Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной инженерии и компьютерной техники

**Лабораторная работа №1**

по дисциплине

«Тестирование программного обеспечения»

**Выполнил**

Ореховский А.,

группа P3317

**Преподаватель**

Исаев И. В.

Санкт-Петербург

2020

Оглавление

[Цель 3](#_Toc43416854)

[Приложение 3](#_Toc43416855)

[Описание NUnit 3](#_Toc43416856)

[Требования 3](#_Toc43416857)

[Найденные баги 5](#_Toc43416858)

[Листинг Кода 5](#_Toc43416859)

[Корень решения 5](#_Toc43416860)

[Commons 6](#_Toc43416861)

[*Parser.cs* 6](#_Toc43416862)

[Controllers 10](#_Toc43416863)

[Hubs 11](#_Toc43416864)

[Models 12](#_Toc43416865)

[Pages 13](#_Toc43416866)

[Settings 14](#_Toc43416867)

[wwwroot/js 15](#_Toc43416868)

[Листинг тестов 16](#_Toc43416869)

[OneTimeControllerTests 16](#_Toc43416870)

[ParserTests 17](#_Toc43416871)

[RequesterTests 23](#_Toc43416872)

[Покрытие тестами 23](#_Toc43416873)

[Выводы 24](#_Toc43416874)

# Цель

Необходимо разработать консольное приложение (по вариантам):

* Сформировать функциональные требования (не менее 30);
* Реализовать требования;
* Покрыть код модульными тестами (среднее покрытие в coverage 70%).

# Приложение

Приложение для «микрообучения». Постройте приложение, высылающее вам ежедневно (или другой интервал) одну страницу по теме, которую вы хотите изучить. Это может быть случайная страница из Википедии, из документации по Kotlin, любое другое.

# Описание NUnit

NUnit — открытая среда юнит-тестирования приложений для .NET. Она была портирована с языка Java (библиотека JUnit). Первые версии NUnit были написаны на J#, но затем весь код был переписан на C# с использованием таких новшеств .NET, как атрибуты.

*Пример теста NUnit*

using NUnit.Framework;

[TestFixture]

public class ExampleTestOfNUnit

{

[Test]

public void TestMultiplication()

{

Assert.AreEqual(6, 3 \* 2, "Умножение");

}

}

# Требования

1. Веб приложение должно предоставлять эндпоинт, на который возможно подписаться
2. Данный эндпоинт должен иметь опциональный параметр action, который говорит об ожидаемом результате работы эндпоинта.
3. В качестве возможных значений параметра action допустимы:
   1. query
   2. help
4. В качестве стандартного значения параметра action следует использовать “query”.
5. В случае если значение параметра action является недопустимым, необходимо возвратить соответствующий ответ.
6. В случае возникновения ошибки, данный эндпоинт должен возвратить стандартный ответ об ошибке.
7. Данный эндпоинт должен опционально предоставлять возможность выбора, в качестве параметра, более детального ответа в случае ошибки.
8. Если параметр ответа в случае ошибки не соответствует значению “detailed”, необходимо возвратить соответствующий ответ.
9. В случае возникновения ошибки, связанной с неподдерживаемым параметром, эндпоинт должен проверить все оставшиеся параметры, и вывести все ошибки, связанные с инвалидностью параметров.
10. В качестве обязательного параметра эндпоинт должен принимать топик, по которому пользователь приложения желает получать статьи.
11. Параметр названия топика должен иметь строковый тип данных.
12. В случае отсутствия названия топика, необходимо возвратить соответствующий ответ.
13. В случае если название топика не найдено в базе данных Wikipedia, необходимо возвратить соответствующий ответ.
14. В качестве интервала времени по умолчанию, необходимо установить 1 сутки (24 часа).
15. Данный эндпоинт должен опционально предоставлять возможность установить интервал времени, через который будет производиться новая отправка сообщений.
16. Данный интервал можно указать в целочисленном типе (int).
17. Если интервал не является целочисленным типом, необходимо возвратить соответствующий ответ.
18. В случае если указанный период времени меньше нуля, необходимо возвратить соответствующий ответ.
19. В случае если указанный период времени равен нулю, необходимо возвратить соответствующий ответ.
20. В качестве возвращаемого формата данных по умолчанию необходимо использовать html.
21. Данный эндпоинт должен опционально предоставлять возможность установить формат возвращаемых данных из следующего списка:
    1. html
    2. title
22. В случае, если параметр формата возвращаемых данных не содержится в списке доступных форматов, необходимо возвратить соответствующий ответ.
23. В качестве порядка возвращаемых статей по умолчанию необходимо использовать алфавитный порядок.
24. Данный эндпоинт должен опционально предоставлять возможность установить порядок возвращаемых статей через параметр, который принадлежит следующему списку:
    1. Алфавитный порядок (alphabet)
    2. Случайный порядок (random)
    3. Порядок, основанный по дате добавления (date)
25. Параметр порядка должен иметь строковый тип данных.
26. В случае, если параметр порядка не содержится в списке допустимых параметров порядка, необходимо возвратить соответствующий ответ.
27. Приложение должно предоставлять эндпоинт, который возвращает единовременно случайную статью по данному топику, название которого передается в качестве параметра.
28. В случае, если название топика не найдено в базе данных Wikipedia, необходимо возвратить соответствующий ответ.
29. Приложение должно предоставлять эндпоинт, который выдает страницу по заданному названию.
30. Последний эндпоинт должен иметь 2 обязательных параметра: название страницы(title) и формат данных (format).
31. В случае, если не было найдено никаких данных по заданному названию, необходимо возвратить соответствующий ответ.
32. В случае если формат не соответствует значениям “html” или “title”, необходимо возвратить соответствующий ответ.

