BÁO CÁO TRANG THANH TOÁN

Table of Contents

Giới thiệu

Báo cáo này mô tả chi tiết về các component được sử dụng để xây dựng trang thanh toán cho dự án Modern Footwear E-commerce Platform. Trang thanh toán bao gồm nhiều component phức tạp, đảm bảo trải nghiệm mượt mà cho người dùng thông qua quy trình thanh toán và xử lý đơn hàng.

Cấu trúc tổng quan

Trang thanh toán được xây dựng bao gồm các thành phần chính:

1. **StepIndicator**: Hiển thị các bước trong quy trình thanh toán
2. **ShippingForm**: Form nhập thông tin giao hàng
3. **PaymentSelection**: Lựa chọn phương thức thanh toán
4. **OrderConfirmation**: Xác nhận đơn hàng trước khi đặt hàng
5. **CheckoutSummary**: Hiển thị tóm tắt đơn hàng và form áp dụng mã giảm giá
6. **DiscountForm**: Component con để áp dụng mã giảm giá

Các component này được tích hợp trong trang chính `app/checkout/page.tsx` và được hiển thị tùy thuộc vào bước hiện tại của quy trình thanh toán.

Kiểu dữ liệu và Định nghĩa

Định nghĩa kiểu trong checkout.ts

```
// Thông tin thanh toán
export interface ShippingInfo {
  fullName: string;
  email: string;
  phone: string;
  address: string;
  city: string;
  province: string;
  postalCode: string;
  notes?: string;
}

// Ph□□ng th□c thanh toán
export type PaymentMethod = 'stripe' | 'cash_on_delivery';

// Thông tin mã gi□m giá
export interface DiscountCode {
  code: string;
  discountAmount: number;
  discountType: 'percentage' | 'fixed';
  isValid: boolean;
}

// Tr□ng thái thanh toán
export interface CheckoutState {
  shippingInfo: ShippingInfo;
  paymentMethod: PaymentMethod | null;
  currentStep: 'shipping' | 'payment' | 'confirmation';
  discountCode: DiscountCode | null;
}

// Thông tin đ□n hàng đ□y đ□
export interface OrderSummary {
  items: {
    id: string;
    name: string;
    quantity: number;
    price: number;
    size?: string;
  }[];
  subtotal: number;
  shipping: number;
  tax: number;
  discount: number;
  total: number;
```

```
}
```

Giải thích:

- `ShippingInfo`: Interface này định nghĩa thông tin giao hàng bao gồm họ tên, email, số điện thoại, địa chỉ, thành phố, tỉnh/thành phố, mã bưu điện và ghi chú (tùy chọn). Mỗi trường đều có kiểu dữ liệu là `string`, riêng `notes` được đánh dấu là tùy chọn với dấu `?`.
- `PaymentMethod`: Đây là kiểu union type định nghĩa hai phương thức thanh toán có sẵn trong hệ thống: 'stripe' (thanh toán trực tuyến) và 'cash_on_delivery' (thanh toán khi nhận hàng).
- `DiscountCode`: Interface này định nghĩa cấu trúc của một mã giảm giá, bao gồm:
  - `code`: Mã giảm giá dưới dạng chuỗi
  - `discountAmount`: Số tiền hoặc phần trăm giảm giá
  - `discountType`: Kiểu giảm giá có thể là 'percentage' (theo phần trăm) hoặc 'fixed' (số tiền cố định)
  - `isValid`: Boolean xác định mã giảm giá có hợp lệ hay không
- `CheckoutState`: Interface này mô tả trạng thái của quy trình thanh toán, gồm thông tin:
  - `shippingInfo`: Thông tin giao hàng
  - `paymentMethod`: Phương thức thanh toán đã chọn hoặc `null` nếu chưa chọn
  - `currentStep`: Bước hiện tại trong quy trình thanh toán ('shipping', 'payment', hoặc 'confirmation')
  - `discountCode`: Mã giảm giá đã áp dụng hoặc `null`
- `OrderSummary`: Interface này định nghĩa cấu trúc tóm tắt đơn hàng, gồm:
  - `items`: Mảng các sản phẩm trong đơn hàng với thông tin id, tên, số lượng, giá và kích cỡ
  - `subtotal`: Tổng tiền tạm tính trước khi áp dụng phí, thuế và giảm giá
  - `shipping`: Phí vận chuyển
  - `tax`: Thuế
  - `discount`: Số tiền giảm giá
  - `total`: Tổng tiền cần thanh toán sau khi tính tất cả các khoản

```
```

Schema xác thực trong form-validation.ts

```
export const shippingSchema = z.object({
  fullName: z
    .string()
    .min(2, { message: 'H tên ph i có ít nh t 2 ký t ' })
    .max(100, { message: 'H tên không đ c quá 100 ký t ' }),
  email: z
    .string()
```

```
    .email({ message: 'Đ□a ch□ email không h□p l□' }),
  phone: z
    .string()
    .min(10, { message: 'S□ đi□n tho□i không h□p l□' })
    .max(15, { message: 'S□ đi□n tho□i không h□p l□' }),
  address: z
    .string()
    .min(5, { message: 'Đ□a ch□ ph□i có ít nh□t 5 ký t□' }),
  city: z
    .string()
    .min(2, { message: 'Thành ph□ không đ□□c đ□ tr□ng' }),
  province: z
    .string()
    .min(1, { message: 'Vui lòng ch□n t□nh/thành ph□' }),
  postalCode: z
    .string()
    .min(5, { message: 'Mã b□u đi□n không h□p l□' })
    .max(10, { message: 'Mã b□u đi□n không h□p l□' }),
  notes: z.string().optional(),
});

export const paymentSchema = z.object({
  paymentMethod: z
    .enum(['stripe', 'cash_on_delivery'], {
      required_error: 'Vui lòng ch□n ph□□ng th□c thanh toán'
    }),
  cardNumber: z
    .string()
    .min(16, { message: 'S□ th□ không h□p l□' })
    .max(19, { message: 'S□ th□ không h□p l□' })
    .optional()
    .refine((val) => val !== undefined && val.length >= 16, {
      message: "S□ th□ không h□p l□",
      path: ["cardNumber"]
    }),
  cardName: z.string().min(2, { message: 'Tên trên th□ không h□p l□' }).optional(),
  expDate: z.string().min(4, { message: 'Ngày h□t h□n không h□p l□' }).optional(),
  cvv: z.string().min(3, { message: 'CVV không h□p l□' }).optional(),
});

export type ShippingFormValues = z.infer<typeof shippingSchema>;
export type PaymentFormValues = z.infer<typeof paymentSchema>;
```

Giải thích:

- `shippingSchema`: Định nghĩa schema xác thực cho form thông tin giao hàng sử dụng thư viện Zod.
  - `fullName`: Yêu cầu họ tên có độ dài từ 2-100 ký tự, với thông báo lỗi cụ thể
  - `email`: Xác thực định dạng email hợp lệ
  - `phone`: Yêu cầu số điện thoại có độ dài từ 10-15 ký tự

    - `address`: Yêu cầu địa chỉ có ít nhất 5 ký tự
    - `city`: Yêu cầu thành phố có ít nhất 2 ký tự
    - `province`: Yêu cầu tỉnh/thành phố được chọn (có ít nhất 1 ký tự)
    - `postalCode`: Yêu cầu mã bưu điện có độ dài từ 5-10 ký tự
    - `notes`: Trường tùy chọn, có thể bỏ trống
- `paymentSchema`: Định nghĩa schema xác thực cho form phương thức thanh toán.
    - `paymentMethod`: Enum giới hạn các giá trị hợp lệ là 'stripe' hoặc 'cash_on_delivery', kèm thông báo lỗi khi không chọn
    - `cardNumber`: Trường tùy chọn, nhưng nếu được nhập thì phải có độ dài từ 16-19 ký tự. Thêm `.refine()` để xác thực nâng cao
    - `cardName`: Tên trên thẻ (tùy chọn), phải có ít nhất 2 ký tự nếu được nhập
    - `expDate`: Ngày hết hạn (tùy chọn), phải có ít nhất 4 ký tự nếu được nhập
    - `cvv`: Mã CVV (tùy chọn), phải có ít nhất 3 ký tự nếu được nhập
- Cuối cùng, chúng ta sử dụng `z.infer<>` để tạo ra các kiểu TypeScript từ schema Zod, đảm bảo sự đồng bộ giữa xác thực form và kiểu dữ liệu TypeScript.

```
```

Component trang thanh toán chính (checkout/page.tsx)

