Documentation du Code Finale - Riziky-Boutic

Guide Complet de la Base de Code

Cette documentation finale présente une analyse exhaustive de l'architecture du code, des patterns utilisés, et des bonnes pratiques implémentées dans la plateforme Riziky-Boutic.

Architecture Globale du Code

Structure Hiérarchique Complète

```
riziky-boutic/
    src/                          # Frontend React/TypeScript
        components/               # Composants UI réutilisables
            layout/              # → Composants de mise en page
              Navbar.tsx          # → Navigation principale avec authentification
              Footer.tsx          # → Pied de page avec liens et informations
              Sidebar.tsx         # → Barre latérale pour administration
              CategoriesDropdown.tsx # → Menu déroulant des catégories
            products/            # → Composants liés aux produits
              ProductCard.tsx     # → Carte produit avec actions
              ProductGrid.tsx     # → Grille responsive de produits
              ProductDetail.tsx   # → Vue détaillée d'un produit
              ProductFilter.tsx   # → Filtres de recherche et tri
            cart/                # → Gestion du panier d'achat
              CartDrawer.tsx      # → Panier latéral coulissant
              CartItem.tsx        # → Item individuel dans le panier
              CartSummary.tsx     # → Résumé et totaux du panier
            auth/                # → Authentification utilisateur
              LoginForm.tsx       # → Formulaire de connexion
              RegisterForm.tsx    # → Formulaire d'inscription
              ProtectedRoute.tsx  # → Protection des routes privées
            admin/               # → Interface d'administration
              Dashboard.tsx       # → Tableau de bord administrateur
              ProductManager.tsx  # → Gestion des produits
              OrderManager.tsx    # → Gestion des commandes
              UserManager.tsx     # → Gestion des utilisateurs
            promotions/          # → Promotions et ventes flash
              FlashSaleBanner.tsx # → Bannière de vente flash
              PromoCode.tsx       # → Codes promotionnels
              CountdownTimer.tsx  # → Compte à rebours promotions
            chat/                # → Service client et chat
              ChatWindow.tsx      # → Fenêtre de chat principal
              MessageBubble.tsx   # → Bulle de message individuel
```

```
      ChatInput.tsx       # → Zone de saisie des messages
    ui/                   # → Composants UI de base (Shadcn)
      button.tsx          # → Composant bouton personnalisé
      input.tsx           # → Champs de saisie avec validation
      dialog.tsx          # → Modales et dialogues
      toast.tsx           # → Notifications utilisateur
hooks/                    # → Hooks personnalisés métier
  useAuth.ts              # → Hook d'authentification
  useProducts.ts          # → Hook de gestion des produits
  useCart.ts              # → Hook de gestion du panier
  useFavorites.ts         # → Hook de gestion des favoris
  useOrders.ts            # → Hook de gestion des commandes
  useSocket.ts            # → Hook WebSocket temps réel
  useLocalStorage.ts      # → Hook stockage local
contexts/                 # → Contextes React état global
  AuthContext.tsx         # → Contexte d'authentification
  StoreContext.tsx        # → Contexte global du magasin
  ThemeContext.tsx        # → Contexte de thème (dark/light)
  VideoCallContext.tsx    # → Contexte appels vidéo
services/                 # → Services et communication API
    core/                 # → Configuration centrale
      apiClient.ts        # → Client HTTP Axios configuré
      errorHandler.ts     # → Gestion centralisée des erreurs
      interceptors.ts     # → Intercepteurs HTTP
    modules/              # → Services par domaine métier
      auth.service.ts     # → Service authentification
      products.service.ts # → Service gestion produits
      cart.service.ts     # → Service gestion panier
      orders.service.ts   # → Service gestion commandes
      users.service.ts    # → Service gestion utilisateurs
      analytics.service.ts # → Service analytics
  secureIds.ts            # → Service de sécurisation des IDs
  secureCategories.ts     # → Sécurisation des catégories
  socket.ts               # → Configuration WebSocket
types/                    # → Définitions TypeScript
  auth.types.ts           # → Types authent
  product.types.ts        # → Types produits et catalogue
  order.types.ts          # → Types commandes et paiements
  user.types.ts           # → Types utilisateurs et profils
  api.types.ts            # → Types réponses API
utils/                    # → Fonctions utilitaires
  formatters.ts           # → Formatage données (prix, dates)
  validators.ts           # → Validation formulaires et données
  constants.ts            # → Constantes application
  helpers.ts              # → Fonctions d'aide génériques
