Manuel du Développeur - Riziky-Boutic

Introduction Technique

Ce manuel est destiné aux développeurs qui travaillent sur la plateforme Riziky-Boutic ou qui souhaitent comprendre son architecture technique en profondeur. Il couvre tous les aspects du développement, de la maintenance, et de l'évolution de l'application.

Architecture Technique Détaillée

Stack Technologique Complète

Frontend (Client React)

```
Framework: React 18.3.1 avec TypeScript 5.0+
Build: Vite 4.0+ (bundling ultra-rapide)
Styling: Tailwind CSS 3.3+ (utility-first)
UI: Shadcn/UI (composants pré-construits)
Routing: React Router 6.26+ (client-side routing)
State: Context API + Custom Hooks (pas de Redux)
Forms: React Hook Form + Zod (validation)
HTTP: Axios (client configuré avec intercepteurs)
WebSocket: Socket.io-client (temps réel)
Testing: Jest + React Testing Library
E2E: Cypress (tests end-to-end)
```

Backend (Serveur Node.js)

```
Runtime: Node.js 18+ (runtime JavaScript serveur)
Framework: Express.js 4.18+ (framework web minimaliste)
Auth: JWT (JSON Web Tokens) + bcrypt (hachage)
Upload: Multer (gestion fichiers multipart)
Security: Helmet + CORS + XSS-Clean + Rate Limiting
WebSocket: Socket.io (communication bidirectionnelle)
Validation: Express-validator + custom sanitizers
Logging: Winston (logging structuré)
Testing: Jest + Supertest (tests API)
Process: PM2 (gestionnaire de processus)
```

Base de Données et Stockage

```
Actuel: JSON Files (développement/prototype)
Migration: PostgreSQL 14+ (production recommandée)
Cache: Node-cache (en mémoire) + Redis (externe)
Files: Local storage + AWS S3 (cloud ready)
```

Search: Fuse.js (recherche fuzzy) + Elasticsearch (ready)
Backup: Automated JSON backup + DB dumps

Architecture en Couches

COUCHE PRÉSENTATION

React Components + Tailwind + Shadcn/UI

- Pages (routing)
- Components (réutilisables)
- UI primitives (boutons, inputs, etc.)

COUCHE LOGIQUE MÉTIER

Custom Hooks + Contexts + Utils

- useAuth, useCart, useProducts
- AuthContext, StoreContext
- Business logic encapsulée

COUCHE COMMUNICATION

API Services + Socket.io Client

- HTTP requests (Axios)
- WebSocket connections
- Error handling centralisé

COUCHE API SERVEUR

Express Routes + Middlewares + Socket Handlers

- REST endpoints
- Authentication middleware
- Business logic services

COUCHE DONNÉES

Database Services + File Storage

- JSON file operations
- PostgreSQL queries (migration)

- File upload/management

Configuration et Installation Développeur

Prérequis Système

Outils Obligatoires :

```
# Node.js et NPM
Node.js >= 18.0.0 (LTS recommandé)
npm >= 8.0.0 ou yarn >= 1.22.0

# Git pour le versioning
Git >= 2.30.0

# Éditeur recommandé
VS Code avec extensions :
- ES7+ React/Redux/React-Native snippets
- Tailwind CSS IntelliSense
- TypeScript Importer
- ESLint
- Prettier
- GitLens
```

Outils de Développement :

```
# CLI globaux utiles
npm install -g @types/node
npm install -g typescript
npm install -g nodemon
npm install -g pm2

# Pour les tests E2E
npm install -g cypress

# Base de données (optionnel pour dev)
PostgreSQL 14+ ou Docker Desktop
```

Installation Complète Étape par Étape

1. Clone et Setup Initial :

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

```
# Vérifier les versions
node --version  # Doit être >= 18.0.0
npm --version   # Doit être >= 8.0.0

# Installer les dépendances frontend
npm install

# Vérifier l'installation
npm list --depth=0
```

2. Configuration Backend :

```
# Naviguer vers le backend
cd server

# Installer les dépendances backend
npm install

# Copier la configuration d'exemple
cp .env.example .env

# Éditer la configuration (voir section Variables d'Environnement)
nano .env  # ou code .env

# Initialiser la base de données de développement
npm run init-db

# Tester le backend
npm run test

# Retourner à la racine
cd ..
```

3. Configuration IDE (VS Code) :

```
// .vscode/settings.json (créer si inexistant)
{
  "typescript.preferences.importModuleSpecifier": "relative",
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
  },
  "tailwindCSS.includeLanguages": {
    "typescript": "javascript",
    "typescriptreact": "javascript"
  },
  "files.associations": {
```

```
    "*.css": "tailwindcss"
  }
}

// .vscode/launch.json (pour debugging)
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug Backend",
      "type": "node",
      "request": "launch",
      "program": "${workspaceFolder}/server/server.js",
      "env": {
        "NODE_ENV": "development"
      },
      "console": "integratedTerminal",
      "restart": true,
      "runtimeExecutable": "nodemon"
    }
  ]
}
```

