et l'évolution du projet.
- Une authentification sécurisée basée sur JWT.
- La protection des API grâce aux middlewares de sécurité (Helmet, CORS et Rate Limiting).
- Une base de données fiable assurant l'intégrité des informations.
- Des performances élevées grâce à Next.js, React Server Components et Prisma ORM.
- Une expérience utilisateur fluide grâce aux animations GSAP et au défilement Lenis.
- Une architecture évolutive permettant l'intégration future de nouveaux fournisseurs d'intelligence artificielle.
- Une compatibilité avec les principaux navigateurs modernes et différents formats d'écran.

Conclusion :

Dans ce chapitre, nous avons présenté le contexte général du projet, identifié la problématique, défini les objectifs poursuivis et analysé les besoins fonctionnels et non fonctionnels de la plateforme MONO. Cette étude a permis d'établir une vision claire des exigences du système et constitue une base solide pour la phase de conception présentée dans le chapitre suivant.

Chapitre 2 : Conception du système

Introduction :

La phase de conception constitue une étape fondamentale dans le cycle de développement d'une application informatique. Elle permet de transformer les besoins identifiés lors de la phase d'analyse en une architecture claire, cohérente et évolutive.
Dans le cadre du projet MONO, cette phase a consisté à définir l'architecture générale du système, à sélectionner les technologies les plus adaptées, à concevoir la base de données relationnelle ainsi qu'à modéliser le fonctionnement de l'application à l'aide des diagrammes UML.
Cette démarche a permis d'établir une base solide avant le développement des différentes fonctionnalités du projet.

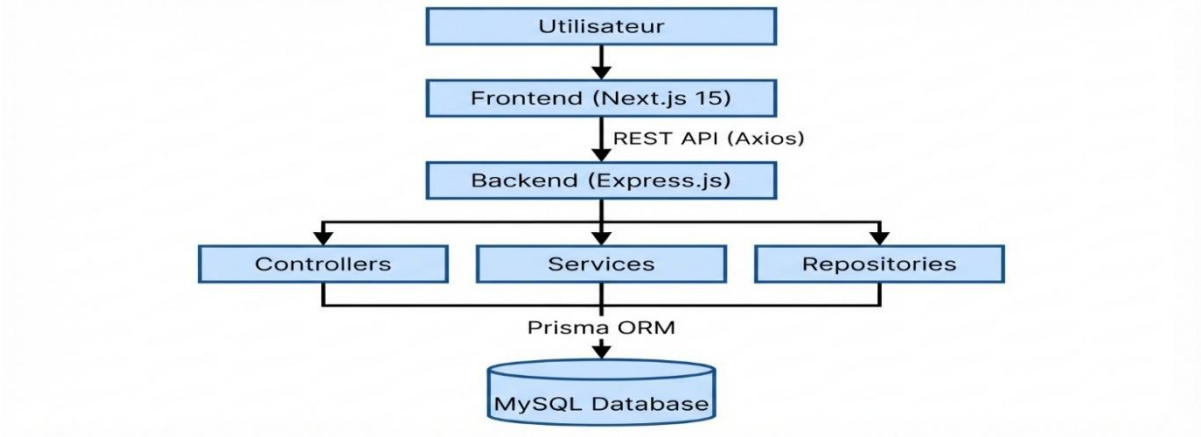
2.1 Architecture générale

L'application MONO repose sur une architecture Full-Stack découplée (Decoupled Architecture) composée de trois couches principales :

- une application Frontend développée avec Next.js 15 ;
- une API Backend développée avec Node.js, Express.js et Prisma ORM ;
- une base de données relationnelle MySQL.

La communication entre le Frontend et le Backend s'effectue exclusivement via des API REST sécurisées. Cette séparation permet de faciliter la maintenance, d'améliorer l'évolutivité du système et de rendre l'API réutilisable par d'autres clients, tels qu'une application mobile.
Le Backend est organisé selon une architecture en couches (Layered Architecture), séparant les responsabilités entre les Routes, les Controllers, les Services et les Repositories. Cette approche garantit une meilleure maintenabilité, favorise la réutilisation du code et simplifie les évolutions futures de la plateforme.

Figure 2.1 : Architecture générale de MONO



Source : Réalisation personnelle.

2.2 Choix technologiques

Le choix des technologies utilisées dans le développement de la plateforme MONO a été réalisé en tenant compte de plusieurs critères, notamment les performances, la maintenabilité, la sécurité, la modularité ainsi que la possibilité d'évolution du projet.
Les principales technologies retenues sont présentées ci-dessous.

Next.js 15

Le Frontend de MONO a été développé avec Next.js 15, un framework basé sur React offrant de nombreuses fonctionnalités avancées telles que le App Router, les React Server Components, le rendu hybride (SSR/CSR) ainsi que des optimisations automatiques des performances.
Ce choix permet d'améliorer l'expérience utilisateur, le référencement naturel (SEO) ainsi que la rapidité de chargement des pages.
Next.js offre également plusieurs optimisations natives améliorant les performances globales de l'application et le référencement naturel (SEO) **[1]**.

React

React est utilisé pour construire une interface utilisateur moderne basée sur une architecture en composants réutilisables.
Cette approche facilite la maintenance du code, améliore sa modularité et permet une meilleure organisation de l'application.
React facilite la conception d'interfaces utilisateur basées sur des composants réutilisables **[2]**.

TypeScript

Le projet utilise TypeScript afin d'assurer un typage statique du code.
Cette technologie permet de détecter les erreurs dès la phase de développement, améliore la lisibilité du code et facilite la maintenance de l'application.
Cette approche améliore également la robustesse et la maintenabilité du code **[11]**..

Tailwind CSS

Le design de l'interface repose sur Tailwind CSS, un framework CSS utilitaire permettant de créer rapidement des interfaces modernes tout en conservant un code propre et facilement personnalisable.
Tailwind CSS permet de développer rapidement des interfaces modernes tout en conservant une excellente flexibilité **[7]**.

GSAP

Les animations de l'interface ont été développées avec GSAP (GreenSock Animation Platform).
Cette bibliothèque offre des animations fluides, performantes et parfaitement adaptées à une interface premium, tout en permettant un contrôle précis des séquences d'animation.
Les animations réalisées avec GSAP contribuent à améliorer l'expérience utilisateur tout en conservant de bonnes performances **[9]**

Zustand

La gestion de l'état global de l'application est assurée par Zustand.
Cette bibliothèque légère permet de centraliser les données partagées (authentification, panier, interface utilisateur et module IA) tout en limitant les re-rendus inutiles des composants**.**
Zustand constitue une solution légère et performante pour la gestion de l'état global des applications React **[8]**.

Tableau 2.1 : Technologies utilisées

TECH STACK : PROJET INNOVANT

Utilisation	Technologie
Développement Frontend	Next.js 15
Interface utilisateur	React
Typage statique	TypeScript
Mise en forme	Tailwind CSS
Animations	GSAP
Gestion de l'état global	Zustand

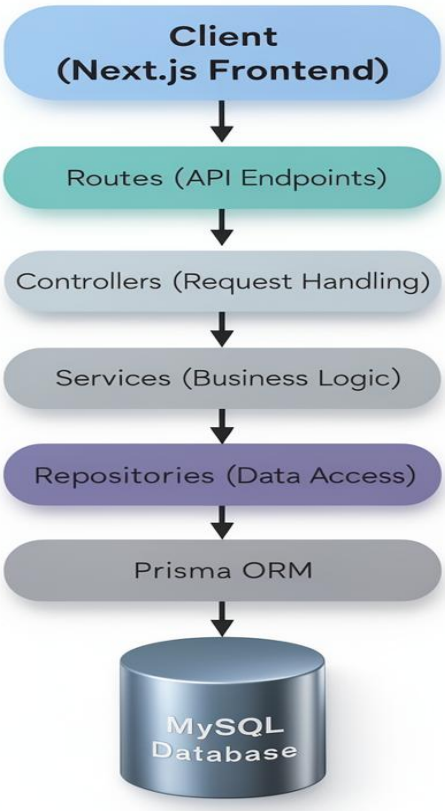
2.3 Architecture détaillée

Cette architecture est composée des couches suivantes :

- **Routes** : définissent les différents points d'entrée de l'API REST.
- **Controllers** : reçoivent les requêtes HTTP, valident les données puis délèguent le traitement.
- **Services** : contiennent l'ensemble de la logique métier de l'application.
- **Repositories** : assurent l'accès aux données via Prisma ORM.
- **Prisma ORM** : traduit les opérations JavaScript en requêtes SQL.
- **MySQL** : stocke les données de manière persistante.

Cette séparation respecte le principe de **Separation of Concerns**, ce qui facilite la maintenance, améliore la lisibilité du code et permet de faire évoluer l'application sans impacter les autres couches.
Par exemple, une modification de la base de données n'affecte que la couche Repository, tandis que les règles métier restent centralisées dans les Services.

Figure 2.2 : Architecture interne du Backend (Layered Architecture)



Source : Réalisation personnelle.

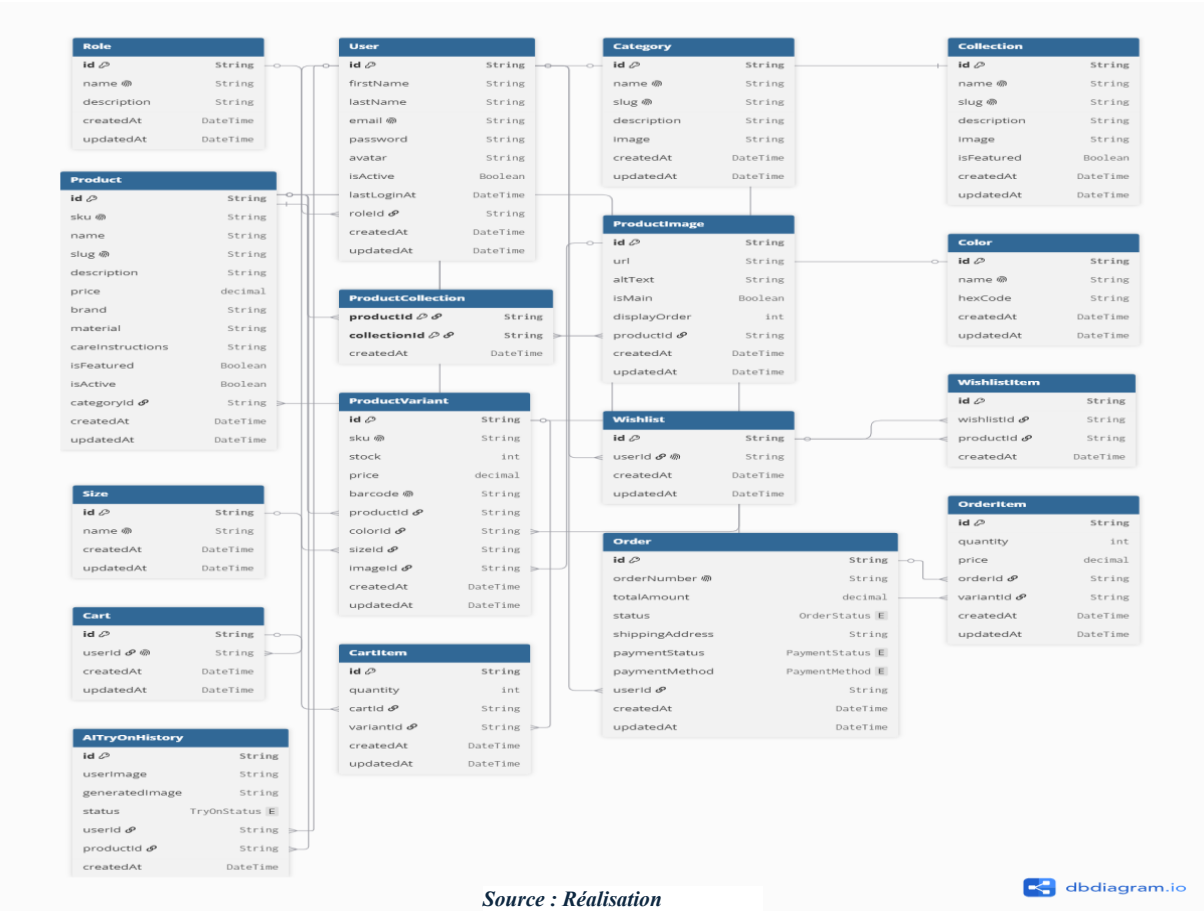
2.4 Conception de la base de données

La base de données constitue le cœur de la plateforme MONO puisqu'elle assure le stockage et la gestion de l'ensemble des informations manipulées par l'application.
Afin de garantir une structure fiable, évolutive et facilement maintenable, une base de données relationnelle MySQL a été retenue. Son accès est assuré par Prisma ORM, qui simplifie les opérations de manipulation des données tout en offrant un haut niveau de sécurité grâce au typage automatique et aux migrations.
Le schéma de la base de données a été conçu selon les principes de normalisation afin de limiter la redondance des données et de garantir leur cohérence.
Les principales entités modélisées sont :

- User : informations des utilisateurs et authentification.
- Product : catalogue des produits.
- Variant : tailles, couleurs et stock de chaque produit.
- Collection : regroupement des produits.
- Category : classification des produits.
- Cart et CartItem : gestion du panier.
- Wishlist : liste des favoris.
- Order et OrderItem : gestion des commandes.
- Address : adresses de livraison.
- TryOnJob : historique des essais virtuels par intelligence artificielle.