  storage.ts              # → Utilitaires stockage local
```

```
    pages/                    # → Pages principales application
      Home.tsx                 # → Page d'accueil e-commerce
      ProductDetail.tsx        # → Page détail produit
      CategoryPage.tsx         # → Page catégorie de produits
      SearchResults.tsx        # → Page résultats de recherche
      Cart.tsx                 # → Page panier complet
      Checkout.tsx             # → Processus de commande
      Profile.tsx              # → Profil utilisateur
      Orders.tsx               # → Historique des commandes
        admin/                 # → Pages d'administration
          AdminDashboard.tsx   # → Dashboard administrateur
          ProductManagement.tsx # → Gestion produits admin
          OrderManagement.tsx # → Gestion commandes admin
server/                       # Backend Node.js/Express
    routes/                   # → Routes API par domaine
      auth.routes.js           # → Routes authentification (/api/auth/*)
      products.routes.js       # → Routes produits (/api/products/*)
      cart.routes.js           # → Routes panier (/api/cart/*)
      orders.routes.js         # → Routes commandes (/api/orders/*)
      users.routes.js          # → Routes utilisateurs (/api/users/*)
      admin.routes.js          # → Routes administration (/api/admin/*)
      categories.routes.js     # → Routes catégories (/api/categories/*)
      analytics.routes.js      # → Routes analytics (/api/analytics/*)
    services/                 # → Logique métier backend
      auth.service.js          # → Service authentification JWT
      products.service.js      # → Service gestion produits
      cart.service.js          # → Service gestion panier
      orders.service.js        # → Service gestion commandes
      users.service.js         # → Service gestion utilisateurs
      email.service.js         # → Service envoi emails
      payment.service.js       # → Service traitement paiements
      analytics.service.js     # → Service analytics et métriques
    middlewares/              # → Middlewares Express
      auth.middleware.js       # → Vérification tokens JWT
      security.js              # → Sécurité avancée et rate limiting
      validation.js            # → Validation et sanitisation données
      upload.middleware.js     # → Upload fichiers sécurisé
      cors.middleware.js       # → Configuration CORS
      logging.middleware.js    # → Logging des requêtes
    core/                     # → Modules centraux backend
      database.js              # → Gestion base données JSON
      logger.js                # → Système logging structuré
      errorHandler.js          # → Gestion centralisée erreurs
      cache.js                 # → Système de cache mémoire
      scheduler.js             # → Tâches programmées
    data/                     # → Fichiers données JSON
```

```
    users.json              # → Base utilisateurs et authentification
    products.json           # → Catalogue produits complet
    categories.json         # → Catégories et hiérarchie
    orders.json             # → Commandes et historique
    cart.json               # → Paniers utilisateurs actifs
    favorites.json          # → Listes de favoris utilisateurs
    reviews.json            # → Avis et commentaires produits
    flash-sales.json        # → Ventes flash et promotions
    contacts.json           # → Messages de contact
    client-chat.json        # → Historique chat service client
    analytics.json          # → Données analytics et statistiques
  socket/                   # → Configuration Socket.io temps réel
    socketHandler.js        # → Gestionnaire principal WebSocket
    chatHandler.js          # → Gestion chat temps réel
    notificationHandler.js  # → Notifications push temps réel
    adminHandler.js         # → Fonctionnalités admin temps réel
  uploads/                  # → Fichiers uploadés
      products/             # → Images et médias produits
      users/                # → Photos profil utilisateurs
      temp/                 # → Fichiers temporaires upload
      backup/               # → Sauvegardes automatiques
  config/                   # → Configuration serveur
    database.config.js      # → Configuration base de données
    jwt.config.js           # → Configuration JWT et sécurité
    cors.config.js          # → Configuration CORS détaillée
    email.config.js         # → Configuration service email
    environment.config.js   # → Variables environnement
server.js                   # → Point d'entrée serveur principal
docs/                       # → Documentation technique
  ARCHITECTURE_TECHNIQUE_FINALE.md
  CAHIER_DES_CHARGES_FINALE.md
  CODE_DOCUMENTATION_FINALE.md
  COMMENTAIRES_TECHNIQUES_FINALE.md
  PROJET_RESUME_FINALE.md
  RESUME_FONCTIONNALITES_FINALE.md
```

Patterns d'Architecture Détaillés

1. Custom Hooks Pattern (Frontend)

Hook d'Authentification Complet

```
/**