Variables d'Environnement Détaillées

Fichier `.env` (racine du projet) :

```
# ========================================
# Configuration Frontend (Vite)
# ========================================
VITE_API_BASE_URL=http://localhost:10000
VITE_APP_NAME=Riziky-Boutic
VITE_APP_VERSION=1.0.0
VITE_ENVIRONMENT=development

# ========================================
# Configuration Générale
# ========================================
NODE_ENV=development
PORT=10000
FRONTEND_URL=http://localhost:8080

# ========================================
# Sécurité JWT
# ========================================
JWT_SECRET=votre_secret_jwt_super_securise_minimum_32_caracteres
JWT_EXPIRES_IN=24h
```

```
JWT_REFRESH_SECRET=autre_secret_pour_refresh_tokens
JWT_REFRESH_EXPIRES_IN=7d

# ========================================
# Upload et Stockage
# ========================================
UPLOAD_MAX_SIZE=5242880
UPLOAD_ALLOWED_TYPES=image/jpeg,image/png,image/webp,image/gif
UPLOADS_DIR=./uploads
MAX_FILES_PER_UPLOAD=10

# ========================================
# CORS et Sécurité
# ========================================
CORS_ORIGIN=http://localhost:8080,http://localhost:3000
CORS_CREDENTIALS=true

# ========================================
# Rate Limiting
# ========================================
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
RATE_LIMIT_LOGIN_MAX=5
RATE_LIMIT_LOGIN_WINDOW=900000

# ========================================
# Base de Données (pour migration PostgreSQL)
# ========================================
DATABASE_URL=postgresql://username:password@localhost:5432/riziky_boutic
DB_HOST=localhost
DB_PORT=5432
DB_NAME=riziky_boutic
DB_USER=username
DB_PASSWORD=password
DB_SSL=false

# ========================================
# Email (pour notifications)
# ========================================
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=votre-email@gmail.com
SMTP_PASSWORD=votre-mot-de-passe-app
EMAIL_FROM=noreply@riziky-boutic.com

# ========================================
```

```
# Services Externes
# ========================================
# Paiement (Stripe)
STRIPE_PUBLIC_KEY=pk_test_...
STRIPE_SECRET_KEY=sk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...

# AWS S3 (stockage cloud)
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=...
AWS_REGION=eu-west-1
AWS_S3_BUCKET=riziky-boutic-uploads

# Google Analytics
GA_MEASUREMENT_ID=G-XXXXXXXXXX

# ========================================
# Développement et Debug
# ========================================
DEBUG_MODE=true
LOG_LEVEL=debug
ENABLE_CORS=true
ENABLE_MORGAN=true
```

Fichier `server/.env` (backend spécifique) :

```
# Héritage de la configuration globale
# Variables spécifiques au serveur si nécessaire
SERVER_NAME=Riziky-Boutic-API
API_VERSION=v1
ENABLE_SWAGGER=true
SWAGGER_URL=/api-docs
```

Scripts de Développement

Scripts Package.json (racine) :

```
{
  "scripts": {
    "dev": "vite --host 0.0.0.0 --port 8080",
    "build": "tsc && vite build",
    "preview": "vite preview",
    "server": "cd server && node server.js",
    "server:dev": "cd server && nodemon server.js",
    "dev:full": "concurrently \"npm run server:dev\" \"npm run dev\"",

    "test": "jest",
```

```
    "test:watch": "jest --watch",
    "test:coverage": "jest --coverage",
    "test:server": "cd server && npm test",
    "test:e2e": "cypress run",
    "test:e2e:open": "cypress open",

    "lint": "eslint src --ext .ts,.tsx",
    "lint:fix": "eslint src --ext .ts,.tsx --fix",
    "format": "prettier --write \"src/**/*.{ts,tsx,js,jsx,json,css,md}\"",
    "type-check": "tsc --noEmit",

    "prepare": "husky install",
    "pre-commit": "lint-staged"
  }
}
```

Scripts Serveur (server/package.json) :

```
{
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "jest",
    "test:watch": "jest --watch",
    "init-db": "node scripts/init-database.js",
    "seed": "node scripts/seed-data.js",
    "backup": "node scripts/backup-data.js",
    "migrate": "node scripts/migrate-to-postgres.js"
  }
}
```

Structure du Code et Organisation

Arborescence Frontend Complète

```
src/
  app/                          # Configuration application
     AppProviders.tsx          # Providers React (Auth, Theme, etc.)
     AppRoutes.tsx             # Configuration des routes
     LoadingFallback.tsx       # Composant de chargement global

  components/                   # Composants réutilisables
     ui/                       # Composants UI primitifs (Shadcn)
        button.tsx
        input.tsx
        card.tsx
```

```
        ...

    layout/                # Composants de mise en page
        Navbar.tsx
        Footer.tsx
        Sidebar.tsx
        Layout.tsx

    products/              # Composants produits
        ProductCard.tsx
        ProductGrid.tsx
        ProductDetail.tsx
        ProductFilters.tsx

    cart/                  # Composants panier
        CartDrawer.tsx
        CartItem.tsx
        CartSummary.tsx

    auth/                  # Composants authentification
        LoginForm.tsx
        RegisterForm.tsx
        PasswordInput.tsx

    admin/                 # Interface administration
        Dashboard.tsx
        ProductManager.tsx
        UserManager.tsx

    common/                # Composants communs
         LoadingSpinner.tsx
         ErrorBoundary.tsx
         ConfirmDialog.tsx

hooks/                      # Hooks personnalisés
    useAuth.ts             # Authentification
    useCart.ts             # Panier
    useProducts.ts         # Produits
    useFavorites.ts        # Favoris
    useOrders.ts           # Commandes
    useLocalStorage.ts     # Stockage local

contexts/                   # Contextes React
    AuthContext.tsx        # Contexte d'authentification
    StoreContext.tsx       # Contexte global du magasin
    ThemeContext.tsx       # Contexte du thème
```

```
services/                    # Services et API
    core/                   # Configuration centrale
        apiClient.ts        # Client HTTP configuré
        errorHandler.ts     # Gestion centralisée des erreurs
        interceptors.ts     # Intercepteurs Axios

    modules/                # Services par domaine métier
        auth.service.ts
        products.service.ts
        cart.service.ts
        orders.service.ts

    security/               # Services de sécurité
        secureIds.ts
        encryption.ts
        validation.ts

    utils/                  # Utilitaires de service
         api.utils.ts
         cache.utils.ts

types/                       # Définitions TypeScript
    auth.types.ts
    product.types.ts
    order.types.ts
    cart.types.ts
    api.types.ts

utils/                       # Fonctions utilitaires
    formatters.ts           # Formatage (prix, dates, etc.)
    validators.ts           # Validations
    constants.ts            # Constantes
    helpers.ts              # Fonctions d'aide

lib/                         # Configurations de librairies
    utils.ts                # Utilitaires (cn function, etc.)
    validations.ts          # Schémas Zod
    api.ts                  # Configuration API

pages/                       # Pages de l'application
    HomePage.tsx
    ProductsPage.tsx
    ProductDetail.tsx
    CartPage.tsx
    CheckoutPage.tsx
```

```
    LoginPage.tsx
    RegisterPage.tsx
    ProfilePage.tsx
    OrdersPage.tsx
    admin/                  # Pages administration
        AdminDashboard.tsx
        AdminProducts.tsx
        AdminOrders.tsx

  assets/                    # Assets statiques
    images/
    icons/
    fonts/

  styles/                    # Styles globaux
    globals.css
    components.css
    tailwind.css

  tests/                     # Tests
     __mocks__/             # Mocks
     components/            # Tests composants
     hooks/                 # Tests hooks
     services/              # Tests services
     utils/                 # Tests utilitaires
```

