Rapport de Stage - Réalisations en Développement React

Ce rapport documente mon parcours et mes réalisations au cours de ma formation et de mon stage en développement React chez NextGen Coding. L'objectif principal était d'acquérir une maîtrise approfondie des concepts fondamentaux et avancés de React, en les appliquant à travers le développement de plusieurs applications web modernes. Ce stage a été une opportunité de démontrer ma capacité à concevoir des solutions robustes, bien architecturées et conformes aux standards de l'industrie.

Mon apprentissage a été structuré autour de quatre projets distincts, chacun représentant une étape progressive dans l'acquisition de compétences clés en développement frontend.

Aperçu de la Progression d'Apprentissage

Chaque projet a été conçu pour me faire progresser de manière significative dans ma maîtrise de React et de l'écosystème du développement web moderne :

- **Fondation** : Appréhender les concepts de base de React, tels que JSX, les composants, les props et le state, ainsi que l'architecture fondamentale des applications.
- **Intégration** : Maîtriser l'intégration avec des APIs externes, la récupération et la gestion avancée des données, et la gestion des erreurs.
- **Architecture** : Comprendre et appliquer les principes d'une organisation de code propre et évolutive, essentielle pour des projets de grande envergure.
- **Maîtrise** : Gérer des états complexes à l'échelle de l'application et implémenter des patterns de développement de niveau entreprise, simulant des scénarios réels.

Mes Projets Réalisés en Détail

Chacun des projets ci-dessous a été l'occasion d'approfondir mes connaissances et de mettre en pratique des compétences spécifiques.

1. Introduction à React - E-Commerce Mode (Amen&Louay Fashion E-Commerce)

Objectif & Description : Ce projet a marqué mes premiers pas concrets dans React. L'objectif était de comprendre les bases de React (JSX, composants, props, state) en créant une mini-application e-commerce moderne et responsive. J'ai mis l'accent sur l'affichage dynamique des produits, l'implémentation de filtres combinables et une recherche instantanée, tout en assurant une interface utilisateur propre et intuitive, inspirée des boutiques de mode réelles. Ce

projet a été fondamental pour établir une architecture de composants claire et une bonne séparation des responsabilités.

Fonctionnalités Clés :

- **Grille de produits responsive** : Affichage attrayant des produits avec des images réalistes et des détails pertinents.
- **Filtres dynamiques et combinables** : Possibilité de filtrer les produits par catégorie (Femme, Homme, Enfant), type (Vêtements, Chaussures, Accessoires), statut (Nouveau, Promo) et marque, avec des mises à jour instantanées de la grille.
- **Recherche instantanée** : Filtrage des produits par nom en temps réel via une barre de recherche dédiée.
- **Header et footer modernes** : Éléments de navigation essentiels incluant logo, liens utiles et icônes (utilisateur, panier, favoris).
- **Code propre et maintenable** : Organisation logique des composants et des données pour une meilleure lisibilité et extensibilité.

Stack Technique :

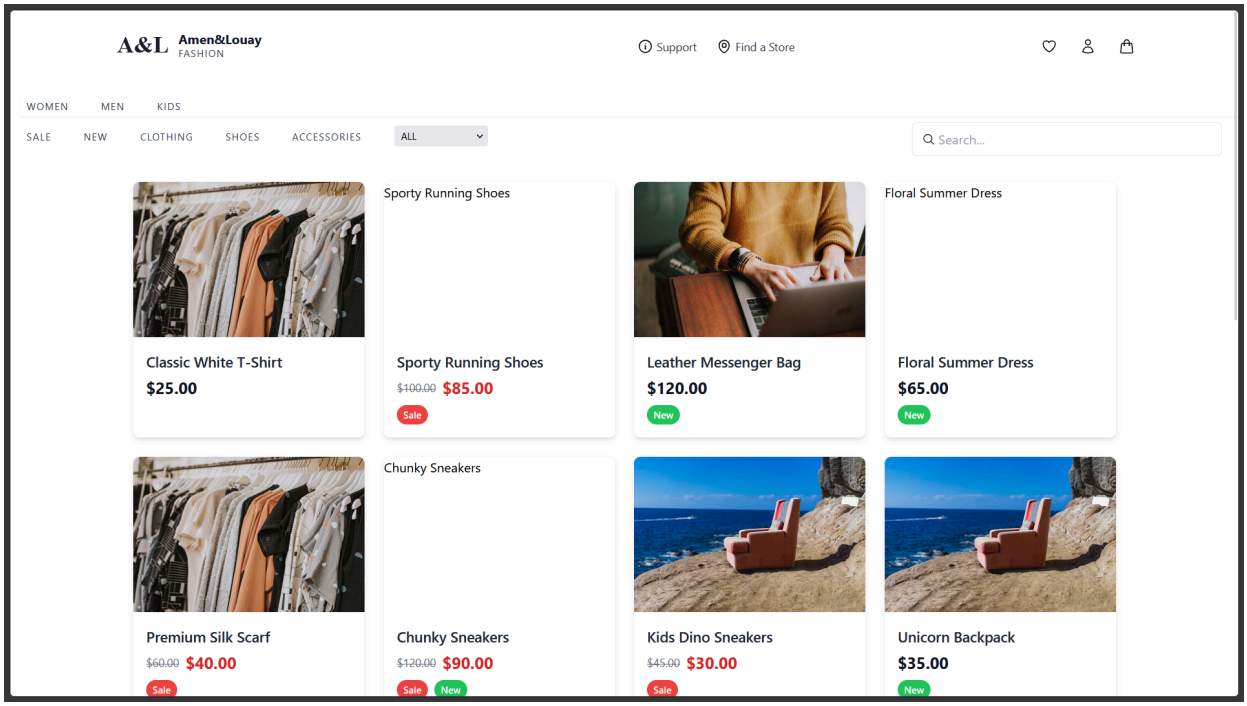
- **React 19** : Utilisation des composants fonctionnels et des hooks pour une gestion d'état efficace.
- **Vite** : Outil de build nouvelle génération pour un développement et un build ultra-rapides.
- **Tailwind CSS** : Framework CSS utility-first pour un styling rapide et moderne.
- **ESLint** : Assurer la qualité et la cohérence du code.
- **PostCSS** : Traitement et optimisation CSS.