 * Hook personnalisé pour la gestion complète de l'authentification
 *
```

```
 * Fonctionnalités:
 * - Gestion de l'état utilisateur connecté
 * - Validation automatique des tokens JWT
 * - Refresh automatique des tokens expirés
 * - Déconnexion automatique en cas d'erreur
 * - Persistance de session cross-tab
 *
 * Usage: const { user, login, logout, isLoading } = useAuth();
 */
export const useAuth = () => {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  //  Validation automatique du token au démarrage de l'application
  const validateToken = useCallback(async () => {
    const token = localStorage.getItem('authToken');
    if (!token) {
      setIsLoading(false);
      return false;
    }

    try {
      // Appel API pour vérifier la validité du token
      const response = await authAPI.verifyToken();
      if (response.data && response.data.valid) {
        setUser(response.data.user);
        return true;
      }
    } catch (error) {
      //  Nettoyage automatique en cas de token invalide
      console.error("Token invalide:", error);
      localStorage.removeItem('authToken');
      setError('Session expirée');
    }

    setIsLoading(false);
    return false;
  }, []);

  //  Processus de connexion avec gestion d'erreurs complète
  const login = useCallback(async (email: string, password: string): Promise<void> => {
    setIsLoading(true);
    setError(null);

    try {
```

```
      console.log("  Tentative de connexion pour:", email);
      const response = await authAPI.login({ email, password });

      //  Stockage sécurisé du token d'authentification
      localStorage.setItem('authToken', response.data.token);
      setUser(response.data.user);

      //  Notification de succès à l'utilisateur
      toast({
        title: 'Connexion réussie',
        description: `Bienvenue ${response.data.user.name}`,
        variant: 'default',
      });

      //  Redirection vers la page d'origine ou accueil
      const redirectUrl = sessionStorage.getItem('redirectAfterLogin') || '/';
      sessionStorage.removeItem('redirectAfterLogin');
      window.location.href = redirectUrl;

    } catch (error: any) {
      console.error("  Erreur de connexion:", error);

      //  Gestion des différents types d'erreurs
      const errorMessage = error.response?.data?.message || "Erreur de connexion";
      setError(errorMessage);
      toast({
        title: 'Erreur de connexion',
        description: errorMessage,
        variant: 'destructive',
      });

      throw error;
    } finally {
      setIsLoading(false);
    }
  }, []);

  //  Déconnexion sécurisée avec nettoyage complet
  const logout = useCallback(async () => {
    try {
      //  Notification au serveur de la déconnexion
      await authAPI.logout();
    } catch (error) {
      console.error("Erreur lors de la déconnexion:", error);
    } finally {
      //  Nettoyage complet des données utilisateur
```

```
    localStorage.removeItem('authToken');
    sessionStorage.clear();
    setUser(null);
    setError(null);

    //  Redirection vers page de connexion
    window.location.href = '/login';
  }
}, []);

//  Processus d'inscription avec validation
const register = useCallback(async (userData: RegisterData): Promise<void> => {
  setIsLoading(true);
  setError(null);

  try {
    console.log(" Tentative d'inscription pour:", userData.email);
    const response = await authAPI.register(userData);

    //  Inscription réussie - connexion automatique
    localStorage.setItem('authToken', response.data.token);
    setUser(response.data.user);

    toast({
      title: 'Inscription réussie',
      description: 'Votre compte a été créé avec succès',
      variant: 'default',
    });

    window.location.href = '/';

  } catch (error: any) {
    console.error(" Erreur d'inscription:", error);
    const errorMessage = error.response?.data?.message || "Erreur d'inscription";
    setError(errorMessage);
    toast({
      title: 'Erreur d\'inscription',
      description: errorMessage,
      variant: 'destructive',
    });
    throw error;
  } finally {
    setIsLoading(false);
  }
}, []);
```

```
  //  Initialisation automatique au montage du hook
  useEffect(() => {
    validateToken();
  }, [validateToken]);

  //  Écoute des changements de stockage (synchronisation multi-onglets)
  useEffect(() => {
    const handleStorageChange = (e: StorageEvent) => {
      if (e.key === 'authToken') {
        if (e.newValue === null) {
          // Token supprimé dans un autre onglet - déconnexion
          setUser(null);
        } else if (e.newValue !== e.oldValue) {
          // Token modifié dans un autre onglet - re-validation
          validateToken();
        }
      }
    };

    window.addEventListener('storage', handleStorageChange);
    return () => window.removeEventListener('storage', handleStorageChange);
  }, [validateToken]);

  return {
    user,                              //  Utilisateur connecté ou null
