Guide Complet des Composants - Riziky-Boutic

Vue d'Ensemble des Composants

Ce document détaille tous les composants de l'application Riziky-Boutic, leur logique métier, leur utilisation, et leur personnalisation. Chaque composant est expliqué avec des exemples pratiques pour faciliter la maintenance et l'évolution.

Système de Design - Composants UI de Base

Composants Shadcn/UI Intégrés

1. `Button` (`src/components/ui/button.tsx`) **Objectif :** Bouton réutilisable avec variantes de style **Props Interface :**

```tsx
interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {
  variant?: 'default' | 'destructive' | 'outline' | 'secondary' | 'ghost' | 'link';
  size?: 'default' | 'sm' | 'lg' | 'icon';
  asChild?: boolean;
}
```

Utilisation :

```tsx
// Bouton primaire standard
<Button variant="default" size="lg">
  Ajouter au panier
</Button>

// Bouton destructif pour suppression
<Button variant="destructive" onClick={() => deleteItem(id)}>
  Supprimer
</Button>

// Bouton avec icône
<Button size="icon" variant="outline">
  <Heart className="h-4 w-4" />
</Button>
```

Personnalisation :

```tsx
// Variantes personnalisées dans button.tsx
const buttonVariants = cva(
  "inline-flex items-center justify-center rounded-md text-sm font-medium ring-offset-backgr
  {
    variants: {
      variant: {
        default: "bg-primary text-primary-foreground hover:bg-primary/90",
        // Ajouter nouvelle variante
```

```
        success: "bg-green-600 text-white hover:bg-green-700",
        warning: "bg-yellow-600 text-white hover:bg-yellow-700"
      }
    }
  }
);
```

2. `Input` (`src/components/ui/input.tsx`) **Objectif :** Champ de saisie standardisé **Utilisation :**

```
// Input basique
<Input
  type="email"
  placeholder="votre@email.com"
  value={email}
  onChange={(e) => setEmail(e.target.value)}
/>

// Input avec validation
<Input
  type="password"
  placeholder="Mot de passe"
  className={cn(
    "border-input",
    errors.password && "border-destructive"
  )}
/>
```

3. `Card` (`src/components/ui/card.tsx`) **Objectif :** Conteneur avec styles cohérents **Composants :**

```
// Structure complète
<Card>
  <CardHeader>
    <CardTitle>Titre de la carte</CardTitle>
    <CardDescription>Description optionnelle</CardDescription>
  </CardHeader>
  <CardContent>
    Contenu principal
  </CardContent>
  <CardFooter>
    <Button>Action</Button>
  </CardFooter>
</Card>
```

4. `Dialog` (`src/components/ui/dialog.tsx`) **Objectif :** Modales et dialogues **Utilisation :**

```
const [isOpen, setIsOpen] = useState(false);

<Dialog open={isOpen} onOpenChange={setIsOpen}>
  <DialogTrigger asChild>
    <Button>Ouvrir Modal</Button>
  </DialogTrigger>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Titre du dialogue</DialogTitle>
      <DialogDescription>
        Description du contenu
      </DialogDescription>
    </DialogHeader>
    <div className="py-4">
      Contenu du dialogue
    </div>
    <DialogFooter>
      <Button variant="outline" onClick={() => setIsOpen(false)}>
        Annuler
      </Button>
      <Button onClick={handleConfirm}>
        Confirmer
      </Button>
    </DialogFooter>
  </DialogContent>
</Dialog>
```

Composants E-commerce

1. `ProductCard` (`src/components/products/ProductCard.tsx`)

Objectif : Affichage d'un produit dans une grille ou liste **Logique Complète :**

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

const ProductCard: FC<ProductCardProps> = ({
  product,
```

```
  variant = 'default',
  showActions = true,
  showQuickView = true
}) => {
  const { addToCart, isLoading } = useCart();
  const { toggleFavorite, isFavorite } = useFavorites();

  // Calcul du prix avec promotion
  const originalPrice = product.prix;
  const discountedPrice = product.promotion
    ? originalPrice - (originalPrice * product.promotion.pourcentage / 100)
    : originalPrice;

  const hasDiscount = product.promotion && product.promotion.pourcentage > 0;

