

Informatikos fakultetas

**T120B162 Programų sistemų testavimas**

**2 laboratorinis darbas**

|  |  |  |
| --- | --- | --- |
|  | Studentas: | Mangirdas Kazlauskas, IFF-4/1 |
| Dėstytojas: | doc. Šarūnas Packevičius |

Kaunas 2017

Turinys

[1. Įvadas 3](#_Toc500084212)

[2. Testavimo priemonės 3](#_Toc500084213)

[3. Testuojamo API programos kodas 3](#_Toc500084214)

[4. Pasiruošimas testavimui 8](#_Toc500084215)

[5. API programos kodo padengimas testais prieš testų sukūrimą 12](#_Toc500084216)

[6. Testavimo atvejai ir jų kodas 13](#_Toc500084217)

[7. API programos kodo padengimas testais po testų sukūrimo 33](#_Toc500084218)

[8. Išvados 33](#_Toc500084219)

# Įvadas

**2 laboratorinio darbo tikslas** – ištestuoti kuriamos sistemos komponentus, sukuriant komponentų vienetų (angl. Unit) testus.

**Darbo uždaviniai:**

1. Susirasti bei tinkamai panaudoti testavimo priemones sistemos komponentų testavimui;
2. Išsiaiškinti, kaip aprašomi bei sukuriami vienetų testai;
3. Sukurti vienetų testus visiems API metodams;
4. Vienetų testais padengti 100% API programos kodo.

# Testavimo priemonės

Atsižvelgiant į kuriamos sistemos kūrimo priemones bei naudojamą karkasą, testų kūrimui pasirinkta naudotis tokias priemones:

1. Testų rašymo karkasas – xUnit (v2.3.1);
2. Programos objektų funkcionalumo imitavimas – Moq (v4.7.145);
3. Priemonė testų vykdymui – JetBrains ReSharper Ultimate 2017.2.2;
4. Įrankis, skirtas nustatyti programos kodo padengimą testais – JetBrains dotCover 2017.2.2;

# Testuojamo API programos kodas

Žemiau pateikiamas programos kodas, kuris bus padengiamas komponentų testais. Iš viso testais bus padengiamos 5 API klasės:

1 lentelė. Testinio API kontrolerio programos kodas

|  |
| --- |
| TestController.cs |
| using Microsoft.AspNetCore.Mvc;  namespace ScatterifyAPI.Controllers  {  public class TestController : Controller  {  [HttpGet]  [Route("api/ping")]  public IActionResult TestConnectionToApi()  {  return Ok("API is working!");  }  }  } |

2 lentelė. Autentifikacijos API kontrolerio programos kodas

|  |
| --- |
| AuthController.cs |
| using System.Threading.Tasks;  using AutoMapper;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using ScatterifyAPI.Entities;  using ScatterifyAPI.Models;  using ScatterifyAPI.Services;  namespace ScatterifyAPI.Controllers  {  [Authorize(Policy="FacebookAuthentication")]  public class AuthController : Controller  {  private readonly IUsersRepository \_usersRepository;  private readonly IFacebookService \_facebookService;  public AuthController(IUsersRepository usersRepository, IFacebookService facebookService)  {  \_usersRepository = usersRepository;  \_facebookService = facebookService;  }  [HttpPost]  [AllowAnonymous]  [Route("auth/facebook")]  public async Task<IActionResult> FacebookLogin([FromBody] FacebookLoginRequestDto request)  {  var response = await \_facebookService.FacebookLogin(request);  if (response == null)  {  return BadRequest();  }  if (!\_usersRepository.UserExists(response.userID))  {  var userForCreation = new UserForCreationDto  {  Username = response.userID  };  var newUser = Mapper.Map<User>(userForCreation);  \_usersRepository.CreateUser(newUser);  if (!\_usersRepository.Save())  {  return StatusCode(500, "A problem happened while handling your request.");  }  }  response.roleID = \_usersRepository.GetUser(response.userID).Role.Id;  return Ok(Mapper.Map<FacebookLoginResponseDto>(response));  }  }  } |

3 lentelė. Prekybos vietų API kontrolerio programos kodas

|  |
| --- |
| BranchesController.cs |
| using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using ScatterifyAPI.Services;  namespace ScatterifyAPI.Controllers  {  [Authorize(Policy = "FacebookAuthentication")]  [Route("api/branches")]  public class BranchesController : Controller  {  private readonly IBranchesRepository \_branchesRepository;  private readonly IUsersRepository \_usersRepository;  public BranchesController(IBranchesRepository branchesRepository, IUsersRepository usersRepository)  {  \_branchesRepository = branchesRepository;  \_usersRepository = usersRepository;  }  [HttpGet]  public IActionResult GetBranches()  {  var user = \_usersRepository.GetAuthenticatedUser(HttpContext.Request);  if (user == null)  {  return BadRequest();  }  return Ok(\_branchesRepository.GetBranches(user));  }  }  } |

4 lentelė. Užsakymų API kontrolerio programos kodas

|  |
| --- |
| OrdersController.cs |
| using AutoMapper;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using ScatterifyAPI.Models;  using ScatterifyAPI.Services;  using System.Collections.Generic;  using System.Threading.Tasks;  namespace ScatterifyAPI.Controllers  {  [Authorize(Policy = "FacebookAuthentication")]  [Route("api")]  public class OrdersController : Controller  {  private readonly IOrdersRepository \_orderRepository;  private readonly IBranchesRepository \_branchesRepository;  public OrdersController(IOrdersRepository orderRepository, IBranchesRepository branchesRepository)  {  \_orderRepository = orderRepository;  \_branchesRepository = branchesRepository;  }  [HttpGet("branches/{branchResourceID}/orders")]  public async Task<IActionResult> GetOrdersAsync(string branchResourceId)  {  if (!\_branchesRepository.BranchExistsAsync(branchResourceId).Result)  {  return NotFound();  }  var orderEntities = await \_orderRepository.GetOrdersAsync(branchResourceId);  var result = Mapper.Map<ICollection<OrderDto>>(orderEntities);  return Ok(result);  }  // **TODO: change to patch**  [HttpPut("orders/{orderResourceID}")]  public async Task<IActionResult> UpdateOrderStatusAsync(string orderResourceId, [FromBody] OrderUpdateDto orderUpdateDto)  {  if (orderUpdateDto == null)  {  return BadRequest();  }  var orderEntity = await \_orderRepository.GetOrderAsync(orderResourceId);  if (orderEntity == null)  {  return NotFound();  }  Mapper.Map(orderUpdateDto, orderEntity);  if (!\_orderRepository.Save())  {  return StatusCode(500, "A problem happened while handling your request.");  }  return Ok(Mapper.Map<OrderDto>(orderEntity));  }  }  } |

