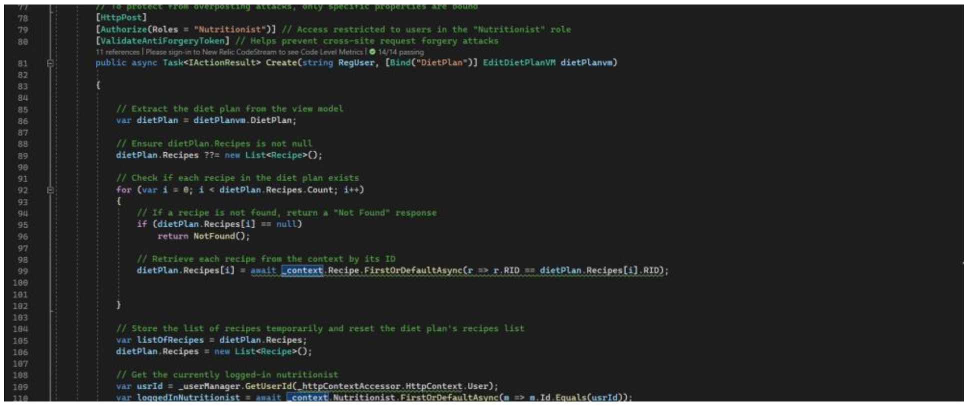
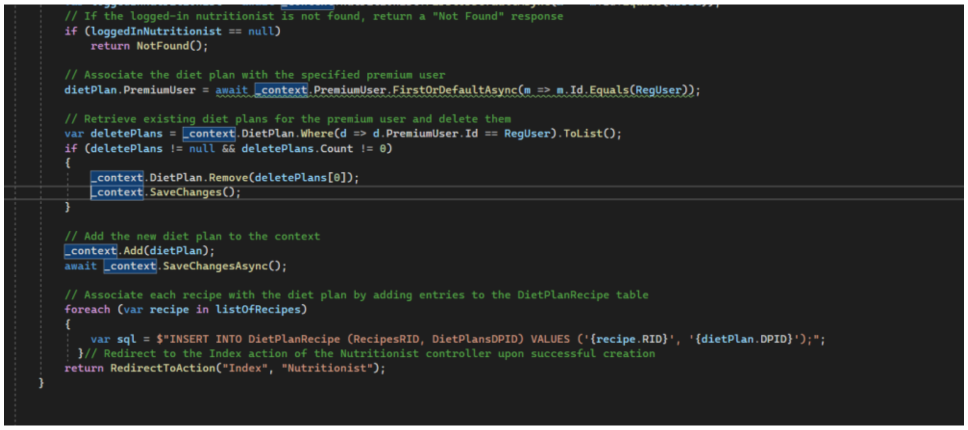
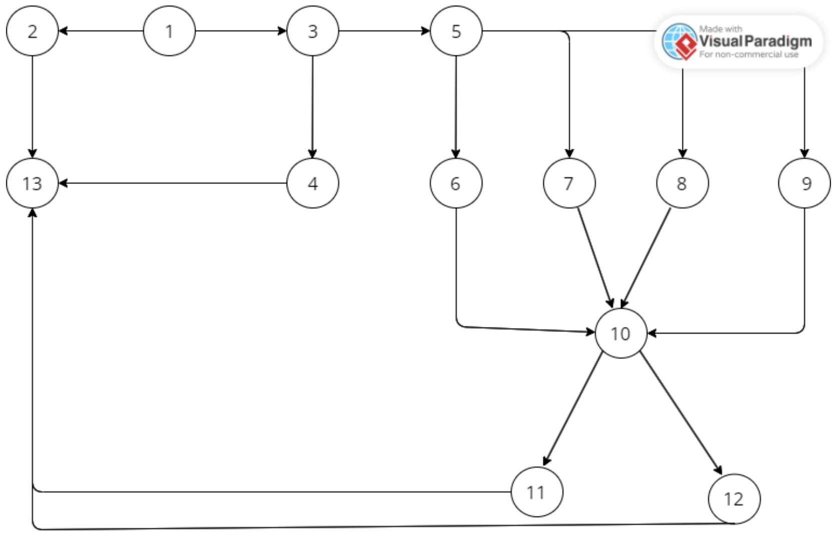
WHITE BOX Testiranje – DietPlan Controller – Metoda POST Create

