Cahier des Charges Final - Riziky-Boutic

Présentation du Projet

Vue d'Ensemble Exécutive

Riziky-Boutic est une plateforme e-commerce complète et moderne développée pour offrir une expérience d'achat en ligne optimale. Ce cahier des charges détaille l'ensemble des spécifications techniques, fonctionnelles et opérationnelles de la solution déployée.

Objectifs Stratégiques

- **Expérience Utilisateur** : Interface intuitive et responsive sur tous supports
- **Performance** : Chargement rapide et navigation fluide
- **Sécurité** : Protection des données et transactions sécurisées
- **Évolutivité** : Architecture modulaire pour croissance future
- **Maintenance** : Code propre et documentation complète

Périmètre Fonctionnel

- Boutique en ligne avec catalogue produits
- Système de commande et paiement sécurisé
- Interface d'administration complète
- Chat support temps réel
- Gestion des utilisateurs et des rôles

Architecture Technique Globale

Stack Technologique Choisi

Frontend (Client)

```
// Technologies principales
React 18.3.1          // Framework UI moderne avec hooks
TypeScript 5.0+        // Sécurité des types et meilleur tooling
Tailwind CSS 3.3+      // Framework CSS utilitaire
Shadcn/UI              // Composants UI pré-construits et accessibles
React Router 6.26+     // Routing côté client avec data loading
Axios 1.6.8            // Client HTTP avec interceptors
Socket.io Client       // Communication temps réel bidirectionnelle
```

Backend (Serveur)

```
// Technologies serveur
Node.js 18+            // Runtime JavaScript haute performance
```

```
Express.js 4.18+        // Framework web minimaliste et flexible
Socket.io 4.8+          // WebSocket serveur pour temps réel
Multer                  // Gestion upload de fichiers
Helmet                  // Sécurité headers HTTP
bcrypt                  // Hachage sécurisé des mots de passe
```

Base de Données et Stockage

```
Actuel: JSON Files      // Fichiers JSON pour développement rapide
Migration: PostgreSQL   // Base de données relationnelle pour production
Redis (optionnel)       // Cache et sessions pour performance
```

Architecture des Dossiers

```
riziky-boutic/
    src/                            # Code source React frontend
        components/                 # Composants réutilisables
            ui/                     # Composants UI de base (shadcn)
                button.tsx          # Bouton avec variantes
                input.tsx           # Champs de saisie
                card.tsx            # Conteneurs avec header/content/footer
                dialog.tsx          # Modales et dialogues
                ...                 # Autres composants UI
            layout/                 # Composants de mise en page
                Navbar.tsx          # Navigation principale responsive
                Footer.tsx          # Pied de page avec liens
                Layout.tsx          # Layout principal avec Outlet
                Sidebar.tsx         # Barre latérale admin
            auth/                   # Composants authentification
                LoginForm.tsx       # Formulaire de connexion
                RegisterForm.tsx    # Formulaire d'inscription
                PasswordStrength.tsx # Indicateur force mot de passe
            products/               # Composants produits
                ProductCard.tsx     # Carte produit avec actions
                ProductGrid.tsx     # Grille responsive de produits
                ProductFilters.tsx  # Filtres de recherche avancée
                QuickView.tsx       # Vue rapide en modal
            cart/                   # Composants panier
                CartDrawer.tsx      # Panier latéral (Sheet)
                CartItem.tsx        # Article du panier
                CartSummary.tsx     # Résumé avec totaux
            admin/                  # Composants administration
                ProductForm.tsx     # Création/édition produits
                OrderManagement.tsx # Gestion des commandes
                UserManagement.tsx  # Gestion des utilisateurs
            modern/                 # Interface moderne
```

```
            ModernCalendar.tsx # Calendrier stylé
            FloatingActionButton.tsx # FAB Material Design
        pages/                    # Pages de l'application
            HomePage.tsx          # Page d'accueil avec héro et produits vedettes
            ProductDetail.tsx     # Détail produit avec avis et suggestions
            CartPage.tsx          # Page panier complète
            CheckoutPage.tsx      # Processus de commande multi-étapes
            ProfilePage.tsx       # Profil utilisateur avec onglets
            admin/                # Pages administration
                AdminDashboard.tsx # Tableau de bord avec métriques
                AdminProducts.tsx  # Gestion produits avec CRUD
                AdminOrders.tsx    # Gestion commandes et statuts
        hooks/                    # Hooks personnalisés React
            useAuth.ts           # Gestion authentification globale
            useCart.ts           # Gestion état du panier
            useProducts.ts       # Récupération et cache des produits
            useFavorites.ts      # Gestion liste de favoris
        contexts/                 # Contextes React pour état global
            AuthContext.tsx      # Contexte utilisateur et auth
            StoreContext.tsx     # État global de la boutique
        services/                 # Services API et logique métier
            api.ts               # Configuration Axios et intercepteurs
            authService.ts       # Service d'authentification
            productsService.ts   # Service de gestion des produits
            ordersService.ts     # Service de gestion des commandes
        lib/                      # Utilitaires et helpers
            utils.ts             # Utilitaires généraux (cn, formatPrice, etc.)
            validations.ts       # Schémas de validation Zod
    server/                       # Code source backend Node.js
        routes/                   # Routes API Express
            auth.js              # Routes authentification (/api/auth/*)
            products.js          # Routes produits (/api/products/*)
            orders.js            # Routes commandes (/api/orders/*)
            panier.js            # Routes panier (/api/panier/*)
            users.js             # Routes utilisateurs (/api/users/*)
        services/                 # Services métier backend
            auth.service.js      # Logique d'authentification JWT
            products.service.js  # Logique de gestion des produits
            orders.service.js    # Logique de traitement des commandes
        middlewares/              # Middlewares Express
            auth.js              # Vérification tokens JWT
            security.js          # Rate limiting et sécurité
        config/                   # Configuration serveur
            cors.js              # Configuration CORS
            security.js          # Headers de sécurité (Helmet)
        data/                     # Fichiers JSON (base de données actuelle)
```

```
        products.json       # Catalogue des produits
        users.json          # Utilisateurs et admins
        orders.json         # Commandes passées
        categories.json     # Catégories de produits
    socket/                 # Configuration WebSocket
        socketHandlers.js   # Gestionnaires événements Socket.io
    server.js               # Point d'entrée serveur Express
docs/                       # Documentation complète
    README_PROJET.md        # Guide principal du projet
    GUIDE_COMPOSANTS.md     # Documentation des composants
    MANUEL_DEVELOPPEUR.md   # Manuel pour développeurs
    ARCHITECTURE_TECHNIQUE.md # Spécifications techniques
```

Spécifications Détaillées des Composants

Composants UI de Base (`src/components/ui/`)

Button - Bouton Universel **Localisation :** `src/components/ui/button.tsx`
Responsabilité : Composant bouton réutilisable avec variantes et tailles

Props Interface :

```tsx
interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {
  variant?: 'default' | 'destructive' | 'outline' | 'secondary' | 'ghost' | 'link'
  size?: 'default' | 'sm' | 'lg' | 'icon'
  asChild?: boolean  // Render comme enfant avec Slot
}
```

Utilisation Standard :

```tsx
// Bouton primaire pour actions principales
<Button variant="default" size="lg">
  Ajouter au panier
</Button>

// Bouton destructif pour suppressions
<Button variant="destructive" onClick={() => handleDelete(id)}>
  <Trash2 className="mr-2 h-4 w-4" />
  Supprimer
</Button>

// Bouton icône uniquement
<Button variant="ghost" size="icon">
  <Heart classNameh="h-4 w-4" />
</Button>
```

Logique de Modification : Pour ajouter une nouvelle variante, modifier le

buttonVariants avec class-variance-authority :

```
const buttonVariants = cva(
  "inline-flex items-center justify-center rounded-md text-sm font-medium...",
  {
    variants: {
      variant: {
        default: "bg-primary text-primary-foreground hover:bg-primary/90",
        // Nouvelle variante
        success: "bg-green-600 text-white hover:bg-green-700",
        warning: "bg-yellow-600 text-white hover:bg-yellow-700"
      }
    }
  }
)
```

Card - Conteneur Structuré **Localisation :** `src/components/ui/card.tsx`
Responsabilité : Conteneur avec sections définies pour structurer le contenu

Structure Complète :

```
<Card className="overflow-hidden">
  <CardHeader className="pb-3">
    <CardTitle>Titre Principal</CardTitle>
    <CardDescription>Description ou sous-titre</CardDescription>
  </CardHeader>
  <CardContent className="space-y-4">
    {/* Contenu principal */}
  </CardContent>
  <CardFooter className="flex justify-between">
    <Button variant="outline">Annuler</Button>
    <Button>Confirmer</Button>
  </CardFooter>
</Card>
```

Personnalisation Avancée :

```
// Card avec gradient et animation
<Card className={cn(
  "transition-all duration-300 hover:shadow-lg",
  "bg-gradient-to-br from-primary/5 to-secondary/5",
  "border-primary/20 hover:border-primary/40"
)}>
```

Dialog - Modal et Dialogues **Localisation :** `src/components/ui/dialog.tsx`
Responsabilité : Gestion des modales, confirmations et formulaires overlay

Pattern Complet :

```
const [isOpen, setIsOpen] = useState(false)

<Dialog open={isOpen} onOpenChange={setIsOpen}>
  <DialogTrigger asChild>
    <Button>Ouvrir Modal</Button>
  </DialogTrigger>

  <DialogContent className="sm:max-w-md">
    <DialogHeader>
      <DialogTitle>Confirmation</DialogTitle>
      <DialogDescription>
        Cette action ne peut pas être annulée.
      </DialogDescription>
    </DialogHeader>

    <div className="space-y-4">
      {/* Contenu de la modal */}
    </div>

    <DialogFooter className="sm:justify-start">
      <DialogClose asChild>
        <Button variant="secondary">Annuler</Button>
      </DialogClose>
      <Button onClick={handleConfirm}>Confirmer</Button>
    </DialogFooter>
  </DialogContent>
</Dialog>
```