Imports

```
import React, { useState, useEffect, useMemo } from 'react';
import { useNavigate } from 'react-router-dom';
import { ShippingForm } from '@/components/checkout/ShippingForm';
import { PaymentSelection } from '@/components/checkout/PaymentSelection';
import { OrderConfirmation } from '@/components/checkout/OrderConfirmation';
import CheckoutSummary from '@/components/checkout/CheckoutSummary';
import { ShippingInfo, PaymentMethod, CheckoutState, OrderSummary, DiscountCode } from
'@/types/checkout';
import { CartItem } from '@/types/cart';
import { StepIndicator, CheckoutStep } from '@/components/checkout/StepIndicator';
import { OrderProcessingSkeleton } from '@/components/ui/skeleton';
import { motion, AnimatePresence } from 'framer-motion';
```

Trạng thái và luồng xử lý

- **Khởi tạo trạng thái**: Sử dụng `useState` để quản lý các trạng thái của quy trình thanh toán
- **Điều hướng**: Sử dụng `useNavigate` để điều hướng người dùng sau khi đặt hàng thành công

- **Hiệu ứng loading**: Hiển thị skeleton loading khi xử lý đơn hàng
- **Quản lý mã giảm giá**: Cho phép người dùng áp dụng mã giảm giá trong quá trình thanh toán

Logic chính

Khởi tạo trạng thái thanh toán

```
// Khởi tạo state cho quy trình thanh toán
const [checkoutState, setCheckoutState] = useState<CheckoutState>({
  shippingInfo: {
    fullName: '',
    email: '',
    phone: '',
    address: '',
    city: '',
    province: '',
    postalCode: '',
    notes: '',
  },
  paymentMethod: null,
  currentStep: 'shipping',
  discountCode: null,
});
```

Giải thích:

- Đoạn code này sử dụng hook `useState` của React để khởi tạo và quản lý trạng thái cho toàn bộ quy trình thanh toán.
- Trạng thái được định nghĩa theo kiểu `CheckoutState` đã được định nghĩa trước đó để đảm bảo tính type-safe.
- Các giá trị khởi tạo:
  - `shippingInfo`: Khởi tạo tất cả các trường thông tin giao hàng là chuỗi rỗng, để sẵn sàng nhận dữ liệu từ người dùng.
  - `paymentMethod`: Ban đầu được set là `null`, vì người dùng chưa chọn phương thức thanh toán.
  - `currentStep`: Được khởi tạo là 'shipping', đây là bước đầu tiên trong quy trình thanh toán 3 bước.
  - `discountCode`: Ban đầu là `null`, cho biết chưa có mã giảm giá nào được áp dụng.
- `setCheckoutState` là hàm setter được trả về từ `useState`, sẽ được sử dụng để cập nhật trạng thái thanh toán trong suốt quy trình.

Tính toán giá trị đơn hàng sử dụng useMemo

```
// Tính giảm giá và tổng tiền cuối cùng - sử dụng useMemo để tránh tính toán lại khi
không cần thiết
```

```
const { discount, orderSummary } = useMemo(() => {
  // Tính s□ ti□n đ□□c gi□m giá
  const discountAmount = checkoutState.discountCode &&
checkoutState.discountCode.isValid
    ? (checkoutState.discountCode.discountType === 'percentage'
        ? Math.round((subtotal * checkoutState.discountCode.discountAmount) / 100)
        : checkoutState.discountCode.discountAmount)
    : 0;

  // Tính t□ng ti□n sau khi áp d□ng gi□m giá
  const finalTotal = subtotal + shipping + tax - discountAmount;

  // T□o orderSummary object
  const summary: OrderSummary = {
    items: cartItems.map((item) => ({
      id: item.id,
      name: item.name,
      quantity: item.quantity,
      price: item.price,
      size: item.size,
    })),
    subtotal,
    shipping,
    tax,
    discount: discountAmount,
    total: finalTotal,
  };

  return { discount: discountAmount, orderSummary: summary };
}, [cartItems, subtotal, shipping, tax, checkoutState.discountCode]);
```

Giải thích:

- Đoạn code này sử dụng hook `useMemo` của React để tối ưu hiệu suất bằng cách lưu trữ kết quả tính toán và chỉ tính toán lại khi các dependency thay đổi.
- Tính toán số tiền giảm giá:
  - Đầu tiên kiểm tra xem có mã giảm giá hợp lệ không thông qua `checkoutState.discountCode && checkoutState.discountCode.isValid`
  - Nếu có, kiểm tra loại giảm giá `discountType`:
- Nếu là 'percentage', tính giảm giá bằng cách lấy phần trăm của tổng tiền và làm tròn: `Math.round((subtotal * checkoutState.discountCode.discountAmount) / 100)`
- Nếu là 'fixed', sử dụng trực tiếp số tiền giảm giá: `checkoutState.discountCode.discountAmount`
  - Nếu không có mã giảm giá hợp lệ, số tiền giảm giá là 0
- Tính tổng tiền cuối cùng:
  - `finalTotal = subtotal + shipping + tax - discountAmount`: Tổng tiền bằng tổng tạm tính cộng với phí vận chuyển và thuế, sau đó trừ đi số tiền giảm giá

- Tạo đối tượng `OrderSummary`:
  - Dùng `cartItems.map()` để tạo danh sách các mặt hàng trong giỏ hàng
  - Thêm các giá trị tổng tiền, phí vận chuyển, thuế, giảm giá và tổng tiền cuối cùng
- Dependency array `[cartItems, subtotal, shipping, tax, checkoutState.discountCode]` đảm bảo tính toán lại chỉ khi một trong các giá trị này thay đổi:
  - `cartItems`: Khi danh sách sản phẩm thay đổi
  - `subtotal`: Khi tổng giá trị đơn hàng thay đổi
  - `shipping`, `tax`: Khi phí vận chuyển hoặc thuế thay đổi
  - `checkoutState.discountCode`: Khi mã giảm giá thay đổi

Xử lý áp dụng mã giảm giá

```
// Hàm x lý áp d ng mã gi m giá
const handleApplyDiscount = (code: string) => {
  // N u code là chu i tr ng, xóa mã gi m giá
  if (!code.trim()) {
    setCheckoutState((prev) => ({
      ...prev,
      discountCode: null
    }));
    return;
  }

  // Thông th ng s g i request lên server đ ki m tra mã gi m giá
  // Trong ví d này, chúng ta gi l p m t vài mã gi m giá có s n
  const validDiscounts: Record<string, DiscountCode> = {
    'WELCOME10': { code: 'WELCOME10', discountAmount: 10, discountType: 'percentage',
isValid: true },
    'FREESHIP': { code: 'FREESHIP', discountAmount: 30000, discountType: 'fixed',
isValid: true },
    'SUMMER25': { code: 'SUMMER25', discountAmount: 25, discountType: 'percentage',
isValid: true },
  };

  // Ki m tra xem mã gi m giá có h p l không
  const upperCaseCode = code.toUpperCase();
  if (upperCaseCode in validDiscounts) {
    setCheckoutState((prev) => ({
      ...prev,
      discountCode: validDiscounts[upperCaseCode]
    }));
  } else {
    // N u mã không h p l
    setCheckoutState((prev) => ({
      ...prev,
      discountCode: {
        code: code,
```

```
        discountAmount: 0,
        discountType: 'fixed' as const,
        isValid: false
      }
    }));
  }
};
```

Giải thích:

- Hàm `handleApplyDiscount` nhận vào một tham số là `code` kiểu string và xử lý việc áp dụng mã giảm giá vào đơn hàng.
- Kiểm tra mã trống:
  - Đầu tiên kiểm tra nếu `code` rỗng sau khi loại bỏ khoảng trắng với `!code.trim()`
  - Nếu rỗng, đặt `discountCode` trong `checkoutState` về `null` để xóa mã giảm giá
  - Kết thúc hàm sớm với lệnh `return`
- Thiết lập danh sách mã giảm giá hợp lệ:
  - Sử dụng `Record<string, DiscountCode>` để định nghĩa một đối tượng với key là string và value là DiscountCode
  - Các mã giảm giá được thiết lập sẵn gồm:
- `WELCOME10`: Giảm 10% tổng giá trị đơn hàng
- `FREESHIP`: Miễn phí vận chuyển 30,000đ
- `SUMMER25`: Giảm 25% tổng giá trị đơn hàng
  - Trong môi trường thực tế, việc kiểm tra mã giảm giá sẽ được thực hiện thông qua API call tới server
- Xác thực mã giảm giá:
  - Chuyển mã giảm giá về dạng chữ hoa với `code.toUpperCase()` để so sánh không phân biệt hoa thường
  - Sử dụng cú pháp `in` để kiểm tra xem mã giảm giá có tồn tại trong danh sách hay không
  - Nếu tồn tại, cập nhật `discountCode` trong `checkoutState` với thông tin mã giảm giá hợp lệ
- Xử lý mã không hợp lệ:
  - Nếu mã không nằm trong danh sách, vẫn cập nhật `discountCode` nhưng đánh dấu `isValid: false`
  - Đặt `discountAmount: 0` và `discountType: 'fixed'` để đảm bảo không có giảm giá nào được áp dụng
  - Sử dụng `as const` để giữ nguyên giá trị 'fixed' cho TypeScript, tránh lỗi type widening

