Project Overview

The project is implemented in C# and .NET platform. It has a checkout controller which implements the API to consume a list of ID's as string and produces an object which specifies the price of the items after applying the two offers if applicable.

Checkout Flow

To implement the checkout flow multiple services have been created.

- Firstly, the cart service creates a cart using the information from product service.
- The product service functions as a product catalogue and gives product information for valid Id's – 001,002,003,004.
- Once the cart is created with 1 or more items, the checkout service calls the billing service to get the bill amount.
- The billing service calculates the actual value and calls the discount service to get the discount amount for the specified cart.
- The discount service gets the current offers from offer service and uses that to calculate discount on the given cart.
- This total calculated amount by the billing service is then returned by the checkout service to the checkout controller, that creates a response object from this.
- In case there is no valid product ID provided, the controller responds with Bad Request and a message specifying that it could not create the cart with the ID's

The checkout service here is a façade behind which these different steps take place to create an order and get the order amount. In future this flow can be extended to add more steps like applying validations, adding user specific information, and saving the information to a database.

Calculating Discount

The cart discount is implemented using chain of responsibility design pattern to apply multiple offers. T
he current implementation of the discount model only implements the quantity discount as both the offers given are of same type. The model can be extended in future to handle multiple types of discounts like amount discount (10% off on X cart value).

Since the offers can be changed in future, the offers are decoupled from the products. Moreover, different carts might be eligible for different offers, for example a cart of a premium user might have access to more offers. Hence, the discount service calls the offer service to get the current offers before calculating the cart discount. This can be extended in future to accommodate personalized shopping experiences.

As currently we do not have a database and we are not using user information for creating an order, the application is limited in capabilities. In future, adding a database with product catalogues and handling creation of user orders by creating and saving order information for a particular user can be implemented.

Moreover, we can also extend the project to include multiple types of offers in future and support personalized discount experience.

Automated Testing

The current project has 31 test cases implemented using the xUnit framework and consists of 23 unit tests and 8 integration tests. The integration test has test cases cart containing invalid ID's, multiple offers and combination of offers. The unit tests test the different service functionalities and cart discount module. However, the current test cases are written for only 4 services right now because of time constraint.

Future Improvements

- Due to the limited time and the scope of the problem, the application needs to store the product information and the offer information. In future, adding a database would be a good way to decouple data from the application.
- Moreover, the unit test cases do not cover all the services currently and I would like to add more unit test cases and functional tests in future.
- Moreover, ideally, it would be useful to extend this functionality in a more personalized way by adding user specific information and generating orders for a particular user. This will also help going forward to implement rate limiting on the usage of offers per user.
- I would have also liked to implement a more generic offer model to accommodate more types of offers for future.
- Moreover, due to limited time, I have hardcoded certain information inside the project and I would want to clean the project and ideally not have any hardcoded information inside the project.

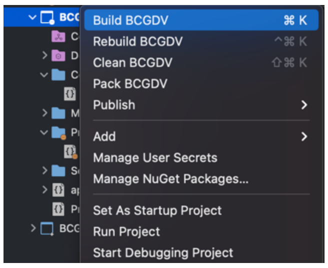
Steps to Install and Run the Application

Perquisites

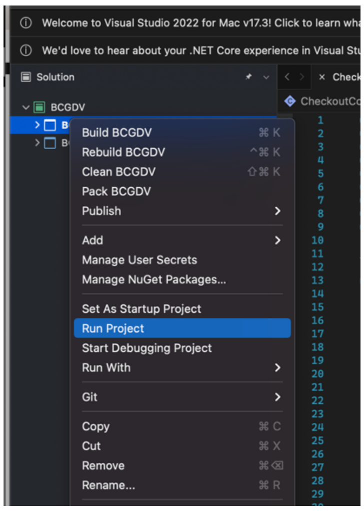
1. Install the .NET from here, and visual studio from here.