  // Gestion de l'ajout au panier
  const handleAddToCart = async (e: React.MouseEvent) => {
    e.preventDefault();
    e.stopPropagation();

    if (product.stock <= 0) {
      toast.error("Produit en rupture de stock");
      return;
    }

    try {
      await addToCart(product.id, 1);
      toast.success(`${product.nom} ajouté au panier`);
    } catch (error) {
      toast.error("Erreur lors de l'ajout au panier");
    }
  };

  // Gestion des favoris
  const handleToggleFavorite = async (e: React.MouseEvent) => {
    e.preventDefault();
    e.stopPropagation();

    try {
      await toggleFavorite(product.id);
      toast.success(
        isFavorite(product.id) ? "Retiré des favoris" : "Ajouté aux favoris"
      );
    } catch (error) {
      toast.error("Erreur lors de la modification des favoris");
    }
```

```
};

return (
  <Card className={cn(
    "group relative overflow-hidden transition-all duration-300",
    "hover:shadow-lg hover:shadow-primary/10 hover:-translate-y-1",
    variant === 'compact' && "max-w-sm",
    variant === 'detailed' && "max-w-md"
  )}>
    {/* Image du produit */}
    <div className="relative aspect-square overflow-hidden bg-gray-100">
      <img
        src={product.images?.[0] || '/placeholder.svg'}
        alt={product.nom}
        className={cn(
          "h-full w-full object-cover transition-transform duration-300",
          "group-hover:scale-105"
        )}
        loading="lazy"
      />

      {/* Badge de promotion */}
      {hasDiscount && (
        <Badge
          variant="destructive"
          className="absolute top-2 left-2 z-10"
        >
          -{product.promotion.pourcentage}%
        </Badge>
      )}

      {/* Badge de stock */}
      {product.stock <= 0 && (
        <Badge
          variant="secondary"
          className="absolute top-2 right-2 z-10"
        >
          Rupture
        </Badge>
      )}

      {/* Overlay d'actions */}
      {showActions && (
        <div className={cn(
          "absolute inset-0 bg-black/50 opacity-0 transition-opacity duration-300",
          "group-hover:opacity-100 flex items-center justify-center space-x-2"
```

```
    )}>
      <Button
        size="icon"
        variant="secondary"
        onClick={handleToggleFavorite}
        className="bg-white/90 hover:bg-white"
      >
        <Heart
          className={cn(
            "h-4 w-4",
            isFavorite(product.id) && "fill-red-500 text-red-500"
          )}
        />
      </Button>

      {showQuickView && (
        <Button
          size="icon"
          variant="secondary"
          onClick={() => openQuickView(product)}
          className="bg-white/90 hover:bg-white"
        >
          <Eye className="h-4 w-4" />
        </Button>
      )}
    </div>
  )}
</div>

{/* Contenu de la carte */}
<CardContent className="p-4">
  <div className="space-y-2">
    {/* Nom du produit */}
    <h3 className={cn(
      "font-semibold line-clamp-2 text-sm",
      variant === 'detailed' && "text-base"
    )}>
      {product.nom}
    </h3>

    {/* Description (variant detailed uniquement) */}
    {variant === 'detailed' && product.description && (
      <p className="text-sm text-muted-foreground line-clamp-2">
        {product.description}
      </p>
    )}
```

```
            {/* Prix */}
            <div className="flex items-center space-x-2">
              <span className="text-lg font-bold text-primary">
                {discountedPrice.toFixed(2)}€
              </span>
              {hasDiscount && (
                <span className="text-sm text-muted-foreground line-through">
                  {originalPrice.toFixed(2)}€
                </span>
              )}
            </div>

            {/* Stock disponible */}
            {variant === 'detailed' && (
              <div className="flex items-center space-x-1 text-sm text-muted-foreground">
                <Package className="h-3 w-3" />
                <span>Stock: {product.stock}</span>
              </div>
            )}
          </div>
        </CardContent>