Xử lý đặt hàng với hiệu ứng loading

```
// Hàm x lý đt hàng
```

```
const handlePlaceOrder = () => {
  // Hiển thị trạng thái xử lý đơn hàng
  setIsProcessing(true);

  // Mô phỏng thời gian xử lý đơn hàng
  setTimeout(() => {
    // Ở đây bạn sẽ gửi dữ liệu đơn hàng lên server
    // Kết thúc trạng thái xử lý
    setIsProcessing(false);

    // Trong ví dụ này, chỉ giả lập thành công
    alert('Đặt hàng thành công! Cảm ơn bạn đã mua sắm.');

    // Sau khi đặt hàng thành công, chuyển đến trang xác nhận
    navigate('/');
  }, 2000); // Giả lập 2 giây xử lý
};
```

Giải thích:

- Hàm `handlePlaceOrder` xử lý việc đặt hàng khi người dùng xác nhận đơn hàng ở bước cuối cùng. Hàm này tích hợp hiệu ứng loading để cải thiện UX.
- Cập nhật trạng thái xử lý:
  - `setIsProcessing(true)` bật trạng thái xử lý, khiến ứng dụng hiển thị loading skeleton thay vì form thông thường
  - Điều này giúp người dùng biết rằng đơn hàng đang được xử lý, không phải bị treo hoặc lỗi
- Mô phỏng API call:
  - Trong ứng dụng thực tế, đoạn code này sẽ gọi API đến server để tạo đơn hàng, xử lý thanh toán, v.v.
  - Ở đây sử dụng `setTimeout` để mô phỏng thời gian gọi API với thời gian delay là 2000ms (2 giây)
- Xử lý sau khi hoàn thành:
  - Đặt `setIsProcessing(false)` để tắt hiệu ứng loading
  - Hiển thị thông báo thành công cho người dùng qua `alert()`
  - Sử dụng `navigate('/')` từ React Router để chuyển hướng người dùng về trang chủ sau khi đặt hàng thành công
- Kết hợp với `OrderProcessingSkeleton` ở phần render, quá trình này tạo ra một trải nghiệm người dùng mượt mà, hiện đại, giúp người dùng theo dõi được tiến trình xử lý đơn hàng

Hiển thị component dựa vào bước hiện tại

```
// Component hiển thị theo bước hiện tại
const renderCurrentStep = () => {
  switch (checkoutState.currentStep) {
```

```
    case 'shipping':
      return (
        <ShippingForm
          shippingInfo={checkoutState.shippingInfo}
          setShippingInfo={handleUpdateShippingInfo}
          onNext={() => handleNextStep('payment')}
        />
      );
    case 'payment':
      return (
        <PaymentSelection
          selectedMethod={checkoutState.paymentMethod}
          setPaymentMethod={handleUpdatePaymentMethod}
          onNext={() => handleNextStep('confirmation')}
          onBack={() => handlePreviousStep('shipping')}
        />
      );
    case 'confirmation':
      return (
        <OrderConfirmation
          shippingInfo={checkoutState.shippingInfo}
          paymentMethod={checkoutState.paymentMethod!}
          orderSummary={orderSummary}
          onBack={() => handlePreviousStep('payment')}
          onPlaceOrder={handlePlaceOrder}
        />
      );
    default:
      return null;
  }
};
```

Giải thích:

- Hàm `renderCurrentStep` là một hàm quan trọng trong trang checkout, có nhiệm vụ hiển thị component thích hợp dựa trên bước hiện tại trong quy trình thanh toán.
- Cấu trúc switch-case:
  - Sử dụng mẫu thiết kế điều kiện dựa trên giá trị `checkoutState.currentStep`
  - Mỗi case tương ứng với một bước trong quy trình thanh toán
  - Default case trả về null để đảm bảo an toàn nếu `currentStep` không khớp với bất kỳ giá trị nào được định nghĩa
- Các component được render theo từng bước:
  - **Bước 'shipping'**: Render component `ShippingForm` với các props:
- `shippingInfo`: Thông tin giao hàng hiện tại từ state
- `setShippingInfo`: Hàm để cập nhật thông tin giao hàng
- `onNext`: Hàm callback chuyển đến bước tiếp theo khi hoàn thành form

    - **Bước 'payment'**: Render component `PaymentSelection` với các props:
- `selectedMethod`: Phương thức thanh toán hiện tại (có thể null nếu chưa chọn)
- `setPaymentMethod`: Hàm để cập nhật phương thức thanh toán
- `onNext`: Hàm callback chuyển đến bước xác nhận
- `onBack`: Hàm callback để quay lại bước nhập thông tin giao hàng
    - **Bước 'confirmation'**: Render component `OrderConfirmation` với các props:
- `shippingInfo`: Thông tin giao hàng đã nhập
- `paymentMethod`: Phương thức thanh toán đã chọn (sử dụng non-null assertion `!` vì ở bước này chắc chắn đã chọn phương thức thanh toán)
- `orderSummary`: Đối tượng tổng kết đơn hàng đã được tính toán
- `onBack`: Hàm callback để quay lại bước chọn phương thức thanh toán
- `onPlaceOrder`: Hàm callback để hoàn tất đặt hàng
- Kiểu thiết kế này cho phép:
    - Chia nhỏ giao diện phức tạp thành các component nhỏ hơn, dễ quản lý
    - Tách biệt logic xử lý của từng bước
    - Linh hoạt trong việc di chuyển qua lại giữa các bước
    - Dễ dàng mở rộng thêm bước nếu cần trong tương lai

Hiển thị tiến trình thanh toán

```
// Component hi□n th□ ti□n trình thanh toán
  const renderCheckoutProgress = () => {
    const steps = [
      { key: 'shipping' as CheckoutStep, label: 'Thông tin giao hàng' },
      { key: 'payment' as CheckoutStep, label: 'Ph□□ng th□c thanh toán' },
      { key: 'confirmation' as CheckoutStep, label: 'Xác nh□n đ□n hàng' },
    ];

    // Cho phép ng□□i dùng quay l□i các b□□c đã hoàn thành
    const handleStepClick = (step: CheckoutStep) => {
      // Ch□ cho phép quay l□i các b□□c đã hoàn thành, không cho nh□y cóc
      const currentStepIndex = steps.findIndex((s) => s.key ===
checkoutState.currentStep);
      const clickedStepIndex = steps.findIndex((s) => s.key === step);

      if (clickedStepIndex <= currentStepIndex) {
        setCheckoutState((prev) => ({
          ...prev,
          currentStep: step
        }));
      }
    };
```

```
    return (
      <StepIndicator
        steps={steps}
        currentStep={checkoutState.currentStep as CheckoutStep}
        onStepClick={handleStepClick}
        allowNavigation={true}
      />
    );
  };
```

Component Step Indicator (StepIndicator.tsx)

Imports

```
import React from 'react';
import { motion } from 'framer-motion';
import { cn } from '@/lib/utils';
```

Định nghĩa kiểu dữ liệu

```
export type CheckoutStep = 'shipping' | 'payment' | 'confirmation';

interface StepIndicatorProps {
  steps: {
    key: CheckoutStep;
    label: string;
  }[];
  currentStep: CheckoutStep;
  onStepClick?: (step: CheckoutStep) => void;
  allowNavigation?: boolean;
}
```

Chức năng và hoạt động

StepIndicator là component hiển thị tiến trình thanh toán với các tính năng:

- Hiển thị các bước trong quy trình thanh toán với trạng thái trực quan (đã hoàn thành, đang thực hiện, chưa thực hiện)
- Cho phép người dùng quay lại các bước đã hoàn thành (tùy chọn)
- Hiệu ứng chuyển động khi thay đổi bước và khi người dùng tương tác
- Kết nối các bước bằng đường nối có thay đổi màu sắc theo tiến độ