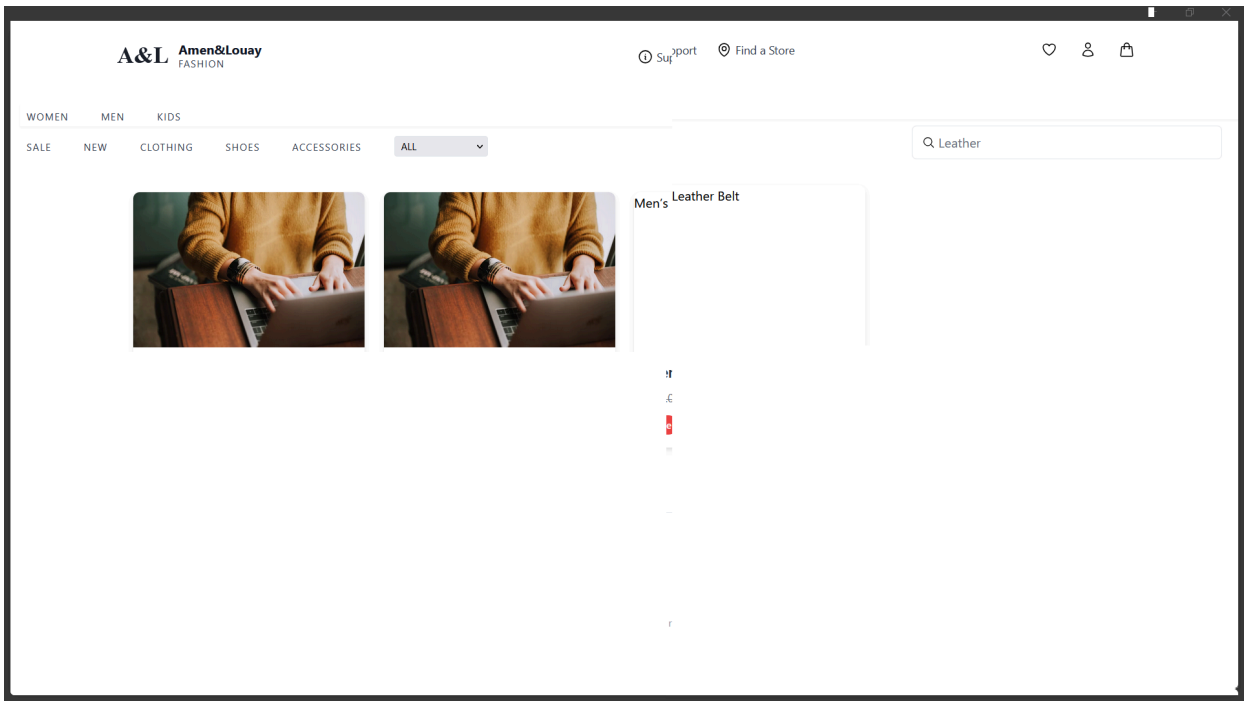
Structure du Projet :

```
src/
  components/
    Footer.jsx          # Footer moderne avec logo et liens
    Header/
      Header.jsx        # Layout principal du header
      MainNav.jsx       # Navigation et filtres
      SearchBar.jsx     # Barre de recherche
      SideNav.jsx       # Icônes utilisateur, panier, favoris
    Logo.jsx            # Logo de la marque (utilise logo.svg)
    ProductGrid.jsx     # Grille de produits
    ProductItem.jsx     # Carte produit individuelle
    UtilityNav.jsx      # Liens support et magasins
  data/
    products.js         # Données produits (images, marques, etc.)
  utils/
    enums.js            # Enums catégorie/type
  assets/
```

```
  icons/              # Icônes SVG (logo, panier, user, etc.)
App.jsx               # Logique principale et état global
main.jsx              # Point d'entrée React
index.css             # Styles de base Tailwind
```

Captures d'écran :





Dépôt GitHub : Introduction - React

2. DataFetch Central - Navigateur d'API REST

Objectif & Description : Ce projet visait à approfondir ma compréhension des Hooks React fondamentaux (`useState`, `useEffect`) et à maîtriser l'intégration d'API. J'ai développé un outil sophistiqué pour explorer, tester et interagir avec des API REST, en mettant un accent particulier sur la gestion avancée des données et des erreurs. La création d'un hook personnalisé (`useApi`) a été essentielle pour centraliser et réutiliser la logique des interactions API, rendant l'application robuste et résiliente.

