Kauno technologijos universitetas

Informatikos fakultetas

T120B162 Programų sistemų testavimas

Vienetų testai

Laboratorinis darbas Nr. 2

Evita Šriupšaitė
Lukas Andrijauskas
Rugilė Petraitytė
Ieva Žukauskaitė

Darbą priėmė:

Greta Rudžionienė
Eduardas Bareiša

Kaunas, 2024

Turinys

Įvadas

Šio laboratorinio darbo tikslas – ištestuoti kuriamos programos komponentus rašant vienetų testus.

Mūsų tikslas – pasirinkti tinkamas priemones, karkasus ir padengti mūsų programą iš serverio ir iš kliento pusės testais, kurie dengtų bent 80% mūsų turimos programos.

1. Vienetų testai

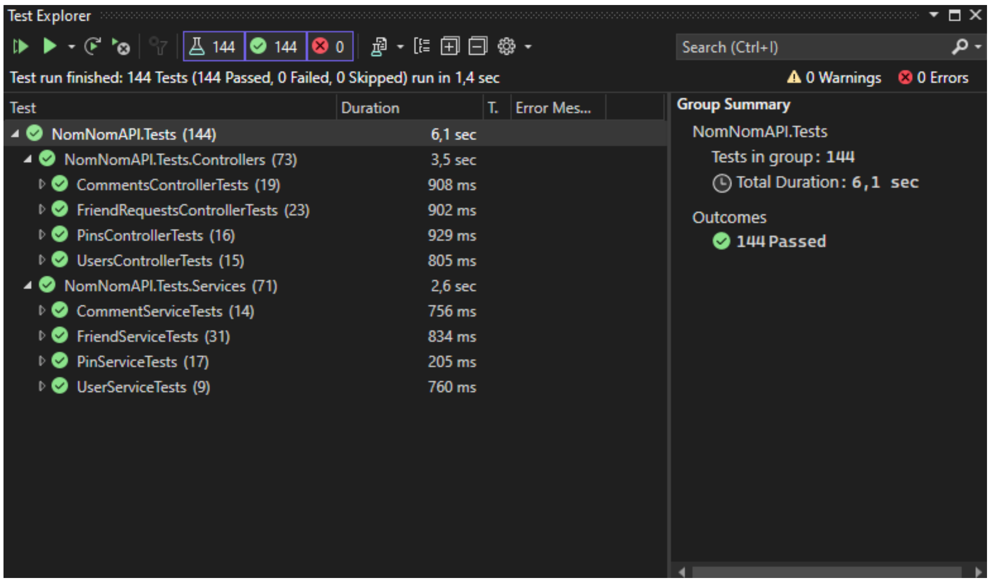
1.1. Serverio testavimas

Kadangi kūrėme ASP.NET Web API, naudojome plačiai naudojamą C# programų testavimo įrankį xUnit.net. Šis testavimo karkasas yra atvirojo kodo, tad turi didelę vartotojų bendruomenę, dėl ko galima tikėtis rasti daug reikiamos informacijos internete.

Mūsų API susideda iš 4 servisų: vartotojų servisas (atsakingas už naujų vartotojų registravimą, esamų autorizavimą), draugų servisas (atsakingas už tarpvartotojiškus ryšius: draugų užklausų siuntimą, atšaukimą, (ne)priėmimą, draugų peržiūrą ir šalinimą), įrašų servisas (atsakingas už vartotojų kuriamo turinio – žemėlapio smeigtukų, su jais susijusių restoranų atsiliepimų – kūrimą, modifikavimą, peržiūrą, šalinimą) bei komentarų servisas (atsakingas už įrašų komentarų kūrimą, modifikavimą, peržiūrą bei šalinimą). Kiekvienas iš šių servisų turi po kontrolerį. Tad ištestavome šių kontrolerių bei servisų klases. Kontrolerių testuose tikrinome, ar kontrolerių metodai teisingai tikrina duomenis, grąžina korektiškus atsakus į užklausas, o servisų testuose tikrinome, ar validacija veikia verslo logikoje, ar teisingai manipuliuojama duomenimis.

Kadangi iš viso gavosi 144 testai, ateinančiuose skyreliuose pateiksime tik dalį jų – po vieną klasę iš kontrolerių ir servisų testų.

Štai „Test Explorer" vaizdas įgyvendinus visus testus:



Test run finished: 144 Tests (144 Passed, 0 Failed, 0 Skipped) run in 1,4 sec
0 Warnings 0 Errors

Test	Duration	T.	Error Mes...
NomNomAPI.Tests (144)	6,1 sec		
NomNomAPI.Tests.Controllers (73)	3,5 sec		
CommentsControllerTests (19)	908 ms		
FriendRequestsControllerTests (23)	902 ms		
PinsControllerTests (16)	929 ms		
UsersControllerTests (15)	805 ms		
NomNomAPI.Tests.Services (71)	2,6 sec		
CommentServiceTests (14)	756 ms		
FriendServiceTests (31)	834 ms		
PinServiceTests (17)	205 ms		
UserServiceTests (9)	760 ms		

Group Summary
NomNomAPI.Tests
Tests in group: 144
Total Duration: 6,1 sec
Outcomes
144 Passed

1.1.1. PinServiceTests

Pateikiame Pin serviso testų klasę PinServiceTests:

```
using ErrorOr;
using FakeItEasy;
using NomNomAPI.Models;
using NomNomAPI.ServiceErrors;
using NomNomAPI.Services.Pins;
using NomNomAPI.Services.Users;

namespace NomNomAPI.Tests.Services
{
    public class PinServiceTests
    {
        private readonly IUserService _mockedUserService;
        private readonly IPinService _mockedPinService;

        public PinServiceTests()
        {
            _mockedUserService = A.Fake<IUserService>();
            _mockedPinService = A.Fake<IPinService>();
        }

        [Fact]
        public async void PinServiceTests_CreatePin_ReturnsOk()
        {
            //Arrange
            var fakeUser = A.Fake<User>();
            var fakePin = A.Fake<Pin>();

            //Act
            var createPinResult = await _mockedPinService.CreatePin(fakePin);

            //Assert
            Assert.False(createPinResult.IsError);
        }

        [Fact]
        public async void PinServiceTests_GetPin_ReturnsOk()
        {
            //Arrange
            var pin = A.Fake<Pin>();
            pin.Id = Guid.NewGuid();
            A.CallTo(() => _mockedPinService.GetPin(pin.Id)).Returns(pin);

            //Act
            var result = await _mockedPinService.GetPin(pin.Id);

            //Assert
            Assert.False(result.IsError);
        }

        [Fact]
        public async void PinServiceTests_GetPin_ReturnsPinNotFound()
        {
            //Arrange
            Guid badId = Guid.NewGuid();
            A.CallTo(()                                                    =>
_mockedPinService.GetPin(badId)).Returns(Errors.PinErrors.PinNotFound);

            //Act
            var result = await _mockedPinService.GetPin(badId);

            //Assert
            Assert.Contains(Errors.PinErrors.PinNotFound, result.Errors);
        }

        [Fact]
        public async void PinServiceTests_GetUserPins_ReturnsOk()
        {
            //Arrange
            var user = A.Fake<User>();
            const int NUM_PINS = 10;
            var pins = A.CollectionOfFake<Pin>(NUM_PINS);
            A.CallTo(() => _mockedPinService.GetUserPins(user)).Returns(pins);

            //Act
            var result = await _mockedPinService.GetUserPins(user);

            //Assert
            Assert.True(result.Count == NUM_PINS);
        }

        [Fact]
        public async void PinsControllerTests_GetUserPosts_ReturnsOk()
        {
            //Arrange
            var user = A.Fake<User>();
            const int NUM_POSTS = 10;
            var posts = A.CollectionOfFake<Post>(NUM_POSTS);
            A.CallTo(() => _mockedPinService.GetUserPosts(user)).Returns(posts);

            //Act
            var result = await _mockedPinService.GetUserPosts(user);

            //Assert
            Assert.True(result.Count == NUM_POSTS);
        }
```

```
[Fact]
public async void PinServiceTests_GetAllPins_ReturnsOk()
{
    //Arrange
    var user = A.Fake<User>();
    const int NUM_PINS = 10;
    var pins = A.CollectionOfFake<Pin>(NUM_PINS);
    A.CallTo(() => _mockedPinService.GetAllPins(user)).Returns(pins);

    //Act
    var result = await _mockedPinService.GetAllPins(user);

    //Assert
    Assert.True(result.Count == NUM_PINS);
}

[Fact]
public async void PinServiceTests_GetAllPosts_ReturnsOk()
{
    //Arrange
    var user = A.Fake<User>();
    const int NUM_POSTS = 10;
    var posts = A.CollectionOfFake<Post>(NUM_POSTS);
    A.CallTo(() => _mockedPinService.GetAllPosts(user)).Returns(posts);

    //Act
    var result = await _mockedPinService.GetAllPosts(user);

    //Assert
    Assert.True(result.Count == NUM_POSTS);
}

[Fact]
public async void PinServiceTests_DeletePin_ReturnsOk()
{
    //Arrange
    var pin = A.Fake<Pin>();
    A.CallTo(() => _mockedPinService.DeletePin(pin.Id)).Returns(Result.Deleted);

    //Act
    var result = await _mockedPinService.DeletePin(pin.Id);

    //Assert
    Assert.True(result.Value == Result.Deleted);
```

```
        }

        [Fact]
        public async void PinServiceTests_AddPost_ReturnsOk()
        {
            //Arrange
            var post = A.Fake<Post>();
            A.CallTo(() => _mockedPinService.AddPost(post)).Returns(Result.Created);

            //Act
            var result = await _mockedPinService.AddPost(post);

            //Assert
            Assert.False(result.IsError);
        }

        [Fact]
        public async void PinServiceTests_AddPost_ReturnsPostAlreadyExists()
        {
            //Arrange
            var post = A.Fake<Post>();
            A.CallTo(()                                                          =>
_mockedPinService.AddPost(post)).Returns(Errors.PinErrors.PostErrors.PostAlreadyExists);

            //Act
            var result = await _mockedPinService.AddPost(post);

            //Assert
            Assert.Contains(Errors.PinErrors.PostErrors.PostAlreadyExists, result.Errors);
        }

        [Fact]
        public async void PinServiceTests_GetPost_ReturnsOk()
        {
            //Arrange
            Guid someId = Guid.NewGuid();
            Post post = A.Fake<Post>();
            A.CallTo(() => _mockedPinService.GetPost(someId)).Returns(post);

            //Act
            var result = await _mockedPinService.GetPost(someId);

            //Assert
            Assert.False(result.IsError);
        }
        [Fact]
        public async void PinServiceTests_GetPost_ReturnsPostDoesntExist()
        {
            //Arrange
            Guid someId = Guid.Empty;
            Post post = A.Fake<Post>();
            A.CallTo(()                                                              =>
_mockedPinService.GetPost(someId)).Returns(Errors.PinErrors.PostErrors.PostDoesntExist);

            //Act
            var result = await _mockedPinService.GetPost(someId);

            //Assert
            Assert.Contains(Errors.PinErrors.PostErrors.PostDoesntExist, result.Errors);
        }

        [Fact]
        public async void PinServiceTests_GetParentPin_ReturnsOk()
        {
            //Arrange
            Guid someId = Guid.NewGuid();
            Pin pin = A.Fake<Pin>();
            A.CallTo(() => _mockedPinService.GetParentPin(someId)).Returns(pin);

            //Act
            var result = await _mockedPinService.GetParentPin(someId);

            //Assert
            Assert.False(result.IsError);
        }

        [Fact]
        public async void PinServiceTests_GetParentPin_ReturnsPinNotFound()
        {
            //Arrange
            Guid someId = Guid.NewGuid();
            Pin pin = A.Fake<Pin>();
            A.CallTo(()                                                              =>
_mockedPinService.GetParentPin(someId)).Returns(Errors.PinErrors.PinNotFound);

            //Act
            var result = await _mockedPinService.GetParentPin(someId);

            //Assert
            Assert.Contains(Errors.PinErrors.PinNotFound, result.Errors);
        }

        [Fact]
        public async void PinServiceTests_GetParentPin_ReturnsPostDoesntExist()
        {
            //Arrange
            Guid someId = Guid.NewGuid();
            Pin pin = A.Fake<Pin>();
            A.CallTo(()                                                    =>
_mockedPinService.GetParentPin(someId)).Returns(Errors.PinErrors.PostErrors.PostDoesntExist
);

            //Act
            var result = await _mockedPinService.GetParentPin(someId);

            //Assert
            Assert.Contains(Errors.PinErrors.PostErrors.PostDoesntExist, result.Errors);
        }

        [Fact]
        public async void PinServiceTests_DeletePost_ReturnsOk()
        {
            //Arrange
            Guid id = Guid.NewGuid();
            A.CallTo(() => _mockedPinService.DeletePost(id)).Returns(Result.Deleted);

            //Act
            var result = await _mockedPinService.DeletePost(id);

            //Assert
            Assert.False(result.IsError);
        }

        [Fact]
        public async void PinServiceTests_DeletePost_ReturnsPostDoesntExist()
        {
            //Arrange
            Guid id = Guid.NewGuid();
            A.CallTo(()                                                    =>
_mockedPinService.DeletePost(id)).Returns(Errors.PinErrors.PostErrors.PostDoesntExist);

            //Act
            var result = await _mockedPinService.DeletePost(id);
            //Assert
            Assert.Contains(Errors.PinErrors.PostErrors.PostDoesntExist, result.Errors);
        }
    }
}
```