Arborescence Backend Complète

```
server/
  config/                    # Configuration
    database.js            # Configuration DB
    auth.js                # Configuration JWT
    cors.js                # Configuration CORS
    upload.js              # Configuration upload
    environment.js         # Variables d'environnement

  controllers/               # Contrôleurs (optionnel)
    AuthController.js
    ProductsController.js
    OrdersController.js

  routes/                    # Routes Express
    auth.js                # Routes d'authentification
    products.js            # Routes des produits
    cart.js                # Routes du panier
    orders.js              # Routes des commandes
```

```
    users.js              # Routes des utilisateurs
    admin.js              # Routes d'administration
    uploads.js            # Routes d'upload

middlewares/               # Middlewares Express
    auth.js               # Middleware d'authentification
    validation.js         # Middleware de validation
    security.js           # Middlewares de sécurité
    upload.js             # Middleware d'upload
    cors.js               # Configuration CORS
    rateLimit.js          # Rate limiting
    errorHandler.js       # Gestion d'erreurs

services/                  # Services métier
    AuthService.js        # Logique d'authentification
    ProductsService.js    # Logique des produits
    CartService.js        # Logique du panier
    OrdersService.js      # Logique des commandes
    EmailService.js       # Service d'email
    FileService.js        # Gestion des fichiers
    SecurityService.js    # Services de sécurité

core/                      # Modules centraux
    database.js           # Gestionnaire de base de données
    logger.js             # Système de logging
    cache.js              # Gestion du cache
    scheduler.js          # Tâches programmées

socket/                    # Configuration WebSocket
    socketConfig.js       # Configuration Socket.io
    socketHandlers.js     # Gestionnaires d'événements
    socketAuth.js         # Authentification WebSocket
    socketMiddleware.js   # Middlewares Socket

utils/                     # Utilitaires
    validation.js         # Fonctions de validation
    encryption.js         # Chiffrement/hachage
    fileUtils.js          # Utilitaires fichiers
    dateUtils.js          # Utilitaires dates

data/                      # Données JSON (développement)
    users.json
    products.json
    orders.json
    categories.json
    settings.json
```

```
uploads/                    # Fichiers uploadés
    products/              # Images produits
    avatars/               # Photos de profil
    documents/             # Documents divers

tests/                      # Tests backend
    unit/                  # Tests unitaires
    integration/           # Tests d'intégration
    fixtures/              # Données de test
    helpers/               # Utilitaires de test

scripts/                    # Scripts utilitaires
    init-database.js       # Initialisation DB
    seed-data.js           # Données de test
    migrate.js             # Migration vers PostgreSQL
    backup.js              # Sauvegarde automatique

docs/                       # Documentation API
    api-spec.yaml          # Spécification OpenAPI
    postman-collection.json

logs/                       # Logs serveur
    error.log
    combined.log
    access.log

.env.example                # Template variables d'env
server.js                   # Point d'entrée
package.json
package-lock.json
```

Patterns et Bonnes Pratiques

Patterns Frontend

1. Pattern de Composant Réutilisable Structure Standard :

```
// interfaces/ComponentProps.ts
interface ComponentProps {
  // Props obligatoires
  data: DataType;
  onAction: (id: string) => Promise<void>;

  // Props optionnelles avec valeurs par défaut
```

```tsx
  variant?: 'default' | 'compact' | 'detailed';
  showActions?: boolean;
  isLoading?: boolean;

  // Props de style
  className?: string;

  // Enfants optionnels
  children?: React.ReactNode;
}

// Component.tsx
import { FC, useState, useCallback, useMemo } from 'react';
import { cn } from '@/lib/utils';

export const Component: FC<ComponentProps> = ({
  data,
  onAction,
  variant = 'default',
  showActions = true,
  isLoading = false,
  className,
  children
}) => {
  // 1. État local en premier
  const [localState, setLocalState] = useState<LocalStateType>();

  // 2. Hooks personnalisés
  const { customData, customAction } = useCustomHook(data.id);

  // 3. Callbacks memoized
  const handleAction = useCallback(async () => {
    try {
      await onAction(data.id);
    } catch (error) {
      console.error('Action failed:', error);
    }
  }, [onAction, data.id]);

  // 4. Valeurs calculées memoized
  const computedValue = useMemo(() => {
    return heavyComputation(data);
  }, [data]);

  // 5. Styles conditionnels
  const componentClasses = cn(
```

```
    'base-component-classes',
    {
      'variant-compact': variant === 'compact',
      'variant-detailed': variant === 'detailed',
      'is-loading': isLoading
    },
    className
  );

  // 6. Early returns
  if (!data) {
    return <ComponentSkeleton />;
  }

  // 7. Rendu principal
  return (
    <div className={componentClasses}>
      <ComponentHeader data={data} />

      {children && (
        <div className="component-content">
          {children}
        </div>
      )}

      {showActions && (
        <ComponentActions
          onAction={handleAction}
          isLoading={isLoading}
          disabled={!customData}
        />
      )}
    </div>
  );
};

// Export avec displayName pour debugging
Component.displayName = 'Component';
```

