Architecture Technique - Riziky-Boutic

Vue d'Ensemble Architecturale

Cette documentation détaille l'architecture technique complète de la plateforme Riziky-Boutic, expliquant chaque composant, sa logique métier, et son interaction avec le système global.

Principes Architecturaux

Architecture en Couches

```
    COUCHE PRÉSENTATION
(React Components + UI)

    COUCHE LOGIQUE MÉTIER
(Hooks + Contexts + Services)

    COUCHE COMMUNICATION
  (API Client + WebSocket)

    COUCHE SERVEUR
(Express Routes + Middlewares)

    COUCHE DONNÉES
  (JSON Files / Database)
```

Patterns Utilisés

- **Component Pattern** : Composants réutilisables et modulaires
- **Hook Pattern** : Logique métier encapsulée dans des hooks personnalisés
- **Context Pattern** : État global partagé via React Context
- **Service Pattern** : Services pour la communication API
- **Middleware Pattern** : Middlewares Express pour la sécurité et validation

---

Architecture Frontend (React)

Structure des Composants

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

`Navbar.tsx` **Objectif :** Navigation principale de l'application **Logique :**

```
// Navigation dynamique selon l'état d'authentification
const { user, isAuthenticated } = useAuth();
const { cartItemCount } = useCart();

// Affichage conditionnel des éléments
{isAuthenticated ? (
  <UserMenu user={user} />
) : (
  <AuthButtons />
)}
```

Utilisation : - Affichage automatique sur toutes les pages - Navigation entre les sections - Indicateur du panier en temps réel - Menu utilisateur contextuel

Modification :

```
// Pour ajouter un nouvel élément de navigation
const navigationItems = [
  { label: "Accueil", href: "/" },
  { label: "Produits", href: "/products" },
  // Ajouter ici
];
```

`Footer.tsx` **Objectif :** Pied de page avec liens et informations légales **Logique :** - Liens statiques vers pages légales - Informations de contact dynamiques - Intégration réseaux sociaux

Modification :

```
// Mise à jour des liens
const footerLinks = {
  company: [
    { label: "À propos", href: "/about" },
    // Ajouter nouveaux liens
  ]
};
```

2. Composants Produits (`src/components/products/`)

`ProductCard.tsx` **Objectif :** Carte produit réutilisable **Logique Complète :**

```
interface ProductCardProps {
  product: Product;
  showQuickView?: boolean;
  showAddToCart?: boolean;
}
```

```tsx
const ProductCard: FC<ProductCardProps> = ({ product, showQuickView = true }) => {
  const { addToCart } = useCart();
  const { toggleFavorite, isFavorite } = useFavorites();

  // Gestion de l'ajout au panier
  const handleAddToCart = async () => {
    try {
      await addToCart(product.id, 1);
      toast.success("Produit ajouté au panier");
    } catch (error) {
      toast.error("Erreur lors de l'ajout");
    }
  };

  // Gestion des favoris
  const handleToggleFavorite = async () => {
    await toggleFavorite(product.id);
  };

  // Calcul du prix avec promotion
  const displayPrice = product.promotion
    ? product.prix - (product.prix * product.promotion.pourcentage / 100)
    : product.prix;
};
```

Utilisation :

```tsx
// Dans une grille de produits
<ProductCard
  product={product}
  showQuickView={true}
  onAddToCart={handleAddToCart}
/>
```

Modification : - Ajouter de nouveaux badges promotionnels - Modifier l'affichage des prix - Personnaliser les actions disponibles

`ProductGrid.tsx` **Objectif :** Grille responsive pour l'affichage des produits
Logique :

```tsx
const ProductGrid: FC<ProductGridProps> = ({ products, isLoading }) => {
  // Responsive grid avec Tailwind
  const gridClasses = cn(
    "grid gap-4",
    "grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4"
  );
```

```
  // Gestion des états de chargement
  if (isLoading) return <ProductGridSkeleton />;
  if (!products.length) return <EmptyProductsMessage />;

  return (
    <div className={gridClasses}>
      {products.map(product => (
        <ProductCard key={product.id} product={product} />
      ))}
    </div>
  );
};
```