    isAuthenticated: !!user,           //  Statut d'authentification
    isAdmin: user?.role === 'admin', //  Permissions administrateur
    isLoading,                         //  État de chargement
    error,                             //  Erreur éventuelle
    login,                             //  Fonction de connexion
    logout,                            //  Fonction de déconnexion
    register,                          //  Fonction d'inscription
  };
};
```

Hook de Gestion des Produits Avancé

```
/**
 * Hook personnalisé pour la gestion complète des produits
 *
 * Fonctionnalités:
 * - Chargement des produits avec cache intelligent
 * - Filtrage et recherche en temps réel
 * - Gestion des promotions avec expiration automatique
 * - Optimisations performance avec debouncing
 * - Synchronisation avec le backend
```

```
 *
 * Usage: const { products, loading, searchProducts, filters } = useProducts();
 */
export const useProducts = (initialCategory?: string) => {
  const [products, setProducts] = useState<Product[]>([]);
  const [filteredProducts, setFilteredProducts] = useState<Product[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [filters, setFilters] = useState<ProductFilters>({
    category: initialCategory || '',
    priceRange: { min: 0, max: Infinity },
    searchTerm: '',
    sortBy: 'name',
    sortOrder: 'asc',
    inStock: true
  });

  //  Cache des produits pour éviter les appels API répétitifs
  const productCache = useRef<Map<string, { data: Product[], timestamp: number }>>(new Map()
  const CACHE_DURATION = 5 * 60 * 1000; // 5 minutes

  //  Récupération des produits avec mise en cache intelligente
  const fetchProducts = useCallback(async (categoryName?: string, forceRefresh = false) => {
    const cacheKey = categoryName || 'all';
    const cached = productCache.current.get(cacheKey);

    //  Utilisation du cache si disponible et valide
    if (!forceRefresh && cached && Date.now() - cached.timestamp < CACHE_DURATION) {
      console.log(" Utilisation du cache pour les produits:", cacheKey);
      setProducts(cached.data);
      setLoading(false);
      return cached.data;
    }

    setLoading(true);
    setError(null);

    try {
      console.log(" Chargement des produits depuis l'API:", cacheKey);
      let response;

      //  Appel API conditionnel selon la catégorie
      if (categoryName) {
        response = await productsAPI.getByCategory(categoryName);
      } else {
        response = await productsAPI.getAll();
```

```
    }

    //  Validation de la structure des données reçues
    if (!response.data || !Array.isArray(response.data)) {
      throw new Error('Format de données incorrect pour les produits');
    }

    //  Traitement des promotions et prix
    const processedProducts = response.data.map(product => ({
      ...product,
      displayPrice: calculateDisplayPrice(product),
      isOnSale: checkIfOnSale(product),
      stockStatus: getStockStatus(product.stock)
    }));

    //  Mise en cache des données
    productCache.current.set(cacheKey, {
      data: processedProducts,
      timestamp: Date.now()
    });

    setProducts(processedProducts);
    console.log(`  ${processedProducts.length} produits chargés avec succès`);

    return processedProducts;

  } catch (error) {
    console.error("  Erreur lors du chargement des produits:", error);
    setError('Erreur lors du chargement des produits');
    toast.error('Impossible de charger les produits');
    setProducts([]);
    return [];
  } finally {
    setLoading(false);
  }
}, []);

//  Recherche de produits avec debouncing pour optimiser les performances
const debouncedSearch = useCallback(
  debounce((searchTerm: string, currentProducts: Product[]) => {
    if (!searchTerm.trim()) {
      setFilteredProducts(currentProducts);
      return;
    }

    //  Recherche multi-critères (nom, description, catégorie)
```

```
    const filtered = currentProducts.filter(product => {
      const searchLower = searchTerm.toLowerCase();
      return (
        product.name.toLowerCase().includes(searchLower) ||
        product.description?.toLowerCase().includes(searchLower) ||
        product.category?.toLowerCase().includes(searchLower)
      );
    });

    setFilteredProducts(filtered);
    console.log(`  Recherche "${searchTerm}": ${filtered.length} résultats`);
  }, 300), []
);

//  Application des filtres avancés
const applyFilters = useCallback((currentProducts: Product[], currentFilters: ProductFilte
  let filtered = [...currentProducts];

  //  Filtre par catégorie
  if (currentFilters.category) {
    filtered = filtered.filter(product =>
      product.category?.toLowerCase() === currentFilters.category.toLowerCase()
    );
  }

  //  Filtre par gamme de prix
  if (currentFilters.priceRange) {
    filtered = filtered.filter(product => {
      const price = product.displayPrice || product.price;
      return price >= currentFilters.priceRange.min && price <= currentFilters.priceRange.
