Guide d'Utilisation Complète - Riziky-Boutic

Introduction au Guide

Ce guide complet vous accompagne dans l'utilisation de la plateforme Riziky-Boutic, que vous soyez utilisateur final, administrateur, ou développeur. Il couvre toutes les fonctionnalités disponibles avec des instructions détaillées et des exemples pratiques.

Pour les Clients - Guide Utilisateur Final

Premiers Pas sur la Plateforme

Création de Compte et Connexion 1. Inscription :

```
Étapes d'inscription :
1. Cliquez sur "S'inscrire" dans le menu principal
2. Remplissez le formulaire avec vos informations :
   - Nom et prénom
   - Adresse email valide
   - Mot de passe sécurisé (8 caractères minimum)
   - Numéro de téléphone
   - Genre (optionnel)
3. Acceptez les conditions d'utilisation et la politique de confidentialité
4. Cliquez sur "Créer mon compte"
5. Vérifiez votre email et confirmez votre compte

Conseils pour un mot de passe sécurisé :
- Minimum 8 caractères
- Mélange de majuscules et minuscules
- Inclure des chiffres et symboles
- Éviter les informations personnelles
```

2. Connexion :

```
Pour vous connecter :
1. Cliquez sur "Connexion" dans le menu
2. Saisissez votre email et mot de passe
3. Cochez "Se souvenir de moi" pour rester connecté
4. Cliquez sur "Se connecter"

En cas d'oubli de mot de passe :
1. Cliquez sur "Mot de passe oublié ?"
2. Saisissez votre adresse email
3. Consultez votre boîte mail pour le lien de réinitialisation
4. Suivez les instructions pour créer un nouveau mot de passe
```

Navigation sur le Site Menu Principal : - **Accueil** : Page d'accueil avec les dernières nouveautés - **Produits** : Catalogue complet avec filtres et recherche - **Nouveautés** : Derniers produits ajoutés - **Promotions** : Ventes flash et offres spéciales - **Contact** : Formulaire de contact et informations

Barre de Recherche :

```
Utilisation de la recherche :
1. Cliquez sur l'icône de recherche ou tapez Ctrl+K
2. Tapez le nom du produit, catégorie, ou mots-clés
3. Utilisez les suggestions automatiques
4. Appuyez sur Entrée pour lancer la recherche

Astuces de recherche :
- "chaussures nike" : recherche exacte
- "t-shirt rouge" : recherche par couleur
- "sous 50€" : recherche par prix
```

Shopping et Achat

Parcourir les Produits 1. Navigation par Catégories :

```
Accès aux catégories :
- Menu déroulant "Catégories" si plus de 8 catégories
- Liens directs dans le menu si moins de 8 catégories
- Page dédiée avec sous-catégories

Hiérarchie des catégories :
Vêtements
  Homme
     T-shirts
     Pantalons
     Chaussures
  Femme
      Robes
      Tops
      Accessoires
```

2. Filtrage des Produits :

```
Filtres disponibles :
- Prix : Fourchette minimum-maximum
- Catégorie : Sélection multiple
- Disponibilité : En stock uniquement
- Marque : Sélection des marques
- Taille : Selon le type de produit
- Couleur : Palette de couleurs
```

```
Application des filtres :
1. Utilisez les filtres dans la sidebar gauche
2. Les résultats se mettent à jour automatiquement
3. Supprimez les filtres via les badges affichés
4. Réinitialisez tous les filtres d'un clic
```

3. Tri des Résultats :

```
Options de tri :
- Pertinence (par défaut)
- Prix croissant
- Prix décroissant
- Popularité
- Nouveauté
- Meilleures notes

Changement de tri :
1. Utilisez le menu déroulant en haut à droite
2. Les résultats se réorganisent automatiquement
```

Fiche Produit Détaillée Informations Disponibles :

```
Sur chaque fiche produit :
- Galerie d'images (zoom et navigation)
- Nom et description détaillée
- Prix avec promotions éventuelles
- Stock disponible en temps réel
- Variantes (taille, couleur) si applicable
- Avis clients et notes
- Produits similaires recommandés
- Informations de livraison

Actions possibles :
- Ajouter au panier (avec sélection quantité)
- Ajouter aux favoris
- Partager sur les réseaux sociaux
- Comparer avec d'autres produits
- Poser une question via chat
```

Navigation dans les Images :

```
Fonctionnalités de la galerie :
- Clic sur les miniatures pour changer l'image principale
- Survol pour zoom automatique
- Clic sur l'image principale pour plein écran
- Navigation par flèches gauche/droite
- Zoom molette de la souris en plein écran
```

Gestion du Panier 1. Ajouter des Produits :

```
Méthodes d'ajout :
- Depuis la fiche produit : bouton "Ajouter au panier"
- Depuis la grille : bouton rapide sur survol
- Vue rapide : modal avec ajout direct
- Recommandations : ajout en un clic

Sélection des options :
1. Choisir la taille (si applicable)
2. Sélectionner la couleur (si disponible)
3. Indiquer la quantité souhaitée
4. Vérifier le stock disponible
5. Cliquer sur "Ajouter au panier"
```