1.1.2. PinsControllerTests

Pateikiame Pins kontrolerio testų klasę PinsControllerTests:

```
using FakeItEasy;
using NomNomAPI.Services.Users;
using NomNomAPI.Services.Pins;
using NomNomAPI.Controllers;
using NomNom.Contracts.Pin;
using Microsoft.AspNetCore.Mvc;
using NomNomAPI.Models;
using Microsoft.AspNetCore.Http;
using System.Security.Claims;
using Microsoft.EntityFrameworkCore;
using NomNom.Contracts.User;
using NomNomAPI.DataAccess;
using NomNomAPI.Services.Friends;
using NomNom.Contracts.FriendRequest;
using NomNomAPI.ServiceErrors;

namespace NomNomAPI.Tests.Controllers
{
    public class PinsControllerTests
    {
        private readonly IUserService _mockedUserService;
        private readonly IPinService _mockedPinService;

        public PinsControllerTests()
        {
            _mockedUserService = A.Fake<IUserService>();
            _mockedPinService = A.Fake<IPinService>();
        }

        [Fact]
        public async void PinsControllerTests_CreatePin_ReturnsOk()
        {
            //Arrange
            var fakeUser = A.Fake<User>();
        var pinController = new PinsController(_mockedPinService, _mockedUserService);

        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, fakeUser.Id.ToString()),
        }));

        pinController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext { User = claimsPrincipal }
        };
        var createPinRequest = new CreatePinRequest(40.5, 40.7844);

        //Act
        var createPinResult = await pinController.CreatePin(createPinRequest);

        //Assert
        Assert.Equal(typeof(OkObjectResult), createPinResult.GetType());
    }

    [Fact]
    public async void PinsControllerTests_GetAllUserPins_ReturnsOk()
    {
        //Arrange
        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB1")
            .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var pinService = new PinService(context, friendService);
        var pinController = new PinsController(pinService, userService);



        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);

        var sender = await userService.GetUser("NBunke");

        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
            {
                new Claim(ClaimTypes.NameIdentifier, sender.Value.Id.ToString()),
            }));

            pinController.ControllerContext = new ControllerContext
            {
                HttpContext = new DefaultHttpContext { User = claimsPrincipal }
            };

            await pinController.CreatePin(new CreatePinRequest(30.4, 30.8));
            await pinController.CreatePin(new CreatePinRequest(30.1, 30.2));
            await pinController.CreatePin(new CreatePinRequest(30.4, 30.5));

            //Act
            var getAllUserPinsResult = await pinController.GetAllUserPins();
            var getAllUserPinsObjectResult = (getAllUserPinsResult as OkObjectResult);
            var getAllUserCount = ((List<PinResponse>)getAllUserPinsObjectResult.Value).Count;

            //Assert
            Assert.Equal(3, getAllUserCount);
        }

        [Fact]
        public async void PinsControllerTests_GetAllUserPosts_ReturnsOk()
        {
            // Arrange

            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase("pinsControllerTestDB2")
                .Options;

            var context = new AppDbContext(options);
            var userService = new UserService(context);
            var friendRequestRepo = new FriendRequestRepository(context);
            var friendService = new FriendService(context, friendRequestRepo);
            var usersController = new UsersController(userService);
            var friendsController = new FriendRequestsController(friendService, userService);
            var pinService = new PinService(context, friendService);
            var pinsController = new PinsController(pinService, userService);

            var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

            await usersController.SignUp(request1);
            var sender = await userService.GetUser("NBunke");

            var senderId = sender.Value.Id;
            var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
            {
                new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
            }));

            friendsController.ControllerContext = new ControllerContext
            {
                HttpContext = new DefaultHttpContext
                { User = claimsPrincipal }
            };

            pinsController.ControllerContext = new ControllerContext
            {
                HttpContext = new DefaultHttpContext
                { User = claimsPrincipal }
            };



            var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

            var createPinResultOkObject = createPinResult1 as OkObjectResult;
            var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

            await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id, "Pizza
    Shack",
                "Very nice place I recommend", DateTime.Now));

            var createPinResult2 = await pinsController.CreatePin(new CreatePinRequest(40.3, 40.2));

            var createPinResultOkObject2 = createPinResult2 as OkObjectResult;
            var createPinResultObject2 = (PinResponse)createPinResultOkObject2.Value;

            await pinsController.AddPost(new AddPostRequest(createPinResultObject2.Id, "Pizza
    Shack 5",
                "Better than Pizza Shack", DateTime.Now));

            await pinsController.CreatePin(new CreatePinRequest(40.2, 40.9));
            await pinsController.CreatePin(new CreatePinRequest(40.1, 40.7));



            // Act
            var getAllUserPostsResult = await pinsController.GetAllUserPosts();

            var getAllUserPostsObjectResult = (getAllUserPostsResult as OkObjectResult);
            var getAllUserPostsCount =
    ((List<PostResponse>)getAllUserPostsObjectResult.Value).Count;


            // Assert
            Assert.Equal(2, getAllUserPostsCount);
        }

        [Fact]
        public async void PinsControllerTests_GetAllPins_ReturnsOk()
        {
            // Arrange

            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase("pinsControllerTestDB3")
                .Options;

            var context = new AppDbContext(options);
            var userService = new UserService(context);
            var friendRequestRepo = new FriendRequestRepository(context);
            var friendService = new FriendService(context, friendRequestRepo);
            var usersController = new UsersController(userService);
            var friendsController = new FriendRequestsController(friendService, userService);
            var pinService = new PinService(context, friendService);
            var pinsController = new PinsController(pinService, userService);

            var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");
            var request2 = new SignUpRequest("Zvonkus", "realzvonkus@gmail.com",
    "Katazina4Life");

            await usersController.SignUp(request1);
            await usersController.SignUp(request2);

            var sender = await userService.GetUser("NBunke");

            var senderId = sender.Value.Id;
            var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
            {
                new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
            }));
```

```
friendsController.ControllerContext = new ControllerContext
{
   HttpContext = new DefaultHttpContext
   { User = claimsPrincipal }
};

pinsController.ControllerContext = new ControllerContext
{
   HttpContext = new DefaultHttpContext
   { User = claimsPrincipal }
};

var friendrequest = new FriendRequestRequest("Zvonkus");
var result = await friendsController.SendFriendRequest(friendrequest);

await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));
await pinsController.CreatePin(new CreatePinRequest(40.3, 40.2));
await pinsController.CreatePin(new CreatePinRequest(40.2, 40.9));
await pinsController.CreatePin(new CreatePinRequest(40.1, 40.7));

var friendRequestOkObject = (result as OkObjectResult);
var friendRequestId = ((FriendRequestResponse)friendRequestOkObject.Value).Id;


var sender2 = await userService.GetUser("Zvonkus");
var senderId2 = sender2.Value.Id;

claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
{
   new Claim(ClaimTypes.NameIdentifier, senderId2.ToString()),
}));

friendsController.ControllerContext = new ControllerContext
{
   HttpContext = new DefaultHttpContext
   { User = claimsPrincipal }
};

pinsController.ControllerContext = new ControllerContext
{
   HttpContext = new DefaultHttpContext
   { User = claimsPrincipal }
};

var result2 = await friendsController.AcceptFriendRequest(friendRequestId);
```

```
        await pinsController.CreatePin(new CreatePinRequest(33.5, 33.7));
        await pinsController.CreatePin(new CreatePinRequest(33.3, 33.2));
        await pinsController.CreatePin(new CreatePinRequest(33.2, 33.9));
        await pinsController.CreatePin(new CreatePinRequest(33.1, 33.7));

        // Act

        var getAllPinsResult = await pinsController.GetAllPins();

        var getAllPinsObjectResult = (getAllPinsResult as OkObjectResult);
        var getAllPinsCount = ((List<PinResponse>)getAllPinsObjectResult.Value).Count;



        // Assert
        Assert.Equal(8, getAllPinsCount);
    }

    [Fact]
    public async void PinsControllerTests_GetAllPosts_ReturnsOk()
    {
        // Arrange

        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB4")
            .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var friendsController = new FriendRequestsController(friendService, userService);
        var pinService = new PinService(context, friendService);
        var pinsController = new PinsController(pinService, userService);

        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");
        var request2 = new SignUpRequest("Zvonkus", "realzvonkus@gmail.com",
"Katazina4Life");

        await usersController.SignUp(request1);
        await usersController.SignUp(request2);

        var sender = await userService.GetUser("NBunke");
        var senderId = sender.Value.Id;
        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
           new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
           HttpContext = new DefaultHttpContext
           { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
           HttpContext = new DefaultHttpContext
           { User = claimsPrincipal }
        };

        var friendrequest = new FriendRequestRequest("Zvonkus");
        var result = await friendsController.SendFriendRequest(friendrequest);

        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id, "Pizza
Shack",
           "Very nice place I recommend", DateTime.Now));

        var createPinResult2 = await pinsController.CreatePin(new CreatePinRequest(40.3, 40.2));

        var createPinResultOkObject2 = createPinResult2 as OkObjectResult;
        var createPinResultObject2 = (PinResponse)createPinResultOkObject2.Value;

        await pinsController.AddPost(new AddPostRequest(createPinResultObject2.Id, "Pizza
Shack 5",
           "Better than Pizza Shack", DateTime.Now));

        await pinsController.CreatePin(new CreatePinRequest(40.2, 40.9));
        await pinsController.CreatePin(new CreatePinRequest(40.1, 40.7));

        var friendRequestOkObject = (result as OkObjectResult);
        var friendRequestId = ((FriendRequestResponse)friendRequestOkObject.Value).Id;
        var sender2 = await userService.GetUser("Zvonkus");
        var senderId2 = sender2.Value.Id;

        claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, senderId2.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };

        var result2 = await friendsController.AcceptFriendRequest(friendRequestId);

        // Act

        var getAllPostsResult = await pinsController.GetAllPosts();

        var getAllPostsObjectResult = (getAllPostsResult as OkObjectResult);
        var getAllPostsCount = ((List<PostResponse>)getAllPostsObjectResult.Value).Count;


        // Assert
        Assert.Equal(2, getAllPostsCount);
    }

    [Fact]
    public async void PinsControllerTests_GetPin_ReturnsOk()
    {
        //Arrange
        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB10")
            .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var pinService = new PinService(context, friendService);
        var pinController = new PinsController(pinService, userService);


        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);

        var sender = await userService.GetUser("NBunke");

        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, sender.Value.Id.ToString()),
        }));