2. Pattern de Hook Personnalisé Hook avec État et Actions :

```
// hooks/useFeature.ts
interface UseFeatureOptions {
  enableRealTimeUpdates?: boolean;
  cacheTimeout?: number;
  onError?: (error: Error) => void;
```

```typescript
}

interface UseFeatureReturn {
  // État
  data: DataType[];
  isLoading: boolean;
  error: Error | null;
  isEmpty: boolean;

  // Actions
  create: (input: CreateInput) => Promise<DataType>;
  update: (id: string, input: UpdateInput) => Promise<DataType>;
  delete: (id: string) => Promise<void>;
  refresh: () => Promise<void>;

  // Utilitaires
  findById: (id: string) => DataType | undefined;
  filterBy: (predicate: (item: DataType) => boolean) => DataType[];
}

export const useFeature = (
  options: UseFeatureOptions = {}
): UseFeatureReturn => {
  const {
    enableRealTimeUpdates = false,
    cacheTimeout = 5 * 60 * 1000, // 5 minutes
    onError = console.error
  } = options;

  // État local
  const [data, setData] = useState<DataType[]>([]);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState<Error | null>(null);
  const [lastFetch, setLastFetch] = useState<number>(0);

  // Service API
  const api = useMemo(() => new FeatureService(), []);

  // Fonction de fetch avec cache
  const fetchData = useCallback(async (forceRefresh = false) => {
    const now = Date.now();
    const isCacheValid = (now - lastFetch) < cacheTimeout;

    if (!forceRefresh && isCacheValid && data.length > 0) {
      return data;
    }
```

```
    setIsLoading(true);
    setError(null);

    try {
      const result = await api.getAll();
      setData(result);
      setLastFetch(now);
      return result;
    } catch (err) {
      const error = err as Error;
      setError(error);
      onError(error);
      throw error;
    } finally {
      setIsLoading(false);
    }
  }, [api, data, lastFetch, cacheTimeout, onError]);

  // Actions CRUD
  const create = useCallback(async (input: CreateInput): Promise<DataType> => {
    try {
      const newItem = await api.create(input);
      setData(prev => [...prev, newItem]);
      return newItem;
    } catch (err) {
      const error = err as Error;
      setError(error);
      onError(error);
      throw error;
    }
  }, [api, onError]);

  const update = useCallback(async (id: string, input: UpdateInput): Promise<DataType> => {
    try {
      const updatedItem = await api.update(id, input);
      setData(prev => prev.map(item =>
        item.id === id ? updatedItem : item
      ));
      return updatedItem;
    } catch (err) {
      const error = err as Error;
      setError(error);
      onError(error);
      throw error;
    }
```

```
}, [api, onError]);

const deleteItem = useCallback(async (id: string): Promise<void> => {
  try {
    await api.delete(id);
    setData(prev => prev.filter(item => item.id !== id));
  } catch (err) {
    const error = err as Error;
    setError(error);
    onError(error);
    throw error;
  }
}, [api, onError]);

// Utilitaires
const findById = useCallback((id: string) => {
  return data.find(item => item.id === id);
}, [data]);

const filterBy = useCallback((predicate: (item: DataType) => boolean) => {
  return data.filter(predicate);
}, [data]);

// Fetch initial
useEffect(() => {
  fetchData();
}, [fetchData]);

// WebSocket pour mises à jour temps réel
useEffect(() => {
  if (!enableRealTimeUpdates) return;

  const socket = socketService.connect();

  socket.on('feature:created', (newItem: DataType) => {
    setData(prev => [...prev, newItem]);
  });

  socket.on('feature:updated', (updatedItem: DataType) => {
    setData(prev => prev.map(item =>
      item.id === updatedItem.id ? updatedItem : item
    ));
  });

  socket.on('feature:deleted', (deletedId: string) => {
    setData(prev => prev.filter(item => item.id !== deletedId));
```

```
    });

    return () => {
      socket.disconnect();
    };
  }, [enableRealTimeUpdates]);

  // Valeurs calculées
  const isEmpty = data.length === 0;

  return {
    data,
    isLoading,
    error,
    isEmpty,
    create,
    update,
    delete: deleteItem,
    refresh: () => fetchData(true),
    findById,
    filterBy
  };
};
```

