Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 3 з дисципліни

«Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

Виконав студент ІП-15, Мочалов Дмитро
(шифр, прізвище, ім'я, по батькові)

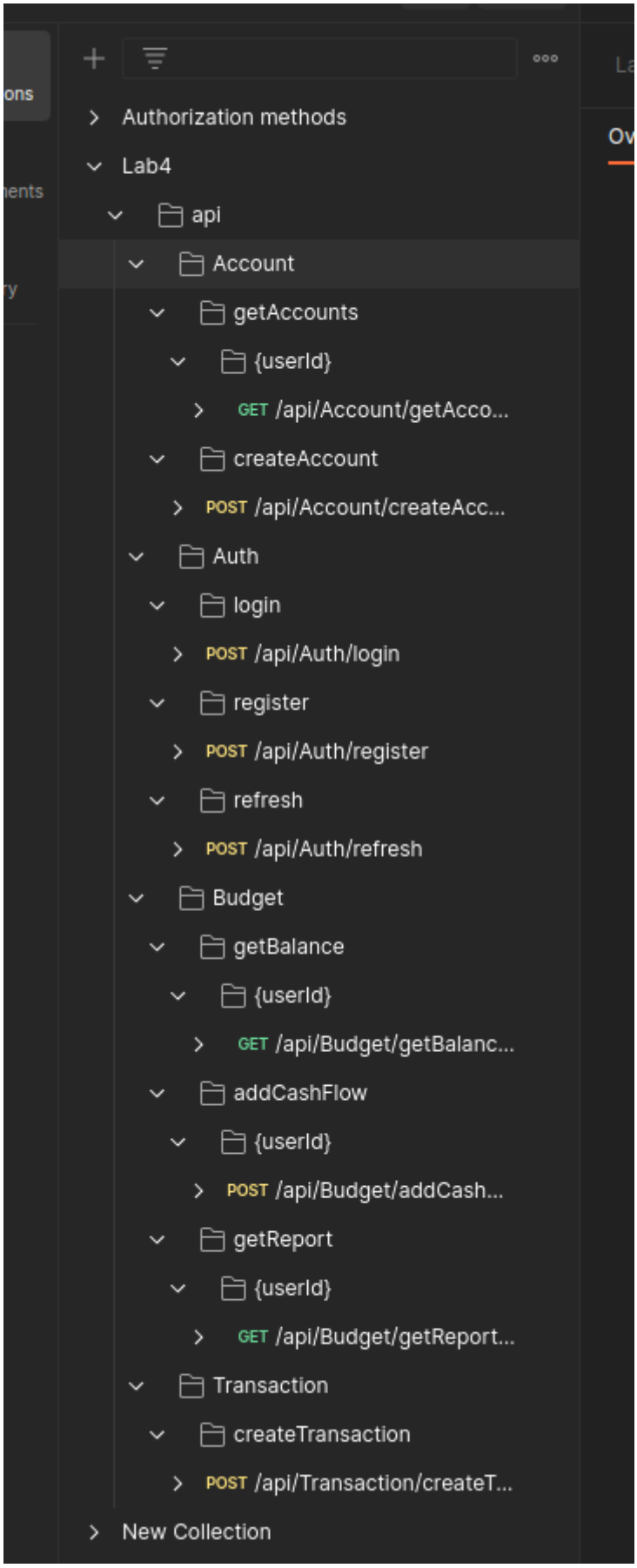
Перевірив Бардін Владислав
(прізвище, ім'я, по батькові)