    });
  }

  //  Filtre par disponibilité en stock
  if (currentFilters.inStock) {
    filtered = filtered.filter(product => product.stock > 0);
  }

  //  Tri des résultats
  filtered.sort((a, b) => {
    const aValue = a[currentFilters.sortBy as keyof Product];
    const bValue = b[currentFilters.sortBy as keyof Product];

    if (currentFilters.sortOrder === 'asc') {
      return aValue > bValue ? 1 : -1;
    } else {
```

```
      return aValue < bValue ? 1 : -1;
    }
  });

  setFilteredProducts(filtered);
  console.log(` Filtres appliqués: ${filtered.length} produits`);
}, []);

// Vérification automatique des promotions expirées
useEffect(() => {
  const checkExpiredPromotions = () => {
    const now = new Date();
    let hasExpired = false;

    const updatedProducts = products.map(product => {
      if (product.promotion && product.promotionEnd && new Date(product.promotionEnd) < no
        hasExpired = true;
        return {
          ...product,
          price: product.originalPrice || product.price,
          promotion: null,
          promotionEnd: null,
          displayPrice: product.originalPrice || product.price,
          isOnSale: false
        };
      }
      return product;
    });

    if (hasExpired) {
      console.log(" Promotions expirées détectées - mise à jour");
      setProducts(updatedProducts);
    }
  };

  // Vérification toutes les minutes
  const interval = setInterval(checkExpiredPromotions, 60000);
  return () => clearInterval(interval);
}, [products]);

// Application de la recherche quand le terme change
useEffect(() => {
  debouncedSearch(filters.searchTerm, products);
}, [filters.searchTerm, products, debouncedSearch]);

// Application des filtres quand ils changent
```

```
  useEffect(() => {
    if (!filters.searchTerm) {
      applyFilters(products, filters);
    }
  }, [products, filters, applyFilters]);

  //  Chargement initial des produits
  useEffect(() => {
    fetchProducts(initialCategory);
  }, [fetchProducts, initialCategory]);

  return {
    products,                          //  Tous les produits
    filteredProducts,                  //  Produits filtrés/recherchés
    loading,                           //  État de chargement
    error,                             //  Erreur éventuelle
    filters,                           //  Filtres actuels
    setFilters,                        //  Modifier les filtres
    fetchProducts,                     //  Recharger les produits
    searchProducts: (term: string) => {  //  Fonction de recherche
      setFilters(prev => ({ ...prev, searchTerm: term }));
    },
    getProductById: (id: string) => {    //  Récupérer un produit par ID
      return products.find(p => p.id === id);
    },
    refreshProducts: () => fetchProducts(undefined, true) //  Force refresh
  };
};

//  Fonctions utilitaires pour le traitement des produits
const calculateDisplayPrice = (product: Product): number => {
  if (product.promotion && product.promotionEnd && new Date(product.promotionEnd) > new Date
    return product.price * (1 - product.promotion / 100);
  }
  return product.price;
};

const checkIfOnSale = (product: Product): boolean => {
  return !!(product.promotion && product.promotionEnd && new Date(product.promotionEnd) > ne
};

const getStockStatus = (stock: number): 'in_stock' | 'low_stock' | 'out_of_stock' => {
  if (stock <= 0) return 'out_of_stock';
  if (stock <= 5) return 'low_stock';
  return 'in_stock';
};
```

```typescript
//  Fonction de debounce pour optimiser les recherches
function debounce<T extends (...args: any[]) => void>(func: T, delay: number): T {
  let timeoutId: NodeJS.Timeout;
  return ((...args: any[]) => {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func(...args), delay);
  }) as T;
}
```

---

2. Service Layer Pattern (Backend)

Service de Sécurité Avancé

```javascript
/**
 * SERVICE DE SÉCURITÉ AVANCÉ - Backend
 *
 * Ce service gère tous les aspects de sécurité de l'application:
 * - Authentification et autorisation JWT
 * - Validation et sanitisation des données
 * - Rate limiting et protection contre les attaques
 * - Monitoring et alertes de sécurité
 * - Chiffrement des données sensibles
 *
 * Utilisation: Middleware appliqué automatiquement sur toutes les routes
 */

const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const rateLimit = require('express-rate-limit');
const helmet = require('helmet');
const xss = require('xss');

class SecurityService {
  constructor() {
    //  Configuration JWT avec rotation des secrets
    this.jwtConfig = {
      secret: process.env.JWT_SECRET || 'default-secret-change-in-production',
      expiresIn: '15m',           // Token d'accès courte durée
      refreshExpiresIn: '7d',     // Token de rafraîchissement
      algorithm: 'HS256',
      issuer: 'riziky-boutic',
      audience: 'riziky-users'
    };
```

```
    //  Configuration de sécurité des mots de passe
    this.passwordConfig = {
      saltRounds: 12,             // Coût de hachage bcrypt
      minLength: 8,               // Longueur minimale
      requireUppercase: true,     // Majuscule obligatoire
      requireLowercase: true,     // Minuscule obligatoire
      requireNumbers: true,       // Chiffre obligatoire
      requireSymbols: true        // Symbole obligatoire
    };

    //  Système de monitoring des tentatives
    this.securityMetrics = {
      failedLoginAttempts: new Map(),  // IP -> {count, lastAttempt}
      suspiciousActivity: new Map(),   // IP -> {events, severity}
      blockedIPs: new Set(),          // IPs temporairement bloquées
      activeTokens: new Map()         // Token -> {userId, createdAt}
    };
  }

  /**
   *  Génération de token JWT avec metadata de sécurité
   * @param {Object} payload - Données utilisateur à encoder
   * @param {string} type - Type de token ('access' | 'refresh')
   * @returns {string} Token JWT signé
   */
  generateToken(payload, type = 'access') {
    const now = Math.floor(Date.now() / 1000);
    const expiresIn = type === 'refresh' ? this.jwtConfig.refreshExpiresIn : this.jwtConfig.