2. Gestion du Panier :

```
Actions dans le panier :
- Modifier les quantités avec +/-
- Supprimer un article (icône poubelle)
- Vider complètement le panier
- Sauvegarder pour plus tard
- Appliquer un code promo

Informations affichées :
- Image et nom du produit
- Prix unitaire et total
- Quantité sélectionnée
- Stock restant
- Sous-total par article
- Total général avec taxes
- Frais de livraison estimés
```

3. Codes Promotionnels :

```
Utilisation des codes promo :
1. Dans le panier, cherchez "Code promotionnel"
2. Saisissez votre code dans le champ prévu
3. Cliquez sur "Appliquer"
4. La réduction s'applique automatiquement
5. Le nouveau total s'affiche

Types de réductions :
- Pourcentage : -20% sur votre commande
- Montant fixe : -10€ de réduction
- Livraison gratuite : frais de port offerts
- Produit offert : cadeau automatique
```

Processus de Commande 1. Étapes du Checkout :

```
Processus en 6 étapes :
1. Connexion / Inscription (si nécessaire)
2. Révision du panier et modifications
3. Adresse de livraison
4. Mode de livraison et délais
5. Méthode de paiement
6. Confirmation et validation

Navigation :
- Boutons "Précédent" et "Suivant"
- Barre de progression en haut
- Possibilité de revenir au panier
- Sauvegarde automatique à chaque étape
```

2. Adresses de Livraison :

```
Gestion des adresses :
- Sélection d'une adresse existante
- Ajout d'une nouvelle adresse
- Modification d'une adresse
- Adresse de facturation différente

Informations requises :
- Nom et prénom du destinataire
- Adresse complète (rue, ville, code postal)
- Pays (France par défaut)
- Téléphone pour la livraison
- Instructions spéciales (optionnel)
```

3. Modes de Livraison :

```
Options disponibles :
- Standard (3-5 jours ouvrés) : 5,99€
- Express (1-2 jours ouvrés) : 9,99€
- Gratuite (5-7 jours, commande >50€) : 0€
- Point relais (3-5 jours) : 3,99€
- Retrait magasin (si disponible) : 0€

Estimation de livraison :
- Délais mis à jour selon l'adresse
- Dates de livraison prévisionnelle
- Suivi de colis inclus
- Assurance perte/vol (optionnelle)
```

4. Méthodes de Paiement :

```
Paiements acceptés :
```

```
- Cartes bancaires (Visa, Mastercard, Amex)
- PayPal (compte existant)
- Apple Pay (sur Safari/iOS)
- Google Pay (sur Chrome/Android)
- Virement bancaire (délai 2-3 jours)
- Paiement en 3 fois (selon montant)

Sécurité des paiements :
- Chiffrement SSL 256 bits
- 3D Secure pour les cartes
- Aucune conservation des données bancaires
- Conformité PCI-DSS
```

Fonctionnalités Personnelles

Liste de Favoris Gestion des Favoris :

```
Ajouter aux favoris :
1. Sur la fiche produit : cliquez sur l'icône cœur
2. Depuis la grille : icône cœur au survol
3. Le cœur se remplit et change de couleur
4. Notification de confirmation

Gérer les favoris :
- Accès via menu utilisateur ou icône cœur navbar
- Vue liste ou grille des favoris
- Tri par date d'ajout, prix, catégorie
- Actions : supprimer, ajouter au panier, partager
- Liste publique ou privée (paramètres)

Notifications favoris :
- Alerte si produit en promotion
- Notification de réapprovisionnement
- Rappel produits oubliés (hebdomadaire)
```

Avis et Commentaires Laisser un Avis :

```
Conditions pour évaluer :
- Avoir acheté le produit
- Commande livrée depuis plus de 48h
- Un seul avis par produit et par client

Processus d'évaluation :
1. Depuis l'historique des commandes
2. Clic sur "Laisser un avis"
3. Attribution d'une note (1 à 5 étoiles)
4. Rédaction du commentaire (optionnel)
```

```
5. Upload de photos (optionnel)
6. Soumission pour modération

Contenu de l'avis :
- Note globale (obligatoire)
- Qualité du produit (optionnel)
- Rapport qualité/prix (optionnel)
- Commentaire libre (500 caractères max)
- Photos du produit reçu
```

Consulter les Avis :

```
Sur chaque fiche produit :
- Note moyenne affichée
- Histogramme de répartition des notes
- Avis les plus récents
- Filtrage par note (5 étoiles, 4 étoiles, etc.)
- Photos clients dans les avis
- Réponses du vendeur si applicable

Utilité des avis :
- Boutons "Utile" / "Pas utile"
- Signalement d'avis inappropriés
- Tri par pertinence, date, note
```

Suivi des Commandes

Historique des Commandes Accès à l'Historique :

```
Navigation :
1. Menu utilisateur > "Mes commandes"
2. Ou directement via /orders
3. Liste de toutes vos commandes

Informations par commande :
- Numéro de commande unique
- Date et heure de passage
- Statut actuel de la commande
- Montant total payé
- Mode de livraison choisi
- Adresse de livraison
- Détail des articles commandés
```

