Báo Cáo Xử Lý Lỗi Trong Validation Schema

Mục Lục

1. Giới thiệu

Báo cáo này trình bày về vấn đề gặp phải khi xác thực form thanh toán (payment form) và cách giải quyết. Vấn đề chính xảy ra khi hệ thống yêu cầu thông tin thẻ tín dụng (cardNumber) ngay cả khi phương thức thanh toán là COD (Cash on Delivery), dẫn đến trải nghiệm người dùng không tốt và lỗi validation không cần thiết.

2. Phân tích vấn đề

2.1. Mô tả lỗi

Trong schema xác thực của form thanh toán (`paymentSchema`), có hai tình huống chính:

1. Người dùng chọn thanh toán qua Stripe: Yêu cầu nhập đầy đủ thông tin thẻ tín dụng

2. Người dùng chọn thanh toán COD (cash_on_delivery): Không cần thông tin thẻ

Tuy nhiên, schema hiện tại yêu cầu `cardNumber` là bắt buộc trong cả hai trường hợp, gây ra lỗi validation không mong muốn.

2.2. Mã nguồn ban đầu

```
export const paymentSchema = z.object({
  paymentMethod: z
    .enum(['stripe', 'cash_on_delivery'], {
      required_error: 'Vui lòng ch□n ph□□ng th□c thanh toán'
    }),
  // Ch□ yêu c□u xác th□c thông tin th□ khi ph□□ng th□c là credit_card
  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(),
}).refine((data) => {
  // Ki□m tra n□u ph□□ng th□c thanh toán là credit_card thì các tr□□ng thông tin th□ ph□i
đ□□c đi□n
  if (data.paymentMethod === 'stripe') {
    return data.cardNumber && data.cardName && data.expDate && data.cvv;
  } else if (data.paymentMethod === 'cash_on_delivery') {
    // N□u ph□□ng th□c thanh toán là cash_on_delivery, không c□n ki□m tra thông tin th□
    return true;
  }
  return false;
}, {
  message: 'Vui lòng đi□n đ□y đ□ thông tin th□ tín d□ng',
  path: ['cardNumber', 'cardName', 'expDate', 'cvv'],
});
```

Vấn đề chính trong mã nguồn trên:

1. Trường `cardNumber` được đánh dấu là `optional()` nhưng sau đó có `refine()` yêu cầu nó phải tồn tại (`!== undefined`) và có ít nhất 16 ký tự
2. Schema sử dụng `refine()` ở cấp đối tượng để xác thực theo điều kiện, nhưng vẫn tồn tại validator nghiêm ngặt ở cấp trường
3. Các trường thông tin thẻ được đánh dấu là `optional()` nhưng thực tế vẫn được yêu cầu khi thanh toán qua Stripe

2.3. Tác động đến người dùng

- Người dùng chọn COD vẫn bị yêu cầu nhập thông tin thẻ tín dụng không cần thiết
- Hiển thị thông báo lỗi không phù hợp với ngữ cảnh
- Trải nghiệm form không trực quan, gây nhầm lẫn

3. Giải pháp

3.1. Chiến lược giải quyết

Để giải quyết vấn đề hiệu quả, ta sử dụng tính năng "Discriminated Union" của Zod, cho phép xác thực dựa trên giá trị của một trường cụ thể.

3.1.1. Ưu điểm của Discriminated Union

1. **Tách biệt schema theo ngữ cảnh**: Mỗi phương thức thanh toán có schema riêng
2. **Xác thực chính xác**: Yêu cầu các trường đầu vào khác nhau tùy theo phương thức
3. **Mã nguồn rõ ràng**: Dễ đọc, dễ bảo trì hơn
4. **Type-safe**: TypeScript có thể suy luận kiểu dữ liệu chính xác hơn

