Guide Développeur Complet - Riziky-Boutic

Vue d'Ensemble pour Développeurs

Ce guide complet s'adresse aux développeurs qui doivent maintenir, modifier ou étendre l'application Riziky-Boutic. Il couvre l'architecture technique, les composants, les patterns utilisés et les bonnes pratiques de développement.

Architecture Technique Détaillée

Structure des Répertoires

```
riziky-boutic/
  src/                          # Code source frontend React
     components/                # Composants réutilisables
        ui/                     # Composants UI de base (shadcn/ui)
        layout/                 # Composants de mise en page
        auth/                   # Composants d'authentification
        products/               # Composants produits
        cart/                   # Composants panier
        admin/                  # Composants administration
        modern/                 # Composants interface moderne
        common/                 # Composants communs
     pages/                     # Pages de l'application
     hooks/                     # Hooks personnalisés React
     contexts/                  # Contextes React (état global)
     services/                  # Services API et logique métier
     lib/                       # Utilitaires et helpers
     types/                     # Définitions TypeScript
  server/                       # Code source backend Node.js
     routes/                    # Routes API Express
     services/                  # Services backend
     middlewares/               # Middlewares Express
     config/                    # Configuration serveur
     data/                      # Fichiers JSON (base de données)
     socket/                    # Configuration WebSocket
  docs/                         # Documentation complète
```

Stack Technologique

Frontend

- **React 18.3.1** : Framework UI avec hooks et composants fonctionnels
- **TypeScript** : Typage statique pour sécurité et maintenabilité
- **Tailwind CSS** : Framework CSS utilitaire pour styling
- **Shadcn/UI** : Composants UI pré-construits

- **React Router** : Gestion du routing côté client
- **React Hook Form** : Gestion des formulaires
- **Zod** : Validation des schémas de données
- **Axios** : Client HTTP pour API calls
- **Socket.io Client** : Communication temps réel

Backend

- **Node.js** : Runtime JavaScript serveur
- **Express.js** : Framework web minimaliste
- **Socket.io** : WebSocket pour temps réel
- **Multer** : Upload de fichiers
- **Helmet** : Sécurité headers HTTP
- **bcrypt** : Hachage des mots de passe

Guide des Composants par Catégorie

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

`Button` - Bouton Réutilisable **Fichier :** `src/components/ui/button.tsx`
Utilisation :

```
import { Button } from "@/components/ui/button"

// Variantes disponibles
<Button variant="default">Bouton Principal</Button>
<Button variant="destructive">Supprimer</Button>
<Button variant="outline">Contour</Button>
<Button variant="secondary">Secondaire</Button>
<Button variant="ghost">Transparent</Button>
<Button variant="link">Lien</Button>

// Tailles disponibles
<Button size="default">Normal</Button>
<Button size="sm">Petit</Button>
<Button size="lg">Grand</Button>
<Button size="icon">Icône uniquement</Button>
```

Comment modifier :

```
// Dans button.tsx, modifier les variantes
const buttonVariants = cva(
  "base-classes",
  {
    variants: {
      variant: {
        // Ajouter nouvelle variante
```

```
        success: "bg-green-600 text-white hover:bg-green-700"
      }
    }
  }
)
```

Input - Champ de Saisie Fichier : `src/components/ui/input.tsx` **Utilisation :**

```
import { Input } from "@/components/ui/input"

<Input
  type="email"
  placeholder="Email"
  value={email}
  onChange={(e) => setEmail(e.target.value)}
  className="additional-classes"
/>
```

Comment étendre :

```
// Créer un composant Input spécialisé
const EmailInput = React.forwardRef<HTMLInputElement, InputProps>(
  ({ ...props }, ref) => {
    return (
      <Input
        ref={ref}
        type="email"
        {...props}
        className={cn("email-specific-classes", props.className)}
      />
    )
  }
)
```