3. Pattern de Service API Service avec Gestion d'Erreur :

```
// services/BaseService.ts
export abstract class BaseService {
  protected baseUrl: string;
  protected client: AxiosInstance;

  constructor(baseUrl: string) {
    this.baseUrl = baseUrl;
    this.client = apiClient; // Instance Axios configurée
  }

  protected async handleRequest<T>(
    request: () => Promise<AxiosResponse<T>>
  ): Promise<T> {
    try {
      const response = await request();
      return response.data;
    } catch (error) {
      this.handleError(error);
      throw error; // Re-throw pour que le caller puisse gérer
    }
```

```
  }

  protected handleError(error: any): void {
    if (axios.isAxiosError(error)) {
      const { response, request, message } = error;

      if (response) {
        // Erreur de réponse serveur
        console.error(`API Error ${response.status}:`, response.data);

        // Gestion spécifique par code de statut
        switch (response.status) {
          case 401:
            // Rediriger vers login
            authService.logout();
            break;
          case 403:
            toast.error('Accès non autorisé');
            break;
          case 404:
            toast.error('Ressource non trouvée');
            break;
          case 500:
            toast.error('Erreur serveur interne');
            break;
          default:
            toast.error('Une erreur est survenue');
        }
      } else if (request) {
        // Erreur de réseau
        console.error('Network Error:', message);
        toast.error('Problème de connexion réseau');
      } else {
        // Erreur de configuration
        console.error('Request Error:', message);
        toast.error('Erreur de configuration');
      }
    } else {
      // Erreur non-Axios
      console.error('Unknown Error:', error);
      toast.error('Erreur inattendue');
    }
  }
}

// services/ProductsService.ts
```

```typescript
interface ProductsResponse {
  products: Product[];
  pagination: {
    total: number;
    page: number;
    limit: number;
    totalPages: number;
  };
}

interface ProductFilters {
  category?: string;
  minPrice?: number;
  maxPrice?: number;
  search?: string;
  inStock?: boolean;
}

class ProductsService extends BaseService {
  constructor() {
    super('/api/products');
  }

  async getAll(
    page = 1,
    limit = 12,
    filters: ProductFilters = {}
  ): Promise<ProductsResponse> {
    return this.handleRequest(async () => {
      return this.client.get(this.baseUrl, {
        params: { page, limit, ...filters }
      });
    });
  }

  async getById(id: string): Promise<Product> {
    // Sécuriser l'ID avant envoi
    const secureId = securityService.encodeId(id);

    return this.handleRequest(async () => {
      return this.client.get(`${this.baseUrl}/${secureId}`);
    });
  }

  async create(productData: CreateProductData): Promise<Product> {
    // Validation côté client
```

```
  const validatedData = ProductSchema.parse(productData);

  // Préparation FormData pour les images
  const formData = new FormData();

  Object.entries(validatedData).forEach(([key, value]) => {
    if (key === 'images' && Array.isArray(value)) {
      value.forEach(file => formData.append('images', file));
    } else {
      formData.append(key, String(value));
    }
  });

  return this.handleRequest(async () => {
    return this.client.post(this.baseUrl, formData, {
      headers: { 'Content-Type': 'multipart/form-data' },
      timeout: 30000 // 30s pour les uploads
    });
  });
}

async update(id: string, updateData: UpdateProductData): Promise<Product> {
  const secureId = securityService.encodeId(id);
  const validatedData = UpdateProductSchema.parse(updateData);

  return this.handleRequest(async () => {
    return this.client.put(`${this.baseUrl}/${secureId}`, validatedData);
  });
}

async delete(id: string): Promise<void> {
  const secureId = securityService.encodeId(id);

  return this.handleRequest(async () => {
    return this.client.delete(`${this.baseUrl}/${secureId}`);
  });
}

// Méthodes spécialisées
async search(query: string, filters: ProductFilters = {}): Promise<Product[]> {
  return this.handleRequest(async () => {
    return this.client.get(`${this.baseUrl}/search`, {
      params: { q: query, ...filters }
    });
  });
}
```

```
  async getByCategory(category: string): Promise<Product[]> {
    return this.handleRequest(async () => {
      return this.client.get(`${this.baseUrl}/category/${category}`);
    });
  }

  async getFeatured(): Promise<Product[]> {
    return this.handleRequest(async () => {
      return this.client.get(`${this.baseUrl}/featured`);
    });
  }
}

export const productsService = new ProductsService();
```