3. Composants Panier (`src/components/cart/`)

`CartDrawer.tsx` **Objectif :** Panneau latéral du panier **Logique Complète :**

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

  // Calcul des totaux en temps réel
  const subtotal = cart.reduce((acc, item) =>
    acc + (item.prix * item.quantite), 0
  );

  const shipping = subtotal > 50 ? 0 : 5.99;
  const total = subtotal + shipping;

  // Gestion de la modification des quantités
  const handleQuantityChange = async (itemId: string, newQuantity: number) => {
    if (newQuantity === 0) {
      await removeFromCart(itemId);
    } else {
      await updateQuantity(itemId, newQuantity);
    }
  };
};
```

Modification : - Changer les seuils de frais de port - Ajouter des codes promo - Modifier le calcul des taxes

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

`LoginForm.tsx` **Objectif :** Formulaire de connexion sécurisé **Logique :**

```
const LoginForm: FC = () => {
  const { login } = useAuth();
  const form = useForm<LoginFormData>({
    resolver: zodResolver(loginSchema)
  });

  const onSubmit = async (data: LoginFormData) => {
    try {
      await login(data.email, data.password);
      navigate('/dashboard');
    } catch (error) {
      setError('email', { message: 'Identifiants incorrects' });
    }
  };
};

// Schéma de validation Zod
const loginSchema = z.object({
  email: z.string().email("Email invalide"),
  password: z.string().min(8, "Mot de passe trop court")
});
```

Hooks Personnalisés

1. `useAuth.ts` Objectif : Gestion centralisée de l'authentification **Logique Complète :**

```
interface AuthContextType {
  user: User | null;
  isAuthenticated: boolean;
  login: (email: string, password: string) => Promise<void>;
  logout: () => void;
  register: (userData: RegisterData) => Promise<void>;
}

const useAuth = () => {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  // Vérification du token au chargement
```

```
  useEffect(() => {
    const checkAuthStatus = async () => {
      const token = localStorage.getItem('token');
      if (token) {
        try {
          const userData = await authAPI.verifyToken(token);
          setUser(userData);
        } catch (error) {
          localStorage.removeItem('token');
        }
      }
      setIsLoading(false);
    };

    checkAuthStatus();
  }, []);

  // Fonction de connexion
  const login = async (email: string, password: string) => {
    const response = await authAPI.login({ email, password });
    const { token, user: userData } = response.data;

    localStorage.setItem('token', token);
    setUser(userData);

    // Configuration du header Authorization pour toutes les requêtes
    apiClient.defaults.headers.Authorization = `Bearer ${token}`;
  };

  // Fonction de déconnexion
  const logout = () => {
    localStorage.removeItem('token');
    delete apiClient.defaults.headers.Authorization;
    setUser(null);
  };
};
```

Utilisation :

```
const MyComponent = () => {
  const { user, login, logout, isAuthenticated } = useAuth();

  if (isAuthenticated) {
    return <DashboardContent user={user} />;
  }

  return <LoginForm onLogin={login} />;
```

```
};
```

Modification : - Ajouter la gestion 2FA - Implémenter le refresh token - Ajouter la validation de session

2. `useCart.ts` Objectif : Gestion du panier d'achat **Logique Complète :**

```
const useCart = () => {
  const [cart, setCart] = useState<CartItem[]>([]);
  const [isLoading, setIsLoading] = useState(false);

  // Synchronisation avec le serveur
  useEffect(() => {
    const syncCart = async () => {
      try {
        const serverCart = await cartAPI.getCart();
        setCart(serverCart);
      } catch (error) {
        // Fallback sur le localStorage
        const localCart = JSON.parse(localStorage.getItem('cart') || '[]');
        setCart(localCart);
      }
    };

    syncCart();
  }, []);

  // Ajout au panier avec gestion des conflits
  const addToCart = async (productId: string, quantity: number = 1) => {
    setIsLoading(true);
    try {
      // Vérifier le stock disponible
      const product = await productsAPI.getProduct(productId);
      if (product.stock < quantity) {
        throw new Error('Stock insuffisant');
      }

      // Ajouter au panier serveur
      const updatedCart = await cartAPI.addItem(productId, quantity);
      setCart(updatedCart);

      // Synchroniser localStorage
      localStorage.setItem('cart', JSON.stringify(updatedCart));