5 lentelė. Organizacijų API kontrolerio programos kodas

|  |
| --- |
| OrganizationsController.cs |
| using AutoMapper;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using ScatterifyAPI.Entities;  using ScatterifyAPI.Models;  using ScatterifyAPI.Services;  using System.Threading.Tasks;  namespace ScatterifyAPI.Controllers  {  [Authorize(Policy = "FacebookAuthentication")]  [Route("api/organizations")]  public class OrganizationsController : Controller  {  private readonly IOrganizationsRepository \_organizationsRepository;  private readonly IUsersRepository \_usersRepository;  private readonly IBranchUsersRepository \_branchUsersRepository;  public OrganizationsController(IOrganizationsRepository organizationsRepository, IUsersRepository usersRepository, IBranchUsersRepository branchUsersRepository)  {  \_organizationsRepository = organizationsRepository;  \_usersRepository = usersRepository;  \_branchUsersRepository = branchUsersRepository;  }  [HttpGet]  public async Task<IActionResult> GetOrganizationWithBranchesAsync()  {  var user = \_usersRepository.GetAuthenticatedUser(HttpContext.Request);  if (user == null)  {  return BadRequest();  }  return Ok(await \_organizationsRepository.GetOrganizationWithBranches(user));  }  [HttpPost]  public async Task<IActionResult> CreateNewOrganizationAsync([FromBody] OrganizationForCreationDto organization)  {  var user = \_usersRepository.GetAuthenticatedUser(HttpContext.Request);  if (user == null)  {  return BadRequest();  }  if (organization == null)  {  return BadRequest();  }  var newBranch = Mapper.Map<Branch>(organization);  \_organizationsRepository.CreateOrganization(newBranch);  if (!\_organizationsRepository.Save())  {  return StatusCode(500, "A problem happened while handling your request.");  }  var newBranchUser = Mapper.Map<BranchUser>(new BranchUserForCreationDto  {  BranchId = newBranch.Id,  UserId = user.Id  });  \_branchUsersRepository.CreateBranchUser(newBranchUser);  if (!\_branchUsersRepository.Save())  {  return StatusCode(500, "A problem happened while handling your request.");  }  return Ok(await \_organizationsRepository.GetOrganizationWithBranches(user));  }  [HttpPut("{organizationResourceID}")]  public async Task<IActionResult> UpdateOrganizationAsync([FromBody] OrganizationForCreationDto organization, string organizationResourceId)  {  var user = \_usersRepository.GetAuthenticatedUser(HttpContext.Request);  if(user == null)  {  return BadRequest();  }  if (organization == null)  {  return BadRequest();  }  var organizationEntity = await \_organizationsRepository.GetOrganization(organizationResourceId);  if (organizationEntity == null)  {  return NotFound();  }  Mapper.Map(organization, organizationEntity);  if (!\_organizationsRepository.Save())  {  return StatusCode(500, "A problem happened while handling your request.");  }  return Ok(await \_organizationsRepository.GetOrganizationWithBranches(user));  }  }  } |

# Pasiruošimas testavimui

Prieš aprašant API komponentų testus, reikia tinkamai paruošti testavimo failų struktūrą, bei sukurti reikiamus elementus. Pirmiausia sukuriamos komponentų testų klasės (kiekvienai API klasei po atskirą testavimo klasę):

6 lentelė. Testinio API kontrolerio testavimo klasė

|  |
| --- |
| TestControllerUnitTests.cs |
| namespace ScatterifyAPI.Tests.UnitTests  {  public class TestControllerUnitTests  {  }  } |

7 lentelė. Autentifikacijos API kontrolerio testavimo klasė

|  |
| --- |
| AuthControllerUnitTests.cs |
| using System;  using Moq;  using ScatterifyAPI.Services;  using Xunit;  namespace ScatterifyAPI.Tests.UnitTests  {  [Collection("Mapper collection")]  public class AuthControllerUnitTests : IDisposable  {  private readonly Mock<IUsersRepository> \_usersRepository;  private readonly Mock<IFacebookService> \_facebookServiceMock;  public AuthControllerUnitTests()  {  \_usersRepository = new Mock<IUsersRepository>();  \_facebookServiceMock = new Mock<IFacebookService>();  }  public void Dispose()  {  \_usersRepository.Reset();  \_facebookServiceMock.Reset();  }  }  } |

8 lentelė. Organizacijų API kontrolerio testavimo klasė

|  |
| --- |
| OrganizationsControllerUnitTests.cs |
| using Microsoft.AspNetCore.Http;  using Moq;  using ScatterifyAPI.Services;  using System;  using Xunit;  namespace ScatterifyAPI.Tests.UnitTests  {  [Collection("Mapper collection")]  public class OrganizationsControllerUnitTests : IDisposable  {  private readonly Mock<HttpContext> \_context;  private readonly Mock<HttpRequest> \_request;  private readonly Mock<IOrganizationsRepository> \_organizationsRepository;  private readonly Mock<IUsersRepository> \_usersRepository;  private readonly Mock<IBranchUsersRepository> \_branchUsersRepository;  public OrganizationsControllerUnitTests()  {  \_context = new Mock<HttpContext>();  \_request = new Mock<HttpRequest>();  \_organizationsRepository = new Mock<IOrganizationsRepository>();  \_usersRepository = new Mock<IUsersRepository>();  \_branchUsersRepository = new Mock<IBranchUsersRepository>();  }    public void Dispose()  {  \_context.Reset();  \_request.Reset();  \_organizationsRepository.Reset();  \_usersRepository.Reset();  \_branchUsersRepository.Reset();  }  }  } |