Utilisation Avancée - Modal de Confirmation :

```
// Hook personnalisé pour confirmations
const useConfirmDialog = () => {
  const [isOpen, setIsOpen] = useState(false)
  const [config, setConfig] = useState<{
    title: string
    message: string
    onConfirm: () => void
  } | null>(null)

  const confirm = (title: string, message: string, onConfirm: () => void) => {
    setConfig({ title, message, onConfirm })
    setIsOpen(true)
  }

  const ConfirmDialog = () => (
    <Dialog open={isOpen} onOpenChange={setIsOpen}>
      <DialogContent>
```

```
        <DialogHeader>
          <DialogTitle>{config?.title}</DialogTitle>
          <DialogDescription>{config?.message}</DialogDescription>
        </DialogHeader>
        <DialogFooter>
          <Button variant="outline" onClick={() => setIsOpen(false)}>
            Annuler
          </Button>
          <Button
            variant="destructive"
            onClick={() => {
              config?.onConfirm()
              setIsOpen(false)
            }}
          >
            Confirmer
          </Button>
        </DialogFooter>
      </DialogContent>
    </Dialog>
  )

  return { confirm, ConfirmDialog }
}
```

Composants E-commerce Spécialisés

ProductCard - Carte Produit Interactive **Localisation :** `src/components/products/ProductCard.ts`
Responsabilité : Affichage produit avec interactions (panier, favoris, vue rapide)

Interface Complète :

```
interface ProductCardProps {
  product: Product                          // Données du produit
  variant?: 'default' | 'compact' | 'detailed' | 'featured'
  showActions?: boolean                     // Afficher boutons d'action
  showQuickView?: boolean                   // Bouton vue rapide
  showPromotion?: boolean                   // Badge de promotion
  onQuickView?: (product: Product) => void  // Callback vue rapide
  onAddToCart?: (productId: string, quantity: number) => void
  onToggleFavorite?: (productId: string) => void
}
```

Logique Métier Intégrée :

```
const ProductCard: FC<ProductCardProps> = ({ product, variant = 'default', ...props }) => {
  const { addToCart, isLoading: cartLoading } = useCart()
```

```
const { toggleFavorite, isFavorite } = useFavorites()

// Calcul du prix avec promotion
const hasPromotion = product.promotion && product.promotion.pourcentage > 0
const originalPrice = product.prix
const finalPrice = hasPromotion
  ? originalPrice - (originalPrice * product.promotion.pourcentage / 100)
  : originalPrice

// Gestion de l'ajout au panier avec validation
const handleAddToCart = async (e: React.MouseEvent) => {
  e.preventDefault()
  e.stopPropagation()

  if (product.stock <= 0) {
    toast.error("Produit en rupture de stock")
    return
  }

  try {
    await addToCart(product.id, 1)
    toast.success(`${product.nom} ajouté au panier`, {
      action: {
        label: "Voir le panier",
        onClick: () => {/* Ouvrir panier */}
      }
    })
  } catch (error) {
    toast.error("Erreur lors de l'ajout au panier")
  }
}

// Gestion des favoris
const handleToggleFavorite = async (e: React.MouseEvent) => {
  e.preventDefault()
  e.stopPropagation()

  try {
    await toggleFavorite(product.id)
    const action = isFavorite(product.id) ? "retiré des" : "ajouté aux"
    toast.success(`Produit ${action} favoris`)
  } catch (error) {
    toast.error("Erreur lors de la modification des favoris")
  }
}
```

```
return (
  <Card className={cn(
    "group relative overflow-hidden transition-all duration-300",
    "hover:shadow-lg hover:shadow-primary/10 hover:-translate-y-1",
    {
      'default': 'max-w-sm',
      'compact': 'max-w-xs',
      'detailed': 'max-w-md',
      'featured': 'max-w-lg border-primary/50 bg-gradient-to-br from-primary/5'
    }[variant]
  )}>
    {/* Image avec overlay d'actions au hover */}
    <div className="relative aspect-square overflow-hidden bg-gray-100">
      <img
        src={product.images?.[0] || '/placeholder-product.jpg'}
        alt={product.nom}
        className="h-full w-full object-cover transition-transform duration-300 group-hove
        loading="lazy"
      />

      {/* Badges de statut */}
      <div className="absolute top-2 left-2 space-y-1">
        {hasPromotion && (
          <Badge variant="destructive" className="font-semibold">
            -{product.promotion.pourcentage}%
          </Badge>
        )}
        {product.stock <= 0 && (
          <Badge variant="secondary">
            Rupture
          </Badge>
        )}
        {product.stock <= 5 && product.stock > 0 && (
          <Badge variant="outline" className="bg-orange-100 text-orange-800">
            Stock faible
          </Badge>
        )}
      </div>

      {/* Overlay d'actions au hover */}
      {props.showActions !== false && (
        <div className="absolute inset-0 bg-black/50 opacity-0 transition-opacity duratio
          {/* Bouton favoris */}
          <Button
            size="icon"
            variant="secondary"
```

```
            onClick={handleToggleFavorite}
            className="bg-white/90 hover:bg-white"
          >
            <Heart className={cn(
              "h-4 w-4 transition-colors",
              isFavorite(product.id) && "fill-red-500 text-red-500"
            )} />
          </Button>

          {/* Bouton vue rapide */}
          {props.showQuickView !== false && (
            <Button
              size="icon"
              variant="secondary"
              onClick={() => props.onQuickView?.(product)}
              className="bg-white/90 hover:bg-white"
            >
              <Eye className="h-4 w-4" />
            </Button>
          )}

          {/* Bouton comparaison */}
          <Button
            size="icon"
            variant="secondary"
            onClick={() => {/* Logique comparaison */}}
            className="bg-white/90 hover:bg-white"
          >
            <Scale className="h-4 w-4" />
          </Button>
        </div>
      )}
    </div>

    {/* Contenu de la carte */}
    <CardContent className="p-4">
      <div className="space-y-2">
        {/* Nom du produit */}
        <h3 className={cn(
          "font-semibold line-clamp-2 hover:text-primary transition-colors",
          {
            'default': 'text-sm',
            'compact': 'text-xs',
            'detailed': 'text-base',
            'featured': 'text-lg'
          }[variant]
```

```
        )}>
          {product.nom}
        </h3>

        {/* Description pour variante détaillée */}
        {variant === 'detailed' && product.description && (
          <p className="text-sm text-muted-foreground line-clamp-2">
            {product.description}
          </p>
        )}

        {/* Prix avec gestion des promotions */}
        <div className="flex items-center space-x-2">
          <span className="text-lg font-bold text-primary">
            {finalPrice.toLocaleString('fr-FR', {
              style: 'currency',
              currency: 'EUR'
            })}
          </span>
          {hasPromotion && (
            <span className="text-sm text-muted-foreground line-through">
              {originalPrice.toLocaleString('fr-FR', {
                style: 'currency',
                currency: 'EUR'
              })}
            </span>
          )}
        </div>

        {/* Informations stock pour variante détaillée */}
        {variant === 'detailed' && (
          <div className="flex items-center justify-between text-sm text-muted-foreground'
            <div className="flex items-center space-x-1">
              <Package className="h-3 w-3" />
              <span>Stock: {product.stock}</span>
            </div>
            {product.categories && product.categories.length > 0 && (
              <Badge variant="outline" className="text-xs">
                {product.categories[0]}
              </Badge>
            )}
          </div>
        )}
      </div>
    </CardContent>
```

```
      {/* Actions en pied de carte */}
      {props.showActions !== false && (
        <CardFooter className="p-4 pt-0">
          <Button
            className="w-full"
            onClick={handleAddToCart}
            disabled={product.stock <= 0 || cartLoading}
          >
            {cartLoading ? (
              <Loader2 className="mr-2 h-4 w-4 animate-spin" />
            ) : (
              <ShoppingCart className="mr-2 h-4 w-4" />
            )}
            {product.stock <= 0 ? 'Indisponible' : 'Ajouter au panier'}
          </Button>
        </CardFooter>
      )}
    </Card>
  )
}
```

Comment Modifier ou Étendre :

1. **Ajouter une nouvelle variante :**

```
// Dans les props, ajouter la variante
variant?: 'default' | 'compact' | 'detailed' | 'featured' | 'mini'

// Dans le className conditionnel
{
  'default': 'max-w-sm',
  'compact': 'max-w-xs',
  'detailed': 'max-w-md',
  'featured': 'max-w-lg border-primary/50',
  'mini': 'max-w-32 text-xs' // Nouvelle variante
}[variant]
```

2. **Personnaliser les actions :**

```
// Ajouter des props pour contrôler les actions
interface ProductCardProps {
  // ... autres props
  actions?: {
    cart?: boolean
    favorites?: boolean
    quickView?: boolean
    compare?: boolean
    share?: boolean  // Nouvelle action
```

```
  }
}

// Dans le composant
const actions = {
  cart: true,
  favorites: true,
  quickView: true,
  compare: false,
  ...props.actions
}
```

3. **Ajouter des métriques :**

```
// Tracking des interactions
const trackProductInteraction = (action: string, productId: string) => {
  // Analytics
  gtag('event', action, {
    event_category: 'Product',
    event_label: productId
  })
}

// Dans les handlers
const handleAddToCart = async (e: React.MouseEvent) => {
  // ... logique existante
  trackProductInteraction('add_to_cart', product.id)
}
```