      // Notification utilisateur
      toast.success(`${product.nom} ajouté au panier`);
```

```
    } catch (error) {
      toast.error(error.message);
      throw error;
    } finally {
      setIsLoading(false);
    }
  };

  // Calculs dérivés
  const itemCount = cart.reduce((acc, item) => acc + item.quantite, 0);
  const totalPrice = cart.reduce((acc, item) =>
    acc + (item.prix * item.quantite), 0
  );
};
```

3. `useProducts.ts` **Objectif :** Gestion des produits et recherche **Logique :**

```
const useProducts = () => {
  const [products, setProducts] = useState<Product[]>([]);
  const [filters, setFilters] = useState<ProductFilters>({});
  const [pagination, setPagination] = useState({
    page: 1,
    limit: 12,
    total: 0
  });

  // Recherche avec debounce
  const debouncedSearch = useDebounce(filters.search, 300);

  useEffect(() => {
    const fetchProducts = async () => {
      const response = await productsAPI.getProducts({
        ...filters,
        search: debouncedSearch,
        page: pagination.page,
        limit: pagination.limit
      });

      setProducts(response.products);
      setPagination(prev => ({
        ...prev,
        total: response.total
      }));
    };
```

```
    fetchProducts();
  }, [filters, debouncedSearch, pagination.page]);
};
```

Services API

1. `authAPI.ts` Objectif : Communication avec l'API d'authentification
Logique :

```
class AuthAPI {
  private baseURL = '/api/auth';

  async login(credentials: LoginCredentials): Promise<AuthResponse> {
    const response = await apiClient.post(`${this.baseURL}/login`, credentials);
    return response.data;
  }

  async register(userData: RegisterData): Promise<AuthResponse> {
    const response = await apiClient.post(`${this.baseURL}/register`, userData);
    return response.data;
  }

  async verifyToken(token: string): Promise<User> {
    const response = await apiClient.get(`${this.baseURL}/verify`, {
      headers: { Authorization: `Bearer ${token}` }
    });
    return response.data;
  }

  async refreshToken(refreshToken: string): Promise<AuthResponse> {
    const response = await apiClient.post(`${this.baseURL}/refresh`, {
      refreshToken
    });
    return response.data;
  }
}

export const authAPI = new AuthAPI();
```

2. `productsAPI.ts` Objectif : Gestion des produits via API **Logique :**

```
class ProductsAPI {
  private baseURL = '/api/products';

  async getProducts(params: ProductQueryParams = {}): Promise<ProductsResponse> {
    const response = await apiClient.get(this.baseURL, { params });
    return response.data;
```

```
  }

  async getProduct(id: string): Promise<Product> {
    const secureId = secureIds.encodeId(id);
    const response = await apiClient.get(`${this.baseURL}/${secureId}`);
    return response.data;
  }

  async createProduct(productData: CreateProductData): Promise<Product> {
    const formData = new FormData();
    Object.keys(productData).forEach(key => {
      if (key === 'images') {
        productData.images?.forEach(image => {
          formData.append('images', image);
        });
      } else {
        formData.append(key, productData[key]);
      }
    });

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

Architecture Backend (Node.js/Express)