# Найденные баги

В ходе тестирования моего приложения я нашел довольно мало баг. По моему мнению, это связанно с тем, что я писал код опираясь на требования и строго их придерживался. Так как это приложение относительно небольшое и требований тут относительно мало, я и не совершил достаточно много ошибок. Тем не менее, мною все же были допущены и исправлены баги. Так, например я забыл указать при составлении требований, что интервал должен быть больше нуля, но смог увидеть этот недочет при тестировании всевозможных комбинаций параметров у моих методов. Таким образом я написал следующие тесты:

*Пример тестов интервала*

[Test]

public void NegativeIntervalTest2()

{

var interval = "-1";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveNegativeIntervalError(),

(string)parseResult.Value);

}

[Test]

public void ZeroIntervalTest1()

{

var interval = "0";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveZeroIntervalError(),

(string)parseResult.Value);

}

# Листинг Кода

## Корень решения

*Program.cs*

namespace ifmo.tpo.lab1

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

*Startup.cs*

namespace ifmo.tpo.lab1

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddRazorPages();

services.AddSignalR(hubOptions => {

hubOptions.EnableDetailedErrors = true;

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapRazorPages();

endpoints.MapControllers();

endpoints.MapHub<BroadcastHub>("/broadcast");

});

}

}

}

## Commons

## *Parser.cs*

namespace ifmo.tpo.lab1.Commons

{

public static class Parser

{

public static async Task<Result> Parse(string action, string topic, string errors, string interval, string format, string order)

{

var error = new List<string>();

var options = new Dictionary<SubscriptionOptions, object>();

var result = ParseString(action, AttributeOptions.Action);

if (result.Success)

{

options.Add(SubscriptionOptions.Action, result.Value);

}

else

{

error.Add((string)result.Value);

}

result = await ParseTopic(topic, action);

if (result.Success)

{

options.Add(SubscriptionOptions.Topic, result.Value);

}

else

{

error.Add((string)result.Value);

}

result = ParseString(errors, AttributeOptions.Errors);

if (result.Success)

{

options.Add(SubscriptionOptions.Errors, result.Value);

}

else

{

error.Add((string)result.Value);

}

result = ParseInterval(interval);

if (result.Success)

{

options.Add(SubscriptionOptions.Interval, result.Value);

}

else

{

error.Add((string)result.Value);

}

result = ParseString(format, AttributeOptions.Format);

if (result.Success)

{

options.Add(SubscriptionOptions.Format, result.Value);

}

else

{

error.Add((string)result.Value);

