Public API Guide

Test IT для интеграции с автотестами предоставляет Public API, который позволяет взаимодействовать с такими сущностями как TestRun, Configuration, AutoTest, TestPoint, TestResult. В качестве ключа авторизации Public API использует связку Secret Key и логина пользователя; данную связку необходимо использовать для каждого запроса к Public API. Secret Key уникален для каждого пользователя, пользователь может обновлять его, но не может иметь несколько Secret Key. Пользователь может генерировать свой Secret Key через личный кабинет. Логин пользователя так же указан в личном кабинете.

Для интеграции с автотестами необходимо реализовать программу-listener, которая будет проверять наличие автотестов на запуск, запускать тесты и проставлять результаты.

Так же необходимо реализовать программу, которая будет связывать автотесты с test case'ами в Test IT.

Данная инструкция построена на основе языка программирования C#, но схема в общем виде применима для любого другого языка программирования.

Для интеграции test case с автотестами необходимо реализовать программу для связывания или выполнить cURL запрос.

cURL запрос:

POST /api/Public/AddAutoTestToTestCase/{testCaseId}

Где:

`{testCaseId}` - это идентификатор теста в Test IT

Тело запроса:

```
{  "autoTestExternalId": "string"  }
```

Пример cURL запроса в общем виде:

```
curl -X POST "http://192.168.88.140/api/Public/AddAutoTestToTestCase/{testCaseGlobalId}" -H "accept: application/json" -H "SecretKeyBase64: {SecretKeyBase64}" -H "UserName: {UserName}" -H "Content-Type: application/json-patch+json" -d "{ \"autoTestExternalId\": \"{autoTestExternalId}\"}"
```

Пример cURL запроса:

```
        curl -X POST "http://192.168.88.140/api/Public/AddAutoTestToTestCase/3834" -H "accept:
application/json" -H "SecretKeyBase64: ZndlUXkyNXFCbkJRUmJvUw==" -H "UserName: admin" -H "Content-Type:
application/json-patch+json" -d "{ \"autoTestExternalId\": \"autoTestExternalId\"}"
```

Программный код на C# для связывания автотеста с test case:

```
public async Task AddAutoTestToTestCase(int globalId, string testName)
{
    string url = "api/Public/AddAutoTestToTestCase/" + globalId;
    StringContent content = new JsonContent(new PublicAutoTestModel { AutoTestExternalId = testName });

    HttpResponseMessage responseMessage = await client.PostAsync(url, content);
}
```

Где:

testName - название автотеста или другой его идентификатор по которому производится соотношение test case с автотестом,

testCaseGlobalId - глобальный идентификатор test case. Его можно получить из редактора test case. Пишется после '#'

JsonContent:

```
public class JsonContent : StringContent
{
    public JsonContent(object obj) : base(JsonConvert.SerializeObject(obj), Encoding.UTF8,
MediaTypeNames.Application.Json) { }

}
```

Для интеграции test case с автотестами необходимо реализовать Listener, который представляет следующую логику работы:

```
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using TestIt.WebApi.Models;

namespace TestIT.Listener
{
    class Program
    {
        private const int DeleyInMilliseconds = 1000;

        public static readonly string Domain = "http://192.168.88.51";
        public static readonly string SecretKey = "QzM3ejdzUUZ2NTdNc0RTRA==";
        public static readonly string UserName = "auto";
        public static readonly string PathToTests = @"D:\autotests\testit";

        static async Task Main(string[] args)
        {
            // Клиент для отправки запросов
            PublicApiClient client = new PublicApiClient(Domain, SecretKey, UserName);
            // Экземпляр класса для запуска автотестов
            TestRunner runner = new TestRunner();
```

```
            // Экземпляр класса для получения результатов автотестов
            AutoTestResultsProvider resultsProvider = new AutoTestResultsProvider();
            // Бесконечный цикл для получения и обработки запроса на запуск автотестов
            while (true)
            {
                // Получение всех тестовых запусков
                IEnumerable<PublicTestRunModel> testRuns = await client.GetAllActiveTestRuns();
                // Получение первого необработанного тестового запуска
                PublicTestRunModel run = testRuns.FirstOrDefault(testRun => testRun.Status ==
TestRunStates.NotStarted);
                // Если тестовый запуск был найден, обрабатываем его
                if (run != null)
                {
                    // Стартуем тестовый запуск, переводим тестовый запуск в статус  In Progress
                    await client.StartTestRun(run.TestRunId.ToString());
                    // Выбираем название тестов (их идеинтификаторы)
                    List<string> testsName = run.AutoTests.Select(at => at.AutotestExternalId).ToList();
                    // Запускам выбранные автотесты
                    runner.RunSelectedTests(PathToTests, testsName);
                    // Получаем результаты автотестов
                    IEnumerable<TestResult> testResults = resultsProvider.GetLastResults(PathToTests);
                    // Проставляем полученные результаты
                    await client.SetAutoTestResult(run, testResults);
                }
                // Задержка
                Thread.Sleep(DeleyInMilliseconds);
            }
        }
    }
}
```