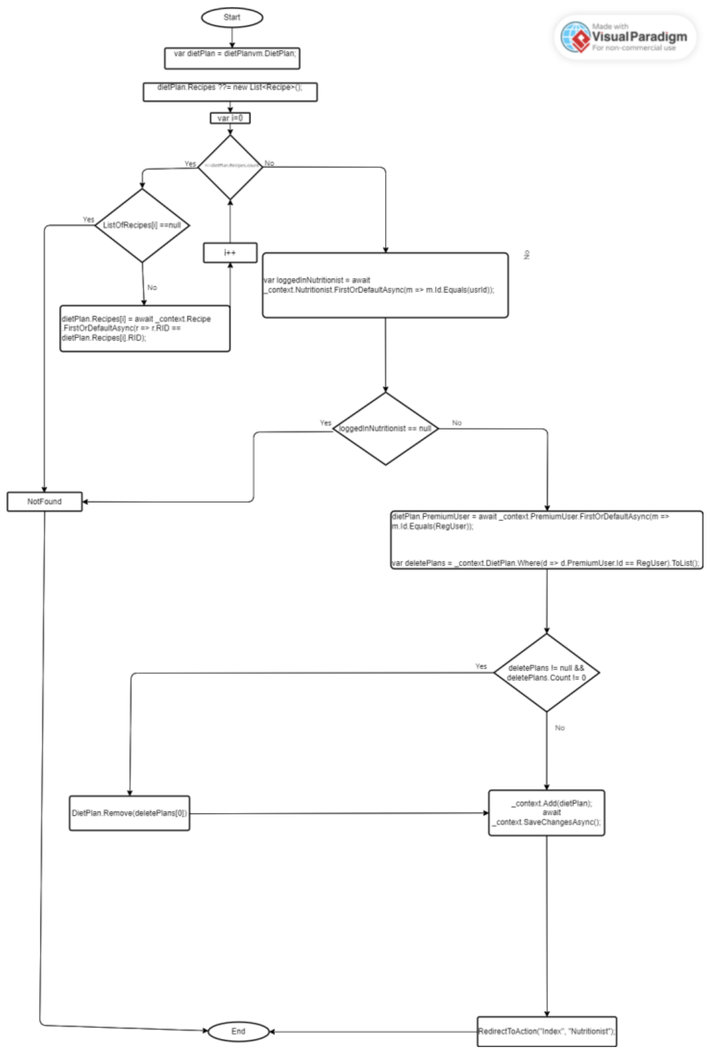


```
// To protect from overposting attacks, only specific properties are bound
[HttpPost]
[Authorize(Roles = "Nutritionist")] // Access restricted to users in the "Nutritionist" role
[ValidateAntiForgeryToken] // Helps prevent cross-site request forgery attacks
11 references | Please sign-in to New Relic CodeStream to see Code Level Metrics | 14/14 passing
public async Task<IActionResult> Create(string RegUser, [Bind("DietPlan")] EditDietPlanVM dietPlanvm)

{

    // Extract the diet plan from the view model
    var dietPlan = dietPlanvm.DietPlan;

    // Ensure dietPlan.Recipes is not null
    dietPlan.Recipes ??= new List<Recipe>();

    // Check if each recipe in the diet plan exists
    for (var i = 0; i < dietPlan.Recipes.Count; i++)
    {
        // If a recipe is not found, return a "Not Found" response
        if (dietPlan.Recipes[i] == null)
            return NotFound();

        // Retrieve each recipe from the context by its ID
        dietPlan.Recipes[i] = await _context.Recipe.FirstOrDefaultAsync(r => r.RID == dietPlan.Recipes[i].RID);


    }

    // Store the list of recipes temporarily and reset the diet plan's recipes list
    var listOfRecipes = dietPlan.Recipes;
    dietPlan.Recipes = new List<Recipe>();

    // Get the currently logged-in nutritionist
    var usrId = _userManager.GetUserId(_httpContextAccessor.HttpContext.User);
    var loggedInNutritionist = await _context.Nutritionist.FirstOrDefaultAsync(m => m.Id.Equals(usrId));
```



```
    // If the logged-in nutritionist is not found, return a "Not Found" response
    if (loggedInNutritionist == null)
        return NotFound();

    // Associate the diet plan with the specified premium user
    dietPlan.PremiumUser = await _context.PremiumUser.FirstOrDefaultAsync(m => m.Id.Equals(RegUser));

    // Retrieve existing diet plans for the premium user and delete them
    var deletePlans = _context.DietPlan.Where(d => d.PremiumUser.Id == RegUser).ToList();
    if (deletePlans != null && deletePlans.Count != 0)
    {
        _context.DietPlan.Remove(deletePlans[0]);
        _context.SaveChanges();
    }

    // Add the new diet plan to the context
    _context.Add(dietPlan);
    await _context.SaveChangesAsync();

    // Associate each recipe with the diet plan by adding entries to the DietPlanRecipe table
    foreach (var recipe in listOfRecipes)
    {
        var sql = $"INSERT INTO DietPlanRecipe (RecipesRID, DietPlansDPID) VALUES ('{recipe.RID}', '{dietPlan.DPID}');";
     }// Redirect to the Index action of the Nutritionist controller upon successful creation
    return RedirectToAction("Index", "Nutritionist");
}
```