Statuts des Commandes :

```
Cycle de vie d'une commande :
1. "Confirmée" : Paiement validé, en attente de traitement
2. "En préparation" : Commande en cours de préparation
```

```
3. "Expédiée" : Colis remis au transporteur
4. "En transit" : En cours de livraison
5. "Livrée" : Colis remis au destinataire
6. "Terminée" : Délai de retour dépassé

Actions possibles selon le statut :
- Confirmée : Modification ou annulation
- En préparation : Annulation possible
- Expédiée : Suivi de colis
- Livrée : Signaler un problème, laisser un avis
```

Suivi en Temps Réel Notifications Push :

```
Notifications automatiques :
- Confirmation de commande (immédiat)
- Paiement validé (2-5 minutes)
- Commande en préparation (24-48h)
- Expédition avec numéro de suivi (24-72h)
- Colis en cours de livraison (temps réel)
- Livraison effectuée (immédiat)

Paramétrage :
- Email : activé par défaut
- SMS : optionnel (numéro requis)
- Push navigateur : après autorisation
- Notifications in-app : toujours activées
```

Suivi de Colis :

```
Informations de suivi :
- Numéro de tracking unique
- Transporteur (Colissimo, Chronopost, etc.)
- Géolocalisation du colis (si disponible)
- Historique des événements
- Estimation de livraison mise à jour
- Contact du transporteur

Accès au suivi :
1. Email d'expédition avec lien direct
2. Historique commandes > "Suivre le colis"
3. Numéro de suivi sur site transporteur
4. Widget de suivi intégré
```

Service Client

Chat en Direct Accès au Chat :

```
Initier une conversation :
```

```
1. Widget de chat en bas à droite
2. Bouton "Aide" dans le menu
3. Depuis une fiche de commande
4. Page de contact

Informations à préparer :
- Numéro de commande (si applicable)
- Description précise du problème
- Captures d'écran si nécessaire
- Coordonnées de contact
```

Utilisation du Chat :

```
Fonctionnalités disponibles :
- Messages texte en temps réel
- Envoi de fichiers et images
- Partage d'écran (si nécessaire)
- Historique des conversations
- Transfert vers un spécialiste
- Évaluation de satisfaction

Horaires du support :
- Lundi-Vendredi : 9h-18h
- Samedi : 10h-16h
- Dimanche : Chat automatique uniquement
- Réponse sous 24h garantie
```

Centre d'Aide FAQ Intégrée :

```
Catégories d'aide :
- Compte et connexion
- Commandes et livraisons
- Paiements et facturation
- Retours et remboursements
- Produits et stock
- Technique et navigation

Recherche dans l'aide :
1. Tapez votre question dans la barre de recherche
2. Parcourez les suggestions automatiques
3. Consultez les articles pertinents
4. Utilisez les liens "Ceci vous a-t-il aidé ?"
```

Pour les Administrateurs - Guide de Gestion

Accès Administration

Connexion Administrateur Prérequis :

```
Compte administrateur requis :
- Role "admin" dans la base de données
- Permissions d'accès au panel admin
- Authentification renforcée (2FA recommandé)

Accès au panel :
1. Connexion avec compte admin
2. Navigation vers /admin
3. Ou menu utilisateur > "Administration"
```

Interface d'Administration :

```
Sections disponibles :
- Tableau de bord : Métriques et aperçu
- Produits : Gestion du catalogue
- Commandes : Traitement des ventes
- Clients : Gestion des utilisateurs
- Marketing : Promotions et codes promo
- Paramètres : Configuration du site
- Rapports : Analytics et statistiques
```

Tableau de Bord

Métriques Temps Réel Indicateurs Clés :

```
KPIs principaux :
- Chiffre d'affaires (jour/semaine/mois)
- Nombre de commandes en cours
- Visiteurs actifs simultanés
- Taux de conversion global
- Panier moyen
- Produits les plus vendus

Graphiques disponibles :
- Évolution des ventes (courbe temporelle)
- Répartition par catégories (camembert)
- Performance par source de trafic
- Analyse de cohorte des clients
- Entonnoir de conversion
```

Alertes et Notifications :

```
Alertes automatiques :
- Stock faible (< 10 unités)
```

```
- Commande en attente (> 24h)
- Retour client demandé
- Avis négatif publié
- Paiement échoué
- Erreur technique détectée

Configuration des alertes :
1. Paramètres > Notifications
2. Définir les seuils d'alerte
3. Choisir les moyens de notification
4. Tester les alertes configurées
```

Gestion des Produits

Création de Produit Formulaire Complet :

```
Informations obligatoires :
- Nom du produit (unique)
- Description courte et détaillée
- Prix de vente (décimal)
- Catégorie principale
- Stock initial
- Référence/SKU (auto-générée)

Informations optionnelles :
- Images du produit (jusqu'à 10)
- Variantes (taille, couleur)
- Poids et dimensions
- Marque et modèle
- Tags SEO
- Produits liés/accessoires
```