Fonctionnalités Clés :

- **Récupération dynamique de données (Opérations GET)** : Interface de recherche intuitive pour saisir n'importe quelle URL d'API REST (notamment JSONPlaceholder), affichage intelligent des données détectant les types de ressources, boutons pré-configurés pour les endpoints courants, et états de chargement en temps réel.
- **Opérations CRUD complètes** : Implémentation des fonctionnalités de Création (POST), Lecture (GET), Mise à Jour (PUT/PATCH) et Suppression (DELETE) avec des formulaires dédiés, validation côté client et dialogues de confirmation.
- **Gestion avancée des erreurs et retours utilisateur** : Le hook `useApi` gère les erreurs HTTP (4xx, 5xx), les erreurs réseau et d'analyse, fournissant des messages clairs et actionnables.
- **Prévention des conditions de course** : Utilisation d'`AbortController` pour annuler les requêtes en cours et éviter les comportements inattendus.
- **Expérience utilisateur améliorée** : Design responsive et mobile-first avec Tailwind CSS, support du thème sombre, visualiseur JSON brut pour inspecter les réponses API brutes, et indicateurs de chargement fluides.

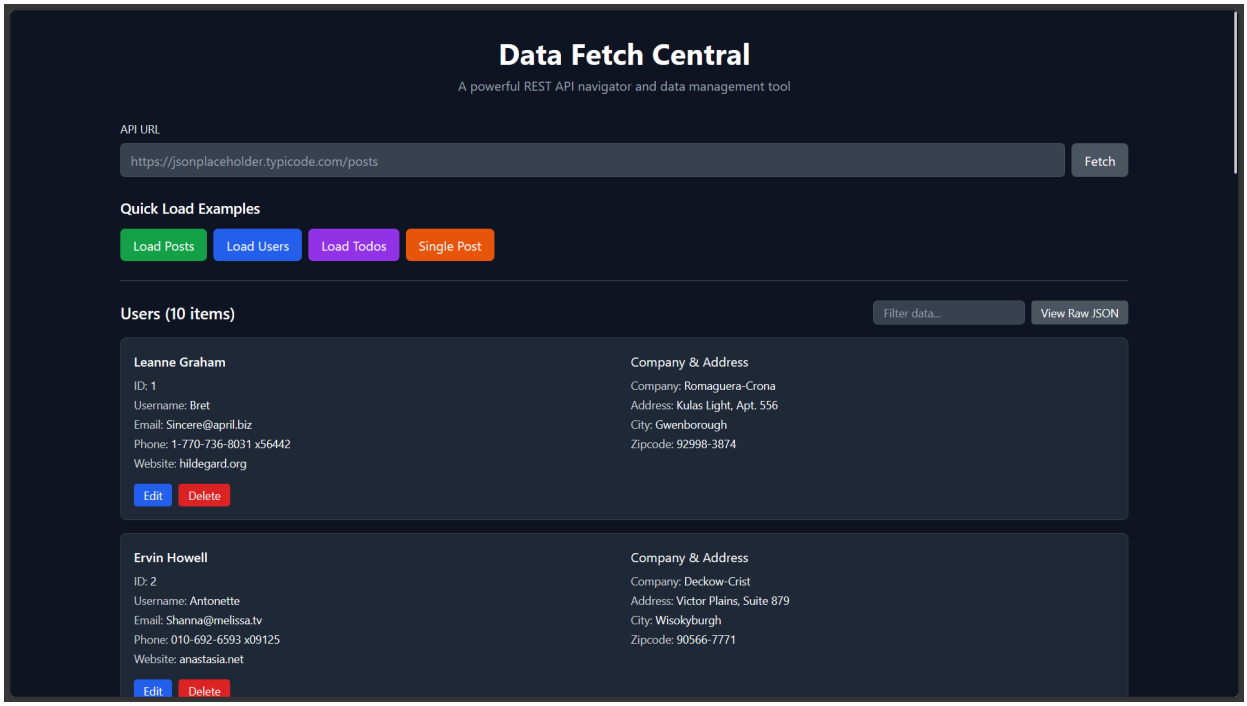
Stack Technique :

- **React 19** : Utilisation des hooks modernes et des fonctionnalités concurrentes.
- **Vite 7.0** : Outil de build ultra-rapide et serveur de développement.
- **JavaScript ES6+** : Utilisation des fonctionnalités et de la syntaxe JavaScript modernes.
- **Tailwind CSS 3.4** et **PostCSS** : Framework CSS utility-first pour un styling rapide et optimisé.
- **ESLint** : Pour l'application de la qualité et la cohérence du code.
- **React Hooks** (`useState`, `useEffect`, `useCallback`, `useRef`) : Pour la gestion d'état et l'optimisation des performances.
- **Hooks Personnalisés** (`useApi`) : Encapsulation de logique réutilisable pour les interactions API.
- **Fetch API** : API native du navigateur pour les requêtes HTTP.
- **AbortController** : Pour l'annulation des requêtes et la prévention des conditions de course.

