Manuel Développeur Complet - Riziky-Boutic

Guide Pratique pour Développeurs

Ce manuel s'adresse aux développeurs qui rejoignent l'équipe ou qui doivent maintenir, déboguer, étendre ou refactoriser la plateforme Riziky-Boutic. Il fournit toutes les informations pratiques pour être opérationnel rapidement.

Onboarding Développeur - Mise en Route Rapide

Prérequis Techniques

```
# Versions minimales requises
Node.js >= 18.0.0
npm >= 9.0.0
Git >= 2.30.0

# Éditeur recommandé avec extensions
Visual Studio Code + Extensions:
- ES7+ React/Redux/React-Native snippets
- TypeScript Importer
- Tailwind CSS IntelliSense
- Auto Rename Tag
- Bracket Pair Colorizer
- GitLens
- Thunder Client (pour tests API)
```

Installation Environnement Développement

```
# 1. Clone du projet
git clone [URL_DEPOT]
cd riziky-boutic

# 2. Installation dépendances frontend
npm install

# 3. Installation dépendances backend
cd server
npm install
cd ..

# 4. Configuration variables d'environnement
cp .env.example .env
# Éditer .env avec les valeurs appropriées
```

```
# 5. Démarrage en mode développement
# Terminal 1 (Backend)
cd server && npm run dev

# Terminal 2 (Frontend)
npm run dev

# 6. Vérification de l'installation
curl http://localhost:10000/api/health
# Doit retourner: {"status": "ok", "timestamp": "..."}
```

Structure du Workspace Recommandée

```
riziky-boutic/                  # Racine du projet
    .vscode/                # Configuration VS Code
        settings.json       # Paramètres éditeur
        extensions.json     # Extensions recommandées
        launch.json         # Configuration debug
    src/                    # Code frontend
    server/                 # Code backend
    docs/                   # Documentation
    package.json            # Dépendances frontend
    README.md               # Guide principal
```

Configuration VS Code Optimisée

```
// .vscode/settings.json
{
  "typescript.preferences.importModuleSpecifier": "relative",
  "typescript.suggest.autoImports": true,
  "editor.formatOnSave": true,
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true,
    "source.organizeImports": true
  },
  "tailwindCSS.includeLanguages": {
    "typescript": "javascript",
    "typescriptreact": "javascript"
  },
  "emmet.includeLanguages": {
    "typescript": "html",
    "typescriptreact": "html"
  },
  "files.associations": {
    "*.tsx": "typescriptreact"
  },
```

```
  "search.exclude": {
    "**/node_modules": true,
    "**/dist": true,
    "**/*.log": true
  }
}
```