9 lentelė. Užsakymų API kontrolerio testavimo klasė

|  |
| --- |
| OrdersControllerUnitTests.cs |
| using System;  using Moq;  using ScatterifyAPI.Services;  using Xunit;  namespace ScatterifyAPI.Tests.UnitTests  {  [Collection("Mapper collection")]  public class OrdersControllerUnitTests : IDisposable  {  private readonly Mock<IOrdersRepository> \_ordersRepository;  private readonly Mock<IBranchesRepository> \_branchesRepository;  public OrdersControllerUnitTests()  {  \_ordersRepository = new Mock<IOrdersRepository>();  \_branchesRepository = new Mock<IBranchesRepository>();  }  public void Dispose()  {  \_ordersRepository.Reset();  \_branchesRepository.Reset();  }  }  } |

10 lentelė. Prekybos vietų API kontrolerio testavimo klasė

|  |
| --- |
| BranchesControllerUnitTests.cs |
| using Microsoft.AspNetCore.Http;  using Moq;  using ScatterifyAPI.Services;  using System;  namespace ScatterifyAPI.Tests.UnitTests  {  public class BranchesControllerUnitTests : IDisposable  {  private readonly Mock<HttpContext> \_context;  private readonly Mock<HttpRequest> \_request;  private readonly Mock<IUsersRepository> \_usersRepository;  private readonly Mock<IBranchesRepository> \_branchesRepository;  public BranchesControllerUnitTests()  {  \_context = new Mock<HttpContext>();  \_request = new Mock<HttpRequest>();  \_usersRepository = new Mock<IUsersRepository>();  \_branchesRepository = new Mock<IBranchesRepository>();  }  public void Dispose()  {  \_context.Reset();  \_request.Reset();  \_usersRepository.Reset();  \_branchesRepository.Reset();  }  }  } |

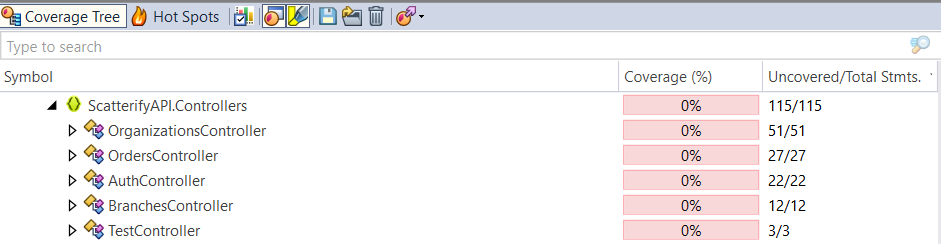
Taip pat sukurta papildoma klasė MapperFixture, kuri reikalinga duomenų objektų surišimo sukūrimui prieš vykdant visus testus.

11 lentelė. Duomenų objektų surišimo sukūrimo klasė

|  |
| --- |
| MapperFixture.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using AutoMapper;  using ScatterifyAPI.Entities;  using ScatterifyAPI.Models;  using Xunit;  namespace ScatterifyAPI.Tests.Fixtures  {  public class MapperFixture : IDisposable  {  public MapperFixture()  {  Mapper.Initialize(cfg => {  cfg.CreateMap<Order, OrderDto>()  .ForMember(dest => dest.branchResourceID, opt => opt.MapFrom(src => src.OrderProducts.Select(op => op.Product.ResourceId).FirstOrDefault()))  .ForMember(dest => dest.orderResourceID, opt => opt.MapFrom(src => src.ResourceId))  .ForMember(dest => dest.orderStatusID, opt => opt.MapFrom(src => src.Status))  .ForMember(dest => dest.orderCreatedAt, opt => opt.Equals(0))  .ForMember(dest => dest.clientUsername, opt => opt.MapFrom(src => src.Client.Username))  .ForMember(dest => dest.clientFullName, opt => opt.MapFrom(src => src.Client.FullName))  .ForMember(dest => dest.clientPhotoUrl, opt => opt.MapFrom(src => src.Client.PhotoUrl))  .ForMember(dest => dest.orderItems, opt => opt.MapFrom(src => Mapper.Map<ICollection<OrderItemDto>>(src.OrderProducts)));  cfg.CreateMap<OrderProduct, OrderItemDto>()  .ForMember(dest => dest.productTitle, opt => opt.MapFrom(src => src.Product.Title))  .ForMember(dest => dest.productPrice, opt => opt.MapFrom(src => src.Product.Price))  .ForMember(dest => dest.categoriesTree, opt => opt.MapFrom(src => string.Join(',', src.InverseOrderProductParent.Select(iop => iop.Product.Title).ToList())));  cfg.CreateMap<UserForCreationDto, User>();  cfg.CreateMap<Branch, BranchDto>();  cfg.CreateMap<UserDto, FacebookLoginResponseDto>();  cfg.CreateMap<OrderUpdateDto, Order>();  cfg.CreateMap<OrganizationForCreationDto, Branch>();  cfg.CreateMap<BranchUserForCreationDto, BranchUser>();  });  }  public void Dispose()  {  Mapper.Reset();  }  }  [CollectionDefinition("Mapper collection")]  public class MapperCollection : ICollectionFixture<MapperFixture> { }  } |

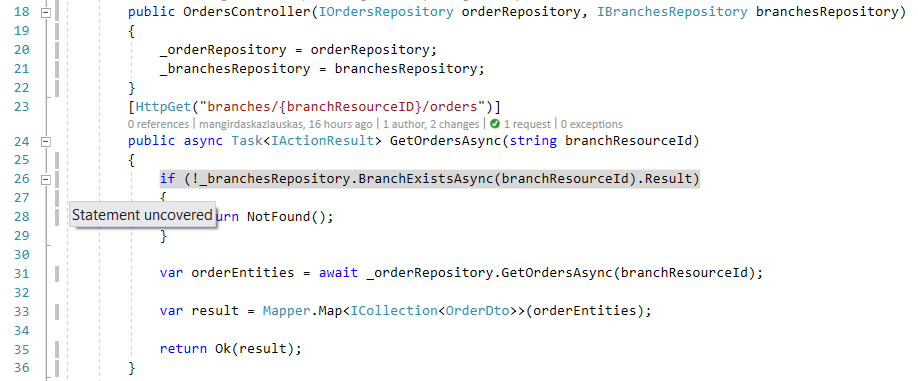
# API programos kodo padengimas testais prieš testų sukūrimą

Prieš sukuriant komponentų testus, dotCover įrankiu buvo patikrintas pirminis API programos kodo padengimas. Kaip ir buvo tikėtasi, įrankis parodo, kad komponentų testais yra padengta 0% kodo (1 pav.).



