Documentation d'Architecture - Plateforme Porsche

👋 Bienvenue !

Ce projet est une plateforme e-commerce pour Porsche comprenant :

- Un **backend Node.js/Express/MongoDB** (déjà développé)
- Un **frontend React/Vite** (à développer selon cette architecture)

📖 Guide de lecture

Pour commencer rapidement

1. 📘 **GUIDE_DEMARRAGE.md**
   - Installation pas à pas
   - Configuration de l'environnement
   - Premiers tests
   - 👉 **COMMENCEZ ICI !**

Pour comprendre l'architecture

2. 📗 **ARCHITECTURE_REACT.md**
   - Analyse du backend existant
   - Architecture React proposée
   - Structure détaillée des dossiers
   - Bonnes pratiques SOLID
   - Exemples de code complets

3. 📙 **STRUCTURE_EXEMPLE.md**
   - Vue simplifiée pour débutants
   - Flow de données expliqué
   - Correspondance Backend ↔ Frontend
   - Checklist progressive
   - Conseils pédagogiques

Fichiers d'exemple prêts à l'emploi

4. 📂 **EXEMPLES_CODE/**
   - `config-api.js` - Configuration Axios
   - `authService.js` - Service d'authentification
   - `AuthContexte.jsx` - Context API
   - `useAuth.js` - Hook personnalisé
   - `RoutePrivee.jsx` - Protection des routes
   - `RouteAdmin.jsx` - Routes admin

- `Bouton.jsx` + `Bouton.css` - Composant réutilisable

🗂️ Architecture du projet

Backend (Node.js) - EXISTANT ✅

```
Node/
├── server.js                # Point d'entrée Express
├── controllers/             # Logique métier (19 contrôleurs)
├── models/                  # Schémas MongoDB (18 modèles)
├── routes/                  # Routes API REST
├── middlewares/             # Auth, validation, sécurité
├── utils/                   # Utilitaires, constants
└── db/                      # Configuration MongoDB
```

Technologies :

- Express.js (serveur)
- MongoDB + Mongoose (base de données)
- JWT (authentification)
- Stripe (paiement)
- Multer (upload fichiers)
- Bcrypt (hachage mots de passe)

Frontend (React/Vite) - À DÉVELOPPER 🚧

```
React/src/
├── config/                  # Configuration (API, routes)
├── services/                # Services API (appels backend)
├── hooks/                   # Hooks personnalisés
├── contextes/               # Context API (état global)
├── composants/              # Composants réutilisables
│   ├── communs/             # Bouton, Modale, Chargement...
│   ├── layout/              # EnTete, Navigation, PiedDePage
│   ├── voiture/             # Composants voitures
│   ├── configurateur/       # Composants configuration
│   ├── panier/              # Composants panier
│   ├── commande/            # Composants commande
│   └── protection/          # Routes protégées
├── pages/                   # Pages de l'application
│   ├── Accueil/
│   ├── Authentification/
│   ├── Catalogue/
│   ├── Configurateur/
│   ├── Panier/
│   ├── Commande/
│   ├── Profil/
```

```
│   └── Administration/
├── utils/                      # Fonctions utilitaires
└── styles/                     # Styles globaux (variables CSS)
```

Technologies :

- React 18
- Vite (build tool)
- React Router (navigation)
- Axios (requêtes HTTP)
- Context API (état global)
- CSS Modules / CSS classique

🎯 Fonctionnalités principales

👤 Authentification

- Connexion / Inscription
- Gestion de profil
- Rôles : Client, Conseiller, Responsable, Admin
- Routes protégées par authentification

🚗 Catalogue de voitures

- Liste des modèles Porsche (911, Cayman, Cayenne...)
- Filtres (neuve/occasion)
- Détails techniques
- Galerie photos

🎨 Configurateur

- Sélection couleur extérieure
- Sélection couleur intérieure
- Choix des jantes
- Choix des sièges
- Sélection de packages
- Calcul du prix en temps réel
- Récapitulatif de configuration

🛍️ Accessoires

- Catalogue d'accessoires
- Détails et photos
- Ajout au panier

🛒 Panier

- Ajout/suppression d'articles
- Modification de quantités
- Calcul du total
- Sauvegarde dans localStorage

💳 Commande & Paiement

- Formulaire de commande (adresse de livraison)
- Intégration Stripe Checkout
- Confirmation de commande
- Historique des commandes

👔 Administration

- Tableau de bord
- Gestion des voitures (CRUD)
- Gestion des accessoires (CRUD)
- Gestion des utilisateurs
- Gestion des commandes
- Upload de photos

🔄 Flow de données (exemple complet)