L'ensemble de ces tables est relié par des clés primaires et étrangères garantissant l'intégrité référentielle de la base de données.
Les données sont stockées dans une base relationnelle MySQL [6].

Figure 2.3 : Schéma relationnel de la base de données (ERD Prisma)



Source : Réalisation

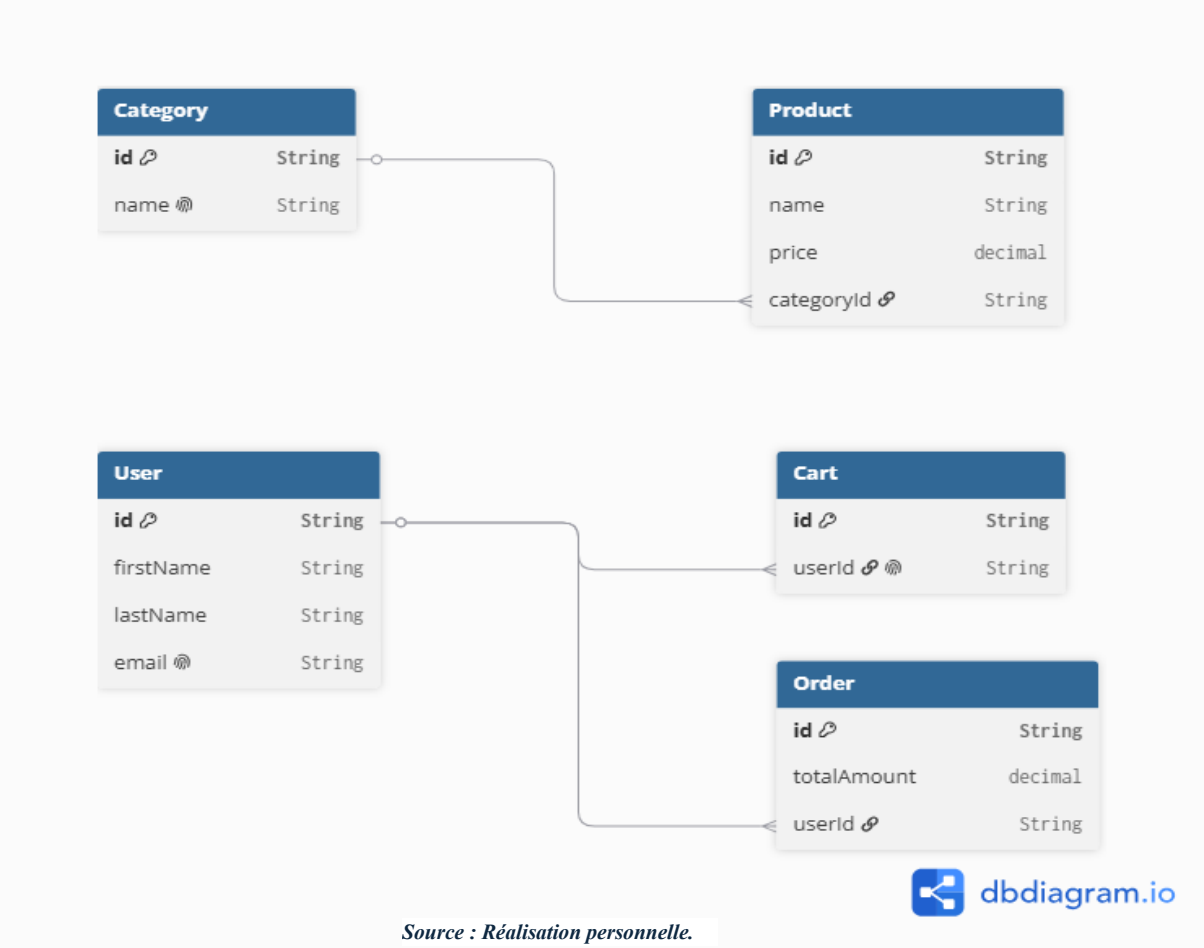
2.5 Relations entre les entités

Le modèle relationnel repose principalement sur des relations de type One-to-Many et Many-to-One.
Les relations les plus importantes sont :

- Un User peut posséder plusieurs Orders.
- Un User peut posséder plusieurs CartItems.
- Un User peut enregistrer plusieurs produits dans sa Wishlist.
- Une Collection contient plusieurs Products.
- Un Product possède plusieurs Variants.
- Un Order contient plusieurs OrderItems.
- Chaque OrderItem est associé à un seul Variant.
- Chaque TryOnJob est associé à un utilisateur et à un produit.

Cette modélisation permet de représenter fidèlement les différents processus métiers de la plateforme tout en assurant la cohérence des données.

Figure 2.4 : Relations principales entre les entités



Source : Réalisation personnelle.

2.6 Diagrammes UML

La modélisation UML a été réalisée afin de représenter le fonctionnement de la plateforme MONO sous différents points de vue. Contrairement au schéma de la base de données qui décrit la structure des informations, les diagrammes UML permettent de visualiser les interactions entre les utilisateurs, les composants logiciels ainsi que les différents processus métiers de l'application.
Ces diagrammes ont constitué une étape importante de la phase de conception puisqu'ils ont permis de mieux organiser le développement et de valider les différents scénarios fonctionnels avant l'implémentation.

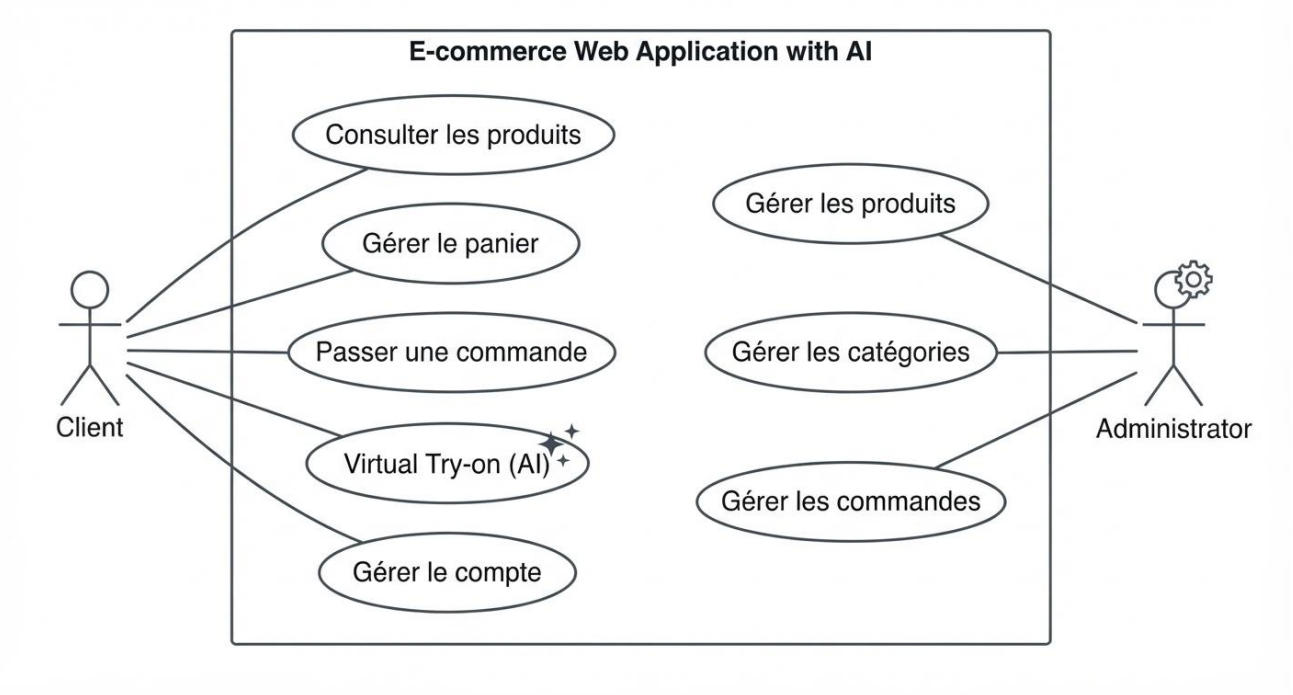
2.6.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation présente les principales fonctionnalités offertes par la plateforme ainsi que les interactions entre les différents acteurs.
Deux acteurs principaux ont été identifiés :

- Le Client, qui consulte les produits, crée un compte, s'authentifie, gère son panier, passe une commande et utilise le module d'essayage virtuel.
- L'Administrateur, qui gère les produits, les catégories, les collections, les commandes ainsi que les utilisateurs.

Ce diagramme offre une vision globale des fonctionnalités disponibles au sein de l'application.

Figure 2.5 : Diagramme de cas d'utilisation

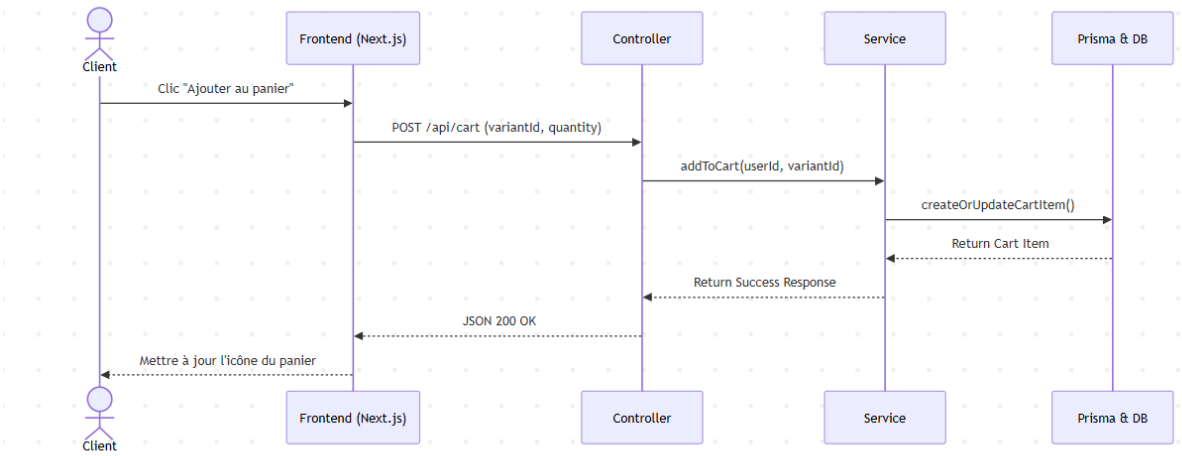


Source : Réalisation personnelle.

2.6.2 Diagramme de séquence

Le diagramme de séquence représente les échanges entre les différents composants du système lors de l'exécution d'une fonctionnalité.
Dans le projet MONO, il illustre le scénario d'ajout d'un produit au panier. La requête est transmise du Frontend vers le Backend à travers l'API REST, puis traitée successivement par les couches Controller, Service, Repository et Prisma avant d'être enregistrée dans la base de données.
Ce diagramme met en évidence la séparation des responsabilités adoptée dans l'architecture du projet.

Figure 2.6 : Diagramme de séquence

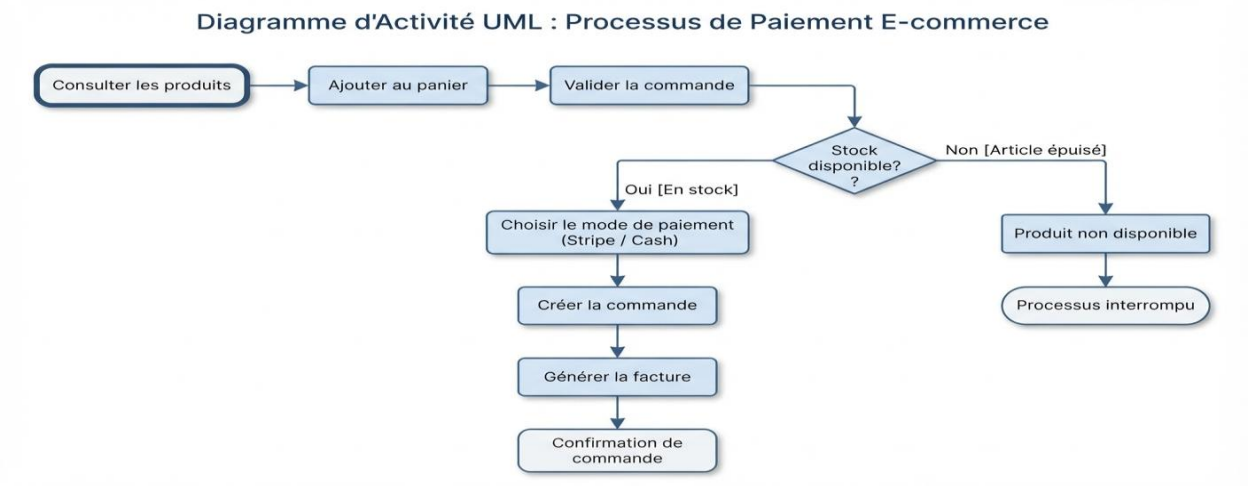


Source : Réalisation personnelle.

