The Coupon Challenge

Business requirement:

The SkiNet app needs a **coupon** feature. Customers should be able to apply coupons at checkout or in their **cart** view, see discounted prices, and receive error messages for invalid coupons. The feature must ensure quick validation, security of coupon codes, and a user-friendly interface.

General guidance and research

You have been given this task to complete. The HTML for the voucher is already in place and it is “just” missing the functionality.

1. Usage of the Stripe API docs will be necessary to complete this task.

Coupon API

Promotion Code API

2. Start from bottom and work your way up. Core ⇒ Infrastructure ⇒ API ⇒ Client
3. If you run into trouble and wish to start from scratch, then you can use either of the following approaches to remove the changes you have made to your code:

```
# Approach 1
git clean -df # removes untracked changes and folders
git checkout -- . # ERASE changes in tracked files (in the current directory)

# Approach 2
git add . # stages the changes
git commit "SadFaceEmoji"
git reset --hard HEAD~1 # remove changes to commit
```

4. Use the demo project here to see it in action.

Steps

Stripe Dashboard

Go to Product Catalog ⇒ Coupons ⇒ Create new coupon.

Create 2 coupons that both have a customer facing coupon code. We will be supporting both percentage discount and fixed amount off in the app.

Suggested coupons:

GIMME10 - 10% discount

GIMME5 - $5 discount

Core project

1. Create a new Class for the Coupon (hint: Stripe already has a type for Coupon so use ‘AppCoupon’ to make life easier) that contains properties for the **Name**, **AmountOff**?, **PercentOff**?, **PromotionCode** and **CouponId.** Ensure the properties are the same types that Stripe uses (decimals for AmountOff and PercentOff)
2. Add this as a new property into the **ShoppingCart**
3. Create an interface called ‘ICouponService’ that has a single method called **GetCouponFromPromoCode(string code)**
4. Update the **Order** class and add a **Discount** property that is a decimal.
5. Update the **GetTotal** method in the Order class.

Infrastructure project

1. Create a new class in the services folder called **CouponService** that implements the **ICouponService** interface. Use the stripe PromotionCodeService() to get the coupon from the promotion code and use that to

return the **AppCoupon**

2. Update the PaymentIntent to accommodate the Discount. In the payment service we have quite a big method for the **CreateOrUpdatePaymentIntent**. This is a good opportunity to refactor this into small functions as we need to add extra here for the coupon functionality. Here is the shell of what would be an improved structure for this code. Note that none of these private methods return null so we need to use exceptions to avoid null ref warnings.

```
public async Task<ShoppingCart?> CreateOrUpdatePaymentIntent(string cartId)
{
    StripeConfiguration.ApiKey = config["StripeSettings:SecretKey"];

    var cart = await cartService.GetCartAsync(cartId)
        ?? throw new Exception("Cart unavailable");

    var shippingPrice = await GetShippingPriceAsync(cart) ?? 0;

    await ValidateCartItemsInCartAsync(cart);

    var subtotal = CalculateSubtotal(cart);

    if (cart.Coupon != null)
    {
        subtotal = await ApplyDiscountAsync(cart.Coupon, subtotal);
    }

    var total = subtotal + shippingPrice;

    await CreateUpdatePaymentIntentAsync(cart, total);

    await cartService.SetCartAsync(cart);

    return cart;
}

private async Task CreateUpdatePaymentIntentAsync(ShoppingCart cart, long amount)
{
    throw new NotImplementedException();
}

private async Task<long> ApplyDiscountAsync(ShoppingCart cart, long amount)
{
    // hint: we have the coupon id in the cart. Consider Stripe.CouponService();
    // hint:
      // var discount = amount * (coupon.PercentOff.Value / 100);
    // amount -= (long)discount;

    throw new NotImplementedException();
}

private long CalculateSubtotal(ShoppingCart cart)
{
    throw new NotImplementedException();
}

private async Task ValidateCartItemsInCartAsync(ShoppingCart cart)
{
        // hint: throw exception if missing product
    throw new NotImplementedException();
}
```

```
    private async Task<long?> GetShippingPriceAsync(ShoppingCart cart)
    {
            // hint: throw exception if cannot find delivery method
            // return null if cart does not have it set
        throw new NotImplementedException();
    }
```

3. Implement the **ApplyDiscountAsync** method to update the intent with the discounted price.
4. Update the **OrderConfiguration** to accommodate the new decimal property for the **Discount**
5. Create a new EF migration at this point to update the DB schema

API Project

1. Update the **Program.cs** to register the new **ICouponService** and its implementation class.
2. Update the **OrderDto** to include the **Discount** property
3. Update the **OrderMappingExtensions** to incorporate this Discount property.
4. Create a CouponsController that has the following endpoint:

```
[HttpGet("{code}")]
public async Task<ActionResult<AppCoupon>> ValidateCoupon(string code)
{
    // implmement the logic for this
    // return bad request for invalid coupon
}
```

5. Run the request in Postman from section 20 to ensure you can validate and return the coupon.
6. Make sure you return a BadRequest for the second request
7. Update the **CreateOrderDto** to take an optional Discount property that is a decimal.
8. Update the **OrdersController** to use this new property.
9. Certain order amounts may cause a rounding difference when comparing the Order amount to the Intent amount. This function if added to the webhook will ensure that the amounts are using the same rounding method:

```
var orderTotalInCents = (long)Math.Round(order.GetTotal() * 100, MidpointRounding.AwayFr
```

Client project

1. Create a type for the **Coupon** in the **cart.ts** file.
2. In the **cart.service.ts** use the **coupon** from the **cart** to update the **totals** signal
3. In the **cart.service.ts** add an **applyDiscount(code: string): Observable<Coupon>** method to validate the **coupon.**
4. In the **order-summary.component.ts** create and implement the following 2 methods:

```
  applyCouponCode(): void {
        // set the cart with the coupon if valid
        // if in checkout update the payment intent (hint: this returns an observable so
  }

  removeCouponCode(): void {
      // remove coupon from cart
      // if in checkout update the payment intent (hint: this returns an observable so us
  }
```

5. Check the **createOrUpdatePaymentIntent** method in the **stripe.service.ts.** Will this cause a problem (hint: yes)? What can we do to prevent the cart being updated if we already have the clientSecret and paymentIntentId?
5. Use the FormsModule from Angular to use 2 way binding to update a **code** property in the **order-summary.component.ts**
6. Use the **ngSubmit** function from the FormsModule in the form to call the **applyDiscount** method from the template.
7. Disable the input if we have a coupon in the cart
8. Disable the button if we have a coupon in the cart
9. Display the name of the coupon that has been applied above the input. Provide an icon button to remove the applied coupon that calls the **removeCouponCode** method.
10. Update the **Order** interface and the **OrderToCreate** in the **order.ts** to include the **discount**
11. Update the **order-detailed.component.html** to display this
12. In the **checkout.component.ts** update the **createOrderModel** method to include the discount when creating the order.
13. Test the new functionality

Challenge complete! Publish changes to production