        {/* Actions */}
        {showActions && (
          <CardFooter className="p-4 pt-0">
            <Button
              className="w-full"
              onClick={handleAddToCart}
              disabled={product.stock <= 0 || isLoading}
            >
              {isLoading ? (
                <Loader2 className="mr-2 h-4 w-4 animate-spin" />
              ) : (
                <ShoppingCart className="mr-2 h-4 w-4" />
              )}
              {product.stock <= 0 ? 'Indisponible' : 'Ajouter au panier'}
            </Button>
          </CardFooter>
        )}
      </Card>
    );
};
```

Utilisation :

```
// Grille de produits
```

```tsx
<ProductGrid>
  {products.map(product => (
    <ProductCard
      key={product.id}
      product={product}
      variant="default"
      showActions={true}
    />
  ))}
</ProductGrid>

// Version compacte pour sidebar
<ProductCard
  product={relatedProduct}
  variant="compact"
  showQuickView={false}
/>
```

Personnalisation :

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

2. `ProductGrid` (`src/components/products/ProductGrid.tsx`)

Objectif : Grille responsive pour afficher les produits **Logique :**

```tsx
interface ProductGridProps {
  products: Product[];
  isLoading?: boolean;
  variant?: 'grid' | 'list';
  columns?: {
    mobile?: number;
    tablet?: number;
    desktop?: number;
  };
}

const ProductGrid: FC<ProductGridProps> = ({
  products,
  isLoading = false,
```

```
  variant = 'grid',
  columns = { mobile: 1, tablet: 2, desktop: 4 }
}) => {
  const gridClasses = cn(
    "grid gap-6",
    variant === 'grid' && [
      `grid-cols-${columns.mobile}`,
      `sm:grid-cols-${columns.tablet}`,
      `lg:grid-cols-${columns.desktop}`
    ],
    variant === 'list' && "grid-cols-1 gap-4"
  );

  // État de chargement avec skeleton
  if (isLoading) {
    return (
      <div className={gridClasses}>
        {Array.from({ length: 8 }).map((_, index) => (
          <ProductCardSkeleton key={index} />
        ))}
      </div>
    );
  }

  // État vide
  if (!products.length) {
    return (
      <div className="text-center py-12">
        <Package className="mx-auto h-12 w-12 text-muted-foreground" />
        <h3 className="mt-4 text-lg font-semibold">Aucun produit trouvé</h3>
        <p className="mt-2 text-muted-foreground">
          Essayez de modifier vos filtres de recherche.
        </p>
      </div>
    );
  }

  return (
    <div className={gridClasses}>
      {products.map(product => (
        <ProductCard
          key={product.id}
          product={product}
          variant={variant === 'list' ? 'detailed' : 'default'}
        />
      ))}
```