Київ 2023

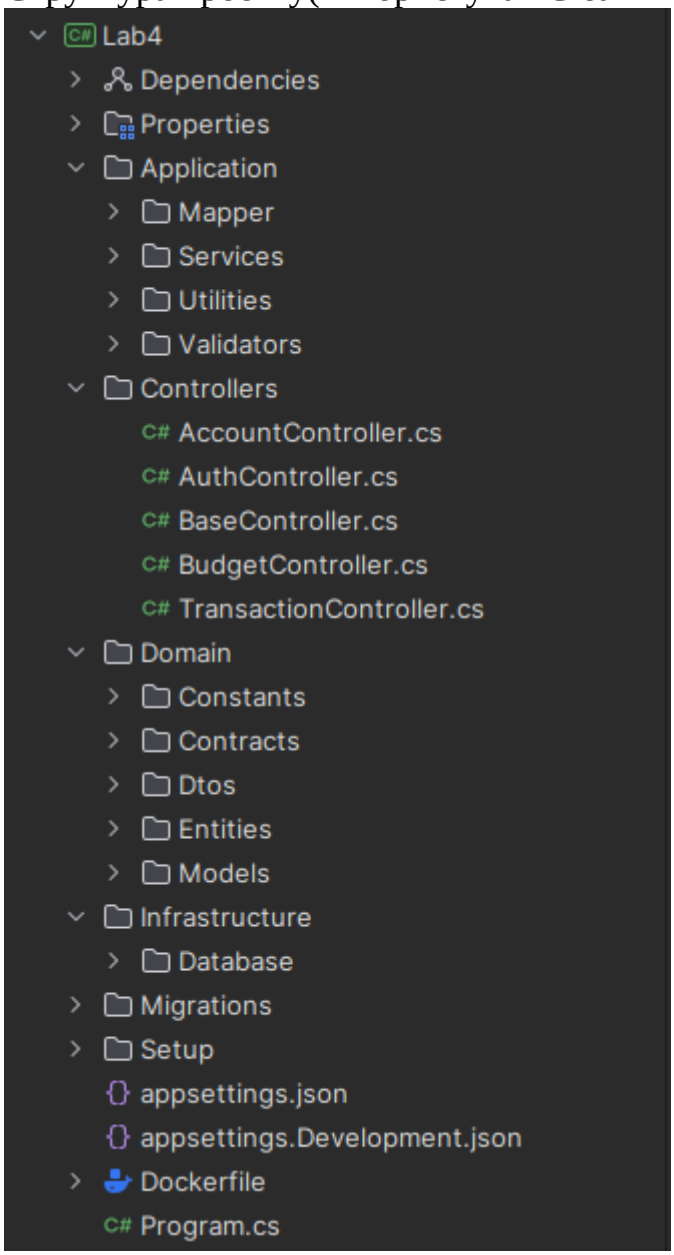
Завдання:

Практична частина:
1. Використовуючи архітектуру розроблену в попередній роботі Імплементувати REST веб-API, використавши N Layered архітектуру.
2. Покрити модульними та інтеграційними тестами основні компоненти рішення.
3. Експортувати розроблене API до Postman.