Patterns Backend

1. Pattern de Route Express Structure Standard :

```
// routes/products.js
const express = require('express');
const { body, param, query, validationResult } = require('express-validator');
const { authenticateToken, requireRole } = require('../middlewares/auth');
const rateLimit = require('express-rate-limit');
const ProductsService = require('../services/ProductsService');

const router = express.Router();

// Rate limiting spécifique aux produits
const productsLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // 100 requêtes par fenêtre
  message: { error: 'Trop de requêtes sur les produits' }
});

// Validation des paramètres de requête
const validateProductQuery = [
  query('page')
    .optional()
    .isInt({ min: 1 })
    .withMessage('Page doit être un entier positif'),
  query('limit')
    .optional()
    .isInt({ min: 1, max: 100 })
    .withMessage('Limite doit être entre 1 et 100'),
```

```
  query('category')
    .optional()
    .isAlphanumeric()
    .withMessage('Catégorie invalide'),
  query('minPrice')
    .optional()
    .isFloat({ min: 0 })
    .withMessage('Prix minimum invalide'),
  query('maxPrice')
    .optional()
    .isFloat({ min: 0 })
    .withMessage('Prix maximum invalide')
];

// Validation création produit
const validateProductCreation = [
  body('nom')
    .trim()
    .isLength({ min: 3, max: 100 })
    .withMessage('Nom doit faire entre 3 et 100 caractères')
    .escape(),
  body('description')
    .optional()
    .isLength({ max: 1000 })
    .withMessage('Description trop longue')
    .escape(),
  body('prix')
    .isFloat({ min: 0.01 })
    .withMessage('Prix doit être positif')
    .toFloat(),
  body('stock')
    .isInt({ min: 0 })
    .withMessage('Stock doit être un entier positif')
    .toInt(),
  body('category')
    .notEmpty()
    .withMessage('Catégorie requise')
    .isAlphanumeric()
    .withMessage('Catégorie invalide')
];

// Middleware de validation
const handleValidationErrors = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({
```

```
      success: false,
      error: 'Données invalides',
      details: errors.array()
    });
  }
  next();
};

// Routes publiques
router.get('/',
  productsLimiter,
  validateProductQuery,
  handleValidationErrors,
  async (req, res) => {
    try {
      const {
        page = 1,
        limit = 12,
        category,
        minPrice,
        maxPrice,
        search,
        sortBy = 'nom',
        sortOrder = 'asc'
      } = req.query;

      const filters = {
        category,
        minPrice: minPrice ? parseFloat(minPrice) : undefined,
        maxPrice: maxPrice ? parseFloat(maxPrice) : undefined,
        search
      };

      // Filtrer les valeurs undefined
      Object.keys(filters).forEach(key => {
        if (filters[key] === undefined) {
          delete filters[key];
        }
      });

      const result = await ProductsService.getProducts({
        page: parseInt(page),
        limit: parseInt(limit),
        filters,
        sort: { [sortBy]: sortOrder }
      });
```

```
      res.json({
        success: true,
        data: result.products,
        pagination: result.pagination,
        meta: {
          total: result.total,
          filters: filters,
          sort: { [sortBy]: sortOrder }
        }
      });

    } catch (error) {
      console.error('Error fetching products:', error);
      res.status(500).json({
        success: false,
        error: 'Erreur lors de la récupération des produits'
      });
    }
  }
);

router.get('/:id',
  param('id').notEmpty().withMessage('ID requis'),
  handleValidationErrors,
  async (req, res) => {
    try {
      const { id } = req.params;

      // Décoder l'ID sécurisé
      const realId = securityService.decodeId(id);

      const product = await ProductsService.getProductById(realId);

      if (!product) {
        return res.status(404).json({
          success: false,
          error: 'Produit non trouvé'
        });
      }

      // Incrémenter le compteur de vues
      await ProductsService.incrementViews(realId);

      res.json({
        success: true,
```

```
        data: product
      });

    } catch (error) {
      if (error.name === 'InvalidIdError') {
        return res.status(400).json({
          success: false,
          error: 'ID produit invalide'
        });
      }

      console.error('Error fetching product:', error);
      res.status(500).json({
        success: false,
        error: 'Erreur lors de la récupération du produit'
      });
    }
  }
);

// Routes protégées (admin uniquement)
router.post('/',
  authenticateToken,
  requireRole('admin'),
  upload.array('images', 10), // Middleware upload
  validateProductCreation,
  handleValidationErrors,
  async (req, res) => {
    try {
      const productData = {
        ...req.body,
        createdBy: req.user.id,
        images: req.files?.map(file => ({
          url: `/uploads/${file.filename}`,
          alt: req.body.nom,
          size: file.size,
          mimetype: file.mimetype
        })) || []
      };

      const product = await ProductsService.createProduct(productData);

      // Log de l'action admin
      console.log(`Admin ${req.user.email} created product: ${product.id}`);

      res.status(201).json({
```

```
        success: true,
        data: product,
        message: 'Produit créé avec succès'
      });

    } catch (error) {
      console.error('Error creating product:', error);

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

      res.status(500).json({
        success: false,
        error: 'Erreur lors de la création du produit'
      });
    }
  }
);

router.put('/:id',
  authenticateToken,
  requireRole('admin'),
  param('id').notEmpty().withMessage('ID requis'),
  validateProductCreation,
  handleValidationErrors,
  async (req, res) => {
    try {
      const { id } = req.params;
      const realId = securityService.decodeId(id);

      const updateData = {
        ...req.body,
        updatedBy: req.user.id,
        updatedAt: new Date().toISOString()
      };

      const product = await ProductsService.updateProduct(realId, updateData);

      console.log(`Admin ${req.user.email} updated product: ${realId}`);

      res.json({
```

```
        success: true,
        data: product,
        message: 'Produit mis à jour avec succès'
      });

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

      console.error('Error updating product:', error);
      res.status(500).json({
        success: false,
        error: 'Erreur lors de la mise à jour du produit'
      });
    }
  }
);

router.delete('/:id',
  authenticateToken,
  requireRole('admin'),
  param('id').notEmpty().withMessage('ID requis'),
  handleValidationErrors,
  async (req, res) => {
    try {
      const { id } = req.params;
      const realId = securityService.decodeId(id);

      await ProductsService.deleteProduct(realId);

      console.log(`Admin ${req.user.email} deleted product: ${realId}`);

      res.json({
        success: true,
        message: 'Produit supprimé avec succès'
      });

    } catch (error) {
      if (error.name === 'NotFoundError') {
        return res.status(404).json({
          success: false,
          error: 'Produit non trouvé'
```

```
        });
      }

      console.error('Error deleting product:', error);
      res.status(500).json({
        success: false,
        error: 'Erreur lors de la suppression du produit'
      });
    }
  }
);

module.exports = router;
```

