

The Coupon Challenge - Solution

.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). Stripe uses a precision of 5,2 for the decimal. Read about the Precision data annotation here: <https://learn.microsoft.com/en-us/ef/core/modeling/entity-properties?tabs=data-annotations%2Cwithout-nrt#precision-and-scale>

```
using Microsoft.EntityFrameworkCore;

namespace API.Entities;

[Owned]
public class AppCoupon
{
    public required string Name { get; set; }
    public long? AmountOff { get; set; }

    [Precision(5,2)]
    public decimal? PercentOff { get; set; }
    public required string PromotionCode { get; set; }
    public required string CouponId { get; set; }
}
```

2. Add the AppCoupon as a optional new property in the Basket

```
using System;

namespace API.Entities;

public class Basket
{
    public int Id { get; set; }
    public required string BasketId { get; set; }
    public List<BasketItem> Items { get; set; } = [];
    public string? ClientSecret { get; set; }
    public string? PaymentIntentId { get; set; }
    public AppCoupon? Coupon { get; set; }

    // Basket methods omitted
}
```

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)

```
using System;
using API.Entities;

namespace API.DTOs;

public class BasketDto
{
    public required string BasketId { get; set; }
    public List<BasketItemDto> Items { get; set; } = [];
    public string? ClientSecret { get; set; }
```

```
    public AppCoupon? Coupon { get; set; }  
}
```

4. Update the BasketExtensions to include this property:

```
using System;  
using API.DTOS;  
using API.Entities;  
using Microsoft.EntityFrameworkCore;  
  
namespace API.Extensions;  
  
public static class BasketExtensions  
{  
    public static BasketDto ToDto(this Basket basket)  
    {  
        return new BasketDto  
        {  
            BasketId = basket.BasketId,  
            ClientSecret = basket.ClientSecret,  
            Coupon = basket.Coupon,  
            Items = basket.Items.Select(x => new BasketItemDto  
            {  
                ProductId = x.ProductId,  
                Name = x.Product.Name,  
                Price = x.Product.Price,  
                Brand = x.Product.Brand,  
                Type = x.Product.Type,  
                PictureUrl = x.Product.PictureUrl,  
                Quantity = x.Quantity  
            }).ToList()  
        };  
    }  
  
    public static async Task<Basket> GetBasketWithItems(this IQueryable<Basket> query,  
        string? basketId)  
    {  
        return await query  
            .Include(x => x.Items)  
            .ThenInclude(x => x.Product)  
            .FirstOrDefaultAsync(x => x.BasketId == basketId)  
            ?? throw new Exception("Cannot get basket");  
    }  
}
```

5. Create a new migration to update the DB schema with these changes. Don't forget to check the migration for accuracy.

```
dotnet ef migrations add CouponsAdded  
  
// migration that was created  
  
using Microsoft.EntityFrameworkCore.Migrations;  
  
#nullable disable  
  
namespace API.Data.Migrations  
{  
    /// <inheritdoc />
```

```

public partial class CouponAdded : Migration
{
    /// <inheritdoc />
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.AddColumn<long>(
            name: "Coupon_AmountOff",
            table: "Baskets",
            type: "bigint",
            nullable: true);

        migrationBuilder.AddColumn<string>(
            name: "Coupon_CouponId",
            table: "Baskets",
            type: "nvarchar(max)",
            nullable: true);

        migrationBuilder.AddColumn<string>(
            name: "Coupon_Name",
            table: "Baskets",
            type: "nvarchar(max)",
            nullable: true);

        migrationBuilder.AddColumn<decimal>(
            name: "Coupon_PercentOff",
            table: "Baskets",
            type: "decimal(5,2)",
            precision: 5,
            scale: 2,
            nullable: true);

        migrationBuilder.AddColumn<string>(
            name: "Coupon_PromotionCode",
            table: "Baskets",
            type: "nvarchar(max)",
            nullable: true);
    }

    /// <inheritdoc />
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropColumn(
            name: "Coupon_AmountOff",
            table: "Baskets");

        migrationBuilder.DropColumn(
            name: "Coupon_CouponId",
            table: "Baskets");

        migrationBuilder.DropColumn(
            name: "Coupon_Name",
            table: "Baskets");

        migrationBuilder.DropColumn(
            name: "Coupon_PercentOff",
            table: "Baskets");

        migrationBuilder.DropColumn(
            name: "Coupon_PromotionCode",
            table: "Baskets");
    }
}