Structure du Projet :

```
DataFetch Central/
├── public/
│   └── index.html                  # Template HTML
├── src/
│   ├── App.jsx                     # Composant principal de l'application et logique de routage
│   ├── main.jsx                    # Point d'entrée de l'application React
│   ├── index.css                   # Styles globaux et directives Tailwind
│   ├── components/
│   │   ├── SearchBar.jsx           # Saisie d'URL et fonctionnalité de recherche
│   │   ├── DataDisplay.jsx         # Rendu et affichage intelligents des données
│   │   ├── PostDataForm.jsx        # Composant de formulaire pour créer de nouvelles données
│   │   └── EditDataForm.jsx        # Composant de formulaire pour éditer les données existantes
│   ├── hooks/
│   │   └── useApi.js               # Hook personnalisé pour toutes les interactions API
│   ├── constants/
│   │   └── resourceTypes.js        # Constantes de types de ressources et mappings
│   └── utils/
│       └── helpers.js              # Fonctions utilitaires et helpers
├── package.json                    # Dépendances et scripts
├── vite.config.js                  # Configuration Vite
├── tailwind.config.js              # Configuration Tailwind CSS
├── postcss.config.js           # Configuration PostCSS
└── eslint.config.js                # Configuration ESLint
```

Captures d'écran :



[Insérer capture d'écran de la gestion d'erreurs ou d'un formulaire CRUD]

Dépôt GitHub : Data Fetch Central

3. Project Template - Fondation d'Architecture Propre

Objectif & Description : Ce projet a été fondamental pour apprendre à bien structurer un projet React afin d'assurer une meilleure organisation du code. J'ai établi un template de projet React méticuleusement conçu, en intégrant les principes d'**Architecture Propre**, l'utilisation de **Tailwind CSS v3**, et les pratiques de développement standards de l'industrie. Ce template fournit une base solide et évolutive, essentielle pour construire des applications React maintenables et professionnelles qui peuvent grandir avec les exigences du projet et la taille de l'équipe.

Fonctionnalités Clés :

- **Organisation modulaire basée sur les fonctionnalités** : Structure de dossiers claire et logique, regroupant le code par domaine métier (ex: `auth`, `products`) plutôt que par type de fichier.
- **Bibliothèque de composants réutilisables** : Séparation des composants génériques (`shared/components`) des composants spécifiques aux fonctionnalités.
- **Abstraction de la couche de service** : Séparation de la logique métier des appels API, facilitant les tests et la maintenance.
- **Structure de dossiers évolutive** : Conçue pour s'adapter à des projets de petite à très grande envergure (1 à 50+ développeurs).

- **Implémentation des meilleures pratiques de développement** : Utilisation d'ESLint pour la qualité du code et des conventions de nommage cohérentes.

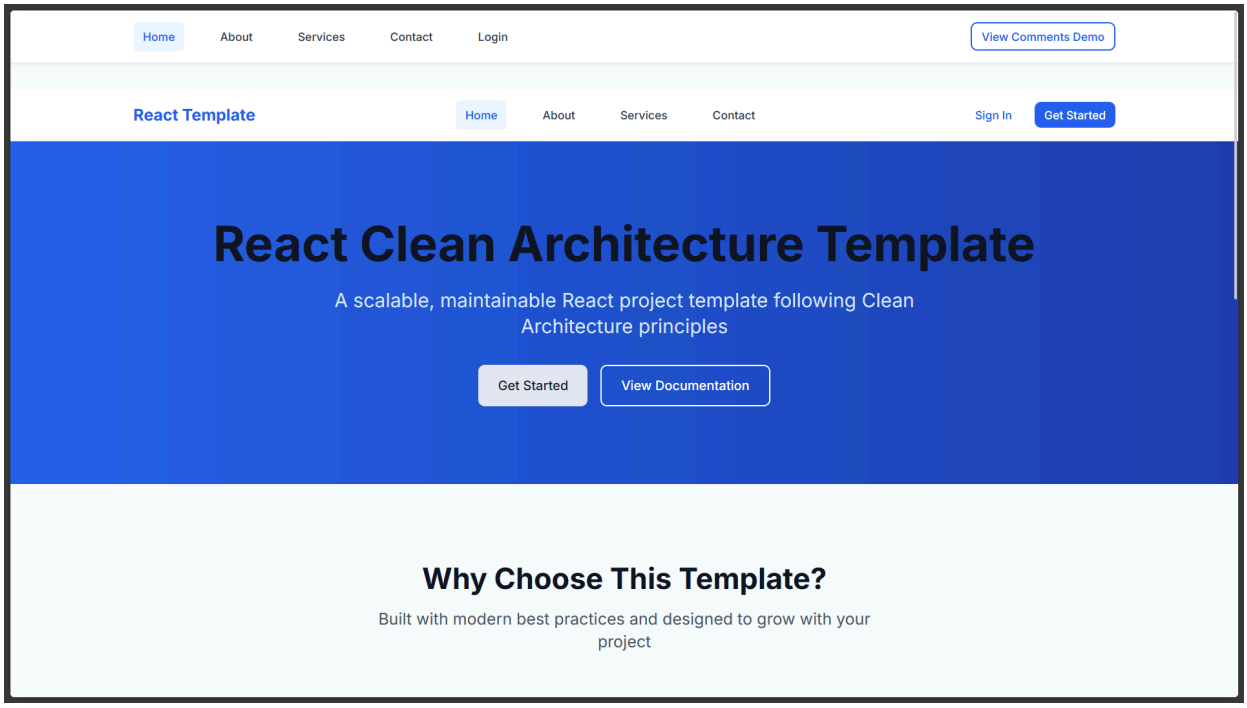
Stack Technique :