```
    </div>
  );
};
```

3. `CartDrawer` (`src/components/cart/CartDrawer.tsx`)

Objectif : Panneau latéral pour la gestion du panier **Logique Complète :**

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

  const [isOpen, setIsOpen] = useState(false);

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

  // Frais de port gratuits au-dessus de 50€
  const freeShippingThreshold = 50;
  const shipping = subtotal >= freeShippingThreshold ? 0 : 5.99;
  const total = subtotal + shipping;

  // Montant restant pour la livraison gratuite
  const remainingForFreeShipping = Math.max(0, freeShippingThreshold - subtotal);

  // Gestion de la modification des quantités
  const handleQuantityChange = async (itemId: string, newQuantity: number) => {
    if (newQuantity === 0) {
      handleRemoveItem(itemId);
    } else {
      try {
        await updateQuantity(itemId, newQuantity);
      } catch (error) {
        toast.error("Erreur lors de la mise à jour");
      }
    }
  };
```

```
// Suppression d'un article
const handleRemoveItem = async (itemId: string) => {
  try {
    await removeFromCart(itemId);
    toast.success("Article retiré du panier");
  } catch (error) {
    toast.error("Erreur lors de la suppression");
  }
};

// Vider le panier
const handleClearCart = async () => {
  try {
    await clearCart();
    toast.success("Panier vidé");
  } catch (error) {
    toast.error("Erreur lors du vidage du panier");
  }
};

return (
  <Sheet open={isOpen} onOpenChange={setIsOpen}>
    <SheetTrigger asChild>
      <Button variant="outline" size="icon" className="relative">
        <ShoppingCart className="h-4 w-4" />
        {itemCount > 0 && (
          <Badge
            variant="destructive"
            className="absolute -top-2 -right-2 h-5 w-5 rounded-full p-0 text-xs"
          >
            {itemCount}
          </Badge>
        )}
      </Button>
    </SheetTrigger>

    <SheetContent className="flex flex-col w-full sm:max-w-lg">
      <SheetHeader>
        <SheetTitle>
          Panier ({itemCount} {itemCount <= 1 ? 'article' : 'articles'})
        </SheetTitle>
        {cart.length > 0 && (
          <Button
            variant="ghost"
            size="sm"
```

```
      onClick={handleClearCart}
      className="text-destructive hover:text-destructive"
    >
      Vider le panier
    </Button>
  )}
</SheetHeader>

{/* Contenu du panier */}
<div className="flex-1 overflow-y-auto">
  {cart.length === 0 ? (
    <EmptyCartMessage />
  ) : (
    <div className="space-y-4">
      {/* Message livraison gratuite */}
      {remainingForFreeShipping > 0 && (
        <Alert>
          <Truck className="h-4 w-4" />
          <AlertDescription>
            Ajoutez {remainingForFreeShipping.toFixed(2)}€ d'achats
            pour bénéficier de la livraison gratuite !
          </AlertDescription>
        </Alert>
      )}

      {/* Articles du panier */}
      {cart.map(item => (
        <CartItemCard
          key={item.id}
          item={item}
          onQuantityChange={handleQuantityChange}
          onRemove={handleRemoveItem}
          isLoading={isLoading}
        />
      ))}
    </div>
  )}
</div>

{/* Résumé et actions */}
{cart.length > 0 && (
  <div className="space-y-4 border-t pt-4">
    <CartSummary
      subtotal={subtotal}
      shipping={shipping}
      total={total}
```

```
            />

            <div className="space-y-2">
              <Button
                className="w-full"
                size="lg"
                onClick={() => {
                  setIsOpen(false);
                  navigate('/checkout');
                }}
              >
                Finaliser la commande
              </Button>

              <Button
                variant="outline"
                className="w-full"
                onClick={() => {
                  setIsOpen(false);
                  navigate('/cart');
                }}
              >
                Voir le panier détaillé
              </Button>
            </div>
          </div>
        )}
      </SheetContent>
    </Sheet>
  );
};
```

4. `CartItemCard` (`src/components/cart/CartItemCard.tsx`)