```

```

    }
}
}

```

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();

        var options = new PromotionCodeListOptions
        {
            Code = code
        };

        var promotionCodes = await promotionService.ListAsync(options);

        var promotionCode = promotionCodes.FirstOrDefault();

        if (promotionCode != null && promotionCode.Coupon != null)
        {
            return new AppCoupon
            {
                Name = promotionCode.Coupon.Name,
                AmountOff = promotionCode.Coupon.AmountOff,
                PercentOff = promotionCode.Coupon.PercentOff,
                CouponId = promotionCode.Coupon.Id,
                PromotionCode = promotionCode.Code
            };
        }

        return null;
    }

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

        var coupon = await couponService.GetAsync(appCoupon.CouponId);

        if (coupon.AmountOff.HasValue && !removeDiscount)
        {
            return (long)coupon.AmountOff;
        }
        else if (coupon.PercentOff.HasValue && !removeDiscount)

```

```

        {
            return (long)Math.Round(amount * (coupon.PercentOff.Value / 100),
                                    MidpointRounding.AwayFromZero);
        }

        return 0;
    }
}

```

7. Add this new Service to the Program class so it is available for Dependency injection:

```

using API.RequestHelpers;
using API.Services;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

// other services omitted
builder.Services.AddScoped<DiscountService>();

```

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.

```

using API.Entities;
using Stripe;

namespace API.Services;

public class PaymentsService(IConfiguration config, DiscountService discountService)
{
    public async Task<PaymentIntent> CreateOrUpdatePaymentIntent(Basket basket,
        bool removeDiscount = false)
    {
        StripeConfiguration.ApiKey = config["StripeSettings:SecretKey"];

        var service = new PaymentIntentService();

        var intent = new PaymentIntent();
        long subtotal = basket.Items.Sum(x => x.Quantity * x.Product.Price);
        long deliveryFee = subtotal > 10000 ? 0 : 500;
        long discount = 0;

        if (basket.Coupon != null)
        {
            discount = await discountService.CalculateDiscountFromAmount(basket.Coupon,
                subtotal, removeDiscount);
        }

        var totalAmount = subtotal - discount + deliveryFee;

        if (string.IsNullOrEmpty(basket.PaymentIntentId))
        {
            var options = new PaymentIntentCreateOptions
            {
                Amount = totalAmount,
                Currency = "usd",
            }

```

```

        PaymentMethodTypes = ["card"]
    };
    intent = await service.CreateAsync(options);
}
else
{
    var options = new PaymentIntentUpdateOptions
    {
        Amount = totalAmount
    };
    await service.UpdateAsync(basket.PaymentIntentId, options);
}

    return intent;
}
}

```

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.

```

using System;
using API.Data;
using API.DTOS;
using API.Entities;
using API.Extensions;
using API.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace API.Controllers;

public class BasketController(StoreContext context,
    DiscountService couponService, PaymentsService paymentsService) : BaseApiController
{
    // other endpoints omitted

    [HttpPost("{code}")]
    public async Task<ActionResult<BasketDto>> AddCouponCode(string code)
    {
        // get the basket
        var basket = await RetrieveBasket();
        if (basket == null || string.IsNullOrEmpty(basket.ClientSecret))
            return BadRequest("Unable to apply voucher");

        // get the coupon
        var coupon = await couponService.GetCouponFromPromoCode(code);
        if (coupon == null) return BadRequest("Invalid coupon");

        // update the basket with the coupon
        basket.Coupon = coupon;

        // update the payment intent
        var intent = await paymentsService.CreateOrUpdatePaymentIntent(basket);
        if (intent == null) return BadRequest("Problem applying coupon to basket");

        // save changes and return BasketDto if successful
        var result = await context.SaveChangesAsync() > 0;

        if (result) return CreatedAtAction(nameof(GetBasket), basket.ToDto());
    }
}

```

