White box testiranje – Izvještaj

Hasičić Ilhan 19074

2. Metoda: OverallRating u HomeController

Metoda:

```
public virtual void OverallRating(){

    List<Order> orders = _context.Orders.ToList();

    double? rating = 0;
    int temp = 0;

    foreach (Order o in orders)
    {
        if (o.Rating != null)
        {
            temp++;
            rating += o.Rating;
        }
    }
    if (temp != 0)
        ViewBag.rating = Math.Round((decimal)rating / temp, 1);
    else ViewBag.rating = 0;

}
```

2.1 Statement/Line coverage

Budući da je logika metode da prolazi kroz listu ordera, dovoljan je jedan testni slučaj sa 4 ordera da pokrije sve linije metode.

TC1:

```
[TestMethod]
public void OverallRating_ShouldCalculateAverageRating()
{
    var orderList = new List<Order>
    {
        new Order { OrderID = 1, Rating = 4 },
        new Order { OrderID = 2, Rating = 5 },
        new Order { OrderID = 3, Rating = 3 },
        new Order { OrderID = 4, Rating = null }, // Should be ignored in the calculation
    };

    var orderDbSetMock = new Mock<DbSet<Order>>();
    orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.Provider).Returns(orderList.AsQueryable().Provider);
    orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
    orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
    orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

    var dbContextMock = new Mock<ApplicationDbContext>();
    dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

    var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

    controller.OverallRating();

    var result = controller.ViewBag.rating;
    Assert.IsNotNull(result, "ViewBag.rating should not be null");
    Assert.AreEqual(4, result, "Should calculate the correct average rating");
}
```

- *Branch/Decision Line coverage*

Da bismo postigli potpun branch coverage potrebno je dodati još jedan test koji će sadržavati sve ordere koji imaju null vrijednosti svojih ratinga

```
    [TestMethod]
    public void OverallRating_ShouldCalculateAverageRating()
    {
        var orderList = new List<Order>
{
    new Order { OrderID = 1, Rating = 4 },
    new Order { OrderID = 2, Rating = 5 },
    new Order { OrderID = 3, Rating = 3 },
    new Order { OrderID = 4, Rating = null }, // Should be ignored in the calculation
};

        var orderDbSetMock = new Mock<DbSet<Order>>();
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

        var dbContextMock = new Mock<ApplicationDbContext>();
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(4, result, "Should calculate the correct average rating");
    }

    [TestMethod]
    public void OverallRating_NoNonNullOrders_ShouldSetViewBagRatingToZero()
    {
        var orderList = new List<Order>
{
    new Order { OrderID = 1, Rating = null },
    new Order { OrderID = 2, Rating = null },
    new Order { OrderID = 3, Rating = null },
    new Order { OrderID = 4, Rating = null },
};
```

```
        var orderDbSetMock = new Mock<DbSet<Order>>();
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

        var dbContextMock = new Mock<ApplicationDbContext>();
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(0, result, "Should set ViewBag.rating to 0 when there are no non-null ratings");
    }
```

- *Conditional coverage*

Da bi se postigla potpuna pokrivenost složenih logičkih uslova, potrebno je dodati testne slučajeve koji imaju:

1. pojedine vrijednosti za rating su null
2. nema ratinga sa null vrijednostima
3. sve rating vrijednosti su null

TC1:

```
    [TestMethod]
    public void OverallRating_ShouldCalculateAverageRating()
    {
        var orderList = new List<Order>
{
    new Order { OrderID = 1, Rating = 4 },
    new Order { OrderID = 2, Rating = 5 },
    new Order { OrderID = 3, Rating = 3 },
    new Order { OrderID = 4, Rating = null }, // Should be ignored in the calculation
};

        var orderDbSetMock = new Mock<DbSet<Order>>();
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

        var dbContextMock = new Mock<ApplicationDbContext>();
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);
        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(4, result, "Should calculate the correct average rating");
    }
```

TC2:

```
    [TestMethod]
    public void OverallRating_NoNonNullOrders_ShouldSetViewBagRatingToZero()
    {
        var orderList = new List<Order>
{
    new Order { OrderID = 1, Rating = null },
    new Order { OrderID = 2, Rating = null },
    new Order { OrderID = 3, Rating = null },
    new Order { OrderID = 4, Rating = null },
};

        var orderDbSetMock = new Mock<DbSet<Order>>();
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

        var dbContextMock = new Mock<ApplicationDbContext>();
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(0, result, "Should set ViewBag.rating to 0 when there are no non-null ratings");
    }
```