        pinController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext { User = claimsPrincipal }
        };

        var createPinResult = await pinController.CreatePin(new CreatePinRequest(30.4, 30.8));
        var createPinObjectResult = (createPinResult as OkObjectResult);
        var createPinObject = (PinResponse)(createPinObjectResult.Value);

        //Act
        var getPinResult = await pinController.GetPin(createPinObject.Id);


        //Assert
        Assert.Equal(typeof(OkObjectResult), getPinResult.GetType());
    }

    [Fact]
    public async void PinsControllerTests_DeletePin_ReturnsOk()
    {
        //Arrange
        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB11")
            .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var pinService = new PinService(context, friendService);
        var pinController = new PinsController(pinService, userService);


        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);

        var sender = await userService.GetUser("NBunke");

        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, sender.Value.Id.ToString()),
        }));

        pinController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext { User = claimsPrincipal }
        };

        var createPinResult = await pinController.CreatePin(new CreatePinRequest(30.4, 30.8));
        var createPinObjectResult = (createPinResult as OkObjectResult);
        var createPinObject = (PinResponse)(createPinObjectResult.Value);

        //Act
        var deletePinResult = await pinController.DeletePin(createPinObject.Id);


        //Assert
        Assert.Equal(typeof(OkResult), deletePinResult.GetType());
    }


    [Fact]
    public async void PinsControllerTests_AddPost_ReturnsOk()
    {
        // Arrange

        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB20")
            .Options;
        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var friendsController = new FriendRequestsController(friendService, userService);
        var pinService = new PinService(context, friendService);
        var pinsController = new PinsController(pinService, userService);

        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);


        var sender = await userService.GetUser("NBunke");

        var senderId = sender.Value.Id;
        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };


        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        // Act

        var addPostResult = await pinsController.AddPost(new
AddPostRequest(createPinResultObject.Id,
            "Pizza Shack", "Cool pizza shack place very nice", DateTime.Now));
        // Assert
        Assert.Equal(typeof(OkObjectResult), addPostResult.GetType());
    }

    [Fact]
    public async void PinsControllerTests_AddPost_ReturnsIncorrectRestaurantNameLength()
    {
        // Arrange

        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB289")
            .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var friendsController = new FriendRequestsController(friendService, userService);
        var pinService = new PinService(context, friendService);
        var pinsController = new PinsController(pinService, userService);

        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);


        var sender = await userService.GetUser("NBunke");

        var senderId = sender.Value.Id;
        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };


        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        // Act

        var addPostResult = await pinsController.AddPost(new
AddPostRequest(createPinResultObject.Id,
            "", "Cool pizza shack place very nice", DateTime.Now));

        //Assert

        var objectResult = addPostResult as ObjectResult;
        Assert.NotNull(objectResult);
        var validationProblemDetails = objectResult.Value as ValidationProblemDetails;

        Assert.NotNull(validationProblemDetails);

Assert.True(validationProblemDetails.Errors.ContainsKey(Errors.PinErrors.PostErrors.Incorrect
RestaurantNameLength.Code));
    }

    [Fact]
    public async void PinsControllerTests_AddPost_ReturnsIncorrectReviewLength()
    {
        // Arrange

        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB2978")
            .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var friendsController = new FriendRequestsController(friendService, userService);
        var pinService = new PinService(context, friendService);
        var pinsController = new PinsController(pinService, userService);
        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);


        var sender = await userService.GetUser("NBunke");

        var senderId = sender.Value.Id;
        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
           new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
           HttpContext = new DefaultHttpContext
           { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
           HttpContext = new DefaultHttpContext
           { User = claimsPrincipal }
        };


        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        // Act

        var addPostResult = await pinsController.AddPost(new
AddPostRequest(createPinResultObject.Id,
           "Pizza Shack 5555", new string('*', 5000), DateTime.Now));

        //Assert

        var objectResult = addPostResult as ObjectResult;
        Assert.NotNull(objectResult);
        var validationProblemDetails = objectResult.Value as ValidationProblemDetails;

        Assert.NotNull(validationProblemDetails);
Assert.True(validationProblemDetails.Errors.ContainsKey(Errors.PinErrors.PostErrors.Incorrect
ReviewLength.Code));
        }

        [Fact]
        public async void PinsControllerTests_AddPost_ReturnsPostAlreadyExists()
        {
        // Arrange

        var options = new DbContextOptionsBuilder<AppDbContext>()
           .UseInMemoryDatabase("pinsControllerTestDB200")
           .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var friendsController = new FriendRequestsController(friendService, userService);
        var pinService = new PinService(context, friendService);
        var pinsController = new PinsController(pinService, userService);

        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);


        var sender = await userService.GetUser("NBunke");

        var senderId = sender.Value.Id;
        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
           new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
           HttpContext = new DefaultHttpContext
           { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
           HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };


        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id,
          "Pizza Shack 5555", "Cool pizza shack place very nice", DateTime.Now));

        // Act

        var addPostResult = await pinsController.AddPost(new
AddPostRequest(createPinResultObject.Id,
            "Pizza Shack 5555", "Cool pizza shack place very nice", DateTime.Now));

        //Assert

        var objectResult = addPostResult as ObjectResult;
        Assert.NotNull(objectResult);
        var validationProblemDetails = objectResult.Value as ValidationProblemDetails;

        Assert.NotNull(validationProblemDetails);

Assert.True(validationProblemDetails.Errors.ContainsKey(Errors.PinErrors.PostErrors.PostAlrea
dyExists.Code));
    }

    [Fact]
    public async void PinsControllerTests_GetPost_ReturnsOk()
    {
        // Arrange

        var options = new DbContextOptionsBuilder<AppDbContext>()
            .UseInMemoryDatabase("pinsControllerTestDB21")
            .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var friendsController = new FriendRequestsController(friendService, userService);
        var pinService = new PinService(context, friendService);
        var pinsController = new PinsController(pinService, userService);

        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

        await usersController.SignUp(request1);


        var sender = await userService.GetUser("NBunke");

        var senderId = sender.Value.Id;
        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };


        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id,
           "Pizza Shack", "Cool pizza shack place very nice", DateTime.Now));

        // Act

        var getPostResult = await pinsController.GetPost(createPinResultObject.Id);



        // Assert
        Assert.Equal(typeof(OkObjectResult), getPostResult.GetType());
    }
```

```
[Fact]
public async void PinsControllerTests_UpdatePost_ReturnsOk()
{
    // Arrange

    var options = new DbContextOptionsBuilder<AppDbContext>()
        .UseInMemoryDatabase("pinsControllerTestDB26")
        .Options;

    var context = new AppDbContext(options);
    var userService = new UserService(context);
    var friendRequestRepo = new FriendRequestRepository(context);
    var friendService = new FriendService(context, friendRequestRepo);
    var usersController = new UsersController(userService);
    var friendsController = new FriendRequestsController(friendService, userService);
    var pinService = new PinService(context, friendService);
    var pinsController = new PinsController(pinService, userService);

    var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

    await usersController.SignUp(request1);


    var sender = await userService.GetUser("NBunke");

    var senderId = sender.Value.Id;
    var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
    {
        new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
    }));

    friendsController.ControllerContext = new ControllerContext
    {
        HttpContext = new DefaultHttpContext
        { User = claimsPrincipal }
    };

    pinsController.ControllerContext = new ControllerContext
    {
        HttpContext = new DefaultHttpContext
        { User = claimsPrincipal }
    };
```

```
        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id,
          "Pizza Shack", "Cool pizza shack place very nice", DateTime.Now));

        var getPostResult = await pinsController.GetPost(createPinResultObject.Id);
        var getPostResultOkObject = getPostResult as OkObjectResult;
        var getPostResultObject = (PostResponse)(getPostResultOkObject.Value);

        // Act

        var updatePostResult = await pinsController.UpdatePost(new
UpdatePostRequest(getPostResultObject.PostId,
          getPostResultObject.RestaurantName, "No this place is actually bad", DateTime.Now));



        // Assert
        Assert.Equal(typeof(OkObjectResult), updatePostResult.GetType());
    }

    [Fact]
    public async void PinsControllerTests_UpdatePost_ReturnsPostDoesntExist()
    {
        // Arrange

        var options = new DbContextOptionsBuilder<AppDbContext>()
          .UseInMemoryDatabase("pinsControllerTestDB260")
          .Options;

        var context = new AppDbContext(options);
        var userService = new UserService(context);
        var friendRequestRepo = new FriendRequestRepository(context);
        var friendService = new FriendService(context, friendRequestRepo);
        var usersController = new UsersController(userService);
        var friendsController = new FriendRequestsController(friendService, userService);
        var pinService = new PinService(context, friendService);
        var pinsController = new PinsController(pinService, userService);

        var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");
        await usersController.SignUp(request1);


        var sender = await userService.GetUser("NBunke");

        var senderId = sender.Value.Id;
        var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
        }));

        friendsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };

        pinsController.ControllerContext = new ControllerContext
        {
            HttpContext = new DefaultHttpContext
            { User = claimsPrincipal }
        };


        var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

        var createPinResultOkObject = createPinResult1 as OkObjectResult;
        var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

        await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id,
            "Pizza Shack", "Cool pizza shack place very nice", DateTime.Now));

        var getPostResult = await pinsController.GetPost(createPinResultObject.Id);
        var getPostResultOkObject = getPostResult as OkObjectResult;
        var getPostResultObject = (PostResponse)(getPostResultOkObject.Value);

        // Act

        var updatePostResult = await pinsController.UpdatePost(new
UpdatePostRequest(Guid.Empty,
            getPostResultObject.RestaurantName, "No this place is actually bad", DateTime.Now));



        // Assert
```