    const tokenPayload = {
      ...payload,
      iat: now,                           // Émis à
      iss: this.jwtConfig.issuer,         // Émetteur
      aud: this.jwtConfig.audience,       // Audience
      type: type,                         // Type de token
      jti: this.generateTokenId()         // ID unique du token
    };

    const token = jwt.sign(tokenPayload, this.jwtConfig.secret, {
      expiresIn,
      algorithm: this.jwtConfig.algorithm
    });

    //  Enregistrement du token actif pour monitoring
    this.securityMetrics.activeTokens.set(token, {
      userId: payload.userId,
```

```
    type: type,
    createdAt: new Date(),
    lastUsed: new Date()
  });

  console.log(`  Token ${type} généré pour l'utilisateur ${payload.userId}`);
  return token;
}

/**
 *  Vérification et validation d'un token JWT
 * @param {string} token - Token à vérifier
 * @returns {Object} Payload décodé ou null si invalide
 */
verifyToken(token) {
  try {
    //  Vérification de la signature et validité
    const decoded = jwt.verify(token, this.jwtConfig.secret, {
      algorithms: [this.jwtConfig.algorithm],
      issuer: this.jwtConfig.issuer,
      audience: this.jwtConfig.audience
    });

    //  Mise à jour de la dernière utilisation
    const tokenInfo = this.securityMetrics.activeTokens.get(token);
    if (tokenInfo) {
      tokenInfo.lastUsed = new Date();
    }

    console.log(`  Token validé pour l'utilisateur ${decoded.userId}`);
    return decoded;

  } catch (error) {
    console.error(`  Token invalide:`, error.message);

    //  Nettoyage du token invalide
    this.securityMetrics.activeTokens.delete(token);

    return null;
  }
}

/**
 *  Hachage sécurisé des mots de passe
 * @param {string} password - Mot de passe en clair
 * @returns {Promise<string>} Hash bcrypt du mot de passe
```

```
 */
async hashPassword(password) {
  //  Validation de la force du mot de passe
  if (!this.validatePasswordStrength(password)) {
    throw new Error('Le mot de passe ne respecte pas les critères de sécurité');
  }

  try {
    const salt = await bcrypt.genSalt(this.passwordConfig.saltRounds);
    const hash = await bcrypt.hash(password, salt);

    console.log('  Mot de passe haché avec succès');
    return hash;

  } catch (error) {
    console.error('  Erreur lors du hachage du mot de passe:', error);
    throw new Error('Erreur lors du traitement du mot de passe');
  }
}

/**
 *  Vérification d'un mot de passe contre son hash
 * @param {string} password - Mot de passe en clair
 * @param {string} hash - Hash stocké en base
 * @returns {Promise<boolean>} True si le mot de passe correspond
 */
async verifyPassword(password, hash) {
  try {
    const isValid = await bcrypt.compare(password, hash);
    console.log(`  Vérification mot de passe: ${isValid ? 'SUCCÈS' : 'ÉCHEC'}`);
    return isValid;

  } catch (error) {
    console.error('  Erreur lors de la vérification du mot de passe:', error);
    return false;
  }
}

/**
 *  Validation de la force d'un mot de passe
 * @param {string} password - Mot de passe à valider
 * @returns {boolean} True si le mot de passe est assez fort
 */
validatePasswordStrength(password) {
  const { minLength, requireUppercase, requireLowercase, requireNumbers, requireSymbols }
```

```
  //  Vérification de la longueur minimale
  if (password.length < minLength) {
    console.log(` Mot de passe trop court (${password.length} < ${minLength})`);
    return false;
  }

  //  Vérification des caractères requis
  const checks = [
    { condition: requireUppercase, regex: /[A-Z]/, name: 'majuscule' },
    { condition: requireLowercase, regex: /[a-z]/, name: 'minuscule' },
    { condition: requireNumbers, regex: /\d/, name: 'chiffre' },
    { condition: requireSymbols, regex: /[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]/, name: 'sy
  ];

  for (const check of checks) {
    if (check.condition && !check.regex.test(password)) {