Start
var dietPlan = dietPlanvm.DietPlan;
dietPlan.Recipes ??= new List<Recipe>();
var i=0
Yes
No
ListOfRecipes[i] ==null
Yes
No
dietPlan.Recipes[i] = await _context.Recipe .FirstOrDefaultAsync(r => r.RID == dietPlan.Recipes[i].RID);
i++
var loggedInNutritionist = await _context.Nutritionist.FirstOrDefaultAsync(m => m.Id.Equals(usrId));
No
loggedInNutritionist == null
Yes
No
NotFound
dietPlan.PremiumUser = await _context.PremiumUser.FirstOrDefaultAsync(m => m.Id.Equals(RegUser));
var deletePlans = _context.DietPlan.Where(d => d.PremiumUser.Id == RegUser).ToList();
deletePlans != null && deletePlans.Count != 0
Yes
No
DietPlan.Remove(deletePlans[0])
_context.Add(dietPlan); await _context.SaveChangesAsync();
RedirectToAction("Index", "Nutritionist");
End
Made with Visual Paradigm For non-commercial use

U ovom slučaju smo imali 10 mogućih putanja koje pokrivaju DietPlan Controller. Ti putevi su:

1 -> 2 -> 13

1 -> 3 -> 4 -> 13

1 -> 3 -> 5 -> 6 -> 10 -> 11 -> 13

1 -> 3 -> 5 -> 6 -> 10 -> 12 -> 13

1 -> 3 -> 5 -> 7 -> 10 -> 11 -> 13

1 -> 3 -> 5 -> 7 -> 10 -> 12 -> 13

1 -> 3 -> 5 -> 8 -> 10 -> 11 -> 13

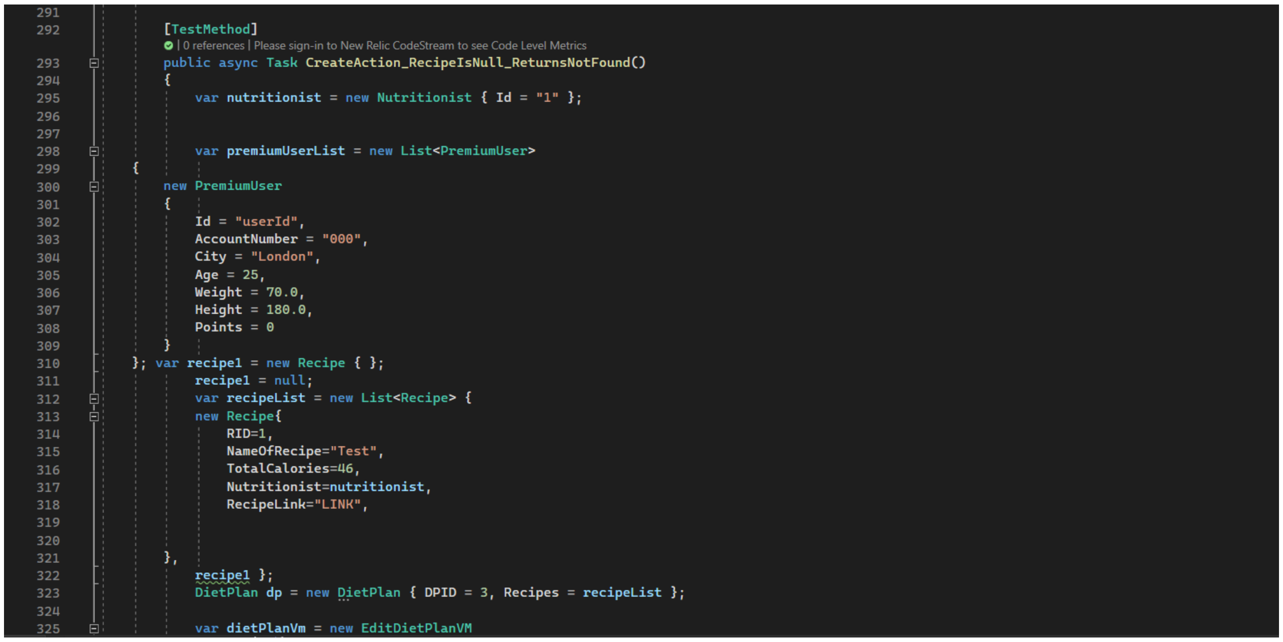
1 -> 3 -> 5 -> 8 -> 10 -> 12 -> 13

1 -> 3 -> 5 -> 9 -> 10 -> 11 -> 13

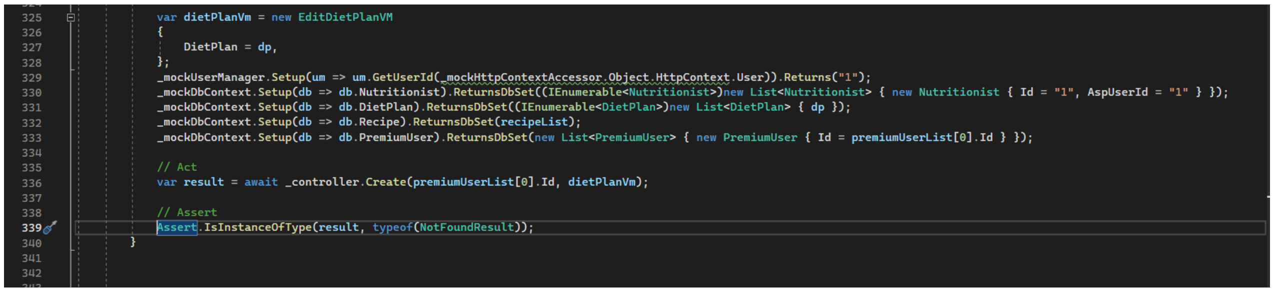
1 -> 3 -> 5 -> 9 -> 10 -> 12 -> 13

Kako bih postigla pokrivenost sa najkraćim putem napisala sam sljedeće testne slučajeve:

Prvenstveno sam testirala slučaj kada se pokuša kreirati novi DietPlan, ali zbog Recipe koji je null, dolazi do vraćanja NotFound. Ime tog testa je CreateAction_RecipeIsNull_ReturnsNotFound()

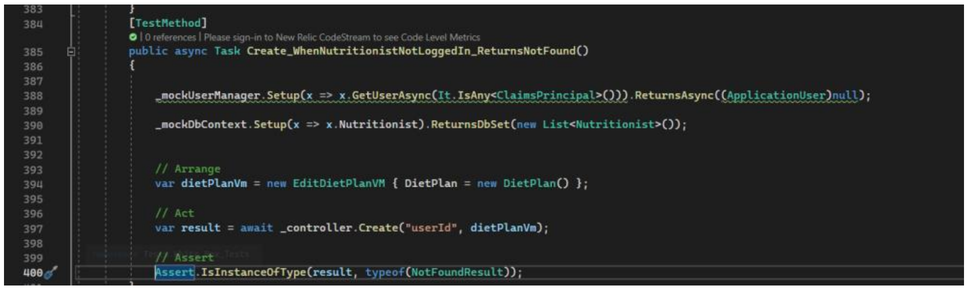


```
        [TestMethod]
        | 0 references | Please sign-in to New Relic CodeStream to see Code Level Metrics
        public async Task CreateAction_RecipeIsNull_ReturnsNotFound()
        {