Scénario : Un client achète une Porsche 911

```
1. ACCUEIL
   └─> Clic sur "Voir les modèles"

2. CATALOGUE
   ├─> GET /voiture (API)
   ├─> Affichage des voitures
   └─> Clic "Configurer" sur 911

3. CONFIGURATEUR
   ├─> GET /model_porsche/:id (API)
   ├─> GET /couleur_exterieur (API)
   ├─> GET /couleur_interieur (API)
   ├─> GET /taille_jante (API)
   ├─> GET /siege (API)
   ├─> GET /package (API)
   ├─> Sélections utilisateur → État local (ConfigurateurContexte)
   ├─> Calcul prix en temps réel (utils/calculPrix.js)
   └─> Clic "Ajouter au panier"

4. PANIER
   ├─> PanierContexte.ajouterArticle()
   ├─> Sauvegarde localStorage
   ├─> Affichage panier
```

```
    └─> Clic "Commander"

5. VÉRIFICATION AUTHENTIFICATION
    ├─> Si non connecté → Redirection /connexion
    └─> Si connecté → Continue

6. COMMANDE
    ├─> Formulaire adresse de livraison
    ├─> Validation
    └─> Clic "Payer"

7. PAIEMENT STRIPE
    ├─> POST /api/payment/create-checkout-session (API)
    ├─> Redirection Stripe Checkout
    ├─> Paiement utilisateur
    ├─> Webhook → Backend crée la commande
    └─> Redirection /confirmation

8. CONFIRMATION
    ├─> Affichage confirmation
    ├─> Email de confirmation (Backend)
    ├─> Vidage du panier
    └─> Commande visible dans "Mes commandes"
```

🧩 Principes de développement

SOLID

1. **S**ingle Responsibility
    - Un composant = une responsabilité
    - Exemple : `Bouton.jsx` ne fait QUE afficher un bouton
2. **O**pen/Closed
    - Composants extensibles via props
    - Exemple : `<Bouton variante="primaire" />` ou `<Bouton variante="danger" />`
3. **L**iskov Substitution
    - Composants interchangeables
    - Exemple : Tous les boutons ont la même interface
4. **I**nterface Segregation
    - Props spécifiques et ciblées
    - Éviter les props géantes et complexes
5. **D**ependency Inversion
    - Services abstraits

- Composants dépendent de services, pas d'implémentation

Séparation des responsabilités

```
SERVICES (services/api/)
↓ Fournissent des données
HOOKS (hooks/)
↓ Gèrent la logique métier
COMPOSANTS CONTENEURS (pages/)
↓ Orchestrent les données
COMPOSANTS DE PRÉSENTATION (composants/)
↓ Affichent l'UI
```

DRY (Don't Repeat Yourself)

- Créez des composants réutilisables
- Utilisez des hooks personnalisés
- Centralisez les constantes
- Partagez les utilitaires

📦 Dépendances recommandées

Production

```
{
  "react": "^18.3.1",
  "react-dom": "^18.3.1",
  "react-router-dom": "^6.x",
  "axios": "^1.x",
  "@stripe/stripe-js": "^2.x",
  "@stripe/react-stripe-js": "^2.x",
  "react-icons": "^5.x",
  "date-fns": "^3.x"
}
```

Développement

```
{
  "@vitejs/plugin-react": "^4.x",
  "vite": "^5.x",
  "eslint": "^8.x",
  "prettier": "^3.x"
}
```

🚀 Commandes utiles

Développement

```
# Backend
cd Node
npm run dev          # Lance le serveur Node.js sur port 3000

# Frontend
cd React
npm run dev          # Lance Vite sur port 5173
```

Production

```
# Backend
cd Node
npm start            # Lance en mode production

# Frontend
cd React
npm run build        # Build les fichiers optimisés
npm run preview      # Prévisualiser le build
```

Autres

```
# Installer une nouvelle dépendance
npm install nom-du-package

# Installer une dépendance de développement
npm install --save-dev nom-du-package

# Mettre à jour les dépendances
npm update

# Vérifier les dépendances obsolètes
npm outdated
```

🎨 Charte graphique Porsche

Couleurs

- **Noir principal** : #000000
- **Rouge Porsche** : #d5001c
- **Or/Accent** : #c0a062

- **Gris clair** : `#f5f5f5`
- **Texte** : `#333333`

Typographie

- **Police principale** : Porsche Next (ou fallback : Arial, sans-serif)
- **Tailles** : 0.875rem (small), 1rem (base), 1.25rem (large), 2rem (titre)

Espacements

- **XS** : 0.25rem (4px)
- **SM** : 0.5rem (8px)
- **MD** : 1rem (16px)
- **LG** : 1.5rem (24px)
- **XL** : 2rem (32px)
- **XXL** : 3rem (48px)

📝 Conventions de nommage

Fichiers

- **Composants** : PascalCase (`Bouton.jsx`, `CarteVoiture.jsx`)
- **Hooks** : camelCase avec préfixe "use" (`useAuth.js`, `usePanier.js`)
- **Services** : camelCase avec suffixe "Service" (`authService.js`)
- **Utils** : camelCase (`formatage.js`, `validation.js`)
- **CSS** : Même nom que le composant (`Bouton.css`)

Variables JavaScript

- **Constantes** : SCREAMING_SNAKE_CASE (`API_URL`, `MAX_ITEMS`)
- **Variables** : camelCase (`utilisateur`, `listVoitures`)
- **Composants** : PascalCase (`MonComposant`)
- **Fonctions** : camelCase (`obtenirUtilisateur`, `calculerPrix`)