Objectif : Affichage d'un article dans le panier **Logique :**

```
interface CartItemCardProps {
  item: CartItem;
  onQuantityChange: (id: string, quantity: number) => void;
  onRemove: (id: string) => void;
  isLoading?: boolean;
  variant?: 'drawer' | 'page';
}

const CartItemCard: FC<CartItemCardProps> = ({
  item,
```

```
  onQuantityChange,
  onRemove,
  isLoading = false,
  variant = 'drawer'
}) => {
  const [localQuantity, setLocalQuantity] = useState(item.quantite);
  const [isUpdating, setIsUpdating] = useState(false);

  // Debounce pour éviter trop d'appels API
  const debouncedQuantityChange = useCallback(
    debounce((id: string, quantity: number) => {
      onQuantityChange(id, quantity);
      setIsUpdating(false);
    }, 500),
    [onQuantityChange]
  );

  // Gestion du changement de quantité
  const handleQuantityChange = (newQuantity: number) => {
    if (newQuantity < 0) return;
    if (newQuantity > item.stockDisponible) {
      toast.error(`Stock maximum: ${item.stockDisponible}`);
      return;
    }

    setLocalQuantity(newQuantity);
    setIsUpdating(true);
    debouncedQuantityChange(item.id, newQuantity);
  };

  // Confirmation de suppression
  const [showDeleteDialog, setShowDeleteDialog] = useState(false);

  const confirmRemove = () => {
    onRemove(item.id);
    setShowDeleteDialog(false);
  };

  const subtotal = item.prix * localQuantity;

  return (
    <>
      <Card className={cn(
        "overflow-hidden",
        variant === 'drawer' && "border-none shadow-none bg-transparent",
        isUpdating && "opacity-50"
```

```
)}>
  <div className="flex gap-4 p-4">
    {/* Image produit */}
    <div className={cn(
      "flex-shrink-0 rounded-md overflow-hidden bg-gray-100",
      variant === 'drawer' ? "w-16 h-16" : "w-20 h-20"
    )}>
      <img
        src={item.image || '/placeholder.svg'}
        alt={item.nom}
        className="w-full h-full object-cover"
      />
    </div>

    {/* Informations produit */}
    <div className="flex-1 min-w-0">
      <div className="flex justify-between items-start">
        <div className="space-y-1">
          <h4 className={cn(
            "font-medium line-clamp-2",
            variant === 'drawer' ? "text-sm" : "text-base"
          )}>
            {item.nom}
          </h4>

          {variant === 'page' && item.description && (
            <p className="text-sm text-muted-foreground line-clamp-1">
              {item.description}
            </p>
          )}

          <div className="flex items-center space-x-2">
            <span className="font-semibold">
              {item.prix.toFixed(2)}€
            </span>
            {item.prixOriginal && item.prixOriginal > item.prix && (
              <span className="text-sm text-muted-foreground line-through">
                {item.prixOriginal.toFixed(2)}€
              </span>
            )}
          </div>
        </div>

        {/* Bouton de suppression */}
        <Button
          variant="ghost"
```

```
    size="icon"
    onClick={() => setShowDeleteDialog(true)}
    className="flex-shrink-0 text-muted-foreground hover:text-destructive"
  >
    <Trash2 className="h-4 w-4" />
  </Button>
</div>

{/* Contrôles de quantité */}
<div className="flex items-center justify-between mt-3">
  <div className="flex items-center space-x-2">
    <Button
      variant="outline"
      size="icon"
      className="h-8 w-8"
      onClick={() => handleQuantityChange(localQuantity - 1)}
      disabled={localQuantity <= 1 || isLoading}
    >
      <Minus className="h-3 w-3" />
    </Button>

    <div className="w-12 text-center">
      <span className="font-medium">{localQuantity}</span>
      {isUpdating && (
        <Loader2 className="h-3 w-3 animate-spin mx-auto" />
      )}
    </div>

    <Button
      variant="outline"
      size="icon"
      className="h-8 w-8"
      onClick={() => handleQuantityChange(localQuantity + 1)}
      disabled={localQuantity >= item.stockDisponible || isLoading}
    >
      <Plus className="h-3 w-3" />
    </Button>
  </div>

  {/* Sous-total */}
  <div className="text-right">
    <div className="font-semibold">
      {subtotal.toFixed(2)}€
    </div>
    {item.stockDisponible < 10 && (
      <div className="text-xs text-orange-600">
```

```
                    Plus que {item.stockDisponible} en stock
                  </div>
                )}
              </div>
            </div>
          </div>
        </div>
      </Card>

      {/* Dialog de confirmation de suppression */}
      <AlertDialog open={showDeleteDialog} onOpenChange={setShowDeleteDialog}>
        <AlertDialogContent>
          <AlertDialogHeader>
            <AlertDialogTitle>Supprimer l'article</AlertDialogTitle>
            <AlertDialogDescription>
              Êtes-vous sûr de vouloir supprimer "{item.nom}" de votre panier ?
              Cette action est irréversible.
            </AlertDialogDescription>
          </AlertDialogHeader>
          <AlertDialogFooter>
            <AlertDialogCancel>Annuler</AlertDialogCancel>
            <AlertDialogAction
              onClick={confirmRemove}
              className="bg-destructive text-destructive-foreground hover:bg-destructive/90"
            >
              Supprimer
            </AlertDialogAction>
          </AlertDialogFooter>
        </AlertDialogContent>
      </AlertDialog>
    </>
  );
};
```