            var nutritionist = new Nutritionist { Id = "1" };


            var premiumUserList = new List<PremiumUser>
    {
        new PremiumUser
        {
            Id = "userId",
            AccountNumber = "000",
            City = "London",
            Age = 25,
            Weight = 70.0,
            Height = 180.0,
            Points = 0
        }
    }; var recipe1 = new Recipe { };
            recipe1 = null;
            var recipeList = new List<Recipe> {
            new Recipe{
                RID=1,
                NameOfRecipe="Test",
                TotalCalories=46,
                Nutritionist=nutritionist,
                RecipeLink="LINK",


        },
            recipe1 };
            DietPlan dp = new DietPlan { DPID = 3, Recipes = recipeList };

            var dietPlanVm = new EditDietPlanVM
```



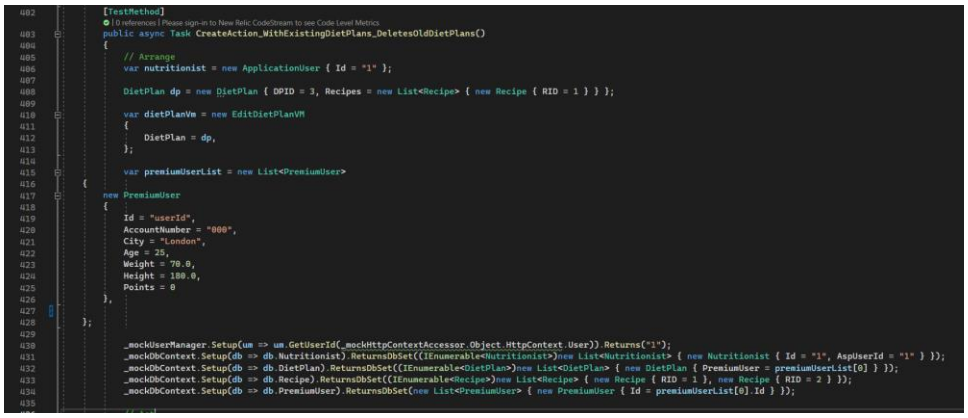
```
            var dietPlanVm = new EditDietPlanVM
            {
                DietPlan = dp,
            };
            _mockUserManager.Setup(um => um.GetUserId(_mockHttpContextAccessor.Object.HttpContext.User)).Returns("1");
            _mockDbContext.Setup(db => db.Nutritionist).ReturnsDbSet((IEnumerable<Nutritionist>)new List<Nutritionist> { new Nutritionist { Id = "1", AspUserId = "1" } });
            _mockDbContext.Setup(db => db.DietPlan).ReturnsDbSet((IEnumerable<DietPlan>)new List<DietPlan> { dp });
            _mockDbContext.Setup(db => db.Recipe).ReturnsDbSet(recipeList);
            _mockDbContext.Setup(db => db.PremiumUser).ReturnsDbSet(new List<PremiumUser> { new PremiumUser { Id = premiumUserList[0].Id } });

            // Act
            var result = await _controller.Create(premiumUserList[0].Id, dietPlanVm);

            // Assert
            Assert.IsInstanceOfType(result, typeof(NotFoundResult));
        }
```