}

result = ParseString(order, AttributeOptions.Order);

if (result.Success)

{

options.Add(SubscriptionOptions.Order, result.Value);

}

else

{

error.Add((string)result.Value);

}

return error.Any() ? new Result(false, error): new Result(true, options);

}

public static async Task<Result> ParseTopic(string topic, string action)

{

if (topic is null)

{

var error = GiveAttributeRequiredError("Topic");

return action == "help" ? new Result(true) : new Result(false, error);

}

var isTopicExists = await Requester.IsTopicExists(topic);

if (isTopicExists)

{

return new Result(true, topic);

}

else

{

var error = GiveNoDataFoundError("Topic", topic);

return new Result(false, error);

}

}

public static Result ParseInterval(string interval)

{

if (interval is null)

{

return new Result(true, TimeSpan.FromDays(1));

}

if (int.TryParse(interval, out var value))

{

if (value < 0)

{

return new Result(false, GiveNegativeIntervalError());

}

if (value == 0)

{

return new Result(false, GiveZeroIntervalError());

}

return new Result(true, TimeSpan.FromSeconds(value));

}

else

{

var error = GiveWrongTypeError("Interval", "Integer");

return new Result(false, error);

}

}

public static Result ParseString(string value, Option option)

=> ParseString(value, option, true);

public static Result ParseString(string value, Option option, bool nullable)

{

if (value is null)

{

var error = GiveAttributeRequiredError(option.AttributeName);

return nullable ? new Result(true, option.Default) : new Result(false, error);

}

if (option.Values.Contains(value))

{

return new Result(true, value);

}

else

{

var error = GiveWrongOptionError(option.AttributeName);

return new Result(false, error);

}

}

}

}

*Requester.cs*

namespace ifmo.tpo.lab1.Commons

{

public static class Requester

{

private static readonly HttpClient Client = new HttpClient();

public static async Task<List<string>> GetPages(string topic)

{

var query = GetListUrl(topic);

var json = await GetJson<ResponseJson>(query);

return json.Query.CategoryMembers

.Where(cm => cm.Ns == 0)

.Select(cm => cm.Title).ToList();

}

public static string GetPageByTitleLink(string title, string format)

{

return GetPageUrl(title, format);

}

public static async Task<string> GetPageByTitle(string title, string format)

{

var url = GetPageUrl(title, format);

HttpResponseMessage response = await Client.GetAsync(url);

if (response.IsSuccessStatusCode)

{

return await response.Content.ReadAsStringAsync();

}

else

{

Console.WriteLine("{0} ({1})", (int)response.StatusCode, response.ReasonPhrase);

return default;

}

}

public static async Task<bool> IsTopicExists(string topic)

{

var query = GetListUrl(topic);

var json = await GetJson<ResponseJson>(query);

return json.Query.CategoryMembers.Any();

}

private static async Task<T> GetJson<T>(string url)

{

var data = await GetRequest(url);

T jsonObject = Newtonsoft.Json.JsonConvert.DeserializeObject<T>(data.ToString());

return jsonObject;

}

private static async Task<object> GetRequest(string url)

{

HttpResponseMessage response = await Client.GetAsync(url);

if (response.IsSuccessStatusCode)

{

return await response.Content.ReadAsAsync<object>();

}

else

{

Console.WriteLine("{0} ({1})", (int)response.StatusCode, response.ReasonPhrase);

return default;

}

}

private static string GetListUrl(string topic)

=> $"https://en.wikipedia.org/w/api.php?action=query&list=categorymembers&cmtitle=Category:{topic}&format=json"

.Replace(" ", "\_");

private static string GetPageUrl(string title, string format)

=> $"https://en.wikipedia.org/api/rest\_v1/page/{format}/{title}"

.Replace(" ", "\_");

}

}

## Controllers

*OneTimeController.cs*

namespace ifmo.tpo.lab1.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class OneTimeController : ControllerBase

{

// https://localhost:44395/api/onetime/page?title=Football&format=html

[HttpGet]

[ActionName("Page")]

public async Task<string> GetPageByTitle([FromQuery] string title, [FromQuery] string format)

{

if (format != "html" && format != "title")

{

return "Wrong format.";