2. Pattern de Service Métier Service avec Logique Métier :

```
// services/ProductsService.js
const fs = require('fs').promises;
const path = require('path');
const sharp = require('sharp');
const slugify = require('slugify');
const DatabaseService = require('../core/DatabaseService');
const CacheService = require('../core/CacheService');
const SecurityService = require('./SecurityService');

class ProductsService {
  constructor() {
    this.db = DatabaseService;
    this.cache = CacheService;
    this.security = SecurityService;
    this.uploadsDir = path.join(__dirname, '../uploads/products');
  }

  async getProducts({ page = 1, limit = 12, filters = {}, sort = { nom: 'asc' } } = {}) {
    // Clé de cache basée sur les paramètres
    const cacheKey = `products:${JSON.stringify({ page, limit, filters, sort })}`;

    // Vérifier le cache
    const cached = await this.cache.get(cacheKey);
    if (cached) {
      return cached;
    }

    try {
      let products = await this.db.readData('products.json');
```

```
    // Application des filtres
    products = this.applyFilters(products, filters);

    // Tri
    products = this.applySorting(products, sort);

    // Pagination
    const total = products.length;
    const startIndex = (page - 1) * limit;
    const endIndex = startIndex + limit;
    const paginatedProducts = products.slice(startIndex, endIndex);

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

    const result = {
      products: secureProducts,
      pagination: {
        total,
        page,
        limit,
        totalPages: Math.ceil(total / limit),
        hasNext: endIndex < total,
        hasPrev: startIndex > 0
      }
    };

    // Mettre en cache (5 minutes)
    await this.cache.set(cacheKey, result, 300);

    return result;

  } catch (error) {
    console.error('Error in getProducts:', error);
    throw new Error('Erreur lors de la récupération des produits');
  }
}

async getProductById(id) {
  const cacheKey = `product:${id}`;

  const cached = await this.cache.get(cacheKey);
  if (cached) {
```

```
    return cached;
  }

  try {
    const products = await this.db.readData('products.json');
    const product = products.find(p => p.id === parseInt(id));

    if (!product) {
      const error = new Error('Produit non trouvé');
      error.name = 'NotFoundError';
      throw error;
    }

    // Enrichir avec des données calculées
    const enrichedProduct = {
      ...product,
      slug: slugify(product.nom, { lower: true }),
      isInStock: product.stock > 0,
      stockStatus: this.getStockStatus(product.stock),
      priceWithTax: this.calculatePriceWithTax(product.prix),
      averageRating: await this.calculateAverageRating(product.id),
      reviewCount: await this.getReviewCount(product.id)
    };

    // Mettre en cache (10 minutes)
    await this.cache.set(cacheKey, enrichedProduct, 600);

    return enrichedProduct;

  } catch (error) {
    if (error.name === 'NotFoundError') {
      throw error;
    }
    console.error('Error in getProductById:', error);
    throw new Error('Erreur lors de la récupération du produit');
  }
}

async createProduct(productData) {
  try {
    const products = await this.db.readData('products.json');

    // Validation métier
    await this.validateProductData(productData);

    // Génération des données automatiques
```

```
    const newProduct = {
      id: Math.max(...products.map(p => p.id), 0) + 1,
      ...productData,
      slug: slugify(productData.nom, { lower: true }),
      sku: await this.generateSKU(productData),
      createdAt: new Date().toISOString(),
      updatedAt: new Date().toISOString(),
      views: 0,
      isActive: true
    };

    // Traitement des images
    if (productData.images && productData.images.length > 0) {
      newProduct.images = await this.processProductImages(
        productData.images,
        newProduct.id
      );
    }

    // Sauvegarde
    products.push(newProduct);
    await this.db.writeData('products.json', products);

    // Invalidation du cache
    await this.cache.invalidatePattern('products:');

    // Log de l'action
    console.log(`Product created: ${newProduct.id} - ${newProduct.nom}`);

    return newProduct;

  } catch (error) {
    console.error('Error in createProduct:', error);

    if (error.name === 'ValidationError') {
      throw error;
    }

    throw new Error('Erreur lors de la création du produit');
  }
}

async updateProduct(id, updateData) {
  try {
    const products = await this.db.readData('products.json');
    const productIndex = products.findIndex(p => p.id === parseInt(id));
```

```
    if (productIndex === -1) {
      const error = new Error('Produit non trouvé');
      error.name = 'NotFoundError';
      throw error;
    }

    // Validation des données de mise à jour
    await this.validateProductData(updateData, true);

    // Mise à jour
    const updatedProduct = {
      ...products[productIndex],
      ...updateData,
      id: products[productIndex].id, // Préserver l'ID
      updatedAt: new Date().toISOString()
    };

    // Mise à jour du slug si le nom a changé
    if (updateData.nom && updateData.nom !== products[productIndex].nom) {
      updatedProduct.slug = slugify(updateData.nom, { lower: true });
    }

    products[productIndex] = updatedProduct;
    await this.db.writeData('products.json', products);

    // Invalidation du cache
    await this.cache.invalidatePattern('products:');
    await this.cache.delete(`product:${id}`);

    console.log(`Product updated: ${id} - ${updatedProduct.nom}`);

    return updatedProduct;

  } catch (error) {
    if (error.name === 'NotFoundError' || error.name === 'ValidationError') {
      throw error;
    }

    console.error('Error in updateProduct:', error);
    throw new Error('Erreur lors de la mise à jour du produit');
  }
}

async deleteProduct(id) {
  try {
```

```
    const products = await this.db.readData('products.json');
    const productIndex = products.findIndex(p => p.id === parseInt(id));

    if (productIndex === -1) {
      const error = new Error('Produit non trouvé');
      error.name = 'NotFoundError';
      throw error;
    }

    const product = products[productIndex];

    // Supprimer les images associées
    if (product.images && product.images.length > 0) {
      await this.deleteProductImages(product.images);
    }

    // Supprimer le produit
