1. Metoda „Cart“

1.1 Obuhvat iskaza/linija *(Statement/Line coverage)*

Za obuhvaćenost svih iskaza prilikom testiranja bila su potrebna dva testna slučaja i to:

Testni slučajevi:

```
TC1:

var cartList = new List<Cart>
{
   new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
   new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
};

discountAmount = “10”,
discountType = “0”,
string discountCode =  "DISCOUNT10"


TC2:

var cartList = new List<Cart>
{
   new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
   new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
};

discountAmount = “10”,
discountType = “1”,
string discountCode =  "DISCOUNT10"
```

Nakon testiranja sa prethodno navedena dva testna slučaja postignuta je poptuna obuhvatnost iskaza/linija. To je takođe i najamanji broj testnih slučajeva da se to postigne.

Testovi koji koriste ove testne slučajeve su sljedeći:

```
[TestMethod]

public void Cart_RedirectActionToCart_ShouldUpdateViewBagCorrectly()
```

```
{
    var cartList = new List<Cart>
    {
        new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
        new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
    };

    var cartDbSetMock = GetDbSetMock(cartList);

    dbContextMock.Setup(d => d.Cart).Returns(cartDbSetMock.Object);

var controllerMock = new Mock<DtoRequestsController>(dbContextMock.Object,
discountCodeVerifierMock.Object);

    controllerMock
        .Setup(c => c.GetCartProducts(It.IsAny<List<Cart>>()))
        .Returns<List<Cart>>(carts =>
        {
            var cartProducts = new List<List<Product>>();

            foreach (var cart in carts)
            {
                var products = cart.ProductID == 1
                                ? new List<Product> { new Product { ProductID = 1, Name = "testProduct", ImageUrl =
"testImageUrl", FlowerType = "testFlowerType", Stock = 0, Category = "testCategory", Description =
"testDescription", productType = "testProductType", Price = 20 } }
                    : new List<Product>();

                cartProducts.Add(products);
            }

            return cartProducts;
        });
    var userMock = new Mock<ClaimsPrincipal>();
                                        userMock.Setup(u          =>          u.FindFirst(ClaimTypes.NameIdentifier)).Returns(new
Claim(ClaimTypes.NameIdentifier, userId));

    controllerMock.Object.ControllerContext = new ControllerContext
    {
        HttpContext = new DefaultHttpContext { User = userMock.Object }
    };

    controllerMock.Object.Cart("10", "0", "DISCOUNT10");

    var viewBag = controllerMock.Object.ViewBag;

    var userCarts = viewBag.UserCarts as List<Cart>;
    var cartProducts = viewBag.CartProducts as List<List<Product>>;
    var discountCode = viewBag.DiscountCode as string;
    var totalAmountToPay = viewBag.TotalAmountToPay as double?;

    controllerMock.Verify(c => c.GetCartProducts(It.IsAny<List<Cart>>()), Times.Once);
}
```

```
[TestMethod]
public void Cart_RedirectActionToCart_DiscountType1_ShouldUpdateViewBagCorrectly()
```

```
{
    var cartList = new List<Cart>
    {
        new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
        new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
    };

    var cartDbSetMock = GetDbSetMock(cartList);

    dbContextMock.Setup(d => d.Cart).Returns(cartDbSetMock.Object);

var controllerMock = new Mock<DtoRequestsController>(dbContextMock.Object,
discountCodeVerifierMock.Object);

    controllerMock
        .Setup(c => c.GetCartProducts(It.IsAny<List<Cart>>()))
        .Returns<List<Cart>>(carts =>
        {
            var cartProducts = new List<List<Product>>();

            foreach (var cart in carts)
            {
                var products = cart.ProductID == 1
                        ? new List<Product> { new Product { ProductID = 1, Name = "testProduct", ImageUrl =
"testImageUrl", FlowerType = "testFlowerType", Stock = 0, Category = "testCategory", Description =
"testDescription", productType = "testProductType", Price = 20 } }
                    : new List<Product>();

                cartProducts.Add(products);
            }

            return cartProducts;
        });
    var userMock = new Mock<ClaimsPrincipal>();
  userMock.Setup(u => u.FindFirst(ClaimTypes.NameIdentifier)).Returns(new Claim(ClaimTypes.NameIdentifier,
userId));

    controllerMock.Object.ControllerContext = new ControllerContext
    {
        HttpContext = new DefaultHttpContext { User = userMock.Object }
    };

    controllerMock.Object.Cart("10", "1", "DISCOUNT10");

    var viewBag = controllerMock.Object.ViewBag;

    var userCarts = viewBag.UserCarts as List<Cart>;
    var cartProducts = viewBag.CartProducts as List<List<Product>>;
    var discountCode = viewBag.DiscountCode as string;
    var totalAmountToPay = viewBag.TotalAmountToPay as double?;

    controllerMock.Verify(c => c.GetCartProducts(It.IsAny<List<Cart>>()), Times.Once);
}
```