Guide des Composants - Développement Pratique

Anatomie d'un Composant Type

```tsx
// Template de composant React optimisé
// src/components/example/ExampleComponent.tsx

import React, { useState, useEffect, useCallback, useMemo } from 'react'
import { cn } from '@/lib/utils'
import { Button } from '@/components/ui/button'
import { useToast } from '@/hooks/use-toast'

// 1. Interface TypeScript stricte
interface ExampleComponentProps {
  // Props obligatoires
  title: string
  data: ExampleData[]

  // Props optionnelles avec valeurs par défaut
  variant?: 'default' | 'compact' | 'expanded'
  showActions?: boolean
  maxItems?: number

  // Callbacks typées
  onItemClick?: (item: ExampleData) => void
  onActionComplete?: (result: ActionResult) => void

  // Props de style
  className?: string
}

// 2. Interface des données
interface ExampleData {
  id: string
  name: string
  description?: string
  status: 'active' | 'inactive'
  metadata?: Record<string, any>
```

```tsx
}

// 3. Composant avec toutes les bonnes pratiques
const ExampleComponent: React.FC<ExampleComponentProps> = ({
  title,
  data,
  variant = 'default',
  showActions = true,
  maxItems = 10,
  onItemClick,
  onActionComplete,
  className
}) => {
  // 4. États locaux
  const [loading, setLoading] = useState(false)
  const [selectedItems, setSelectedItems] = useState<Set<string>>(new Set())

  // 5. Hooks personnalisés
  const { toast } = useToast()

  // 6. Mémorisation des calculs coûteux
  const processedData = useMemo(() => {
    return data
      .filter(item => item.status === 'active')
      .slice(0, maxItems)
      .map(item => ({
        ...item,
        displayName: item.name.toUpperCase(),
        hasDescription: Boolean(item.description)
      }))
  }, [data, maxItems])

  // 7. Callbacks mémorisés
  const handleItemClick = useCallback((item: ExampleData) => {
    console.log(' Item cliqué:', item.name, { itemId: item.id })

    setSelectedItems(prev => {
      const newSet = new Set(prev)
      if (newSet.has(item.id)) {
        newSet.delete(item.id)
      } else {
        newSet.add(item.id)
      }
      return newSet
    })
```

```
  onItemClick?.(item)
}, [onItemClick])

const handleBulkAction = useCallback(async (action: string) => {
  setLoading(true)

  try {
    console.log('  Action en lot:', action, {
      selectedCount: selectedItems.size,
      selectedIds: Array.from(selectedItems)
    })

    // Simulation action asynchrone
    await new Promise(resolve => setTimeout(resolve, 1000))

    toast({
      title: "Action réussie",
      description: `${action} appliqué à ${selectedItems.size} élément(s)`
    })

    onActionComplete?.({ success: true, action, count: selectedItems.size })
    setSelectedItems(new Set())

  } catch (error) {
    console.error('  Erreur action en lot:', error)

    toast({
      variant: "destructive",
      title: "Erreur",
      description: "Impossible d'exécuter l'action"
    })

    onActionComplete?.({ success: false, action, error: error.message })

  } finally {
    setLoading(false)
  }
}, [selectedItems, toast, onActionComplete])

// 8. Effets avec nettoyage
useEffect(() => {
  console.log('  Données mises à jour:', {
    itemCount: data.length,
    processedCount: processedData.length
  })
```

```
  // Nettoyage des sélections si les données changent
  return () => {
    setSelectedItems(new Set())
  }
}, [data, processedData.length])

// 9. Rendu conditionnel et états de chargement
if (processedData.length === 0) {
  return (
    <div className={cn("text-center py-8", className)}>
      <p className="text-muted-foreground">Aucun élément à afficher</p>
    </div>
  )
}

return (
  <div className={cn(
    "space-y-4 p-4 border rounded-lg",
    {
      'default': 'bg-background',
      'compact': 'bg-muted/20 p-2',
      'expanded': 'bg-card p-6 shadow-lg'
    }[variant],
    className
  )}>
    {/* 10. Header avec actions */}
    <div className="flex items-center justify-between">
      <h2 className="text-lg font-semibold">
        {title} ({processedData.length})
      </h2>

      {showActions && selectedItems.size > 0 && (
        <div className="flex space-x-2">
          <Button
            variant="outline"
            size="sm"
            onClick={() => handleBulkAction('archive')}
            disabled={loading}
          >
            Archiver ({selectedItems.size})
          </Button>
          <Button
            variant="destructive"
            size="sm"
            onClick={() => handleBulkAction('delete')}
            disabled={loading}
```

```
            >
              Supprimer ({selectedItems.size})
            </Button>
          </div>
        )}
      </div>

      {/* 11. Liste des éléments avec rendu optimisé */}
      <div className="space-y-2">
        {processedData.map(item => (
          <ExampleItem
            key={item.id}
            item={item}
            selected={selectedItems.has(item.id)}
            onClick={() => handleItemClick(item)}
            variant={variant}
          />
        ))}
      </div>

      {/* 12. Indicateur de chargement */}
      {loading && (
        <div className="flex items-center justify-center py-4">
          <div className="animate-spin rounded-full h-6 w-6 border-b-2 border-primary"></div
          <span className="ml-2 text-sm text-muted-foreground">
            Traitement en cours...
          </span>
        </div>
      )}
    </div>
  )
}

// 13. Sous-composant optimisé
interface ExampleItemProps {
  item: ExampleData & { displayName: string; hasDescription: boolean }
  selected: boolean
  onClick: () => void
  variant: 'default' | 'compact' | 'expanded'
}

const ExampleItem = React.memo<ExampleItemProps>(({
  item,
  selected,
  onClick,
  variant
```

```
}) => {
  return (
    <div
      className={cn(
        "p-3 border rounded cursor-pointer transition-colors",
        selected ? "bg-primary/10 border-primary" : "hover:bg-muted/50",
        variant === 'compact' && "p-2 text-sm"
      )}
      onClick={onClick}
    >
      <div className="flex items-center justify-between">
        <div>
          <h4 className="font-medium">{item.displayName}</h4>
          {item.hasDescription && variant !== 'compact' && (
            <p className="text-sm text-muted-foreground mt-1">
              {item.description}
            </p>
          )}
        </div>

        <div className="flex items-center space-x-2">
          <span className={cn(
            "px-2 py-1 text-xs rounded-full",
            item.status === 'active'
              ? "bg-green-100 text-green-800"
              : "bg-gray-100 text-gray-800"
          )}>
            {item.status}
          </span>

          {selected && (
            <div className="w-4 h-4 rounded-full bg-primary flex items-center justify-center
              <div className="w-2 h-2 rounded-full bg-white"></div>
            </div>
          )}
        </div>
      </div>
    </div>
  )
}, (prevProps, nextProps) => {
  // Comparaison personnalisée pour éviter les re-renders inutiles
  return (
    prevProps.item.id === nextProps.item.id &&
    prevProps.selected === nextProps.selected &&
    prevProps.variant === nextProps.variant
  )
```

```
})

ExampleItem.displayName = 'ExampleItem'

export default ExampleComponent
export type { ExampleComponentProps, ExampleData }

// 14. Hooks personnalisés associés
// src/hooks/useExampleData.ts
export const useExampleData = (filters?: ExampleFilters) => {
  const [data, setData] = useState<ExampleData[]>([])
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState<string | null>(null)

  const fetchData = useCallback(async () => {
    setLoading(true)
    setError(null)

    try {
      const response = await api.get('/example', { params: filters })
      setData(response.data)
    } catch (err) {
      setError(err.message)
      console.error(' Erreur chargement données:', err)
    } finally {
      setLoading(false)
    }
  }, [filters])

  useEffect(() => {
    fetchData()
  }, [fetchData])

  return { data, loading, error, refetch: fetchData }
}
```

