The Coupon Challenge

Business requirement:

The Re-Store app needs a coupon feature. Customers should be able to apply coupons at the checkout page, 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. They should also be able to remove an applied coupon.

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. Entity classes ⇒ Services ⇒ Controller ⇒ 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 -m "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

Git

1. Create a new branch in your Git repo for this. Following the last lesson you will have changes in the GitHub repo that need to be pulled down into the local Git repository so do this first:

```
# pull changes to local git repo.  Switch back to main branch first
git checkout main

# then pull the changes into the main branch
git pull
```

2. Then create and checkout a new branch to work from:

```
# create and checkout new branch
git checkout -b coupons
```

.Net Project

1. Create a new **owned** entity class for the Coupon which will be owned by the basket (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 (nullable long for AmountOff and nullable decimal for PercentOff). For the Percentage then Stripe uses a precision of 5 (number of digits) and 2 (number of digits after the decimal point) which accommodates a range of 0.00 to 100.00. Hint: Read about how to use Data annotations to configure precision here.
2. Add the AppCoupon as a optional new property in the Basket
3. Update the **BasketDto** to include the **AppCoupon** (optional - you can remove the PaymentIntentId from this as it is not required to send this to the client)
4. Update the **BasketExtensions** to ensure the **AppCoupon** is returned with the **BasketDto**
5. Create a new migration to update the DB schema with these changes. Don't forget to check the migration for accuracy.
6. Create a new Service called **DiscountService** that has the 2 methods populated below

```
using System;
using API.Entities;
using Stripe;

namespace API.Services;

public class DiscountService
{
    public DiscountService(IConfiguration config)
    {
        StripeConfiguration.ApiKey = config["StripeSettings:SecretKey"];
    }

    public async Task<AppCoupon?> GetCouponFromPromoCode(string code)
    {
        var promotionService = new PromotionCodeService();

        // create and return the AppCoupon
    }

    public async Task<long> CalculateDiscountFromAmount(AppCoupon appCoupon, long amount,
        bool removeDiscount = false)
    {
        var couponService = new CouponService();

                // Calculate and return the amount in long format for both the Amount
                // off and Percent off.

                // As casting to Long will automatically truncate rather than use roundi
                // can use the following logic for the calculation for percentage:
                // return (long)Math.Round(amount * (coupon.PercentOff.Value / 100),
                //                   MidpointRounding.AwayFromZero)
    }
}
```

7. Add this new service to the **Program** class.
8. Update the **PaymentService** to accommodate the **Discount**. This needs to accommodate the calculation of the discount if a coupon is being added as well as the removal of the discount if the coupon is being removed.
9. Add 2 endpoints to the BasketController. In both endpoints ensure the basket has the client secret. We are only allowing usage of coupon at checkout stage so the PaymentIntent should already be created.

```
[HttpPost("{code}")]
public async Task<ActionResult<BasketDto>> AddCouponCode(string code)
```

```
{
    // get the basket and check to ensure it has the client secret
    // get the coupon
    // update the basket with the coupon
    // update the payment intent
    // save changes and return BasketDto if successful
}

[HttpDelete("remove-coupon")]
public async Task<ActionResult> RemoveCouponFromBasket()
{
    // get the basket and check to ensure it has the client secret
    // update the payment intent
    // Remove the coupon from the basket (set it to null)
    // save changes and return Ok() if successful
}
```

10. Update the **OrdersController CreateOrder()** method to calculate the discount amount using the **DiscountService** and update the Order with this.
11. Update the **OrderExtensions** to include the **Discount** property as well in both the **ProjectToDto** and **ToDTO** methods:
10. Execute the Postman requests in the challenge section.   You will need to create a payment intent before you can apply the coupon so please ensure you execute these requests in order.   Recommend removing the BasketId cookie from Postman if you have one there so you start with a clean basket. Do not move onto the client without these checks working as expected.
11. Ensure you get the bad request for the Invalid coupon request in postman

Client Project

1. Create a type for Coupon in the basket.ts file (optional: you may want to simplify this and just use a Type rather than class as this did not work for use anyway in the Redux store.
2. Add 2 new endpoints to the basketApi.ts to add and remove a coupon. Use the updateQueryData method to update the basket with the response back from the API in the case of adding a coupon, and setting the basket.coupon to null when removing a coupon.
3. Update the **useBasket** hook to calculate the discount if the basket has a coupon and return a **discount** property from this hook for use in other components. Hint: For the percentage calculation use the following to ensure rounding consistency with the .Net API:

```
discount = Math.round((subtotal * (basket.coupon.percentOff / 100)) * 100) / 100;
```

4. Use the **discount** property in the **OrderSummary.tsx**
5. Use a conditional in the **OrderSummary.tsx** to ensure the Voucher is only visible to the user when at the checkout stage.
6. Update the OrderSummary to use the new basketApi.ts hooks for adding and removing the coupon and use react-hook-form to provide the form functionality. Creating a schema is optional (and would be overkill) as we only have a single input form for our vouchers.
7. Update the UI to display the applied voucher when one is added, disable the button and the text input when a voucher is applied, and give them the option to remove the voucher.   The UI should update in response to these events without the user needing to refresh the page.
8. Test it all works!
9. When satisfied it is working stage and commit the changes to GitHub, and then create a pull request to then merge the changes into the main branch.   Then test it is all still working in production.
10. Challenge complete!