2.6.3 Diagramme d'activité

Le diagramme d'activité décrit le déroulement logique d'un processus métier.
Dans notre projet, il représente les différentes étapes suivies par un utilisateur depuis la consultation des produits jusqu'à la validation d'une commande. Il met également en évidence les décisions prises par le système, notamment la vérification du stock et la création de la commande.
Ce diagramme facilite la compréhension du comportement global de l'application.

Figure 2.7 : Diagramme d'activité

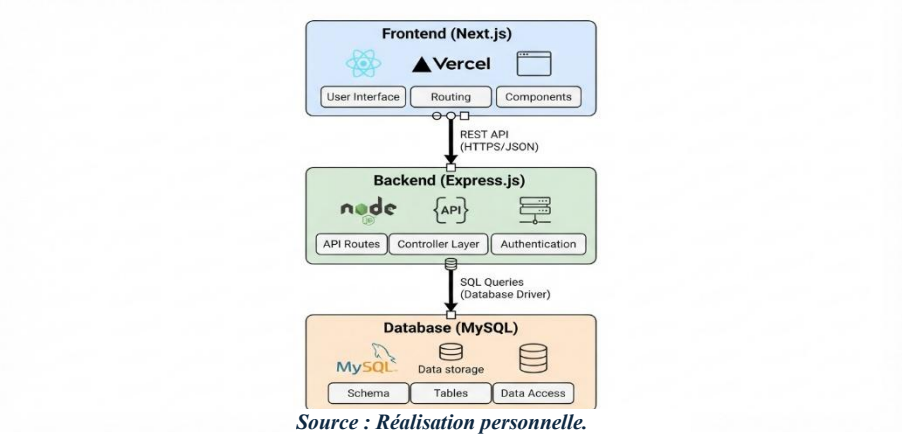


Source : Réalisation personnelle.

2.6.4 Diagramme de composants

Le diagramme de composants illustre l'organisation logicielle de la plateforme MONO.
Il met en évidence les principaux composants de l'application, notamment le Frontend développé avec Next.js, le Backend développé avec Express.js ainsi que la base de données MySQL. Il représente également les échanges entre ces différents composants via les API REST.
Cette représentation permet de mieux comprendre la structure modulaire adoptée pour le développement du projet.

Figure 2.8 : Diagramme de composants

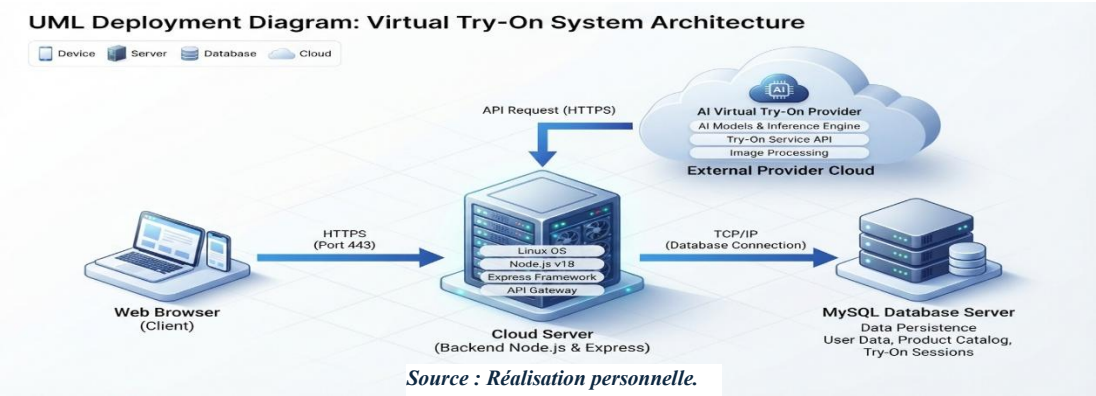


Source : Réalisation personnelle.

2.6.5 Diagramme de déploiement

Le diagramme de déploiement présente l'architecture physique de la solution.
Il montre les différents nœuds composant le système, notamment le navigateur Web, le serveur hébergeant l'application Backend, la base de données MySQL ainsi que le fournisseur d'intelligence artificielle utilisé pour le module Virtual Try-On.
Cette architecture favorise une meilleure évolutivité du système et facilite son déploiement dans un environnement de production.

Figure 2.9 : Diagramme de déploiement



Source : Réalisation personnelle.

Conclusion du chapitre

Au cours de ce chapitre, nous avons présenté la conception globale de la plateforme MONO en détaillant son architecture générale, les technologies retenues, la conception de la base de données ainsi que les différents diagrammes UML. Cette phase de conception a permis de définir une architecture modulaire, cohérente et évolutive, constituant une base solide pour l'implémentation de l'application.
Le chapitre suivant sera consacré à la réalisation du projet, en présentant les différentes étapes d'implémentation du Frontend, du Backend ainsi que les principales fonctionnalités développées.

Chapitre 3 : Réalisation et implémentation
Introduction

Introduction :

La phase de réalisation représente l'étape de concrétisation de l'ensemble des travaux de conception présentés dans le chapitre précédent. Elle consiste à transformer les modèles conceptuels et architecturaux en une application web Full-Stack fonctionnelle répondant aux objectifs définis lors de l'analyse.
Dans le cadre du projet MONO, le développement a été réalisé selon une architecture découplée (Decoupled Architecture) séparant totalement le Frontend du Backend. Cette approche permet d'améliorer la maintenabilité, la modularité ainsi que l'évolutivité de l'application tout en facilitant la réutilisation des services par d'autres clients.
Le Frontend a été développé avec **Next.js 15**, **React** et **TypeScript** afin de proposer une interface moderne, performante et responsive. Le Backend repose sur **Node.js**, **Express.js** et **Prisma ORM**, organisés selon une architecture en couches (Layered Architecture) composée des Routes, Controllers, Services et Repositories.
Ce chapitre présente les différentes étapes de réalisation de la plateforme, les principaux choix techniques effectués ainsi que les fonctionnalités développées au cours du projet.

3.1 Architecture globale de l'application

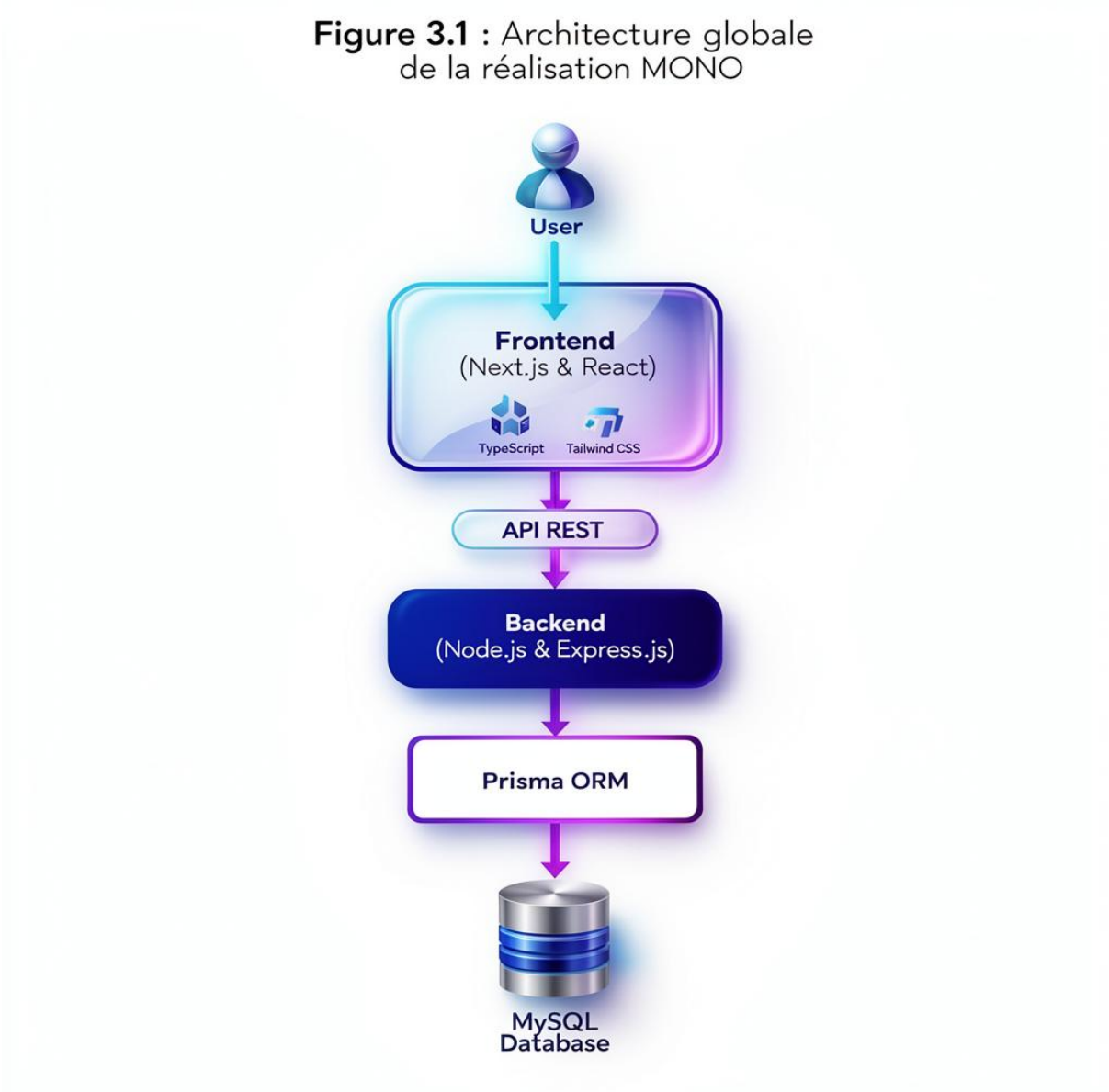
L'application MONO repose sur une architecture Full-Stack découplée dans laquelle chaque partie du système possède une responsabilité bien définie.
L'interface utilisateur est entièrement développée avec Next.js et constitue le point d'entrée de l'application. Elle est responsable de l'affichage des données, des interactions utilisateur ainsi que de la gestion de l'expérience utilisateur.
Toutes les opérations métiers transitent par une API REST développée avec Express.js. Cette API centralise les traitements métier, la gestion des utilisateurs, des commandes, du catalogue ainsi que des fonctionnalités d'intelligence artificielle.
Les données sont stockées dans une base MySQL accessible exclusivement via Prisma ORM, garantissant un accès sécurisé et typé aux informations.

Cette séparation entre les différentes couches présente plusieurs avantages :

- une meilleure organisation du code ;
- une maintenance simplifiée ;
- une évolutivité facilitée ;
- la possibilité de réutiliser la même API pour une future application mobile.

Le flux global de communication est le suivant :

Figure 3.1 : Architecture globale de la réalisation



Source : Réalisation personnelle.

3.2 Implémentation du Frontend

Le Frontend de MONO constitue la couche de présentation de l'application. Son rôle principal est d'offrir une interface moderne, intuitive et performante permettant aux utilisateurs d'interagir avec les différentes fonctionnalités de la plateforme.

Le développement de cette partie a été réalisé avec **Next.js 15**, **React**, **TypeScript** et **Tailwind CSS**. Ce choix permet de bénéficier d'une architecture moderne basée sur l'App Router, d'un typage statique améliorant la qualité du code ainsi que d'une interface responsive adaptée aux différents appareils.

L'architecture du Frontend a été conçue selon une approche modulaire afin de faciliter la maintenance, la réutilisation des composants et l'évolution future de l'application.

3.2.1 Organisation du Frontend

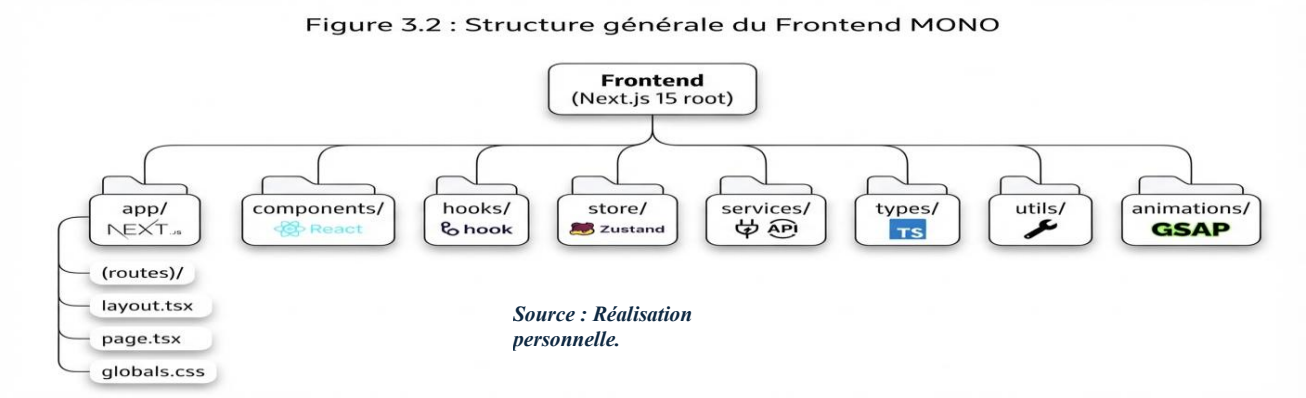
Le projet est organisé selon une structure modulaire où chaque dossier possède une responsabilité bien définie.