Patterns de Développement Récurrents

1. Composants de Formulaire Robustes

```
// Pattern pour formulaires avec validation
// src/components/forms/ProductForm.tsx

import { useForm } from 'react-hook-form'
import { zodResolver } from '@hookform/resolvers/zod'
import { z } from 'zod'
```

```tsx
// Schéma de validation Zod
const productSchema = z.object({
  nom: z.string()
    .min(2, "Le nom doit contenir au moins 2 caractères")
    .max(255, "Le nom ne peut dépasser 255 caractères"),

  description: z.string()
    .min(10, "La description doit contenir au moins 10 caractères")
    .max(2000, "La description ne peut dépasser 2000 caractères"),

  prix: z.number()
    .positive("Le prix doit être positif")
    .min(0.01, "Le prix minimum est 0.01€")
    .max(99999.99, "Le prix maximum est 99999.99€"),

  stock: z.number()
    .int("Le stock doit être un nombre entier")
    .min(0, "Le stock ne peut être négatif"),

  categories: z.array(z.string())
    .min(1, "Au moins une catégorie est requise")
    .max(5, "Maximum 5 catégories autorisées"),

  images: z.array(z.instanceof(File))
    .max(6, "Maximum 6 images autorisées")
    .optional(),

  promotion: z.object({
    type: z.enum(['percentage', 'fixed']),
    value: z.number().positive(),
    startDate: z.date(),
    endDate: z.date()
  }).optional().refine((promotion) => {
    if (!promotion) return true
    return promotion.endDate > promotion.startDate
  }, "La date de fin doit être après la date de début")
})

type ProductFormData = z.infer<typeof productSchema>

interface ProductFormProps {
  initialData?: Partial<ProductFormData>
  onSubmit: (data: ProductFormData) => Promise<void>
  isEditing?: boolean
}
```

```tsx
const ProductForm: React.FC<ProductFormProps> = ({
  initialData,
  onSubmit,
  isEditing = false
}) => {
  const [isSubmitting, setIsSubmitting] = useState(false)
  const { toast } = useToast()

  // Configuration du formulaire
  const form = useForm<ProductFormData>({
    resolver: zodResolver(productSchema),
    defaultValues: {
      nom: initialData?.nom || '',
      description: initialData?.description || '',
      prix: initialData?.prix || 0,
      stock: initialData?.stock || 0,
      categories: initialData?.categories || [],
      images: [],
      promotion: initialData?.promotion || undefined
    }
  })

  // Soumission avec gestion d'erreurs complète
  const handleSubmit = async (data: ProductFormData) => {
    setIsSubmitting(true)

    try {
      console.log(' Soumission formulaire produit:', {
        action: isEditing ? 'update' : 'create',
        productName: data.nom,
        hasImages: (data.images?.length || 0) > 0
      })

      await onSubmit(data)

      toast({
        title: isEditing ? "Produit mis à jour" : "Produit créé",
        description: `${data.nom} a été ${isEditing ? 'mis à jour' : 'créé'} avec succès`
      })

      if (!isEditing) {
        form.reset() // Réinitialiser le formulaire après création
      }

    } catch (error) {
```

```
    console.error('  Erreur soumission produit:', error)

    // Gestion des erreurs spécifiques
    if (error.response?.status === 409) {
      form.setError('nom', {
        message: 'Un produit avec ce nom existe déjà'
      })
    } else if (error.response?.status === 400) {
      toast({
        variant: 'destructive',
        title: 'Données invalides',
        description: error.response.data.message || 'Vérifiez les informations saisies'
      })
    } else {
      toast({
        variant: 'destructive',
        title: 'Erreur',
        description: 'Une erreur inattendue s\'est produite'
      })
    }
  } finally {
    setIsSubmitting(false)
  }
}

return (
  <form onSubmit={form.handleSubmit(handleSubmit)} className="space-y-6">
    {/* Informations de base */}
    <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
      <FormField
        control={form.control}
        name="nom"
        render={({ field }) => (
          <FormItem>
            <FormLabel>Nom du produit *</FormLabel>
            <FormControl>
              <Input
                placeholder="Nom du produit"
                {...field}
                disabled={isSubmitting}
              />
            </FormControl>
            <FormMessage />
          </FormItem>
        )}
      />
```

```
  <FormField
    control={form.control}
    name="prix"
    render={({ field }) => (
      <FormItem>
        <FormLabel>Prix (€) *</FormLabel>
        <FormControl>
          <Input
            type="number"
            step="0.01"
            min="0.01"
            placeholder="0.00"
            {...field}
            onChange={(e) => field.onChange(parseFloat(e.target.value) || 0)}
            disabled={isSubmitting}
          />
        </FormControl>
        <FormMessage />
      </FormItem>
    )}
  />
</div>

{/* Description */}
<FormField
  control={form.control}
  name="description"
  render={({ field }) => (
    <FormItem>
      <FormLabel>Description *</FormLabel>
      <FormControl>
        <Textarea
          placeholder="Description détaillée du produit"
          rows={4}
          {...field}
          disabled={isSubmitting}
        />
      </FormControl>
      <FormDescription>
        Décrivez les caractéristiques principales du produit
      </FormDescription>
      <FormMessage />
    </FormItem>
  )}
/>
```

```
{/* Upload d'images avec prévisualisation */}
<FormField
  control={form.control}
  name="images"
  render={({ field }) => (
    <FormItem>
      <FormLabel>Images du produit</FormLabel>
      <FormControl>
        <ImageUploadZone
          value={field.value || []}
          onChange={field.onChange}
          maxFiles={6}
          acceptedTypes={['image/jpeg', 'image/png', 'image/webp']}
          maxSize={5 * 1024 * 1024} // 5MB
          disabled={isSubmitting}
        />
      </FormControl>
      <FormDescription>
        Ajoutez jusqu'à 6 images (JPEG, PNG, WebP - max 5MB chacune)
      </FormDescription>
      <FormMessage />
    </FormItem>
  )}
/>

{/* Actions */}
<div className="flex justify-end space-x-4 pt-6 border-t">
  <Button
    type="button"
    variant="outline"
    onClick={() => form.reset()}
    disabled={isSubmitting}
  >
    Réinitialiser
  </Button>

  <Button
    type="submit"
    disabled={isSubmitting || !form.formState.isValid}
  >
    {isSubmitting ? (
      <>
        <Loader2 className="mr-2 h-4 w-4 animate-spin" />
        {isEditing ? 'Mise à jour...' : 'Création...'}
      </>
```

```
          ) : (
            isEditing ? 'Mettre à jour' : 'Créer le produit'
          )}
        </Button>
      </div>
    </form>
  )
}

export default ProductForm
```