Upload d'Images :

```
Processus d'ajout d'images :
1. Glisser-déposer ou clic pour sélectionner
2. Formats acceptés : JPG, PNG, WebP
3. Taille max : 5MB par image
4. Recommandé : 800x600px minimum
5. Optimisation automatique
6. Réorganisation par drag & drop

Bonnes pratiques :
- Première image = image principale
- Fond blanc ou neutre recommandé
- Plusieurs angles de vue
- Zoom sur les détails importants
```

```
- Images de contexte d'utilisation
```

Gestion des Stocks Suivi en Temps Réel :

```
Informations de stock :
- Quantité actuelle en stock
- Quantité réservée (commandes en cours)
- Stock disponible à la vente
- Historique des mouvements
- Seuil d'alerte personnalisable
- Prévision de rupture

Actions sur le stock :
- Ajustement manuel du stock
- Réception de marchandises
- Retour produit en stock
- Dépréciation/perte déclarée
- Transfert entre entrepôts (si applicable)
```

Approvisionnement :

```
Processus de réapprovisionnement :
1. Identification des produits en stock faible
2. Analyse de la demande historique
3. Calcul des quantités à commander
4. Génération de bons de commande
5. Suivi des livraisons fournisseurs
6. Mise à jour des stocks à réception

Alertes automatiques :
- Stock critique (< seuil défini)
- Produit en rupture
- Commande fournisseur en retard
- Écart d'inventaire détecté
```

Promotions et Ventes Flash Création de Promotion :

```
Types de promotions :
- Réduction pourcentage (-20%)
- Réduction montant fixe (-10€)
- Deuxième article offert
- Livraison gratuite
- Lot/Bundle avec remise
- Cadeau avec achat

Paramétrage :
1. Sélection des produits concernés
```

```
2. Type et valeur de la réduction
3. Dates de début et fin
4. Conditions d'application
5. Limite d'utilisation (optionnel)
6. Code promo associé (optionnel)
```

Ventes Flash :

```
Configuration d'une vente flash :
1. Produits en promotion
2. Durée limitée (24-72h max)
3. Stock limité pour l'offre
4. Compte à rebours visible
5. Bannière promotionnelle
6. Notifications push aux clients

Gestion pendant la vente :
- Suivi des ventes en temps réel
- Ajustement du stock si nécessaire
- Prolongation si objectif non atteint
- Communication sur les réseaux sociaux
```

Gestion des Commandes

Traitement des Commandes Workflow Standard :

```
Étapes de traitement :
1. Réception commande (automatique)
2. Vérification paiement (2-5 min)
3. Validation stock (immédiat)
4. Préparation commande (24-48h)
5. Génération étiquette (automatique)
6. Expédition (scan colis)
7. Suivi livraison (temps réel)
8. Confirmation réception (client)

Actions administrateur :
- Valider manuellement si paiement suspect
- Modifier l'adresse avant expédition
- Ajouter des notes internes
- Contacter le client si nécessaire
- Gérer les exceptions et problèmes
```

États des Commandes :

```
Statuts et actions possibles :
- "En attente paiement" : Relancer client, annuler
- "Confirmée" : Préparer, modifier, annuler
```

```
- "En préparation" : Marquer prête, annuler
- "Prête" : Expédier, modifier transporteur
- "Expédiée" : Suivre, gérer incidents
- "Livrée" : Clôturer, traiter retours
- "Annulée" : Rembourser, analyser causes

Filtres de recherche :
- Par statut de commande
- Par date de commande
- Par montant (min/max)
- Par client (nom, email)
- Par produit commandé
- Par transporteur
```

Gestion des Remboursements Processus de Remboursement :

```
Types de remboursements :
- Annulation client (avant expédition)
- Retour produit (dans les 14 jours)
- Produit défectueux
- Erreur de livraison
- Geste commercial

Étapes du remboursement :
1. Demande client ou décision admin
2. Vérification des conditions
3. Validation du retour (si applicable)
4. Calcul du montant à rembourser
5. Traitement du remboursement
6. Notification client
7. Mise à jour des comptes
```

Gestion des Clients

Base de Données Clients Informations Client :

```
Profil complet :
- Données personnelles (nom, email, téléphone)
- Adresses de livraison et facturation
- Historique des commandes
- Montant total dépensé
- Fréquence d'achat
- Produits favoris
- Avis laissés
- Tickets de support

Segmentation automatique :
```

```
- Nouveaux clients (première commande)
- Clients fidèles (3+ commandes)
- Clients VIP (montant élevé)
- Clients inactifs (pas d'achat 6 mois)
- Clients à risque (avis négatifs)
```

Communication Client :

```
Outils de communication :
- Email personnalisé
- SMS (si numéro fourni)
- Notification in-app
- Chat direct depuis l'admin
- Courrier postal (adresse validée)

Templates d'email :
- Confirmation de commande
- Notification d'expédition
- Relance panier abandonné
- Demande d'avis après livraison
- Offres personnalisées
- Newsletter produits
```

Service Client Intégré Centre de Support :

```
Gestion des tickets :
- Vue d'ensemble des conversations
- Attribution automatique ou manuelle
- Niveaux de priorité (bas, normal, urgent)
- Délais de réponse (SLA)
- Historique complet des échanges
- Notes internes entre agents

Réponses rapides :
- Templates de réponses courantes
- Snippets de texte réutilisables
- Procédures standardisées
- Base de connaissances interne
- FAQ automatisée
```