Running the application

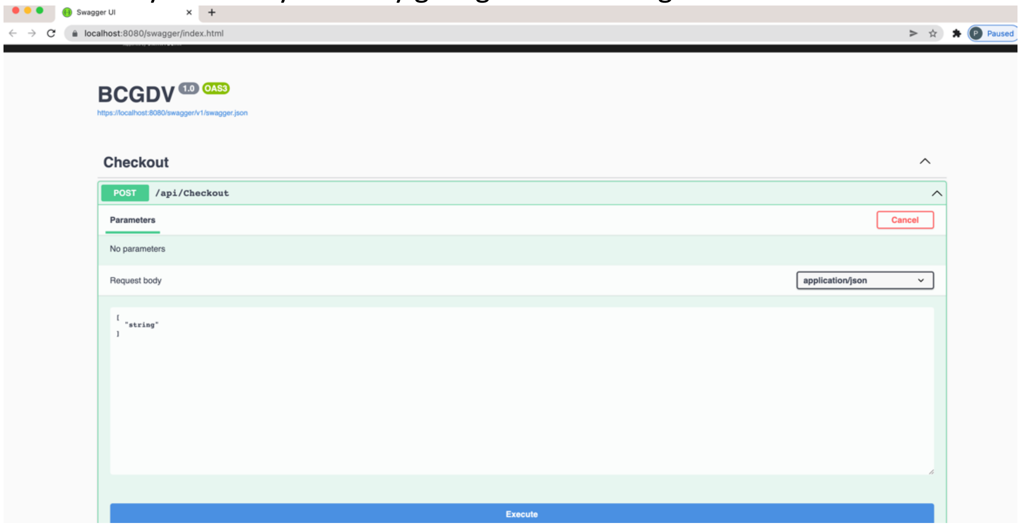
1. Open the BCGDV solution using visual studio. The solution contains two projects – the source code and the test code.
2. The source code project is named BCGDV and the test project is named BCGDV.Test
3. Navigate to BCGDV project and you can right click on the solution explorer to see the menu and select the option first build the project as shown below.



4. If the build overrides your launchsettings.json. Replace it with the following code - https://github.com/priyankasyal/BCGDV/blob/master/BCGDV/Properties/launchSettings.json
5. Now run the project by selecting the run project option in your solution explorer.



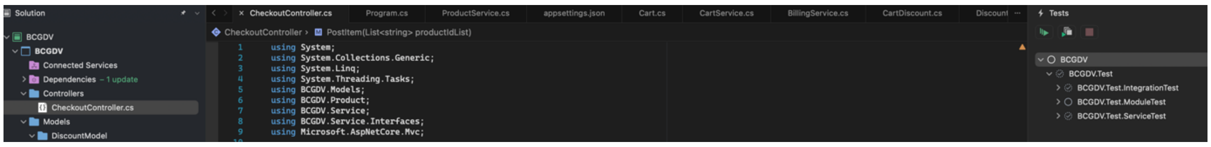
6. The project will open a new window in Google chrome when it starts as shown below and you can try it out by giving a list of string id's. as shown below.



```
[
  "string"
]
```

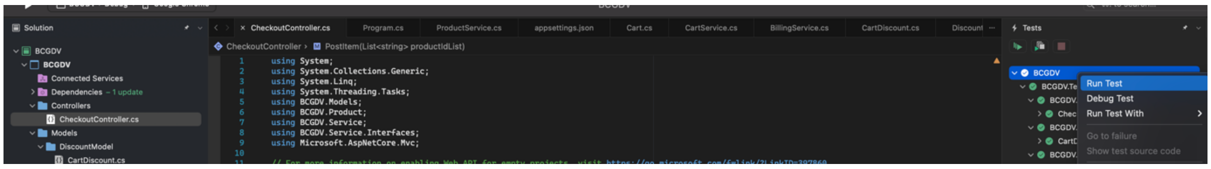
Running the tests

1. Build the BCGDV test solution.
2. Open the test explorer on the right hand side of visual studio as shown below to see the tests present as shown below.