Card - Conteneur Structuré Fichier : `src/components/ui/card.tsx` **Structure :**

```
<Card>
  <CardHeader>
    <CardTitle>Titre</CardTitle>
    <CardDescription>Description</CardDescription>
  </CardHeader>
  <CardContent>Contenu</CardContent>
  <CardFooter>Actions</CardFooter>
</Card>
```

Composants Layout (`src/components/layout/`)

`Navbar` - Navigation Principale **Fichier :** `src/components/layout/Navbar.tsx`
Responsabilités : - Affichage du logo et navigation - Menu utilisateur avec dropdown - Panier et favoris - Recherche - Navigation mobile responsive

Comment modifier :

```
// Ajouter un nouvel élément de navigation
const navigationItems = [
  { label: 'Accueil', href: '/', icon: Home },
  // Ajouter ici
  { label: 'Nouveau', href: '/nouveau', icon: Plus }
]

// Modifier le style
<nav className="hidden md:flex items-center space-x-6">
  {/* Navigation desktop */}
</nav>
```

`Footer` - Pied de Page **Fichier :** `src/components/layout/Footer.tsx`
Sections : - Informations entreprise - Liens utiles - Contact - Réseaux sociaux - Mentions légales

Composants Authentification (`src/components/auth/`)

Gestion Complète de l'Auth **Fichiers principaux :** - `LoginForm.tsx` - Formulaire de connexion - `RegisterForm.tsx` - Formulaire d'inscription - `PasswordStrengthIndicator.tsx` - Indicateur force mot de passe

Pattern de validation :

```
// Utilisation de Zod pour validation
const loginSchema = z.object({
  email: z.string().email("Email invalide"),
  password: z.string().min(8, "Minimum 8 caractères")
})

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

Composants E-commerce (`src/components/products/`, `src/components/cart/`)

`ProductCard` - Carte Produit **Fichier :** `src/components/products/ProductCard.tsx`
Fonctionnalités : - Affichage image, nom, prix - Gestion des promotions - Ajout au panier - Ajout aux favoris - Vue rapide

Comment personnaliser :

```
interface ProductCardProps {
  product: Product;
  variant?: 'default' | 'compact' | 'detailed';
  showActions?: boolean;
  showQuickView?: boolean;
}

// Ajouter nouvelle variante
const cardVariants = {
  default: "max-w-sm",
  compact: "max-w-xs",
  detailed: "max-w-md",
  // Nouvelle variante
  featured: "max-w-lg bg-gradient-to-br from-primary/5"
}
```

`CartDrawer` - Panier Latéral **Fichier :** `src/components/cart/CartDrawer.tsx`
Fonctionnalités : - Affichage items du panier - Modification quantités - Suppression d'articles - Calcul totaux et livraison - Navigation vers checkout

État du panier géré par :

```
// Hook personnalisé
const {
  cart,
  addToCart,
  removeFromCart,
  updateQuantity,
  totalPrice,
  itemCount
} = useCart()
```

Composants Administrateur (`src/components/admin/`)

`ProductForm` - Création/Édition Produits **Fichier :** `src/components/admin/ProductForm.tsx`
Fonctionnalités : - Formulaire complet produit - Upload d'images multiples - Gestion catégories - Validation côté client - Prévisualisation

Comment étendre :

```
// Ajouter nouveaux champs au schéma
const productSchema = z.object({
  nom: z.string().min(1, "Nom requis"),
  description: z.string(),
  prix: z.number().positive(),
  // Nouveaux champs
```

```
  weight: z.number().optional(),
  dimensions: z.string().optional()
})
```

`AdminPageHeader` - En-tête Pages Admin **Fichier :** `src/components/admin/AdminPageHeader.tsx` **Pattern réutilisable :**