L'application est principalement composée des répertoires suivants :

- **app/** : gestion des routes grâce à l'App Router de Next.js.
- **components/** : composants réutilisables de l'interface utilisateur.
- **hooks/** : hooks personnalisés regroupant la logique réutilisable.
- **store/** : gestion de l'état global avec Zustand.
- **services/** : communication avec l'API REST.
- **types/** : définitions des types TypeScript.
- **utils/** : fonctions utilitaires.
- **animations/** : gestion des animations GSAP.

Cette organisation permet d'éviter la duplication du code et facilite l'ajout de nouvelles fonctionnalités.

Figure 3.2 : Structure générale du Frontend



Source : Réalisation personnelle.

3.2.2 App Router de Next.js

MONO utilise le nouvel **App Router** introduit par Next.js, permettant une organisation plus claire des routes de l'application. Cette architecture repose sur les recommandations officielles de Next.js concernant l'App Router et les React Server Components **[1]**.

Chaque dossier placé dans le répertoire **app/** représente automatiquement une route accessible par l'utilisateur.

Par exemple :

Dossier	**Route générée**
app/page.tsx	/
app/products/page.tsx	/products
app/product/[slug]/page.tsx	/product/:slug
app/cart/page.tsx	/cart
app/checkout/page.tsx	/checkout
app/account/page.tsx	/account

Cette approche simplifie considérablement l'organisation du projet tout en facilitant la maintenance des différentes pages.

L'App Router permet également de distinguer les **Server Components**, utilisés pour optimiser le chargement des données et le référencement (SEO), des **Client Components**, destinés aux fonctionnalités interactives telles que les animations, la gestion du panier ou les formulaires.

3.2.3 Architecture des composants

Afin d'assurer une meilleure réutilisation du code, l'interface utilisateur est construite à partir de composants indépendants.

Chaque composant possède une responsabilité unique et peut être utilisé dans plusieurs pages de l'application.

Cette approche présente plusieurs avantages :

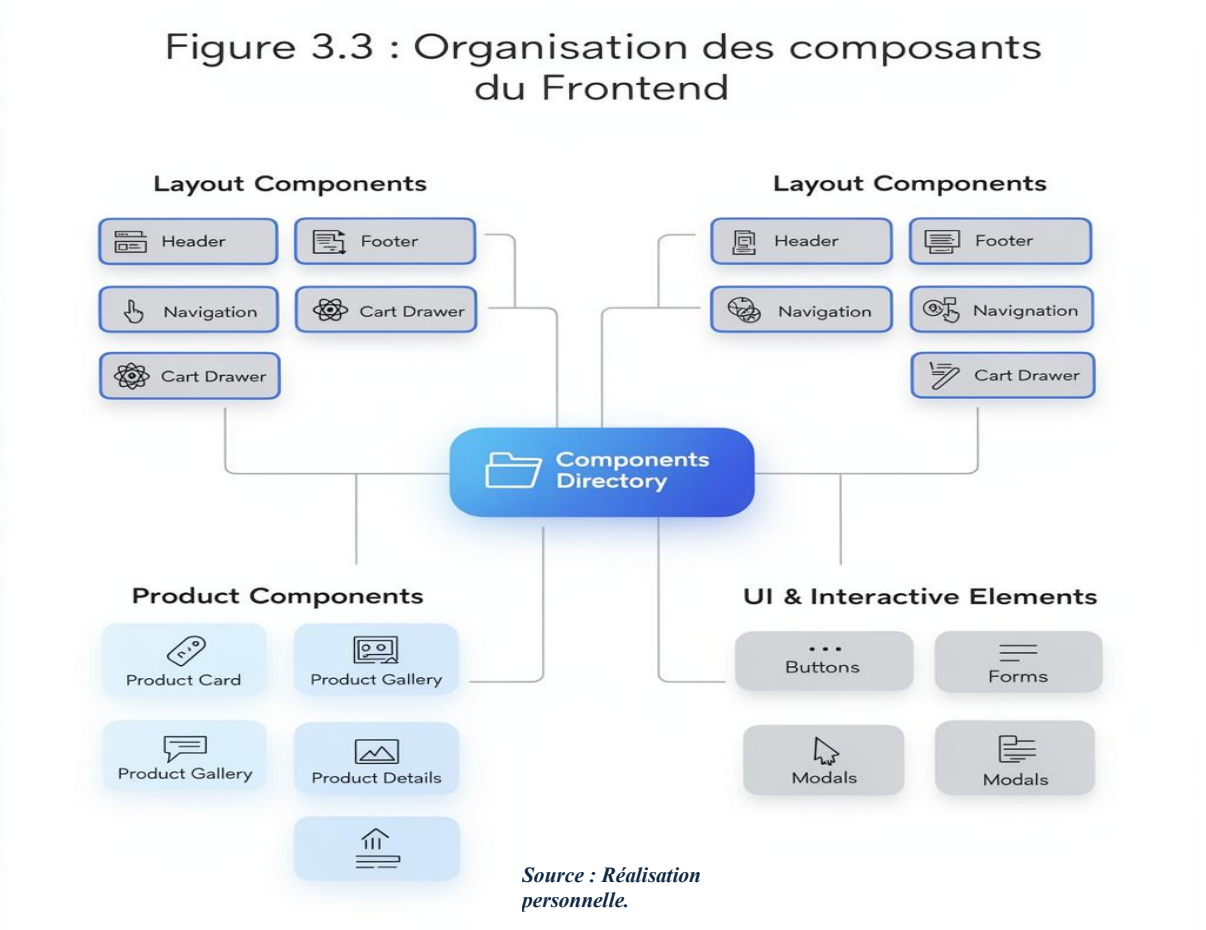
- réduction de la duplication du code ;
- meilleure lisibilité ;
- maintenance simplifiée ;
- évolution plus rapide de l'interface.

Les composants développés regroupent notamment :

- Header
- Footer
- Product Card
- Product Gallery
- Product Details
- Cart Drawer
- Navigation
- Buttons
- Forms
- Modals

L'ensemble de ces composants permet de construire les différentes interfaces de MONO de manière cohérente tout en respectant l'identité graphique de la plateforme.

Figure 3.3 : Organisation des composants du Frontend



Source : Réalisation personnelle.

3.2.4 Gestion de l'état global avec Zustand

Afin d'assurer une gestion centralisée des données de l'application, MONO utilise la bibliothèque **Zustand** comme solution de gestion d'état global (*State Management*). Ce choix permet de partager efficacement les informations entre les différents composants tout en limitant la complexité du code.

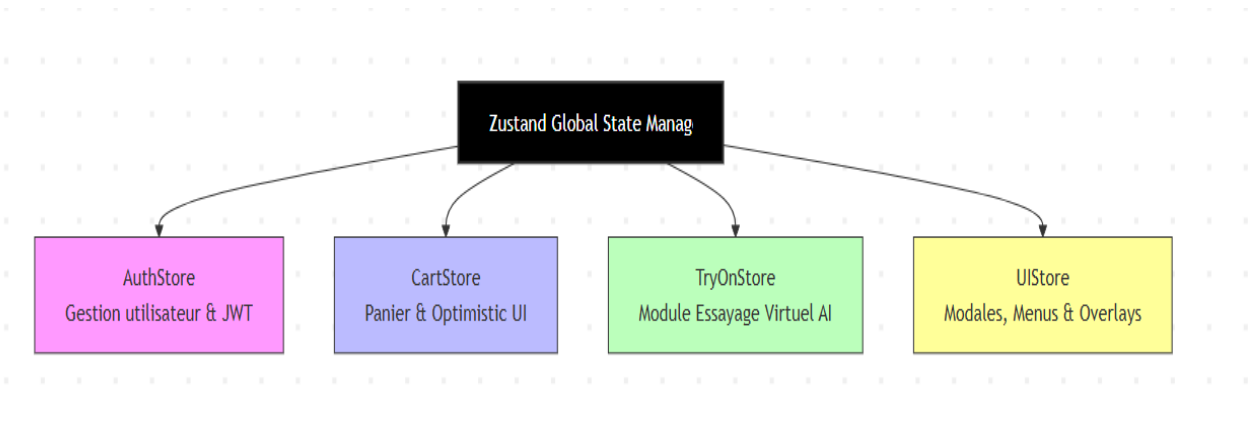
Contrairement à Redux, Zustand nécessite très peu de configuration (*Boilerplate*), offre de bonnes performances et facilite la maintenance d'une architecture modulaire.

L'application repose sur quatre Stores principaux, chacun étant dédié à une fonctionnalité spécifique.

Store	Responsabilité
AuthStore	Gestion de l'authentification et des informations de l'utilisateur
CartStore	Gestion du panier et synchronisation avec le Backend
TryOnStore	Gestion du module d'essayage virtuel par intelligence artificielle
UIStore	Gestion des états de l'interface (modales, menus, overlays, etc.)

Le **CartStore** implémente également le principe d'**Optimistic UI**. Lorsqu'un utilisateur ajoute un produit au panier, l'interface est mise à jour immédiatement afin d'offrir une meilleure expérience utilisateur, tandis que la synchronisation avec le Backend est effectuée en arrière-plan.

Figure 3.4 : Organisation des Stores Zustand



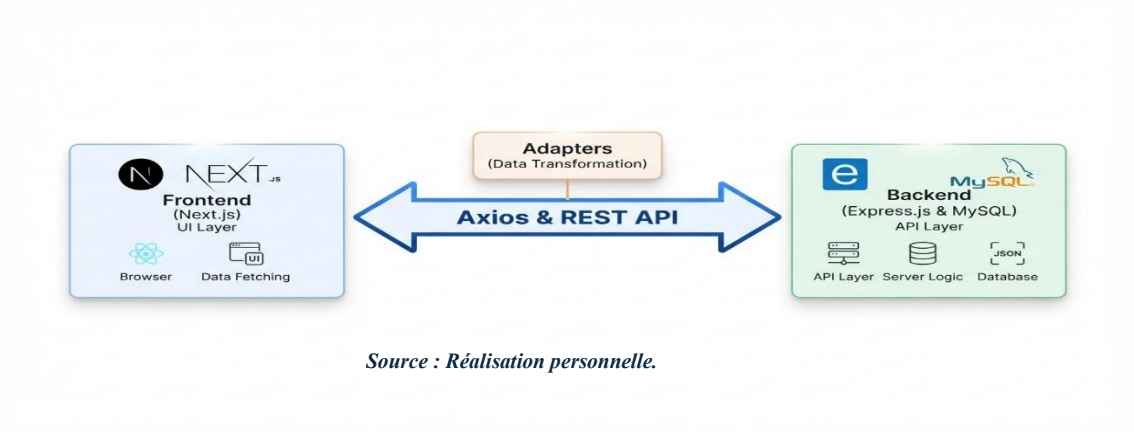
Source : Réalisation personnelle.

3.2.5 Communication entre le Frontend et le Backend

La communication entre le Frontend et le Backend est assurée exclusivement à travers des **API REST**. Les requêtes HTTP sont envoyées à l'aide de la bibliothèque **Axios**, qui facilite les échanges de données entre les différentes couches de l'application.
Afin de limiter le couplage entre le Frontend et le Backend, MONO utilise des **Adapters** chargés de transformer les données reçues depuis l'API en un format uniforme avant leur utilisation par les composants React. Cette approche améliore la maintenabilité du projet et simplifie l'évolution du Backend sans impacter directement l'interface utilisateur.
Les principales opérations réalisées via ces API concernent notamment l'authentification, la gestion du catalogue des produits, le panier, les commandes ainsi que le module d'essayage virtuel.

Les échanges HTTP respectent les principes des API REST décrits dans la documentation officielle de MDN **[11]**.

Figure 3.5 : Flux de communication entre le Frontend et le Backend

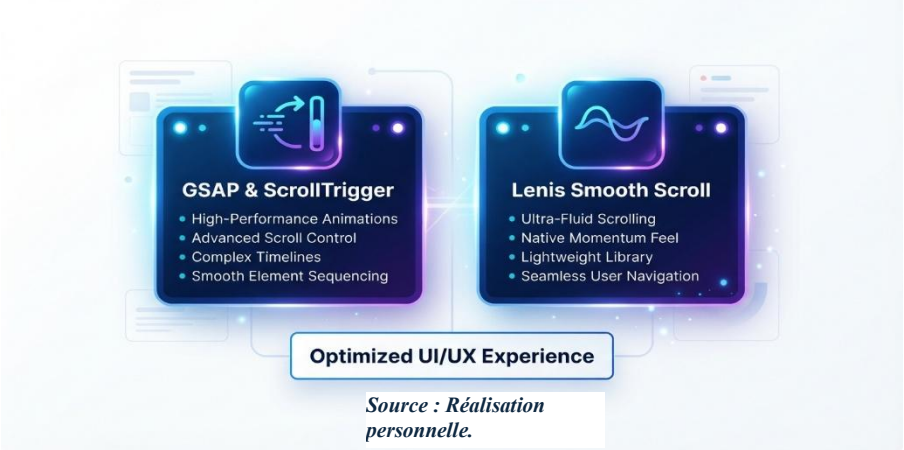


Source : Réalisation personnelle.