}

var page = await Requester.GetPageByTitle(title, format);

return page ?? "No data found.";

}

// https://localhost:44395/api/onetime/random?topic=football

[HttpGet]

[ActionName("Random")]

public async Task<string> GetRandomPage([FromQuery] string topic)

{

var rng = new Random();

var pages = await Requester.GetPages(topic);

if (!pages.Any()) return "No data found.";

var title = pages[rng.Next(pages.Count)];

var page = await Requester.GetPageByTitle(title, "html");

return page;

}

}

}

## Hubs

*BroadcastHub.cs*

namespace ifmo.tpo.lab1.Hubs

{

public class BroadcastHub : Hub

{

public async Task GetSubscription(string action, string topic,

string errors, string interval, string format, string order)

{

var parseResult = await Parser.Parse(action, topic, errors, interval, format, order);

if (parseResult.Success)

{

var dictionary = (Dictionary<SubscriptionOptions, object>)parseResult.Value;

var delayTime = (TimeSpan)dictionary[SubscriptionOptions.Interval];

var parsedFormat = (string)dictionary[SubscriptionOptions.Format];

var parsedOrder = (string)dictionary[SubscriptionOptions.Order];

var parsedAction = (string) dictionary[SubscriptionOptions.Action];

if (parsedAction == "help")

{

var helpMessage = GetHelp();

await Clients.Caller.SendAsync("ReceiveResponse", new List<string>() { helpMessage });

}

var pages = await Requester.GetPages(topic);

pages = SortPages(pages, parsedOrder);

foreach (var pageTitle in pages)

{

var data = Requester.GetPageByTitleLink(pageTitle, parsedFormat);

await Clients.Caller.SendAsync("ReceiveSubscription", data);

await Task.Delay(delayTime);

}

}

else

{

var parsedErrors = Parser.ParseString(errors, AttributeOptions.Errors);

if (parsedErrors.Success & (string)parsedErrors.Value == "detailed")

{

await Clients.Caller.SendAsync("ReceiveResponse", parseResult.Value);

}

else

{

var generalError = GiveGeneralError();

await Clients.Caller.SendAsync("ReceiveResponse", new List<string>() { generalError });

}

}

}

private List<string> SortPages(List<string> pages, string order)

{

switch (order)

{

case "alphabet":

pages.Sort();

break;

case "random":

var rng = new Random();

int n = pages.Count;

while (n > 1)

{

n--;

int k = rng.Next(n + 1);

var value = pages[k];

pages[k] = pages[n];

pages[n] = value;

}

break;

default:

break;

}

return pages;

}

private string GetHelp()

=> "Help";

}

}

## Models

*Option.cs*

namespace ifmo.tpo.lab1.Models

{

public class Option

{

public string Default { get; }

public List<string> Values { get; }

public string AttributeName { get; }

public Option(string name, List<string> values, string def)

{

AttributeName = name;

Values = values;

Default = def;

}

public Option(List<string> values)

{

Values = values;

Default = null;

}

}

}

*PageLink.cs*

namespace ifmo.tpo.lab1.Models

{

public class PageLink

{

public string Title;

public string Url;

}

}

*ResponseJson.cs*

namespace ifmo.tpo.lab1.Models

{

public class ResponseJson

{

public Query Query;

}

public class Query

{

public List<CategoryMember> CategoryMembers;

}

public class CategoryMember

{

public int Pageid;

public int Ns;

public string Title;

}

}

*Result.cs*

namespace ifmo.tpo.lab1.Models

{

public class Result

{

public bool Success { get; }

public object Value { get; }

public Result()

{

Success = default;

Value = null;

}

public Result(bool success)

{

Success = success;

Value = null;

}

public Result(bool success, object value)

{

Success = success;

Value = value;