1.2 Obuhvat grana/odluka *(Branch/Decision Line coverage)*

Za obuhvaćenost svih grana/odluka prilikom testiranja bila su potrebna dva testna slučaja i to:

Testni slučajevi:

TC1:

```
var cartList = new List<Cart>
{
    new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
};

discountAmount = “10”,
discountType = “1”,
string discountCode =  "DISCOUNT10"



TC2:
var cartList = new List<Cart>
{
   new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
   new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
};

discountAmount = “10”,
discountType = “0”,
string discountCode =  "DISCOUNT10"
```

Nakon testiranja sa prethodno navedena dva testna slučaja postignuta je poptuna obuhvatnost grana/odluka. To je takođe i najamanji broj testnih slučajeva da se to postigne.

Testovi koji koriste ove testne slučajeve su sljedeći:

```
[TestMethod]


public void Cart_RedirectActionToCart_DiscountType1_NoUsersCarts_ShouldUpdateViewBagCorrectly()

{

   var cartList = new List<Cart>

   {

      new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },

   };


   var cartDbSetMock = GetDbSetMock(cartList);


  dbContextMock.Setup(d => d.Cart).Returns(cartDbSetMock.Object);
```

```
varcontrollerMock = new Mock<DtoRequestsController>(dbContextMock.Object,
discountCodeVerifierMock.Object);

  controllerMock
    .Setup(c => c.GetCartProducts(It.IsAny<List<Cart>>()))
    .Returns<List<Cart>>(carts =>
    {
      var cartProducts = new List<List<Product>>();

      foreach (var cart in carts)
      {
        var products = cart.ProductID == 1
                ? new List<Product> { new Product { ProductID = 1, Name = "testProduct", ImageUrl =
"testImageUrl", FlowerType = "testFlowerType", Stock = 0, Category = "testCategory", Description =
"testDescription", productType = "testProductType", Price = 20 } }
          : new List<Product>();

        cartProducts.Add(products);
      }

      return cartProducts;
    });

  var userMock = new Mock<ClaimsPrincipal>();
                    userMock.Setup(u => u.FindFirst(ClaimTypes.NameIdentifier)).Returns(new
Claim(ClaimTypes.NameIdentifier, userId));

  controllerMock.Object.ControllerContext = new ControllerContext
  {
    HttpContext = new DefaultHttpContext { User = userMock.Object }
  };
```

```
    controllerMock.Object.Cart("10", "1", "DISCOUNT10");

    var viewBag = controllerMock.Object.ViewBag;

    var userCarts = viewBag.UserCarts as List<Cart>;
    var cartProducts = viewBag.CartProducts as List<List<Product>>;
    var discountCode = viewBag.DiscountCode as string;
    var totalAmountToPay = viewBag.TotalAmountToPay as double?;

    controllerMock.Verify(c => c.GetCartProducts(It.IsAny<List<Cart>>()), Times.Once);
}

[TestMethod]
public void Cart_RedirectActionToCart_DiscountType1_ShouldUpdateViewBagCorrectly()
{
    var cartList = new List<Cart>
    {
        new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
        new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
    };

    var cartDbSetMock = GetDbSetMock(cartList);
  dbContextMock.Setup(d => d.Cart).Returns(cartDbSetMock.Object);

var controllerMock = new Mock<DtoRequestsController>(dbContextMock.Object,
discountCodeVerifierMock.Object);

    controllerMock
        .Setup(c => c.GetCartProducts(It.IsAny<List<Cart>>()))
        .Returns<List<Cart>>(carts =>
        {
            var cartProducts = new List<List<Product>>();
            foreach (var cart in carts)
            {
                var products = cart.ProductID == 1
                    ? new List<Product> { new Product { ProductID = 1, Name = "testProduct", ImageUrl = "testImageUrl", FlowerType = "testFlowerType", Stock = 0, Category = "testCategory", Description = "testDescription", productType = "testProductType", Price = 20 } }
                    : new List<Product>();

                cartProducts.Add(products);
            }

            return cartProducts;
        });

    var userMock = new Mock<ClaimsPrincipal>();
    userMock.Setup(u => u.FindFirst(ClaimTypes.NameIdentifier)).Returns(new Claim(ClaimTypes.NameIdentifier, userId));

    controllerMock.Object.ControllerContext = new ControllerContext
    {
        HttpContext = new DefaultHttpContext { User = userMock.Object }
    };

    controllerMock.Object.Cart("10", "0", "DISCOUNT10");

    var viewBag = controllerMock.Object.ViewBag;

    var userCarts = viewBag.UserCarts as List<Cart>;
    var cartProducts = viewBag.CartProducts as List<List<Product>>;
    var discountCode = viewBag.DiscountCode as string;
    var totalAmountToPay = viewBag.TotalAmountToPay as double?;
    controllerMock.Verify(c => c.GetCartProducts(It.IsAny<List<Cart>>()), Times.Once);

}
```