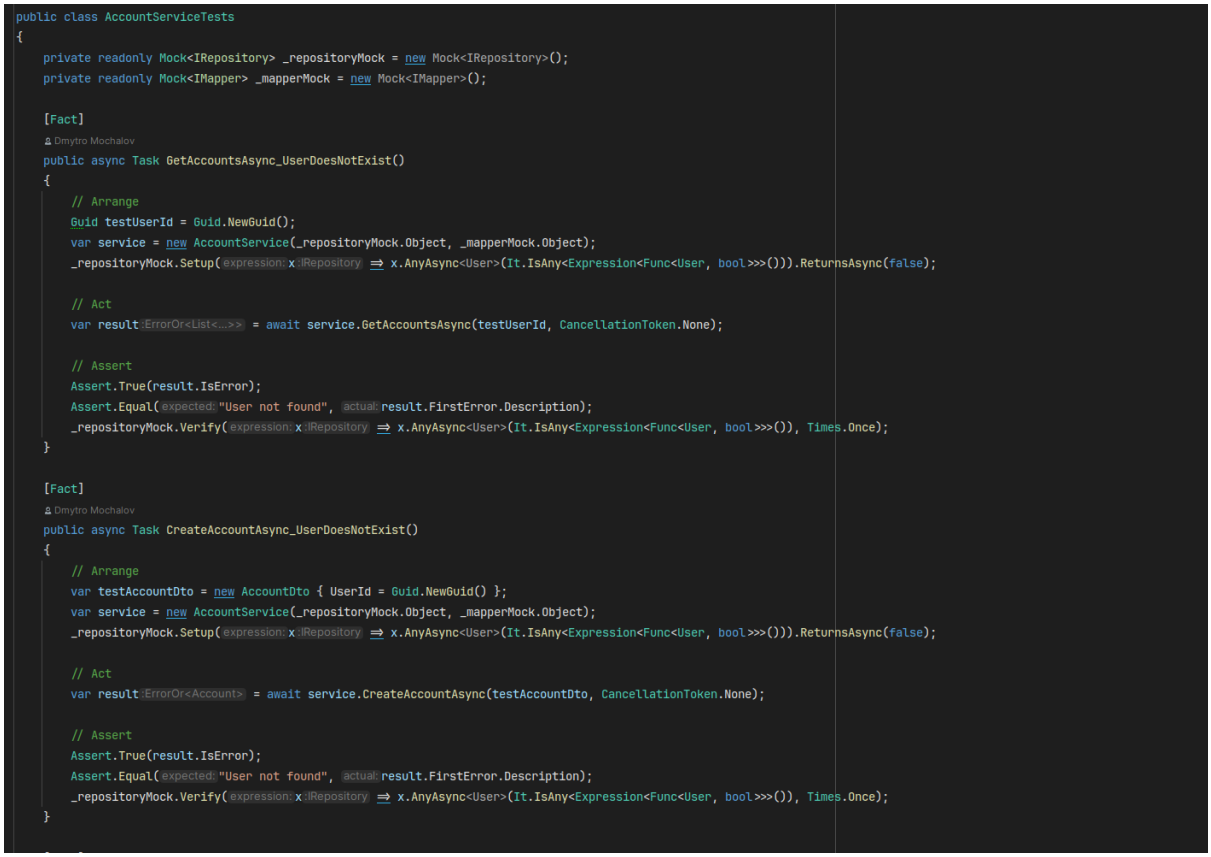
Експортоване API в Postman(експортував за допомогою сваггера):


Authorization methods
Lab4
api
Account
getAccounts
{userId}
GET /api/Account/getAcco...
createAccount
POST /api/Account/createAcc...
Auth
login
POST /api/Auth/login
register
POST /api/Auth/register
refresh
POST /api/Auth/refresh
Budget
getBalance
{userId}
GET /api/Budget/getBalanc...
addCashFlow
{userId}
POST /api/Budget/addCash...
getReport
{userId}
GET /api/Budget/getReport...
Transaction
createTransaction
POST /api/Transaction/createT...
New Collection

Структура проекту(використував Clean Architecture):



Юніт тести:



```
public class AccountServiceTests
{
    private readonly Mock<IRepository> _repositoryMock = new Mock<IRepository>();
    private readonly Mock<IMapper> _mapperMock = new Mock<IMapper>();

    [Fact]
    Dmytro Mochalov
    public async Task GetAccountsAsync_UserDoesNotExist()
    {
        // Arrange
        Guid testUserId = Guid.NewGuid();
        var service = new AccountService(_repositoryMock.Object, _mapperMock.Object);
        _repositoryMock.Setup(expression: x :IRepository => x.AnyAsync<User>(It.IsAny<Expression<Func<User, bool>>>())).ReturnsAsync(false);

        // Act
        var result :ErrorOr<List<...>> = await service.GetAccountsAsync(testUserId, CancellationToken.None);

        // Assert
        Assert.True(result.IsError);
        Assert.Equal(expected: "User not found", actual: result.FirstError.Description);
        _repositoryMock.Verify(expression: x :IRepository => x.AnyAsync<User>(It.IsAny<Expression<Func<User, bool>>>()), Times.Once);
    }

    [Fact]
    Dmytro Mochalov
    public async Task CreateAccountAsync_UserDoesNotExist()
    {
        // Arrange
        var testAccountDto = new AccountDto { UserId = Guid.NewGuid() };
        var service = new AccountService(_repositoryMock.Object, _mapperMock.Object);
        _repositoryMock.Setup(expression: x :IRepository => x.AnyAsync<User>(It.IsAny<Expression<Func<User, bool>>>())).ReturnsAsync(false);

        // Act
        var result :ErrorOr<Account> = await service.CreateAccountAsync(testAccountDto, CancellationToken.None);

        // Assert
        Assert.True(result.IsError);
        Assert.Equal(expected: "User not found", actual: result.FirstError.Description);
        _repositoryMock.Verify(expression: x :IRepository => x.AnyAsync<User>(It.IsAny<Expression<Func<User, bool>>>()), Times.Once);
    }
```