Sljedeći testni slučaj koji sam pokrila jeste kada Nutritionist nije pronađen, čime sam pokrila uslov kada je Nutritionist null. Test se zove Create_WhenNutritionistNotLoggedIn_ReturnsNotFound()

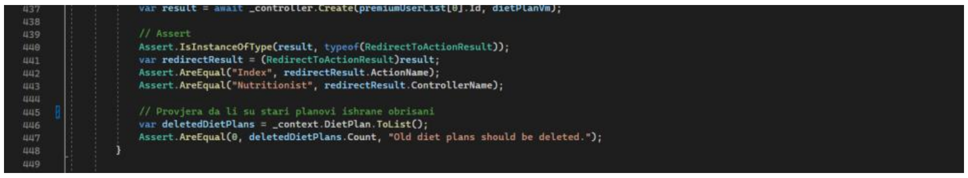


```
        }
        [TestMethod]
        | 0 references | Please sign-in to New Relic CodeStream to see Code Level Metrics
        public async Task Create_WhenNutritionistNotLoggedIn_ReturnsNotFound()
        {

            _mockUserManager.Setup(x => x.GetUserAsync(It.IsAny<ClaimsPrincipal>())).ReturnsAsync((ApplicationUser)null);

            _mockDbContext.Setup(x => x.Nutritionist).ReturnsDbSet(new List<Nutritionist>());


            // Arrange
            var dietPlanVm = new EditDietPlanVM { DietPlan = new DietPlan() };

            // Act
            var result = await _controller.Create("userId", dietPlanVm);

            // Assert
            Assert.IsInstanceOfType(result, typeof(NotFoundResult));
```

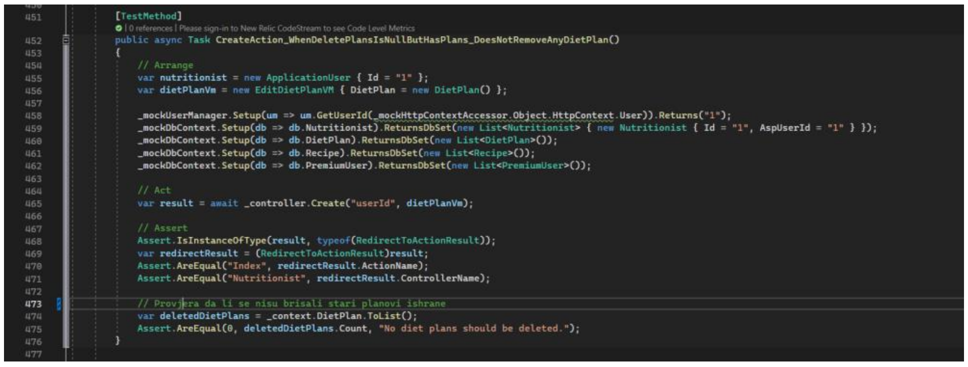
Sljedeći uslov koji je trebalo ispitati sve slučajeve, te tako postići pokrivenost koda je u slučaju brisanja DietPlan-ova. To sam pokrila sljedećim slučajevima. Testovi koji to pokrivaju su:
CreateAction_WhenDeletePlansIsNotNullButHasNoPlans_DoesNotRemoveAnyDietPlan()
CreateAction_WhenDeletePlansIsNullButHasPlans_DoesNotRemoveAnyDietPlan()
CreateAction_WithExistingDietPlans_DeletesOldDietPlans()