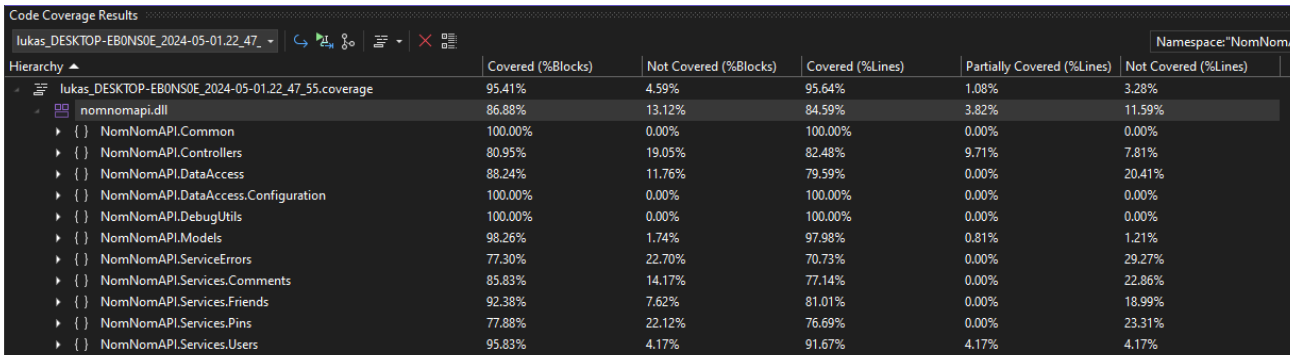
```
            var objectResult = updatePostResult as ObjectResult;
            Assert.NotNull(objectResult);
            var validationProblemDetails = objectResult.Value as ValidationProblemDetails;

            Assert.NotNull(validationProblemDetails);

Assert.True(validationProblemDetails.Errors.ContainsKey(Errors.PinErrors.PostErrors.PostDoes
ntExist.Code));
        }

        [Fact]
        public async void PinsControllerTests_DeletePost_ReturnsOk()
        {
            // Arrange

            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase("pinsControllerTestDB25")
                .Options;

            var context = new AppDbContext(options);
            var userService = new UserService(context);
            var friendRequestRepo = new FriendRequestRepository(context);
            var friendService = new FriendService(context, friendRequestRepo);
            var usersController = new UsersController(userService);
            var friendsController = new FriendRequestsController(friendService, userService);
            var pinService = new PinService(context, friendService);
            var pinsController = new PinsController(pinService, userService);

            var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

            await usersController.SignUp(request1);



            var sender = await userService.GetUser("NBunke");

            var senderId = sender.Value.Id;
            var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
            {
                new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
            }));

            friendsController.ControllerContext = new ControllerContext
            {
                HttpContext = new DefaultHttpContext
                { User = claimsPrincipal }
            };

            pinsController.ControllerContext = new ControllerContext
            {
                HttpContext = new DefaultHttpContext
                { User = claimsPrincipal }
            };


            var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

            var createPinResultOkObject = createPinResult1 as OkObjectResult;
            var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

            await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id,
                "Pizza Shack", "Cool pizza shack place very nice", DateTime.Now));

            var getPostResult = await pinsController.GetPost(createPinResultObject.Id);
            var getPostResultOkObject = getPostResult as OkObjectResult;
            var getPostResultObject = (PostResponse)(getPostResultOkObject.Value);

            // Act

            var deletePostResult = await pinsController.DeletePost(new
DeletePostRequest(getPostResultObject.PostId));


            // Assert
            Assert.Equal(typeof(OkResult), deletePostResult.GetType());
        }

        [Fact]
        public async void PinsControllerTests_DeletePost_ReturnsPostDoesntExist()
        {
            // Arrange

            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase("pinsControllerTestDB250")
                .Options;

            var context = new AppDbContext(options);
            var userService = new UserService(context);
            var friendRequestRepo = new FriendRequestRepository(context);
            var friendService = new FriendService(context, friendRequestRepo);
var usersController = new UsersController(userService);
var friendsController = new FriendRequestsController(friendService, userService);
var pinService = new PinService(context, friendService);
var pinsController = new PinsController(pinService, userService);

var request1 = new SignUpRequest("NBunke", "bunke@hotmail.com", "Kosmosas2024");

await usersController.SignUp(request1);


var sender = await userService.GetUser("NBunke");

var senderId = sender.Value.Id;
var claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
{
   new Claim(ClaimTypes.NameIdentifier, senderId.ToString()),
}));

friendsController.ControllerContext = new ControllerContext
{
   HttpContext = new DefaultHttpContext
   { User = claimsPrincipal }
};

pinsController.ControllerContext = new ControllerContext
{
   HttpContext = new DefaultHttpContext
   { User = claimsPrincipal }
};


var createPinResult1 = await pinsController.CreatePin(new CreatePinRequest(40.5, 40.7));

var createPinResultOkObject = createPinResult1 as OkObjectResult;
var createPinResultObject = (PinResponse)createPinResultOkObject.Value;

await pinsController.AddPost(new AddPostRequest(createPinResultObject.Id,
   "Pizza Shack", "Cool pizza shack place very nice", DateTime.Now));

var getPostResult = await pinsController.GetPost(createPinResultObject.Id);
var getPostResultOkObject = getPostResult as OkObjectResult;
var getPostResultObject = (PostResponse)(getPostResultOkObject.Value);

await pinsController.DeletePost(new DeletePostRequest(getPostResultObject.PostId));
```

```
            // Act

            var deletePostResult = await pinsController.DeletePost(new
DeletePostRequest(getPostResultObject.PostId));

            // Assert

            var objectResult = deletePostResult as ObjectResult;
            Assert.NotNull(objectResult);
            var validationProblemDetails = objectResult.Value as ValidationProblemDetails;

            Assert.NotNull(validationProblemDetails);

Assert.True(validationProblemDetails.Errors.ContainsKey(Errors.PinErrors.PostErrors.PostDoes
ntExist.Code));
        }
    }



}
```

1.1.3. Padengimo ataskaita

Padengimą skaičiavome su Coverlet ir Microsoft.CodeCoverage įrankiais, o ataskaitos vizualizacijai naudojome ReportGenerator įrankį.

Microsoft.CodeCoverage sugeneruota ataskaita:

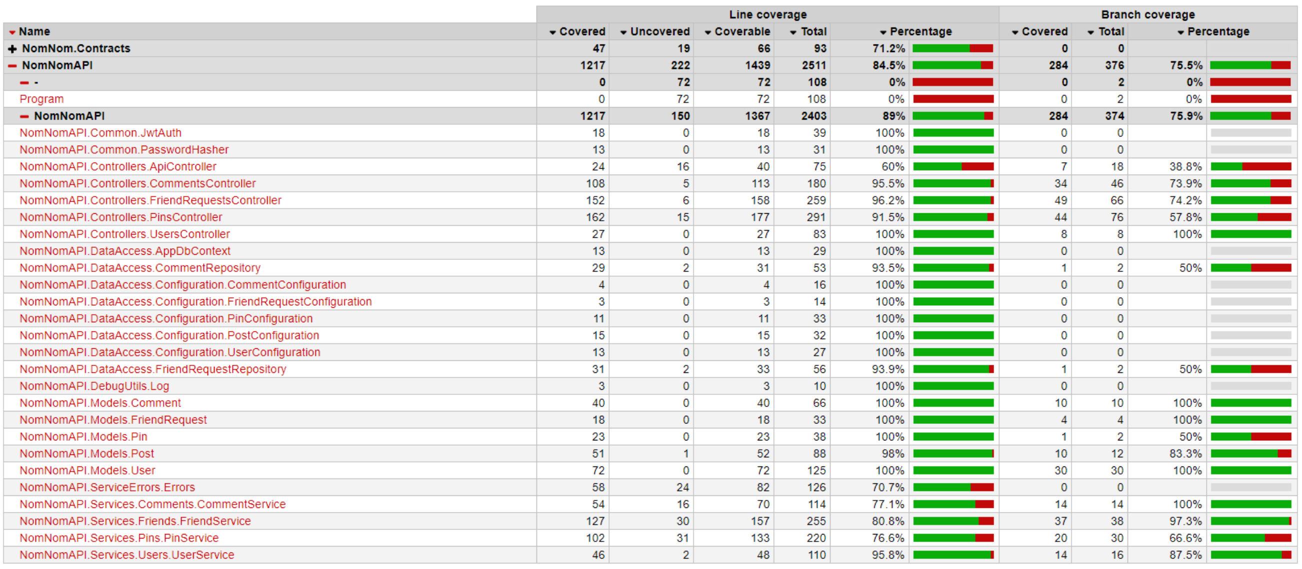

Code Coverage Results

lukas_DESKTOP-EB0NS0E_2024-05-01.22_47_

Namespace:"NomNom

Hierarchy	Covered (%Blocks)	Not Covered (%Blocks)	Covered (%Lines)	Partially Covered (%Lines)	Not Covered (%Lines)
lukas_DESKTOP-EB0NS0E_2024-05-01.22_47_55.coverage	95.41%	4.59%	95.64%	1.08%	3.28%
nomnomapi.dll	86.88%	13.12%	84.59%	3.82%	11.59%
{ } NomNomAPI.Common	100.00%	0.00%	100.00%	0.00%	0.00%
{ } NomNomAPI.Controllers	80.95%	19.05%	82.48%	9.71%	7.81%
{ } NomNomAPI.DataAccess	88.24%	11.76%	79.59%	0.00%	20.41%
{ } NomNomAPI.DataAccess.Configuration	100.00%	0.00%	100.00%	0.00%	0.00%
{ } NomNomAPI.DebugUtils	100.00%	0.00%	100.00%	0.00%	0.00%
{ } NomNomAPI.Models	98.26%	1.74%	97.98%	0.81%	1.21%
{ } NomNomAPI.ServiceErrors	77.30%	22.70%	70.73%	0.00%	29.27%
{ } NomNomAPI.Services.Comments	85.83%	14.17%	77.14%	0.00%	22.86%
{ } NomNomAPI.Services.Friends	92.38%	7.62%	81.01%	0.00%	18.99%
{ } NomNomAPI.Services.Pins	77.88%	22.12%	76.69%	0.00%	23.31%
{ } NomNomAPI.Services.Users	95.83%	4.17%	91.67%	4.17%	4.17%

ReportGenerator sugeneruota ataskaita (padengimas apskaičiuotas su Coverlet):