PublicApiClient - HttpClient для отправки запросов

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using TestIt.WebApi.Models;
using static System.Net.Mime.MediaTypeNames;

namespace TestIT.Listener
{
    public class PublicApiClient : IDisposable
    {
        private readonly HttpClient client;

        public PublicApiClient(string host, string secretKey, string userName)
        {
            client = new HttpClient();
            client.DefaultRequestHeaders.Add("secretKeyBase64", secretKey);
            client.DefaultRequestHeaders.Add("userName", userName);
            client.BaseAddress = new Uri(host);
        }

        public async Task AddAutoTestToTestCase(int globalId, string testName)
        {
            string url = "api/Public/AddAutoTestToTestCase/" + globalId;
            StringContent content = new JsonContent(new PublicAutoTestModel { AutoTestExternalId =
testName });

            HttpResponseMessage responseMessage = await client.PostAsync(url, content);
        }
```

```
        public async Task StartTestRun(string testRunId)
        {
            string url = $"api/Public/StartTestRun/{testRunId}";

            HttpResponseMessage responseMessage = await client.PostAsync(url, new
JsonContent(string.Empty));
        }

        public async Task<IEnumerable<PublicTestRunModel>> GetAllActiveTestRuns()
        {
            string url = "api/Public/GetTestRuns";
            HttpResponseMessage responseMessage = await client.GetAsync(url);
            string response = await responseMessage.Content.ReadAsStringAsync();

            return JsonConvert.DeserializeObject<IEnumerable<PublicTestRunModel>>(response);
        }

        public async Task<IEnumerable<PublicTestRunModel>> GetActiveTestRunsByProject(int globalId)
        {
            string url = "api/Public/GetTestRuns/" + globalId;
            HttpResponseMessage responseMessage = await client.GetAsync(url);
            string response = await responseMessage.Content.ReadAsStringAsync();

            return JsonConvert.DeserializeObject<IEnumerable<PublicTestRunModel>>(response);
        }

        public async Task SetAutoTestResult(PublicTestRunModel publicTestRunModel, IEnumerable<TestResult>
testResults)
        {
            string url = "api/Public/SetAutoTestResult";

            foreach (ConfigurationModel config in publicTestRunModel.Configurations)
            {
                foreach (AutoTestModel autoTest in publicTestRunModel.AutoTests)
                {
                    foreach (TestResult testResult in testResults)
                    {
                        if (autoTest.AutotestExternalId == testResult.Name)
                        {
                            PublicTestPointPostModel point = new PublicTestPointPostModel()
                            {
                                TestRunId = publicTestRunModel.TestRunId,
                                ConfigurationGlobalId = config.GlobalId,
                                TestPlanGlobalId = publicTestRunModel.TestPlanGlobalId,
                                Status = "Ready",
                                Outcome = testResult.IsSuccess ? TestResultOutcomes.Passed :
TestResultOutcomes.NotPassed,
                                Message = testResult.Messenge,
                                StackTrace = testResult.StackTrace,
                                AutoTestExternalId = testResult.Name,
                                Name = testResult.Name
                            };