1 pav. API programos kodo padengimas prieš sukuriant testus

Taip pat prie atitinkamo programos kodo (API metodų kodo eilučių) dotCover įrankis rodo, kad eilutės yra nepadengtos jokiais testais – prie atitinkamų eilučių yra rodomas pilkas laukelis (2 pav.).



2 pav. Užsakymo kontrolerio kodo eilutės nėra nepadengtos testais

# Testavimo atvejai ir jų kodas

Šiame skyriuje pateikiami aprašyti komponentų testai. Kiekvienas testas yra pateikiamas lentele, kurią sudaro 3 dalys: 1-oji nurodo komponento testo pavadinimą, antroji – ką testuoja atitinkamas testas, o trečioje lentelės dalyje pateikiamas testo kodas. Iš viso buvo parašyti 33 komponentų testai.

1. **TestController** komponentų testai:

|  |
| --- |
| Should\_Return\_Ok\_Result\_When\_Testing\_Connection\_To\_Api |
| Testuojama, ar galima siųsti užklausą į API. |
| [Fact]  public void Should\_Return\_Ok\_Result\_When\_Testing\_Connection\_To\_Api()  {  // Arrange  var controller = new TestController();  // Act  var result = controller.TestConnectionToApi();  // Assert  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  okResult.Value.Should().Be("API is working!");  } |

1. **AuthController** komponentų testai:

|  |
| --- |
| Should\_Not\_be\_Null\_With\_Valid\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su teisingais parametrais. |
| [Fact]  public void Should\_Not\_be\_Null\_With\_Valid\_Parameters()  {  // Arrange  // Act  var controller = new AuthController(\_usersRepository.Object, \_facebookServiceMock.Object);  // Assert  var authController = controller.Should().BeOfType<AuthController>().Subject;  authController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Not\_Be\_Null\_With\_Null\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su tuščiais parametrais. |
| [Fact]  public void Should\_Not\_Be\_Null\_With\_Null\_Parameters()  {  // Arrange  // Act  var controller = new AuthController(null, null);  // Assert  var authController = controller.Should().BeOfType<AuthController>().Subject;  authController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_Facebook\_Login\_Request\_Is\_Null\_On\_Facebook\_Login |
| Testuojamas Facebook prisijungimo metodas, kai siunčiamas užklausos objektas yra netinkamas. |
| [Fact]  public async Task Should\_Return\_Bad\_Request\_When\_Facebook\_Login\_Request\_Is\_Null\_On\_Facebook\_Login()  {  // Arrange  \_facebookServiceMock.Setup(mockService => mockService.FacebookLogin(It.IsAny<FacebookLoginRequestDto>())).ReturnsAsync(() => null);  var controller = new AuthController(\_usersRepository.Object, \_facebookServiceMock.Object);  // Act  var result = await controller.FacebookLogin(new FacebookLoginRequestDto());  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

|  |
| --- |
| Should\_Return\_Internal\_Server\_Error\_When\_New\_User\_Is\_Not\_Saved\_On\_Facebook\_Login |
| Testuojamas Facebook prisijungimo metodas, kai naujas sistemos naudotojas nėra išsaugomas pirmo prisijungimo prie sistemos metu. |
| [Fact]  public async Task Should\_Return\_Internal\_Server\_Error\_When\_New\_User\_Is\_Not\_Saved\_On\_Facebook\_Login()  {  // Arrange  \_facebookServiceMock.Setup(mockService => mockService.FacebookLogin(It.IsAny<FacebookLoginRequestDto>())).ReturnsAsync(() => new UserDto());  \_usersRepository.Setup(mockRep => mockRep.UserExists(It.IsAny<string>())).Returns(false);  \_usersRepository.Setup(mockRep => mockRep.Save()).Returns(false);  var controller = new AuthController(\_usersRepository.Object, \_facebookServiceMock.Object);  // Act  var result = await controller.FacebookLogin(new FacebookLoginRequestDto());  // Assert  var internalServerErrorResult = result.Should().BeOfType<ObjectResult>().Subject;  internalServerErrorResult.StatusCode.Should().Be(500);  internalServerErrorResult.Value.Should().Be("A problem happened while handling your request.");  } |

|  |
| --- |
| Should\_Return\_Facebook\_Login\_Response\_When\_New\_User\_Is\_Saved\_Successfully\_On\_Facebook\_Login |
| Testuojamas Facebook prisijungimo metodas, kai naujas sistemos naudotojas yra išsaugomas pirmo prisijungimo prie sistemos metu. |
| [Fact]  public async Task Should\_Return\_Facebook\_Login\_Response\_When\_New\_User\_Is\_Saved\_Successfully\_On\_Facebook\_Login()  {  // Arrange  \_facebookServiceMock.Setup(mockService => mockService.FacebookLogin(It.IsAny<FacebookLoginRequestDto>())).ReturnsAsync(() => new UserDto());  \_usersRepository.Setup(mockRep => mockRep.UserExists(It.IsAny<string>())).Returns(false);  \_usersRepository.Setup(mockRep => mockRep.Save()).Returns(true);  \_usersRepository.Setup(mockRep => mockRep.GetUser(It.IsAny<string>())).Returns(() => new User  {  Role = new Role  {  Id = 1  }  });  var controller = new AuthController(\_usersRepository.Object, \_facebookServiceMock.Object);  // Act  var result = await controller.FacebookLogin(new FacebookLoginRequestDto());  // Assert  // Verify that Save and CreateUser methods were called once in usersRepository  \_usersRepository.Verify(mockRep => mockRep.CreateUser(It.IsAny<User>()), Times.Once);  \_usersRepository.Verify(mockRep => mockRep.Save(), Times.Once);  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  okResult.Value.Should().BeAssignableTo<FacebookLoginResponseDto>();  } |