Logic hiển thị

```
export const StepIndicator = ({
  steps,
  currentStep,
  onStepClick,
  allowNavigation = false,
}: StepIndicatorProps) => {
  const currentStepIndex = steps.findIndex((step) => step.key === currentStep);

  return (
    <div className="mb-8">
      <div className="flex items-center justify-center">
        {steps.map((step, index) => {
          // Xác đ□nh tr□ng thái c□a m□i b□□c
          const isCompleted = index < currentStepIndex;
          const isActive = step.key === currentStep;
          const isPending = index > currentStepIndex;

          return (
            <React.Fragment key={step.key}>
              <motion.div
                className={cn(
                  "flex flex-col items-center",
                  allowNavigation && index <= currentStepIndex && "cursor-pointer"
                )}
                onClick={() => {
                  if (allowNavigation && index <= currentStepIndex && onStepClick) {
                    onStepClick(step.key);
                  }
                }}
                whileHover={
                  allowNavigation && index <= currentStepIndex
                    ? { scale: 1.05 }
                    : {}
                }
              >
                <motion.div
                  className={cn(
                    "w-10 h-10 flex items-center justify-center rounded-full",
                    isActive && "bg-primary text-primary-foreground",
                    isCompleted && "bg-green-500 text-white",
                    isPending && "bg-gray-200 text-gray-600"
                  )}
                  initial={false}
                  animate={{
                    scale: isActive ? 1.1 : 1,
                    transition: { type: "spring", stiffness: 500, damping: 30 }
                  }}
                >
```

```
                {isCompleted ? (
                  <svg
                    xmlns="http://www.w3.org/2000/svg"
                    className="h-6 w-6"
                    fill="none"
                    viewBox="0 0 24 24"
                    stroke="currentColor"
                  >
                    <path
                      strokeLinecap="round"
                      strokeLinejoin="round"
                      strokeWidth={2}
                      d="M5 13l4 4L19 7"
                    />
                  </svg>
                ) : (
                  index + 1
                )}
              </motion.div>
              <motion.span
                className={cn(
                  "mt-2 text-sm",
                  isActive && "font-medium",
                  isPending && "text-gray-500"
                )}
                animate={{
                  fontWeight: isActive ? 600 : 400
                }}
              >
                {step.label}
              </motion.span>
            </motion.div>

            {/* Dòng k□t n□i gi□a các b□□c */}
            {index < steps.length - 1 && (
              <motion.div
                className="flex-grow h-0.5 mx-2"
                initial={{ backgroundColor: "#e5e7eb" }} // Gray-200
                animate={{
                  backgroundColor: index < currentStepIndex ? "#10b981" : "#e5e7eb"
// Green-500 n□u đã hoàn thành, Gray-200 n□u ch□a
                }}
                transition={{ duration: 0.5 }}
              />
            )}
          </React.Fragment>
        );
      })}
    </div>
  </div>
);
```

```
};
```

Giải thích:

- Component `StepIndicator` là một component phức tạp hiển thị tiến trình thanh toán với hiệu ứng trực quan, giúp người dùng biết được họ đang ở bước nào trong quy trình.
- Nhận các props:
  - `steps`: Mảng các bước cần hiển thị, mỗi bước gồm key và label
  - `currentStep`: Bước hiện tại đang được hiển thị
  - `onStepClick`: Callback function khi người dùng nhấp vào một bước
  - `allowNavigation`: Boolean cho phép/không cho phép điều hướng giữa các bước (mặc định là false)
- Tính toán `currentStepIndex`:
  - Sử dụng `findIndex` để xác định vị trí của bước hiện tại trong mảng steps
  - Chỉ số này được sử dụng để xác định các bước đã hoàn thành và chưa hoàn thành
- Hiển thị từng bước:
  - Sử dụng `steps.map()` để lặp qua tất cả các bước và render từng bước
  - Xác định 3 trạng thái: đã hoàn thành (isCompleted), đang thực hiện (isActive), hoặc chưa thực hiện (isPending)
  - Mỗi bước được bọc trong `React.Fragment` với key duy nhất
- Xử lý click và hover:
  - Chỉ cho phép click vào các bước đã hoàn thành hoặc bước hiện tại nếu `allowNavigation` là true
  - Tạo hiệu ứng hover với `whileHover={{ scale: 1.05 }}` khi hover vào các bước có thể click
- Tùy chỉnh hiển thị bước:
  - Sử dụng thư viện utility class `cn()` để điều chỉnh class dựa trên điều kiện
  - Mỗi trạng thái có màu nền và màu chữ khác nhau: xanh lá (đã hoàn thành), xanh dương (đang thực hiện), xám (chưa thực hiện)
  - Bước đang thực hiện được phóng to nhẹ (scale: 1.1) để nổi bật
- Hiển thị nội dung bước:
  - Nếu bước đã hoàn thành, hiển thị biểu tượng dấu tích SVG
  - Nếu chưa hoàn thành, hiển thị số thứ tự của bước (index + 1)
  - Label của bước được hiển thị bên dưới với độ đậm khác nhau theo trạng thái
- Dòng kết nối giữa các bước:
  - Thêm dòng kết nối nằm ngang giữa các chỉ số bước (trừ bước cuối cùng)
  - Màu sắc dòng kết nối thay đổi dựa trên trạng thái: xanh lá nếu bước trước đã hoàn thành, xám nếu chưa

- Sử dụng animation với `transition={{ duration: 0.5 }}` để tạo hiệu ứng chuyển màu mượt mà

```
```

Component ShippingForm (ShippingForm.tsx)

Imports

```
import { ShippingInfo } from '@/types/checkout';
import { Input } from '@/components/ui/input';
import { Textarea } from '@/components/ui/textarea';
import { Button } from '@/components/ui/button';
import { SelectComponent, SelectItem } from '@/components/ui/select';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { shippingSchema, ShippingFormValues } from '@/lib/form-validation';
import {
  Form,
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from '@/components/ui/form';
import { motion } from 'framer-motion';
```

Thành phần RadixUI sử dụng

- **Form**: Component từ shadcn/ui dựa trên React Hook Form để quản lý form
- **FormControl**, **FormField**, **FormItem**, **FormLabel**, **FormMessage**: Các component con của Form
- **SelectComponent**: Component select tùy chỉnh dựa trên Radix UI Select

Chức năng và hoạt động

ShippingForm là component hiển thị và xử lý form nhập thông tin giao hàng với các tính năng:

- Biểu mẫu với xác thực dữ liệu sử dụng React Hook Form và Zod
- Hiển thị thông báo lỗi khi dữ liệu không hợp lệ
- Hiệu ứng animation khi form được hiển thị
- Lưu thông tin giao hàng vào trạng thái của trang checkout

Logic xử lý form

```
export const ShippingForm = ({ shippingInfo, setShippingInfo, onNext }:
ShippingFormProps) => {
  // S□ d□ng react-hook-form và zod đ□ xác th□c form
  const form = useForm<ShippingFormValues>({
    resolver: zodResolver(shippingSchema),
    defaultValues: {
      fullName: shippingInfo.fullName || '',
      email: shippingInfo.email || '',
      phone: shippingInfo.phone || '',
      address: shippingInfo.address || '',
      city: shippingInfo.city || '',
      province: shippingInfo.province || '',
      postalCode: shippingInfo.postalCode || '',
      notes: shippingInfo.notes || '',
    },
  });

  const onSubmit = (data: ShippingFormValues) => {
    setShippingInfo(data);
    onNext();
  };

  return (
    <motion.div
      initial={{ opacity: 0, x: -20 }}
      animate={{ opacity: 1, x: 0 }}
      exit={{ opacity: 0, x: 20 }}
      transition={{ duration: 0.3 }}
      className="bg-white rounded-lg shadow-md p-6"
    >
      <h2 className="text-xl font-bold mb-4">Thông tin giao hàng</h2>
      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6">
          {/* Form fields implementation */}
        </form>
      </Form>
    </motion.div>
  );
};
```

Giải thích:

- Component `ShippingForm` chịu trách nhiệm thu thập thông tin giao hàng từ người dùng với tính năng xác thực dữ liệu mạnh mẽ.
- Khởi tạo form với React Hook Form:
  - Sử dụng `useForm<ShippingFormValues>()` để tạo một instance của form với kiểu dữ liệu được

định nghĩa từ schema Zod

- `resolver: zodResolver(shippingSchema)` đăng ký Zod làm thư viện xác thực form
- `defaultValues` thiết lập giá trị ban đầu cho các trường từ props `shippingInfo`, nếu có, hoặc chuỗi rỗng nếu không
- Các trường bao gồm thông tin cá nhân (họ tên, email, phone), địa chỉ giao hàng (địa chỉ, thành phố, tỉnh/thành phố, mã bưu điện) và ghi chú tùy chọn