                            string jsonContent = JsonConvert.SerializeObject(point);
                            StringContent content = new StringContent(jsonContent, Encoding.UTF8,
Application.Json);
                            HttpResponseMessage responseMessage = await client.PostAsync(url, content);
                        }
                    }
                }
            }
        }

        public void Dispose()
        {
            client?.Dispose();
        }
    }
}
```

TestRunner - Класс для запуска автотестов, формирует и выполняет команду на выполнение автотестов:

```
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;

namespace TestIT.Listener
{
    public class TestRunner
    {
        public void RunSelectedTests(string path, List<string> testsName)
        {
            string filter = string.Empty;

            if (testsName.Count == 1)
            {
                filter = testsName[0];
            }
            else
            {
                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.Append("\"");
                foreach (var testName in testsName)
                {
                    stringBuilder.Append($"{testName}|");
                }
                stringBuilder.Remove(stringBuilder.Length - 1, 1);
                stringBuilder.Append("\"");

                filter = stringBuilder.ToString();
            }

            string arguments = $"test {path} --filter {filter} --logger:trx -r ./build/reports/";

            Process process = Process.Start($"dotnet", arguments);

            process.WaitForExit();
        }
    }
}
```

AutoTestResultsProvider - Получает результаты автотестов:

```
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace TestIT.Listener
{
    public class AutoTestResultsProvider
    {
        private readonly string pathToResult = @"\TestIT.Tests\build\reports\";

        public IEnumerable<TestResult> GetLastResults(string path)
        {
            DirectoryInfo directory = new DirectoryInfo(path + pathToResult);

            FileInfo file = directory.GetFiles()
                                .OrderByDescending(f => f.LastWriteTime)
                                .FirstOrDefault();
```

```
            using (Stream stream = file.OpenRead())
            {
                return TestResultConverter.ConvertFromXml(stream);
            }
        }
    }
}
```

TestResultConverter - конвертирует полученный результат от атотестов

```
using System.Collections.Generic;
using System.IO;
using System.Xml;

namespace TestIT.Listener
{
    public static class TestResultConverter
    {
        public static IEnumerable<TestResult> ConvertFromXml(Stream stream)
        {
            List<TestResult> testResults = new List<TestResult>();
            XmlDocument xmlDocument = new XmlDocument();
            xmlDocument.Load(stream);
            XmlElement xRoot = xmlDocument.DocumentElement;
            foreach (XmlNode results in xRoot)
            {
                if (results.Name != "Results")
                    continue;
                foreach (XmlNode unitTestResult in results)
                {
                    if (unitTestResult.Name != "UnitTestResult")
                        continue;
                    TestResult testResult = new TestResult();
                    XmlNode outcomeNode = unitTestResult.Attributes.GetNamedItem("outcome");
                    if (outcomeNode != null)
                        testResult.IsSuccess = outcomeNode.Value == TestResultOutcomes.Passed;
                    XmlNode testNameNode = unitTestResult.Attributes.GetNamedItem("testName");
                    if (testNameNode != null)
                        testResult.Name = testNameNode.Value;
                    if (unitTestResult.ChildNodes[0] != null && unitTestResult.ChildNodes[0].Name ==
"Output")
                    {
                        XmlNode outputNode = unitTestResult.ChildNodes[0];
                        if (outputNode.ChildNodes[0].Name == "ErrorInfo")
                        {
                            XmlNode errorInfoNode = outputNode.ChildNodes[0];
                            foreach (XmlNode info in errorInfoNode)
                            {
                                switch (info.Name)
                                {
                                    case "Message":
                                        testResult.Messenge = info.InnerText;
                                        break;
                                    case "StackTrace":
                                        testResult.StackTrace = info.InnerText;
                                        break;
                                }
                            }
                        }
                    }
                    testResults.Add(testResult);
                }
            }
            return testResults;
        }
    }
}
```