Structure des Routes

1. Routes d'Authentification (`server/routes/auth.js`) Logique Complète :

```
const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const rateLimit = require('express-rate-limit');
const { body, validationResult } = require('express-validator');

const router = express.Router();

// Rate limiting pour les tentatives de connexion
const loginLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
```

```
  max: 5, // 5 tentatives max
  skipSuccessfulRequests: true,
  message: { error: 'Trop de tentatives de connexion' }
});

// Validation des données d'entrée
const loginValidation = [
  body('email').isEmail().normalizeEmail(),
  body('password').isLength({ min: 8 })
];

// Route de connexion
router.post('/login', loginLimiter, loginValidation, async (req, res) => {
  try {
    // Vérifier les erreurs de validation
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { email, password } = req.body;

    // Rechercher l'utilisateur
    const users = await db.getUsers();
    const user = users.find(u => u.email === email);

    if (!user) {
      return res.status(401).json({ error: 'Identifiants incorrects' });
    }

    // Vérifier le mot de passe
    const isValidPassword = await bcrypt.compare(password, user.motDePasse);
    if (!isValidPassword) {
      return res.status(401).json({ error: 'Identifiants incorrects' });
    }

    // Générer le token JWT
    const token = jwt.sign(
      {
        userId: user.id,
        email: user.email,
        role: user.role
      },
      process.env.JWT_SECRET,
      {
        expiresIn: '24h',
```

```
        issuer: 'riziky-boutic',
        audience: 'riziky-users'
      }
    );

    // Supprimer le mot de passe de la réponse
    const { motDePasse, ...userWithoutPassword } = user;

    // Log de sécurité
    console.log(`Connexion réussie: ${email} à ${new Date().toISOString()}`);

    res.json({
      token,
      user: userWithoutPassword,
      expiresIn: '24h'
    });

  } catch (error) {
    console.error('Erreur lors de la connexion:', error);
    res.status(500).json({ error: 'Erreur interne du serveur' });
  }
});

// Route d'inscription
router.post('/register', async (req, res) => {
  try {
    const { nom, prenom, email, motDePasse, phone, genre } = req.body;

    // Vérifier si l'utilisateur existe déjà
    const users = await db.getUsers();
    const existingUser = users.find(u => u.email === email);

    if (existingUser) {
      return res.status(409).json({ error: 'Cet email est déjà utilisé' });
    }

    // Hacher le mot de passe
    const saltRounds = 12;
    const hashedPassword = await bcrypt.hash(motDePasse, saltRounds);

    // Créer le nouvel utilisateur
    const newUser = {
      id: users.length + 1,
      nom,
      prenom,
      email: email.toLowerCase(),
```

```
      motDePasse: hashedPassword,
      phone,
      genre,
      role: 'user',
      createdAt: new Date().toISOString(),
      isActive: true
    };

    // Sauvegarder dans la base
    users.push(newUser);
    await db.saveUsers(users);

    // Générer le token pour connexion automatique
    const token = jwt.sign(
      { userId: newUser.id, email: newUser.email, role: newUser.role },
      process.env.JWT_SECRET,
      { expiresIn: '24h' }
    );

    const { motDePasse: _, ...userWithoutPassword } = newUser;

    res.status(201).json({
      message: 'Compte créé avec succès',
      token,
      user: userWithoutPassword
    });

  } catch (error) {
    console.error('Erreur lors de l\'inscription:', error);
    res.status(500).json({ error: 'Erreur interne du serveur' });
  }
});
```

2. Routes des Produits (`server/routes/products.js`) Logique :

```
// Récupération des produits avec filtrage et pagination
router.get('/', async (req, res) => {
  try {
    const {
      page = 1,
      limit = 12,
      category,
      search,
      minPrice,
      maxPrice,
      sortBy = 'nom',
```

```
  sortOrder = 'asc'
} = req.query;

let products = await db.getProducts();

// Filtrage par catégorie
if (category && category !== 'all') {
  products = products.filter(p => p.category === category);
}

// Recherche textuelle
if (search) {
  const searchLower = search.toLowerCase();
  products = products.filter(p =>
    p.nom.toLowerCase().includes(searchLower) ||
    p.description.toLowerCase().includes(searchLower)
  );
}

// Filtrage par prix
if (minPrice) {
  products = products.filter(p => p.prix >= parseFloat(minPrice));
}
if (maxPrice) {
  products = products.filter(p => p.prix <= parseFloat(maxPrice));
}

// Tri
products.sort((a, b) => {
  let aVal = a[sortBy];
  let bVal = b[sortBy];

  if (typeof aVal === 'string') {
    aVal = aVal.toLowerCase();
    bVal = bVal.toLowerCase();
  }

  if (sortOrder === 'desc') {
    return bVal > aVal ? 1 : -1;
  }
  return aVal > bVal ? 1 : -1;
});

// Pagination
const startIndex = (page - 1) * limit;
const endIndex = startIndex + parseInt(limit);
```

```
    const paginatedProducts = products.slice(startIndex, endIndex);

    // Sécuriser les IDs
    const secureProducts = paginatedProducts.map(product => ({
      ...product,
      id: secureIds.encodeId(product.id)
    }));

    res.json({
      products: secureProducts,
      pagination: {
        currentPage: parseInt(page),
        totalPages: Math.ceil(products.length / limit),
        totalItems: products.length,
        hasNext: endIndex < products.length,
        hasPrev: startIndex > 0
      }
    });

  } catch (error) {
    console.error('Erreur lors de la récupération des produits:', error);
    res.status(500).json({ error: 'Erreur interne du serveur' });
  }
});
```