3.2.6 Animations et expérience utilisateur

L'expérience utilisateur constitue un élément central de la plateforme MONO. Afin d'offrir une navigation fluide et immersive, plusieurs bibliothèques spécialisées ont été intégrées.
Les animations de l'interface sont développées avec **GSAP (GreenSock Animation Platform)** et son plugin **ScrollTrigger**, permettant de déclencher des animations en fonction du défilement de la page.
Le défilement fluide (*Smooth Scroll*) est assuré par **Lenis**, offrant une navigation plus naturelle et améliorant la perception de fluidité de l'application.
L'ensemble de ces animations est implémenté de manière optimisée afin de préserver les performances de l'application tout en renforçant l'identité visuelle de la plateforme.

Figure 3.6 : Animations GSAP et Smooth Scroll avec Lenis

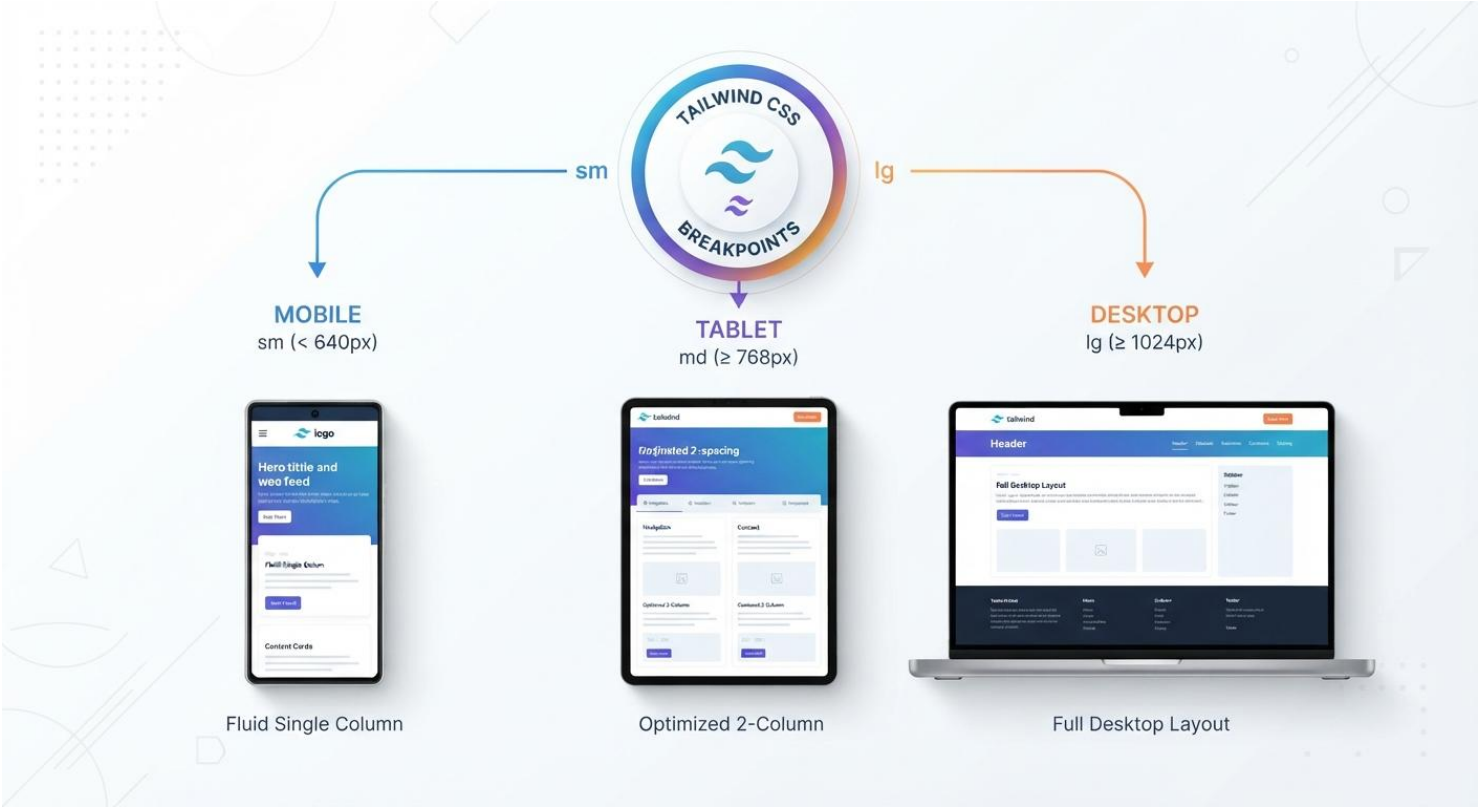


Source : Réalisation personnelle.

3.2.7 Responsive Design

L'interface de MONO a été développée selon une approche **Responsive Design** afin de garantir une expérience utilisateur optimale sur les différents types d'écrans.
Grâce à **Tailwind CSS**, l'application adapte automatiquement sa mise en page aux appareils mobiles, tablettes et ordinateurs de bureau à l'aide de points de rupture (*Breakpoints*).
Cette approche permet d'assurer une navigation cohérente, une bonne lisibilité des contenus ainsi qu'une utilisation confortable des différentes fonctionnalités, indépendamment de la taille de l'écran.

Figure 3.7 : Adaptation de l'interface sur Mobile, Tablette et Desktop



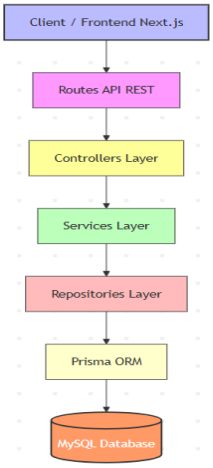
Source : Réalisation personnelle.

3.3 Implémentation du Backend

Le Backend de MONO constitue le cœur fonctionnel de la plateforme. Il est développé avec **Node.js** et **Express.js**, selon une architecture modulaire en couches (*Layered Architecture*), permettant de séparer les responsabilités entre les différents composants de l'application.
Cette organisation facilite la maintenance, améliore la réutilisation du code et permet de faire évoluer le système sans impacter les autres couches de l'application.
Le Backend est responsable de la gestion de l'authentification, des produits, des commandes, du panier, des collections ainsi que de la communication avec le module d'essayage virtuel basé sur l'intelligence artificielle.
Le Backend repose sur Express.js pour la création d'une API REST légère et performante [4].

Figure 3.8 : Architecture générale du Backend

(graph TD)



Source : Réalisation personnelle.

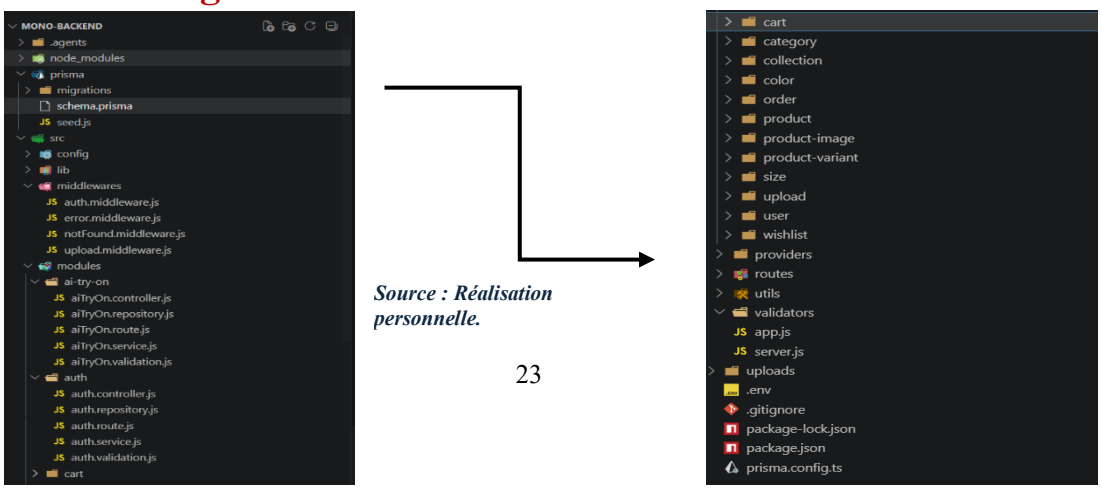
3.3.1 Architecture en couches

Afin d'assurer une meilleure organisation du projet, le Backend est structuré selon une architecture en couches (*Layered Architecture*). Chaque couche possède une responsabilité bien définie.

Couche	Responsabilité
Routes	Définition des points d'entrée de l'API REST
Controllers	Réception des requêtes HTTP et envoi des réponses
Services	Implémentation de la logique métier
Repositories	Accès aux données via Prisma ORM
Database	Stockage des données dans MySQL

Cette séparation permet de respecter le principe de **Single Responsibility**, de limiter le couplage entre les différentes couches et de simplifier les opérations de maintenance ou d'évolution du système.

Figure 3.9 : Organisation des couches du Backend



Source : Réalisation personnelle.

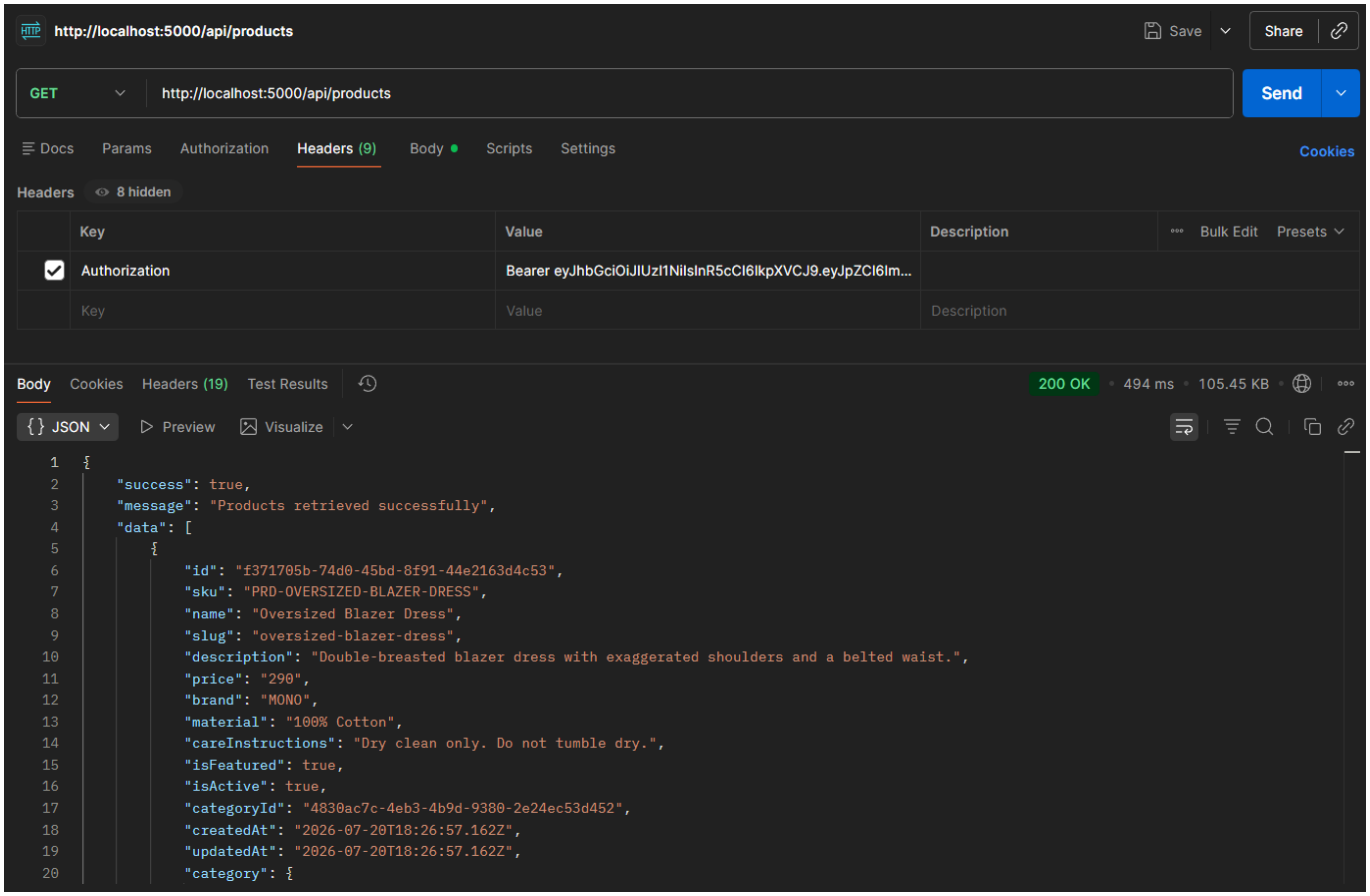
3.3.2 Développement des API REST

L'ensemble des fonctionnalités de MONO est exposé sous forme d'API REST. Chaque ressource possède son propre ensemble de routes permettant de réaliser les opérations CRUD (*Create, Read, Update, Delete*).
Les principales API développées concernent :

- Authentification des utilisateurs ;
- Gestion des produits ;
- Gestion des catégories et collections ;
- Gestion du panier ;
- Gestion des favoris ;
- Gestion des commandes ;
- Tableau de bord administrateur ;
- Module d'essayage virtuel par IA.