CartDrawer - Panier Latéral Intelligent **Localisation :** `src/components/cart/CartDrawer.tsx`
Responsabilité : Interface de panier avec gestion complète du workflow d'achat

Logique Métier Avancée :

```
const CartDrawer: FC = () => {
  const {
    cart,
    updateQuantity,
    removeFromCart,
    clearCart,
    totalPrice,
    itemCount,
    isLoading
  } = useCart()

  const navigate = useNavigate()
```

```tsx
const [isOpen, setIsOpen] = useState(false)

// Calculs sophistiqués
const subtotal = cart.reduce((acc, item) => acc + (item.prix * item.quantite), 0)
const freeShippingThreshold = 50
const shipping = subtotal >= freeShippingThreshold ? 0 : 5.99
const tax = subtotal * 0.20 // TVA 20%
const total = subtotal + shipping + tax

// Montant restant pour livraison gratuite
const remainingForFreeShipping = Math.max(0, freeShippingThreshold - subtotal)

// Suggestions cross-sell
const [suggestedProducts, setSuggestedProducts] = useState<Product[]>([])

useEffect(() => {
  if (cart.length > 0) {
    // Charger des produits complémentaires basés sur le panier
    const categories = [...new Set(cart.flatMap(item => item.categories || []))]
    loadProductSuggestions(categories).then(setSuggestedProducts)
  }
}, [cart])

return (
  <Sheet open={isOpen} onOpenChange={setIsOpen}>
    <SheetTrigger asChild>
      <Button variant="outline" size="icon" className="relative">
        <ShoppingCart className="h-4 w-4" />
        {itemCount > 0 && (
          <Badge
            variant="destructive"
            className="absolute -top-2 -right-2 h-5 w-5 rounded-full p-0 flex items-center
          >
            {itemCount > 99 ? '99+' : itemCount}
          </Badge>
        )}
      </Button>
    </SheetTrigger>

    <SheetContent className="flex flex-col w-full sm:max-w-lg">
      <SheetHeader className="space-y-2.5">
        <SheetTitle className="flex items-center justify-between">
          <span>Panier ({itemCount} {itemCount <= 1 ? 'article' : 'articles'})</span>
          {cart.length > 0 && (
            <Button
              variant="ghost"
```

```
            size="sm"
            onClick={handleClearCart}
            className="text-destructive hover:text-destructive h-auto p-1"
          >
            <Trash2 className="h-4 w-4" />
          </Button>
        )}
      </SheetTitle>

      {/* Indicateur de livraison gratuite */}
      {remainingForFreeShipping > 0 ? (
        <Alert className="border-blue-200 bg-blue-50">
          <Truck className="h-4 w-4" />
          <AlertDescription className="text-sm">
            Ajoutez <strong>{remainingForFreeShipping.toFixed(2)}€</strong> d'achats
            pour bénéficier de la <strong>livraison gratuite</strong> !
          </AlertDescription>
        </Alert>
      ) : subtotal > 0 && (
        <Alert className="border-green-200 bg-green-50">
          <CheckCircle className="h-4 w-4" />
          <AlertDescription className="text-sm text-green-800">
             <strong>Livraison gratuite</strong> incluse !
          </AlertDescription>
        </Alert>
      )}
    </SheetHeader>

    {/* Contenu scrollable du panier */}
    <ScrollArea className="flex-1 -mx-6 px-6">
      {cart.length === 0 ? (
        <EmptyCartMessage />
      ) : (
        <div className="space-y-4">
          {/* Articles du panier */}
          {cart.map(item => (
            <CartItemCard
              key={item.id}
              item={item}
              variant="drawer"
              onQuantityChange={handleQuantityChange}
              onRemove={handleRemoveItem}
              isLoading={isLoading}
            />
          ))}
```

```
        {/* Suggestions de produits */}
        {suggestedProducts.length > 0 && (
          <div className="border-t pt-4 space-y-3">
            <h4 className="font-medium text-sm">Produits recommandés</h4>
            <div className="grid grid-cols-2 gap-2">
              {suggestedProducts.slice(0, 2).map(product => (
                <ProductCard
                  key={product.id}
                  product={product}
                  variant="compact"
                  showActions={false}
                />
              ))}
            </div>
          </div>
        )}
      </div>
    )}
  </ScrollArea>

  {/* Résumé et actions */}
  {cart.length > 0 && (
    <div className="space-y-4 border-t pt-4 -mx-6 px-6">
      {/* Résumé détaillé */}
      <div className="space-y-2 text-sm">
        <div className="flex justify-between">
          <span>Sous-total ({itemCount} articles)</span>
          <span>{subtotal.toFixed(2)}€</span>
        </div>
        <div className="flex justify-between">
          <span>Livraison</span>
          <span className={shipping === 0 ? "text-green-600 font-medium" : ""}>
            {shipping === 0 ? 'Gratuite' : `${shipping.toFixed(2)}€`}
          </span>
        </div>
        <div className="flex justify-between">
          <span>TVA (20%)</span>
          <span>{tax.toFixed(2)}€</span>
        </div>
        <Separator />
        <div className="flex justify-between font-semibold text-base">
          <span>Total</span>
          <span>{total.toFixed(2)}€</span>
        </div>
      </div>
```

```
                {/* Actions principales */}
                <div className="space-y-2">
                  <Button
                    className="w-full"
                    size="lg"
                    onClick={() => {
                      setIsOpen(false)
                      navigate('/checkout')
                    }}
                  >
                    <CreditCard className="mr-2 h-4 w-4" />
                    Finaliser la commande
                  </Button>

                  <Button
                    variant="outline"
                    className="w-full"
                    onClick={() => {
                      setIsOpen(false)
                      navigate('/cart')
                    }}
                  >
                    Voir le panier détaillé
                  </Button>
                </div>

                {/* Badges de confiance */}
                <div className="flex items-center justify-center space-x-4 text-xs text-muted-fo
                  <div className="flex items-center space-x-1">
                    <Shield className="h-3 w-3" />
                    <span>Paiement sécurisé</span>
                  </div>
                  <div className="flex items-center space-x-1">
                    <Truck className="h-3 w-3" />
                    <span>Livraison rapide</span>
                  </div>
                </div>
              </div>
            )}
          </SheetContent>
        </Sheet>
      )
    }
```

Composants d'Authentification

LoginForm - Formulaire de Connexion Sécurisé **Localisation :** `src/components/auth/LoginForm.tsx` **Responsabilité :** Authentification utilisateur avec sécurité renforcée