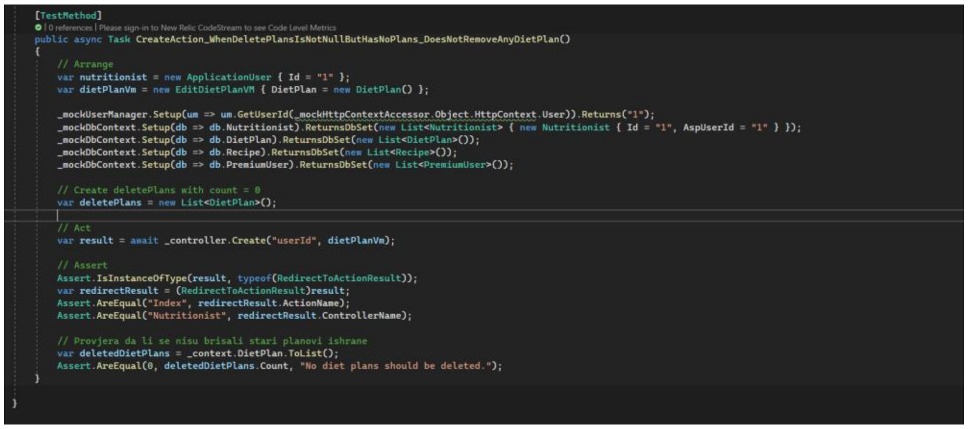
```
        [TestMethod]
        | 0 references | Please sign-in to New Relic CodeStream to see Code Level Metrics
        public async Task CreateAction_WithExistingDietPlans_DeletesOldDietPlans()
        {
            // Arrange
            var nutritionist = new ApplicationUser { Id = "1" };

            DietPlan dp = new DietPlan { DPID = 3, Recipes = new List<Recipe> { new Recipe { RID = 1 } } };

            var dietPlanVm = new EditDietPlanVM
            {
                DietPlan = dp,
            };

            var premiumUserList = new List<PremiumUser>
    {
        new PremiumUser
        {
            Id = "userId",
            AccountNumber = "000",
            City = "London",
            Age = 25,
            Weight = 70.0,
            Height = 180.0,
            Points = 0
        },

    };

            _mockUserManager.Setup(um => um.GetUserId(_mockHttpContextAccessor.Object.HttpContext.User)).Returns("1");
            _mockDbContext.Setup(db => db.Nutritionist).ReturnsDbSet((IEnumerable<Nutritionist>)new List<Nutritionist> { new Nutritionist { Id = "1", AspUserId = "1" } });
            _mockDbContext.Setup(db => db.DietPlan).ReturnsDbSet((IEnumerable<DietPlan>)new List<DietPlan> { new DietPlan { PremiumUser = premiumUserList[0] } });
            _mockDbContext.Setup(db => db.Recipe).ReturnsDbSet((IEnumerable<Recipe>)new List<Recipe> { new Recipe { RID = 1 }, new Recipe { RID = 2 } });
            _mockDbContext.Setup(db => db.PremiumUser).ReturnsDbSet(new List<PremiumUser> { new PremiumUser { Id = premiumUserList[0].Id } });

```



```
            var result = await _controller.Create(premiumUserList[0].Id, dietPlanVm);

            // Assert
            Assert.IsInstanceOfType(result, typeof(RedirectToActionResult));
            var redirectResult = (RedirectToActionResult)result;
            Assert.AreEqual("Index", redirectResult.ActionName);
            Assert.AreEqual("Nutritionist", redirectResult.ControllerName);

            // Provjera da li su stari planovi ishrane obrisani
            var deletedDietPlans = _context.DietPlan.ToList();
            Assert.AreEqual(0, deletedDietPlans.Count, "Old diet plans should be deleted.");
        }