	Line coverage					Branch coverage		
Name	Covered	Uncovered	Coverable	Total	Percentage	Covered	Total	Percentage
+ NomNom.Contracts	**47**	**19**	**66**	**93**	**71.2%**	**0**	**0**	
− NomNomAPI	**1217**	**222**	**1439**	**2511**	**84.5%**	**284**	**376**	**75.5%**
− -	**0**	**72**	**72**	**108**	**0%**	**0**	**2**	**0%**
Program	0	72	72	108	0%	0	2	0%
− NomNomAPI	**1217**	**150**	**1367**	**2403**	**89%**	**284**	**374**	**75.9%**
NomNomAPI.Common.JwtAuth	18	0	18	39	100%	0	0	
NomNomAPI.Common.PasswordHasher	13	0	13	31	100%	0	0	
NomNomAPI.Controllers.ApiController	24	16	40	75	60%	7	18	38.8%
NomNomAPI.Controllers.CommentsController	108	5	113	180	95.5%	34	46	73.9%
NomNomAPI.Controllers.FriendRequestsController	152	6	158	259	96.2%	49	66	74.2%
NomNomAPI.Controllers.PinsController	162	15	177	291	91.5%	44	76	57.8%
NomNomAPI.Controllers.UsersController	27	0	27	83	100%	8	8	100%
NomNomAPI.DataAccess.AppDbContext	13	0	13	29	100%	0	0	
NomNomAPI.DataAccess.CommentRepository	29	2	31	53	93.5%	1	2	50%
NomNomAPI.DataAccess.Configuration.CommentConfiguration	4	0	4	16	100%	0	0	
NomNomAPI.DataAccess.Configuration.FriendRequestConfiguration	3	0	3	14	100%	0	0	
NomNomAPI.DataAccess.Configuration.PinConfiguration	11	0	11	33	100%	0	0	
NomNomAPI.DataAccess.Configuration.PostConfiguration	15	0	15	32	100%	0	0	
NomNomAPI.DataAccess.Configuration.UserConfiguration	13	0	13	27	100%	0	0	
NomNomAPI.DataAccess.FriendRequestRepository	31	2	33	56	93.9%	1	2	50%
NomNomAPI.DebugUtils.Log	3	0	3	10	100%	0	0	
NomNomAPI.Models.Comment	40	0	40	66	100%	10	10	100%
NomNomAPI.Models.FriendRequest	18	0	18	33	100%	4	4	100%
NomNomAPI.Models.Pin	23	0	23	38	100%	1	2	50%
NomNomAPI.Models.Post	51	1	52	88	98%	10	12	83.3%
NomNomAPI.Models.User	72	0	72	125	100%	30	30	100%
NomNomAPI.ServiceErrors.Errors	58	24	82	126	70.7%	0	0	
NomNomAPI.Services.Comments.CommentService	54	16	70	114	77.1%	14	14	100%
NomNomAPI.Services.Friends.FriendService	127	30	157	255	80.8%	37	38	97.3%
NomNomAPI.Services.Pins.PinService	102	31	133	220	76.6%	20	30	66.6%
NomNomAPI.Services.Users.UserService	46	2	48	110	95.8%	14	16	87.5%

1.2. Kliento testavimas

1.2.1. Pasiruošimas

Kliento pusę pasirinkome testuoti su „Jest“ karkasu, kadangi jis yra vienas iš populiariausių karkasų skirtų rašyti vienetų testus JavaScript kalbai, taigi, tai reiškia, kad bus galima internete daug lengviau rasti reikiamą informaciją. Be to, pačio „Jest“ dokumentacija yra aiški ir pakankamai plati. Dar vienas „Jest“ pliusas – jį lengva suinstaliuoti ir galima iškart naudoti, nereikia jokių papildomų įrankių.

Po instaliavimo reikėjo paredaguoti konfigūracijos failą – pagrindinis aspektas, kurį keitėme, yra tam tikrų failų, kurių patys nerašėme, išskyrimas, kad jie nebūtų įtraukiami į padengimo ataskaitą. Taip pat neįtraukėme failų, kurie yra susiję su navigacija, kadangi manome, kad paprasčiau ir aiškiau tai ištestuoti patiems su realiai veikiančia programėle nei aprašyti tai vienetų testais. Taip pat išskyrėme kreipimosi į API failus bei susijusius failus su tuo, kadangi šuos funkcionalumus jau testuojame serverio pusėje.
Mūsų „Jest“ konfigūracijos atrodo taip:

```
"jest": {
    "preset": "jest-expo",
    "setupFiles": [
      "./jest/setup.js"
    ],
    "transformIgnorePatterns": [
                                  "node_modules/(?!((jest-)?react-native|@react-native(-
community)?)|expo(nent)?|@expo(nent)?/.*|@expo-google-fonts/.*|react-navigation|@react-
navigation/.*|@unimodules/.*|unimodules|sentry-expo|native-base|react-native-
svg|native-notify)"
    ],
    "collectCoverage": true,
    "collectCoverageFrom": [
      "**/*.{js,jsx}",
      "!**/coverage/**",
      "!**/node_modules/**",
      "!**/babel.config.js",
      "!**/jest.setup.js"
    ],
    "coveragePathIgnorePatterns": [
    "NomNom-frontend/.expo/metro/",
    "NomNom-frontend/api",
    "NomNom-frontend/navigation",
    "NomNom-frontend/context",
    "NomNom-frontend/store",
    "NomNom-frontend/App.js"
  ]
}
```

1.2.2. Vienetų testai

Iš viso aprašėme 14 testų rinkinių (test suites), kuriuos sudaro iš viso 73 testai. Pasirinkome ataskaitoje pateikti tik šiuos tris: sign up. mapComp ir postCreation. Išskyrėme būtent šiuos, kadangi sign up apima „formik“ karkaso testavimą, o mapComp ir postCreation buvo vienos sudėtingesnių klasių, kurias reikėjo testuoti.

Signup Tests

```
import React from "react";
import { render, fireEvent, screen, waitFor } from "@testing-library/react-native";
import { userEvent } from "@testing-library/react-native";

import Signup from "../screens/signup";
import { HandleSignUp } from "../api/auth";
import { NavigationContainer } from "@react-navigation/native";
import { getByTestId } from "@testing-library/dom";

jest.mock('@react-navigation/native', () => ({
   useNavigation: () => ({
      navigate: jest.fn(),
   }),
}));

jest.mock('../api/auth', () => ({
   HandleSignUp: jest.fn(() => Promise.resolve(true))
}))

//Helper functions that give needed input in correct format:
const usernameSuccessCase = async () => {
   const user = userEvent.setup();
   const { getByTestId } = render(<Signup />);
   await user.type(getByTestId('username'), 'aOper7');
};

const emailSuccessCase = async () => {
   const user = userEvent.setup();
   const { getByTestId } = render(<Signup/>);
   await user.type(getByTestId('email'), 'a7@gmail.com');
}

const passwordSuccessCase = async () => {
   const user = userEvent.setup();
   const { getByTestId } = render(<Signup />);
   await user.type(getByTestId('password'), 'QwertYui8o');
   await user.type(getByTestId('match'), 'QwertYui8o');
```

```
}

test('renders initial form fields and elements', () => {
  const { getByTestId, getByText } = render(<Signup />)
  // Input label
  expect(getByTestId('username')).toBeTruthy();
  expect(getByTestId('email')).toBeTruthy();
  expect(getByTestId('password')).toBeTruthy();
  expect(getByTestId('match')).toBeTruthy();

  // Button and link
  expect(getByTestId('signUpButton')).toBeTruthy();
  expect(getByText('Log In')).toBeTruthy();
});

describe('sign up', () => {
  it('should be successful login because data is correct', async () => {
    const { getByTestId } = render(<Signup/>);
    const user = userEvent.setup();

    await user.type(getByTestId('username'), 'aOper7');
    await user.type(getByTestId('email'), 'a7@gmail.com');
    await user.type(getByTestId('password'), 'QwertYui8o');
    await user.type(getByTestId('match'), 'QwertYui8o');

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(jest.mocked(HandleSignUp)).toHaveBeenCalledTimes(1); // HandleSignUp called
once
      expect(jest.mocked(HandleSignUp)).toHaveBeenCalledWith(
        'aOper7',
        'a7@gmail.com',
        'QwertYui8o',
        expect.any(Function), // Don't assert navigation logic here
        null
      );
    });
  });

  it('should be unsuccessful because username is taken', async () => {
    const { getByTestId, queryByText } = render(<Signup/>);
    const user = userEvent.setup();
    HandleSignUp.mockResolvedValueOnce(false);
    await user.type(getByTestId("username"), "evita");
    await user.type(getByTestId('email'), 'a7@gmail.com');
    await user.type(getByTestId('password'), 'QwertYui8o');
    await user.type(getByTestId('match'), 'QwertYui8o');

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(jest.mocked(HandleSignUp)).toHaveBeenCalledWith(
        'evita',
        'a7@gmail.com',
        'QwertYui8o',
        expect.any(Function),
        null
      );
    });

    await waitFor(() => {
      expect(queryByText("Unsuccessful sign up! Try again.")).toBeTruthy();
    });
  });
})

describe('username validation', () => {
  it('should be an error about too short of a username', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await user.type(getByTestId('username'), 'a');
    await emailSuccessCase();
    await passwordSuccessCase();

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(queryByText("Must be at least 5 characters!")).toBeTruthy();
    });
  });

  it('should be an error about too long of a username', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();
```

```
        await user.type(getByTestId('username'), 'aaaaaaaaaaaaaaaa');
        await emailSuccessCase();
        await passwordSuccessCase();

        const signUpButton = screen.getByTestId('signUpButton');
        await user.press(signUpButton);

        await waitFor(() => {
            expect(queryByText("Must be 15 characters or less!")).toBeTruthy();
        });
    });

    it('should be an error about empty username', async () => {
        const { getByTestId, queryByText } = render(<Signup />)
        const user = userEvent.setup();

        await user.type(getByTestId('username'), '');
        await emailSuccessCase();
        await passwordSuccessCase();

        const signUpButton = screen.getByTestId('signUpButton');
        await user.press(signUpButton);

        await waitFor(() => {
            expect(queryByText("Required")).toBeTruthy();
        });
    });
});
describe('email validation', () => {
    it('should be an error about empty email', async () => {
        const { getByTestId, queryByText } = render(<Signup />)
        const user = userEvent.setup();

        await usernameSuccessCase();
        await user.type(getByTestId('email'), '');
        await passwordSuccessCase();

        const signUpButton = screen.getByTestId('signUpButton');
        await user.press(signUpButton);

        await waitFor(() => {
            expect(queryByText("Required")).toBeTruthy();
        });
    });
```

```
it('should be an error about email address without @', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await user.type(getByTestId('email'), 'gmail.com');
    await passwordSuccessCase();

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
        expect(queryByText("Invalid email address!")).toBeTruthy();
    });
});

it('should be an error about email address without .', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await user.type(getByTestId('email'), 'e@gmailcom');
    await passwordSuccessCase();

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
        expect(queryByText("Invalid email address!")).toBeTruthy();
    });
});

it('should be an error about email address without text before @', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await user.type(getByTestId('email'), '@gmail.com');
    await passwordSuccessCase();

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
            expect(queryByText("Invalid email address!")).toBeTruthy();
        });
    });

    it('should be an error about email address without with @ and . next to each other', async () =>
{
        const { getByTestId, queryByText } = render(<Signup />)
        const user = userEvent.setup();

        await usernameSuccessCase();
        await user.type(getByTestId('email'), 'a@.com');
        await passwordSuccessCase();

        const signUpButton = screen.getByTestId('signUpButton');
        await user.press(signUpButton);

        await waitFor(() => {
            expect(queryByText("Invalid email address!")).toBeTruthy();
        });
    });
});

describe('Password validation', () => {
    it('should be an error about missing password', async () => {
        const { getByTestId, queryByText } = render(<Signup />)
        const user = userEvent.setup();

        await usernameSuccessCase();
        await emailSuccessCase();
        await user.type(getByTestId('password'), '');
        await user.type(getByTestId('match'), 'QwertYui8o');

        const signUpButton = screen.getByTestId('signUpButton');
        await user.press(signUpButton);

        await waitFor(() => {
            expect(queryByText('Required')).toBeTruthy();
        });
    });

    it('should be an error about too short of a password', async () => {
        const { getByTestId, queryByText } = render(<Signup />)
        const user = userEvent.setup();

        await usernameSuccessCase();
    await emailSuccessCase();
    await user.type(getByTestId('password'), 'qwertyu');
    await user.type(getByTestId('match'), 'qwertyu');

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(queryByText('Password must be at least 8 characters!')).toBeTruthy();
    });
  });

  it('should be an error about too missing lowercase letters', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await emailSuccessCase();
    await user.type(getByTestId('password'), 'QWERTYUIO1P');
    await user.type(getByTestId('match'), 'QWERTYUIO1P');

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(queryByText('Your password must have at least one lowercase')).toBeTruthy();
    });
  });

  it('should be an error about too missing uppercase letters', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await emailSuccessCase();
    await user.type(getByTestId('password'), 'qwertyui10o');
    await user.type(getByTestId('match'), 'qwertyui10o');

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(queryByText('Your password must have at least one uppercase')).toBeTruthy();
    });
  });
  it('should be an error about too missing digits', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await emailSuccessCase();
    await user.type(getByTestId('password'), 'QwErtYUiop');
    await user.type(getByTestId('match'), 'QwErtYUiop');

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(queryByText('Your password must have at least one digit')).toBeTruthy();
    });
  });
});

describe('PasswordConfirmation', () => {
  it('should be an error about empty password confirmation', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await emailSuccessCase();
    await user.type(getByTestId('password'), 'QwertYui8o');
    await user.type(getByTestId('match'), '');

    const signUpButton = screen.getByTestId('signUpButton');
    await user.press(signUpButton);

    await waitFor(() => {
      expect(queryByText('Required')).toBeTruthy();
    });
  });

  it('should be an error about password confirmation not matching', async () => {
    const { getByTestId, queryByText } = render(<Signup />)
    const user = userEvent.setup();

    await usernameSuccessCase();
    await emailSuccessCase();
    await user.type(getByTestId('password'), 'QwertYui8o');
    await user.type(getByTestId('match'), 'Qwertui8o');
```

```
        const signUpButton = screen.getByTestId('signUpButton');
        await user.press(signUpButton);

        await waitFor(() => {
            expect(queryByText('Does not match with the password!')).toBeTruthy();
        });
    });
});

describe('Hide password', () => {
    it('password should be hidden at first', async () => {
        const { getByTestId } = render(<Signup />);
        const user = userEvent.setup();

        await passwordSuccessCase();
        const passwordInput = getByTestId('password');
        expect(passwordInput.props.secureTextEntry).toBe(true);
    })
    it('password should be shown at first', async () => {
        const { getByTestId } = render(<Signup/>);
        const user = userEvent.setup();

        await passwordSuccessCase();
        const passwordInput = getByTestId('password');
        fireEvent.press(getByTestId('hidePassword'));
        expect(passwordInput.props.secureTextEntry).toBe(false);
    });
});

describe('Hide password match', () => {
    it('password match should be hidden at first', async () => {
        const { getByTestId } = render(<Signup />);
        const user = userEvent.setup();

        await passwordSuccessCase();
        const matchInput = getByTestId('match');
        expect(matchInput.props.secureTextEntry).toBe(true);
    })
    it('password match should be shown at first', async () => {
        const { getByTestId } = render(<Signup/>);
        const user = userEvent.setup();

        await passwordSuccessCase();
        const matchInput = getByTestId('match');
        fireEvent.press(getByTestId('hideMatch'));
        expect(matchInput.props.secureTextEntry).toBe(false);
    });
});
```