}

}

}

## Pages

*Index.cshtml*

@page

<div class="container">

<div class="row">&nbsp;</div>

<div class="row">

<div class="col-2">Action</div>

<div class="col-4"><input type="text" id="actionInput" /></div>

</div>

<div class="row">

<div class="col-2">Topic</div>

<div class="col-4"><input type="text" id="topicInput" /></div>

</div>

<div class="row">

<div class="col-2">Errors</div>

<div class="col-4"><input type="text" id="errorsInput" /></div>

</div>

<div class="row">

<div class="col-2">Inteval</div>

<div class="col-4"><input type="text" id="intervalInput" /></div>

</div>

<div class="row">

<div class="col-2">Format</div>

<div class="col-4"><input type="text" id="formatInput" /></div>

</div>

<div class="row">

<div class="col-2">Order</div>

<div class="col-4"><input type="text" id="orderInput" /></div>

</div>

<div class="row">&nbsp;</div>

<div class="row">

<div class="col-6">

<input type="button" id="sendButton" value="Send Message" />

</div>

</div>

</div>

<div class="row">

<div class="col-12">

<hr />

</div>

</div>

<div class="row">

<div class="col-12">

<div id="messages"> </div>

</div>

</div>

<div class="row">

<div class="alert alert-danger">

<ul class ="list-group" id="errorsList" hidden="true"></ul>

</div>

</div>

<script src="~/js/signalr/dist/browser/signalr.js"></script>

<script src="~/js/chat.js"></script>

## Settings

*AttributeOptions.cs*

namespace ifmo.tpo.lab1.Settings

{

public class AttributeOptions

{

public static readonly Option Action = new Option("Action", new List<string>() { "query", "help" }, "query");

public static readonly Option Errors = new Option("Errors", new List<string>() { "default", "detailed" }, "default");

public static readonly Option Format = new Option("Format", new List<string>() { "html", "title" }, "html");

public static readonly Option Order = new Option("Order", new List<string>() { "alphabet", "random", "date" }, "alphabet");

}

}

*Errors.cs*

namespace ifmo.tpo.lab1.Settings

{

public static class Errors

{

public static string GiveUnsupportedTypeError(string attr)

=> $"Attribute with the '{attr}' name has unsupported type.";

public static string GiveWrongTypeError(string attr, string type)

=> $"Attribute with the '{attr}' name should has '{type} type.";

public static string GiveNoDataFoundError(string attr, string value)

=> $"No data found by given value ('{value}') of the attribute named '{attr}'.";

public static string GiveAttributeRequiredError(string attr)

=> $"Attribute with the '{attr}' name required.";

public static string GiveWrongOptionError(string attr)

=> $"Value given to attribute '{attr}' is unexpected.";

public static string GiveNegativeIntervalError()

=> $"Value given to attribute 'Interval' can not be negative.";

public static string GiveZeroIntervalError()

=> $"Value given to attribute 'Interval' can not be zero.";

public static string GiveGeneralError()

=> $"An error occured";

}

}

*SubscriptionOptions.cs*

namespace ifmo.tpo.lab1.Settings

{

public enum SubscriptionOptions

{

Action,

Topic,

Errors,

Interval,

Format,

Order

}

}

## wwwroot/js

*chat.js*

"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/broadcast").build();

document.getElementById("sendButton").disabled = true;

connection.on("ReceiveResponse", function (message) {

document.getElementById("messages").innerHTML = "";

document.getElementById("errorsList").hidden = false;

document.getElementById("errorsList").innerHTML = "";

for (var i = 0; i < message.length; i++) {

var msg = message[i].replace(/&/g, "&amp;").replace(/</g, "&lt;").replace(/>/g, "&gt;");

var li = document.createElement("li");

li.textContent = msg;

document.getElementById("errorsList").appendChild(li);

}

});

connection.on("ReceiveSubscription", function (data) {

document.getElementById("errorsList").hidden = true;

document.getElementById("errorsList").innerHTML = "";

document.getElementById("messages").innerHTML += "<p><a href=\"" + data + "\">" + data +"</a></p>"

});

connection.start().then(function () {

document.getElementById("sendButton").disabled = false;

}).catch(function (err) {

return console.error(err.toString());