```

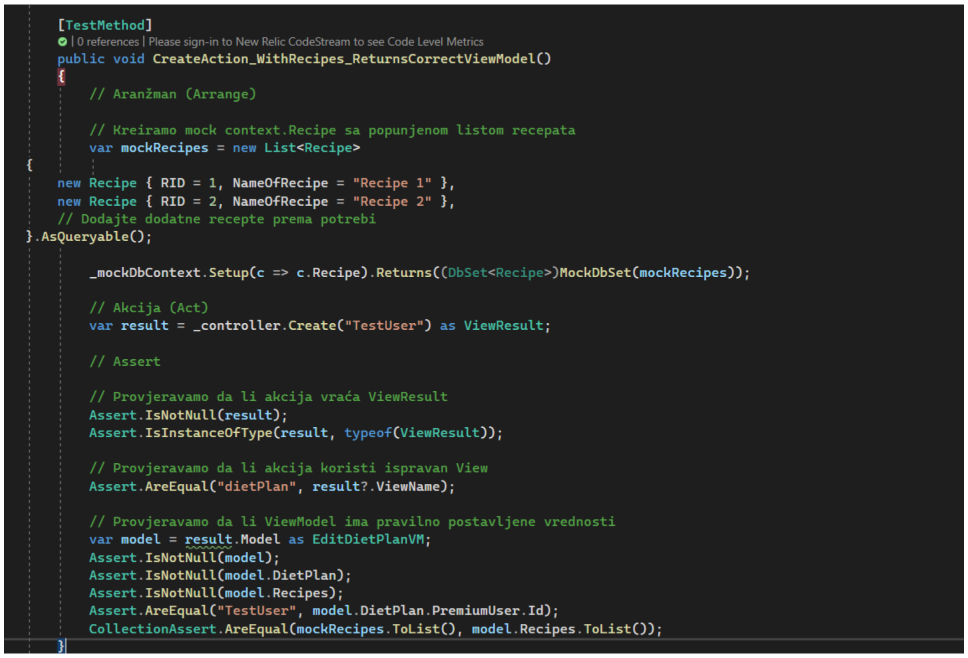


```
        [TestMethod]
        | 0 references | Please sign-in to New Relic CodeStream to see Code Level Metrics
        public async Task CreateAction_WhenDeletePlansIsNullButHasPlans_DoesNotRemoveAnyDietPlan()
        {
            // Arrange
            var nutritionist = new ApplicationUser { Id = "1" };
            var dietPlanVm = new EditDietPlanVM { DietPlan = new DietPlan() };

            _mockUserManager.Setup(um => um.GetUserId(_mockHttpContextAccessor.Object.HttpContext.User)).Returns("1");
            _mockDbContext.Setup(db => db.Nutritionist).ReturnsDbSet(new List<Nutritionist> { new Nutritionist { Id = "1", AspUserId = "1" } });
            _mockDbContext.Setup(db => db.DietPlan).ReturnsDbSet(new List<DietPlan>());
            _mockDbContext.Setup(db => db.Recipe).ReturnsDbSet(new List<Recipe>());
            _mockDbContext.Setup(db => db.PremiumUser).ReturnsDbSet(new List<PremiumUser>());

            // Act
            var result = await _controller.Create("userId", dietPlanVm);

            // Assert
            Assert.IsInstanceOfType(result, typeof(RedirectToActionResult));
            var redirectResult = (RedirectToActionResult)result;
            Assert.AreEqual("Index", redirectResult.ActionName);
            Assert.AreEqual("Nutritionist", redirectResult.ControllerName);

            // Provjera da li se nisu brisali stari planovi ishrane
            var deletedDietPlans = _context.DietPlan.ToList();
            Assert.AreEqual(0, deletedDietPlans.Count, "No diet plans should be deleted.");
        }

```

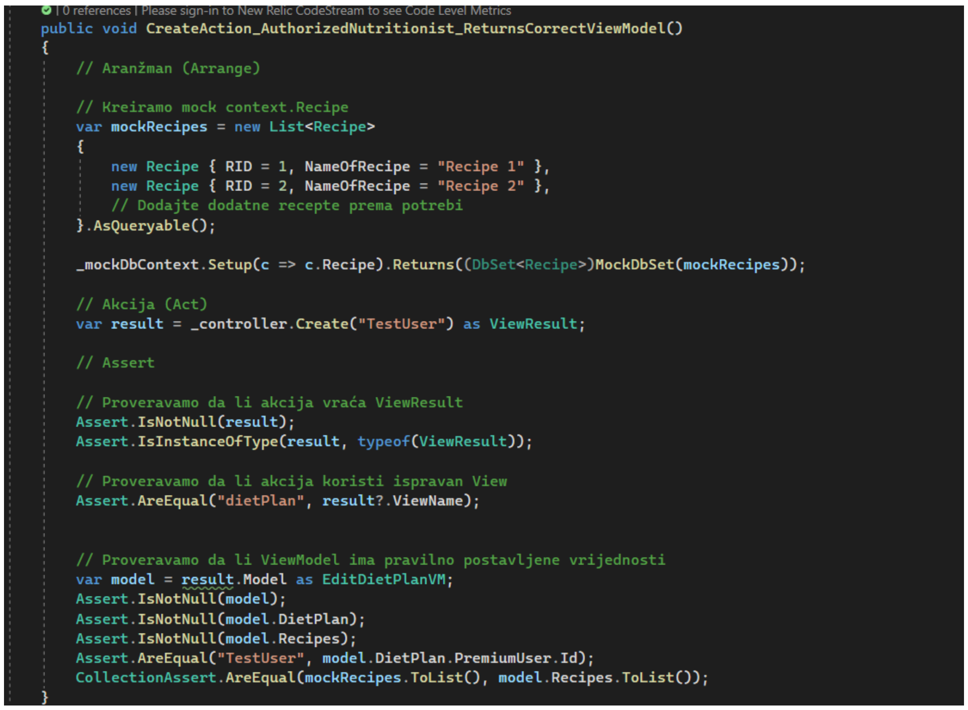


```
    [TestMethod]
    | 0 references | Please sign-in to New Relic CodeStream to see Code Level Metrics
    public async Task CreateAction_WhenDeletePlansIsNotNullButHasNoPlans_DoesNotRemoveAnyDietPlan()
    {
        // Arrange
        var nutritionist = new ApplicationUser { Id = "1" };
        var dietPlanVm = new EditDietPlanVM { DietPlan = new DietPlan() };

        _mockUserManager.Setup(um => um.GetUserId(_mockHttpContextAccessor.Object.HttpContext.User)).Returns("1");
        _mockDbContext.Setup(db => db.Nutritionist).ReturnsDbSet(new List<Nutritionist> { new Nutritionist { Id = "1", AspUserId = "1" } });
        _mockDbContext.Setup(db => db.DietPlan).ReturnsDbSet(new List<DietPlan>());
        _mockDbContext.Setup(db => db.Recipe).ReturnsDbSet(new List<Recipe>());
        _mockDbContext.Setup(db => db.PremiumUser).ReturnsDbSet(new List<PremiumUser>());

        // Create deletePlans with count = 0
        var deletePlans = new List<DietPlan>();

        // Act
        var result = await _controller.Create("userId", dietPlanVm);

        // Assert
        Assert.IsInstanceOfType(result, typeof(RedirectToActionResult));
        var redirectResult = (RedirectToActionResult)result;
        Assert.AreEqual("Index", redirectResult.ActionName);
        Assert.AreEqual("Nutritionist", redirectResult.ControllerName);

        // Provjera da li se nisu brisali stari planovi ishrane
        var deletedDietPlans = _context.DietPlan.ToList();
        Assert.AreEqual(0, deletedDietPlans.Count, "No diet plans should be deleted.");
    }

}
```