Classes CSS

- **BEM** recommandé :

```
.bouton {
} /* Block */
.bouton__texte {
} /* Element */
.bouton--primaire {
} /* Modifier */
```

🧪 Tests (optionnel mais recommandé)

Installation

```
npm install --save-dev vitest @testing-library/react @testing-
library/jest-dom
```

Exemple de test

```
// Bouton.test.jsx
import { render, screen } from "@testing-library/react";
import Bouton from "./Bouton";

test("affiche le texte du bouton", () => {
  render(<Bouton texte="Cliquez-moi" />);
  expect(screen.getByText("Cliquez-moi")).toBeInTheDocument();
});
```

🐛 Debugging

Outils recommandés

1. **React DevTools** (extension navigateur)
    - Inspecter les composants
    - Voir les props et state
    - Profiler les performances
2. **Console du navigateur**
    - `console.log()` pour déboguer
    - Network tab pour voir les requêtes API
    - Application tab pour voir localStorage
3. **VS Code**
    - Installer l'extension "ES7+ React/Redux/React-Native snippets"
    - Installer "Prettier" pour formater le code
    - Installer "ESLint" pour détecter les erreurs

Commandes de débogage

```
// Afficher une variable
console.log("Ma variable:", maVariable);

// Afficher un objet de manière structurée
console.table(monTableau);
```

```
// Timer pour mesurer la performance
console.time("Operation");
// ... code ...
console.timeEnd("Operation");

// Breakpoint dans le code
debugger; // Le navigateur s'arrêtera ici si DevTools ouvert
```

📊 Progression recommandée

Semaine 1-2 : Configuration et authentification

- [ ] Installation et configuration
- [ ] Services API de base
- [ ] Authentification (connexion/inscription)
- [ ] Layout (header, navigation, footer)

Semaine 3-4 : Catalogue

- [ ] Service voiture
- [ ] Liste des voitures
- [ ] Détails d'une voiture
- [ ] Filtres

Semaine 5-6 : Configurateur

- [ ] Service personnalisation
- [ ] Étapes de configuration
- [ ] Calcul de prix
- [ ] Récapitulatif

Semaine 7-8 : Panier et commande

- [ ] Gestion du panier
- [ ] Formulaire de commande
- [ ] Intégration Stripe
- [ ] Confirmation

Semaine 9-10 : Administration

- [ ] Tableau de bord admin
- [ ] CRUD voitures
- [ ] CRUD accessoires
- [ ] Gestion utilisateurs

Semaine 11-12 : Polish et tests

- [ ] Responsive design
- [ ] Animations
- [ ] Tests
- [ ] Optimisations

📞 Support

Documentation de référence

- **React** : https://react.dev
- **Vite** : https://vitejs.dev
- **React Router** : https://reactrouter.com
- **Axios** : https://axios-http.com
- **Stripe** : https://stripe.com/docs/stripe-js/react

Fichiers de support du projet

- `ARCHITECTURE_REACT.md` - Architecture détaillée
- `STRUCTURE_EXEMPLE.md` - Exemples et explications
- `GUIDE_DEMARRAGE.md` - Guide d'installation
- `EXEMPLES_CODE/` - Fichiers d'exemple

✅ Checklist finale

Configuration

- [ ] Node.js installé (v18+)
- [ ] Dépendances installées
- [ ] Variables d'environnement configurées
- [ ] Backend fonctionnel (port 3000)
- [ ] Frontend fonctionnel (port 5173)

Architecture

- [ ] Structure de dossiers créée
- [ ] Fichiers d'exemple copiés
- [ ] Configuration Axios
- [ ] Services de base créés
- [ ] Contextes créés
- [ ] Routes configurées

Tests

- [ ] Authentification fonctionne
- [ ] Requêtes API passent
- [ ] Routes protégées fonctionnent
- [ ] Pas d'erreurs dans la console

🎓 Conseils finaux

Pour les étudiants débutants

1. **Prenez votre temps**
    - Ne vous précipitez pas
    - Comprenez chaque concept avant de passer au suivant
2. **Pratiquez régulièrement**
    - Codez un peu chaque jour
    - Relisez votre code
3. **N'ayez pas peur d'expérimenter**
    - Testez des choses
    - Cassez le code pour comprendre comment il fonctionne
    - Git vous permet de revenir en arrière
4. **Demandez de l'aide**
    - Consultez la documentation
    - Cherchez sur Google/Stack Overflow
    - Demandez à un mentor
5. **Commentez votre code**
    - Expliquez ce que fait votre code
    - Votre "vous" du futur vous remerciera

Ressources d'apprentissage

- **FreeCodeCamp** : https://www.freecodecamp.org
- **MDN Web Docs** : https://developer.mozilla.org
- **JavaScript.info** : https://javascript.info
- **CSS-Tricks** : https://css-tricks.com

Bonne chance pour votre projet ! 🚗💨

Développé avec ❤️ pour les étudiants en développement web