Cette architecture permet une communication claire entre le Frontend et le Backend tout en facilitant l'intégration d'autres clients, tels qu'une application mobile.

Figure 3.10 : Exemple des routes de l'API REST



Source : Réalisation personnelle.

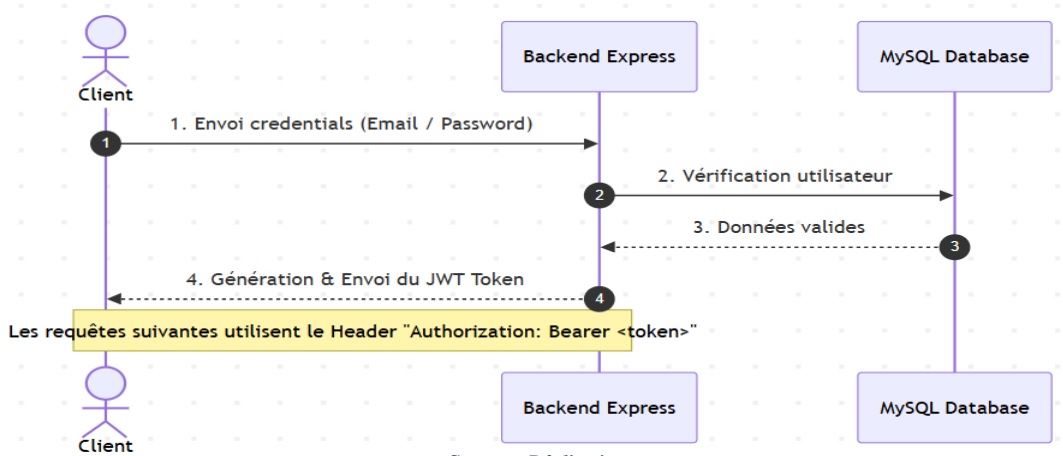
3.3.3 Authentification et sécurisation de l'application

La sécurité constitue un élément essentiel de la plateforme MONO. Afin de protéger les différentes ressources de l'application, un système d'authentification basé sur les **JSON Web Tokens (JWT)** a été mis en place. Lorsqu'un utilisateur s'authentifie avec succès, le serveur génère un jeton JWT contenant les informations nécessaires à son identification. Ce jeton est ensuite utilisé pour accéder aux différentes routes sécurisées de l'application.
Le Backend intègre également plusieurs mécanismes de protection afin de renforcer la sécurité de l'API :

- authentification par JWT ;
- contrôle des rôles utilisateurs (Administrateur / Client) ;
- validation des données reçues ;
- protection des en-têtes HTTP grâce à **Helmet** ;
- limitation du nombre de requêtes (**Rate Limiter**) afin de réduire les risques d'attaques.

Ces mécanismes permettent de garantir la confidentialité des données ainsi que la protection des fonctionnalités sensibles de la plateforme.
Le mécanisme d'authentification est basé sur les JSON Web Tokens (JWT) **[10]**.

Figure 3.11 : Processus d'authentification avec JWT

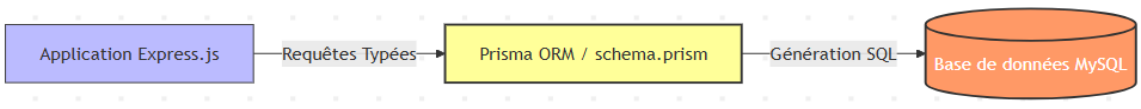


Source : Réalisation personnelle.

3.3.4 Gestion de la base de données avec Prisma ORM

La persistance des données est assurée par une base de données **MySQL**, accessible via **Prisma ORM**.
Prisma joue le rôle d'intermédiaire entre l'application et la base de données en générant automatiquement les requêtes SQL nécessaires. Cette approche améliore la lisibilité du code, réduit les risques d'erreurs et facilite les opérations de maintenance.
Le schéma Prisma centralise l'ensemble des entités de la plateforme telles que les utilisateurs, les produits, les commandes, les variantes, les collections ainsi que les historiques d'essayage virtuel.
Les migrations Prisma permettent également de versionner les évolutions de la structure de la base de données afin d'assurer une synchronisation fiable entre les différents environnements de développement.
Prisma ORM facilite l'accès aux données grâce à un typage fort et à un système de migrations automatisées **[5]**.

Figure 3.12 : Schéma Prisma et communication avec MySQL

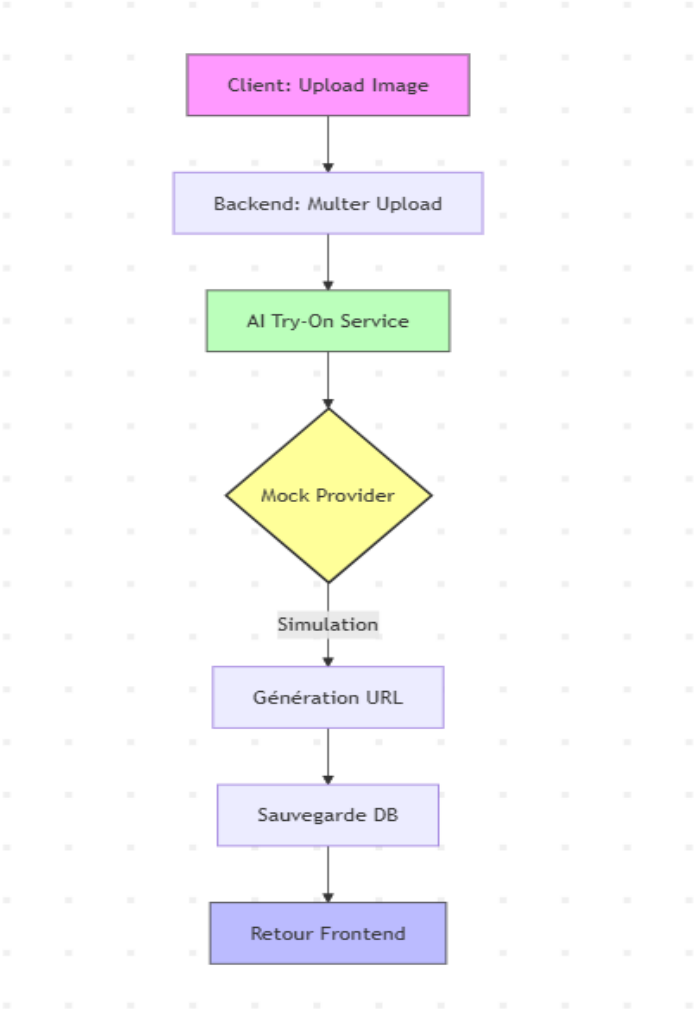


Source : Réalisation personnelle.

3.3.5 Module d'essayage virtuel par Intelligence Artificielle

L'une des principales innovations de MONO réside dans l'intégration d'un module d'essayage virtuel basé sur l'intelligence artificielle.
L'architecture du Backend repose sur un système de **Provider** permettant de dissocier la logique métier du fournisseur d'intelligence artificielle utilisé.
Durant le développement, un **Mock Provider** a été mis en place afin de simuler le comportement du service d'IA sans dépendre d'une API externe. Cette approche facilite les phases de développement et de test tout en préparant l'intégration future d'un fournisseur réel, tel que **FASHN AI** ou **Replicate**, sans modification majeure de l'architecture.
Chaque traitement est enregistré dans la base de données afin d'assurer le suivi de son état et de conserver l'historique des essais réalisés par les utilisateurs.

Figure 3.13 : Flux de traitement du module d'essayage virtuel



Source : Réalisation personnelle.

Conclusion :

Ce chapitre a présenté les différentes étapes de développement de la plateforme ainsi que les solutions techniques mises en œuvre. Le chapitre suivant est consacré à la validation du projet et à la présentation des résultats obtenus.

Chapitre 4 : Validation et Résultats

Introduction :

À l'issue de la phase de développement, plusieurs tests fonctionnels ont été réalisés afin de vérifier le bon fonctionnement des différentes fonctionnalités de la plateforme MONO. Cette étape a permis de valider la conformité de l'application par rapport aux besoins définis lors de la phase d'analyse, tout en s'assurant de la qualité des échanges entre le Frontend, le Backend et la base de données.
Les validations ont principalement porté sur les fonctionnalités métiers, la communication via les API REST ainsi que l'expérience utilisateur offerte par l'application.

4.1 Validation fonctionnelle

Les principales fonctionnalités développées ont été testées manuellement afin de vérifier leur bon fonctionnement dans différentes situations d'utilisation.
Le tableau suivant présente les principaux scénarios testés.

Tableau 4.1 auteur

Tableau 4.1 : Résultats des tests fonctionnels

Fonctionnalité	Description	Statut	Résultat
Authentification	Connexion et inscription des utilisateurs avec chiffrement	OK	✅ Validé
Catalogue des produits	Affichage, recherche et filtrage des articles disponibles	OK	✅ Validé
Gestion du panier	Ajout, modification et suppression d'articles avant achat	OK	✅ Validé
Liste des favoris	Sauvegarde et gestion des produits préférés des utilisateurs	OK	✅ Validé
Tableau de bord admin	Interface de gestion des commandes, stocks et utilisateurs	OK	✅ Validé
Module Try-On IA	Essai virtuel de vêtements et accessoires via réalité augmentée	OK	✅ Validé

Tableau 4.1 : Résultats des tests fonctionnels

Ces tests ont permis de confirmer le bon fonctionnement des principales fonctionnalités de la plateforme ainsi que la cohérence des traitements réalisés par le Backend.

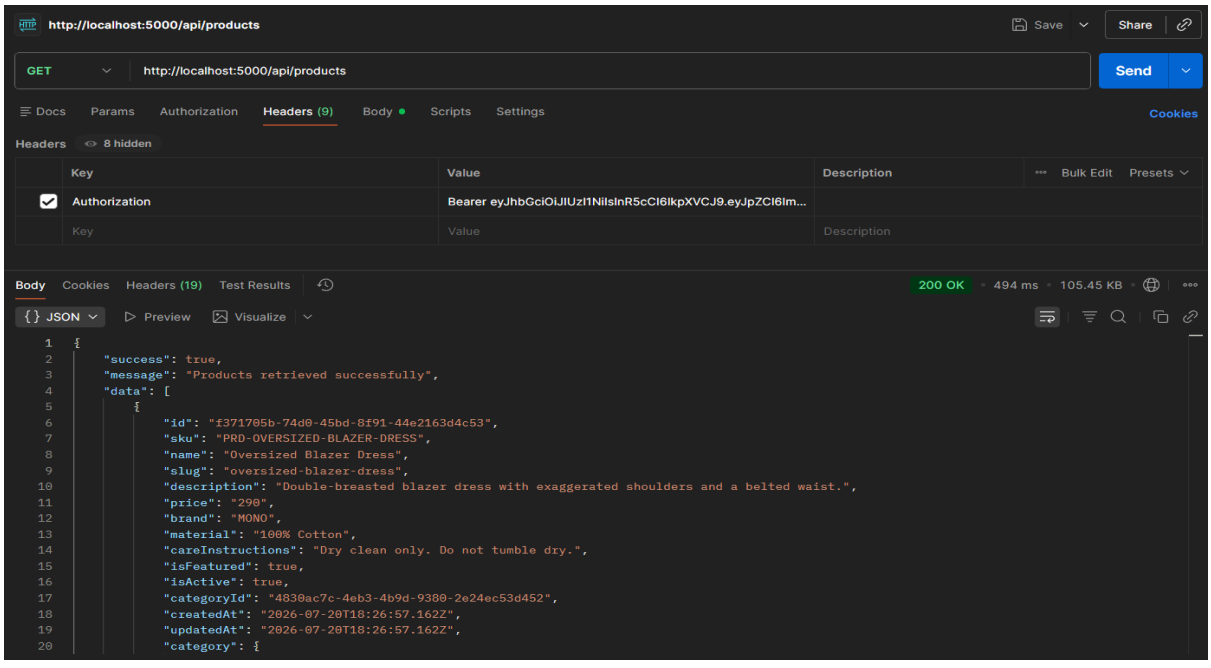
4.2 Validation des API REST

Les différentes API développées ont été vérifiées à l'aide de **Postman** afin de contrôler le bon fonctionnement des échanges entre le Frontend et le Backend.
Les principaux endpoints testés concernent :

- l'authentification des utilisateurs ;
- la gestion des produits ;
- les opérations sur le panier ;
- les collections ;
- les commandes.

Les réponses retournées par le serveur ont été vérifiées afin de s'assurer de la validité des données échangées ainsi que des codes de réponse HTTP.