```
<AdminPageHeader
  title="Gestion des Produits"
  description="Créer et gérer les produits"
  action={
    <Button onClick={() => setShowForm(true)}>
      <Plus className="mr-2 h-4 w-4" />
      Nouveau Produit
    </Button>
  }
/>
```

Composants Interface Moderne (`src/components/modern/`)

`FloatingActionButton` - Bouton d'Action Flottant **Fichier :** `src/components/modern/FloatingActionButton.tsx` **Utilisation :**

```
const actions = [
  {
    icon: Plus,
    label: "Nouveau RDV",
    onClick: () => setShowAppointmentModal(true)
  },
  {
    icon: UserPlus,
    label: "Nouveau Client",
    onClick: () => setShowClientModal(true)
  }
]

<FloatingActionButton actions={actions} />
```

`ModernCalendar` - Calendrier Interface Moderne **Fichier :** `src/components/modern/ModernCalendar.` **Fonctionnalités :** - Vue mensuelle avec navigation - Affichage rendez-vous - Interactions tactiles - Responsive design

---

Hooks Personnalisés

useAuth - Gestion Authentification

Fichier : `src/hooks/useAuth.ts` (via AuthContext) **Utilisation :**

```
const {
  user,           // Utilisateur actuel
  login,          // Fonction connexion
  logout,         // Fonction déconnexion
  isAuthenticated // État authentifié
} = useAuth()

// Connexion
await login(email, password)

// Vérification auth
if (!isAuthenticated) {
  // Rediriger vers login
}
```

useCart - Gestion Panier

Fichier : `src/hooks/useCart.ts` **Fonctionnalités complètes :**

```
const {
  cart,              // Articles du panier
  addToCart,         // Ajouter un article
  removeFromCart,    // Supprimer un article
  updateQuantity,    // Modifier quantité
  clearCart,         // Vider le panier
  totalPrice,        // Prix total
  itemCount,         // Nombre d'articles
  isLoading         // État chargement
} = useCart()

// Utilisation
await addToCart(productId, quantity)
await updateQuantity(itemId, newQuantity)
```

useProducts - Gestion Produits

Fichier : `src/hooks/useProducts.ts` **API complète :**

```
const {
  products,          // Liste des produits
  loading,          // État chargement
  error,           // Erreurs
  searchProducts,   // Recherche
```

```
  filterProducts,   // Filtrage
  getProductById,   // Produit par ID
  refetch           // Recharger
} = useProducts()
```

Comment créer un nouveau Hook

```
// Template pour nouveau hook
import { useState, useEffect } from 'react'

export const useCustomHook = (initialValue?: any) => {
  const [state, setState] = useState(initialValue)
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState<string | null>(null)

  const customAction = async () => {
    setLoading(true)
    setError(null)

    try {
      // Logique métier
      const result = await api.call()
      setState(result)
    } catch (err) {
      setError(err.message)
    } finally {
      setLoading(false)
    }
  }

  return {
    state,
    loading,
    error,
    customAction
  }
}
```

Pages et Routing

Structure des Pages (`src/pages/`)

Pages Principales

- `HomePage.tsx` - Page d'accueil avec produits vedettes

- `ProductDetail.tsx` - Détail d'un produit
- `CartPage.tsx` - Page panier complète
- `CheckoutPage.tsx` - Processus de commande
- `LoginPage.tsx` / `RegisterPage.tsx` - Authentification

Pages Administrateur (`src/pages/admin/`)

- `AdminDashboardPage.tsx` - Tableau de bord
- `AdminProductsPage.tsx` - Gestion produits
- `AdminOrdersPage.tsx` - Gestion commandes
- `AdminUsersPage.tsx` - Gestion utilisateurs

Configuration Routing (`src/app/AppRoutes.tsx`)

```
// Structure du routing
<Routes>
  <Route path="/" element={<Layout />}>
    <Route index element={<HomePage />} />
    <Route path="products/:id" element={<ProductDetail />} />

    {/* Routes protégées */}
    <Route element={<ProtectedRoute />}>
      <Route path="cart" element={<CartPage />} />
      <Route path="profile" element={<ProfilePage />} />
    </Route>