2. Hooks Personnalisés Avancés

```
// Hook pour gestion d'état complexe avec optimisations
// src/hooks/useOptimizedState.ts

import { useState, useCallback, useRef, useMemo } from 'react'

interface UseOptimizedStateOptions<T> {
  initialState: T
  validator?: (value: T) => boolean
  serializer?: {
    serialize: (value: T) => string
    deserialize: (value: string) => T
  }
  persistence?: {
    key: string
    storage: 'localStorage' | 'sessionStorage'
  }
}

export const useOptimizedState = <T>(options: UseOptimizedStateOptions<T>) => {
  const {
    initialState,
    validator,
    serializer,
    persistence
  } = options

  // État avec initialisation depuis le stockage si configuré
  const [state, setState] = useState<T>(() => {
    if (persistence && serializer) {
      try {
        const storage = persistence.storage === 'localStorage'
          ? localStorage
          : sessionStorage
```

```
      const stored = storage.getItem(persistence.key)
      if (stored) {
        const deserialized = serializer.deserialize(stored)
        return validator?.(deserialized) ? deserialized : initialState
      }
    } catch (error) {
      console.warn('Erreur lecture stockage:', error)
    }
  }
  return initialState
})

// Référence pour éviter les re-renders inutiles
const stateRef = useRef(state)
stateRef.current = state

// Setter optimisé avec validation et persistence
const setOptimizedState = useCallback((newValue: T | ((prev: T) => T)) => {
  setState(prevState => {
    const nextState = typeof newValue === 'function'
      ? (newValue as (prev: T) => T)(prevState)
      : newValue

    // Validation si configurée
    if (validator && !validator(nextState)) {
      console.warn('Valeur rejetée par le validator:', nextState)
      return prevState
    }

    // Persistence si configurée
    if (persistence && serializer) {
      try {
        const storage = persistence.storage === 'localStorage'
          ? localStorage
          : sessionStorage

        storage.setItem(persistence.key, serializer.serialize(nextState))
      } catch (error) {
        console.warn('Erreur sauvegarde stockage:', error)
      }
    }

    return nextState
  })
}, [validator, serializer, persistence])
```

```
  // Reset vers l'état initial
  const resetState = useCallback(() => {
    setOptimizedState(initialState)
  }, [initialState, setOptimizedState])

  // Getter pour accès direct sans re-render
  const getCurrentState = useCallback(() => stateRef.current, [])

  // Mémos pour dérivations d'état fréquentes
  const memoizedSelectors = useMemo(() => ({
    isEmpty: Array.isArray(state) ? state.length === 0 :
             typeof state === 'object' && state !== null ?
             Object.keys(state).length === 0 : !state,

    isDefault: JSON.stringify(state) === JSON.stringify(initialState),

    size: Array.isArray(state) ? state.length :
          typeof state === 'object' && state !== null ?
          Object.keys(state).length : 0
  }), [state, initialState])

  return {
    state,
    setState: setOptimizedState,
    resetState,
    getCurrentState,
    ...memoizedSelectors
  }
}

// Hook pour API calls avec cache et retry
// src/hooks/useApiCall.ts
interface UseApiCallOptions<T> {
  cacheTime?: number
  retryCount?: number
  retryDelay?: number
  onSuccess?: (data: T) => void
  onError?: (error: Error) => void
}

const cache = new Map<string, { data: any; timestamp: number }>()

export const useApiCall = <T>(
  key: string,
  apiCall: () => Promise<T>,
```

```
  options: UseApiCallOptions<T> = {}
) => {
  const {
    cacheTime = 5 * 60 * 1000, // 5 minutes
    retryCount = 3,
    retryDelay = 1000,
    onSuccess,
    onError
  } = options

  const [data, setData] = useState<T | null>(null)
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState<Error | null>(null)

  // Vérification du cache
  const getCachedData = useCallback(() => {
    const cached = cache.get(key)
    if (cached && Date.now() - cached.timestamp < cacheTime) {
      return cached.data as T
    }
    return null
  }, [key, cacheTime])

  // Appel API avec retry automatique
  const executeCall = useCallback(async (attemptCount = 0): Promise<void> => {
    setLoading(true)
    setError(null)

    try {
      // Vérification cache avant appel
      const cachedData = getCachedData()
      if (cachedData) {
        setData(cachedData)
        onSuccess?.(cachedData)
        return
      }

      console.log(`  API Call: ${key} (tentative ${attemptCount + 1})`)

      const result = await apiCall()