Pour les Développeurs - Guide Technique

Installation et Configuration

Environnement de Développement Prérequis Techniques :

```
# Versions requises
Node.js >= 18.0.0
npm >= 8.0.0 ou yarn >= 1.22.0
Git >= 2.30.0

# Vérification des versions
node --version
npm --version
git --version
```

Installation Complète :

```
# 1. Cloner le repository
git clone https://github.com/votre-repo/riziky-boutic.git
cd riziky-boutic

# 2. Installer les dépendances frontend
npm install

# 3. Installer les dépendances backend
cd server
npm install
cd ..

# 4. Configuration de l'environnement
cp .env.example .env
# Éditer les variables d'environnement

# 5. Initialiser la base de données (si première installation)
cd server
npm run init-db
cd ..
```

Variables d'Environnement :

```
# Fichier .env à créer à la racine
VITE_API_BASE_URL=http://localhost:10000
NODE_ENV=development
JWT_SECRET=your_super_secret_jwt_key_here
JWT_EXPIRES_IN=24h
JWT_REFRESH_EXPIRES_IN=7d
UPLOAD_MAX_SIZE=5242880
CORS_ORIGIN=http://localhost:8080
RATE_LIMIT_WINDOW=900000
RATE_LIMIT_MAX=100
```

Structure de Développement Scripts de Développement :

```
# Démarrage en mode développement
npm run dev          # Frontend uniquement
npm run server:dev   # Backend uniquement
npm run dev:full     # Frontend + Backend simultanément

# Build de production
npm run build        # Build frontend optimisé
npm run server:prod  # Démarrage backend production

# Tests
npm run test         # Tests unitaires frontend
npm run test:server  # Tests backend
npm run test:e2e     # Tests end-to-end

# Linting et formatage
npm run lint         # ESLint check
npm run lint:fix     # Fix automatique
npm run format       # Prettier
npm run type-check   # Vérification TypeScript
```

Architecture et Patterns

Patterns Frontend Structure des Composants :

```
// Pattern de composant réutilisable
interface ComponentProps {
  // Props requises
  data: DataType;
  onAction: (id: string) => void;

  // Props optionnelles avec defaults
  variant?: 'default' | 'compact';
  showActions?: boolean;
  className?: string;

  // Props enfants
  children?: React.ReactNode;
}

const Component: FC<ComponentProps> = ({
  data,
  onAction,
  variant = 'default',
  showActions = true,
  className,
  children
```

```
}) => {
  // Hooks en premier
  const [state, setState] = useState();
  const { customHook } = useCustomHook();

  // Handlers
  const handleAction = useCallback((id: string) => {
    onAction(id);
  }, [onAction]);

  // Render
  return (
    <div className={cn('base-classes', className)}>
      {children}
    </div>
  );
};
```

Hooks Personnalisés :

```
// Pattern de hook métier
interface UseFeatureOptions {
  enableCache?: boolean;
  autoRefresh?: boolean;
}

interface UseFeatureReturn {
  data: DataType[];
  isLoading: boolean;
  error: Error | null;
  refetch: () => Promise<void>;
  // Actions
  create: (data: CreateData) => Promise<DataType>;
  update: (id: string, data: UpdateData) => Promise<DataType>;
  delete: (id: string) => Promise<void>;
}

const useFeature = (options: UseFeatureOptions = {}): UseFeatureReturn => {
  const { enableCache = true, autoRefresh = false } = options;

  // État local
  const [data, setData] = useState<DataType[]>([]);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState<Error | null>(null);

  // Logique métier
  const fetchData = useCallback(async () => {
```

```
    setIsLoading(true);
    try {
      const result = await api.getData();
      setData(result);
      setError(null);
    } catch (err) {
      setError(err as Error);
    } finally {
      setIsLoading(false);
    }
  }, []);

  // Actions CRUD
  const create = useCallback(async (createData: CreateData) => {
    const newItem = await api.create(createData);
    setData(prev => [...prev, newItem]);
    return newItem;
  }, []);

  // Auto-refresh si activé
  useEffect(() => {
    if (autoRefresh) {
      const interval = setInterval(fetchData, 30000); // 30s
      return () => clearInterval(interval);
    }
  }, [autoRefresh, fetchData]);

  return {
    data,
    isLoading,
    error,
    refetch: fetchData,
    create,
    update,
    delete
  };
};
```