- Hàm xử lý gửi form:
  - `onSubmit` được gọi khi form hợp lệ và được gửi đi
  - `setShippingInfo(data)` cập nhật thông tin giao hàng trong state của component cha
  - `onNext()` chuyển người dùng đến bước tiếp theo trong quy trình thanh toán
- Hiệu ứng animation với framer-motion:
  - `initial={{ opacity: 0, x: -20 }}`: Component ban đầu sẽ trong suốt và dịch chuyển 20px sang trái
  - `animate={{ opacity: 1, x: 0 }}`: Animation hiện component với độ trong suốt đầy đủ và về vị trí gốc
  - `exit={{ opacity: 0, x: 20 }}`: Khi component biến mất sẽ mờ dần và dịch chuyển 20px sang phải
  - `transition={{ duration: 0.3 }}`: Thời gian chuyển động là 0.3 giây
  - Hiệu ứng này tạo cảm giác mượt mà khi người dùng di chuyển qua các bước thanh toán
- Cấu trúc form:
  - Sử dụng component `Form` từ shadcn/ui để tích hợp với React Hook Form
  - Các trường input được thiết kế theo lưới grid với các kích thước phản hồi (responsive)
  - Mỗi trường input đều có nhãn, trường nhập liệu và vùng hiển thị thông báo lỗi
  - Các trường bắt buộc được xác thực theo schema đã định nghĩa trước khi cho phép gửi form
- Tiện ích cho người dùng:
  - Định dạng grid linh hoạt cho phép hiển thị tốt trên cả thiết bị di động và máy tính
  - Placeholder text trong mỗi trường input giúp người dùng hiểu rõ thông tin cần nhập
  - Thông báo lỗi hiển thị ngay dưới trường input giúp người dùng dễ dàng sửa lỗi

Component PaymentSelection (PaymentSelection.tsx)

Imports

```
import React from 'react';
import { PaymentMethod } from '@/types/checkout';
import { Button } from '@/components/ui/button';
import { RadioGroup, RadioGroupItem } from '@/components/ui/radio-group';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { paymentSchema, PaymentFormValues } from '@/lib/form-validation';
import {
  Form,
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from '@/components/ui/form';
import { motion } from 'framer-motion';
```

Thành phần RadixUI sử dụng

- **RadioGroup**, **RadioGroupItem**: Components từ Radix UI Radio Group
- **Form**: Component từ shadcn/ui dựa trên React Hook Form
- **FormControl**, **FormField**, **FormItem**, **FormLabel**, **FormMessage**: Components con của Form

Chức năng và hoạt động

PaymentSelection là component hiển thị và xử lý lựa chọn phương thức thanh toán với các tính năng:

- Hiển thị các phương thức thanh toán có sẵn (Stripe và Thanh toán khi nhận hàng)
- Hiển thị thông tin chi tiết về phương thức thanh toán khi được chọn
- Xác thực lựa chọn của người dùng trước khi cho phép tiếp tục
- Hiệu ứng animation khi component được hiển thị

Logic xử lý

```
export const PaymentSelection = ({
  selectedMethod,
  setPaymentMethod,
  onNext,
  onBack
}: PaymentSelectionProps) => {
  // S dng react-hook-form và zod  xác th c form
  const form = useForm<PaymentFormValues>({
    resolver: zodResolver(paymentSchema),
    defaultValues: {
```

```
      paymentMethod: selectedMethod || undefined,
    },
  });

  const onSubmit = (data: PaymentFormValues) => {
    setPaymentMethod(data.paymentMethod);
    onNext();
  };

  // L□y giá tr□ ph□□ng th□c thanh toán hi□n t□i t□ form
  const watchPaymentMethod = form.watch('paymentMethod');

  return (
    <motion.div
      initial={{ opacity: 0, x: -20 }}
      animate={{ opacity: 1, x: 0 }}
      exit={{ opacity: 0, x: 20 }}
      transition={{ duration: 0.3 }}
      className="bg-white rounded-lg shadow-md p-6"
    >
      <h2 className="text-xl font-bold mb-4">Ph□□ng th□c thanh toán</h2>
      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6">
          <FormField
            control={form.control}
            name="paymentMethod"
            render={({ field }) => (
              <FormItem className="space-y-3">
                <FormLabel>Ch□n ph□□ng th□c thanh toán</FormLabel>
                <FormControl>
                  <RadioGroup
                    onValueChange={field.onChange}
                    value={field.value}
                    className="space-y-3"
                  >
                    <div className="flex items-center p-4 border rounded-md cursor-
pointer hover:bg-gray-50 transition-colors">
                      <FormItem className="flex items-center space-x-3 space-y-0">
                        <FormControl>
                          <RadioGroupItem value="stripe" />
                        </FormControl>
                        <div className="w-full">
                          <FormLabel className="font-medium cursor-pointer">
                            Thanh toán qua Stripe
                          </FormLabel>
                          <p className="text-sm text-gray-500">Thanh toán an toàn v□i
Visa, Mastercard, JCB</p>
                        </div>
                      </FormItem>
                    </div>
```

```
                    <div className="flex items-center p-4 border rounded-md cursor-
pointer hover:bg-gray-50 transition-colors">
                      <FormItem className="flex items-center space-x-3 space-y-0">
                        <FormControl>
                          <RadioGroupItem value="cash_on_delivery" />
                        </FormControl>
                        <div className="w-full">
                          <FormLabel className="font-medium cursor-pointer">
                            Thanh toán khi nh□n hàng (COD)
                          </FormLabel>
                          <p className="text-sm text-gray-500">Thanh toán b□ng ti□n m□t
khi nh□n hàng</p>
                        </div>
                      </FormItem>
                    </div>
                  </RadioGroup>
                </FormControl>
                <FormMessage />
              </FormItem>
            )}
          />

          {watchPaymentMethod === 'stripe' && (
            <div className="mt-6 p-4 border rounded-md">
              <p className="font-medium mb-2">Thanh toán qua Stripe:</p>
              <p className="text-sm mb-4">
                B□n s□ đ□□c chuy□n đ□n trang thanh toán an toàn c□a Stripe sau khi xác
nh□n đ□n hàng.
              </p>
              <div className="flex items-center space-x-2 mb-4">
                <img src="/assets/images/visa.svg" alt="Visa" className="h-8" />
                <img src="/assets/images/mastercard.svg" alt="Mastercard"
className="h-8" />
                <img src="/assets/images/amex.svg" alt="American Express"
className="h-8" />
                <img src="/assets/images/jcb.svg" alt="JCB" className="h-8" />
              </div>
              <p className="text-xs text-gray-500">
                T□t c□ thông tin th□ c□a b□n đ□□c mã hóa và b□o m□t b□i Stripe. Chúng tôi
không l□u tr□ thông tin th□ c□a b□n.
              </p>
            </div>
          )}

          <div className="flex justify-between mt-6">
            <Button
              type="button"
              variant="outline"
              onClick={onBack}
            >
              Quay l□i
```

```
            </Button>
            <Button
              type="submit"
              disabled={!watchPaymentMethod}
            >
              Ti□p t□c đ□n xác nh□n đ□n hàng
            </Button>
          </div>
        </form>
      </Form>
    </motion.div>
  );
};
```

Giải thích:

- Component `PaymentSelection` quản lý bước chọn phương thức thanh toán trong quy trình checkout, cung cấp cho người dùng các lựa chọn thanh toán và yêu cầu họ chọn một phương thức trước khi tiếp tục.
- Khởi tạo form với React Hook Form:
  - Tương tự như form thông tin giao hàng, sử dụng `useForm<PaymentFormValues>()` kết hợp với Zod để xác thực dữ liệu
  - `defaultValues` khởi tạo phương thức thanh toán từ props `selectedMethod` nếu có, hoặc `undefined` nếu người dùng chưa chọn phương thức nào
  - Cấu trúc này giúp giữ nguyên lựa chọn của người dùng khi họ quay lại bước này
- Hook Theo dõi giá trị form:
  - `const watchPaymentMethod = form.watch('paymentMethod')` theo dõi thay đổi của trường `paymentMethod`
  - Điều này cho phép component phản ứng ngay lập tức khi người dùng chọn một phương thức thanh toán mà không cần đợi form được gửi đi
- Render UI dựa trên phương thức thanh toán:
  - Sử dụng `RadioGroup` từ Radix UI để hiển thị các lựa chọn phương thức thanh toán
  - Các lựa chọn được trình bày dưới dạng thẻ với hiệu ứng hover và cursor pointer để cải thiện UX
  - Mỗi phương thức có icon, tên và mô tả ngắn để người dùng hiểu rõ lựa chọn
- Hiển thị có điều kiện thông tin thanh toán:
  - Khi người dùng chọn 'stripe', hiển thị thêm thông tin về phương thức thanh toán này
  - Hiển thị logo các loại thẻ được hỗ trợ, giúp người dùng xác nhận rằng thẻ của họ được chấp nhận
  - Thông báo về bảo mật tạo sự tin tưởng với thông tin "Tất cả thông tin thẻ của bạn được mã hóa và bảo mật bởi Stripe"
- Điều hướng và chức năng gửi form:

    - Nút "Quay lại" gọi hàm `onBack` để quay lại bước thông tin giao hàng
    - Nút "Tiếp tục đến xác nhận đơn hàng" được kích hoạt khi form hợp lệ (đã chọn phương thức thanh toán)
    - Thuộc tính `disabled={!watchPaymentMethod}` ngăn người dùng tiếp tục nếu chưa chọn phương thức thanh toán
    - Khi form được gửi, dữ liệu được chuyển đến component cha thông qua `setPaymentMethod` và chuyển sang bước tiếp theo với `onNext()`
- Animation và thiết kế:
    - Sử dụng framer-motion cho hiệu ứng chuyển đổi mượt mà giữa các màn hình
    - Giao diện tối giản, tập trung vào lựa chọn phương thức thanh toán
    - Các phần tử UI được thiết kế để dễ sử dụng trên cả thiết bị di động và desktop

```
```

Component OrderConfirmation (OrderConfirmation.tsx)

Imports

```
import React from 'react';
import { ShippingInfo, PaymentMethod, OrderSummary } from '@/types/checkout';
import { Button } from '@/components/ui/button';
import { motion } from 'framer-motion';
```

Thành phần RadixUI sử dụng

- **Button**: Component từ shadcn/ui

Chức năng và hoạt động

OrderConfirmation là component hiển thị trang xác nhận đơn hàng trước khi đặt hàng chính thức với các tính năng:

- Hiển thị tóm tắt thông tin giao hàng
- Hiển thị phương thức thanh toán đã chọn
- Hiển thị chi tiết đơn hàng
- Hiển thị tổng tiền, phí vận chuyển, thuế và giảm giá nếu có
- Cho phép xác nhận đơn hàng hoặc quay lại bước trước

Logic xử lý

```
export const OrderConfirmation = ({
  shippingInfo,
  paymentMethod,
  orderSummary,
  onBack,
  onPlaceOrder,
}: OrderConfirmationProps) => {
  const getPaymentMethodName = (method: PaymentMethod) => {
    const methods = {
      stripe: 'Thanh toán qua Stripe',
      cash_on_delivery: 'Thanh toán khi nh□n hàng (COD)',
    };
    return methods[method];
  };

  return (
    <motion.div
      initial={{ opacity: 0, x: -20 }}
      animate={{ opacity: 1, x: 0 }}
      exit={{ opacity: 0, x: 20 }}
      transition={{ duration: 0.3 }}
      className="bg-white rounded-lg shadow-md p-6"
    >
      <h2 className="text-xl font-bold mb-4">Xác nh□n đ□n hàng</h2>

      <div className="border-b pb-4 mb-4">
        <h3 className="font-medium mb-2">Thông tin giao hàng</h3>
        <div className="grid grid-cols-1 md:grid-cols-2 gap-2 text-sm">
          <p><span className="font-medium">H□ tên:</span> {shippingInfo.fullName}</p>
          <p><span className="font-medium">Email:</span> {shippingInfo.email}</p>
          <p><span className="font-medium">S□ đi□n tho□i:</span>
{shippingInfo.phone}</p>
          <p className="col-span-full"><span className="font-medium">Đ□a ch□:</span>
{shippingInfo.address}, {shippingInfo.city}, {shippingInfo.province},
{shippingInfo.postalCode}</p>
          {shippingInfo.notes && (
            <p className="col-span-full"><span className="font-medium">Ghi chú:</span>
{shippingInfo.notes}</p>
          )}
        </div>
      </div>

      <div className="border-b pb-4 mb-4">
        <h3 className="font-medium mb-2">Ph□□ng th□c thanh toán</h3>
        <p>{getPaymentMethodName(paymentMethod)}</p>
      </div>

      <div className="border-b pb-4 mb-4">
```

```
        <h3 className="font-medium mb-2">Chi ti□t đ□n hàng</h3>
        <div className="space-y-2">
          {orderSummary.items.map((item) => (
            <div key={item.id} className="flex justify-between text-sm">
              <span>{item.name} {item.size && `(${item.size})`}
x{item.quantity}</span>
              <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency:
'VND' }).format(item.price * item.quantity)}</span>
            </div>
          ))}
        </div>
      </div>

      <div className="space-y-2 mb-6">
        <div className="flex justify-between">
          <span>T□m tính</span>
          <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(orderSummary.subtotal)}</span>
        </div>
        <div className="flex justify-between">
          <span>Phí v□n chuy□n</span>
          <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(orderSummary.shipping)}</span>
        </div>
        <div className="flex justify-between">
          <span>Thu□ (VAT)</span>
          <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(orderSummary.tax)}</span>
        </div>
        {orderSummary.discount > 0 && (
          <div className="flex justify-between text-green-600">
            <span>Gi□m giá</span>
            <span>-{new Intl.NumberFormat('vi-VN', { style: 'currency', currency:
'VND' }).format(orderSummary.discount)}</span>
          </div>
        )}
        <div className="flex justify-between font-bold text-lg border-t pt-2">
          <span>T□ng c□ng</span>
          <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(orderSummary.total)}</span>
        </div>
      </div>

      <div className="flex justify-between mt-6">
        <Button
          variant="outline"
          onClick={onBack}
        >
          Quay l□i
        </Button>
        <Button
```

```
          onClick={onPlaceOrder}
        >
          Đ□t hàng
        </Button>
      </div>
    </motion.div>
  );
};
```

Giải thích:

- Component `OrderConfirmation` là bước cuối cùng trong quy trình thanh toán, hiển thị tất cả thông tin đơn hàng để người dùng xác nhận trước khi hoàn tất việc đặt hàng.
- Xử lý hiển thị tên phương thức thanh toán:
  - Hàm `getPaymentMethodName` nhận PaymentMethod và trả về tên hiển thị người dùng có thể đọc được
  - Sử dụng object mapping thay vì nhiều câu lệnh if-else để đơn giản hóa code và dễ bảo trì
  - Hỗ trợ cả 'stripe' (thanh toán trực tuyến) và 'cash_on_delivery' (thanh toán khi nhận hàng)
- Cấu trúc giao diện rõ ràng, chia thành các phần riêng biệt:
  - Phần đầu hiển thị tiêu đề "Xác nhận đơn hàng"
  - Phần thông tin giao hàng hiển thị họ tên, email, số điện thoại, và địa chỉ đầy đủ
  - Phần phương thức thanh toán hiển thị tên phương thức đã chọn được chuyển đổi bởi hàm `getPaymentMethodName`
  - Phần chi tiết đơn hàng liệt kê từng sản phẩm với tên, kích cỡ, số lượng và giá tiền
  - Phần tổng kết thanh toán hiển thị tạm tính, phí vận chuyển, thuế, giảm giá (nếu có) và tổng cộng
- Hiển thị có điều kiện với giảm giá:
  - Phần hiển thị giảm giá chỉ xuất hiện khi `orderSummary.discount > 0`
  - Giá trị giảm giá được hiển thị với dấu trừ (-) trước để chỉ rằng đây là khoản giảm trừ
  - Sử dụng màu xanh lá (`text-green-600`) để tạo ấn tượng tích cực về khoản giảm giá
- Định dạng tiền tệ nhất quán:
  - Sử dụng `Intl.NumberFormat` để định dạng tất cả số tiền theo tiêu chuẩn Việt Nam (VND)
  - Giúp hiển thị số tiền dễ đọc với dấu phân cách hàng nghìn và ký hiệu tiền tệ
- Bố cục Responsive:
  - Thông tin giao hàng sử dụng grid layout với 1 cột trên thiết bị di động và 2 cột trên màn hình lớn hơn
  - Địa chỉ và ghi chú (nếu có) trải dài toàn bộ chiều rộng với `col-span-full`
  - Khoảng cách thích hợp giữa các phần với padding và margin
- Điều hướng:

    - Nút "Quay lại" cho phép người dùng quay lại bước chọn phương thức thanh toán để chỉnh sửa nếu cần
    - Nút "Đặt hàng" nổi bật và gọi hàm `onPlaceOrder` để hoàn tất quy trình thanh toán khi được nhấn
- Animation mượt mà:
    - Sử dụng framer-motion để tạo hiệu ứng chuyển đổi khi component xuất hiện và biến mất
    - Tạo cảm giác chuyển động liền mạch trong quá trình thanh toán, nâng cao UX

```
```

Component CheckoutSummary (CheckoutSummary.tsx)

Imports

```
import React, { useState } from 'react';
import { CartItem } from '@/types/cart';
import { DiscountCode, OrderSummary } from '@/types/checkout';
import { DiscountForm } from './DiscountForm';
```