Middlewares de Sécurité

1. Authentification (`server/middlewares/auth.js`) Logique :

```
const jwt = require('jsonwebtoken');

const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

  if (!token) {
    return res.status(401).json({ error: 'Token d\'accès requis' });
  }

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) {
      if (err.name === 'TokenExpiredError') {
        return res.status(401).json({ error: 'Token expiré' });
      }
      return res.status(403).json({ error: 'Token invalide' });
    }
```

```
    req.user = decoded;
    next();
  });
};

const requireAdmin = (req, res, next) => {
  if (!req.user || req.user.role !== 'admin') {
    return res.status(403).json({ error: 'Accès administrateur requis' });
  }
  next();
};

module.exports = { authenticateToken, requireAdmin };
```

2. Sécurité Générale (server/middlewares/security.js) Logique :

```
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const xss = require('xss-clean');

// Configuration Helmet pour la sécurité des headers
const helmetConfig = helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'", "https://fonts.googleapis.com"],
      fontSrc: ["'self'", "https://fonts.gstatic.com"],
      imgSrc: ["'self'", "data:", "https:"],
      scriptSrc: ["'self'"],
    },
  },
  crossOriginEmbedderPolicy: false
});

// Rate limiting global
const generalLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // 100 requêtes par IP
  message: { error: 'Trop de requêtes, veuillez réessayer plus tard' }
});

// Rate limiting pour l'API
const apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 50,
```

```
  message: { error: 'Limite API atteinte' }
});

module.exports = {
  helmetConfig,
  generalLimiter,
  apiLimiter,
  xssProtection: xss()
};
```

Services Métier

1. Service des Produits (server/services/products.service.js)
Logique :

```
const fs = require('fs').promises;
const path = require('path');
const sharp = require('sharp'); // Pour l'optimisation d'images

class ProductsService {
  constructor() {
    this.productsFile = path.join(__dirname, '../data/products.json');
    this.uploadsDir = path.join(__dirname, '../uploads');
  }

  async getAllProducts(filters = {}) {
    try {
      const data = await fs.readFile(this.productsFile, 'utf8');
      let products = JSON.parse(data);

      // Appliquer les filtres
      if (filters.category) {
        products = products.filter(p => p.category === filters.category);
      }

      if (filters.inStock) {
        products = products.filter(p => p.stock > 0);
      }

      return products;
    } catch (error) {
      throw new Error('Erreur lors de la récupération des produits');
    }
  }

  async getProductById(id) {
```

```
    const products = await this.getAllProducts();
    const product = products.find(p => p.id === parseInt(id));

    if (!product) {
      throw new Error('Produit non trouvé');
    }

    return product;
  }

  async createProduct(productData, imageFiles = []) {
    try {
      const products = await this.getAllProducts();

      // Traitement des images
      const imageUrls = await this.processImages(imageFiles);

      const newProduct = {
        id: Math.max(...products.map(p => p.id)) + 1,
        ...productData,
        images: imageUrls,
        createdAt: new Date().toISOString(),
        updatedAt: new Date().toISOString()
      };

      products.push(newProduct);
      await this.saveProducts(products);

      return newProduct;
    } catch (error) {
      throw new Error('Erreur lors de la création du produit');
    }
  }

  async processImages(imageFiles) {
    const imageUrls = [];

    for (const file of imageFiles) {
      // Génération d'un nom unique
      const filename = `product-${Date.now()}-${Math.random().toString(36).substr(2, 9)}.web
      const filepath = path.join(this.uploadsDir, filename);

      // Optimisation avec Sharp
      await sharp(file.buffer)
        .resize(800, 600, { fit: 'contain', background: { r: 255, g: 255, b: 255 } })
        .webp({ quality: 85 })
```

```
        .toFile(filepath);

      imageUrls.push(`/uploads/${filename}`);
    }

    return imageUrls;
  }

  async updateProduct(id, updateData) {
    const products = await this.getAllProducts();
    const productIndex = products.findIndex(p => p.id === parseInt(id));

    if (productIndex === -1) {
      throw new Error('Produit non trouvé');
    }

    products[productIndex] = {
      ...products[productIndex],
      ...updateData,
      updatedAt: new Date().toISOString()
    };

    await this.saveProducts(products);
    return products[productIndex];
  }

  async deleteProduct(id) {
    const products = await this.getAllProducts();
    const productIndex = products.findIndex(p => p.id === parseInt(id));

    if (productIndex === -1) {
      throw new Error('Produit non trouvé');
    }

    // Supprimer les images associées
    const product = products[productIndex];
    if (product.images) {
      for (const imageUrl of product.images) {
        const imagePath = path.join(__dirname, '..', imageUrl);
        try {
          await fs.unlink(imagePath);
        } catch (error) {
          console.warn('Impossible de supprimer l\'image:', imagePath);
        }
      }
    }
```

```
    products.splice(productIndex, 1);
    await this.saveProducts(products);

    return { message: 'Produit supprimé avec succès' };
  }

  async saveProducts(products) {
    await fs.writeFile(this.productsFile, JSON.stringify(products, null, 2));
  }
}

module.exports = new ProductsService();
```