1.3 Obuhvat uslova *(Conditional coverage)*

Za obuhvaćenost grana/odluka prilikom testiranja bila su potrebna dva testna slučaja i to identična kao i u prethodnom dijelu(obuhvat grana/odluka):

Nakon testiranja sa prethodno navedena tri testna slučaja postignuta je poptuna obuhvatnost grana/odluka u kodu.

Testovi koji koriste ove testne slučajeve su sljedeći:

Testovi koji se koriste testni slučajevi su također isti.

1.4 Modifikovani uslov/odluka obuhvat *(Modified condition/decision coverage MCDC)*

Za obuhvaćenost modifikiovanih uslova/odluka prilikom testiranja bila su potrebna tri testna slučaja i to:

```
    TC1:
var cartList = new List<Cart>

{
    new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
};

discountAmount = “10”,
discountType = “1”,
string discountCode =  "DISCOUNT10"

    TC2:

var cartList = new List<Cart>
{
   new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
    new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
};

discountAmount = “10”,
discountType = “0”,
string discountCode =  "DISCOUNT10"
```

Prikaz tabele za sledeći uslov:

```
if (cart != null)

{

    if(product!=null)

    amountToPayWithoutDiscount += (double)(product.Price * cart.ProductQuantity);

}
```

Uslov 1	**Uslov 2**	**Ishod**
True	True	True
False	False	False
True	False	False
False	True	False

Dok za sljedeći if imamo samo dva slučaja i to true ili false, ali oni su pokriveni sa prvim testnim slučajem:

```
if (intDiscountType == 1)
```

Testovi koji koriste ove testne slučajeve su sljedeći:

Testovi koji se koriste testni slučajevi su također isti.