      // Mise en cache
      cache.set(key, {
        data: result,
        timestamp: Date.now()
      })
```

```
        setData(result)
        onSuccess?.(result)

        console.log(`✅ API Success: ${key}`)

      } catch (err) {
        console.error(`❌ API Error: ${key}`, err)

        // Retry automatique
        if (attemptCount < retryCount - 1) {
          setTimeout(() => {
            executeCall(attemptCount + 1)
          }, retryDelay * (attemptCount + 1)) // Backoff exponentiel
          return
        }

        const error = err instanceof Error ? err : new Error('API call failed')
        setError(error)
        onError?.(error)

      } finally {
        setLoading(false)
      }
    }, [key, apiCall, retryCount, retryDelay, onSuccess, onError, getCachedData])

    // Invalidation manuelle du cache
    const invalidateCache = useCallback(() => {
      cache.delete(key)
    }, [key])

    // Refetch manuel
    const refetch = useCallback(() => {
      invalidateCache()
      return executeCall()
    }, [executeCall, invalidateCache])

    return {
      data,
      loading,
      error,
      execute: executeCall,
      refetch,
      invalidateCache
    }
  }
```

```typescript
// Hook pour pagination avancée
// src/hooks/usePagination.ts
interface UsePaginationOptions {
  initialPage?: number
  initialPageSize?: number
  totalItems: number
  onPageChange?: (page: number) => void
}

export const usePagination = (options: UsePaginationOptions) => {
  const {
    initialPage = 1,
    initialPageSize = 20,
    totalItems,
    onPageChange
  } = options

  const [currentPage, setCurrentPage] = useState(initialPage)
  const [pageSize, setPageSize] = useState(initialPageSize)

  // Calculs dérivés mémorisés
  const paginationInfo = useMemo(() => {
    const totalPages = Math.ceil(totalItems / pageSize)
    const startIndex = (currentPage - 1) * pageSize
    const endIndex = Math.min(startIndex + pageSize, totalItems)
    const hasNextPage = currentPage < totalPages
    const hasPreviousPage = currentPage > 1

    return {
      totalPages,
      startIndex,
      endIndex,
      hasNextPage,
      hasPreviousPage,
      isFirstPage: currentPage === 1,
      isLastPage: currentPage === totalPages,
      itemsOnCurrentPage: endIndex - startIndex,
      displayStart: startIndex + 1,
      displayEnd: endIndex
    }
  }, [currentPage, pageSize, totalItems])

  // Navigation
  const goToPage = useCallback((page: number) => {
    if (page >= 1 && page <= paginationInfo.totalPages) {
```

```
    setCurrentPage(page)
    onPageChange?.(page)
  }
}, [paginationInfo.totalPages, onPageChange])

const nextPage = useCallback(() => {
  if (paginationInfo.hasNextPage) {
    goToPage(currentPage + 1)
  }
}, [currentPage, paginationInfo.hasNextPage, goToPage])

const previousPage = useCallback(() => {
  if (paginationInfo.hasPreviousPage) {
    goToPage(currentPage - 1)
  }
}, [currentPage, paginationInfo.hasPreviousPage, goToPage])

const goToFirst = useCallback(() => goToPage(1), [goToPage])
const goToLast = useCallback(() => goToPage(paginationInfo.totalPages), [goToPage, paginat

// Changement de taille de page
const changePageSize = useCallback((newSize: number) => {
  const newTotalPages = Math.ceil(totalItems / newSize)
  const newCurrentPage = Math.min(currentPage, newTotalPages)

  setPageSize(newSize)
  setCurrentPage(newCurrentPage)
  onPageChange?.(newCurrentPage)
}, [totalItems, currentPage, onPageChange])

// Génération des numéros de pages pour l'affichage
const getPageNumbers = useCallback((maxVisible: number = 5) => {
  const { totalPages } = paginationInfo

  if (totalPages <= maxVisible) {
    return Array.from({ length: totalPages }, (_, i) => i + 1)
  }

  const halfVisible = Math.floor(maxVisible / 2)
  let startPage = Math.max(1, currentPage - halfVisible)
  let endPage = Math.min(totalPages, currentPage + halfVisible)

  // Ajustement pour avoir toujours maxVisible pages si possible
  if (endPage - startPage + 1 < maxVisible) {
    if (startPage === 1) {
      endPage = Math.min(totalPages, startPage + maxVisible - 1)
```

```
      } else {
        startPage = Math.max(1, endPage - maxVisible + 1)
      }
    }

    return Array.from(
      { length: endPage - startPage + 1 },
      (_, i) => startPage + i
    )
  }, [currentPage, paginationInfo])

  return {
    currentPage,
    pageSize,
    ...paginationInfo,
    goToPage,
    nextPage,
    previousPage,
    goToFirst,
    goToLast,
    changePageSize,
    getPageNumbers
  }
}
```