      console.log(` Mot de passe manque: ${check.name}`);
      return false;
    }
  }

  console.log(' Mot de passe respecte tous les critères');
  return true;
}

/**
 *  Sanitisation avancée des données d'entrée
 * @param {*} data - Données à nettoyer
 * @param {Object} rules - Règles de nettoyage
 * @returns {*} Données nettoyées
 */
sanitizeInput(data, rules = {}) {
  const defaultRules = {
    stripHtml: true,      // Supprimer HTML
    trimWhitespace: true, // Supprimer espaces
    maxLength: 1000,      // Longueur maximale
    allowedChars: null    // Caractères autorisés
  };

  const appliedRules = { ...defaultRules, ...rules };

  if (typeof data === 'string') {
    let sanitized = data;

    //  Suppression des balises HTML malveillantes
    if (appliedRules.stripHtml) {
```

```
      sanitized = xss(sanitized, {
        whiteList: {},          // Aucune balise autorisée
        stripIgnoreTag: true,   // Supprimer les balises inconnues
        stripIgnoreTagBody: ['script', 'style'] // Supprimer le contenu aussi
      });
    }

    //  Suppression des espaces en début/fin
    if (appliedRules.trimWhitespace) {
      sanitized = sanitized.trim();
    }

    //  Limitation de la longueur
    if (appliedRules.maxLength && sanitized.length > appliedRules.maxLength) {
      sanitized = sanitized.substring(0, appliedRules.maxLength);
      console.log(` Données tronquées à ${appliedRules.maxLength} caractères`);
    }

    //  Filtrage des caractères autorisés
    if (appliedRules.allowedChars) {
      sanitized = sanitized.replace(appliedRules.allowedChars, '');
    }

    return sanitized;

  } else if (typeof data === 'object' && data !== null) {
    //  Nettoyage récursif des objets
    const sanitized = {};
    for (const [key, value] of Object.entries(data)) {
      sanitized[key] = this.sanitizeInput(value, rules);
    }
    return sanitized;

  } else if (Array.isArray(data)) {
    //  Nettoyage des tableaux
    return data.map(item => this.sanitizeInput(item, rules));
  }

  return data;
}

/**
 *  Détection d'activité suspecte
 * @param {string} ip - Adresse IP à analyser
 * @param {string} event - Type d'événement
 * @param {Object} context - Contexte additionnel
```

```
 * @returns {boolean} True si l'activité est suspecte
 */
detectSuspiciousActivity(ip, event, context = {}) {
  //  Vérification si l'IP est déjà bloquée
  if (this.securityMetrics.blockedIPs.has(ip)) {
    console.log(` IP bloquée tentant d'accéder: ${ip}`);
    return true;
  }

  //  Récupération ou création de l'historique IP
  if (!this.securityMetrics.suspiciousActivity.has(ip)) {
    this.securityMetrics.suspiciousActivity.set(ip, {
      events: [],
      score: 0,
      firstSeen: new Date()
    });
  }

  const ipActivity = this.securityMetrics.suspiciousActivity.get(ip);

  //  Enregistrement de l'événement
  ipActivity.events.push({
    type: event,
    timestamp: new Date(),
    context: context
  });

  //  Calcul du score de suspicion
  const suspicionScore = this.calculateSuspicionScore(ipActivity.events);
  ipActivity.score = suspicionScore;

  //  Seuil d'alerte dépassé
  if (suspicionScore > 50) {
    console.log(` ALERTE: Activité hautement suspecte détectée pour IP ${ip} (score: ${su

    //  Blocage temporaire de l'IP
    this.blockIP(ip, '1h');

    //  Notification aux administrateurs
    this.notifyAdmins('suspicious_activity', {
      ip: ip,
      score: suspicionScore,
      events: ipActivity.events.slice(-5) // 5 derniers événements
    });

    return true;
```

```
  }

  return false;
}

/**
 *  Calcul du score de suspicion basé sur les événements
 * @param {Array} events - Liste des événements de l'IP
 * @returns {number} Score de suspicion (0-100)
 */
calculateSuspicionScore(events) {
  let score = 0;
  const now = new Date();
  const recentEvents = events.filter(e => now - e.timestamp < 3600000); // 1 heure

  //  Scoring basé sur la fréquence
  const eventCounts = {};