```

        return BadRequest("Problem updating basket");
    }

    [HttpDelete("remove-coupon")]
    public async Task<ActionResult> RemoveCouponFromBasket()
    {
        // get the basket
        var basket = await RetrieveBasket();
        if (basket == null || basket.Coupon == null
            || string.IsNullOrEmpty(basket.ClientSecret))
            return BadRequest("Unable to update basket with coupon");

        var intent = await paymentsService.CreateOrUpdatePaymentIntent(basket, true);
        if (intent == null) return BadRequest("Problem removing coupon from basket");

        basket.Coupon = null;

        var result = await context.SaveChangesAsync() > 0;

        if (result) return Ok();

        return BadRequest("Problem updating basket");
    }

    private Basket CreateBasket()
    {
        var basketId = Guid.NewGuid().ToString();
        var cookieOptions = new CookieOptions
        {
            IsEssential = true,
            Expires = DateTime.UtcNow.AddDays(30)
        };
        Response.Cookies.Append("basketId", basketId, cookieOptions);
        var basket = new Basket { BasketId = basketId };
        context.Baskets.Add(basket);
        return basket;
    }

    private async Task<Basket?> RetrieveBasket()
    {
        return await context.Baskets
            .Include(x => x.Items)
            .ThenInclude(x => x.Product)
            .FirstOrDefaultAsync(x => x.BasketId == Request.Cookies["basketId"]);
    }
}

```

10. Update the OrdersController CreateOrder method to calculate the discount amount using the Discount service and populate this in the Order entity

```

// using statements omitted

namespace API.Controllers;

[Authorize]
public class OrdersController(StoreContext context, DiscountService discountService) : BaseApiController
{
    // other endpoints omitted
}

```

```

[HttpPost]
public async Task<ActionResult<Order>> CreateOrder(CreateOrderDto orderDto)
{
    var basket = await context.Baskets.GetBasketWithItems(Request.Cookies["basketId"]);

    if (basket == null || basket.Items.Count == 0
        || string.IsNullOrEmpty(basket.PaymentIntentId))
        return BadRequest("Basket is empty or not found");

    var items = CreateOrderItems(basket.Items);
    if (items == null) return BadRequest("Some items out of stock");

    var subtotal = items.Sum(x => x.Price * x.Quantity);
    var deliveryFee = CalculateDeliveryFee(subtotal);
    long discount = 0;

    if (basket.Coupon != null)
    {
        discount = await discountService.CalculateDiscountFromAmount(basket.Coupon,
            subtotal);
    }

    var order = await context.Orders
        .Include(x => x.OrderItems)
        .FirstOrDefaultAsync(x => x.PaymentIntentId == basket.PaymentIntentId);

    if (order == null)
    {
        order = new Order
        {
            OrderItems = items,
            BuyerEmail = User.GetUsername(),
            ShippingAddress = orderDto.ShippingAddress,
            DeliveryFee = deliveryFee,
            Subtotal = subtotal,
            Discount = discount,
            PaymentSummary = orderDto.PaymentSummary,
            PaymentIntentId = basket.PaymentIntentId
        };

        context.Orders.Add(order);
    }
    else
    {
        order.OrderItems = items;
    }

    var result = await context.SaveChangesAsync() > 0;

    if (!result) return BadRequest("Problem creating order");

    return CreatedAtAction(nameof(GetOrderDetails), new { id = order.Id },
        order.ToDto());
}

// helper methods omitted
}

```


11. Update the OrderExtensions to include the Discount property as well in both the ProjectToDto and ToDTO methods:

```
using System;
using API.DTOs;
using API.Entities.OrderAggregate;
using Microsoft.EntityFrameworkCore;

namespace API.Extensions;

public static class OrderExtensions
{
    public static IQueryable<OrderDto> ProjectToDto(this IQueryable<Order> query)
    {
        return query.Select(order => new OrderDto
        {
            Id = order.Id,
            BuyerEmail = order.BuyerEmail,
            OrderDate = order.OrderDate,
            ShippingAddress = order.ShippingAddress,
            PaymentSummary = order.PaymentSummary,
            DeliveryFee = order.DeliveryFee,
            Subtotal = order.Subtotal,
            Discount = order.Discount,
            OrderStatus = order.OrderStatus.ToString(),
            Total = order.GetTotal(),
            OrderItems = order.OrderItems.Select(item => new OrderItemDto
            {
                ProductId = item.ItemOrdered.ProductId,
                Name = item.ItemOrdered.Name,
                PictureUrl = item.ItemOrdered.PictureUrl,
                Price = item.Price,
                Quantity = item.Quantity
            }).ToList()
        }).AsNoTracking();
    }

    public static OrderDto ToDto(this Order order)
    {
        return new OrderDto
        {
            Id = order.Id,
            BuyerEmail = order.BuyerEmail,
            OrderDate = order.OrderDate,
            ShippingAddress = order.ShippingAddress,
            PaymentSummary = order.PaymentSummary,
            DeliveryFee = order.DeliveryFee,
            Subtotal = order.Subtotal,
            Discount = order.Discount,
            OrderStatus = order.OrderStatus.ToString(),
            Total = order.GetTotal(),
            OrderItems = order.OrderItems.Select(item => new OrderItemDto
            {
                ProductId = item.ItemOrdered.ProductId,
                Name = item.ItemOrdered.Name,
                PictureUrl = item.ItemOrdered.PictureUrl,
                Price = item.Price,
                Quantity = item.Quantity
            }).ToList()
        }
    }
}
```

```

    };
  }
}

```

- 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.

Client Project

- Create a type for Coupon in the basket.ts file

```

export type Basket = {
  basketId: string
  items: Item[]
  clientSecret?: string
  paymentIntentId?: string
  coupon: Coupon | null
}

export type Item = {
  productId: number
  name: string
  price: number
  pictureUrl: string
  brand: string
  type: string
  quantity: number
}

export type Coupon = {
  name: string;
  amountOff?: number;
  percentOff?: number;
  promotionCode: string;
  couponId: string;
}

```

- 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.

```

import { createApi } from "@reduxjs/toolkit/query/react";
import { baseQueryWithErrorHandling } from "../../app/api/baseApi";
import { Basket, Item } from "../../app/models/basket";
import { Product } from "../../app/models/product";
import Cookies from 'js-cookie';

function isBasketItem(product: Product | Item): product is Item {
  return (product as Item).quantity !== undefined;
}

export const basketApi = createApi({
  reducerPath: 'basketApi',
  baseQuery: baseQueryWithErrorHandling,
  tagTypes: ['Basket'],
  endpoints: (builder) => ({
    // other endpoints omitted

```

```

    addCoupon: builder.mutation<Basket, string>({
      query: (code: string) => ({
        url: `basket/${code}`,
        method: 'POST'
      }),
      onQueryStarted: async (_, {dispatch, queryFulfilled}) => {
        const {data: updatedBasket} = await queryFulfilled;

        dispatch(basketApi.util.updateQueryData('fetchBasket',
          undefined, (draft)=> {
            Object.assign(draft, updatedBasket)
          })
        ))
      }
    }),
    removeCoupon: builder.mutation<Basket, void>({
      query: () => ({
        url: 'basket/remove-coupon',
        method: 'DELETE'
      }),
      onQueryStarted: async (_, {dispatch, queryFulfilled}) => {
        await queryFulfilled;

        dispatch(basketApi.util.updateQueryData('fetchBasket', undefined,
          (draft)=> {
            draft.coupon = null
          })
        ))
      }
    })
  })
});

export const { useFetchBasketQuery, useAddBasketItemMutation,
  useAddCouponMutation, useRemoveCouponMutation,
  useRemoveBasketItemMutation, useClearBasketMutation } = basketApi;

```

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.