Figure 4.1 : Validation des API REST avec Postman

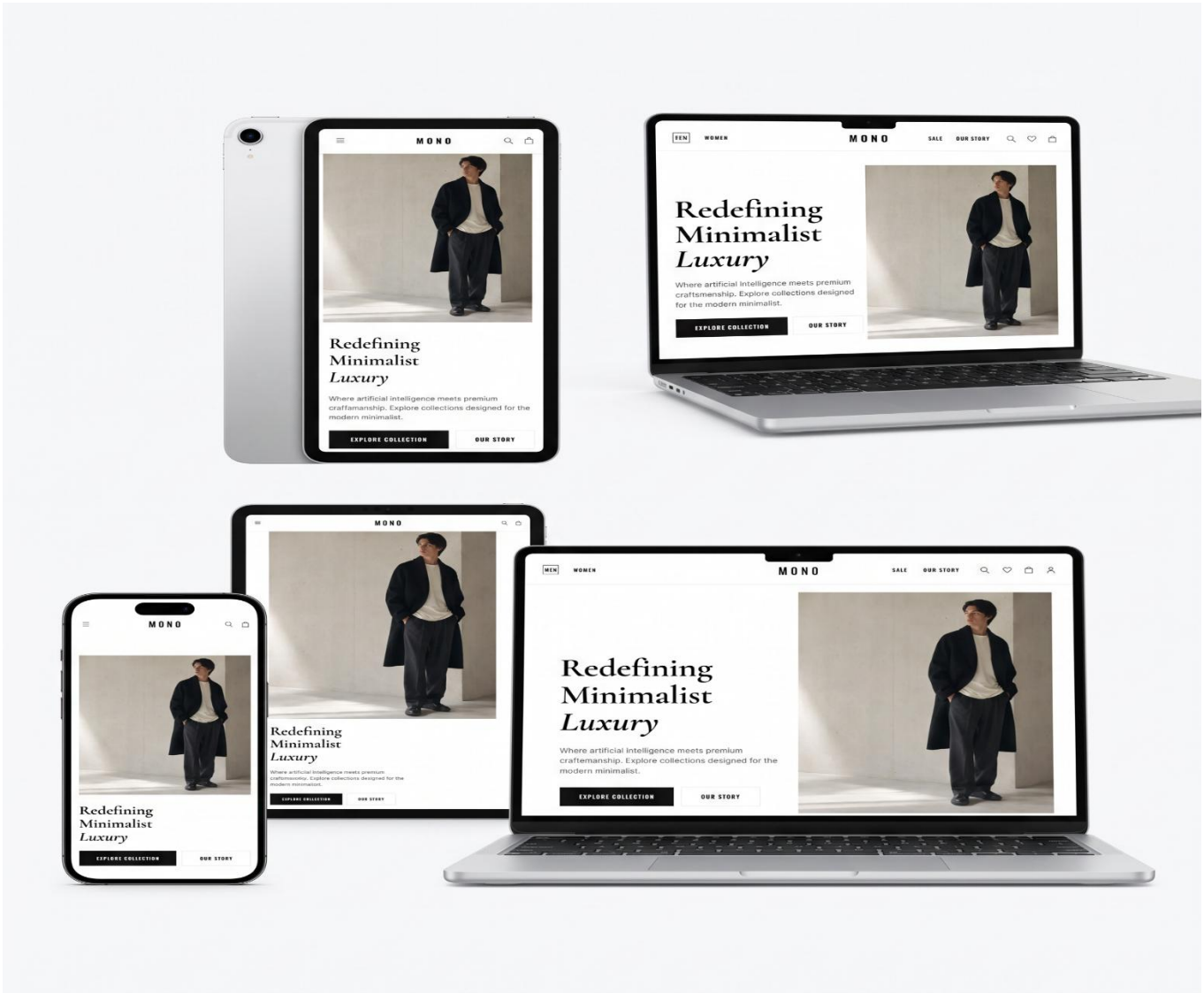


Source : Réalisation personnelle.

4.3 Validation de l'interface utilisateur

L'application a été testée sur plusieurs tailles d'écran afin de garantir une expérience utilisateur homogène.
Grâce à l'utilisation de **Tailwind CSS**, l'interface s'adapte automatiquement aux différents appareils (ordinateur, tablette et smartphone) tout en conservant une navigation fluide et une présentation cohérente.
Les animations développées avec **GSAP** ainsi que le défilement fluide assuré par **Lenis** ont également été vérifiés afin de garantir une expérience utilisateur immersive sans dégradation des performances.

Figure 4.2 : Interface responsive de MONO



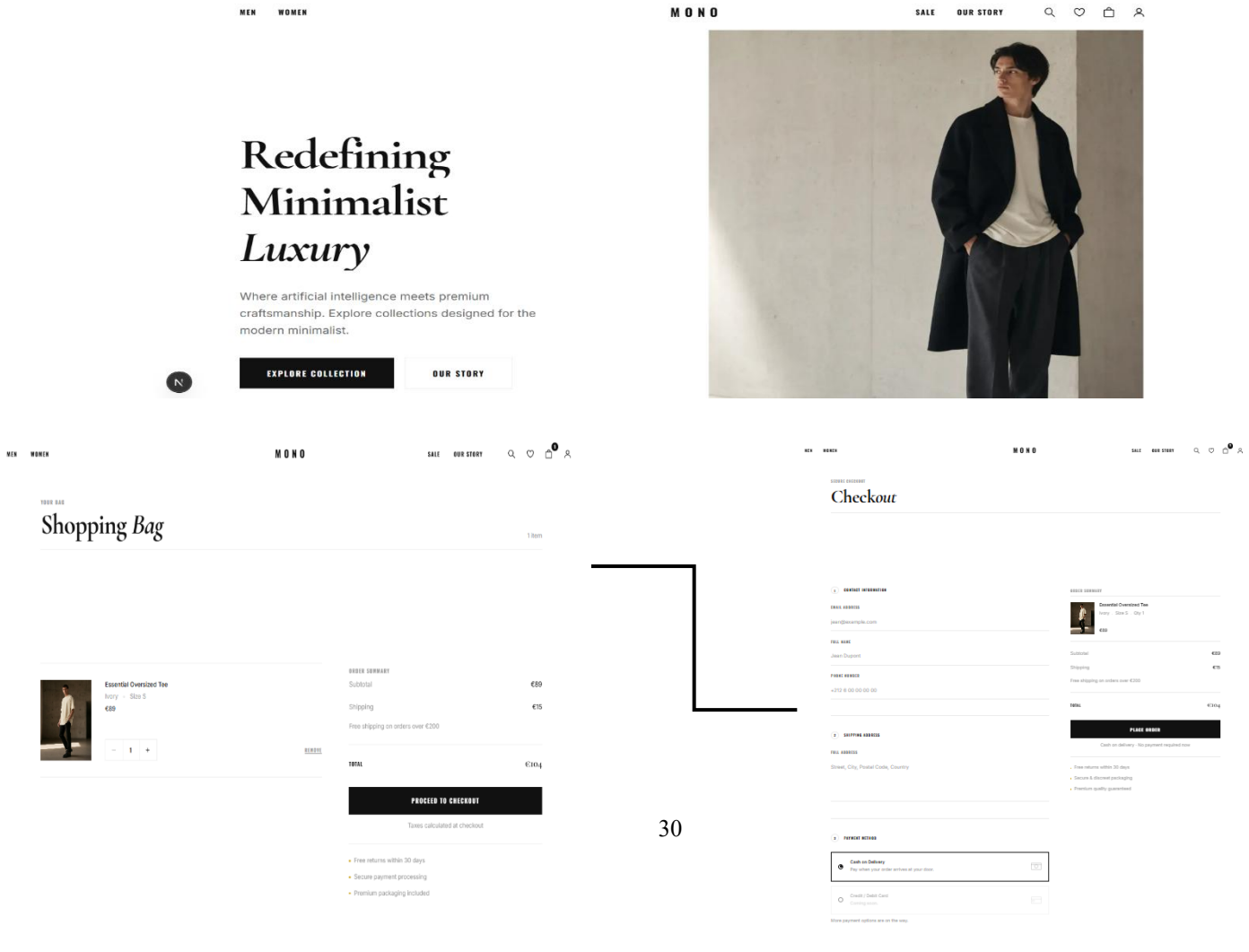
4.4 Résultat final de l'application

Les différentes phases de validation ont permis de confirmer que la plateforme répond aux objectifs fixés au début du projet.
MONO propose aujourd'hui une solution e-commerce moderne intégrant :

- une architecture Full-Stack modulaire ;
- une interface utilisateur responsive et performante ;
- une authentification sécurisée par JWT ;
- une gestion complète des produits, des commandes et du panier ;
- un tableau de bord d'administration ;
- un module d'essayage virtuel reposant sur une architecture compatible avec l'intelligence artificielle.

L'ensemble de ces fonctionnalités démontre la faisabilité technique de la solution et valide les choix architecturaux réalisés durant le développement.

Figure 4.3 : Résultat final de la plateforme MONO



4.5 Limites du projet

Malgré les résultats obtenus, certaines limites subsistent.

- Le module d'essayage virtuel utilise actuellement un **Mock Provider** destiné aux phases de développement et de validation.
- Aucun système de paiement électronique n'a été intégré dans cette première version.
- Le projet n'a pas encore été déployé dans un environnement de production.

Ces choix ont été réalisés afin de respecter les contraintes de temps et le périmètre fixé pour ce projet de fin d'études.

4.6 Perspectives d'amélioration

Plusieurs évolutions pourront être envisagées dans les prochaines versions de la plateforme :

- intégration d'un fournisseur d'intelligence artificielle réel (FASHN AI ou Replicate) ;
- ajout d'une solution de paiement en ligne telle que Stripe ;
- développement d'une application mobile consommant la même API REST ;
- mise en place d'un tableau de bord analytique avancé ;
- déploiement de la plateforme sur une infrastructure Cloud.

Ces perspectives permettront d'améliorer davantage les performances, les fonctionnalités et l'expérience utilisateur de MONO.

Conclusion

Ce chapitre a présenté les différentes phases de validation de la plateforme MONO à travers des tests fonctionnels, des tests des API REST ainsi que la validation de l'interface utilisateur. Les résultats obtenus confirment le bon fonctionnement des principales fonctionnalités développées et montrent que les objectifs fixés au début du projet ont été atteints. Les limites identifiées ainsi que les perspectives d'amélioration ouvrent la voie à de futures évolutions de la plateforme.

Conclusion Générale

Au terme de ce projet de fin d'études, nous avons conçu et développé **MONO**, une plateforme e-commerce de mode haut de gamme intégrant une architecture Full-Stack moderne ainsi qu'un module d'essayage virtuel basé sur l'intelligence artificielle.
Ce projet nous a permis de mettre en pratique les connaissances acquises durant notre formation en Licence Professionnelle, notamment en développement Frontend et Backend, conception de bases de données, architecture logicielle, sécurité des applications web et intégration des API REST.

Sur le plan technique, l'application repose sur une architecture découplée composée d'un Frontend développé avec **Next.js 15**, d'un Backend construit avec **Node.js**, **Express.js** et **Prisma ORM**, ainsi que d'une base de données **MySQL**. Cette architecture garantit une meilleure maintenabilité, une évolutivité importante et une séparation claire des responsabilités entre les différentes couches de l'application.

L'intégration d'un module d'essayage virtuel reposant sur une architecture de type **Provider Pattern** constitue l'une des principales valeurs ajoutées du projet. Bien que la version actuelle utilise un **Mock Provider** pour des raisons de développement, l'architecture mise en place permet l'intégration future d'un fournisseur d'intelligence artificielle réel sans modification majeure du système.

Les différents tests fonctionnels réalisés ont confirmé le bon fonctionnement des principales fonctionnalités de la plateforme, notamment l'authentification, la gestion des produits, du panier, des commandes, ainsi que l'administration de l'application.
Au-delà des aspects techniques, ce projet nous a permis de développer des compétences essentielles en analyse, conception, résolution de problèmes, organisation d'un projet logiciel et adoption des bonnes pratiques de développement.
Enfin, plusieurs perspectives d'évolution peuvent être envisagées, telles que l'intégration d'une solution d'intelligence artificielle en production, l'ajout d'un système de paiement en ligne, le développement d'une application mobile ainsi que le déploiement de la plateforme sur une infrastructure Cloud.

Ce projet constitue ainsi une expérience particulièrement enrichissante, tant sur le plan académique que professionnel, et représente une base solide pour le développement de solutions e-commerce intelligentes répondant aux besoins des utilisateurs et aux exigences du marché.

Bibliographie

[1] Next.js Documentation. https://nextjs.org/docs Consulté le : juillet 2026.

[2] React Documentation. https://react.dev Consulté le : juillet 2026.

[3] Node.js Documentation. https://nodejs.org/docs Consulté le : juillet 2026.

[4] Express.js Documentation. https://expressjs.com Consulté le : juillet 2026.

[5] Prisma ORM Documentation. https://www.prisma.io/docs Consulté le : juillet 2026.

[6] MySQL Documentation. https://dev.mysql.com/doc Consulté le : juillet 2026.