Pored POST metode za Create, testirala sam i samu Create metodu. Za ovu metodu sam pokrila neke moguće slučajeve, ali s obzirom na to da nije previše kompleksna metoda nije napisano ni previše testova.



```
    [TestMethod]
    | 0 references | Please sign-in to New Relic CodeStream to see Code Level Metrics
    public void CreateAction_WithRecipes_ReturnsCorrectViewModel()
    {
        // Aranžman (Arrange)

        // Kreiramo mock context.Recipe sa popunjenom listom recepata
        var mockRecipes = new List<Recipe>
{
    new Recipe { RID = 1, NameOfRecipe = "Recipe 1" },
    new Recipe { RID = 2, NameOfRecipe = "Recipe 2" },
    // Dodajte dodatne recepte prema potrebi
}.AsQueryable();

        _mockDbContext.Setup(c => c.Recipe).Returns((DbSet<Recipe>)MockDbSet(mockRecipes));

        // Akcija (Act)
        var result = _controller.Create("TestUser") as ViewResult;

        // Assert

        // Provjeravamo da li akcija vraća ViewResult
        Assert.IsNotNull(result);
        Assert.IsInstanceOfType(result, typeof(ViewResult));

        // Provjeravamo da li akcija koristi ispravan View
        Assert.AreEqual("dietPlan", result?.ViewName);

        // Provjeravamo da li ViewModel ima pravilno postavljene vrednosti
        var model = result.Model as EditDietPlanVM;
        Assert.IsNotNull(model);
        Assert.IsNotNull(model.DietPlan);
        Assert.IsNotNull(model.Recipes);
        Assert.AreEqual("TestUser", model.DietPlan.PremiumUser.Id);
        CollectionAssert.AreEqual(mockRecipes.ToList(), model.Recipes.ToList());
    }
```



```
| 0 references | Please sign-in to New Relic CodeStream to see Code Level Metrics
public void CreateAction_AuthorizedNutritionist_ReturnsCorrectViewModel()
{
    // Aranžman (Arrange)

    // Kreiramo mock context.Recipe
    var mockRecipes = new List<Recipe>
    {
        new Recipe { RID = 1, NameOfRecipe = "Recipe 1" },
        new Recipe { RID = 2, NameOfRecipe = "Recipe 2" },
        // Dodajte dodatne recepte prema potrebi
    }.AsQueryable();

    _mockDbContext.Setup(c => c.Recipe).Returns((DbSet<Recipe>)MockDbSet(mockRecipes));

    // Akcija (Act)
    var result = _controller.Create("TestUser") as ViewResult;

    // Assert

    // Proveravamo da li akcija vraća ViewResult
    Assert.IsNotNull(result);
    Assert.IsInstanceOfType(result, typeof(ViewResult));

    // Proveravamo da li akcija koristi ispravan View
    Assert.AreEqual("dietPlan", result?.ViewName);


    // Proveravamo da li ViewModel ima pravilno postavljene vrijednosti
    var model = result.Model as EditDietPlanVM;
    Assert.IsNotNull(model);
    Assert.IsNotNull(model.DietPlan);
    Assert.IsNotNull(model.Recipes);
    Assert.AreEqual("TestUser", model.DietPlan.PremiumUser.Id);
    CollectionAssert.AreEqual(mockRecipes.ToList(), model.Recipes.ToList());
}
```