Composants d'Authentification

1. LoginForm (src/components/auth/LoginForm.tsx)

Objectif : Formulaire de connexion sécurisé **Logique :**

```
interface LoginFormData {
  email: string;
  password: string;
  rememberMe: boolean;
}
```

```tsx
const loginSchema = z.object({
  email: z.string().email("Adresse email invalide"),
  password: z.string().min(8, "Le mot de passe doit contenir au moins 8 caractères")
});

const LoginForm: FC = () => {
  const { login, isLoading } = useAuth();
  const navigate = useNavigate();
  const location = useLocation();

  const form = useForm<LoginFormData>({
    resolver: zodResolver(loginSchema),
    defaultValues: {
      email: "",
      password: "",
      rememberMe: false
    }
  });

  // Page de redirection après connexion
  const from = location.state?.from?.pathname || "/";

  const onSubmit = async (data: LoginFormData) => {
    try {
      await login(data.email, data.password, data.rememberMe);

      toast.success("Connexion réussie !");
      navigate(from, { replace: true });

    } catch (error: any) {
      // Gestion des erreurs spécifiques
      if (error.response?.status === 401) {
        form.setError("root", {
          message: "Email ou mot de passe incorrect"
        });
      } else if (error.response?.status === 429) {
        form.setError("root", {
          message: "Trop de tentatives. Veuillez réessayer plus tard."
        });
      } else {
        form.setError("root", {
          message: "Erreur de connexion. Veuillez réessayer."
        });
      }
    }
```

```
};

return (
  <Card className="w-full max-w-md mx-auto">
    <CardHeader>
      <CardTitle className="text-2xl text-center">Connexion</CardTitle>
      <CardDescription className="text-center">
        Connectez-vous à votre compte Riziky-Boutic
      </CardDescription>
    </CardHeader>

    <CardContent>
      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
          {/* Champ Email */}
          <FormField
            control={form.control}
            name="email"
            render={({ field }) => (
              <FormItem>
                <FormLabel>Email</FormLabel>
                <FormControl>
                  <Input
                    type="email"
                    placeholder="votre@email.com"
                    {...field}
                    disabled={isLoading}
                  />
                </FormControl>
                <FormMessage />
              </FormItem>
            )}
          />

          {/* Champ Mot de passe */}
          <FormField
            control={form.control}
            name="password"
            render={({ field }) => (
              <FormItem>
                <FormLabel>Mot de passe</FormLabel>
                <FormControl>
                  <PasswordInput
                    placeholder="Votre mot de passe"
                    {...field}
                    disabled={isLoading}
```

```
        />
      </FormControl>
      <FormMessage />
    </FormItem>
  )}
/>

{/* Se souvenir de moi */}
<FormField
  control={form.control}
  name="rememberMe"
  render={({ field }) => (
    <FormItem className="flex flex-row items-start space-x-3 space-y-0">
      <FormControl>
        <Checkbox
          checked={field.value}
          onCheckedChange={field.onChange}
          disabled={isLoading}
        />
      </FormControl>
      <div className="space-y-1 leading-none">
        <FormLabel>
          Se souvenir de moi
        </FormLabel>
      </div>
    </FormItem>
  )}
/>

{/* Erreur générale */}
{form.formState.errors.root && (
  <Alert variant="destructive">
    <AlertCircle className="h-4 w-4" />
    <AlertDescription>
      {form.formState.errors.root.message}
    </AlertDescription>
  </Alert>
)}

{/* Bouton de connexion */}
<Button
  type="submit"
  className="w-full"
  disabled={isLoading}
>
  {isLoading ? (
```

```
                <>
                  <Loader2 className="mr-2 h-4 w-4 animate-spin" />
                  Connexion en cours...
                </>
              ) : (
                "Se connecter"
              )}
            </Button>
          </form>
        </Form>

        {/* Liens additionnels */}
        <div className="mt-6 space-y-4">
          <div className="text-center">
            <Button variant="link" asChild>
              <Link to="/forgot-password">
                Mot de passe oublié ?
              </Link>
            </Button>
          </div>

          <div className="text-center text-sm">
            Pas encore de compte ?{" "}
            <Button variant="link" asChild className="p-0">
              <Link to="/register">
                S'inscrire
              </Link>
            </Button>
          </div>
        </div>
      </CardContent>
    </Card>
  );
};
```

2. `PasswordInput` (Composant personnalisé)