```

import { Item } from "../../app/models/basket";
import { useClearBasketMutation, useFetchBasketQuery } from "../../features/basket/basketApi";

export const useBasket = () => {
  const {data: basket} = useFetchBasketQuery();
  const [clearBasket] = useClearBasketMutation();

  const subtotal = basket?.items.reduce((sum: number, item: Item) =>
    sum + item.quantity * item.price, 0) ?? 0;
  const deliveryFee = subtotal > 10000 ? 0 : 500;

  let discount = 0;

  if (basket?.coupon) {
    if (basket.coupon.amountOff) {
      discount = basket.coupon.amountOff
    } else if (basket.coupon.percentOff) {
      discount = Math.round((subtotal *
        (basket.coupon.percentOff / 100)) * 100) / 100;
    }
  }
}

```

```

    }

    const total = Math.round(
      (subtotal - discount + deliveryFee) * 100) / 100;

    return {basket, subtotal, deliveryFee, discount, total, clearBasket}
  }

```

4. Use the **discount** property in the **OrderSummary.tsx**

```

export default function OrderSummary() {
  const {subtotal, deliveryFee, discount} = useBasket();
  const location = useLocation();

  // omitted

  <Box display="flex" justifyContent="space-between" mb={1}>
    <Typography color="textSecondary">Discount</Typography>
    <Typography color="success">
      -{currencyFormat(discount)}
    </Typography>
  </Box>

```

5. Use a conditional in the **OrderSummary.tsx** to ensure the Voucher is only visible to the user when at the checkout stage.

```

    {/* Coupon Code Section */}
    {location.pathname.includes('checkout') &&
    <Paper sx={{ width: '100%', borderRadius: 3, p: 3 }}>

      <form>
        <Typography variant="subtitle1" component="label">
          Do you have a voucher code?
        </Typography>

        <TextField
          label="Voucher code"
          variant="outlined"
          fullWidth
          sx={{ my: 2 }}
        />

        <Button
          type="submit"
          variant="contained"
          color="primary"
          fullWidth
        >
          Apply code
        </Button>
      </form>
    </Paper>}
  </Box>

```

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.

```
import { Box, Typography, Divider, Button, TextField, Paper } from "@mui/material";
import { currencyFormat } from "../../lib/util";
import { Link, useLocation } from "react-router-dom";
import { useBasket } from "../../lib/hooks/useBasket";
import { FieldValues, useForm } from "react-hook-form";
import { LoadingButton } from "@mui/lab";
import { useAddCouponMutation, useRemoveCouponMutation } from "../../features/basket/";
import { Delete } from "@mui/icons-material";

export default function OrderSummary() {
  const {subtotal, deliveryFee, discount, basket, total} = useBasket();
  const location = useLocation();
  const {register, handleSubmit, formState: {isSubmitting}} = useForm();
  const [addCoupon] = useAddCouponMutation();
  const [removeCoupon, {isLoading}] = useRemoveCouponMutation();

  const onSubmit = async (data: FieldValues) => {
    await addCoupon(data.code);
  }

  // other code omitted
}
```

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.

```
// other order summary code omitted

{/* Coupon Code Section */}
{location.pathname.includes('checkout') &&
<Paper sx={{ width: '100%', borderRadius: 3, p: 3 }}>

  <form onSubmit={handleSubmit(onSubmit)}>
    <Typography variant="subtitle1" component="label">
      Do you have a voucher code?
    </Typography>

    {basket?.coupon &&
    <Box display='flex' justify-content='space-between' align-items='center'>
      <Typography font-weight='bold' variant='body2'>
        {basket.coupon.name} applied
      </Typography>
      <LoadingButton loading={isLoading} onClick={() => removeCoupon()}>
        <Delete color="error" />
      </LoadingButton>
    </Box>}

    <TextField
      label="Voucher code"
      variant="outlined"
      fullWidth
      disabled={!basket?.coupon}
      {...register('code', {required: 'Voucher code missing'})}
      sx={{ my: 2 }}
    />

    <LoadingButton
      loading={isSubmitting}
```

```
        type="submit"
        variant="contained"
        color="primary"
        fullWidth
        disabled={!basket?.coupon}
      >
        Apply code
      </LoadingButton>
    </form>
  </Paper>}
</Box>
```

8. Test it all works!
9. When satisfied it is working stage and commit changes to GitHub and then create a Pull Request to then merge the changes into the main branch and ensure the functionality works in production.
10. Challenge complete!