Spécifications Techniques :

```tsx
interface LoginFormData {
  email: string
  password: string
  rememberMe: boolean
  captcha?: string  // Pour protection bot après échecs
}

// Schéma de validation Zod avec règles métier
const loginSchema = z.object({
  email: z
    .string()
    .min(1, "L'email est requis")
    .email("Adresse email invalide")
    .max(255, "Email trop long"),

  password: z
    .string()
    .min(1, "Le mot de passe est requis")
    .min(8, "Le mot de passe doit contenir au moins 8 caractères")
    .max(128, "Mot de passe trop long"),

  rememberMe: z.boolean().optional()
})

const LoginForm: FC<LoginFormProps> = ({ onSuccess, redirectTo }) => {
  const { login, isLoading } = useAuth()
  const navigate = useNavigate()
  const location = useLocation()

  // États de sécurité
  const [attemptCount, setAttemptCount] = useState(0)
  const [isBlocked, setIsBlocked] = useState(false)
  const [blockTimeRemaining, setBlockTimeRemaining] = useState(0)

  const form = useForm<LoginFormData>({
    resolver: zodResolver(loginSchema),
    defaultValues: {
      email: "",
      password: "",
```

```
    rememberMe: false
  }
})

// Gestion des tentatives de connexion
useEffect(() => {
  const storedAttempts = localStorage.getItem('loginAttempts')
  const storedBlockTime = localStorage.getItem('loginBlockTime')

  if (storedAttempts) {
    setAttemptCount(parseInt(storedAttempts))
  }

  if (storedBlockTime) {
    const blockTime = parseInt(storedBlockTime)
    const now = Date.now()

    if (now < blockTime) {
      setIsBlocked(true)
      setBlockTimeRemaining(Math.ceil((blockTime - now) / 1000))

      // Décompte en temps réel
      const interval = setInterval(() => {
        const remaining = Math.ceil((blockTime - Date.now()) / 1000)
        if (remaining <= 0) {
          setIsBlocked(false)
          setBlockTimeRemaining(0)
          localStorage.removeItem('loginBlockTime')
          clearInterval(interval)
        } else {
          setBlockTimeRemaining(remaining)
        }
      }, 1000)

      return () => clearInterval(interval)
    }
  }
}, [])

const onSubmit = async (data: LoginFormData) => {
  if (isBlocked) {
    toast.error(`Trop de tentatives. Réessayez dans ${blockTimeRemaining} secondes.`)
    return
  }

  try {
```

```
      // Tentative de connexion
      await login(data.email, data.password, data.rememberMe)

      // Réinitialisation des tentatives en cas de succès
      localStorage.removeItem('loginAttempts')
      localStorage.removeItem('loginBlockTime')
      setAttemptCount(0)

      toast.success("Connexion réussie !", {
        description: `Bienvenue ${data.email.split('@')[0]} !`
      })

      // Redirection
      const from = location.state?.from?.pathname || redirectTo || "/"
      navigate(from, { replace: true })

      onSuccess?.()

    } catch (error: any) {
      const newAttemptCount = attemptCount + 1
      setAttemptCount(newAttemptCount)
      localStorage.setItem('loginAttempts', newAttemptCount.toString())

      // Gestion des erreurs spécifiques
      if (error.response?.status === 401) {
        form.setError("root", {
          message: "Email ou mot de passe incorrect"
        })

        // Blocage après 5 tentatives
        if (newAttemptCount >= 5) {
          const blockTime = Date.now() + (15 * 60 * 1000) // 15 minutes
          localStorage.setItem('loginBlockTime', blockTime.toString())
          setIsBlocked(true)
          setBlockTimeRemaining(15 * 60)

          form.setError("root", {
            message: "Trop de tentatives incorrectes. Compte bloqué pendant 15 minutes."
          })
        } else {
          const remaining = 5 - newAttemptCount
          form.setError("root", {
            message: `Email ou mot de passe incorrect. ${remaining} tentative(s) restante(s)
          })
        }
```

```
    } else if (error.response?.status === 429) {
      form.setError("root", {
        message: "Trop de requêtes. Veuillez patienter avant de réessayer."
      })
    } else if (error.response?.status === 423) {
      form.setError("root", {
        message: "Compte temporairement verrouillé. Contactez le support."
      })
    } else {
      form.setError("root", {
        message: "Erreur de connexion. Veuillez réessayer."
      })
    }

    // Analytics des échecs (sans données sensibles)
    trackEvent('login_failed', {
      error_type: error.response?.status || 'network_error',
      attempt_count: newAttemptCount
    })
  }
}

return (
  <Card className="w-full max-w-md mx-auto shadow-lg">
    <CardHeader className="space-y-2 text-center">
      <div className="mx-auto w-12 h-12 rounded-full bg-primary/10 flex items-center justi
        <Lock className="h-6 w-6 text-primary" />
      </div>
      <CardTitle className="text-2xl">Connexion</CardTitle>
      <CardDescription>
        Connectez-vous à votre compte Riziky-Boutic
      </CardDescription>
    </CardHeader>

    <CardContent>
      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
          {/* Champ Email avec validation en temps réel */}
          <FormField
            control={form.control}
            name="email"
            render={({ field }) => (
              <FormItem>
                <FormLabel>Adresse email</FormLabel>
                <FormControl>
                  <div className="relative">
```

```
            <Mail className="absolute left-3 top-3 h-4 w-4 text-muted-foreground" />
            <Input
              type="email"
              placeholder="votre@email.com"
              className="pl-10"
              {...field}
              disabled={isLoading || isBlocked}
              autoComplete="email"
            />
          </div>
        </FormControl>
        <FormMessage />
      </FormItem>
    )}
  />

  {/* Champ Mot de passe avec visibilité toggle */}
  <FormField
    control={form.control}
    name="password"
    render={({ field }) => (
      <FormItem>
        <div className="flex items-center justify-between">
          <FormLabel>Mot de passe</FormLabel>
          <Button
            type="button"
            variant="link"
            size="sm"
            className="px-0 font-normal text-muted-foreground"
            asChild
          >
            <Link to="/forgot-password">Mot de passe oublié ?</Link>
          </Button>
        </div>
        <FormControl>
          <PasswordInput
            placeholder="Votre mot de passe"
            {...field}
            disabled={isLoading || isBlocked}
            autoComplete="current-password"
          />
        </FormControl>
        <FormMessage />
      </FormItem>
    )}
  />
```

```tsx
{/* Se souvenir de moi */}
<FormField
  control={form.control}
  name="rememberMe"
  render={({ field }) => (
    <FormItem className="flex flex-row items-start space-x-3 space-y-0 rounded-m
      <FormControl>
        <Checkbox
          checked={field.value}
          onCheckedChange={field.onChange}
          disabled={isLoading || isBlocked}
        />
      </FormControl>
      <div className="space-y-1 leading-none">
        <FormLabel>
          Se souvenir de moi
        </FormLabel>
        <FormDescription>
          Restez connecté sur cet appareil pendant 30 jours
        </FormDescription>
      </div>
    </FormItem>
  )}
/>

{/* Indicateur de tentatives */}
{attemptCount > 0 && attemptCount < 5 && (
  <Alert variant="destructive">
    <AlertTriangle className="h-4 w-4" />
    <AlertDescription>
      {attemptCount} tentative(s) échouée(s). {5 - attemptCount} restante(s).
    </AlertDescription>
  </Alert>
)}

{/* Message de blocage */}
{isBlocked && (
  <Alert variant="destructive">
    <Clock className="h-4 w-4" />
    <AlertDescription>
      Compte bloqué. Réessayez dans {Math.floor(blockTimeRemaining / 60)}:{(bloc
    </AlertDescription>
  </Alert>
)}
```

```
    {/* Erreur générale */}
    {form.formState.errors.root && (
      <Alert variant="destructive">
        <AlertCircle className="h-4 w-4" />
        <AlertDescription>
          {form.formState.errors.root.message}
        </AlertDescription>
      </Alert>
    )}

    {/* Bouton de connexion */}
    <Button
      type="submit"
      className="w-full"
      size="lg"
      disabled={isLoading || isBlocked}
    >
      {isLoading ? (
        <>
          <Loader2 className="mr-2 h-4 w-4 animate-spin" />
          Connexion en cours...
        </>
      ) : isBlocked ? (
        <>
          <Clock className="mr-2 h-4 w-4" />
          Compte bloqué
        </>
      ) : (
        <>
          <LogIn className="mr-2 h-4 w-4" />
          Se connecter
        </>
      )}
    </Button>
  </form>
</Form>

{/* Liens et informations supplémentaires */}
<div className="mt-6 space-y-4">
  <Separator />

  <div className="text-center text-sm">
    Pas encore de compte ?{" "}
    <Button variant="link" asChild className="p-0 h-auto">
      <Link to="/register" className="font-semibold">
        Créer un compte
```

```
              </Link>
            </Button>
          </div>

          {/* Méthodes de connexion alternatives */}
          <div className="space-y-2">
            <Button variant="outline" className="w-full" type="button">
              <svg className="mr-2 h-4 w-4" viewBox="0 0 24 24">
                {/* Google Icon SVG */}
              </svg>
              Continuer avec Google
            </Button>

            <Button variant="outline" className="w-full" type="button">
              <Facebook className="mr-2 h-4 w-4" />
              Continuer avec Facebook
            </Button>
          </div>
        </div>
      </CardContent>
    </Card>
  )
}
```

Pages et Interfaces Utilisateur

HomePage - Page d'Accueil Dynamique

Localisation : `src/pages/HomePage.tsx` **Responsabilité :** Page d'accueil avec sections dynamiques et personnalisées

Architecture de la Page :

```
const HomePage: FC = () => {
  // Hooks pour les données
  const {
    featuredProducts,
    categories,
    flashSales,
    testimonials,
    isLoading,
    error
  } = useHomePageData()

  const { user } = useAuth()
```

```
// Personnalisation basée sur l'utilisateur
const [personalizedRecommendations, setPersonalizedRecommendations] = useState<Product[]>(

useEffect(() => {
  if (user) {
    // Charger les recommandations personnalisées
    loadPersonalizedProducts(user.id).then(setPersonalizedRecommendations)
  }
}, [user])

if (isLoading) return <HomePageSkeleton />
if (error) return <ErrorMessage error={error} retry={() => window.location.reload()} />

return (
  <div className="min-h-screen">
    {/* Hero Section avec CTA dynamique */}
    <HeroSection
      slides={[
        {
          image: '/images/hero-1.jpg',
          title: 'Collection Été 2024',
          subtitle: 'Découvrez nos nouveautés exclusives',
          cta: { text: 'Découvrir', href: '/new-arrivals' },
          overlay: 'dark'
        },
        {
          image: '/images/hero-2.jpg',
          title: 'Ventes Flash',
          subtitle: 'Jusqu\'à -70% sur une sélection d\'articles',
          cta: { text: 'Profiter des offres', href: '/flash-sales' },
          overlay: 'light'
        }
      ]}
      autoplayInterval={5000}
      showIndicators={true}
    />

    {/* Bande promotionnelle */}
    <PromoBanner
      message=" Livraison gratuite dès 50€ d'achat"
      backgroundColor="bg-primary"
      textColor="text-primary-foreground"
      dismissible={true}
    />

    {/* Ventes Flash si actives */}
```

```
{flashSales.length > 0 && (
  <section className="py-12 bg-gradient-to-r from-red-50 to-orange-50">
    <div className="container">
      <FlashSaleBanner
        flashSales={flashSales}
        showCountdown={true}
        maxItems={4}
      />
    </div>
  </section>
)}

{/* Catégories populaires */}
<section className="py-16">
  <div className="container">
    <div className="text-center mb-12">
      <h2 className="text-3xl font-bold tracking-tight sm:text-4xl">
        Nos Catégories
      </h2>
      <p className="mt-4 text-lg text-muted-foreground">
        Explorez notre large gamme de produits
      </p>
    </div>

    <CategoriesGrid
      categories={categories.slice(0, 8)}
      variant="image-overlay"
      showProductCount={true}
      className="grid-cols-2 md:grid-cols-4"
    />
  </div>
</section>

{/* Produits vedettes */}
<section className="py-16 bg-muted/30">
  <div className="container">
    <div className="flex items-center justify-between mb-12">
      <div>
        <h2 className="text-3xl font-bold tracking-tight">
          Produits Vedettes
        </h2>
        <p className="mt-2 text-muted-foreground">
          Nos coups de cœur sélectionnés pour vous
        </p>
      </div>
      <Button variant="outline" asChild>
```

```
        <Link to="/products/featured">
          Voir tout
          <ArrowRight className="ml-2 h-4 w-4" />
        </Link>
      </Button>
    </div>

    <FeaturedProductsCarousel
      products={featuredProducts}
      autoplay={true}
      showDots={true}
      responsive={{
        mobile: 1,
        tablet: 2,
        desktop: 4
      }}
    />
  </div>
</section>

{/* Recommandations personnalisées (si utilisateur connecté) */}
{user && personalizedRecommendations.length > 0 && (
  <section className="py-16">
    <div className="container">
      <div className="text-center mb-12">
        <h2 className="text-3xl font-bold tracking-tight">
          Recommandé pour vous
        </h2>
        <p className="mt-2 text-muted-foreground">
          Basé sur vos achats et préférences
        </p>
      </div>

      <ProductGrid
        products={personalizedRecommendations}
        variant="grid"
        columns={{ mobile: 2, tablet: 3, desktop: 4 }}
        showPagination={false}
      />
    </div>
  </section>
)}

{/* Section testimonials */}
<section className="py-16 bg-primary/5">
  <div className="container">
```

```
    <TestimonialSection
      testimonials={testimonials}
      showRatings={true}
      autoRotate={true}
      interval={6000}
    />
  </div>
</section>

{/* Newsletter et avantages */}
<section className="py-16">
  <div className="container">
    <div className="grid lg:grid-cols-2 gap-12 items-center">
      {/* Newsletter */}
      <div className="space-y-6">
        <div>
          <h3 className="text-2xl font-bold">
            Restez informé de nos nouveautés
          </h3>
          <p className="mt-2 text-muted-foreground">
            Recevez nos offres exclusives et nos dernières collections
          </p>
        </div>

        <NewsletterForm
          placeholder="Votre adresse email"
          buttonText="S'abonner"
          includePreferences={true}
          showPrivacyNote={true}
        />
      </div>

      {/* Avantages */}
      <div className="space-y-8">
        <BenefitsSection
          benefits={[
            {
              icon: Truck,
              title: 'Livraison rapide',
              description: 'Expédition sous 24h et livraison gratuite dès 50€'
            },
            {
              icon: Shield,
              title: 'Paiement sécurisé',
              description: 'Transactions protégées SSL et conformité RGPD'
            },
```

```
                {
                  icon: RefreshCw,
                  title: 'Retours gratuits',
                  description: '30 jours pour changer d\'avis, retours sans frais'
                },
                {
                  icon: Headphones,
                  title: 'Support 7j/7',
                  description: 'Service client disponible par chat, email ou téléphone'
                }
              ]}
              variant="vertical"
            />
          </div>
        </div>
      </div>
    </section>

    {/* Produits récemment vus (si applicable) */}
    <RecentlyViewedProducts />

    {/* Widget de chat support */}
    <LiveChatWidget />
  </div>
 )
}
```