Objectif : Champ de mot de passe avec visibilité toggle **Logique :**

```
interface PasswordInputProps extends React.InputHTMLAttributes<HTMLInputElement> {
  showStrengthIndicator?: boolean;
}

const PasswordInput = React.forwardRef<HTMLInputElement, PasswordInputProps>(
  ({ showStrengthIndicator = false, ...props }, ref) => {
    const [showPassword, setShowPassword] = useState(false);
```

```
const [password, setPassword] = useState("");

// Calcul de la force du mot de passe
const calculateStrength = (pwd: string): number => {
  let score = 0;

  if (pwd.length >= 8) score += 1;
  if (pwd.length >= 12) score += 1;
  if (/[a-z]/.test(pwd)) score += 1;
  if (/[A-Z]/.test(pwd)) score += 1;
  if (/\d/.test(pwd)) score += 1;
  if (/[^A-Za-z0-9]/.test(pwd)) score += 1;

  return score;
};

const strength = calculateStrength(password);

const strengthColors = {
  0: 'bg-gray-200',
  1: 'bg-red-500',
  2: 'bg-orange-500',
  3: 'bg-yellow-500',
  4: 'bg-blue-500',
  5: 'bg-green-500',
  6: 'bg-green-600'
};

const strengthLabels = {
  0: '',
  1: 'Très faible',
  2: 'Faible',
  3: 'Moyen',
  4: 'Bon',
  5: 'Fort',
  6: 'Très fort'
};

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  setPassword(e.target.value);
  props.onChange?.(e);
};

return (
  <div className="space-y-2">
    <div className="relative">
```

```
            <Input
              {...props}
              ref={ref}
              type={showPassword ? "text" : "password"}
              onChange={handleChange}
              className="pr-10"
            />
            <Button
              type="button"
              variant="ghost"
              size="icon"
              className="absolute right-0 top-0 h-full px-3 py-2 hover:bg-transparent"
              onClick={() => setShowPassword(!showPassword)}
            >
              {showPassword ? (
                <EyeOff className="h-4 w-4 text-muted-foreground" />
              ) : (
                <Eye className="h-4 w-4 text-muted-foreground" />
              )}
            </Button>
          </div>

          {/* Indicateur de force */}
          {showStrengthIndicator && password && (
            <div className="space-y-2">
              <div className="flex space-x-1">
                {Array.from({ length: 6 }).map((_, index) => (
                  <div
                    key={index}
                    className={cn(
                      "h-1 flex-1 rounded-full transition-colors",
                      index < strength
                        ? strengthColors[strength as keyof typeof strengthColors]
                        : "bg-gray-200"
                    )}
                  />
                ))}
              </div>
              <p className="text-sm text-muted-foreground">
                Force du mot de passe: {strengthLabels[strength as keyof typeof strengthLabels
              </p>
            </div>
          )}
        </div>
      );
    }
```

```
);
```