|  |
| --- |
| Should\_Return\_Facebook\_Login\_Response\_When\_User\_Exists\_On\_Facebook\_Login |
| Testuojamas Facebook prisijungimo metodas, kai sistemos naudotojas prie sistemos jungiasi ne pirmą kartą – jau egzistuoja. |
| [Fact]  public async Task Should\_Return\_Facebook\_Login\_Response\_When\_User\_Exists\_On\_Facebook\_Login()  {  // Arrange  \_facebookServiceMock.Setup(mockService => mockService.FacebookLogin(It.IsAny<FacebookLoginRequestDto>())).ReturnsAsync(() => new UserDto());  \_usersRepository.Setup(mockRep => mockRep.UserExists(It.IsAny<string>())).Returns(true);  \_usersRepository.Setup(mockRep => mockRep.Save()).Returns(true);  \_usersRepository.Setup(mockRep => mockRep.GetUser(It.IsAny<string>())).Returns(() => new User  {  Role = new Role  {  Id = 1  }  });  var controller = new AuthController(\_usersRepository.Object, \_facebookServiceMock.Object);  // Act  var result = await controller.FacebookLogin(new FacebookLoginRequestDto());  // Assert  // Verify that Save and CreateUser methods were not called in usersRepository  \_usersRepository.Verify(mockRep => mockRep.CreateUser(It.IsAny<User>()), Times.Never);  \_usersRepository.Verify(mockRep => mockRep.Save(), Times.Never);  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  okResult.Value.Should().BeAssignableTo<FacebookLoginResponseDto>();  } |

1. **OrganizationsController** komponentų testai:

|  |
| --- |
| Should\_Not\_be\_Null\_With\_Valid\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su teisingais parametrais. |
| [Fact]  public void Should\_Not\_Be\_Null\_With\_Valid\_Parameters()  {  // Arrange  // Act  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object);  // Assert  var organizationsController = controller.Should().BeOfType<OrganizationsController>().Subject;  organizationsController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Not\_Be\_Null\_With\_Null\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su tuščiais parametrais. |
| [Fact]  public void Should\_Not\_Be\_Null\_With\_Null\_Parameters()  {  // Arrange  // Act  var controller = new OrganizationsController(null, null, null);  // Assert  var organizationsController = controller.Should().BeOfType<OrganizationsController>().Subject;  organizationsController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_User\_Is\_Null |
| Testuojamas GetOrganizationWithBranchesAsync() metodas, kai autentifikuotas naudotojas nėra randamas. |
| [Fact]  public async Task Should\_Return\_Bad\_Request\_When\_User\_Is\_Null()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => null);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.GetOrganizationWithBranchesAsync();  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

|  |
| --- |
| Should\_Return\_Organizations\_With\_Branches\_List\_When\_User\_Exists |
| Testuojamas GetOrganizationWithBranchesAsync() metodas, kai autentifikuotas naudotojas yra randamas. |
| [Fact]  public async Task Should\_Return\_Organizations\_With\_Branches\_List\_When\_User\_Exists()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_organizationsRepository.Setup(mockRep => mockRep.GetOrganizationWithBranches(It.IsAny<User>())).ReturnsAsync(new List<OrganizationDto>  {  new OrganizationDto { organizationResourceID = "1"},  new OrganizationDto { organizationResourceID = "2"},  new OrganizationDto { organizationResourceID = "3"}  });  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.GetOrganizationWithBranchesAsync();  // Assert  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  var organizations = okResult.Value.Should().BeAssignableTo<IEnumerable<OrganizationDto>>().Subject;  organizations.Count().Should().Be(3);  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_User\_Is\_Null\_On\_Organization\_Create |
| Testuojamas organizacijos sukūrimo metodas, kai autentifikuotas naudotojas nėra randamas. |
| [Fact]  public async Task Should\_Return\_Bad\_Request\_When\_User\_Is\_Null\_On\_Organization\_Create()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => null);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.CreateNewOrganizationAsync(new OrganizationForCreationDto());  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_Request\_Body\_Is\_Null\_On\_Organization\_Create |
| Testuojamas organizacijos sukūrimo metodas, kai užklausos duomenys nėra teisingi. |
| [Fact]  public async Task Should\_Return\_Bad\_Request\_When\_Request\_Body\_Is\_Null\_On\_Organization\_Create()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.CreateNewOrganizationAsync(null);  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

|  |
| --- |
| Should\_Return\_Internal\_Server\_Error\_When\_Organization\_Is\_Not\_Saved\_On\_Organization\_Create |
| Testuojamas organizacijos sukūrimo metodas, kai organizacija nėra išsaugoma. |
| [Fact]  public async Task Should\_Return\_Internal\_Server\_Error\_When\_Organization\_Is\_Not\_Saved\_On\_Organization\_Create()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_organizationsRepository.Setup(mockRep => mockRep.CreateOrganization(It.IsAny<Branch>()));  \_organizationsRepository.Setup(mockRep => mockRep.Save()).Returns(false);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.CreateNewOrganizationAsync(new OrganizationForCreationDto());  // Assert  var internalServerError = result.Should().BeOfType<ObjectResult>().Subject;  internalServerError.StatusCode.Should().Be(500);  internalServerError.Value.Should().Be("A problem happened while handling your request.");  } |

|  |
| --- |
| Should\_Return\_Internal\_Server\_Error\_When\_Branch\_User\_Is\_Not\_Saved\_On\_Organization\_Create |
| Testuojamas organizacijos sukūrimo metodas, kai organizacijos vadybininkas nėra išsaugomas. |
| [Fact]  public async Task Should\_Return\_Internal\_Server\_Error\_When\_Branch\_User\_Is\_Not\_Saved\_On\_Organization\_Create()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_organizationsRepository.Setup(mockRep => mockRep.CreateOrganization(It.IsAny<Branch>()));  \_organizationsRepository.Setup(mockRep => mockRep.Save()).Returns(true);  \_branchUsersRepository.Setup(mockRep => mockRep.CreateBranchUser(It.IsAny<BranchUser>()));  \_branchUsersRepository.Setup(mockRep => mockRep.Save()).Returns(false);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.CreateNewOrganizationAsync(new OrganizationForCreationDto());  // Assert  var internalServerError = result.Should().BeOfType<ObjectResult>().Subject;  internalServerError.StatusCode.Should().Be(500);  internalServerError.Value.Should().Be("A problem happened while handling your request.");  } |