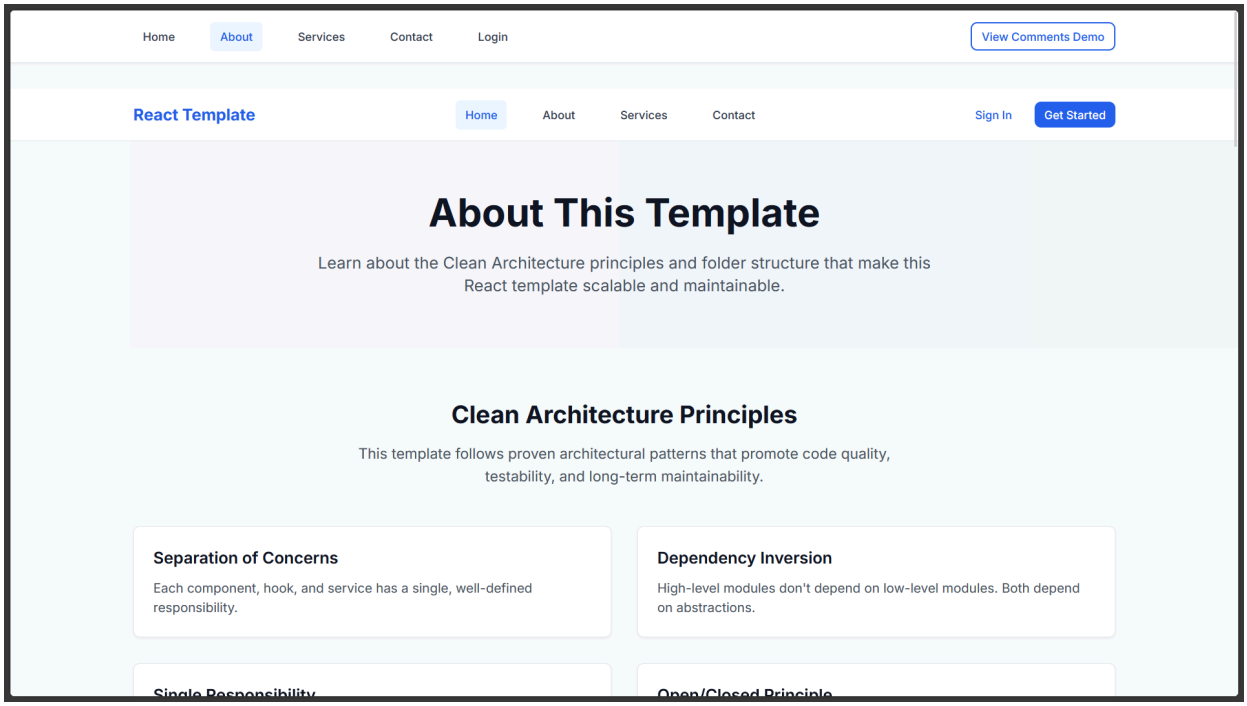
- **React 19** : Fondation pour le développement de l'interface utilisateur.
- **Vite** : Outil de build rapide pour un environnement de développement efficace.
- **Tailwind CSS v3** : Framework CSS utility-first pour un styling rapide et cohérent.
- **ESLint** : Pour l'application des règles de qualité et de style de code.