Chức năng và hoạt động

CheckoutSummary là component hiển thị tóm tắt đơn hàng ở sidebar bên phải của trang thanh toán với các tính năng:

- Hiển thị danh sách sản phẩm trong giỏ hàng
- Hiển thị form nhập mã giảm giá
- Hiển thị tổng tiền tạm tính, phí vận chuyển, thuế và giảm giá
- Tính và hiển thị tổng số tiền cần thanh toán

Logic xử lý

```
export const CheckoutSummary: React.FC<CheckoutSummaryProps> = ({
  items,
  subtotal,
  discountCode = null,
  onApplyDiscount = () => {}
}) => {
  const [isProcessing, setIsProcessing] = useState(false);

  // Tính toán giá trị đơn hàng
```

```
  const shipping = 30000; // Phí v□n chuy□n c□ đ□nh, có th□ thay đ□i theo logic c□a b□n
  const tax = Math.round(subtotal * 0.1); // VAT 10%

  // Tính s□ ti□n đ□□c gi□m giá
  const discount = discountCode && discountCode.isValid
    ? (discountCode.discountType === 'percentage'
        ? Math.round((subtotal * discountCode.discountAmount) / 100)
        : discountCode.discountAmount)
    : 0;

  // Tính t□ng ti□n sau khi tr□ gi□m giá
  const total = subtotal + shipping + tax - discount;

  // Hàm x□ lý áp d□ng mã gi□m giá
  const handleApplyDiscount = (code: string) => {
    setIsProcessing(true);
    // G□i hàm t□ props
    onApplyDiscount(code);

    // Gi□ l□p th□i gian x□ lý
    setTimeout(() => {
      setIsProcessing(false);
    }, 1000);
  };

  return (
    <div className="bg-white rounded-lg shadow-md p-6 h-fit sticky top-6">
      <h2 className="text-xl font-bold mb-4">Tóm t□t đ□n hàng</h2>
      <div className="space-y-4">
        <div className="max-h-64 overflow-y-auto">
          {items.map((item) => (
            <div key={item.id} className="flex items-center py-2 border-b">
              <div className="w-16 h-16 flex-shrink-0">
                <img
                  src={item.image}
                  alt={item.name}
                  className="w-full h-full object-cover rounded"
                />
              </div>
              <div className="ml-4 flex-grow">
                <h3 className="font-medium">{item.name}</h3>
                <p className="text-sm text-gray-500">
                  {item.size && `Size: ${item.size}`} | S□ l□□ng: {item.quantity}
                </p>
              </div>
              <div className="font-medium">
                {new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(item.price * item.quantity)}
              </div>
            </div>
          ))}
```

```
        </div>

        {/* Form nh□p mã gi□m giá */}
        <DiscountForm
          onApplyDiscount={handleApplyDiscount}
          appliedDiscount={discountCode}
          isLoading={isProcessing}
        />

        <div className="space-y-2 pt-2">
          <div className="flex justify-between">
            <span>T□m tính</span>
            <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(subtotal)}</span>
          </div>
          <div className="flex justify-between">
            <span>Phí v□n chuy□n</span>
            <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(shipping)}</span>
          </div>
          <div className="flex justify-between">
            <span>Thu□ (VAT)</span>
            <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(tax)}</span>
          </div>

          {/* Hi□n th□ gi□m giá n□u có */}
          {discount > 0 && (
            <div className="flex justify-between text-green-600">
              <span>Gi□m giá</span>
              <span>-{new Intl.NumberFormat('vi-VN', { style: 'currency', currency:
'VND' }).format(discount)}</span>
            </div>
          )}

          <div className="flex justify-between font-bold text-lg border-t pt-2">
            <span>T□ng c□ng</span>
            <span>{new Intl.NumberFormat('vi-VN', { style: 'currency', currency: 'VND'
}).format(total)}</span>
          </div>
        </div>
      </div>
    </div>
  );
};
```

Giải thích:

- Component `CheckoutSummary` đóng vai trò là sidebar tóm tắt đơn hàng, xuất hiện xuyên suốt các bước trong quy trình thanh toán, giúp người dùng luôn nắm được thông tin đơn hàng của họ.

- Khởi tạo trạng thái và tính toán giá trị:
  - Sử dụng `useState` để theo dõi trạng thái xử lý mã giảm giá (`isProcessing`)
  - Xác định phí vận chuyển cố định (30,000đ) - trong dự án thực tế, giá trị này có thể được tính toán dựa trên vị trí, khối lượng đơn hàng, v.v.
  - Tính thuế VAT 10% dựa trên tổng giá trị đơn hàng với `Math.round(subtotal * 0.1)` để đảm bảo làm tròn đến đơn vị đồng
  - Tính toán số tiền giảm giá dựa trên loại mã giảm giá (phần trăm hoặc cố định)
  - Tính tổng tiền cuối cùng bằng cách cộng tạm tính, phí vận chuyển, thuế và trừ đi số tiền giảm giá
- Xử lý áp dụng mã giảm giá:
  - Hàm `handleApplyDiscount` được gọi khi người dùng áp dụng mã giảm giá thông qua component `DiscountForm`
  - Đặt `isProcessing` thành `true` để hiển thị trạng thái loading
  - Gọi callback `onApplyDiscount` được truyền vào qua props để xác thực mã giảm giá (thông thường sẽ gửi API request đến server)
  - Sử dụng `setTimeout` để mô phỏng thời gian xử lý mạng, sau đó đặt `isProcessing` về `false`
- Thiết kế giao diện:
  - Sử dụng `sticky top-6` để sidebar luôn hiển thị khi người dùng cuộn trang
  - Danh sách sản phẩm được giới hạn chiều cao với `max-h-64 overflow-y-auto` để tránh sidebar quá dài khi có nhiều sản phẩm
  - Mỗi sản phẩm hiển thị hình ảnh thu nhỏ, tên, kích cỡ, số lượng và giá tiền
  - Sử dụng component `DiscountForm` để cho phép người dùng nhập và áp dụng mã giảm giá
- Hiển thị thông tin thanh toán:
  - Phần tóm tắt chi phí hiển thị các mục: Tạm tính, Phí vận chuyển, Thuế
  - Khoản giảm giá chỉ được hiển thị khi `discount > 0` với màu xanh lá để nổi bật
  - Tổng cộng được phân tách với đường kẻ phía trên và font chữ đậm để làm nổi bật
  - Tất cả giá trị tiền tệ được định dạng nhất quán bằng `Intl.NumberFormat` với định dạng Việt Nam
- Ưu điểm thiết kế UX:
  - Cấu trúc rõ ràng giúp người dùng dễ dàng hiểu được chi phí đơn hàng
  - Hiển thị giá chi tiết từng mục tạo sự minh bạch trong quá trình thanh toán
  - Tích hợp form áp dụng mã giảm giá ngay trong component giúp người dùng dễ dàng sử dụng
  - Thiết kế responsive và sticky giúp người dùng luôn thấy được thông tin đơn hàng trong suốt quy trình thanh toán

Component DiscountForm (DiscountForm.tsx)

Imports

```
import React, { useState } from 'react';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { DiscountCode } from '@/types/checkout';
```

Thành phần RadixUI sử dụng

- **Button**: Component từ shadcn/ui
- **Input**: Component từ shadcn/ui

Chức năng và hoạt động

DiscountForm là component cho phép người dùng nhập và áp dụng mã giảm giá với các tính năng:

- Form nhập mã giảm giá
- Hiển thị trạng thái loading khi đang xử lý mã
- Hiển thị thông tin mã giảm giá đã áp dụng
- Cho phép xóa mã giảm giá đã áp dụng

Logic xử lý

```
export const DiscountForm = ({ onApplyDiscount, appliedDiscount, isLoading = false }:
DiscountFormProps) => {
  const [code, setCode] = useState<string>('');
  const [error, setError] = useState<string | null>(null);

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();

    if (!code.trim()) {
      setError('Vui lòng nh□p mã gi□m giá');
      return;
    }

    setError(null);
    onApplyDiscount(code);
  };
```

```
  const handleRemoveDiscount = () => {
    // G□i mã tr□ng đ□ xóa mã gi□m giá
    onApplyDiscount('');
    setCode('');
  };

  return (
    <div className="border rounded-md p-4 mb-4">
      <h3 className="text-sm font-medium mb-3">Mã gi□m giá</h3>

      {appliedDiscount && appliedDiscount.isValid ? (
        <div>
          <div className="flex items-center justify-between bg-green-50 border border-
green-200 rounded p-2 mb-3">
            <div>
              <p className="text-sm font-medium">{appliedDiscount.code}</p>
              <p className="text-xs text-green-600">
                {appliedDiscount.discountType === 'percentage'
                  ? `Gi□m ${appliedDiscount.discountAmount}%`
                  : `Gi□m ${appliedDiscount.discountAmount.toLocaleString('vi-VN')}đ`}
              </p>
            </div>
            <Button
              variant="ghost"
              size="sm"
              onClick={handleRemoveDiscount}
              disabled={isLoading}
              className="h-8 text-sm"
            >
              Xóa
            </Button>
          </div>
        </div>
      ) : (
        <form onSubmit={handleSubmit} className="flex gap-2">
          <div className="flex-1">
            <Input
              placeholder="Nh□p mã gi□m giá"
              value={code}
              onChange={(e) => setCode(e.target.value)}
              disabled={isLoading}
              className="h-9"
            />
            {error && <p className="text-xs text-red-500 mt-1">{error}</p>}
          </div>
          <Button
            type="submit"
            disabled={isLoading}
            className="whitespace-nowrap h-9"
          >
            {isLoading ? 'Đang x□ lý...' : 'Áp d□ng'}
```