    {/* Routes admin */}
    <Route path="admin" element={<AdminLayout />}>
      <Route index element={<AdminDashboard />} />
      <Route path="products" element={<AdminProducts />} />
    </Route>
  </Route>
</Routes>

// Comment ajouter une nouvelle route
<Route path="nouvelle-page" element={<NouvellePage />} />
```

Services API

Configuration Axios (`src/services/api.ts`)

```
// Instance Axios configurée
const api = axios.create({
  baseURL: process.env.VITE_API_BASE_URL || 'http://localhost:10000/api',
  timeout: 30000,
```

```
  headers: {
    'Content-Type': 'application/json'
  }
})

// Intercepteur pour authentification
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('authToken')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

Services Métier

authService.ts - Service Authentification

```
class AuthService {
  async login(email: string, password: string) {
    const response = await api.post('/auth/login', { email, password })
    return response.data
  }

  async register(userData: RegisterData) {
    const response = await api.post('/auth/register', userData)
    return response.data
  }

  async logout() {
    await api.post('/auth/logout')
    localStorage.removeItem('authToken')
  }
}

export const authService = new AuthService()
```

productsService.ts - Service Produits

```
class ProductsService {
  async getAll(params?: ProductFilters) {
    const response = await api.get('/products', { params })
    return response.data
  }

  async getById(id: string) {
    const response = await api.get(`/products/${id}`)
```

```
    return response.data
  }

  async create(productData: ProductFormData) {
    // Gestion upload images
    const formData = new FormData()
    formData.append('data', JSON.stringify(productData))
    productData.images?.forEach(image => {
      formData.append('images', image)
    })

    const response = await api.post('/products', formData, {
      headers: { 'Content-Type': 'multipart/form-data' }
    })
    return response.data
  }
}
```

Comment ajouter un nouveau Service

```
// Template service
class NewService {
  private baseEndpoint = '/new-endpoint'

  async getAll() {
    const response = await api.get(this.baseEndpoint)
    return response.data
  }

  async create(data: NewEntityData) {
    const response = await api.post(this.baseEndpoint, data)
    return response.data
  }

  async update(id: string, data: Partial<NewEntityData>) {
    const response = await api.put(`${this.baseEndpoint}/${id}`, data)
    return response.data
  }

  async delete(id: string) {
    await api.delete(`${this.baseEndpoint}/${id}`)
  }
}

export const newService = new NewService()
```

Système de Design et Styling

Configuration Tailwind (`tailwind.config.ts`)

```
// Thème personnalisé
theme: {
  extend: {
    colors: {
      // Couleurs sémantiques HSL
      primary: "hsl(var(--primary))",
      secondary: "hsl(var(--secondary))",
      destructive: "hsl(var(--destructive))",
      // Ajout nouvelles couleurs
      success: "hsl(142 76% 36%)",
      warning: "hsl(38 92% 50%)"
    }
  }
}
```

Variables CSS (`src/index.css`)

```
:root {
  /* Couleurs principales */
  --primary: 0 84% 60%;
  --primary-foreground: 0 0% 98%;

  /* Ajout nouvelles variables */
  --success: 142 76% 36%;
  --warning: 38 92% 50%;
}

/* Mode sombre */
[data-theme="dark"] {
  --primary: 0 84% 70%;
  /* Adapter les couleurs pour le mode sombre */
}
```

Utilitaire `cn` - Class Names

```
import { cn } from "@/lib/utils"

// Combinaison conditionnelle de classes
<div className={cn(
  "base-classes",
  variant === "primary" && "primary-classes",
```

```
  isActive && "active-classes",
  className  // Props externes
)} />
```