CheckoutPage - Processus de Commande Multi-Étapes

Localisation : `src/pages/CheckoutPage.tsx` **Responsabilité :** Tunnel d'achat sécurisé avec validation à chaque étape

Architecture du Processus :

```
interface CheckoutStep {
  id: string
  title: string
  description: string
  component: React.ComponentType<any>
  validation: ZodSchema
  canSkip: boolean
}

const checkoutSteps: CheckoutStep[] = [
  {
    id: 'cart-review',
    title: 'Révision du panier',
```

```
    description: 'Vérifiez vos articles avant de continuer',
    component: CartReviewStep,
    validation: cartReviewSchema,
    canSkip: false
  },
  {
    id: 'shipping',
    title: 'Livraison',
    description: 'Adresse et options de livraison',
    component: ShippingStep,
    validation: shippingSchema,
    canSkip: false
  },
  {
    id: 'payment',
    title: 'Paiement',
    description: 'Informations de paiement sécurisées',
    component: PaymentStep,
    validation: paymentSchema,
    canSkip: false
  },
  {
    id: 'confirmation',
    title: 'Confirmation',
    description: 'Vérification finale de votre commande',
    component: ConfirmationStep,
    validation: confirmationSchema,
    canSkip: false
  }
]

const CheckoutPage: FC = () => {
  const { cart, totalPrice, clearCart } = useCart()
  const { user, isAuthenticated } = useAuth()
  const navigate = useNavigate()

  // État du processus de commande
  const [currentStep, setCurrentStep] = useState(0)
  const [completedSteps, setCompletedSteps] = useState<Set<number>>(new Set())
  const [checkoutData, setCheckoutData] = useState<CheckoutData>({
    cart: cart,
    shipping: null,
    payment: null,
    totals: {
      subtotal: 0,
      shipping: 0,
```

```
    tax: 0,
    total: 0
  }
})
const [isProcessing, setIsProcessing] = useState(false)

// Redirection si panier vide
useEffect(() => {
  if (cart.length === 0) {
    toast.error("Votre panier est vide")
    navigate('/cart')
  }
}, [cart, navigate])

// Redirection si non authentifié (optionnel selon la logique métier)
useEffect(() => {
  if (!isAuthenticated) {
    toast.info("Connectez-vous pour finaliser votre commande")
    navigate('/login', { state: { from: { pathname: '/checkout' } } })
  }
}, [isAuthenticated, navigate])

// Calcul des totaux à chaque changement
useEffect(() => {
  const subtotal = cart.reduce((sum, item) => sum + (item.prix * item.quantite), 0)
  const shippingCost = calculateShipping(subtotal, checkoutData.shipping?.method)
  const tax = calculateTax(subtotal, checkoutData.shipping?.address?.country)
  const total = subtotal + shippingCost + tax

  setCheckoutData(prev => ({
    ...prev,
    totals: {
      subtotal,
      shipping: shippingCost,
      tax,
      total
    }
  }))
}, [cart, checkoutData.shipping])

// Navigation entre étapes avec validation
const goToStep = async (stepIndex: number) => {
  // Validation de l'étape actuelle avant passage à la suivante
  if (stepIndex > currentStep) {
    const currentStepData = checkoutSteps[currentStep]
    try {
```

```
      await currentStepData.validation.parseAsync(getStepData(currentStep))
      setCompletedSteps(prev => new Set([...prev, currentStep]))
    } catch (error) {
      toast.error("Veuillez compléter les informations requises")
      return
    }
  }

  setCurrentStep(stepIndex)
}

const nextStep = () => {
  if (currentStep < checkoutSteps.length - 1) {
    goToStep(currentStep + 1)
  }
}

const prevStep = () => {
  if (currentStep > 0) {
    setCurrentStep(currentStep - 1)
  }
}

// Récupération des données de l'étape actuelle
const getStepData = (stepIndex: number) => {
  switch (stepIndex) {
    case 0: return { cart: checkoutData.cart }
    case 1: return checkoutData.shipping
    case 2: return checkoutData.payment
    case 3: return checkoutData
    default: return null
  }
}

// Mise à jour des données d'étape
const updateStepData = (stepIndex: number, data: any) => {
  setCheckoutData(prev => {
    switch (stepIndex) {
      case 1:
        return { ...prev, shipping: data }
      case 2:
        return { ...prev, payment: data }
      default:
        return prev
    }
  })
```

```
}

// Finalisation de la commande
const finalizeOrder = async () => {
  setIsProcessing(true)

  try {
    // Validation finale complète
    await Promise.all(
      checkoutSteps.slice(0, -1).map((step, index) =>
        step.validation.parseAsync(getStepData(index))
      )
    )

    // Création de la commande
    const orderData = {
      items: checkoutData.cart.map(item => ({
        productId: item.id,
        quantity: item.quantite,
        price: item.prix,
        name: item.nom
      })),
      shipping: checkoutData.shipping,
      payment: {
        method: checkoutData.payment?.method,
        // Les données sensibles sont traitées côté serveur
      },
      totals: checkoutData.totals,
      userId: user?.id,
      metadata: {
        userAgent: navigator.userAgent,
        timestamp: new Date().toISOString()
      }
    }

    // Appel API pour créer la commande
    const response = await ordersService.createOrder(orderData)

    if (response.success) {
      // Vider le panier
      await clearCart()

      // Tracking de conversion
      trackConversion({
        orderId: response.order.id,
        value: checkoutData.totals.total,
```

```
          currency: 'EUR',
          items: checkoutData.cart
        })

        // Redirection vers page de succès
        navigate(`/order-success/${response.order.id}`, {
          replace: true,
          state: {
            order: response.order,
            fromCheckout: true
          }
        })

        toast.success("Commande créée avec succès !", {
          description: `Numéro de commande: ${response.order.number}`
        })

      } else {
        throw new Error(response.message || 'Erreur lors de la création de la commande')
      }

    } catch (error: any) {
      console.error('Erreur finalisation commande:', error)

      if (error.response?.status === 409) {
        toast.error("Stock insuffisant pour certains articles")
      } else if (error.response?.status === 402) {
        toast.error("Erreur de paiement. Vérifiez vos informations.")
      } else {
        toast.error("Erreur lors de la finalisation. Veuillez réessayer.")
      }

      // En cas d'erreur critique, revenir à l'étape de paiement
      setCurrentStep(2)

    } finally {
      setIsProcessing(false)
    }
  }

  const currentStepComponent = checkoutSteps[currentStep]
  const StepComponent = currentStepComponent.component

  return (
    <div className="min-h-screen bg-muted/30">
      <div className="container py-8">
```

```
{/* Header avec navigation par étapes */}
<CheckoutHeader
  steps={checkoutSteps}
  currentStep={currentStep}
  completedSteps={completedSteps}
  onStepClick={goToStep}
/>

<div className="grid lg:grid-cols-3 gap-8 mt-8">
  {/* Contenu principal de l'étape */}
  <div className="lg:col-span-2">
    <Card className="p-6">
      <div className="space-y-6">
        {/* Titre et description de l'étape */}
        <div>
          <h2 className="text-2xl font-semibold">
            {currentStepComponent.title}
          </h2>
          <p className="text-muted-foreground mt-1">
            {currentStepComponent.description}
          </p>
        </div>

        {/* Composant de l'étape actuelle */}
        <div className="min-h-[400px]">
          <StepComponent
            data={getStepData(currentStep)}
            checkoutData={checkoutData}
            onDataChange={(data) => updateStepData(currentStep, data)}
            onNext={nextStep}
            onPrev={prevStep}
            isProcessing={isProcessing}
          />
        </div>

        {/* Navigation */}
        <div className="flex justify-between pt-6 border-t">
          <Button
            variant="outline"
            onClick={prevStep}
            disabled={currentStep === 0 || isProcessing}
          >
            <ArrowLeft className="mr-2 h-4 w-4" />
            Précédent
          </Button>
```

```
            {currentStep === checkoutSteps.length - 1 ? (
              <Button
                onClick={finalizeOrder}
                disabled={isProcessing}
                size="lg"
                className="min-w-[140px]"
              >
                {isProcessing ? (
                  <>
                    <Loader2 className="mr-2 h-4 w-4 animate-spin" />
                    Traitement...
                  </>
                ) : (
                  <>
                    <CreditCard className="mr-2 h-4 w-4" />
                    Finaliser
                  </>
                )}
              </Button>
            ) : (
              <Button
                onClick={nextStep}
                disabled={!completedSteps.has(currentStep) || isProcessing}
              >
                Suivant
                <ArrowRight className="ml-2 h-4 w-4" />
              </Button>
            )}
          </div>
        </div>
      </Card>
    </div>

    {/* Résumé de commande fixe */}
    <div className="lg:col-span-1">
      <div className="sticky top-8">
        <OrderSummary
          cart={checkoutData.cart}
          totals={checkoutData.totals}
          shipping={checkoutData.shipping}
          showEditLink={currentStep > 0}
          onEdit={() => setCurrentStep(0)}
        />

        {/* Badges de confiance */}
        <Card className="mt-4 p-4">
```

```
                <TrustIndicators
                  indicators={[
                    { icon: Shield, text: "Paiement sécurisé SSL" },
                    { icon: Truck, text: "Livraison rapide" },
                    { icon: RefreshCw, text: "Retours gratuits 30j" }
                  ]}
                />
              </Card>
            </div>
          </div>
        </div>
      </div>
    </div>
  )
}
```