Debug et Troubleshooting

Outils de Debug Intégrés

```
// Utilitaires de debug pour développement
// src/utils/debug.ts

class Debugger {
  private isEnabled: boolean
  private logs: Array<{
    timestamp: Date
    level: string
    message: string
    data?: any
  }> = []

  constructor() {
    this.isEnabled = process.env.NODE_ENV === 'development'
  }
```

```
// Logger avec contexte et couleurs
log(message: string, data?: any, level: 'info' | 'warn' | 'error' = 'info') {
  if (!this.isEnabled) return

  const logEntry = {
    timestamp: new Date(),
    level,
    message,
    data
  }

  this.logs.push(logEntry)

  const styles = {
    info: 'color: #2563eb; font-weight: bold;',
    warn: 'color: #d97706; font-weight: bold;',
    error: 'color: #dc2626; font-weight: bold;'
  }

  console.group(`%c[${level.toUpperCase()}] ${message}`, styles[level])

  if (data) {
    console.log('Data:', data)
  }

  console.log('Timestamp:', logEntry.timestamp.toISOString())
  console.trace('Stack trace')
  console.groupEnd()
}

// Performance timing
time(label: string) {
  if (!this.isEnabled) return
  console.time(`  ${label}`)
}

timeEnd(label: string) {
  if (!this.isEnabled) return
  console.timeEnd(`  ${label}`)
}

// Profiling de composants React
profileComponent<P>(Component: React.ComponentType<P>, name?: string) {
  if (!this.isEnabled) return Component

  const componentName = name || Component.displayName || Component.name || 'Anonymous'
```

```
  return React.memo<P>((props) => {
    const renderCount = useRef(0)
    renderCount.current++

    console.log(`  Render #${renderCount.current} - ${componentName}`, props)

    const startTime = performance.now()
    const result = Component(props)
    const endTime = performance.now()

    console.log(`  ${componentName} rendered in ${(endTime - startTime).toFixed(2)}ms`)

    return result
  }, (prevProps, nextProps) => {
    const isEqual = JSON.stringify(prevProps) === JSON.stringify(nextProps)
    console.log(`  ${componentName} props comparison:`, {
      isEqual,
      prevProps,
      nextProps
    })
    return isEqual
  })
}

// Analyse des re-renders
useRenderCount(componentName: string) {
  if (!this.isEnabled) return

  const renderCount = useRef(0)
  renderCount.current++

  console.log(`  ${componentName} - Render #${renderCount.current}`)

  useEffect(() => {
    console.log(`  ${componentName} - Effect triggered`)
  })
}

// Inspection d'état
inspectState<T>(state: T, label: string) {
  if (!this.isEnabled) return

  console.group(`  State Inspection: ${label}`)
  console.log('Current state:', state)
  console.log('State type:', typeof state)
```

```
    console.log('Is array:', Array.isArray(state))
    console.log('Keys:', typeof state === 'object' && state !== null ? Object.keys(state) :
    console.groupEnd()
  }

  // Export des logs pour analyse
  exportLogs() {
    if (!this.isEnabled) return

    const logsData = {
      exported: new Date().toISOString(),
      userAgent: navigator.userAgent,
      url: window.location.href,
      logs: this.logs
    }

    const dataStr = JSON.stringify(logsData, null, 2)
    const dataBlob = new Blob([dataStr], { type: 'application/json' })
    const url = URL.createObjectURL(dataBlob)

    const link = document.createElement('a')
    link.href = url
    link.download = `debug-logs-${Date.now()}.json`
    link.click()

    URL.revokeObjectURL(url)
  }
}

export const debugger = new Debugger()

// Hook pour debug de props
export const useDebugProps = <T>(props: T, componentName: string) => {
  const prevProps = useRef<T>()

  useEffect(() => {
    if (process.env.NODE_ENV === 'development') {
      if (prevProps.current) {
        const changedProps: Partial<T> = {}

        Object.keys(props as any).forEach(key => {
          if ((props as any)[key] !== (prevProps.current as any)[key]) {
            (changedProps as any)[key] = {
              from: (prevProps.current as any)[key],
              to: (props as any)[key]
            }
```

```
          }
        })

        if (Object.keys(changedProps).length > 0) {
          console.log(`  ${componentName} props changed:`, changedProps)
        }
      }

      prevProps.current = props
    }
  })
}

// Hook pour debug de performance
export const useDebugPerformance = (componentName: string) => {
  const renderCount = useRef(0)
  const startTime = useRef<number>()

  // Début du render
  startTime.current = performance.now()
  renderCount.current++

  useLayoutEffect(() => {
    // Fin du render
    const endTime = performance.now()
    const duration = endTime - (startTime.current || 0)

    if (duration > 16) { // Plus de 16ms = problématique pour 60fps
      console.warn(`  Slow render detected: ${componentName} took ${duration.toFixed(2)}ms`)
    } else {
      console.log(`  ${componentName} rendered in ${duration.toFixed(2)}ms (render #${render
    }
  })
}
```