MapComp tests

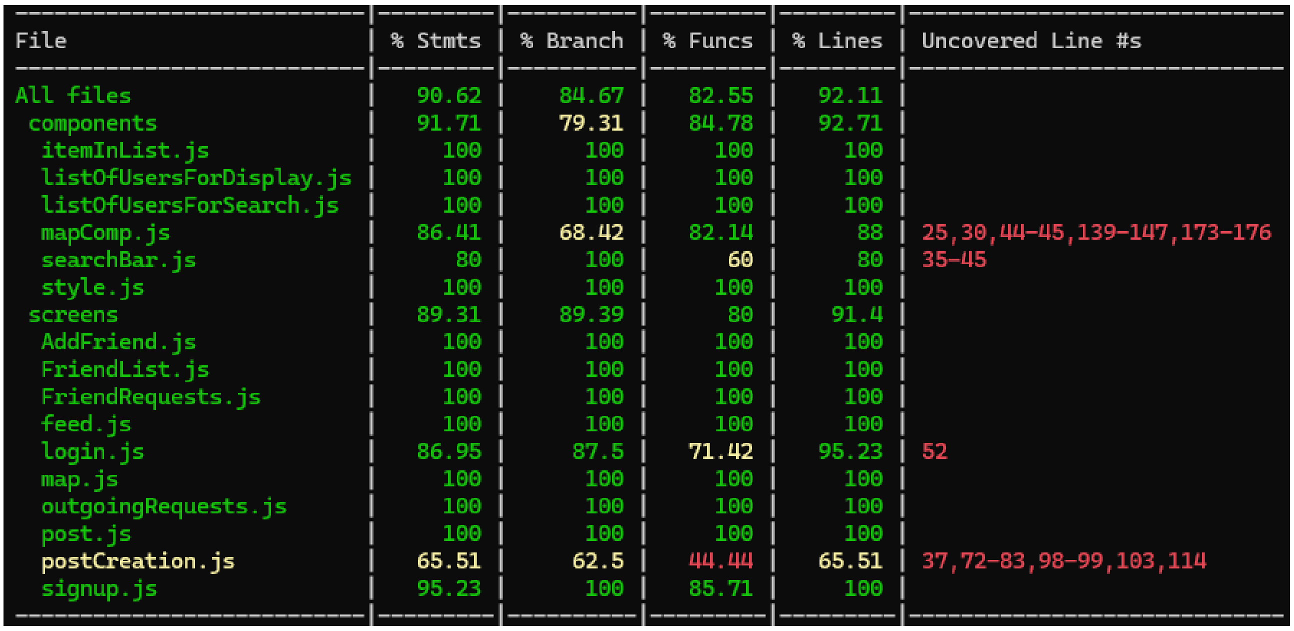
```
import React from 'react';
import { render, fireEvent, act, waitFor } from '@testing-library/react-native';
import MapComp from '../components/mapComp';
import * as Location from 'expo-location';
import { CreatePin, DeletePin } from '../api/pin';
import { getByTestId } from '@testing-library/react-native';
import { GetPost } from '../api/pin';

import { useNavigation } from '@react-navigation/native';

jest.mock('@react-navigation/native', () => ({
   useNavigation: jest.fn(),
}));




jest.mock('expo-location', () => ({
   requestForegroundPermissionsAsync: jest.fn().mockResolvedValue({ status: 'granted' }),
   getCurrentPositionAsync: jest.fn().mockResolvedValue({ coords: { latitude: 41.7128,
longitude: -75.0060 } }),
}));

jest.mock('react-native-maps', () => {
   const { View } = require('react-native');
   return {
     __esModule: true,
     default: jest.fn().mockImplementation((props) => (
       <View {...props} testID="mapComp">{props.children}</View>
     )),
     Marker: ({ onPress, children, ...props }) => (
       <View {...props} onPress={onPress}>
          {children}
       </View>
     ),
   };
  });

jest.mock('../api/pin', () => ({
   GetPins: jest.fn().mockImplementation((success) => success([{id: '2', latitude: 40.7128,
longitude: -74.0060}])),
   CreatePin: jest.fn().mockResolvedValue(),
   DeletePin: jest.fn().mockResolvedValue(),
   GetPost: jest.fn().mockImplementation((id, successCallback) => successCallback({})),
}));

jest.mock('@react-navigation/native', () => ({
   useNavigation: () => ({ navigate: jest.fn() }), // Mocking navigation object
   useIsFocused: jest.fn().mockReturnValue(true), // Mocking useIsFocused hook
}));



it('should render the map component', async () => {
   const { queryByTestId } = render(<MapComp />);

   await act(async () => {
      // Simulate the asynchronous behavior of requesting and getting location
      await    Location.requestForegroundPermissionsAsync.mockResolvedValueOnce({    status:
'granted' });
      await   Location.getCurrentPositionAsync.mockResolvedValueOnce({   coords:   {   latitude:
40.7128, longitude: -74.0060 } });

      // Wait for the component to update after location retrieval
      await new Promise(resolve => setTimeout(resolve, 0));
   });

   const map = queryByTestId('mapComp');

   expect(map).toBeTruthy();
});

it('should render pin confirmation popup when a location is clicked on the map', async () => {
   const { queryByTestId, getByText } = render(<MapComp />);

   await act(async () => {
      // Simulate the asynchronous behavior of requesting and getting location
      await    Location.requestForegroundPermissionsAsync.mockResolvedValueOnce({    status:
'granted' });
      await   Location.getCurrentPositionAsync.mockResolvedValueOnce({   coords:   {   latitude:
40.7128, longitude: -74.0060 } });

      // Wait for the component to update after location retrieval
      await new Promise(resolve => setTimeout(resolve, 0));
   });

   const map = queryByTestId('mapComp');
   expect(map).toBeTruthy();

   // Mock the event object with necessary properties
```

```
    const mockEvent = {
        nativeEvent: {
            coordinate: {
                latitude: 40.7,
                longitude: -74.0
            }
        }
    };

    // Click on the map to simulate user interaction
    fireEvent.press(map, mockEvent);

    // Wait for the component to update after click
    await new Promise(resolve => setTimeout(resolve, 0));

    // Check if the pin confirmation popup appears
    const popupText = getByText('Do you want to place the pin here?');
    expect(popupText).toBeTruthy();
});

it('should place a pin when "Place Pin" button is pressed', async () => {
    const { queryByTestId, getByText } = render(<MapComp />);

    await act(async () => {
        // Simulate the asynchronous behavior of requesting and getting location
        await Location.requestForegroundPermissionsAsync.mockResolvedValueOnce({ status: 
'granted' });
        await Location.getCurrentPositionAsync.mockResolvedValueOnce({ coords: { latitude: 
40.7128, longitude: -74.0060 } });

        // Wait for the component to update after location retrieval
        await new Promise(resolve => setTimeout(resolve, 0));
    });

    const map = queryByTestId('mapComp');
    expect(map).toBeTruthy();

    // Mock the event object with necessary properties
    const mockEvent = {
        nativeEvent: {
            coordinate: {
                latitude: 40.7,
                longitude: -74.0
            }
        }
    };

    // Click on the map to simulate user interaction
    fireEvent.press(map, mockEvent);

    // Wait for the component to update after click
    await new Promise(resolve => setTimeout(resolve, 0));

    // Check if the pin confirmation popup appears
    const popupText = getByText('Do you want to place the pin here?');
    expect(popupText).toBeTruthy();

    // Find and press the "Place Pin" button
    const placePinButton = getByText('Place Pin');
    fireEvent.press(placePinButton);

    // Wait for the pin to be created (API call is mocked)
    await new Promise(resolve => setTimeout(resolve, 0));

    // Verify that CreatePin function was called with correct coordinates
    expect(CreatePin).toHaveBeenCalledWith(mockEvent.nativeEvent.coordinate.latitude,
mockEvent.nativeEvent.coordinate.longitude, expect.any(Function), null);
});

test('shows "View Post" button when pin with a post is clicked', async () => {
    const { getByTestId, getByText, findByTestId } = render(<MapComp />);

    await waitFor(() => getByTestId('mapComp'));

    const marker = await findByTestId('MCmarker');
    fireEvent.press(marker);

    expect(getByText("View post")).toBeTruthy();
    expect(getByText("Delete Pin")).toBeTruthy();
    expect(getByText("Cancel")).toBeTruthy();
});

test('should delete pin when "Delete Pin" button is pressed', async () => {
    const { getByTestId, getByText, findByTestId } = render(<MapComp />);

    await waitFor(() => getByTestId('mapComp'));

    const marker = await findByTestId('MCmarker');
    fireEvent.press(marker);
    // Check if the "Delete Pin" button exists and press it
    const deletePinButton = getByText("Delete Pin");
    fireEvent.press(deletePinButton);

    // Wait for the pin to be deleted (API call is mocked)
    await new Promise(resolve => setTimeout(resolve, 0));

    // Verify that DeletePin function was called
    expect(DeletePin).toHaveBeenCalled();
});

it('should call api GetPost "View post" button is pressed', async () => {
    const { getByTestId, getByText, findByTestId } = render(<MapComp />);

    await waitFor(() => getByTestId('mapComp'));

    const marker = await findByTestId('MCmarker');
    fireEvent.press(marker);

    const viewPostButton = getByText("View post");
    fireEvent.press(viewPostButton);

    // Verify if GetPost function was called with the correct pin ID
    expect(GetPost).toHaveBeenCalled();
});

it('should close pin confirmation popup when cancel button is pressed', async () => {
    const { getByTestId, getByText, findByTestId, queryByText } = render(<MapComp />);

    await waitFor(() => getByTestId('mapComp'));

    const marker = await findByTestId('MCmarker');
    fireEvent.press(marker);

    // Find and press the "Cancel" button in the pin confirmation popup
    const cancelButton = getByText('Cancel');
    fireEvent.press(cancelButton);

    // Add a small delay to allow the component to update
    await new Promise(resolve => setTimeout(resolve, 100));

    // Check if the pin confirmation popup content is no longer present
    expect(queryByText("Pin options")).toBeNull();
});
```