Backend et API

Architecture API REST

Localisation : `server/routes/`, `server/services/`

Routes Principales et Spécifications

Authentification (`server/routes/auth.js`) Base URL : `/api/auth`

```
// POST /api/auth/login - Connexion utilisateur
router.post('/login', [
  // Validation des données
  body('email').isEmail().normalizeEmail(),
  body('password').isLength({ min: 8 }).trim(),

  // Rate limiting spécifique
  rateLimit({
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: 5, // 5 tentatives par IP
    message: { error: 'Trop de tentatives de connexion' }
  }),

  // Middleware de validation
  validationErrorHandler
], async (req, res) => {
  try {
    const { email, password, rememberMe } = req.body
```

```
// Vérification des credentials
const user = await authService.authenticate(email, password)

if (!user) {
  // Log des tentatives échouées pour sécurité
  logger.warn('Tentative de connexion échouée', {
    email,
    ip: req.ip,
    userAgent: req.get('User-Agent')
  })

  return res.status(401).json({
    error: 'Identifiants incorrects',
    code: 'INVALID_CREDENTIALS'
  })
}

// Génération des tokens JWT
const accessToken = generateAccessToken(user, rememberMe ? '30d' : '15m')
const refreshToken = generateRefreshToken(user, '7d')

// Stockage du refresh token (sécurisé)
await authService.storeRefreshToken(user.id, refreshToken)

// Log de connexion réussie
logger.info('Connexion réussie', {
  userId: user.id,
  email: user.email,
  ip: req.ip
})

// Configuration des cookies sécurisés
const cookieOptions = {
  httpOnly: true,
  secure: process.env.NODE_ENV === 'production',
  sameSite: 'strict',
  maxAge: rememberMe ? 30 * 24 * 60 * 60 * 1000 : 24 * 60 * 60 * 1000
}

res.cookie('refreshToken', refreshToken, cookieOptions)

res.json({
  success: true,
  user: {
    id: user.id,
    email: user.email,
```

```
        nom: user.nom,
        prenom: user.prenom,
        role: user.role,
        permissions: user.permissions || []
      },
      accessToken,
      expiresIn: rememberMe ? '30d' : '15m'
    })

  } catch (error) {
    logger.error('Erreur lors de la connexion', {
      error: error.message,
      stack: error.stack
    })

    res.status(500).json({
      error: 'Erreur interne du serveur',
      code: 'INTERNAL_SERVER_ERROR'
    })
  }
})

// POST /api/auth/refresh - Renouvellement du token
router.post('/refresh', async (req, res) => {
  try {
    const { refreshToken } = req.cookies

    if (!refreshToken) {
      return res.status(401).json({
        error: 'Token de rafraîchissement manquant',
        code: 'MISSING_REFRESH_TOKEN'
      })
    }

    // Vérification et validation du refresh token
    const payload = jwt.verify(refreshToken, process.env.JWT_REFRESH_SECRET)
    const user = await getUserById(payload.userId)

    if (!user || !await authService.isRefreshTokenValid(user.id, refreshToken)) {
      return res.status(401).json({
        error: 'Token de rafraîchissement invalide',
        code: 'INVALID_REFRESH_TOKEN'
      })
    }

    // Génération d'un nouveau access token
```

```
    const newAccessToken = generateAccessToken(user, '15m')

    res.json({
      success: true,
      accessToken: newAccessToken,
      expiresIn: '15m'
    })

  } catch (error) {
    logger.error('Erreur lors du refresh token', {
      error: error.message
    })

    res.status(401).json({
      error: 'Token de rafraîchissement expiré',
      code: 'REFRESH_TOKEN_EXPIRED'
    })
  }
})

// POST /api/auth/logout - Déconnexion
router.post('/logout', authenticateToken, async (req, res) => {
  try {
    // Invalidation du refresh token
    const { refreshToken } = req.cookies
    if (refreshToken) {
      await authService.revokeRefreshToken(req.user.id, refreshToken)
    }

    // Suppression des cookies
    res.clearCookie('refreshToken')

    logger.info('Déconnexion', { userId: req.user.id })

    res.json({
      success: true,
      message: 'Déconnexion réussie'
    })

  } catch (error) {
    logger.error('Erreur lors de la déconnexion', {
      error: error.message,
      userId: req.user?.id
    })

    res.status(500).json({
```

```
      error: 'Erreur lors de la déconnexion',
      code: 'LOGOUT_ERROR'
    })
  }
})
```

Produits (`server/routes/products.js`) Base URL : `/api/products`

```
// GET /api/products - Liste des produits avec filtres avancés
router.get('/', [
  // Validation des paramètres de requête
  query('category').optional().isString(),
  query('search').optional().isString().isLength({ max: 100 }),
  query('minPrice').optional().isFloat({ min: 0 }),
  query('maxPrice').optional().isFloat({ min: 0 }),
  query('inStock').optional().isBoolean(),
  query('onSale').optional().isBoolean(),
  query('sortBy').optional().isIn(['price', 'name', 'createdAt', 'popularity']),
  query('sortOrder').optional().isIn(['asc', 'desc']),
  query('limit').optional().isInt({ min: 1, max: 100 }),
  query('offset').optional().isInt({ min: 0 }),

  validationErrorHandler
], async (req, res) => {
  try {
    // Parsing des paramètres avec valeurs par défaut
    const filters = {
      category: req.query.category,
      search: req.query.search,
      priceRange: {
        min: parseFloat(req.query.minPrice) || 0,
        max: parseFloat(req.query.maxPrice) || Number.MAX_SAFE_INTEGER
      },
      inStock: req.query.inStock === 'true',
      onSale: req.query.onSale === 'true',
      sortBy: req.query.sortBy || 'createdAt',
      sortOrder: req.query.sortOrder || 'desc',
      pagination: {
        limit: parseInt(req.query.limit) || 20,
        offset: parseInt(req.query.offset) || 0
      }
    }

    // Récupération des produits avec filtres
    const result = await productsService.getFilteredProducts(filters)
```

```
    // Enrichissement des données (prix avec promotions, stock, etc.)
    const enrichedProducts = await Promise.all(
      result.products.map(async (product) => ({
        ...product,
        finalPrice: await calculateFinalPrice(product),
        isInStock: product.stock > 0,
        stockStatus: getStockStatus(product.stock),
        images: product.images?.map(img => `${process.env.BASE_URL}/uploads/${img}`) || []
      }))
    )

    // Méta-données pour la pagination
    const totalCount = await productsService.getProductCount(filters)
    const hasMore = (filters.pagination.offset + filters.pagination.limit) < totalCount

    res.json({
      success: true,
      products: enrichedProducts,
      pagination: {
        total: totalCount,
        offset: filters.pagination.offset,
        limit: filters.pagination.limit,
        hasMore
      },
      filters: {
        applied: Object.entries(filters)
          .filter(([key, value]) => value !== undefined && value !== '')
          .reduce((acc, [key, value]) => ({ ...acc, [key]: value }), {}),
        available: await productsService.getAvailableFilters()
      }
    })

  } catch (error) {
    logger.error('Erreur lors de la récupération des produits', {
      error: error.message,
      filters: req.query
    })

    res.status(500).json({
      error: 'Erreur lors de la récupération des produits',
      code: 'PRODUCTS_FETCH_ERROR'
    })
  }
})

// POST /api/products - Création d'un produit (Admin uniquement)
```

```
router.post('/', [
  authenticateToken,
  requireRole(['admin', 'manager']),
  upload.array('images', 6), // Maximum 6 images

  // Validation des données produit
  body('nom').isLength({ min: 2, max: 255 }).trim(),
  body('description').isLength({ min: 10, max: 2000 }).trim(),
  body('prix').isFloat({ min: 0.01 }),
  body('stock').isInt({ min: 0 }),
  body('categories').isArray().notEmpty(),
  body('weight').optional().isFloat({ min: 0 }),
  body('dimensions').optional().isObject(),

  validationErrorHandler
], async (req, res) => {
  try {
    // Parsing des données JSON dans le champ 'data'
    const productData = JSON.parse(req.body.data || '{}')

    // Validation des catégories
    const validCategories = await categoriesService.validateCategories(productData.categorie
    if (!validCategories) {
      return res.status(400).json({
        error: 'Une ou plusieurs catégories sont invalides',
        code: 'INVALID_CATEGORIES'
      })
    }

    // Traitement des images uploadées
    const processedImages = []
    if (req.files && req.files.length > 0) {
      for (const file of req.files) {
        // Validation et optimisation de l'image
        const processedImage = await imageService.processImage(file, {
          maxWidth: 800,
          maxHeight: 800,
          quality: 85,
          format: 'webp'
        })
        processedImages.push(processedImage.filename)
      }
    }

    // Génération du slug SEO automatique
    const slug = generateSlug(productData.nom)
```

```
    // Construction de l'objet produit complet
    const newProduct = {
      id: generateUniqueId(),
      ...productData,
      slug,
      images: processedImages,
      createdAt: new Date().toISOString(),
      updatedAt: new Date().toISOString(),
      createdBy: req.user.id,
      status: 'active',

      // Méta-données SEO automatiques si non fournies
      metaTitle: productData.metaTitle || productData.nom,
      metaDescription: productData.metaDescription ||
        productData.description.substring(0, 160) + '...',

      // Statistiques initiales
      viewCount: 0,
      purchaseCount: 0,
      averageRating: 0,
      reviewCount: 0
    }

    // Création du produit
    const createdProduct = await productsService.createProduct(newProduct)

    // Indexation pour la recherche (optionnel)
    await searchService.indexProduct(createdProduct)

    // Log de l'action admin
    logger.info('Produit créé', {
      productId: createdProduct.id,
      adminId: req.user.id,
      productName: createdProduct.nom
    })

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

  } catch (error) {
    logger.error('Erreur lors de la création du produit', {
      error: error.message,
```

```
      adminId: req.user?.id,
      productData: req.body
    })

    // Nettoyage des fichiers uploadés en cas d'erreur
    if (req.files) {
      req.files.forEach(file => {
        fs.unlink(file.path, () => {}) // Suppression silencieuse
      })
    }

    res.status(500).json({
      error: 'Erreur lors de la création du produit',
      code: 'PRODUCT_CREATION_ERROR'
    })
  }
})
```