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

Giải thích:

- Component `DiscountForm` là component đặc biệt được thiết kế để xử lý việc thêm và xóa mã giảm giá trong quá trình thanh toán.
- Quản lý trạng thái nội bộ:
  - Sử dụng `useState` để theo dõi mã giảm giá người dùng nhập vào (`code`)
  - Quản lý thông báo lỗi (`error`) khi người dùng cố gắng áp dụng mã giảm giá không hợp lệ
  - Nhận trạng thái `isLoading` từ component cha để hiển thị trạng thái loading khi đang xử lý mã giảm giá
- Xử lý gửi form:
  - Hàm `handleSubmit` được gọi khi người dùng gửi form nhập mã giảm giá
  - Ngăn chặn hành vi gửi form mặc định với `e.preventDefault()`
  - Kiểm tra mã giảm giá có rỗng không với `!code.trim()`, nếu rỗng, hiển thị thông báo lỗi và kết thúc sớm
  - Nếu mã không rỗng, xóa thông báo lỗi và gọi hàm `onApplyDiscount` được truyền vào qua props để xác minh mã giảm giá
- Xử lý xóa mã giảm giá:
  - Hàm `handleRemoveDiscount` được gọi khi người dùng nhấn nút "Xóa" mã giảm giá đã áp dụng
  - Gọi `onApplyDiscount('')` với chuỗi rỗng để báo hiệu xóa mã giảm giá
  - Đặt lại `code` về chuỗi rỗng để chuẩn bị cho việc nhập mã mới
- Hiển thị điều kiện dựa trên trạng thái:
  - Sử dụng toán tử điều kiện để hiển thị một trong hai giao diện khác nhau:
    1. Nếu `appliedDiscount` tồn tại và hợp lệ (`appliedDiscount && appliedDiscount.isValid`), hiển thị thông tin mã giảm giá đã áp dụng
    2. Nếu không, hiển thị form nhập mã giảm giá
- Hiển thị thông tin mã giảm giá:
  - Khi mã giảm giá đã áp dụng, hiển thị trong khung có nền xanh nhạt (`bg-green-50`) để tạo phản hồi tích cực
  - Hiển thị mã giảm giá và giá trị giảm giá, có thể là phần trăm hoặc số tiền cố định
  - Nút "Xóa" cho phép người dùng dễ dàng xóa mã giảm giá đã áp dụng
- Thiết kế form nhập liệu:
  - Sử dụng layout flex với `flex gap-2` để sắp xếp input và button trên cùng một dòng

    - Input nhận hầu hết không gian với `flex-1`, trong khi button chỉ chiếm không gian cần thiết
    - Hiển thị thông báo lỗi dưới input với màu đỏ (`text-red-500`) khi có lỗi
    - Button "Áp dụng" được vô hiệu hóa khi đang xử lý (`disabled={isLoading}`) và văn bản thay đổi thành "Đang xử lý..." để cung cấp phản hồi
- Trải nghiệm người dùng:
    - Giao diện đơn giản, rõ ràng giúp người dùng dễ dàng áp dụng mã giảm giá
    - Phản hồi tức thì khi mã giảm giá được áp dụng thành công hoặc khi có lỗi
    - Có thể dễ dàng xóa mã giảm giá để thử mã khác

```
```

Component Skeleton (skeleton.tsx)

Imports

```
import * as React from "react"
import { cn } from "@/lib/utils"
```

Chức năng và hoạt động

Skeleton component cung cấp hiệu ứng loading cho trang thanh toán với hai component:

1. **Skeleton**: Component cơ bản với hiệu ứng pulse
2. **OrderProcessingSkeleton**: Component hiển thị giao diện loading khi đơn hàng đang được xử lý

Mã nguồn

```
export function Skeleton({ className, ...props }: SkeletonProps) {
  return (
    <div
      className={cn("animate-pulse rounded-md bg-muted", className)}
      {...props}
    />
  )
}

export function OrderProcessingSkeleton() {
  return (
    <div className="bg-white rounded-lg shadow-md p-6 w-full space-y-6">
      <div className="flex items-center space-x-4">
        <Skeleton className="h-12 w-12 rounded-full" />
        <div className="space-y-2">
```

```
          <Skeleton className="h-4 w-[250px]" />
          <Skeleton className="h-4 w-[200px]" />
        </div>
      </div>

      <Skeleton className="h-4 w-full" />
      <Skeleton className="h-4 w-full" />
      <Skeleton className="h-4 w-3/4" />

      <div className="pt-4 space-y-2">
        <Skeleton className="h-4 w-full" />
        <Skeleton className="h-4 w-full" />
        <Skeleton className="h-4 w-2/3" />
      </div>

      <div className="flex justify-between items-center pt-4">
        <Skeleton className="h-10 w-24 rounded-md" />
        <Skeleton className="h-10 w-40 rounded-md" />
      </div>
    </div>
  )
}
```

Các cải tiến và thay đổi thực hiện

1. Tạo Step Indicator động

Component Step Indicator mới được tạo để hiển thị tiến trình thanh toán với các tính năng:

- Hiển thị trạng thái các bước với màu sắc trực quan
- Cho phép người dùng quay lại các bước đã hoàn thành
- Thêm hiệu ứng animation khi chuyển đổi giữa các bước
- Đánh dấu rõ ràng bước hiện tại và các bước đã hoàn thành

2. Thêm hiệu ứng skeleton loading

- Tạo component OrderProcessingSkeleton cho hiển thị trạng thái loading
- Thêm logic xử lý trạng thái loading trong trang thanh toán
- Animation mờ dần khi chuyển đổi giữa form thông thường và skeleton loading

3. Thêm chức năng mã giảm giá

- Tạo component DiscountForm cho việc nhập và áp dụng mã giảm giá
- Thêm logic xử lý mã giảm giá trong trang thanh toán

- Cập nhật hiển thị tổng đơn hàng khi áp dụng giảm giá
- Cho phép người dùng xóa mã giảm giá đã áp dụng

4. Cải thiện hiệu ứng animation

- Sử dụng framer-motion để tạo hiệu ứng chuyển đổi mượt mà giữa các bước
- Thêm hiệu ứng animation khi người dùng tương tác với các phần tử
- Tạo trải nghiệm người dùng liền mạch khi di chuyển qua các bước thanh toán

5. Tăng cường xác thực form

- Sử dụng React Hook Form và Zod cho xác thực dữ liệu form
- Hiển thị thông báo lỗi rõ ràng khi dữ liệu nhập vào không hợp lệ
- Ngăn người dùng tiến đến bước tiếp theo khi dữ liệu chưa hợp lệ

6. Đồng bộ hóa định nghĩa phương thức thanh toán

- Thống nhất định nghĩa PaymentMethod trên toàn hệ thống
- Đảm bảo hiển thị nhất quán giữa các component
- Sửa các lỗi không tương thích kiểu dữ liệu

Kết luận

Trang thanh toán đã được phát triển với nhiều tính năng hiện đại và trải nghiệm người dùng tốt. Các component được thiết kế để tái sử dụng và dễ dàng mở rộng trong tương lai. Việc sử dụng các thư viện và công nghệ mới nhất như React Hook Form, Zod, và Framer Motion giúp trang thanh toán hoạt động mượt mà và đáng tin cậy.

Các tính năng chính bao gồm: - Quy trình thanh toán với nhiều bước rõ ràng - Xác thực dữ liệu form chặt chẽ - Hiệu ứng animation mượt mà - Chức năng áp dụng mã giảm giá - Hiển thị skeleton loading khi xử lý đơn hàng

Trang thanh toán cung cấp trải nghiệm người dùng chuyên nghiệp và hiện đại, giúp tăng tỷ lệ hoàn thành đơn hàng và cải thiện sự hài lòng của khách hàng.