Composants Styled

```
// Pattern pour composant stylé
interface StyledComponentProps {
  variant?: 'default' | 'outlined' | 'filled'
  size?: 'sm' | 'md' | 'lg'
  className?: string
}

const StyledComponent = ({ variant = 'default', size = 'md', className, ...props }) => {
  return (
    <div className={cn(
      // Classes de base
      "rounded-md transition-colors",

      // Variantes
      {
        'default': 'bg-background border',
        'outlined': 'border-2 bg-transparent',
        'filled': 'bg-primary text-primary-foreground'
      }[variant],

      // Tailles
      {
        'sm': 'px-2 py-1 text-sm',
        'md': 'px-4 py-2',
        'lg': 'px-6 py-3 text-lg'
      }[size],

      className
    )} {...props} />
  )
}
```

Gestion de l'État Global

AuthContext - Contexte Authentification

Fichier : `src/contexts/AuthContext.tsx` **Responsabilités :** - État utilisateur global - Persistance de session - Auto-logout sur expiration token - Gestion des permissions

```
// Utilisation dans composants
const AuthProvider = ({ children }) => {
  const [user, setUser] = useState<User | null>(null)
  const [isAuthenticated, setIsAuthenticated] = useState(false)

  // Vérification token au chargement
  useEffect(() => {
    const token = localStorage.getItem('authToken')
    if (token) {
      verifyToken(token)
    }
  }, [])

  const login = async (email: string, password: string) => {
    const response = await authService.login(email, password)
    setUser(response.user)
    setIsAuthenticated(true)
    localStorage.setItem('authToken', response.token)
  }

  return (
    <AuthContext.Provider value={{ user, isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  )
}
```

StoreContext - État Application Global

Fichier : `src/contexts/StoreContext.tsx` **Gestion :** - Panier global - Favoris utilisateur - Préférences - Cache produits

Comment créer un nouveau Contexte

```
// Template pour nouveau contexte
interface NewContextType {
  state: any
  actions: {
    action1: () => void
    action2: (param: any) => void
  }
}

const NewContext = createContext<NewContextType | undefined>(undefined)

export const NewProvider = ({ children }: { children: ReactNode }) => {
```

```
  const [state, setState] = useState(initialState)

  const actions = {
    action1: () => {
      // Logique action 1
    },
    action2: (param) => {
      // Logique action 2
    }
  }

  return (
    <NewContext.Provider value={{ state, actions }}>
      {children}
    </NewContext.Provider>
  )
}

// Hook pour utiliser le contexte
export const useNewContext = () => {
  const context = useContext(NewContext)
  if (!context) {
    throw new Error('useNewContext must be used within NewProvider')
  }
  return context
}
```

Sécurité et Bonnes Pratiques

Validation des Données

```
// Schémas Zod pour validation
const userSchema = z.object({
  email: z.string().email(),
  password: z.string().min(8).regex(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/),
  nom: z.string().min(2),
  prenom: z.string().min(2)
})

// Validation dans composants
const form = useForm({
  resolver: zodResolver(userSchema),
  defaultValues: defaultUser
})
```

Protection des Routes

```
// ProtectedRoute pour routes authentifiées
const ProtectedRoute = () => {
  const { isAuthenticated } = useAuth()
  const location = useLocation()

  if (!isAuthenticated) {
    return <Navigate to="/login" state={{ from: location }} replace />
  }

  return <Outlet />
}

// AdminRoute pour routes admin uniquement
const AdminRoute = () => {
  const { user } = useAuth()

  if (user?.role !== 'admin') {
    return <Navigate to="/unauthorized" replace />
  }

  return <Outlet />
}
```

Sanitisation des Données

```
// Nettoyage des entrées utilisateur
const sanitizeInput = (input: string): string => {
  return input
    .trim()
    .replace(/[<>]/g, '') // Suppression balises HTML
    .substring(0, 1000)   // Limitation longueur
}

// Dans les formulaires
const handleSubmit = (data: FormData) => {
  const sanitizedData = {
    ...data,
    nom: sanitizeInput(data.nom),
    description: sanitizeInput(data.description)
  }
  // Traitement des données nettoyées
}
```