|  |
| --- |
| Should\_Return\_Organizations\_With\_Branches\_List\_On\_Successful\_Organization\_Create |
| Testuojamas organizacijos sukūrimo metodas, kai organizacija yra sėkmingai išsaugoma. |
| [Fact]  public async Task Should\_Return\_Organizations\_With\_Branches\_List\_On\_Successful\_Organization\_Create()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_organizationsRepository.Setup(mockRep => mockRep.CreateOrganization(It.IsAny<Branch>()));  \_organizationsRepository.Setup(mockRep => mockRep.Save()).Returns(true);  \_organizationsRepository.Setup(mockRep => mockRep.GetOrganizationWithBranches(It.IsAny<User>())).ReturnsAsync(new List<OrganizationDto>  {  new OrganizationDto { organizationResourceID = "1"},  new OrganizationDto { organizationResourceID = "2"},  new OrganizationDto { organizationResourceID = "3"}  });  \_branchUsersRepository.Setup(mockRep => mockRep.CreateBranchUser(It.IsAny<BranchUser>()));  \_branchUsersRepository.Setup(mockRep => mockRep.Save()).Returns(true);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.CreateNewOrganizationAsync(new OrganizationForCreationDto());  // Assert  // Verify that CreateOrganization and Save methods were called once in organizationsRepository  \_organizationsRepository.Verify(mockRep => mockRep.CreateOrganization(It.IsAny<Branch>()), Times.Once);  \_organizationsRepository.Verify(mockRep => mockRep.Save(), Times.Once);  // Verify that CreateBranchUser and Save methods were called once in branchUsersRepository  \_branchUsersRepository.Verify(mockRep => mockRep.CreateBranchUser(It.IsAny<BranchUser>()), Times.Once);  \_branchUsersRepository.Verify(mockRep => mockRep.Save(), Times.Once);  // Result  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  var organizations = okResult.Value.Should().BeAssignableTo<IEnumerable<OrganizationDto>>().Subject;  organizations.Should().NotBeNull();  organizations.Count().Should().Be(3);  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_User\_Is\_Null\_On\_Organization\_Update |
| Testuojamas organizacijos duomenų pakeitimo metodas, kai autentifikuotas naudotojas nėra randamas. |
| [Fact]  public async Task Should\_Return\_Bad\_Request\_When\_User\_Is\_Null\_On\_Organization\_Update()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => null);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.UpdateOrganizationAsync(new OrganizationForCreationDto(), null);  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_Organization\_Is\_Null\_On\_Organization\_Update |
| Testuojamas organizacijos duomenų pakeitimo metodas, kai užklausos duomenys nėra teisingi. |
| [Fact]  public async Task Should\_Return\_Bad\_Request\_When\_Organization\_Is\_Null\_On\_Organization\_Update()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.UpdateOrganizationAsync(null, null);  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

|  |
| --- |
| Should\_Return\_Internal\_Server\_Error\_When\_Organization\_Changes\_Are\_Not\_Saved\_On\_Organization\_Update |
| Testuojamas organizacijos duomenų pakeitimo metodas, kai organizacija nėra išsaugoma. |
| [Fact]  public async Task Should\_Return\_Internal\_Server\_Error\_When\_Organization\_Changes\_Are\_Not\_Saved\_On\_Organization\_Update()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_organizationsRepository.Setup(mockRep => mockRep.GetOrganization(It.IsAny<string>())).ReturnsAsync(() => new Branch  {  ResourceId = "1",  Title = "New Branch"  });  \_organizationsRepository.Setup(mockRep => mockRep.Save()).Returns(false);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.UpdateOrganizationAsync(new OrganizationForCreationDto(), "1");  // Assert  var internalServerError = result.Should().BeOfType<ObjectResult>().Subject;  internalServerError.StatusCode.Should().Be(500);  internalServerError.Value.Should().Be("A problem happened while handling your request.");  } |

|  |
| --- |
| Should\_Return\_Not\_Found\_When\_Organization\_With\_Provided\_ID\_Does\_Not\_Exist\_On\_Organization\_Update |
| Testuojamas organizacijos duomenų pakeitimo metodas, kai organizacija, kurios duomenis norima pakeisti, nėra randama. |
| [Fact]  public async Task Should\_Return\_Not\_Found\_When\_Organization\_With\_Provided\_ID\_Does\_Not\_Exist\_On\_Organization\_Update()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_organizationsRepository.Setup(mockRep => mockRep.GetOrganization(It.IsAny<string>())).ReturnsAsync(() => null);  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.UpdateOrganizationAsync(new OrganizationForCreationDto(), "1");  // Assert  var notFoundResult = result.Should().BeOfType<NotFoundResult>().Subject;  notFoundResult.StatusCode.Should().Be(404);  } |

|  |
| --- |
| Should\_Return\_Organization\_With\_Branches\_List\_On\_Successful\_Organization\_Update |
| Testuojamas organizacijos duomenų pakeitimo metodas, kai organizacijos duomenys yra sėkmingai pakeičiami. |
| [Fact]  public async Task Should\_Return\_Organization\_With\_Branches\_List\_On\_Successful\_Organization\_Update()  {  // Arrange  \_context.Setup(c => c.Request).Returns(new Mock<HttpRequest>().Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_organizationsRepository.Setup(mockRep => mockRep.GetOrganization(It.IsAny<string>())).ReturnsAsync(() => new Branch  {  ResourceId = "1",  Title = "New Branch"  });  \_organizationsRepository.Setup(mockRep => mockRep.Save()).Returns(true);  \_organizationsRepository.Setup(mockRep => mockRep.GetOrganizationWithBranches(It.IsAny<User>())).ReturnsAsync(new List<OrganizationDto>  {  new OrganizationDto { organizationResourceID = "1"},  new OrganizationDto { organizationResourceID = "2"},  new OrganizationDto { organizationResourceID = "3"}  });  var controller = new OrganizationsController(\_organizationsRepository.Object, \_usersRepository.Object, \_branchUsersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = await controller.UpdateOrganizationAsync(new OrganizationForCreationDto() {  Title = "Updated Branch"  }, "1");  // Assert  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  var organizations = okResult.Value.Should().BeAssignableTo<IEnumerable<OrganizationDto>>().Subject;  organizations.Should().NotBeNull();  organizations.Count().Should().Be(3);  } |