});

document.getElementById("sendButton").addEventListener("click", function (event) {

var action = document.getElementById("actionInput").value == "" ? null : document.getElementById("actionInput").value;

var topic = document.getElementById("topicInput").value == "" ? null : document.getElementById("topicInput").value;

var errors = document.getElementById("errorsInput").value == "" ? null : document.getElementById("errorsInput").value;

var interval = document.getElementById("intervalInput").value == "" ? null : document.getElementById("intervalInput").value;

var format = document.getElementById("formatInput").value == "" ? null : document.getElementById("formatInput").value;

var order = document.getElementById("orderInput").value == "" ? null : document.getElementById("orderInput").value;

connection.invoke("GetSubscription", action, topic, errors, interval, format, order);

event.preventDefault();

});

# Листинг тестов

## OneTimeControllerTests

*GetPageByTitleTests.cs*

namespace ifmo.tpo.lab1.Tests.OneTimeControllerTests

{

[TestFixture]

class GetPageByTitleTests

{

[Test]

public async Task WrongFormatTest()

{

var title = "football";

var format = "xml";

var oneTimeController = new OneTimeController();

var result = await oneTimeController.GetPageByTitle(title, format);

Assert.AreEqual("Wrong format.", result);

}

[Test]

public async Task WrongTitleTest()

{

var title = "adgsdfkgh";

var format = "html";

var oneTimeController = new OneTimeController();

var result = await oneTimeController.GetPageByTitle(title, format);

Assert.AreEqual("No data found.", result);

}

[Test]

public async Task CorrectAttributesTest()

{

var title = "Football";

var format = "html";

var oneTimeController = new OneTimeController();

var result = await oneTimeController.GetPageByTitle(title, format);

Assert.IsTrue("No data found." != result && "Wrong format." != result);

}

}

}

*GetRandomPageTests.cs*

namespace ifmo.tpo.lab1.Tests.OneTimeControllerTests

{

[TestFixture]

class GetRandomPageTests

{

[Test]

public async Task WrongTopicTest()

{

var topic = "gasdfhs";

var oneTimeController = new OneTimeController();

var result = await oneTimeController.GetRandomPage(topic);

Assert.AreEqual("No data found.", result);

}

[Test]

public async Task CorrectAttributes()

{

var topic = "Physics";

var oneTimeController = new OneTimeController();

var result = await oneTimeController.GetRandomPage(topic);

Assert.IsTrue("No data found." != result);

}

}

}

## ParserTests

*ParseInterval.cs*

namespace ifmo.tpo.lab1.Tests.ParserTests

{

[TestFixture]

class ParseIntervalTests

{

[Test]

public void NullIntervalTest()

{

string interval = null;

var parseResult = Parser.ParseInterval(interval);

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(TimeSpan.FromDays(1),

(TimeSpan)parseResult.Value);

}

[Test]

public void WrongTypeTest1()

{

var interval = "5 seconds";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongTypeError("Interval", "Integer"),

(string)parseResult.Value);

}

[Test]

public void WrongTypeTest2()

{

var interval = TimeSpan.FromSeconds(5).ToString();

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongTypeError("Interval", "Integer"),

(string)parseResult.Value);

}

[Test]

public void WrongTypeTest3()

{

var interval = "5,5";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongTypeError("Interval", "Integer"),

(string)parseResult.Value);

}

[Test]

public void NegativeIntervalTest1()

{

var interval = "-5";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveNegativeIntervalError(),

(string)parseResult.Value);

}

[Test]

public void NegativeIntervalTest2()

{

var interval = "-1";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveNegativeIntervalError(),

(string)parseResult.Value);

}

[Test]

public void ZeroIntervalTest1()

{

var interval = "0";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveZeroIntervalError(),

(string)parseResult.Value);

}

[Test]

public void ZeroIntervalTest2()

{

var interval = "-0";

var parseResult = Parser.ParseInterval(interval);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveZeroIntervalError(),

(string)parseResult.Value);

}

[Test]

public void PositiveIntervalTest1()

{

var interval = "5";

var parseResult = Parser.ParseInterval(interval);

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(TimeSpan.FromSeconds(5), (TimeSpan)parseResult.Value);

}

[Test]

public void PositiveIntervalTest2()

{

var interval = "1";

var parseResult = Parser.ParseInterval(interval);

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(TimeSpan.FromSeconds(1), (TimeSpan)parseResult.Value);

}

}

}

*ParseString.cs*

namespace ifmo.tpo.lab1.Tests.ParserTests

{

[TestFixture]

class ParseStringTests

{

[Test]

public void NullableStringTest1()