Sécurisation des IDs

1. Service de Sécurisation (`src/services/secureIds.ts`) Objectif : Masquer les vrais IDs des ressources **Logique :**

```
class SecureIdService {
  private mapping = new Map<string, string>();
  private reverseMapping = new Map<string, string>();

  encodeId(realId: string | number): string {
    const id = String(realId);

    if (this.mapping.has(id)) {
      return this.mapping.get(id)!;
    }

    // Générer un ID sécurisé unique
    const secureId = this.generateSecureId();

    this.mapping.set(id, secureId);
    this.reverseMapping.set(secureId, id);

    return secureId;
  }

  decodeId(secureId: string): string {
    const realId = this.reverseMapping.get(secureId);

    if (!realId) {
      throw new Error('ID sécurisé invalide');
    }
```

```
    return realId;
  }

  private generateSecureId(): string {
    // Combinaison de timestamp et random pour unicité
    const timestamp = Date.now().toString(36);
    const random = Math.random().toString(36).substr(2, 15);

    return `${timestamp}_${random}`;
  }

  // Nettoyage périodique des mappings anciens
  cleanup(): void {
    const maxAge = 24 * 60 * 60 * 1000; // 24 heures
    const now = Date.now();

    for (const [secureId, _] of this.reverseMapping) {
      const [timestampPart] = secureId.split('_');
      const timestamp = parseInt(timestampPart, 36);

      if (now - timestamp > maxAge) {
        const realId = this.reverseMapping.get(secureId)!;
        this.mapping.delete(realId);
        this.reverseMapping.delete(secureId);
      }
    }
  }
}

export const secureIds = new SecureIdService();

// Nettoyage automatique toutes les heures
setInterval(() => secureIds.cleanup(), 60 * 60 * 1000);
```

Utilisation :

```
// Lors de l'affichage des produits
const secureProductId = secureIds.encodeId(product.id);
const productUrl = `/product/${secureProductId}`;

// Lors de la récupération d'un produit
const realProductId = secureIds.decodeId(params.id);
const product = await getProduct(realProductId);
```

Communication Temps Réel (WebSocket)