PostCreation tests

```
import React from "react";
import { render, fireEvent, screen, waitFor } from "@testing-library/react-native";
import { userEvent } from "@testing-library/react-native";
import PostCreation from "../screens/postCreation";
import { AddPost, GetPost, UpdatePost } from "../api/pin";
import { act } from 'react-test-renderer';

jest.mock('@react-navigation/native', () => ({
   useNavigation: () => ({
      navigate: jest.fn(),
      goBack: jest.fn()
   }),
}));

jest.mock('../context/loginProvider', () => ({
   useLogin: () => ({
      isLoggedIn: true,
   }),
}));



jest.mock('../api/pin', () => ({
   AddPost: jest.fn(),
   GetPost: jest.fn(),
   UpdatePost: jest.fn(),
}));

jest.mock('@react-native-community/datetimepicker', () => ({
   DateTimePickerAndroid: {
      open: jest.fn(),
   },
}));

const initialProps = {
   route: {
      params: {
         pinID: 1,
         latitude: '34.0522',
         longitude: '45.2437',
         initialPost: {
            id: 1,
            restaurantName: 'Test Restaurant',
            dateVisited: '2024-04-15',
            review: 'Great place! Would visit again 10/10',
            },
            editMode: false,
        },
    },
};

const initialPropsForEdit = {
    route: {
        params: {
            pinID: 1,
            latitude: '34.0522',
            longitude: '45.2437',
            initialPost: {
                id: 1,
                restaurantName: 'Test Restaurant',
                dateVisited: '2024-04-15',
                review: 'Great place! Would visit again 10/10',
            },
            editMode: true,
        },
    },
};

const nameSuccessCase = async () => {
    const user = userEvent.setup();
    const { getByPlaceholderText } = render(<PostCreation {...initialProps} />);
    await user.type(getByPlaceholderText("Name of a restaurant you've visited"), "aaaaaa");
};

const reviewSuccessCase = async () => {
    const user = userEvent.setup();
    const { getByPlaceholderText } = render(<PostCreation {...initialProps} />);
    await user.type(getByPlaceholderText("What was your experience like?"), "qwer tyuiop asdf");
};

describe("render initial form fields and elements correctly", () => {
    it('should render correctly when editMode = false', () => {
        const { getByText, getByTestId, getByPlaceholderText } = render(<PostCreation
{...initialProps} />);
        // text:
        expect(getByText("Tell us about your experience at the restaurant!")).toBeTruthy();
        expect(getByText("Location of the visited restaurant")).toBeTruthy();
        expect(getByText(`latitude: ${initialProps.route.params.latitude}`)).toBeTruthy();
        expect(getByText(`longitude: ${initialProps.route.params.longitude}`)).toBeTruthy();
        expect(getByText("Restaurant name")).toBeTruthy();
        expect(getByText("Date of your visit")).toBeTruthy();
        expect(getByText("Choose date")).toBeTruthy();
        expect(getByText("Description")).toBeTruthy();

        // input:
        expect(getByPlaceholderText("Name of a restaurant you've visited")).toBeTruthy();
        expect(getByPlaceholderText("What was your experience like?")).toBeTruthy();
        // buttons
        expect(getByTestId("datePicker")).toBeTruthy();
        expect(getByTestId("postButton")).toBeTruthy();
    });

    it('should render correctly when editMode = true', () => {
        const { getByText, getByTestId, getByPlaceholderText } = render(<PostCreation
{...initialPropsForEdit} />);
        // labels
        expect(getByText("Tell us about your experience at the restaurant!")).toBeTruthy();
        expect(getByText("Location of the visited restaurant")).toBeTruthy();
        expect(getByText(`latitude: ${initialProps.route.params.latitude}`)).toBeTruthy();
        expect(getByText(`longitude: ${initialProps.route.params.longitude}`)).toBeTruthy();
        expect(getByText("Restaurant name")).toBeTruthy();
        expect(getByText("Date of your visit")).toBeTruthy();
        expect(getByText("Choose date")).toBeTruthy();
        expect(getByText("Description")).toBeTruthy();

        // input:
        expect(getByPlaceholderText("Name of a restaurant you've visited")).toBeTruthy();
        expect(getByPlaceholderText("What was your experience like?")).toBeTruthy();
        // buttons
        expect(getByTestId("datePicker")).toBeTruthy();
        expect(getByTestId("postButton")).toBeTruthy();

        // post info:
        expect(getByTestId("name").props.value).toBe("Test Restaurant");
        expect(getByTestId("review").props.value).toBe("Great place! Would visit again 10/10");
        // expect(getByText("Great place! Would visit again 10/10")).toBeTruthy();
    });
});

describe('form submission', () => {
    it('should call AddPost', async () => {
        const { getByTestId, getByPlaceholderText } = render(<PostCreation {...initialProps} />);
        const user = userEvent.setup();
        await userEvent.type(getByPlaceholderText("Name of a restaurant you've visited"), "New Restaurant");
        await userEvent.type(getByPlaceholderText("What was your experience like?"), "This is a great place!");

        const postButton = getByTestId("postButton");
        await user.press(postButton);

        await act(async () => {
            expect(AddPost).toHaveBeenCalledWith(
                1,
                "New Restaurant",
                "This is a great place!",
                expect.any(Date),
                expect.any(Function),
                null
            );
        });
    });
});

describe('initial form values are empty', () => {
    it('name should be empty', () => {
        const { getByPlaceholderText } = render(<PostCreation {...initialProps} />);
        expect(getByPlaceholderText("Name of a restaurant you've visited").props.value).toBe("");
    });
    it('review should be empty', () => {
        const { getByPlaceholderText } = render(<PostCreation {...initialProps} />);
        expect(getByPlaceholderText("What was your experience like?").props.value).toBe("");
    });
});
```