    products.splice(productIndex, 1);
    await this.db.writeData('products.json', products);

    // Invalidation du cache
    await this.cache.invalidatePattern('products:');
    await this.cache.delete(`product:${id}`);

    console.log(`Product deleted: ${id} - ${product.nom}`);

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

  } catch (error) {
    if (error.name === 'NotFoundError') {
      throw error;
    }

    console.error('Error in deleteProduct:', error);
    throw new Error('Erreur lors de la suppression du produit');
  }
}

// Méthodes utilitaires privées

applyFilters(products, filters) {
  return products.filter(product => {
    // Filtre par catégorie
    if (filters.category && product.category !== filters.category) {
      return false;
    }
```

```
    // Filtre par prix
    if (filters.minPrice && product.prix < filters.minPrice) {
      return false;
    }
    if (filters.maxPrice && product.prix > filters.maxPrice) {
      return false;
    }

    // Filtre par stock
    if (filters.inStock && product.stock <= 0) {
      return false;
    }

    // Recherche textuelle
    if (filters.search) {
      const searchLower = filters.search.toLowerCase();
      return (
        product.nom.toLowerCase().includes(searchLower) ||
        product.description?.toLowerCase().includes(searchLower) ||
        product.category.toLowerCase().includes(searchLower)
      );
    }

    return true;
  });
}

applySorting(products, sort) {
  const [field, order] = Object.entries(sort)[0];

  return products.sort((a, b) => {
    let aVal = a[field];
    let bVal = b[field];

    // Gestion des types
    if (typeof aVal === 'string') {
      aVal = aVal.toLowerCase();
      bVal = bVal.toLowerCase();
    }

    // Tri
    let comparison = 0;
    if (aVal > bVal) comparison = 1;
    else if (aVal < bVal) comparison = -1;
```

```
    return order === 'desc' ? -comparison : comparison;
  });
}

async processProductImages(images, productId) {
  const processedImages = [];

  for (let i = 0; i < images.length; i++) {
    const image = images[i];
    const filename = `product-${productId}-${i + 1}-${Date.now()}.webp`;
    const filepath = path.join(this.uploadsDir, filename);

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

    processedImages.push({
      url: `/uploads/products/${filename}`,
      alt: image.alt || `Produit ${productId} - Image ${i + 1}`,
      size: (await fs.stat(filepath)).size,
      width: 800,
      height: 600
    });
  }

  return processedImages;
}

async deleteProductImages(images) {
  for (const image of images) {
    try {
      const imagePath = path.join(__dirname, '..', image.url);
      await fs.unlink(imagePath);
    } catch (error) {
      console.warn(`Could not delete image ${image.url}:`, error.message);
    }
  }
}

async validateProductData(data, isUpdate = false) {
  const errors = [];
```

```
    // Validation nom
    if (!isUpdate || data.nom !== undefined) {
      if (!data.nom || data.nom.length < 3) {
        errors.push('Le nom doit faire au moins 3 caractères');
      }
      if (data.nom && data.nom.length > 100) {
        errors.push('Le nom ne peut pas dépasser 100 caractères');
      }
    }

    // Validation prix
    if (!isUpdate || data.prix !== undefined) {
      if (!data.prix || data.prix <= 0) {
        errors.push('Le prix doit être positif');
      }
      if (data.prix && data.prix > 99999) {
        errors.push('Le prix ne peut pas dépasser 99999€');
      }
    }

    // Validation stock
    if (!isUpdate || data.stock !== undefined) {
      if (data.stock < 0) {
        errors.push('Le stock ne peut pas être négatif');
      }
    }

    // Validation catégorie
    if (!isUpdate || data.category !== undefined) {
      const categories = await this.db.readData('categories.json');
      const categoryExists = categories.some(cat => cat.id === data.category);
      if (!categoryExists) {
        errors.push('La catégorie spécifiée n\'existe pas');
      }
    }

    if (errors.length > 0) {
      const error = new Error('Données de produit invalides');
      error.name = 'ValidationError';
      error.details = errors;
      throw error;
    }
  }

  async generateSKU(productData) {
```

```
  const prefix = productData.category.substring(0, 3).toUpperCase();
  const timestamp = Date.now().toString().slice(-6);
  const random = Math.random().toString(36).substring(2, 5).toUpperCase();

  return `${prefix}-${timestamp}-${random}`;
}

getStockStatus(stock) {
  if (stock <= 0) return 'out_of_stock';
  if (stock <= 5) return 'low_stock';
  if (stock <= 20) return 'medium_stock';
  return 'in_stock';
}

calculatePriceWithTax(price, taxRate = 0.20) {
  return Math.round((price * (1 + taxRate)) * 100) / 100;
}

async calculateAverageRating(productId) {
  // Implémentation du calcul de la note moyenne
  try {
    const reviews = await this.db.readData('reviews.json');
    const productReviews = reviews.filter(review => review.productId === productId);

    if (productReviews.length === 0) return 0;

    const totalRating = productReviews.reduce((sum, review) => sum + review.rating, 0);
    return Math.round((totalRating / productReviews.length) * 10) / 10;
  } catch (error) {
    console.error('Error calculating average rating:', error);
    return 0;
  }
}

async getReviewCount(productId) {
  try {
    const reviews = await this.db.readData('reviews.json');
    return reviews.filter(review => review.productId === productId).length;
  } catch (error) {
    console.error('Error getting review count:', error);
    return 0;
  }
}

async incrementViews(productId) {
  try {
```

```
      const products = await this.db.readData('products.json');
      const product = products.find(p => p.id === parseInt(productId));

      if (product) {
        product.views = (product.views || 0) + 1;
        await this.db.writeData('products.json', products);

        // Invalider le cache du produit
        await this.cache.delete(`product:${productId}`);
      }
    } catch (error) {
      console.error('Error incrementing views:', error);
      // Ne pas faire échouer la requête pour cette erreur
    }
  }
}

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

Ce manuel fournit une base complète pour le développement et la maintenance de Riziky-Boutic. Il couvre les aspects techniques essentiels tout en maintenant les bonnes pratiques de développement moderne.