Checklist de Debug Fréquents

```
// Problèmes fréquents et leurs solutions
// src/utils/troubleshooting.ts

export const troubleshootingGuide = {
  // Problèmes de state
  stateIssues: {
    "State ne se met pas à jour": [
      "Vérifier que setState est bien appelé",
      "Vérifier l'immutabilité (pas de mutation directe)",
```

```
    "Vérifier les dépendances des useEffect",
    "Utiliser la forme fonction de setState si basé sur l'état précédent"
  ],

  "Boucle infinie de re-renders": [
    "Vérifier les dépendances des useEffect",
    "Mémoriser les objets et fonctions avec useMemo/useCallback",
    "Éviter les objets/fonctions dans JSX",
    "Utiliser React.memo pour les composants enfants"
  ],

  "Props ne se mettent pas à jour": [
    "Vérifier que le parent passe bien les nouvelles props",
    "Vérifier React.memo et sa fonction de comparaison",
    "Vérifier l'immutabilité des données passées",
    "Utiliser React DevTools pour inspecter"
  ]
},

// Problèmes d'API
apiIssues: {
  "Requête bloquée par CORS": [
    "Vérifier la configuration CORS du serveur",
    "Vérifier l'URL de base de l'API",
    "Vérifier les headers de la requête",
    "Tester avec un client REST externe"
  ],

  "Token JWT expiré": [
    "Implémenter le refresh automatique des tokens",
    "Vérifier la durée de validité du token",
    "Gérer l'intercepteur Axios pour le refresh",
    "Rediriger vers login si refresh échoue"
  ],

  "Données non à jour": [
    "Vérifier le cache de React Query",
    "Forcer un refetch si nécessaire",
    "Vérifier les clés de cache",
    "Implémenter l'invalidation appropriée"
  ]
},

// Problèmes de performance
performanceIssues: {
  "Composant lent à rendre": [
```

```
      "Utiliser React DevTools Profiler",
      "Mémoriser les calculs coûteux avec useMemo",
      "Éviter les re-renders inutiles avec useCallback",
      "Diviser en composants plus petits",
      "Utiliser la virtualisation pour les longues listes"
    ],

    "Bundle trop volumineux": [
      "Analyser le bundle avec webpack-bundle-analyzer",
      "Implémenter le code splitting",
      "Lazy load les routes non critiques",
      "Optimiser les imports (tree shaking)",
      "Utiliser des alternatives plus légères"
    ]
  },

  // Problèmes TypeScript
  typescriptIssues: {
    "Type 'undefined' n'est pas assignable": [
      "Utiliser des optional chaining (?.)",
      "Ajouter des vérifications de nullité",
      "Utiliser des types union avec undefined",
      "Initialiser avec des valeurs par défaut"
    ],

    "Erreur sur les props de composant": [
      "Vérifier l'interface des props",
      "Utiliser React.ComponentProps si nécessaire",
      "Étendre les props natives avec &",
      "Utiliser Partial<> pour les props optionnelles"
    ]
  }
}

// Fonction d'auto-diagnostic
export const diagnoseIssue = (symptoms: string[]) => {
  const suggestions: string[] = []

  // Analyse des symptômes
  symptoms.forEach(symptom => {
    const lowerSymptom = symptom.toLowerCase()

    if (lowerSymptom.includes('render') || lowerSymptom.includes('re-render')) {
      suggestions.push(...troubleshootingGuide.stateIssues["Boucle infinie de re-renders"])
    }
```

```
    if (lowerSymptom.includes('cors') || lowerSymptom.includes('blocked')) {
      suggestions.push(...troubleshootingGuide.apiIssues["Requête bloquée par CORS"])
    }

    if (lowerSymptom.includes('slow') || lowerSymptom.includes('lent')) {
      suggestions.push(...troubleshootingGuide.performanceIssues["Composant lent à rendre"])
    }

    if (lowerSymptom.includes('undefined') || lowerSymptom.includes('null')) {
      suggestions.push(...troubleshootingGuide.typescriptIssues["Type 'undefined' n'est pas
    }
  })

  // Suppression des doublons
  return [...new Set(suggestions)]
}

// Exemple d'utilisation
console.log(diagnoseIssue([
  "Component re-renders infinitely",
  "Network request blocked by CORS"
]))
```