Debugging et Maintenance

Logs et Monitoring

```
// Système de logging structuré
const logger = {
  info: (message: string, data?: any) => {
    console.log(`[INFO] ${message}`, data)
  },
  error: (message: string, error?: Error) => {
    console.error(`[ERROR] ${message}`, error)
  },
  warn: (message: string, data?: any) => {
    console.warn(`[WARN] ${message}`, data)
  }
}

// Utilisation dans composants
useEffect(() => {
  logger.info('Composant monté', { componentName: 'ProductCard' })

  return () => {
    logger.info('Composant démonté', { componentName: 'ProductCard' })
  }
}, [])
```

Gestion d'Erreurs

```
// Hook pour gestion d'erreurs
const useErrorHandler = () => {
  const handleError = (error: Error, context?: string) => {
    logger.error(`Erreur dans ${context}`, error)

    // Affichage utilisateur
    toast.error(error.message || 'Une erreur est survenue')

    // Reporting (Sentry, LogRocket, etc.)
    // errorReporter.captureException(error)
  }

  return { handleError }
}

// Error Boundary pour React
class ErrorBoundary extends Component {
  constructor(props) {
    super(props)
```

```
    this.state = { hasError: false }
  }

  static getDerivedStateFromError(error) {
    return { hasError: true }
  }

  componentDidCatch(error, errorInfo) {
    logger.error('Error Boundary déclenché', { error, errorInfo })
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="error-fallback">
          <h2>Quelque chose s'est mal passé</h2>
          <button onClick={() => this.setState({ hasError: false })}>
            Réessayer
          </button>
        </div>
      )
    }

    return this.props.children
  }
}
```

Tests et Qualité Code

```
// Tests unitaires avec Jest/React Testing Library
import { render, screen, fireEvent } from '@testing-library/react'
import { ProductCard } from '../ProductCard'

describe('ProductCard', () => {
  const mockProduct = {
    id: '1',
    nom: 'Test Product',
    prix: 29.99,
    stock: 10
  }

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

    expect(screen.getByText('Test Product')).toBeInTheDocument()
    expect(screen.getByText('29.99€')).toBeInTheDocument()
```

```
  })

  test('ajoute au panier au clic', () => {
    const mockAddToCart = jest.fn()
    render(<ProductCard product={mockProduct} onAddToCart={mockAddToCart} />)

    fireEvent.click(screen.getByText('Ajouter au panier'))
    expect(mockAddToCart).toHaveBeenCalledWith(mockProduct.id, 1)
  })
})
```

Performance et Optimisations

Code Splitting et Lazy Loading

```
// Lazy loading des pages
const HomePage = lazy(() => import('../pages/HomePage'))
const ProductDetail = lazy(() => import('../pages/ProductDetail'))

// Utilisation avec Suspense
<Suspense fallback={<LoadingSpinner />}>
  <Routes>
    <Route path="/" element={<HomePage />} />
    <Route path="/products/:id" element={<ProductDetail />} />
  </Routes>
</Suspense>

// Lazy loading conditionnel
const AdminPanel = lazy(() =>
  import('../components/admin/AdminPanel').then(module => ({
    default: module.AdminPanel
  }))
)
```

Mémorisation et Optimisations

```
// useMemo pour calculs coûteux
const expensiveValue = useMemo(() => {
  return products.filter(product =>
    product.category === selectedCategory &&
    product.price >= minPrice &&
    product.price <= maxPrice
  )
}, [products, selectedCategory, minPrice, maxPrice])
```

```tsx
// useCallback pour fonctions
const handleAddToCart = useCallback((productId: string, quantity: number) => {
  addToCart(productId, quantity)
}, [addToCart])

// React.memo pour composants
const ProductCard = React.memo(({ product, onAddToCart }) => {
  return (
    <div>
      {/* Contenu du composant */}
    </div>
  )
}, (prevProps, nextProps) => {
  // Comparaison personnalisée
  return prevProps.product.id === nextProps.product.id
})
```