1. **OrdersController** komponentų testai:

|  |
| --- |
| Should\_Not\_be\_Null\_With\_Valid\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su teisingais parametrais. |
| [Fact]  public void Should\_Not\_be\_Null\_With\_Valid\_Parameters()  {  // Arrange  // Act  var controller = new OrdersController(\_ordersRepository.Object, \_branchesRepository.Object);  // Assert  var ordersController = controller.Should().BeOfType<OrdersController>().Subject;  ordersController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Not\_Be\_Null\_With\_Null\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su tuščiais parametrais. |
| [Fact]  public void Should\_Not\_Be\_Null\_With\_Null\_Parameters()  {  // Arrange  // Act  var controller = new OrdersController(null, null);  // Assert  var ordersController = controller.Should().BeOfType<OrdersController>().Subject;  ordersController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Return\_Not\_Found\_When\_Branch\_Does\_Not\_Exist\_On\_Get\_Orders |
| Testuojamas GetOrdersAsync() metodas, kai norimų gražinti užsakymų pardavimo vieta neegzistuoja. |
| [Fact]  public async Task Should\_Return\_Not\_Found\_When\_Branch\_Does\_Not\_Exist\_On\_Get\_Orders()  {  // Arrange  \_branchesRepository.Setup(mockRep => mockRep.BranchExistsAsync(It.IsAny<string>())).ReturnsAsync(false);  var controller = new OrdersController(\_ordersRepository.Object, \_branchesRepository.Object);  // Act  var result = await controller.GetOrdersAsync(It.IsAny<string>());    // Assert  var notFoundResult = result.Should().BeOfType<NotFoundResult>().Subject;  notFoundResult.StatusCode.Should().Be(404);  } |

|  |
| --- |
| Should\_Return\_Orders\_List\_On\_Get\_Orders |
| Testuojamas GetOrdersAsync() metodas, kai grąžinami visi pardavimo vietai priskirti užsakymai. |
| [Fact]  public async Task Should\_Return\_Orders\_List\_On\_Get\_Orders()  {  // Arrange  \_branchesRepository.Setup(mockRep => mockRep.BranchExistsAsync(It.IsAny<string>())).ReturnsAsync(true);  \_ordersRepository.Setup(mockRep => mockRep.GetOrdersAsync(It.IsAny<string>())).ReturnsAsync(new List<Order>  {  new Order{ResourceId = "1"},  new Order{ResourceId = "2"},  new Order{ResourceId = "3"}  });  var controller = new OrdersController(\_ordersRepository.Object, \_branchesRepository.Object);  // Act  var result = await controller.GetOrdersAsync(It.IsAny<string>());  // Assert  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  var orders = okResult.Value.Should().BeAssignableTo<ICollection<OrderDto>>().Subject;  orders.Should().NotBeNull();  orders.Count().Should().Be(3);  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_Request\_Body\_Is\_Null\_On\_Order\_Status\_Update |
| Testuojamas užsakymo statuso pakeitimo metodas, kai užklausos duomenys yra neteisingi. |
| [Fact]  public async Task Should\_Return\_Bad\_Request\_When\_Request\_Body\_Is\_Null\_On\_Order\_Status\_Update()  {  // Arrange  var controller = new OrdersController(\_ordersRepository.Object, \_branchesRepository.Object);  // Act  var result = await controller.UpdateOrderStatusAsync(It.IsAny<string>(), null);  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

|  |
| --- |
| Should\_Return\_Not\_Found\_When\_Order\_Does\_Not\_Exist\_On\_Order\_Status\_Update |
| Testuojamas užsakymo statuso pakeitimo metodas, kai užsakymas, kurio duomenis norima pakeisti, neegzistuoja. |
| [Fact]  public async Task Should\_Return\_Not\_Found\_When\_Order\_Does\_Not\_Exist\_On\_Order\_Status\_Update()  {  // Arrange  \_ordersRepository.Setup(mockRep => mockRep.GetOrderAsync(It.IsAny<string>())).ReturnsAsync(() => null);  var controller = new OrdersController(\_ordersRepository.Object, \_branchesRepository.Object);  // Act  var result = await controller.UpdateOrderStatusAsync(It.IsAny<string>(), new OrderUpdateDto());  // Assert  var notFoundResult = result.Should().BeOfType<NotFoundResult>().Subject;  notFoundResult.StatusCode.Should().Be(404);  } |

|  |
| --- |
| Should\_Return\_Internal\_Server\_Error\_When\_Order\_Changes\_Are\_Not\_Saved\_On\_Order\_Status\_Update |
| Testuojamas užsakymo statuso pakeitimo metodas, kai pakeisti užsakymo duomenys nėra išsaugomi. |
| [Fact]  public async Task Should\_Return\_Internal\_Server\_Error\_When\_Order\_Changes\_Are\_Not\_Saved\_On\_Order\_Status\_Update()  {  // Arrange  \_ordersRepository.Setup(mockRep => mockRep.GetOrderAsync(It.IsAny<string>())).ReturnsAsync(new Order());  \_ordersRepository.Setup(mockRep => mockRep.Save()).Returns(false);  var controller = new OrdersController(\_ordersRepository.Object, \_branchesRepository.Object);  // Act  var result = await controller.UpdateOrderStatusAsync(It.IsAny<string>(), new OrderUpdateDto());  // Assert  var internalServerError = result.Should().BeOfType<ObjectResult>().Subject;  internalServerError.StatusCode.Should().Be(500);  internalServerError.Value.Should().Be("A problem happened while handling your request.");  } |

|  |
| --- |
| Should\_Return\_Updated\_Order\_On\_Order\_Status\_Update |
| Testuojamas užsakymo statuso pakeitimo metodas, kai pakeisti užsakymo duomenys yra išsaugomi. |
| [Fact]  public async Task Should\_Return\_Updated\_Order\_On\_Order\_Status\_Update()  {  // Arrange  \_ordersRepository.Setup(mockRep => mockRep.GetOrderAsync(It.IsAny<string>())).ReturnsAsync(new Order());  \_ordersRepository.Setup(mockRep => mockRep.Save()).Returns(true);  var controller = new OrdersController(\_ordersRepository.Object, \_branchesRepository.Object);  // Act  var result = await controller.UpdateOrderStatusAsync(It.IsAny<string>(), new OrderUpdateDto());  // Assert  // Verify that Save method was called once in ordersRepository  \_ordersRepository.Verify(mockRep => mockRep.Save(), Times.Once);  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  okResult.StatusCode.Should().Be(200);  var updatedOrder = okResult.Value.Should().BeAssignableTo<OrderDto>().Subject;  updatedOrder.Should().NotBeNull();  } |