Сетап інтеграційних тестів та приклад самих тестів:

```
7 usages  Dmytro Mochalov  1 exposing API
public class IntegrationTestWebAppFactory : WebApplicationFactory<Program>, IAsyncLifetime
{
    private readonly PostgreSqlContainer _dbContainer = new PostgreSqlBuilder()
        .WithImage("postgres")
        .WithDatabase("FinanceDb")
        .WithUsername("postgres_finance")
        .WithPassword("postgres") // PostgreSqlBuilder
        .Build();

    Dmytro Mochalov
    protected override void ConfigureWebHost(IWebHostBuilder builder)
    {
        builder.ConfigureTestServices(services: IServiceCollection =>
        {
            var descriptor = services.SingleOrDefault(
                d: ServiceDescriptor => d.ServiceType ==
                    typeof(DbContextOptions<FinanceDbContext>));
            if (descriptor is not null)
            {
                services.Remove(descriptor);
            }

            services.AddDbContext<FinanceDbContext>(optionsAction: opt: DbContextOptionsBuilder =>
            {
                opt.UseNpgsql(_dbContainer.GetConnectionString());
            });
        });
    }

    Dmytro Mochalov
    public Task InitializeAsync()
    {
        return _dbContainer.StartAsync();
    }

    Dmytro Mochalov
    public new Task DisposeAsync()
    {
        return _dbContainer.StopAsync();
    }
}
```

```
using Microsoft.Extensions.DependencyInjection;

namespace Lab4IntegrationTests;

4 usages  4 inheritors  Dmytro Mochalov
public abstract class BaseIntegrationTest : IClassFixture<IntegrationTestWebAppFactory>
{
    protected readonly IntegrationTestWebAppFactory _factory;
    private readonly IServiceScope _scope;
    protected readonly FinanceDbContext FinanceDbContext;
    protected HttpClient _client;

    4 usages  Dmytro Mochalov
    protected BaseIntegrationTest(IntegrationTestWebAppFactory factory)
    {
        _scope = factory.Services.CreateScope();
        _factory = factory;
        FinanceDbContext = _scope.ServiceProvider // IServiceProvider
            .GetRequiredService<FinanceDbContext>(); // FinanceDbContext
        _client = factory.CreateClient(new WebApplicationFactoryClientOptions
        {
            AllowAutoRedirect = false
        }); // HttpClient
        _client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(scheme: "Bearer", parameter: JwtMock.GenerateJwtToken());
    }
}
```

```
public class AccountTests : BaseIntegrationTest
{
    Dmytro Mochalov
    public AccountTests(IntegrationTestWebAppFactory factory) : base(factory)
    {

    }

    [Fact]
    Dmytro Mochalov
    public async Task Get_Account_Without_Auth_Returns_Unauthorized()
    {
        // Arrange
        var userId = Guid.NewGuid();
        _client.DefaultRequestHeaders.Authorization = null;
        // Act
        var request:HttpResponseMessage = await _client.GetAsync(requestUri: $"api/Account/getAccounts/{userId}");

        // Assert
        Assert.Equal(expected: HttpStatusCode.Unauthorized, actual: request.StatusCode);
    }

    [Fact]
    Dmytro Mochalov
    public async Task Get_Not_Existing_Account_Returns_BadRequest()
    {
        // Arrange
        var userId = Guid.NewGuid();



        // Act
        var request:HttpResponseMessage = await _client.GetAsync(requestUri: $"api/Account/getAccounts/{userId}");

        // Assert
        Assert.Equal(expected: HttpStatusCode.BadRequest, actual: request.StatusCode);
    }
```