Patterns Backend Structure des Controllers :

```
// Pattern de controller REST
class ProductController {
  // GET /api/products
  async getProducts(req, res) {
    try {
      const { page = 1, limit = 12, ...filters } = req.query;
```

```
    // Validation des paramètres
    const validatedFilters = this.validateFilters(filters);

    // Logique métier via service
    const result = await ProductService.getProducts({
      page: parseInt(page),
      limit: parseInt(limit),
      ...validatedFilters
    });

    // Response formatée
    res.json({
      success: true,
      data: result.products,
      pagination: result.pagination,
      meta: {
        total: result.total,
        page: result.page,
        limit: result.limit
      }
    });

  } catch (error) {
    this.handleError(res, error);
  }
}

// POST /api/products
async createProduct(req, res) {
  try {
    // Validation des données
    const validationResult = await this.validateProductData(req.body);
    if (!validationResult.isValid) {
      return res.status(400).json({
        success: false,
        error: 'Données invalides',
        details: validationResult.errors
      });
    }

    // Traitement des fichiers uploadés
    const images = req.files?.map(file => ({
      url: `/uploads/${file.filename}`,
      alt: req.body.nom
    })) || [];
```

```
      // Création via service
      const product = await ProductService.createProduct({
        ...req.body,
        images,
        createdBy: req.user.id
      });

      res.status(201).json({
        success: true,
        data: product,
        message: 'Produit créé avec succès'
      });

    } catch (error) {
      this.handleError(res, error);
    }
  }

  // Gestion centralisée des erreurs
  handleError(res, error) {
    console.error('Controller error:', error);

    if (error.name === 'ValidationError') {
      return res.status(400).json({
        success: false,
        error: 'Données invalides',
        details: error.details
      });
    }

    if (error.name === 'NotFoundError') {
      return res.status(404).json({
        success: false,
        error: 'Ressource non trouvée'
      });
    }

    res.status(500).json({
      success: false,
      error: 'Erreur interne du serveur'
    });
  }
}
```

Sécurité et Bonnes Pratiques

Sécurisation Frontend Validation des Données :

```
// Schémas Zod pour validation
const ProductSchema = z.object({
  nom: z.string()
    .min(3, 'Nom trop court')
    .max(100, 'Nom trop long')
    .refine(val => !/<script/i.test(val), 'Contenu non autorisé'),

  prix: z.number()
    .positive('Prix doit être positif')
    .max(99999, 'Prix trop élevé'),

  description: z.string()
    .max(1000, 'Description trop longue')
    .optional(),

  category: z.string()
    .uuid('Catégorie invalide'),

  stock: z.number()
    .int('Stock doit être entier')
    .min(0, 'Stock ne peut pas être négatif')
});

// Utilisation dans un formulaire
const form = useForm<ProductFormData>({
  resolver: zodResolver(ProductSchema),
  mode: 'onChange' // Validation en temps réel
});
```

Protection XSS :

```
// Sanitisation des entrées utilisateur
import DOMPurify from 'dompurify';

const sanitizeHtml = (html: string): string => {
  return DOMPurify.sanitize(html, {
    ALLOWED_TAGS: ['b', 'i', 'u', 'strong', 'em'],
    ALLOWED_ATTR: []
  });
};

// Composant d'affichage sécurisé
const SafeHTML: FC<{ content: string }> = ({ content }) => {
```

```
  const sanitized = useMemo(() => sanitizeHtml(content), [content]);

  return (
    <div dangerouslySetInnerHTML={{ __html: sanitized }} />
  );
};
```

Sécurisation Backend Middleware de Sécurité :

```
const securityMiddleware = (req, res, next) => {
  // Headers de sécurité
  res.setHeader('X-Content-Type-Options', 'nosniff');
  res.setHeader('X-Frame-Options', 'DENY');
  res.setHeader('X-XSS-Protection', '1; mode=block');
  res.setHeader('Referrer-Policy', 'strict-origin-when-cross-origin');

  // Validation de l'origine
  const allowedOrigins = process.env.CORS_ORIGIN?.split(',') || [];
  const origin = req.headers.origin;

  if (origin && !allowedOrigins.includes(origin)) {
    return res.status(403).json({ error: 'Origine non autorisée' });
  }

  next();
};
```

Validation et Sanitisation :

```
const { body, validationResult } = require('express-validator');
const xss = require('xss');

const productValidationRules = [
  body('nom')
    .trim()
    .isLength({ min: 3, max: 100 })
    .escape()
    .customSanitizer(value => xss(value)),

  body('prix')
    .isFloat({ min: 0.01, max: 99999 })
    .toFloat(),

  body('description')
    .optional()
    .isLength({ max: 1000 })
    .customSanitizer(value => xss(value)),
```

```
  body('stock')
    .isInt({ min: 0 })
    .toInt()
];

const validateRequest = (req, res, next) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({
      success: false,
      error: 'Données invalides',
      details: errors.array()
    });
  }

  next();
};
```

Tests et Quality Assurance

Tests Frontend Tests de Composants :

```
// Test unitaire avec React Testing Library
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import { ProductCard } from '../ProductCard';

const mockProduct = {
  id: '1',
  nom: 'Produit Test',
  prix: 29.99,
  stock: 10,
  images: ['/test-image.jpg']
};

const mockAddToCart = jest.fn();
jest.mock('../hooks/useCart', () => ({
  useCart: () => ({
    addToCart: mockAddToCart,
    isLoading: false
  })
}));

describe('ProductCard', () => {
```

```
  beforeEach(() => {
    mockAddToCart.mockClear();
  });

  it('affiche les informations du produit', () => {
    render(<ProductCard product={mockProduct} />);

    expect(screen.getByText('Produit Test')).toBeInTheDocument();
    expect(screen.getByText('29,99€')).toBeInTheDocument();
    expect(screen.getByRole('img', { name: 'Produit Test' })).toBeInTheDocument();
  });

  it('ajoute le produit au panier au clic', async () => {
    const user = userEvent.setup();
    render(<ProductCard product={mockProduct} />);

    const addButton = screen.getByRole('button', { name: /ajouter au panier/i });
    await user.click(addButton);

    await waitFor(() => {
      expect(mockAddToCart).toHaveBeenCalledWith('1', 1);
    });
  });

  it('désactive le bouton si produit en rupture', () => {
    const outOfStock = { ...mockProduct, stock: 0 };
    render(<ProductCard product={outOfStock} />);

    const addButton = screen.getByRole('button', { name: /indisponible/i });
    expect(addButton).toBeDisabled();
  });
});
```