1.2.3. Padengimo ataskaita

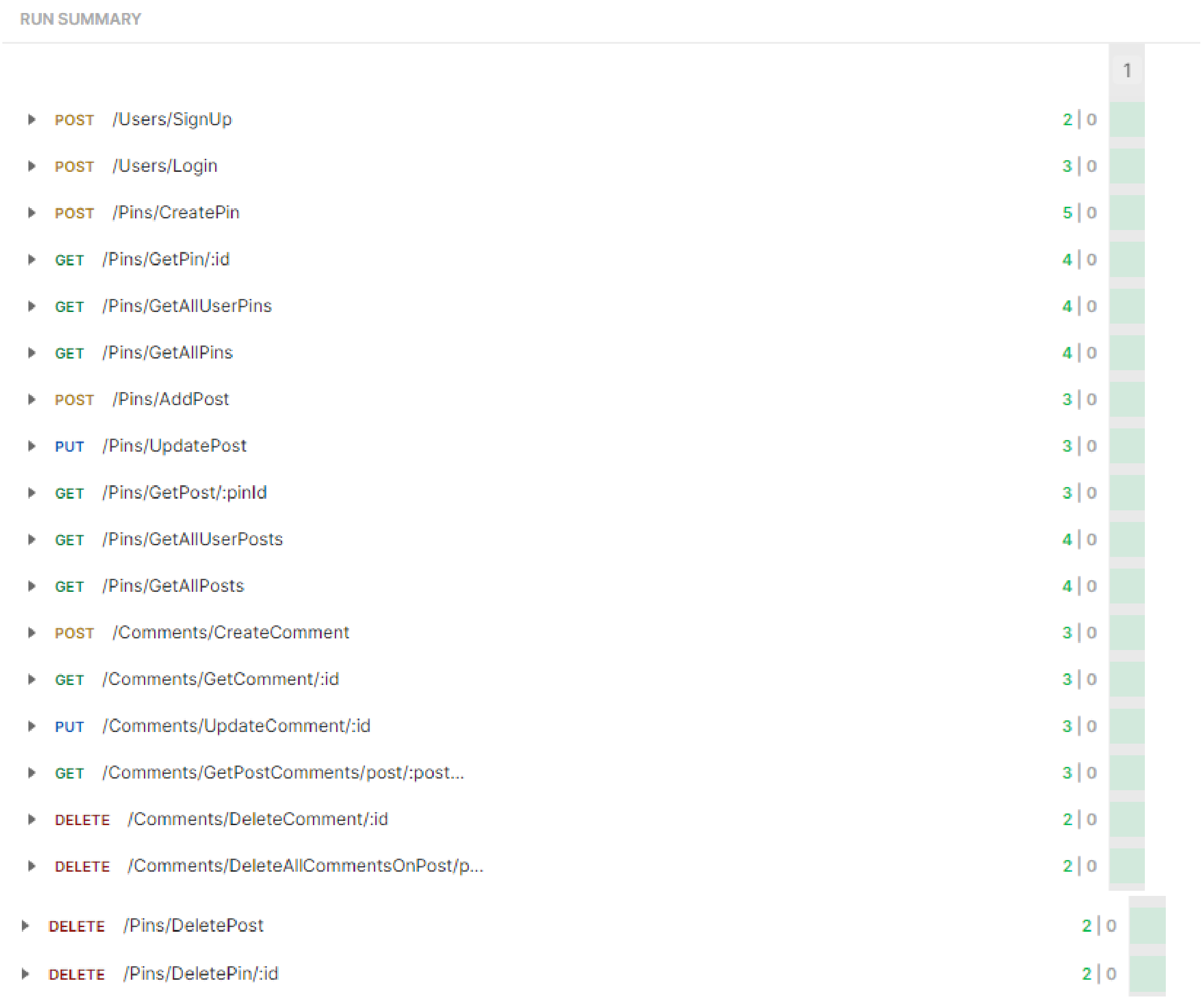


```
---------------------------|---------|----------|---------|---------|-----------------------------
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------------|---------|----------|---------|---------|-----------------------------
All files                  |   90.62 |    84.67 |   82.55 |   92.11 |
 components                |   91.71 |    79.31 |   84.78 |   92.71 |
  itemInList.js            |     100 |      100 |     100 |     100 |
  listOfUsersForDisplay.js |     100 |      100 |     100 |     100 |
  listOfUsersForSearch.js  |     100 |      100 |     100 |     100 |
  mapComp.js               |   86.41 |    68.42 |   82.14 |      88 | 25,30,44-45,139-147,173-176
  searchBar.js             |      80 |      100 |      60 |      80 | 35-45
  style.js                 |     100 |      100 |     100 |     100 |
 screens                   |   89.31 |    89.39 |      80 |    91.4 |
  AddFriend.js             |     100 |      100 |     100 |     100 |
  FriendList.js            |     100 |      100 |     100 |     100 |
  FriendRequests.js        |     100 |      100 |     100 |     100 |
  feed.js                  |     100 |      100 |     100 |     100 |
  login.js                 |   86.95 |     87.5 |   71.42 |   95.23 | 52
  map.js                   |     100 |      100 |     100 |     100 |
  outgoingRequests.js      |     100 |      100 |     100 |     100 |
  post.js                  |     100 |      100 |     100 |     100 |
  postCreation.js          |   65.51 |     62.5 |   44.44 |   65.51 | 37,72-83,98-99,103,114
  signup.js                |   95.23 |      100 |   85.71 |     100 |
---------------------------|---------|----------|---------|---------|-----------------------------
```

2. Integraciniai testai

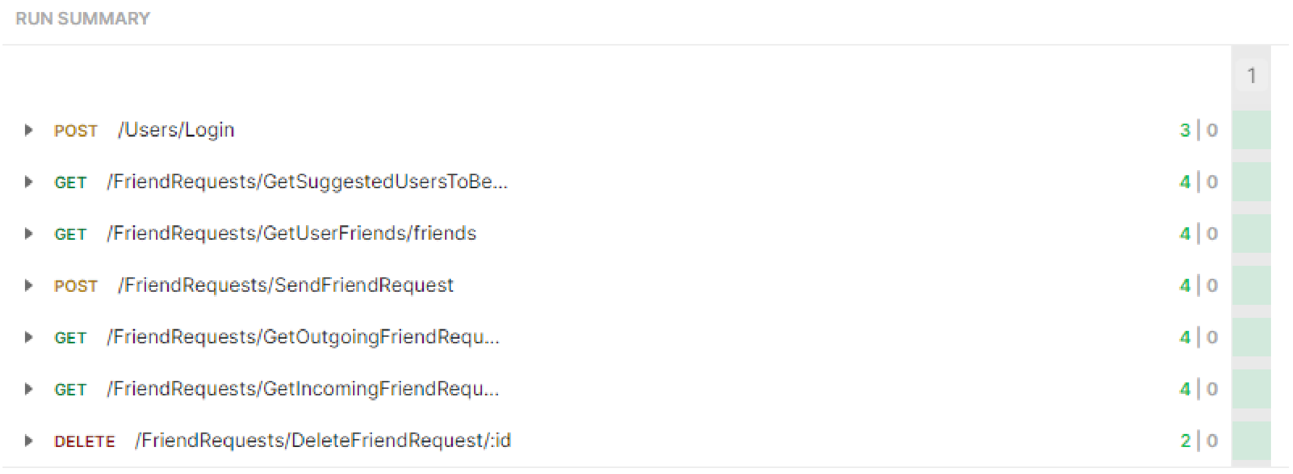
Integracinius testus vykdėme naudodami Postman programą. Tikslas – patikrinti, ar sistemos komponentai vienas su kitu tinkamai komunikuoja nuo pat užklausos išsiuntimo, iki duomenų gavimo/saugojimo duomenų bazėje.

Savo API įsikėlėme į Postman kolekciją .json formatu. Tuomet nustatėme reikiamus aplinkos kintamuosius, užpildėme užklausų kūnus bei parašėme užklausoms JavaScript testus. Kolekcijos užklausas paleidome su Collection Run funkcija. Rezultatas Postman programoje:

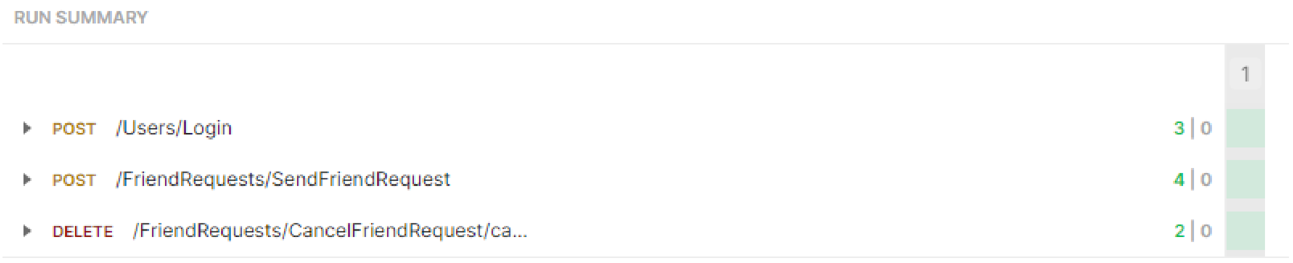


RUN SUMMARY

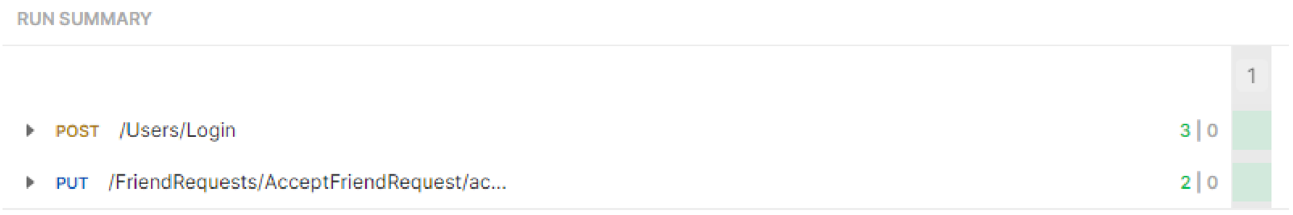
			1
POST	/Users/SignUp	2 | 0	🟩
POST	/Users/Login	3 | 0	🟩
POST	/Pins/CreatePin	5 | 0	🟩
GET	/Pins/GetPin/:id	4 | 0	🟩
GET	/Pins/GetAllUserPins	4 | 0	🟩
GET	/Pins/GetAllPins	4 | 0	🟩
POST	/Pins/AddPost	3 | 0	🟩
PUT	/Pins/UpdatePost	3 | 0	🟩
GET	/Pins/GetPost/:pinId	3 | 0	🟩
GET	/Pins/GetAllUserPosts	4 | 0	🟩
GET	/Pins/GetAllPosts	4 | 0	🟩
POST	/Comments/CreateComment	3 | 0	🟩
GET	/Comments/GetComment/:id	3 | 0	🟩
PUT	/Comments/UpdateComment/:id	3 | 0	🟩
GET	/Comments/GetPostComments/post/:post...	3 | 0	🟩
DELETE	/Comments/DeleteComment/:id	2 | 0	🟩
DELETE	/Comments/DeleteAllCommentsOnPost/p...	2 | 0	🟩
DELETE	/Pins/DeletePost	2 | 0	🟩
DELETE	/Pins/DeletePin/:id	2 | 0	🟩



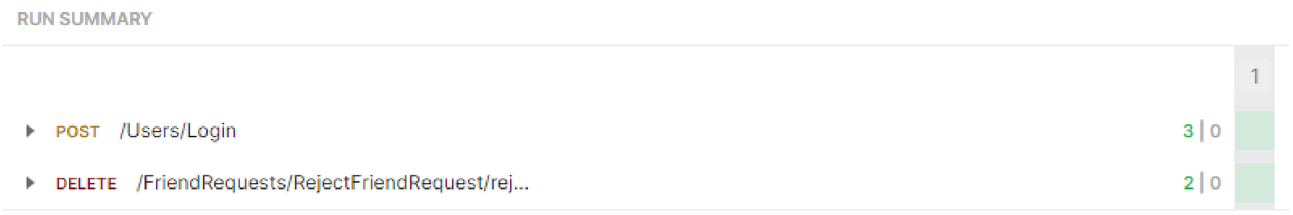
```
RUN SUMMARY
                                                              1
▸ POST    /Users/Login                                    3 | 0
▸ GET     /FriendRequests/GetSuggestedUsersToBe...        4 | 0
▸ GET     /FriendRequests/GetUserFriends/friends          4 | 0
▸ POST    /FriendRequests/SendFriendRequest               4 | 0
▸ GET     /FriendRequests/GetOutgoingFriendRequ...        4 | 0
▸ GET     /FriendRequests/GetIncomingFriendRequ...        4 | 0
▸ DELETE  /FriendRequests/DeleteFriendRequest/:id         2 | 0
```



```
RUN SUMMARY
                                                              1
▸ POST    /Users/Login                                    3 | 0
▸ POST    /FriendRequests/SendFriendRequest               4 | 0
▸ DELETE  /FriendRequests/CancelFriendRequest/ca...       2 | 0
```



```
RUN SUMMARY
                                                              1
▸ POST    /Users/Login                                    3 | 0
▸ PUT     /FriendRequests/AcceptFriendRequest/ac...       2 | 0
```



```
RUN SUMMARY
                                                              1
▸ POST    /Users/Login                                    3 | 0
▸ DELETE  /FriendRequests/RejectFriendRequest/rej...      2 | 0
```

Išvados

1. Vienetų testais pavyko padengti 90.62% kliento pusėje ir 89% serverio pusėje.
2. Testuodami išbandėme objektų „mockinimą“.
3. Integraciniais testais validavome komunikaciją tarp API endpoint’ų ir Data Access sluoksnio.