3.2. Mã nguồn sau khi sửa lỗi

```
// Đ□nh nghĩa các schema riêng cho t□ng ph□□ng th□c thanh toán
const basePaymentSchema = {
  paymentMethod: z.enum(['stripe', 'cash_on_delivery'], {
    required_error: 'Vui lòng ch□n ph□□ng th□c thanh toán'
  }),
};

const stripePaymentSchema = z.object({
  ...basePaymentSchema,
  paymentMethod: z.literal('stripe'),
  cardNumber: z
    .string()
    .min(16, { message: 'S□ th□ không h□p l□' })
    .max(19, { message: 'S□ th□ không h□p l□' }),
  cardName: z.string().min(2, { message: 'Tên trên th□ không h□p l□' }),
  expDate: z.string().min(4, { message: 'Ngày h□t h□n không h□p l□' }),
  cvv: z.string().min(3, { message: 'CVV không h□p l□' }),
});

const codPaymentSchema = z.object({
  ...basePaymentSchema,
  paymentMethod: z.literal('cash_on_delivery'),
});
```

```
// K□t h□p các schema d□a trên discriminator là paymentMethod
export const paymentSchema = z.discriminatedUnion('paymentMethod', [
  stripePaymentSchema,
  codPaymentSchema,
]);
```

4. Cách triển khai

4.1. 1. Tạo Base Schema

Đầu tiên, tạo schema cơ sở định nghĩa trường `paymentMethod` làm phân biệt (discriminator):

```
const basePaymentSchema = {
  paymentMethod: z.enum(['stripe', 'cash_on_delivery'], {
    required_error: 'Vui lòng ch□n ph□□ng th□c thanh toán'
  }),
};
```

4.2. 2. Định nghĩa Schema riêng cho Stripe

Schema dành cho thanh toán Stripe yêu cầu đầy đủ thông tin thẻ:

```
const stripePaymentSchema = z.object({
  ...basePaymentSchema,
  paymentMethod: z.literal('stripe'),
  cardNumber: z
    .string()
    .min(16, { message: 'S□ th□ không h□p l□' })
    .max(19, { message: 'S□ th□ không h□p l□' }),
  cardName: z.string().min(2, { message: 'Tên trên th□ không h□p l□' }),
  expDate: z.string().min(4, { message: 'Ngày h□t h□n không h□p l□' }),
  cvv: z.string().min(3, { message: 'CVV không h□p l□' }),
});
```

4.3. 3. Định nghĩa Schema riêng cho COD

Schema dành cho COD không yêu cầu thông tin thẻ:

```
const codPaymentSchema = z.object({
  ...basePaymentSchema,
  paymentMethod: z.literal('cash_on_delivery'),
});
```

4.4. 4. Kết hợp các Schema với Discriminated Union

```
export const paymentSchema = z.discriminatedUnion('paymentMethod', [
  stripePaymentSchema,
  codPaymentSchema,
]);
```

4.5. 5. Thay đổi trong Component

Component hiển thị form cần cập nhật để phản ánh cấu trúc schema mới:

- Chỉ hiển thị các trường thông tin thẻ khi phương thức thanh toán là Stripe
- Reset các trường thẻ khi chuyển sang phương thức COD
- Xử lý form submit theo phương thức thanh toán

5. Kết quả

5.1. Cải thiện trải nghiệm người dùng

1. **Form thông minh**: Chỉ hiển thị các trường phù hợp với phương thức thanh toán
2. **Thông báo lỗi chính xác**: Người dùng chỉ thấy lỗi liên quan đến phương thức đã chọn
3. **Quy trình mượt mà**: Người dùng chọn COD có thể tiếp tục ngay mà không cần điền thông tin thẻ

5.2. Cải thiện chất lượng mã nguồn

1. **Tăng tính tái sử dụng**: Dễ dàng thêm phương thức thanh toán mới
2. **Dễ bảo trì**: Mỗi phương thức thanh toán có schema riêng, dễ chỉnh sửa
3. **An toàn về kiểu dữ liệu**: TypeScript có thể suy luận đúng kiểu dữ liệu dựa trên phương thức thanh toán

6. Kết luận

Việc sử dụng tính năng Discriminated Union của Zod là một giải pháp hiệu quả để xử lý xác thực có điều kiện. Thay vì sử dụng `refine()` để kiểm tra điều kiện, việc tách schema theo từng trường hợp sử dụng giúp mã nguồn dễ đọc hơn, dễ bảo trì hơn và cung cấp trải nghiệm người dùng tốt hơn.

Cách tiếp cận này còn mở rộng dễ dàng cho các phương thức thanh toán mới trong tương lai, chỉ cần thêm schema mới và đưa vào discriminated union.

7. Tài liệu tham khảo

- Zod Documentation
- Zod Discriminated Unions
- React Hook Form