  recentEvents.forEach(event => {
    eventCounts[event.type] = (eventCounts[event.type] || 0) + 1;
  });

  //  Règles de scoring
  const scoringRules = {
    'failed_login': 10,      // Tentative de connexion échouée
    'invalid_token': 15,     // Token invalide utilisé
    'rate_limit_hit': 20,    // Rate limit atteint
    'sql_injection': 50,     // Tentative d'injection SQL
    'xss_attempt': 40,       // Tentative XSS
    'directory_traversal': 45 // Tentative de directory traversal
  };

  //  Calcul du score total
  for (const [eventType, count] of Object.entries(eventCounts)) {
    const baseScore = scoringRules[eventType] || 5;
    score += baseScore * count;

    //  Bonus pour les événements répétés
    if (count > 3) {
      score += (count - 3) * 10;
    }
  }

  return Math.min(score, 100); // Cap à 100
}

/**
```

```
 *  Blocage temporaire d'une adresse IP
 * @param {string} ip - IP à bloquer
 * @param {string} duration - Durée du blocage
 */
blockIP(ip, duration) {
  this.securityMetrics.blockedIPs.add(ip);

  //  Conversion de la durée en millisecondes
  const durationMs = this.parseDuration(duration);

  setTimeout(() => {
    this.securityMetrics.blockedIPs.delete(ip);
    console.log(`  IP ${ip} débloquée après ${duration}`);
  }, durationMs);

  console.log(`  IP ${ip} bloquée pour ${duration}`);
}

/**
 *  Conversion d'une durée textuelle en millisecondes
 * @param {string} duration - Durée (ex: "1h", "30m", "10s")
 * @returns {number} Durée en millisecondes
 */
parseDuration(duration) {
  const match = duration.match(/^(\d+)([smhd])$/);
  if (!match) return 3600000; // 1 heure par défaut

  const value = parseInt(match[1]);
  const unit = match[2];

  const multipliers = {
    's': 1000,        // secondes
    'm': 60000,       // minutes
    'h': 3600000,     // heures
    'd': 86400000     // jours
  };

  return value * (multipliers[unit] || 3600000);
}

/**
 *  Génération d'un ID unique pour les tokens
 * @returns {string} ID unique
 */
generateTokenId() {
  return `token_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`;
```

```
}

/**
 *  Notification aux administrateurs
 * @param {string} type - Type d'alerte
 * @param {Object} data - Données de l'alerte
 */
notifyAdmins(type, data) {
  //  Log de l'alerte pour le monitoring
  console.log(` NOTIFICATION ADMIN [${type}]:`, JSON.stringify(data, null, 2));

  // TODO: Implémenter l'envoi d'email ou webhook vers système de monitoring
  // EmailService.sendAlert(type, data);
  // WebhookService.sendAlert(type, data);
}

/**
 *  Nettoyage périodique des métriques de sécurité
 */
cleanupSecurityMetrics() {
  const now = new Date();
  const maxAge = 24 * 60 * 60 * 1000; // 24 heures

  //  Nettoyage des tentatives de connexion échouées
  for (const [ip, data] of this.securityMetrics.failedLoginAttempts.entries()) {
    if (now - data.lastAttempt > maxAge) {
      this.securityMetrics.failedLoginAttempts.delete(ip);
    }
  }

  //  Nettoyage de l'activité suspecte
  for (const [ip, data] of this.securityMetrics.suspiciousActivity.entries()) {
    if (now - data.firstSeen > maxAge) {
      this.securityMetrics.suspiciousActivity.delete(ip);
    }
  }

  //  Nettoyage des tokens expirés
  for (const [token, data] of this.securityMetrics.activeTokens.entries()) {
    if (now - data.lastUsed > maxAge) {
      this.securityMetrics.activeTokens.delete(token);
    }
  }

  console.log(' Nettoyage des métriques de sécurité effectué');
}
```

```
}

// Export du service de sécurité singleton
const securityService = new SecurityService();

// Nettoyage automatique toutes les heures
setInterval(() => {
  securityService.cleanupSecurityMetrics();
}, 3600000);

module.exports = securityService;
```

This enhanced documentation showcases the detailed code structure and advanced patterns used throughout the Riziky-Boutic platform. The code is thoroughly commented in French to explain each functionality and its purpose.