Tests et Qualité Code

```
// Configuration de tests robustes
// src/__tests__/utils/testUtils.tsx

import React from 'react'
import { render, RenderOptions } from '@testing-library/react'
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
import { BrowserRouter } from 'react-router-dom'
import { AuthProvider } from '@/contexts/AuthContext'
import { StoreProvider } from '@/contexts/StoreContext'

// Wrapper personnalisé pour les tests
const TestProviders: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const queryClient = new QueryClient({
    defaultOptions: {
      queries: { retry: false },
      mutations: { retry: false }
    }
  })

  return (
    <QueryClientProvider client={queryClient}>
```

```
      <BrowserRouter>
        <AuthProvider>
          <StoreProvider>
            {children}
          </StoreProvider>
        </AuthProvider>
      </BrowserRouter>
    </QueryClientProvider>
  )
}

// Fonction de rendu personnalisée
const customRender = (
  ui: React.ReactElement,
  options?: Omit<RenderOptions, 'wrapper'>
) => render(ui, { wrapper: TestProviders, ...options })

// Mocks pour les API
export const mockApiResponse = <T>(data: T) => {
  return Promise.resolve({
    data,
    status: 200,
    statusText: 'OK',
    headers: {},
    config: {}
  })
}

export const mockApiError = (status: number, message: string) => {
  const error = new Error(message) as any
  error.response = {
    status,
    data: { message },
    statusText: status === 404 ? 'Not Found' : 'Error'
  }
  return Promise.reject(error)
}

// Factory pour créer des données de test
export const createMockProduct = (overrides?: Partial<Product>): Product => ({
  id: '1',
  nom: 'Produit Test',
  description: 'Description du produit test',
  prix: 29.99,
  stock: 10,
  categories: ['electronique'],
```

```
  images: ['test-image.jpg'],
  slug: 'produit-test',
  status: 'active',
  createdAt: '2024-01-01T00:00:00Z',
  updatedAt: '2024-01-01T00:00:00Z',
  createdBy: 'admin',
  viewCount: 0,
  purchaseCount: 0,
  averageRating: 0,
  reviewCount: 0,
  mainImageIndex: 0,
  ...overrides
})

export const createMockUser = (overrides?: Partial<User>): User => ({
  id: '1',
  email: 'test@example.com',
  nom: 'Dupont',
  prenom: 'Jean',
  role: 'client',
  status: 'active',
  createdAt: '2024-01-01T00:00:00Z',
  updatedAt: '2024-01-01T00:00:00Z',
  loginCount: 1,
  ...overrides
})

// Utilitaires d'assertion personnalisés
export const expectToBeInDocument = (element: HTMLElement | null) => {
  expect(element).toBeInTheDocument()
}

export const expectToHaveClass = (element: HTMLElement | null, className: string) => {
  expect(element).toHaveClass(className)
}

export const expectToBeDisabled = (element: HTMLElement | null) => {
  expect(element).toBeDisabled()
}

// Re-export de tout ce dont on a besoin
export * from '@testing-library/react'
export * from '@testing-library/jest-dom'
export * from '@testing-library/user-event'
export { customRender as render }
```

```tsx
// Exemple de test complet
// src/components/__tests__/ProductCard.test.tsx
import { render, screen, fireEvent, waitFor } from '@/test-utils'
import { ProductCard } from '../ProductCard'
import { createMockProduct } from '@/test-utils'

describe('ProductCard', () => {
  const mockProduct = createMockProduct()
  const mockOnAddToCart = jest.fn()

  beforeEach(() => {
    jest.clearAllMocks()
  })

  it('affiche correctement les informations du produit', () => {
    render(
      <ProductCard
        product={mockProduct}
        onAddToCart={mockOnAddToCart}
      />
    )

    expect(screen.getByText(mockProduct.nom)).toBeInTheDocument()
    expect(screen.getByText(`${mockProduct.prix}€`)).toBeInTheDocument()
    expect(screen.getByAltText(mockProduct.nom)).toBeInTheDocument()
  })

  it('appelle onAddToCart avec les bons paramètres', async () => {
    render(
      <ProductCard
        product={mockProduct}
        onAddToCart={mockOnAddToCart}
      />
    )

    const addButton = screen.getByText('Ajouter au panier')
    fireEvent.click(addButton)

    await waitFor(() => {
      expect(mockOnAddToCart).toHaveBeenCalledWith(mockProduct.id, 1)
    })
  })

  it('affiche "Rupture" quand le stock est à 0', () => {
    const outOfStockProduct = createMockProduct({ stock: 0 })
```

```
    render(
      <ProductCard
        product={outOfStockProduct}
        onAddToCart={mockOnAddToCart}
      />
    )

    expect(screen.getByText('Rupture')).toBeInTheDocument()
    expect(screen.getByText('Indisponible')).toBeInTheDocument()
  })

  it('désactive le bouton quand isLoading est true', () => {
    render(
      <ProductCard
        product={mockProduct}
        onAddToCart={mockOnAddToCart}
        isLoading={true}
      />
    )

    const addButton = screen.getByText(/ajouter au panier/i)
    expect(addButton).toBeDisabled()
  })
})
```

Ce manuel développeur complet fournit tous les outils et pratiques nécessaires pour travailler efficacement sur la plateforme Riziky-Boutic, du développement au débogage en passant par les tests.