Images et Assets

```tsx
// Optimisation images avec lazy loading
const OptimizedImage = ({ src, alt, className }) => {
  const [loaded, setLoaded] = useState(false)
  const [inView, setInView] = useState(false)
  const imgRef = useRef(null)

  useEffect(() => {
    const observer = new IntersectionObserver(
      ([entry]) => {
        if (entry.isIntersecting) {
          setInView(true)
          observer.disconnect()
        }
      },
      { threshold: 0.1 }
    )

    if (imgRef.current) {
      observer.observe(imgRef.current)
    }

    return () => observer.disconnect()
  }, [])

  return (
    <div ref={imgRef} className={className}>
      {inView && (
```

```
        <img
          src={src}
          alt={alt}
          loading="lazy"
          onLoad={() => setLoaded(true)}
          className={cn(
            "transition-opacity duration-300",
            loaded ? "opacity-100" : "opacity-0"
          )}
        />
      )}
    </div>
  )
}
```

Utilitaires et Helpers

Fonctions Utilitaires (src/lib/utils.ts)

```
// Formatage prix
export const formatPrice = (price: number): string => {
  return new Intl.NumberFormat('fr-FR', {
    style: 'currency',
    currency: 'EUR'
  }).format(price)
}

// Formatage dates
export const formatDate = (date: Date): string => {
  return new Intl.DateTimeFormat('fr-FR').format(date)
}

// Debounce pour recherche
export const debounce = <T extends (...args: any[]) => any>(
  func: T,
  delay: number
): ((...args: Parameters<T>) => void) => {
  let timeoutId: NodeJS.Timeout

  return (...args: Parameters<T>) => {
    clearTimeout(timeoutId)
    timeoutId = setTimeout(() => func(...args), delay)
  }
}
```

```typescript
// Génération d'IDs uniques
export const generateId = (): string => {
  return Date.now().toString(36) + Math.random().toString(36).substr(2)
}

// Validation email
export const isValidEmail = (email: string): boolean => {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/
  return emailRegex.test(email)
}
```

Helpers E-commerce (`src/lib/ecommerce-utils.ts`)

```typescript
// Calcul de promotion
export const calculateDiscount = (price: number, discount: number): number => {
  return price - (price * discount / 100)
}

// Vérification stock
export const isInStock = (product: Product): boolean => {
  return product.stock > 0
}

// Calcul frais de port
export const calculateShipping = (total: number, freeShippingThreshold: number = 50): numbe
  return total >= freeShippingThreshold ? 0 : 5.99
}

// Formatage statut commande
export const getOrderStatusLabel = (status: OrderStatus): string => {
  const statusLabels = {
    'pending': 'En attente',
    'confirmed': 'Confirmée',
    'shipped': 'Expédiée',
    'delivered': 'Livrée',
    'cancelled': 'Annulée'
  }
  return statusLabels[status] || status
}
```

Gestion des Dépendances

Packages Principaux

```
{
  "dependencies": {
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-router-dom": "^6.26.2",
    "axios": "^1.6.8",
    "react-hook-form": "^7.53.0",
    "@hookform/resolvers": "^3.9.0",
    "zod": "^3.23.8",
    "tailwindcss": "^3.4.0",
    "class-variance-authority": "^0.7.1",
    "clsx": "^2.1.1",
    "tailwind-merge": "^2.5.2"
  }
}
```

Ajout de Nouvelles Dépendances

```
# Installation package
npm install nouvelle-dependance

# Types TypeScript si nécessaire
npm install -D @types/nouvelle-dependance

# Dans le code
import { feature } from 'nouvelle-dependance'
```

Mise à Jour Dépendances

```
# Vérifier versions obsolètes
npm outdated

# Mise à jour sécurisée
npm update

# Mise à jour majeure (attention breaking changes)
npm install package@latest
```