Services Backend Spécialisés

Service d'Authentification (server/services/auth.service.js)

```
class AuthService {
  constructor() {
    this.users = this.loadUsers()
    this.refreshTokens = new Map() // En production: Redis ou DB
    this.loginAttempts = new Map()
    this.blockedIPs = new Set()
  }

  async authenticate(email, password) {
    const clientIP = this.getCurrentClientIP()

    // Vérification du blocage IP
    if (this.blockedIPs.has(clientIP)) {
      throw new Error('IP temporairement bloquée')
    }

    // Vérification des tentatives de connexion
    const attempts = this.loginAttempts.get(email) || { count: 0, lastAttempt: 0 }

    if (attempts.count >= 5) {
      const timeSinceLastAttempt = Date.now() - attempts.lastAttempt
      if (timeSinceLastAttempt < 15 * 60 * 1000) { // 15 minutes
        throw new Error('Compte temporairement verrouillé')
      } else {
```

```
      // Réinitialisation après 15 minutes
      this.loginAttempts.delete(email)
    }
  }

  // Recherche de l'utilisateur
  const user = this.users.find(u => u.email.toLowerCase() === email.toLowerCase())

  if (!user) {
    this.recordFailedAttempt(email)
    return null
  }

  // Vérification du mot de passe
  const isPasswordValid = await bcrypt.compare(password, user.password)

  if (!isPasswordValid) {
    this.recordFailedAttempt(email)
    return null
  }

  // Vérification du statut du compte
  if (user.status === 'blocked' || user.status === 'inactive') {
    throw new Error('Compte désactivé')
  }

  // Connexion réussie - réinitialisation des tentatives
  this.loginAttempts.delete(email)

  // Mise à jour de la dernière connexion
  user.lastLogin = new Date().toISOString()
  user.loginCount = (user.loginCount || 0) + 1

  // Sauvegarde des modifications
  await this.saveUsers()

  // Retour des informations utilisateur (sans mot de passe)
  const { password: _, ...userWithoutPassword } = user
  return userWithoutPassword
}

recordFailedAttempt(email) {
  const current = this.loginAttempts.get(email) || { count: 0, lastAttempt: 0 }
  this.loginAttempts.set(email, {
    count: current.count + 1,
    lastAttempt: Date.now()
```

```
  })

  // Blocage IP après 20 tentatives échouées
  const totalAttempts = Array.from(this.loginAttempts.values())
    .reduce((sum, attempt) => sum + attempt.count, 0)

  if (totalAttempts >= 20) {
    const clientIP = this.getCurrentClientIP()
    this.blockedIPs.add(clientIP)

    // Déblocage automatique après 1 heure
    setTimeout(() => {
      this.blockedIPs.delete(clientIP)
    }, 60 * 60 * 1000)
  }
}

async generateTokens(user, rememberMe = false) {
  const accessTokenExpiry = rememberMe ? '30d' : '15m'
  const refreshTokenExpiry = '7d'

  const accessTokenPayload = {
    userId: user.id,
    email: user.email,
    role: user.role,
    permissions: user.permissions || [],
    type: 'access'
  }

  const refreshTokenPayload = {
    userId: user.id,
    type: 'refresh',
    tokenId: generateUniqueId()
  }

  const accessToken = jwt.sign(
    accessTokenPayload,
    process.env.JWT_SECRET,
    { expiresIn: accessTokenExpiry }
  )

  const refreshToken = jwt.sign(
    refreshTokenPayload,
    process.env.JWT_REFRESH_SECRET,
    { expiresIn: refreshTokenExpiry }
  )
```

```
  // Stockage sécurisé du refresh token
  await this.storeRefreshToken(user.id, refreshToken, refreshTokenPayload.tokenId)

  return {
    accessToken,
    refreshToken,
    expiresIn: accessTokenExpiry
  }
}

async storeRefreshToken(userId, token, tokenId) {
  const tokenData = {
    token,
    tokenId,
    userId,
    createdAt: Date.now(),
    isActive: true
  }

  // Stockage en mémoire (en production: Redis/DB)
  if (!this.refreshTokens.has(userId)) {
    this.refreshTokens.set(userId, [])
  }

  const userTokens = this.refreshTokens.get(userId)

  // Limitation à 5 tokens actifs par utilisateur
  if (userTokens.length >= 5) {
    userTokens.shift() // Suppression du plus ancien
  }

  userTokens.push(tokenData)
}

async isRefreshTokenValid(userId, token) {
  const userTokens = this.refreshTokens.get(userId) || []
  return userTokens.some(tokenData =>
    tokenData.token === token && tokenData.isActive
  )
}

async revokeRefreshToken(userId, token) {
  const userTokens = this.refreshTokens.get(userId) || []
  const tokenIndex = userTokens.findIndex(t => t.token === token)
```

```
    if (tokenIndex !== -1) {
      userTokens[tokenIndex].isActive = false
    }
  }

  async revokeAllRefreshTokens(userId) {
    const userTokens = this.refreshTokens.get(userId) || []
    userTokens.forEach(token => token.isActive = false)
  }

  // Méthodes utilitaires
  getCurrentClientIP() {
    // À implémenter selon le contexte (req.ip, req.connection.remoteAddress, etc.)
    return '127.0.0.1' // Placeholder
  }

  async loadUsers() {
    try {
      const usersData = await fs.readFile(path.join(__dirname, '../data/users.json'), 'utf8'
      return JSON.parse(usersData)
    } catch (error) {
      logger.error('Erreur lors du chargement des utilisateurs', { error: error.message })
      return []
    }
  }

  async saveUsers() {
    try {
      await fs.writeFile(
        path.join(__dirname, '../data/users.json'),
        JSON.stringify(this.users, null, 2),
        'utf8'
      )
    } catch (error) {
      logger.error('Erreur lors de la sauvegarde des utilisateurs', { error: error.message }
    }
  }
}

module.exports = new AuthService()
```