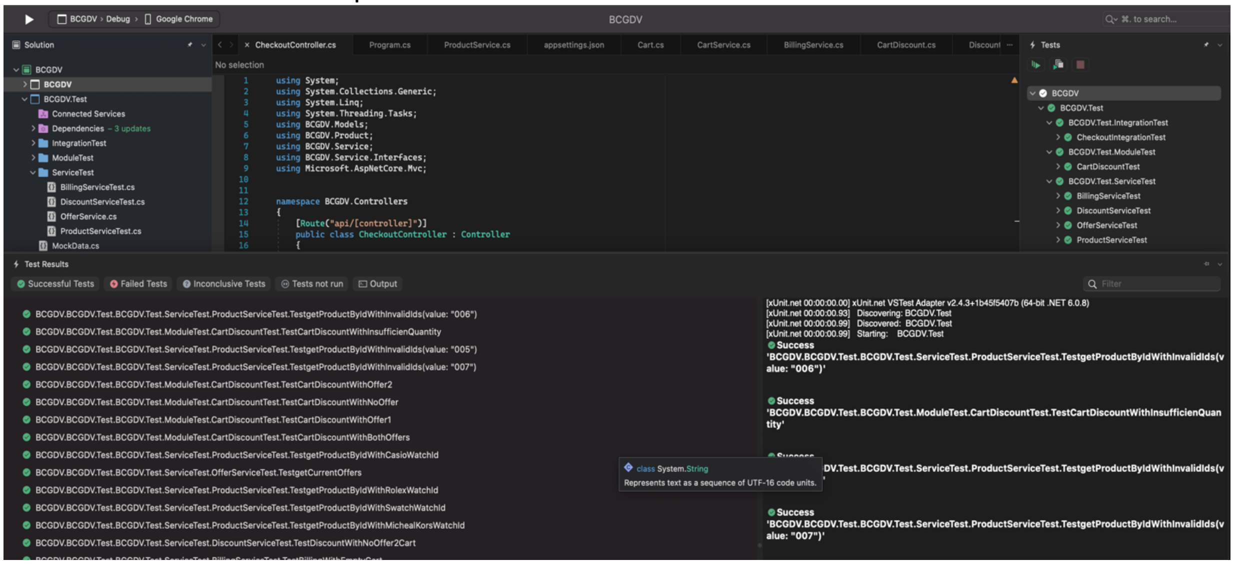
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BCGDV.Models;
using BCGDV.Product;
using BCGDV.Service;
using BCGDV.Service.Interfaces;
using Microsoft.AspNetCore.Mvc;
```

3. The tests can be run by right clicking on the test explorer and selecting run test option as shown below.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BCGDV.Models;
using BCGDV.Product;
using BCGDV.Service;
using BCGDV.Service.Interfaces;
using Microsoft.AspNetCore.Mvc;
```

4. The test results with the report can be seen in the run window as shown below.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BCGDV.Models;
using BCGDV.Product;
using BCGDV.Service;
using BCGDV.Service.Interfaces;
using Microsoft.AspNetCore.Mvc;


namespace BCGDV.Controllers
{
    [Route("api/[controller]")]
    public class CheckoutController : Controller
    {
```

```
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithInvalidIds(value: "006")
BCGDV.BCGDV.Test.BCGDV.Test.ModuleTest.CartDiscountTest.TestCartDiscountWithInsufficienQuantity
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithInvalidIds(value: "005")
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithInvalidIds(value: "007")
BCGDV.BCGDV.Test.BCGDV.Test.ModuleTest.CartDiscountTest.TestCartDiscountWithOffer2
BCGDV.BCGDV.Test.BCGDV.Test.ModuleTest.CartDiscountTest.TestCartDiscountWithNoOffer
BCGDV.BCGDV.Test.BCGDV.Test.ModuleTest.CartDiscountTest.TestCartDiscountWithOffer1
BCGDV.BCGDV.Test.BCGDV.Test.ModuleTest.CartDiscountTest.TestCartDiscountWithBothOffers
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithCasioWatchId
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.OfferServiceTest.TestgetCurrentOffers
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithRolexWatchId
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithSwatchWatchId
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithMichealKorsWatchId
BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.DiscountServiceTest.TestDiscountWithNoOffer2Cart
```

```
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.4.3+1b45f5407b (64-bit .NET 6.0.8)
[xUnit.net 00:00:00.93]   Discovering: BCGDV.Test
[xUnit.net 00:00:00.99]   Discovered:  BCGDV.Test
[xUnit.net 00:00:00.99]   Starting:    BCGDV.Test
Success
'BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithInvalidIds(v
alue: "006")'

Success
'BCGDV.BCGDV.Test.BCGDV.Test.ModuleTest.CartDiscountTest.TestCartDiscountWithInsufficienQuan
tity'

DV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithInvalidIds(v

Success
'BCGDV.BCGDV.Test.BCGDV.Test.ServiceTest.ProductServiceTest.TestgetProductByIdWithInvalidIds(v
alue: "007")'
```