Processus de Développement

Workflow Git Recommandé

```
# Créer branche feature
git checkout -b feature/nouvelle-fonctionnalite

# Commits atomiques
git add .
git commit -m "feat: ajouter composant ProductFilter"

# Push et Pull Request
git push origin feature/nouvelle-fonctionnalite
```

Convention de Nommage

```
// Composants: PascalCase
const ProductCard = () => {}

// Hooks: camelCase avec préfixe 'use'
const useProducts = () => {}

// Fonctions utilitaires: camelCase
const formatPrice = () => {}

// Constantes: SNAKE_CASE
const API_BASE_URL = 'https://api.example.com'

// Fichiers: kebab-case ou PascalCase selon type
// product-card.tsx ou ProductCard.tsx
```

Structure des Commits

```
type(scope): description

Types:
- feat: nouvelle fonctionnalité
- fix: correction de bug
- docs: documentation
- style: formatage
- refactor: refactoring
- test: ajout de tests
- chore: tâches maintenance

Exemples:
feat(products): ajouter filtre par prix
fix(cart): corriger calcul total
```

```
docs(api): documenter endpoint products
```

Erreurs Communes et Solutions

Erreurs React Fréquentes

```
// ❌ Hook dans condition
if (condition) {
  const value = useHook() // Erreur!
}

// ✅ Hook toujours au top level
const value = useHook()
if (condition && value) {
  // Logique
}

// ❌ Clés manquantes dans listes
{items.map(item =>
  <div>{item.name}</div> // Erreur!
)}

// ✅ Clés uniques
{items.map(item =>
  <div key={item.id}>{item.name}</div>
)}

// ❌ Mutation directe du state
setState(prevState => {
  prevState.items.push(newItem) // Erreur!
  return prevState
})

// ✅ Immutabilité
setState(prevState => ({
  ...prevState,
  items: [...prevState.items, newItem]
}))
```

Erreurs TypeScript

```
// ❌ Any partout
const handleData = (data: any) => {} // Éviter

// ✅ Types appropriés
```

```tsx
interface UserData {
  id: string
  name: string
  email: string
}
const handleData = (data: UserData) => {}

//  Propriétés optionnelles non vérifiées
const displayUser = (user: User) => {
  return user.profile.avatar // Erreur si profile undefined
}

//  Vérification des propriétés
const displayUser = (user: User) => {
  return user.profile?.avatar || '/default-avatar.png'
}
```

Erreurs de Performance

```tsx
//  Re-render inutiles
const Component = ({ items }) => {
  const processedItems = items.map(item => ({ // Recalcul à chaque render
    ...item,
    processed: true
  }))

  return <List items={processedItems} />
}

//  Mémorisation
const Component = ({ items }) => {
  const processedItems = useMemo(() =>
    items.map(item => ({ ...item, processed: true }))
  , [items])

  return <List items={processedItems} />
}
```

Checklist de Développement

Avant de Committer

- ☐ Code fonctionne sans erreurs
- ☐ Types TypeScript corrects
- ☐ Pas de console.log en production

- ☐ Tests unitaires passent
- ☐ Responsive design vérifié
- ☐ Accessibilité respectée
- ☐ Performance acceptable

Avant de Déployer

- ☐ Build production réussie
- ☐ Variables d'environnement configurées
- ☐ Base de données migrée si nécessaire
- ☐ Tests e2e passent
- ☐ Documentation mise à jour
- ☐ Monitoring configuré

Code Review

- ☐ Lisibilité du code
- ☐ Respect des conventions
- ☐ Sécurité vérifiée
- ☐ Performance optimisée
- ☐ Tests appropriés
- ☐ Documentation suffisante

Ce guide développeur complet vous permet de comprendre, maintenir et étendre l'application Riziky-Boutic. Consultez régulièrement cette documentation lors du développement de nouvelles fonctionnalités.