Structure du Projet (Exemple type d'Architecture Propre) :

```
Project Template/
├── src/
│   ├── app/                    # Configuration globale, providers (ex: AuthProvider, ThemeProvider)
│   ├── features/               # Modules métier (authentification, produits, utilisateurs, etc.)
│   │   ├── auth/
│   │   │   ├── components/ # Composants spécifiques à l'authentification
│   │   │   ├── hooks/          # Hooks personnalisés pour la logique d'authentification
│   │   │   ├── services/       # Logique d'appel API pour l'authentification
│   │   │   └── pages/          # Pages spécifiques à l'authentification (Login, Register)
│   ├── shared/                 # Composants, hooks, utilitaires réutilisables à travers l'application
│   │   ├── components/         # Composants UI génériques (Button, Modal, Input)
│   │   ├── hooks/              # Hooks réutilisables (useDebounce, useLocalStorage)
│   │   ├── utils/              # Fonctions utilitaires (formatage de date, validation)
│   │   └── constants/          # Constantes globales (API_URLS, ENUMS)
│   ├── pages/                  # Composants de page (assemblent les fonctionnalités des "features")
│   ├── assets/                 # Images, icônes, styles globaux, polices
│   ├── App.jsx                 # Composant racine de l'application
│   └── main.jsx                # Point d'entrée React
├── public/                     # Fichiers statiques
├── package.json                # Dépendances et scripts
├── vite.config.js              # Configuration Vite
├── tailwind.config.js          # Configuration Tailwind CSS
├── postcss.config.js           # Configuration PostCSS
└── eslint.config.js            # Configuration ESLint
```

Captures d'écran :





Sign in to your account

Or create a new account

Email address *

Enter your email

Password *

Enter your password

Remember me

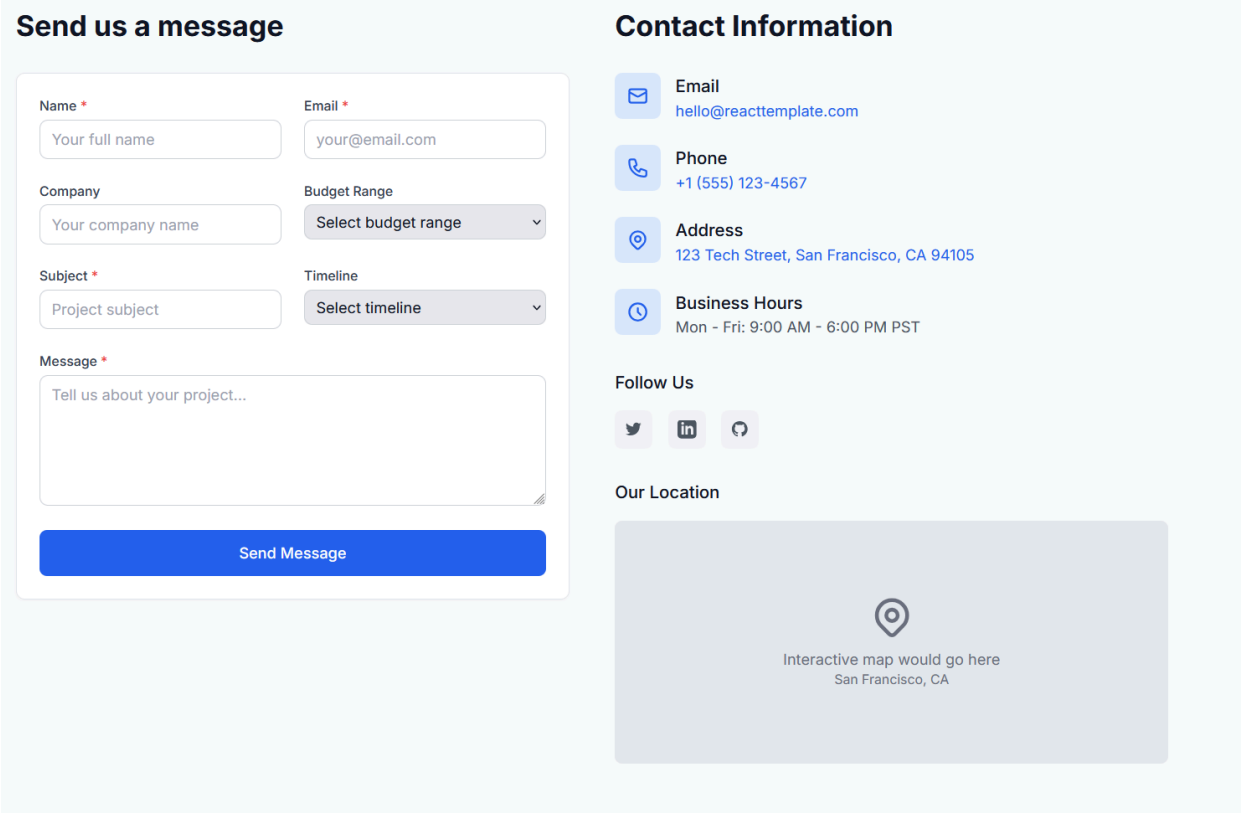
Forgot your password?