TC3:

```
    [TestMethod]
    public void OverallRating_SomeNonNullOrders_ShouldCalculateAverageRating()
    {
        var orderList = new List<Order>
{
    new Order { OrderID = 1, Rating = 4 },
    new Order { OrderID = 2, Rating = null },
    new Order { OrderID = 3, Rating = 3 },
    new Order { OrderID = 4, Rating = 5 },
};

        var orderDbSetMock = new Mock<DbSet<Order>>();
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
```

```
        orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

        var dbContextMock = new Mock<ApplicationDbContext>();
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(4, result, "Should calculate the correct average rating considering only non-null ratings");
    }
```

- *Modified condition/decision coverage MCDC*

TC1:

```
    [TestMethod]
    public void OverallRating_ShouldCalculateAverageRating()
    {
        var orderList = new List<Order>
{
    new Order { OrderID = 1, Rating = 4 },
    new Order { OrderID = 2, Rating = 5 },
    new Order { OrderID = 3, Rating = 3 },
    new Order { OrderID = 4, Rating = null }, // Should be ignored in the calculation
};

        var orderDbSetMock = new Mock<DbSet<Order>>();
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

        var dbContextMock = new Mock<ApplicationDbContext>();
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(4, result, "Should calculate the correct average rating");
    }
```

Za sljedeći if uslov imamo samo dva moguća slučaja, true ili false. Oni su pokriveni u TC1.

```
if (o.Rating != null)
```

Za if uslov

```
if (temp != 0)
```

imamo dva moguća slučaja, true ili false. Oni su pokriveni u TC1.

- *Loop coverage*

Prilikom obuhvata petlji potrebno je proći kroz sljedeće strategije:

1. Preskočiti tijelo petlje

```
[TestMethod]
public void OverallRating_ShouldSetViewBagRatingToZeroForEmptyList()
{
   var orderList = new List<Order>();

   var orderDbSetMock = new Mock<DbSet<Order>>();
   orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.Provider).Returns(orderList.AsQueryable().Provider);
   orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
   orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
   orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

   var dbContextMock = new Mock<ApplicationDbContext>();
   dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

   var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

   controller.OverallRating();

   var result = controller.ViewBag.rating;
   Assert.IsNotNull(result, "ViewBag.rating should not be null");
   Assert.AreEqual(0, result, "Should set ViewBag.rating to 0 for an empty list");
}
```

2. Uraditi jedan prolaz kroz petlju, kada lista sadrži samo jedan order

```
   [TestMethod]
   public void OverallRating_ShouldCalculateAverageRatingForSingleOrder()
   {
      var orderList = new List<Order>
{
   new Order { OrderID = 1, Rating = 4 },
};

      var orderDbSetMock = new Mock<DbSet<Order>>();
      orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
      orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
      orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
      orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

      var dbContextMock = new Mock<ApplicationDbContext>();
```

```
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(4, result, "Should set ViewBag.rating to the rating of the single order");
    }
```

3. Uraditi 2 prolaza kroz petlju, kada lista ordera sadrži dva elementa

```
[TestMethod]
public void OverallRating_ShouldCalculateAverageRatingForTwoOrders()
{
    var orderList = new List<Order>
    {
        new Order { OrderID = 1, Rating = 4 },
        new Order { OrderID = 2, Rating = 5 },
    };

    var orderDbSetMock = new Mock<DbSet<Order>>();
    orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.Provider).Returns(orderList.AsQueryable().Provider);
    orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
    orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
    orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

    var dbContextMock = new Mock<ApplicationDbContext>();
    dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

    var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

    controller.OverallRating();

    var result = controller.ViewBag.rating;
    Assert.IsNotNull(result, "ViewBag.rating should not be null");
    Assert.AreEqual(4.5, result, "Should calculate the correct average rating for two orders");
}
```