[7] Tailwind CSS Documentation. https://tailwindcss.com/docs Consulté le : juillet 2026.

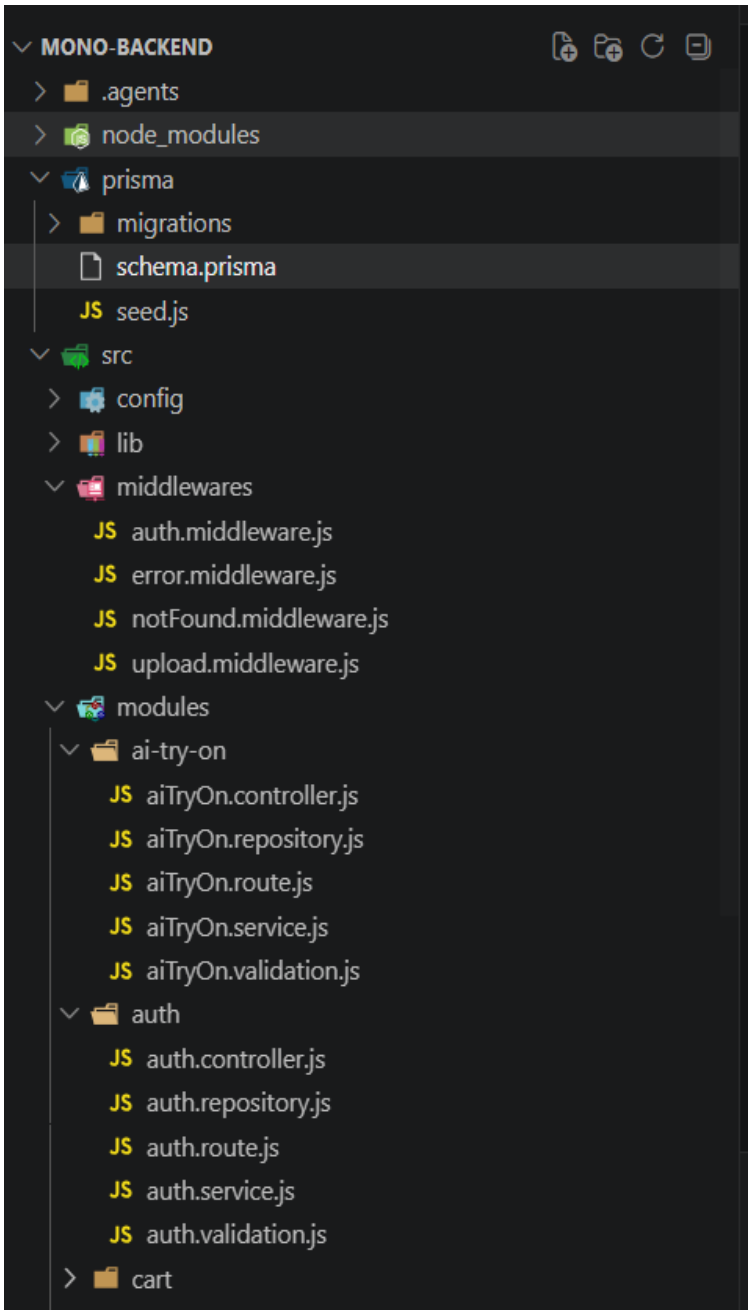
[8] Zustand Documentation. https://zustand-demo.pmnd.rs Consulté le : juillet 2026.

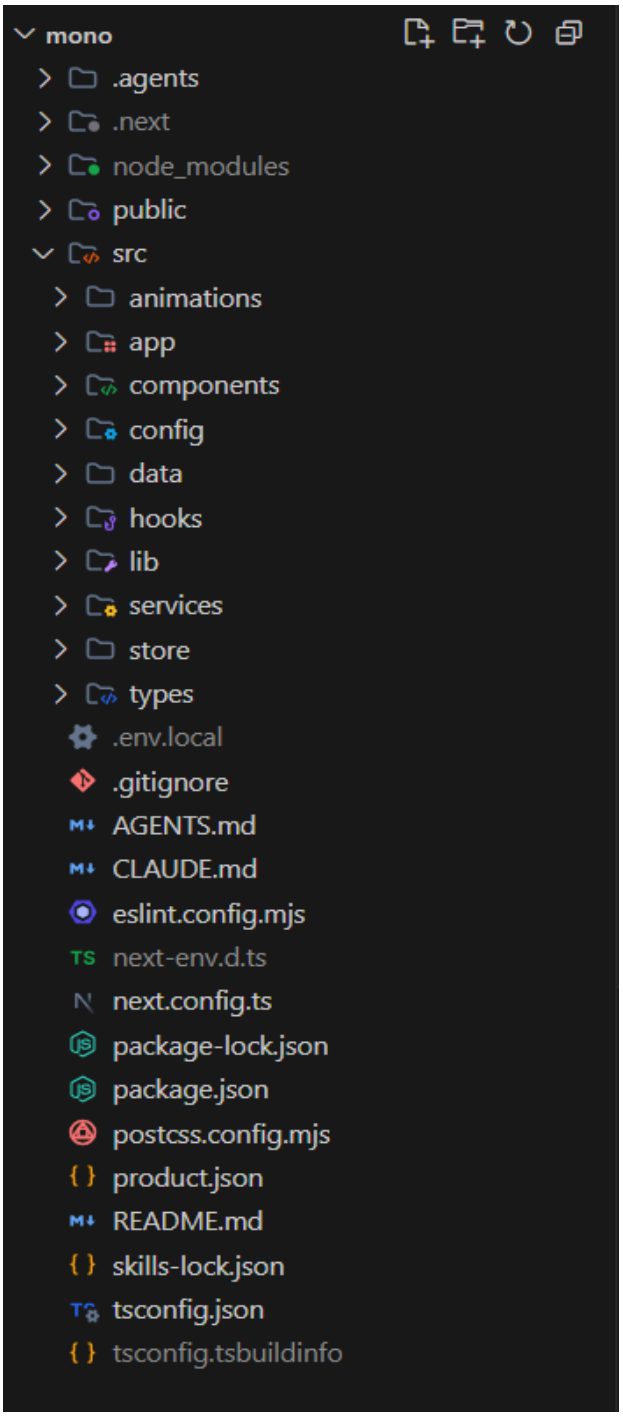
[9] GSAP Documentation. https://gsap.com/docs Consulté le : juillet 2026.

[10] JSON Web Token (JWT). https://jwt.io Consulté le : juillet 2026.

[11] MDN Web Docs. https://developer.mozilla.org Consulté le : juillet 2026.

Annexe A — Arborescence du projet



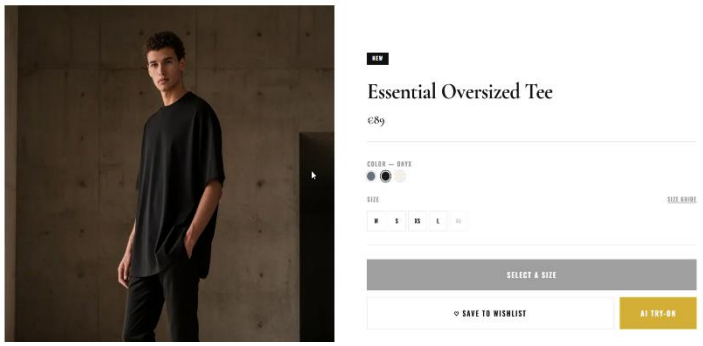


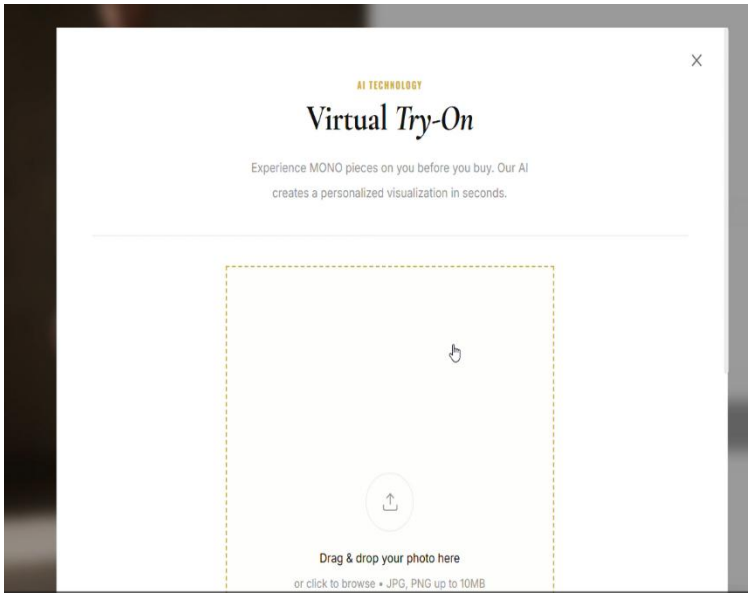
Annexe B — Schéma Prisma

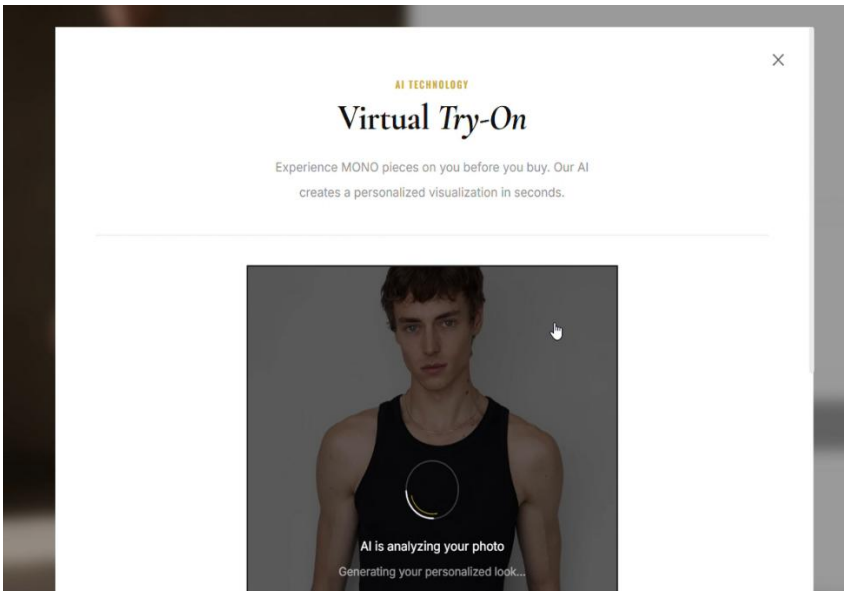
prisma > schema.prisma

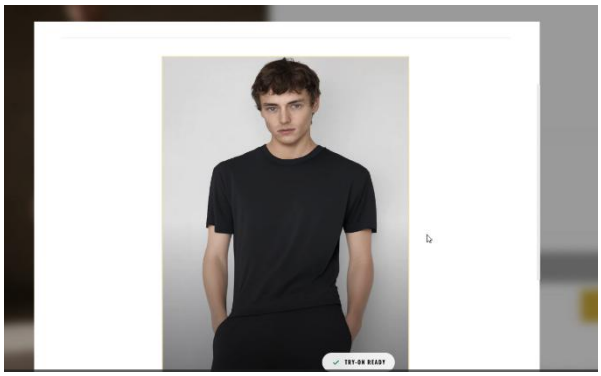
```
// ------------------------------------------------------------
// DATABASE CONFIGURATION
// ------------------------------------------------------------

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

// ------------------------------------------------------------
// Enums (Data Integrity & Type Safety)
// ------------------------------------------------------------

enum OrderStatus {
  PENDING
  PROCESSING
  SHIPPED
  DELIVERED
  CANCELLED
}

enum PaymentStatus {
  UNPAID
  PAID
  REFUNDED
}

enum PaymentMethod {
  STRIPE
  CASH_ON_DELIVERY
}

enum TryOnStatus {
  PROCESSING
  SUCCESS
  FAILED
```

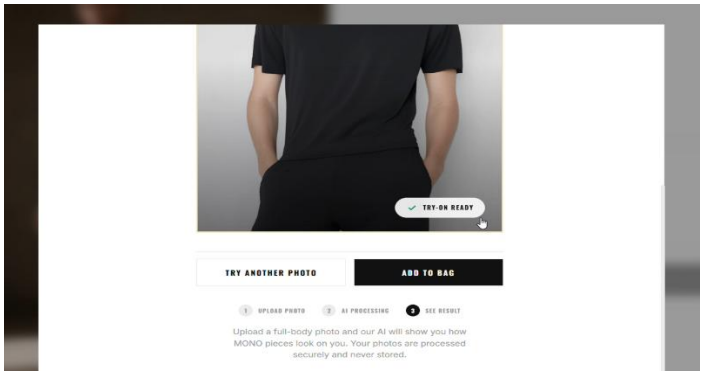
Annexe C — Résultats du module AI Virtual Try-On











iSMAGi®

Institut Supérieur de Management
d'Administration et de Génie Informatique
المعهد العالي للتدبير و الإدارة و الهندسة المعلوماتية

iSMAGi®
ENGINEERING
SCHOOL