Composants de Navigation

1. Navbar (`src/components/layout/Navbar.tsx`)

Objectif : Navigation principale adaptative **Logique Complète :**

```tsx
const Navbar: FC = () => {
  const { user, logout, isAuthenticated } = useAuth();
  const { itemCount } = useCart();
  const { favoriteCount } = useFavorites();
  const [isMobileMenuOpen, setIsMobileMenuOpen] = useState(false);

  // Éléments de navigation
  const navigationItems = [
    { label: 'Accueil', href: '/', icon: Home },
    { label: 'Produits', href: '/products', icon: Package },
    { label: 'Nouveautés', href: '/new-arrivals', icon: Sparkles },
    { label: 'Promotions', href: '/flash-sales', icon: Zap },
    { label: 'Contact', href: '/contact', icon: Phone }
  ];

  // Gestion de la déconnexion
  const handleLogout = async () => {
    try {
      await logout();
      toast.success("Déconnecté avec succès");
      navigate("/");
    } catch (error) {
      toast.error("Erreur lors de la déconnexion");
    }
  };

  return (
    <header className="sticky top-0 z-50 w-full border-b bg-background/95 backdrop-blur supp
      <div className="container flex h-16 items-center justify-between">
        {/* Logo */}
        <Link to="/" className="flex items-center space-x-2">
          <img
            src="/images/Logo/Logo.png"
            alt="Riziky-Boutic"
            className="h-8 w-auto"
          />
          <span className="hidden sm:inline-block font-bold text-xl text-primary">
```

```
    Riziky-Boutic
  </span>
</Link>

{/* Navigation Desktop */}
<nav className="hidden md:flex items-center space-x-6">
  {navigationItems.map((item) => (
    <Link
      key={item.href}
      to={item.href}
      className={cn(
        "flex items-center space-x-1 text-sm font-medium transition-colors hover:tex
        "relative after:absolute after:left-0 after:-bottom-1 after:h-0.5 after:w-0
      )}
    >
      <item.icon className="h-4 w-4" />
      <span>{item.label}</span>
    </Link>
  ))}
</nav>

{/* Actions utilisateur */}
<div className="flex items-center space-x-2">
  {/* Recherche */}
  <SearchDialog />

  {/* Favoris */}
  <Button variant="ghost" size="icon" asChild>
    <Link to="/favorites" className="relative">
      <Heart className="h-5 w-5" />
      {favoriteCount > 0 && (
        <Badge
          variant="destructive"
          className="absolute -top-1 -right-1 h-4 w-4 p-0 text-xs"
        >
          {favoriteCount}
        </Badge>
      )}
    </Link>
  </Button>

  {/* Panier */}
  <CartDrawer />

  {/* Menu utilisateur */}
  {isAuthenticated ? (
```

```
<DropdownMenu>
  <DropdownMenuTrigger asChild>
    <Button variant="ghost" size="icon">
      <UserAvatar user={user} size="sm" />
    </Button>
  </DropdownMenuTrigger>
  <DropdownMenuContent align="end" className="w-56">
    <DropdownMenuLabel>
      <div className="flex flex-col space-y-1">
        <p className="text-sm font-medium leading-none">
          {user?.prenom} {user?.nom}
        </p>
        <p className="text-xs leading-none text-muted-foreground">
          {user?.email}
        </p>
      </div>
    </DropdownMenuLabel>
    <DropdownMenuSeparator />

    <DropdownMenuItem asChild>
      <Link to="/profile">
        <User className="mr-2 h-4 w-4" />
        Mon profil
      </Link>
    </DropdownMenuItem>

    <DropdownMenuItem asChild>
      <Link to="/orders">
        <Package className="mr-2 h-4 w-4" />
        Mes commandes
      </Link>
    </DropdownMenuItem>

    <DropdownMenuItem asChild>
      <Link to="/favorites">
        <Heart className="mr-2 h-4 w-4" />
        Mes favoris
      </Link>
    </DropdownMenuItem>

    {user?.role === 'admin' && (
      <>
        <DropdownMenuSeparator />
        <DropdownMenuItem asChild>
          <Link to="/admin">
            <Shield className="mr-2 h-4 w-4" />
```

```
                    Administration
                  </Link>
                </DropdownMenuItem>
              </>
            )}

            <DropdownMenuSeparator />
            <DropdownMenuItem
              onClick={handleLogout}
              className="text-destructive focus:text-destructive"
            >
              <LogOut className="mr-2 h-4 w-4" />
              Se déconnecter
            </DropdownMenuItem>
          </DropdownMenuContent>
        </DropdownMenu>
      ) : (
        <div className="flex items-center space-x-2">
          <Button variant="ghost" size="sm" asChild>
            <Link to="/login">Connexion</Link>
          </Button>
          <Button size="sm" asChild>
            <Link to="/register">S'inscrire</Link>
          </Button>
        </div>
      )}

      {/* Menu mobile */}
      <Sheet open={isMobileMenuOpen} onOpenChange={setIsMobileMenuOpen}>
        <SheetTrigger asChild>
          <Button variant="ghost" size="icon" className="md:hidden">
            <Menu className="h-5 w-5" />
          </Button>
        </SheetTrigger>
        <MobileNavigation
          items={navigationItems}
          user={user}
          isAuthenticated={isAuthenticated}
          onLogout={handleLogout}
          onClose={() => setIsMobileMenuOpen(false)}
        />
      </Sheet>
    </div>
  </div>
</header>
);
```

```
};
```

Ce guide détaille les composants principaux de Riziky-Boutic avec leur logique métier complète, leur utilisation, et les possibilités de personnalisation. Chaque composant est conçu pour être modulaire, réutilisable et facilement extensible.