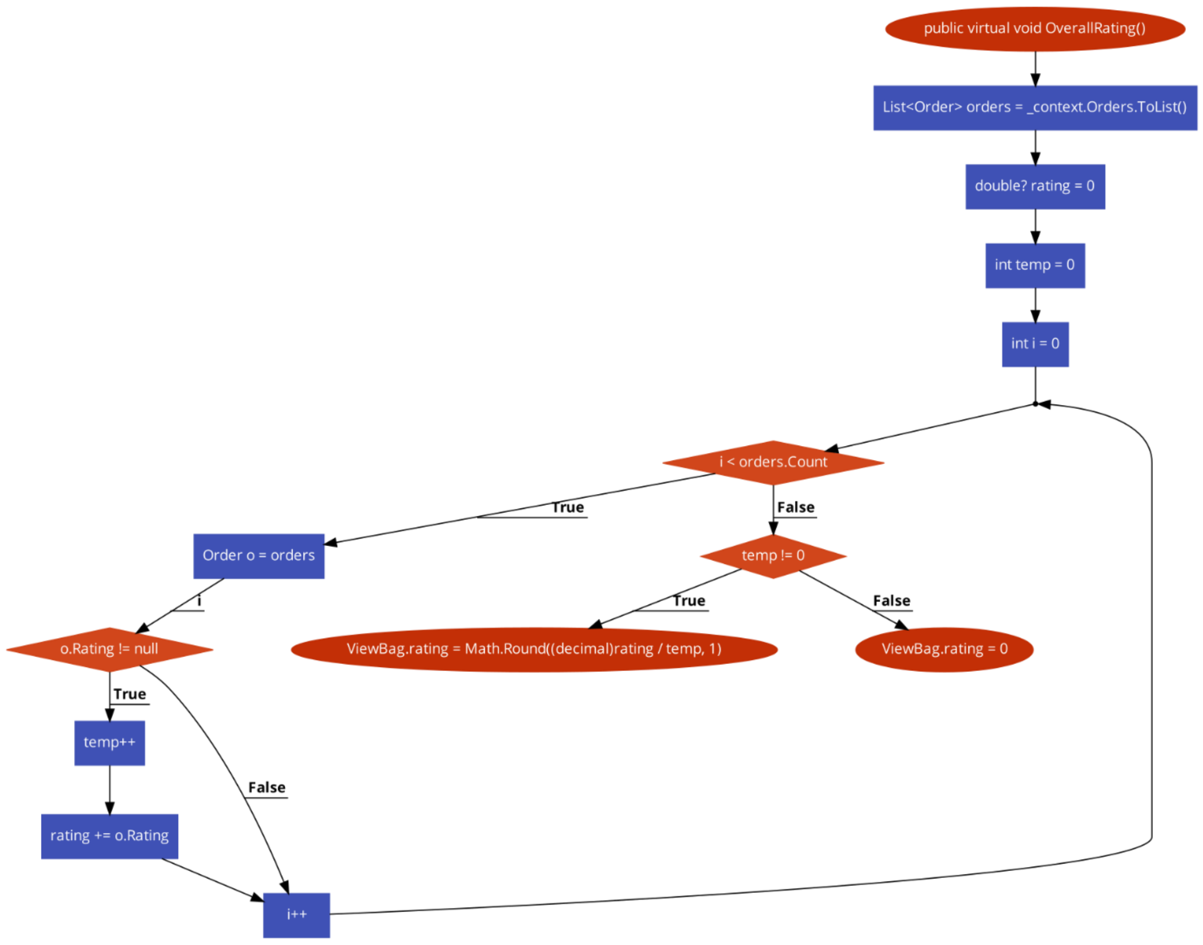
4. Uraditi slučajan broj prolaza kroz petlju

```
    [TestMethod]
    public void OverallRating_ShouldCalculateAverageRating()
    {
        var orderList = new List<Order>
{
    new Order { OrderID = 1, Rating = 4 },
    new Order { OrderID = 2, Rating = 5 },
    new Order { OrderID = 3, Rating = 3 },
    new Order { OrderID = 4, Rating = null }, // Should be ignored in the calculation
};

        var orderDbSetMock = new Mock<DbSet<Order>>();
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Provider).Returns(orderList.AsQueryable().Provider);
```

```
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.Expression).Returns(orderList.AsQueryable().Expression);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m =>
m.ElementType).Returns(orderList.AsQueryable().ElementType);
        orderDbSetMock.As<IQueryable<Order>>().Setup(m => m.GetEnumerator()).Returns(() =>
orderList.GetEnumerator());

        var dbContextMock = new Mock<ApplicationDbContext>();
        dbContextMock.Setup(d => d.Orders).Returns(orderDbSetMock.Object);

        var controller = new HomeController(Mock.Of<ILogger<HomeController>>(), dbContextMock.Object);

        controller.OverallRating();

        var result = controller.ViewBag.rating;
        Assert.IsNotNull(result, "ViewBag.rating should not be null");
        Assert.AreEqual(4, result, "Should calculate the correct average rating");
    }
```

5. Uradi n, n-1, n+1 prolaza kroz petlju. N znači maksimalni broj prolaza kroz petlju Nema smisla probati sa maksimalnim n za orders.count() (može biti beskonačan broj ponavljanja ordera, 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.

- *Path coverage*



Broj puta	Put
1	1-2-3-4-5-6-7-8
2	1-2-3-4-5-6-7-9
3	1-2-3-4-5-6-10-11-12-13-14-6-7-8
4	1-2-3-4-5-6-10-11-12-13-14-6-7-9
5	1-2-3-4-5-6-10-11-14-6-7-8
6	1-2-3-4-5-6-10-11-14-6-7-9