Monitoring et Maintenance

Système de Logging Avancé

Localisation : `server/utils/logger.js`

```
const winston = require('winston')
const path = require('path')

// Configuration des niveaux de log personnalisés
const customLevels = {
  levels: {
    error: 0,
    warn: 1,
    info: 2,
    http: 3,
    debug: 4
  },
  colors: {
    error: 'red',
    warn: 'yellow',
    info: 'green',
    http: 'magenta',
    debug: 'blue'
  }
}

// Formatage personnalisé des logs
const logFormat = winston.format.combine(
  winston.format.timestamp({ format: 'YYYY-MM-DD HH:mm:ss:ms' }),
  winston.format.colorize({ all: true }),
  winston.format.printf(info => {
    if (info.stack) {
      return `${info.timestamp} ${info.level}: ${info.message}\n${info.stack}`
    }
    return `${info.timestamp} ${info.level}: ${info.message} ${
      info.meta ? JSON.stringify(info.meta) : ''
    }`
  })
)

// Configuration du logger principal
const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  levels: customLevels.levels,
  format: winston.format.combine(
    winston.format.timestamp(),
```

```
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  defaultMeta: {
    service: 'riziky-boutic-api',
    version: process.env.npm_package_version || '1.0.0'
  },
  transports: [
    // Fichier pour toutes les erreurs
    new winston.transports.File({
      filename: path.join(__dirname, '../logs/error.log'),
      level: 'error',
      maxsize: 10485760, // 10MB
      maxFiles: 5,
      tailable: true
    }),

    // Fichier pour tous les logs
    new winston.transports.File({
      filename: path.join(__dirname, '../logs/combined.log'),
      maxsize: 10485760, // 10MB
      maxFiles: 5,
      tailable: true
    }),

    // Fichier spécialisé pour les requêtes HTTP
    new winston.transports.File({
      filename: path.join(__dirname, '../logs/access.log'),
      level: 'http',
      maxsize: 10485760,
      maxFiles: 3
    }),

    // Logs de sécurité séparés
    new winston.transports.File({
      filename: path.join(__dirname, '../logs/security.log'),
      level: 'warn',
      maxsize: 5242880, // 5MB
      maxFiles: 10
    })
  ],

  // Gestion des exceptions non capturées
  exceptionHandlers: [
    new winston.transports.File({
      filename: path.join(__dirname, '../logs/exceptions.log')
```

```
    })
  ],

  // Gestion des rejets de promesses
  rejectionHandlers: [
    new winston.transports.File({
      filename: path.join(__dirname, '../logs/rejections.log')
    })
  ]
})

// Console en développement
if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: logFormat,
    level: 'debug'
  }))
}

// Ajout des couleurs
winston.addColors(customLevels.colors)

// Middleware de logging des requêtes HTTP
const httpLogger = (req, res, next) => {
  const start = Date.now()

  // Log de la requête entrante
  logger.http(`${req.method} ${req.originalUrl}`, {
    method: req.method,
    url: req.originalUrl,
    ip: req.ip,
    userAgent: req.get('User-Agent'),
    userId: req.user?.id,
    timestamp: new Date().toISOString()
  })

  // Hook sur la fin de la réponse
  res.on('finish', () => {
    const duration = Date.now() - start
    const level = res.statusCode >= 400 ? 'warn' : 'http'

    logger.log(level, `${req.method} ${req.originalUrl} ${res.statusCode} ${duration}ms`, {
      method: req.method,
      url: req.originalUrl,
      statusCode: res.statusCode,
      responseTime: duration,
```

```
      ip: req.ip,
      userId: req.user?.id,
      contentLength: res.get('Content-Length')
    })
  })

  next()
}

// Fonction de logging de sécurité
const securityLogger = {
  logFailedLogin: (email, ip, userAgent) => {
    logger.warn('Tentative de connexion échouée', {
      event: 'login_failed',
      email,
      ip,
      userAgent,
      timestamp: new Date().toISOString()
    })
  },

  logSuspiciousActivity: (activity, userId, ip, details = {}) => {
    logger.warn('Activité suspecte détectée', {
      event: 'suspicious_activity',
      activity,
      userId,
      ip,
      details,
      timestamp: new Date().toISOString()
    })
  },

  logDataAccess: (resource, action, userId, success) => {
    logger.info('Accès aux données', {
      event: 'data_access',
      resource,
      action,
      userId,
      success,
      timestamp: new Date().toISOString()
    })
  }
}

// Fonction de logging métier
const businessLogger = {
```

```
  logOrder: (orderId, userId, amount, status) => {
    logger.info('Événement commande', {
      event: 'order_event',
      orderId,
      userId,
      amount,
      status,
      timestamp: new Date().toISOString()
    })
  },

  logPayment: (paymentId, orderId, amount, method, status) => {
    logger.info('Événement paiement', {
      event: 'payment_event',
      paymentId,
      orderId,
      amount,
      method,
      status,
      timestamp: new Date().toISOString()
    })
  },

  logProductAction: (action, productId, userId, details = {}) => {
    logger.info('Action produit', {
      event: 'product_action',
      action,
      productId,
      userId,
      details,
      timestamp: new Date().toISOString()
    })
  }
}

module.exports = {
  logger,
  httpLogger,
  securityLogger,
  businessLogger
}
```

Monitoring des Performances

Localisation : `server/middlewares/monitoring.js`

```
const performanceMonitoring = () => {
  // Métriques en mémoire (en production: Prometheus/InfluxDB)
  const metrics = {
    requests: {
      total: 0,
      byStatus: {},
      byEndpoint: {},
      averageResponseTime: 0
    },
    errors: {
      total: 0,
      by4xx: 0,
      by5xx: 0,
      byType: {}
    },
    system: {
      memoryUsage: process.memoryUsage(),
      cpuUsage: process.cpuUsage(),
      uptime: process.uptime()
    }
  }

  // Mise à jour périodique des métriques système
  setInterval(() => {
    metrics.system = {
      memoryUsage: process.memoryUsage(),
      cpuUsage: process.cpuUsage(),
      uptime: process.uptime()
    }
  }, 30000) // Toutes les 30 secondes

  return (req, res, next) => {
    const startTime = process.hrtime.bigint()

    // Comptage des requêtes
    metrics.requests.total++

    // Comptage par endpoint
    const endpoint = `${req.method} ${req.route?.path || req.path}`
    metrics.requests.byEndpoint[endpoint] = (metrics.requests.byEndpoint[endpoint] || 0) + 1

    // Hook sur la fin de la réponse
    res.on('finish', () => {
      const endTime = process.hrtime.bigint()
      const responseTime = Number(endTime - startTime) / 1000000 // Conversion en milliseco
```

```
      // Mise à jour des métriques de réponse
      const status = res.statusCode
      metrics.requests.byStatus[status] = (metrics.requests.byStatus[status] || 0) + 1

      // Calcul de la moyenne des temps de réponse
      metrics.requests.averageResponseTime =
        (metrics.requests.averageResponseTime * (metrics.requests.total - 1) + responseTime)
        metrics.requests.total

      // Comptage des erreurs
      if (status >= 400) {
        metrics.errors.total++
        if (status < 500) {
          metrics.errors.by4xx++
        } else {
          metrics.errors.by5xx++
        }
      }

      // Alerte sur temps de réponse élevé
      if (responseTime > 5000) { // 5 secondes
        logger.warn('Temps de réponse élevé détecté', {
          endpoint,
          responseTime,
          statusCode: status,
          userAgent: req.get('User-Agent')
        })
      }

      // Alerte sur erreur 5xx
      if (status >= 500) {
        logger.error('Erreur serveur détectée', {
          endpoint,
          statusCode: status,
          responseTime,
          userId: req.user?.id,
          ip: req.ip
        })
      }
    })

    next()
  }
}

// Endpoint de health check avec métriques détaillées
```

```
const healthCheck = (req, res) => {
  const health = {
    status: 'healthy',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    memory: process.memoryUsage(),
    cpu: process.cpuUsage(),
    requests: metrics.requests,
    errors: metrics.errors,
    environment: process.env.NODE_ENV,
    version: process.env.npm_package_version
  }

  // Vérification de la santé du système
  const memoryUsage = process.memoryUsage()
  const memoryUsagePercent = (memoryUsage.heapUsed / memoryUsage.heapTotal) * 100

  if (memoryUsagePercent > 90) {
    health.status = 'warning'
    health.warnings = health.warnings || []
    health.warnings.push('Utilisation mémoire élevée')
  }

  if (metrics.errors.by5xx > 10) { // Plus de 10 erreurs 5xx
    health.status = 'unhealthy'
    health.errors = health.errors || []
    health.errors.push('Trop d\'erreurs serveur')
  }

  const statusCode = health.status === 'healthy' ? 200 :
                     health.status === 'warning' ? 200 : 503

  res.status(statusCode).json(health)
}

// Endpoint de métriques (format Prometheus compatible)
const metricsEndpoint = (req, res) => {
  let output = ''

  // Format Prometheus
  output += '# HELP http_requests_total Total number of HTTP requests\n'
  output += '# TYPE http_requests_total counter\n'
  output += `http_requests_total ${metrics.requests.total}\n\n`

  output += '# HELP http_request_duration_ms Average HTTP request duration\n'
  output += '# TYPE http_request_duration_ms gauge\n'
```

```
  output += `http_request_duration_ms ${metrics.requests.averageResponseTime.toFixed(2)}\n\n`

  output += '# HELP http_errors_total Total number of HTTP errors\n'
  output += '# TYPE http_errors_total counter\n'
  output += `http_errors_total ${metrics.errors.total}\n\n`

  // Métriques par status code
  Object.entries(metrics.requests.byStatus).forEach(([status, count]) => {
    output += `http_requests_by_status{status="${status}"} ${count}\n`
  })

  res.set('Content-Type', 'text/plain')
  res.send(output)
}

module.exports = {
  performanceMonitoring,
  healthCheck,
  metricsEndpoint
}
```

Ce cahier des charges final constitue la documentation technique complète de la plateforme Riziky-Boutic. Il détaille l'ensemble des spécifications, composants, logiques métier et processus nécessaires pour comprendre, utiliser, maintenir et faire évoluer la solution e-commerce.

Résumé des Livrables : - Architecture technique complète et documentée - Spécifications détaillées de tous les composants - Documentation des APIs et services backend - Guides d'utilisation pour chaque interface - Procédures de maintenance et monitoring - Standards de sécurité et bonnes pratiques - Instructions de déploiement et mise en production

Cette documentation permet à toute équipe technique de reprendre, maintenir et faire évoluer la plateforme en toute autonomie.