Tests d'Intégration :

```
// Test d'intégration avec API mock
import { rest } from 'msw';
import { setupServer } from 'msw/node';
import { render, screen, waitFor } from '@testing-library/react';
import { ProductList } from '../ProductList';

const server = setupServer(
  rest.get('/api/products', (req, res, ctx) => {
    return res(ctx.json({
      products: [mockProduct],
      pagination: { total: 1, page: 1 }
    }));
```

```
  })
);

beforeAll(() => server.listen());
afterEach(() => server.resetHandlers());
afterAll(() => server.close());

describe('ProductList Integration', () => {
  it('charge et affiche les produits depuis l\'API', async () => {
    render(<ProductList />);

    expect(screen.getByText(/chargement/i)).toBeInTheDocument();

    await waitFor(() => {
      expect(screen.getByText('Produit Test')).toBeInTheDocument();
    });

    expect(screen.queryByText(/chargement/i)).not.toBeInTheDocument();
  });
});
```

Tests Backend Tests d'API :

```
// Test des endpoints avec Supertest
const request = require('supertest');
const app = require('../app');

describe('Products API', () => {
  describe('GET /api/products', () => {
    it('retourne la liste des produits', async () => {
      const response = await request(app)
        .get('/api/products')
        .expect(200);

      expect(response.body.success).toBe(true);
      expect(Array.isArray(response.body.data)).toBe(true);
      expect(response.body.pagination).toBeDefined();
    });

    it('filtre les produits par catégorie', async () => {
      const response = await request(app)
        .get('/api/products?category=vetements')
        .expect(200);

      response.body.data.forEach(product => {
        expect(product.category).toBe('vetements');
```

```
    });
  });
});

describe('POST /api/products', () => {
  it('crée un nouveau produit avec auth admin', async () => {
    const productData = {
      nom: 'Nouveau Produit',
      prix: 49.99,
      stock: 100,
      category: 'test'
    };

    const response = await request(app)
      .post('/api/products')
      .set('Authorization', `Bearer ${adminToken}`)
      .send(productData)
      .expect(201);

    expect(response.body.success).toBe(true);
    expect(response.body.data.nom).toBe(productData.nom);
  });

  it('rejette la création sans authentification', async () => {
    const productData = { nom: 'Test' };

    await request(app)
      .post('/api/products')
      .send(productData)
      .expect(401);
  });
});
});
```

Performance et Optimisation

Optimisations Frontend Code Splitting et Lazy Loading :

```
// Lazy loading des pages
const LazyAdminDashboard = lazy(() => import('../pages/admin/AdminDashboard'));
const LazyProductDetail = lazy(() => import('../pages/ProductDetail'));

// Route avec Suspense
<Route
  path="/admin"
  element={
```

```
    <Suspense fallback={<LoadingSpinner />}>
      <LazyAdminDashboard />
    </Suspense>
  }
/>

// Lazy loading conditionnel
const LazyChartComponent = lazy(() =>
  import('../components/charts/AdvancedChart').then(module => ({
    default: module.AdvancedChart
  }))
);

const Dashboard = () => {
  const [showAdvancedCharts, setShowAdvancedCharts] = useState(false);

  return (
    <div>
      <Button onClick={() => setShowAdvancedCharts(true)}>
        Afficher graphiques avancés
      </Button>

      {showAdvancedCharts && (
        <Suspense fallback={<ChartSkeleton />}>
          <LazyChartComponent />
        </Suspense>
      )}
    </div>
  );
};
```

Optimisation des Images :

```
// Composant Image optimisé
interface OptimizedImageProps {
  src: string;
  alt: string;
  width?: number;
  height?: number;
  quality?: number;
  loading?: 'lazy' | 'eager';
}

const OptimizedImage: FC<OptimizedImageProps> = ({
  src,
  alt,
  width,
```

```
  height,
  quality = 80,
  loading = 'lazy'
}) => {
  const [isLoaded, setIsLoaded] = useState(false);
  const [hasError, setHasError] = useState(false);

  // Génération des sources responsive
  const srcSet = useMemo(() => {
    if (!width) return undefined;

    return [
      `${src}?w=${width}&q=${quality} ${width}w`,
      `${src}?w=${width * 2}&q=${quality} ${width * 2}w`
    ].join(', ');
  }, [src, width, quality]);

  return (
    <div className="relative overflow-hidden bg-gray-100">
      {!isLoaded && !hasError && (
        <div className="absolute inset-0 animate-pulse bg-gray-200" />
      )}

      <img
        src={`${src}?w=${width}&h=${height}&q=${quality}`}
        srcSet={srcSet}
        sizes={`(max-width: ${width}px) 100vw, ${width}px`}
        alt={alt}
        loading={loading}
        className={cn(
          "transition-opacity duration-300",
          isLoaded ? "opacity-100" : "opacity-0"
        )}
        onLoad={() => setIsLoaded(true)}
        onError={() => setHasError(true)}
      />

      {hasError && (
        <div className="absolute inset-0 flex items-center justify-center bg-gray-100">
          <ImageIcon className="h-8 w-8 text-gray-400" />
        </div>
      )}
    </div>
  );
};
```