1. **BranchesController** komponentų testai:

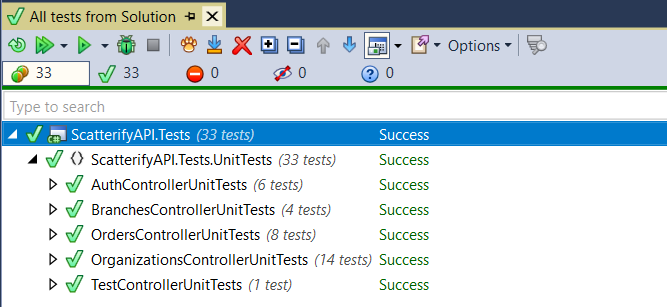
|  |
| --- |
| Should\_Not\_be\_Null\_With\_Valid\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su teisingais parametrais. |
| [Fact]  public void Should\_Not\_Be\_Null\_With\_Valid\_Parameters()  {  // Arrange  // Act  var controller = new BranchesController(\_branchesRepository.Object, \_usersRepository.Object);  // Assert  var branchesController = controller.Should().BeOfType<BranchesController>().Subject;  branchesController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Not\_Be\_Null\_With\_Null\_Parameters |
| Testuojamas kontrolerio konstruktoriaus sukūrimas su tuščiais parametrais. |
| [Fact]  public void Should\_Not\_Be\_Null\_With\_Null\_Parameters()  {  // Arrange  // Act  var controller = new BranchesController(null, null);  // Assert  var branchesController = controller.Should().BeOfType<BranchesController>().Subject;  branchesController.Should().NotBeNull();  } |

|  |
| --- |
| Should\_Return\_Branches\_With\_Top\_Organization\_List\_When\_User\_Exists |
| Testuojamas GetBranches() metodas, kai prekybos vietos yra sėkmingai grąžinamos. |
| [Fact]  public void Should\_Return\_Branches\_With\_Top\_Organization\_List\_When\_User\_Exists()  {  // Arrange  \_context.Setup(c => c.Request).Returns(\_request.Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => new User());  \_branchesRepository.Setup(mockRep => mockRep.GetBranches(It.IsAny<User>())).Returns(() => new List<BranchesWithOrganizationResponseDto>  {  new BranchesWithOrganizationResponseDto{organizationTitle="title1", organizationResourceID=new Guid().ToString(), organizationBranches=new List<BranchResponseDto>()},  new BranchesWithOrganizationResponseDto{organizationTitle="title2", organizationResourceID=new Guid().ToString(), organizationBranches=new List<BranchResponseDto>()},  new BranchesWithOrganizationResponseDto{organizationTitle="title3", organizationResourceID=new Guid().ToString(), organizationBranches=new List<BranchResponseDto>()}  });  var controller = new BranchesController(\_branchesRepository.Object, \_usersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = controller.GetBranches();  // Assert  var okResult = result.Should().BeOfType<OkObjectResult>().Subject;  var organizations = okResult.Value.Should().BeAssignableTo<IEnumerable<BranchesWithOrganizationResponseDto>>().Subject;  organizations.Count().Should().Be(3);  } |

|  |
| --- |
| Should\_Return\_Bad\_Request\_When\_User\_Is\_Null |
| Testuojamas GetBranches() metodas, kai autentifikuotas naudotojas nėra randamas. |
| [Fact]  public void Should\_Return\_Bad\_Request\_When\_User\_Is\_Null()  {  // Arrange  \_context.Setup(c => c.Request).Returns(\_request.Object);  \_usersRepository.Setup(mockRep => mockRep.GetAuthenticatedUser(It.IsAny<HttpRequest>())).Returns(() => null);  var controller = new BranchesController(\_branchesRepository.Object, \_usersRepository.Object)  {  ControllerContext = new ControllerContext  {  HttpContext = \_context.Object  }  };  // Act  var result = controller.GetBranches();  // Assert  var badRequestResult = result.Should().BeOfType<BadRequestResult>().Subject;  badRequestResult.StatusCode.Should().Be(400);  } |

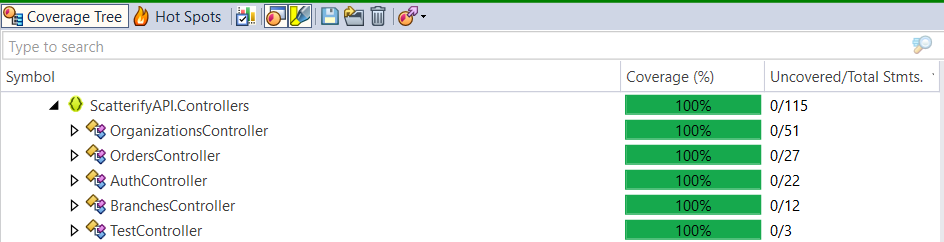
Naudojantis ReSharper testų paleidimo įrankiu nustatyta, kad visi testai yra teisingi (3 pav.)



3 pav. Sukurti komponentų testai yra teisingi

# API programos kodo padengimas testais po testų sukūrimo

Sukūros komponentų testus dar kartą buvo panaudotas įrankis dotCover, kuris šįkart jau parodė, kad visi API metodai yra 100% padengti testais (4 pav.).



4 pav. Testuojamo kodo padengimas testais

Taip pat prie API metodų kodo eilučių yra rodomi žali laukeliai, kurie nurodo, kad API metodų kodo eilutės yra padengtos testais (5 pav.).



5 pav. API programos kodo eilutės yra padengtos testais

# Išvados

1. Laboratoriniam darbui atlikti buvo surastos, kaip vėliau paaiškėjo, tinkamos priemonės sistemos komponentų testavimui;
2. Prieš realizuojant testus buvo išsiaiškinta, kaip reikia tinkamai aprašyti komponentų vienetų testus bei kaip juos reikia tinkamai aprašyti programos kodu;
3. Komponentų testais buvo padengti visi API metodai;
4. Kodo padengimas po testų sukūrimo rodo, kad komponentų vienetų testais buvo sėkmingai padengta 100% API programos kodo.