{

string str = null;

var isNullable = true;

var parseResult = Parser.ParseString(str, Errors, isNullable);

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(Errors.Default,

(string)parseResult.Value);

}

[Test]

public void NullableStringTest2()

{

string str = null;

var parseResult = Parser.ParseString(str, Format);

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(Format.Default,

(string)parseResult.Value);

}

[Test]

public void NotNullableStringTest1()

{

string str = null;

var isNullable = false;

var parseResult = Parser.ParseString(str, Format, isNullable);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveAttributeRequiredError(Format.AttributeName),

(string)parseResult.Value);

}

[Test]

public void NotNullableStringTest2()

{

string str = null;

var isNullable = false;

var parseResult = Parser.ParseString(str, Errors, isNullable);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveAttributeRequiredError(Errors.AttributeName),

(string)parseResult.Value);

}

[Test]

public void WrongOptionStringTest1()

{

var str = "HTML";

var parseResult = Parser.ParseString(str, Format);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongOptionError(Format.AttributeName),

(string)parseResult.Value);

}

[Test]

public void WrongOptionStringTest2()

{

var str = "html";

var parseResult = Parser.ParseString(str, Errors);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongOptionError(Errors.AttributeName),

(string)parseResult.Value);

}

[Test]

public void CorrectOptionStringTest1()

{

var str = "html";

var parseResult = Parser.ParseString(str, Format);

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(str,

(string) parseResult.Value);

}

[Test]

public void CorrectOptionStringTest2()

{

var str = "detailed";

var parseResult = Parser.ParseString(str, Errors);

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(str,

(string)parseResult.Value);

}

}

}

*ParseTests.cs*

namespace ifmo.tpo.lab1.Tests.ParserTests

{

[TestFixture]

public class ParseTests

{

[Test]

public async Task WrongActionTest()

{

var correctAttributes = new CorrectAttributes();

var parseResult = await Parser.Parse("wrong", correctAttributes.Topic, correctAttributes.Errors,

correctAttributes.Interval, correctAttributes.Format, correctAttributes.Order);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongOptionError("Action"), ((List<string>)parseResult.Value).First());

}

[Test]

public async Task WrongTopicTest()

{

var correctAttributes = new CorrectAttributes();

var topic = "abcabcabc";

var parseResult = await Parser.Parse(correctAttributes.Action, topic, correctAttributes.Errors,

correctAttributes.Interval, correctAttributes.Format, correctAttributes.Order);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveNoDataFoundError("Topic", topic), ((List<string>)parseResult.Value).First());

}

[Test]

public async Task WrongErrorsTest()

{

var correctAttributes = new CorrectAttributes();

var parseResult = await Parser.Parse(correctAttributes.Action, correctAttributes.Topic, "wrong",

correctAttributes.Interval, correctAttributes.Format, correctAttributes.Order);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongOptionError("Errors"), ((List<string>)parseResult.Value).First());

}

[Test]

public async Task WrongIntervalTest()

{

var correctAttributes = new CorrectAttributes();

var parseResult = await Parser.Parse(correctAttributes.Action, correctAttributes.Topic, correctAttributes.Errors,

"wrong", correctAttributes.Format, correctAttributes.Order);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongTypeError("Interval", "Integer"), ((List<string>)parseResult.Value).First());

}

[Test]

public async Task WrongFormatTest()

{

var correctAttributes = new CorrectAttributes();

var parseResult = await Parser.Parse(correctAttributes.Action, correctAttributes.Topic, correctAttributes.Errors,

correctAttributes.Interval, "wrong", correctAttributes.Order);

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongOptionError("Format"), ((List<string>)parseResult.Value).First());

}

[Test]

public async Task WrongOrderTest()

{

var correctAttributes = new CorrectAttributes();

var parseResult = await Parser.Parse(correctAttributes.Action, correctAttributes.Topic, correctAttributes.Errors,

correctAttributes.Interval, correctAttributes.Format, "wrong");

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveWrongOptionError("Order"), ((List<string>)parseResult.Value).First());