Configuration Socket.io

Server (server/socket/socketHandlers.js)

```
const socketAuth = require('./socketAuth');

const setupSocketHandlers = (io) => {
  // Middleware d'authentification
  io.use(socketAuth.authenticate);

  io.on('connection', (socket) => {
    console.log(`Utilisateur connecté: ${socket.user.email}`);

    // Rejoindre la room utilisateur
    socket.join(`user_${socket.user.id}`);

    // Chat service client
    socket.on('client:join_support', async (data) => {
      const { orderId } = data;
      const supportRoom = `support_${orderId}`;

      socket.join(supportRoom);

      // Notifier les admins
      socket.to('admin_room').emit('client:support_request', {
        userId: socket.user.id,
        orderId,
        timestamp: new Date().toISOString()
      });
    });

    // Messages chat
    socket.on('message:send', async (data) => {
      const { content, roomId, type } = data;

      const message = {
        id: `msg_${Date.now()}`,
        content,
        senderId: socket.user.id,
        senderName: `${socket.user.prenom} ${socket.user.nom}`,
        timestamp: new Date().toISOString(),
        type
      };

      // Sauvegarder le message
```

```
      await chatService.saveMessage(roomId, message);

      // Diffuser aux participants
      socket.to(roomId).emit('message:received', message);
    });

    // Notifications en temps réel
    socket.on('order:status_update', async (data) => {
      const { orderId, status } = data;

      // Mettre à jour la commande
      await ordersService.updateOrderStatus(orderId, status);

      // Notifier le client
      const order = await ordersService.getOrder(orderId);
      socket.to(`user_${order.userId}`).emit('order:updated', {
        orderId,
        status,
        timestamp: new Date().toISOString()
      });
    });

    // Déconnexion
    socket.on('disconnect', () => {
      console.log(`Utilisateur déconnecté: ${socket.user.email}`);
    });
  });
};

module.exports = setupSocketHandlers;
```

Client (src/services/socket.ts)

```
import io from 'socket.io-client';

class SocketService {
  private socket: any = null;
  private listeners: Map<string, Function[]> = new Map();

  connect(token: string): void {
    this.socket = io(process.env.VITE_API_BASE_URL || 'http://localhost:10000', {
      auth: { token },
      transports: ['websocket', 'polling']
    });

    this.socket.on('connect', () => {
```

```
    console.log('Connected to server via WebSocket');
  });

  this.socket.on('disconnect', (reason: string) => {
    console.log('Disconnected:', reason);

    if (reason === 'io server disconnect') {
      // Reconnexion manuelle si le serveur a fermé la connexion
      this.socket.connect();
    }
  });

  // Réattacher tous les listeners
  for (const [event, callbacks] of this.listeners) {
    callbacks.forEach(callback => {
      this.socket.on(event, callback);
    });
  }
}

disconnect(): void {
  if (this.socket) {
    this.socket.disconnect();
    this.socket = null;
  }
}

emit(event: string, data: any): void {
  if (this.socket) {
    this.socket.emit(event, data);
  }
}

on(event: string, callback: Function): void {
  // Stocker le listener pour la reconnexion
  if (!this.listeners.has(event)) {
    this.listeners.set(event, []);
  }
  this.listeners.get(event)!.push(callback);

  if (this.socket) {
    this.socket.on(event, callback);
  }
}

off(event: string, callback?: Function): void {
```

```
    if (callback) {
      const callbacks = this.listeners.get(event) || [];
      const index = callbacks.indexOf(callback);
      if (index > -1) {
        callbacks.splice(index, 1);
      }
    } else {
      this.listeners.delete(event);
    }

    if (this.socket) {
      this.socket.off(event, callback);
    }
  }
}

export const socketService = new SocketService();
```

Gestion des Données

Base de Données JSON Actuelle

Structure des Fichiers

```
server/data/
  users.json          # Utilisateurs et authentification
  products.json       # Catalogue des produits
  orders.json         # Commandes clients
  categories.json     # Catégories de produits
  cart.json          # Paniers utilisateurs
  favorites.json     # Listes de favoris
  reviews.json       # Avis et commentaires
  flash-sales.json   # Ventes flash et promotions
  site-settings.json # Configuration du site
```