Sign in

Or continue with

Google

GitHub



Dépôt GitHub : [Project Template]

4. Project Advanced - Clone Amazon E-Commerce

Objectif & Description : Ce projet a été le point culminant de ma formation, me permettant de mettre en pratique l'ensemble de mes connaissances en réalisant une application complète. J'ai développé une réplique fonctionnelle et détaillée de la plateforme Amazon, démontrant une maîtrise avancée des patterns React, de la gestion d'état complexe (avec Zustand), et des interactions utilisateur sophistiquées. Ce projet simule un environnement de développement de niveau entreprise, intégrant des fonctionnalités telles que l'authentification, la gestion du panier, les commandes, et les services de localisation.

Fonctionnalités Clés :

- **Fonctionnalités E-commerce Core** : Catalogue de produits avec affichage en grille responsive et pagination, recherche avancée avec filtres multiples (catégorie, prix, note, marque), système de panier complet (gestion des quantités, calcul des taxes/remises), processus de commande détaillé, historique des commandes avec suivi, et listes de souhaits multiples.
- **Gestion Utilisateur Avancée** : Authentification sécurisée (JWT) avec validation, tableau de bord personnalisé, gestion complète du profil, et routes protégées avec middleware d'autorisation.

- **Services de Localisation Intelligents** : Géolocalisation automatique, sélection dynamique de localisation par code postal avec validation, options de livraison adaptées et calcul en temps réel des coûts.
- **Interface Utilisateur Premium** : Design responsive optimisé pour tous les appareils, thème Amazon authentique, animations et micro-interactions fluides, et conformité WCAG pour l'accessibilité.
- **Gestion d'état complexe** : Utilisation de Zustand avec middleware de persistance pour une gestion légère et performante de l'état global.
- **Système de recherche robuste** : Recherche textuelle, filtres combinés et tri intelligent.
- **Optimisations de performance** : Lazy loading des routes et composants, mémoïsation, code splitting et compression.

Stack Technique :

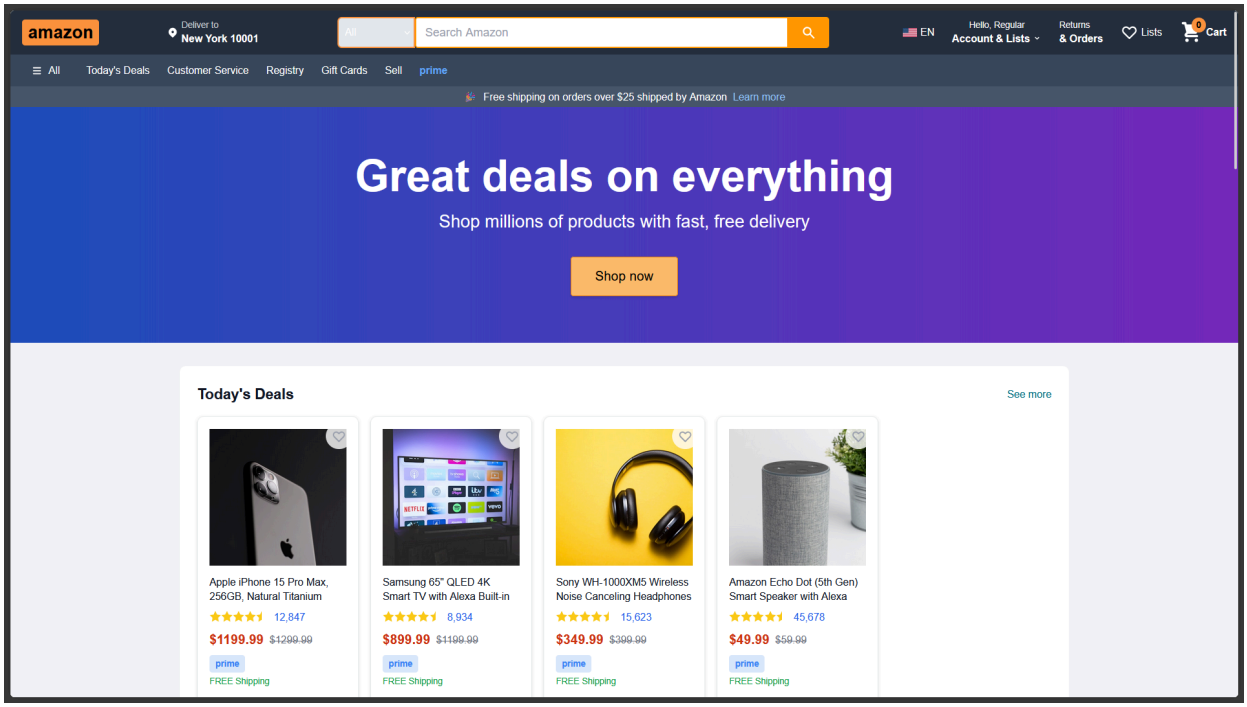
- **Frontend Framework** : **React 19.1.0** (bibliothèque UI avec hooks modernes), **Vite 7.0.4** (build tool ultra-rapide avec HMR), **React Router DOM 7.6.3** (routage côté client).
- **Styling et UI** : **Tailwind CSS 3.4.17** (framework CSS utility-first), **PostCSS 8.5.6** et **Autoprefixer 10.4.21** (processeurs CSS).
- **Gestion d'État** : **Zustand 5.0.6** (store management léger et performant), **Zustand Persist Middleware** (persistance automatique des données).
- **Outils de Développement** : **ESLint 9.30.1** (linting JavaScript/React), **Vite Plugin React 4.6.0** (intégration React optimisée).
- **Bibliothèques Utilitaires** : **Axios 1.10.0** (client HTTP pour les appels API), **Date-fns** (manipulation des dates).