}

}

public struct CorrectAttributes

{

public string Action { get => "query"; }

public string Topic{ get => "Football"; }

public string Errors { get => "detailed"; }

public string Interval { get => "5"; }

public string Format { get => "html"; }

public string Order { get => "alphabet"; }

}

}

*ParseTopicTests.cs*

namespace ifmo.tpo.lab1.Tests.ParserTests

{

[TestFixture]

class ParseTopicTests

{

[Test]

public async Task NullTopicWithActionHelpTest()

{

string topic = null;

var parseResult = await Parser.ParseTopic(topic, "help");

Assert.IsTrue(parseResult.Success);

}

[Test]

public async Task NullTopicTest()

{

string topic = null;

var parseResult = await Parser.ParseTopic(topic, "query");

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveAttributeRequiredError("Topic"),

(string)parseResult.Value);

}

[Test]

public async Task TopicDoesNotExistTest1()

{

string topic = "abcabcabc";

var parseResult = await Parser.ParseTopic(topic, "query");

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveNoDataFoundError("Topic", topic),

(string)parseResult.Value);

}

[Test]

public async Task TopicDoesNotExistTest2()

{

string topic = "Test";

var parseResult = await Parser.ParseTopic(topic, "query");

Assert.IsFalse(parseResult.Success);

Assert.AreEqual(GiveNoDataFoundError("Topic", topic),

(string)parseResult.Value);

}

[Test]

public async Task ExistingTopicTest1()

{

string topic = "Football";

var parseResult = await Parser.ParseTopic(topic, "query");

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(topic, (string)parseResult.Value);

}

[Test]

public async Task ExistingTopicTest2()

{

string topic = "Physics";

var parseResult = await Parser.ParseTopic(topic, "query");

Assert.IsTrue(parseResult.Success);

Assert.AreEqual(topic, (string)parseResult.Value);

}

}

}

## RequesterTests

*IsTopicExistsTests.cs*

namespace ifmo.tpo.lab1.Tests.RequesterTests

{

[TestFixture]

public class IsTopicExistsTests

{

[Test]

public async Task DoesNotExistTest1()

{

var topic = "asgasdg";

var exists = await Requester.IsTopicExists(topic);

Assert.IsFalse(exists);

}

[Test]

public async Task DoesNotExistTest2()

{

var topic = "Test";

var exists = await Requester.IsTopicExists(topic);

Assert.IsFalse(exists);

}

[Test]

public async Task ExistsTest1()

{

var topic = "Football";

var exists = await Requester.IsTopicExists(topic);

Assert.IsTrue(exists);

}

[Test]

public async Task ExistsTest2()

{

var topic = "Physics";

var exists = await Requester.IsTopicExists(topic);

Assert.IsTrue(exists);

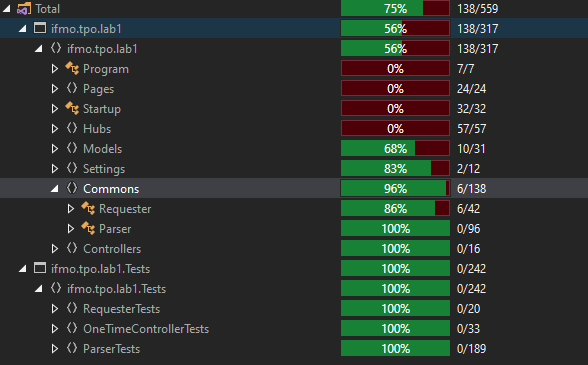
}

}

}

# Покрытие тестами

*Тестовое покрытие приложения*



Несмотря на то, что многие элементы приложения не покрыты тестами, я считаю, что покрыл весь код, который нужно было покрыть. Так, например основная логика моего приложения находится в разделе Commons с тестовым покрытием 96%.

# Выводы

Несмотря на то, что я покрыл почти весь значимый код, я считаю, что тестирование данного приложения далеко от завершения. Причина этому это то, что я писал тесты только для заполнения тестового покрытия, но 100% покрытие кода (coverage tests) не гарантирует полного покрытия тестовых случаев. Да, мною были покрыты все условные ответвления по отдельности, но все возможные вариации условных веток – нет.