Optimisations Backend Cache Multi-niveaux :

```
const NodeCache = require('node-cache');

class CacheManager {
  constructor() {
    // Cache en mémoire (TTL par défaut: 5 minutes)
    this.memoryCache = new NodeCache({
      stdTTL: 300,
      checkperiod: 60,
      useClones: false
    });
  }

  async get(key, fetcher, ttl = 300) {
    // 1. Vérifier le cache mémoire
    const cached = this.memoryCache.get(key);
    if (cached !== undefined) {
      return cached;
    }

    // 2. Exécuter le fetcher
    const data = await fetcher();

    // 3. Stocker en cache
    this.memoryCache.set(key, data, ttl);

    return data;
  }

  invalidate(pattern) {
    const keys = this.memoryCache.keys();
    const toDelete = keys.filter(key => key.includes(pattern));
    this.memoryCache.del(toDelete);
  }
}

// Utilisation dans un service
class ProductService {
  constructor() {
    this.cache = new CacheManager();
  }

  async getProducts(filters) {
    const cacheKey = `products:${JSON.stringify(filters)}`;
```

```
    return this.cache.get(cacheKey, async () => {
      return this.fetchProductsFromDB(filters);
    }, 600); // Cache 10 minutes
  }

  async updateProduct(id, data) {
    const product = await this.updateProductInDB(id, data);

    // Invalider les caches liés
    this.cache.invalidate(`products:`);
    this.cache.invalidate(`product:${id}`);

    return product;
  }
}
```

Monitoring et Observabilité

Logging Structuré Logger Frontend :

```
interface LogEntry {
  level: 'debug' | 'info' | 'warn' | 'error';
  message: string;
  timestamp: string;
  userId?: string;
  sessionId?: string;
  metadata?: Record<string, any>;
}

class Logger {
  private sessionId = crypto.randomUUID();

  private createEntry(
    level: LogEntry['level'],
    message: string,
    metadata?: Record<string, any>
  ): LogEntry {
    return {
      level,
      message,
      timestamp: new Date().toISOString(),
      sessionId: this.sessionId,
      userId: this.getCurrentUserId(),
      metadata
    };
  }
```

```
  info(message: string, metadata?: Record<string, any>) {
    const entry = this.createEntry('info', message, metadata);
    console.log(JSON.stringify(entry));

    // Envoyer au service de logging en production
    if (process.env.NODE_ENV === 'production') {
      this.sendToLogService(entry);
    }
  }

  error(message: string, error?: Error, metadata?: Record<string, any>) {
    const entry = this.createEntry('error', message, {
      ...metadata,
      error: error ? {
        name: error.name,
        message: error.message,
        stack: error.stack
      } : undefined
    });

    console.error(JSON.stringify(entry));
    this.sendToLogService(entry);
  }

  private async sendToLogService(entry: LogEntry) {
    try {
      await fetch('/api/logs', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(entry)
      });
    } catch (error) {
      // Fallback : stocker en localStorage pour retry
      this.storeForRetry(entry);
    }
  }
}

export const logger = new Logger();
```

Métriques Business :

```
// Tracking des événements business
class Analytics {
  track(event: string, properties?: Record<string, any>) {
    const payload = {
```

```
      event,
      properties,
      timestamp: new Date().toISOString(),
      userId: this.getCurrentUserId(),
      sessionId: this.getSessionId(),
      page: window.location.pathname
    };

    // Envoyer immédiatement les événements critiques
    if (this.isCriticalEvent(event)) {
      this.sendImmediately(payload);
    } else {
      // Batch les autres événements
      this.addToBatch(payload);
    }
  }

  // Événements business typiques
  productViewed(productId: string, productName: string) {
    this.track('Product Viewed', { productId, productName });
  }

  addedToCart(productId: string, quantity: number, price: number) {
    this.track('Added to Cart', { productId, quantity, price, revenue: quantity * price });
  }

  purchaseCompleted(orderId: string, total: number, items: any[]) {
    this.track('Purchase Completed', {
      orderId,
      total,
      itemCount: items.length,
      revenue: total
    });
  }
}

export const analytics = new Analytics();
```

Cette documentation complète couvre tous les aspects d'utilisation de Riziky-Boutic pour chaque type d'utilisateur. Elle fournit les informations nécessaires pour comprendre, utiliser, et maintenir efficacement la plateforme.