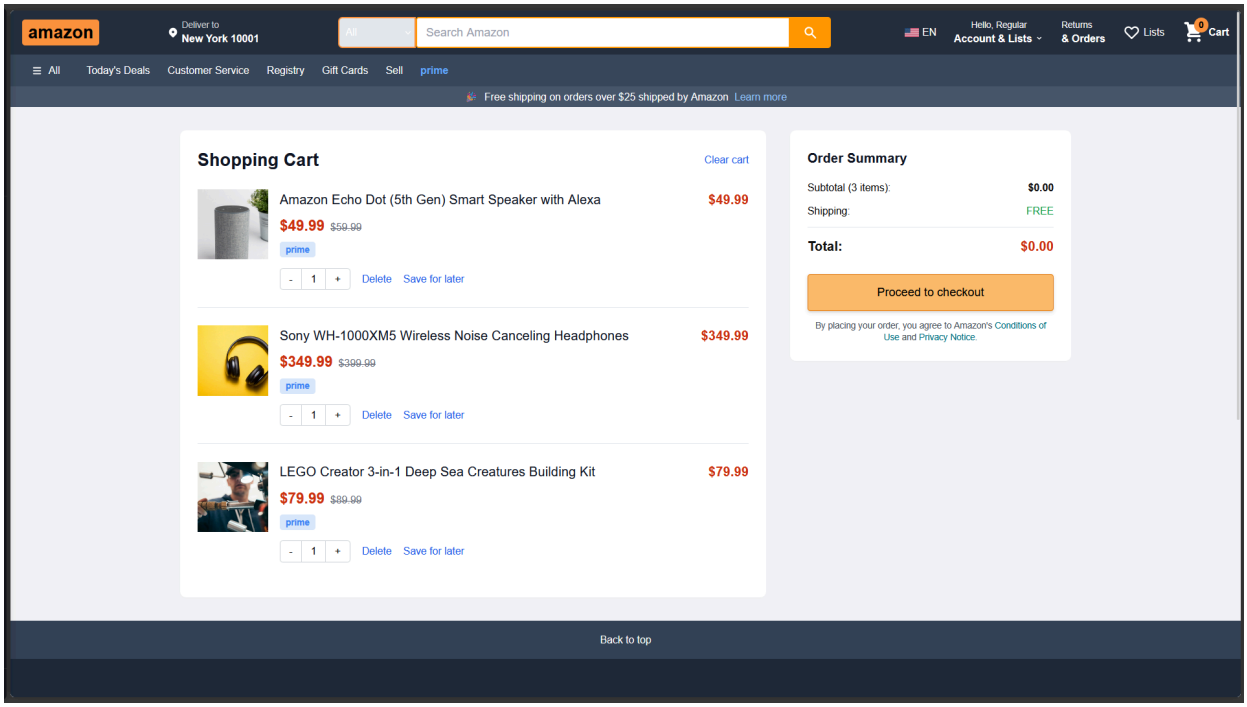
Structure du Projet :

```
project-advanced/
├── public/                    # Fichiers statiques
│   └── vite.svg
├── src/                       # Code source principal
│   ├── components/            # Composants réutilisables
│   │   ├── Amazon/            # Composants spécifiques Amazon (Header, Navigation,
Footer, ProductCard, ProductGrid)
│   │   ├── Auth/              # Composants d'authentification (Login, Register)
│   │   ├── LocationModal.jsx  # Modal de sélection de localisation
│   │   └── ProtectedRoute.jsx # Route protégée
│   ├── pages/                 # Pages de l'application (Home, SearchResults,
ProductDetail, Cart, Checkout, etc.)
│   ├── store/                 # Gestion d'état Zustand (authStore, cartStore,
wishlistStore, orderStore, locationStore)
│   ├── services/              # Services et API (api.js)
│   ├── data/                  # Données statiques (products.js)
│   ├── App.jsx                # Composant racine
│   ├── main.jsx               # Point d'entrée
```

```
│   └── index.css                 # Styles globaux et thème Amazon
├── eslint.config.js              # Configuration ESLint
├── tailwind.config.js            # Configuration Tailwind
├── postcss.config.js             # Configuration PostCSS
├── vite.config.js                # Configuration Vite
└── package.json                  # Dépendances et scripts
```

Captures d'écran :





Dépôt GitHub : [Amazon Copy]

Auteur

Amen Ellah Kerimi
Étudiant en Ingénierie des Systèmes Informatiques
📧 Email FSB : amenellah.kerimi@fsb.ucar.tn
LinkedIn GitHub