Service de Base de Données (`server/core/database.js`)

```
const fs = require('fs').promises;
const path = require('path');

class DatabaseService {
  constructor() {
    this.dataDir = path.join(__dirname, '../data');
    this.cache = new Map();
    this.cacheTTL = new Map();
  }
```

```
// Lecture avec cache
async readData(filename, useCache = true) {
  const cacheKey = filename;

  if (useCache && this.cache.has(cacheKey)) {
    const ttl = this.cacheTTL.get(cacheKey);
    if (Date.now() < ttl) {
      return this.cache.get(cacheKey);
    }
  }

  try {
    const filePath = path.join(this.dataDir, filename);
    const data = await fs.readFile(filePath, 'utf8');
    const parsed = JSON.parse(data);

    // Mise en cache (5 minutes)
    if (useCache) {
      this.cache.set(cacheKey, parsed);
      this.cacheTTL.set(cacheKey, Date.now() + 5 * 60 * 1000);
    }

    return parsed;
  } catch (error) {
    console.error(`Erreur lecture ${filename}:`, error);
    return [];
  }
}

// Écriture avec sauvegarde
async writeData(filename, data) {
  try {
    const filePath = path.join(this.dataDir, filename);
    const backupPath = `${filePath}.backup`;

    // Créer une sauvegarde
    try {
      await fs.copyFile(filePath, backupPath);
    } catch (error) {
      // Fichier n'existe pas encore, c'est normal
    }

    // Écrire les nouvelles données
    await fs.writeFile(filePath, JSON.stringify(data, null, 2));
```

```
      // Invalider le cache
      this.cache.delete(filename);
      this.cacheTTL.delete(filename);

      return true;
    } catch (error) {
      console.error(`Erreur écriture ${filename}:`, error);
      throw error;
    }
  }

  // Méthodes spécialisées
  async getUsers() { return this.readData('users.json'); }
  async saveUsers(users) { return this.writeData('users.json', users); }

  async getProducts() { return this.readData('products.json'); }
  async saveProducts(products) { return this.writeData('products.json', products); }

  async getOrders() { return this.readData('orders.json'); }
  async saveOrders(orders) { return this.writeData('orders.json', orders); }
}

module.exports = new DatabaseService();
```

Migration vers PostgreSQL (Préparation)

Schéma de Base de Données

```
-- Utilisateurs
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  nom VARCHAR(100) NOT NULL,
  prenom VARCHAR(100) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  mot_de_passe VARCHAR(255) NOT NULL,
  phone VARCHAR(20),
  genre VARCHAR(10),
  role VARCHAR(20) DEFAULT 'user',
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Catégories
CREATE TABLE categories (
  id SERIAL PRIMARY KEY,
```

```sql
  nom VARCHAR(100) NOT NULL,
  description TEXT,
  parent_id INTEGER REFERENCES categories(id),
  image_url VARCHAR(255),
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Produits
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  nom VARCHAR(255) NOT NULL,
  description TEXT,
  prix DECIMAL(10,2) NOT NULL,
  stock INTEGER DEFAULT 0,
  category_id INTEGER REFERENCES categories(id),
  images JSONB,
  specifications JSONB,
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Index pour les performances
CREATE INDEX idx_products_category ON products(category_id);
CREATE INDEX idx_products_active ON products(is_active);
CREATE INDEX idx_products_prix ON products(prix);
```

Service de Migration

```javascript
const { Pool } = require('pg');

class MigrationService {
  constructor() {
    this.pool = new Pool({
      connectionString: process.env.DATABASE_URL
    });
  }

  async migrateFromJSON() {
    const client = await this.pool.connect();

    try {
      await client.query('BEGIN');

      // Migrer les utilisateurs
```

```
      const users = await db.getUsers();
      for (const user of users) {
        await client.query(`
          INSERT INTO users (nom, prenom, email, mot_de_passe, phone, genre, role, is_active
          VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
        `, [
          user.nom, user.prenom, user.email, user.motDePasse,
          user.phone, user.genre, user.role || 'user', true
        ]);
      }

      // Migrer les catégories
      const categories = await db.readData('categories.json');
      for (const category of categories) {
        await client.query(`
          INSERT INTO categories (id, nom, description, image_url)
          VALUES ($1, $2, $3, $4)
        `, [category.id, category.nom, category.description, category.image]);
      }

      // Migrer les produits
      const products = await db.getProducts();
      for (const product of products) {
        await client.query(`
          INSERT INTO products (nom, description, prix, stock, category_id, images)
          VALUES ($1, $2, $3, $4, $5, $6)
        `, [
          product.nom, product.description, product.prix, product.stock,
          product.category, JSON.stringify(product.images)
        ]);
      }

      await client.query('COMMIT');
      console.log('Migration terminée avec succès');

    } catch (error) {
      await client.query('ROLLBACK');
      console.error('Erreur lors de la migration:', error);
      throw error;
    } finally {
      client.release();
    }
  }
}
```

Cette documentation technique complète détaille tous les aspects architecturaux de Riziky-Boutic. Chaque composant, service et logique métier est expliqué avec des exemples d'utilisation et de modification. Cette base permet une maintenance efficace et une évolution maîtrisée de la plateforme.