1.5 Obuhvat petlji *(Loop coverage)*

Prilikom obuhvata petlji potrebno je proći kroz sljedeće strategije:

1. **Preskoči unutrašnjost (tijelo) petlje**
   - Kada je lista korisničkih korpi prazna.

```
TC1: var cartList = new List<Cart>
{
        new Cart { CustomerID = "otherUserId", ProductID = 2, ProductQuantity = 1 },
};
```

2. **Uradi jedan prolaz kroz pelju**
   - Kada lista korisničkih korpi sadrži samo jedan element sa adekvatnim ‘userId’.

```
TC2: var cartList = new List<Cart>
{
new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },

};
```

3. **Uraditi dva prolaza kroz petlju**
   - Kada lista produkata sadrži dva elementa(produkta).

```
TC3: var cartList = new List<Cart>
{
  new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
 new Cart { CustomerID = userId, ProductID = 2, ProductQuantity = 5 },
};
```

4. **Uraditi slučajan broj prolaza kroz petlju**

```
TC4:   var cartList = new List<Cart>
  {

     new Cart { CustomerID = userId, ProductID = 2, ProductQuantity = 1 },
     new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
     new Cart { CustomerID = userId, ProductID = 2, ProductQuantity = 3 },
     new Cart { CustomerID = userId, ProductID = 2, ProductQuantity = 3 },
  };
```

5. **Uradi n, n-1, n+1 prolaza kroz petlju (n znači maksimalni broj prolaza kroz petlju)**
   - Nema smisla sa maksimalnim n probati za userCarts.Count() (može biti beskonačan broj ponavljanja/produkata, prekoračenje memorije). Moguća greška u programu.

Nakon provođenja ovih principa zaključak je da je potrebno 4 testna slučaja koja su navedena prethodno, s tim da nije moguće testirati kada je n maksimalan broj pa je moguća greška u programu.

Testovi koji koriste ove testne slučajeve su sljedeći:

```
    [TestMethod]

 public void Cart_RedirectActionToCart_DiscountType1_4Carts_ShouldUpdateViewBagCorrectly()
 {
     var cartList = new List<Cart>
     {

         new Cart { CustomerID = userId, ProductID = 2, ProductQuantity = 1 },
         new Cart { CustomerID = userId, ProductID = 1, ProductQuantity = 1 },
         new Cart { CustomerID = userId, ProductID = 2, ProductQuantity = 3 },
         new Cart { CustomerID = userId, ProductID = 2, ProductQuantity = 3 },
     };

     var cartDbSetMock = GetDbSetMock(cartList);

     dbContextMock.Setup(d => d.Cart).Returns(cartDbSetMock.Object);

     var controllerMock = new Mock<DtoRequestsController>(dbContextMock.Object,
discountCodeVerifierMock.Object);

     controllerMock
         .Setup(c => c.GetCartProducts(It.IsAny<List<Cart>>()))
         .Returns<List<Cart>>(carts =>
         {
             var cartProducts = new List<List<Product>>();

             foreach (var cart in carts)
             {
                 var products = cart.ProductID == 1
                     ? new List<Product> { new Product { ProductID = 1, Name = "testProduct", ImageUrl =
"testImageUrl", FlowerType = "testFlowerType", Stock = 0, Category = "testCategory", Description =
"testDescription", productType = "testProductType", Price = 20 } }
                     : new List<Product>();

                 cartProducts.Add(products);
             }

             return cartProducts;
         });

     var userMock = new Mock<ClaimsPrincipal>();
     userMock.Setup(u => u.FindFirst(ClaimTypes.NameIdentifier)).Returns(new
Claim(ClaimTypes.NameIdentifier, userId));

     controllerMock.Object.ControllerContext = new ControllerContext
     {
         HttpContext = new DefaultHttpContext { User = userMock.Object }
     };

     controllerMock.Object.Cart("10", "1", "DISCOUNT10");
     var viewBag = controllerMock.Object.ViewBag;

     var userCarts = viewBag.UserCarts as List<Cart>;
     var cartProducts = viewBag.CartProducts as List<List<Product>>;
     var discountCode = viewBag.DiscountCode as string;
     var totalAmountToPay = viewBag.TotalAmountToPay as double?;

     controllerMock.Verify(c => c.GetCartProducts(It.IsAny<List<Cart>>()), Times.Once);
 }
```