**Проект №7 — Back-end тестирование №2.**

Привет, участник Школы 21! Снова групповой проект и снова Back-end тестирование! Тему тестирования API мы продолжим изучать и в этом проекте, подробно поговорим про балансировщики, окружение, инструменты и этапы тестирования API, составим первый тест-план и немного окунёмся в полуавтоматизированное тестирование. Поехали!😄

**Instructions**

Напоминаем, что все отчёты по результатам выполнения заданий тебе нужно оформлять в файлах с расширением .md. Если они уже созданы, то пересоздавать или удалять их не нужно (просто отредактируй этот файл). Все созданные отчёты и файлы тебе нужно будет загрузить в папку src/ в корне проекта (обязательно в ветку *develop*).

**Contents**

1. [Chapter I](https://edu.21-school.ru/project/62670/task#chapter-i)  
   1.1. [Общая инструкция](https://edu.21-school.ru/project/62670/task#%D0%BE%D0%B1%D1%89%D0%B0%D1%8F-%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F)
2. [Chapter II](https://edu.21-school.ru/project/62670/task#chapter-ii)  
   2.1. [Балансировка нагрузки](https://edu.21-school.ru/project/62670/task#%D0%B1%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0-%D0%BD%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B8)  
   2.2. [Тестовое окружение](https://edu.21-school.ru/project/62670/task#%D1%82%D0%B5%D1%81%D1%82%D0%BE%D0%B2%D0%BE%D0%B5-%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)  
   2.3. [Задание №1](https://edu.21-school.ru/project/62670/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-1-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B-%D0%B1%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8-%D0%BD%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B8)
3. [Chapter III](https://edu.21-school.ru/project/62670/task#chapter-iii)  
   3.1. [Проверка спецификации API](https://edu.21-school.ru/project/62670/task#%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D0%BA%D0%B0-%D1%81%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-api)  
   3.2. [Этапы тестирования API](https://edu.21-school.ru/project/62670/task#%D1%8D%D1%82%D0%B0%D0%BF%D1%8B-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-api)  
   3.3. [Категории тестовых сценариев](https://edu.21-school.ru/project/62670/task#%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8-%D1%82%D0%B5%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D1%85-%D1%81%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%B8%D0%B5%D0%B2)  
   3.4. [Тест-план](https://edu.21-school.ru/project/62670/task#%D1%82%D0%B5%D1%81%D1%82-%D0%BF%D0%BB%D0%B0%D0%BD)  
   3.5. [Задание №2](https://edu.21-school.ru/project/62670/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-2-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%82%D0%B5%D1%81%D1%82-%D0%BF%D0%BB%D0%B0%D0%BD%D0%B0)
4. [Chapter IV](https://edu.21-school.ru/project/62670/task#chapter-iv)  
   4.1. [Инструменты тестирования API](https://edu.21-school.ru/project/62670/task#%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-api)  
   4.2. [Коллекции и запросы](https://edu.21-school.ru/project/62670/task#%D0%BA%D0%BE%D0%BB%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B8-%D0%B8-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D1%8B)  
   4.3. [Задание №3](https://edu.21-school.ru/project/62670/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-3-%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D0%BE-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-%D1%81-postman)  
   4.4. [Основные вкладки конструктора запросов](https://edu.21-school.ru/project/62670/task#%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-%D0%B2%D0%BA%D0%BB%D0%B0%D0%B4%D0%BA%D0%B8-%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D0%BE%D1%80%D0%B0-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2)  
   4.5. [Переменные окружения](https://edu.21-school.ru/project/62670/task#%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5-%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)  
   4.6. [Pre-request script и tests](https://edu.21-school.ru/project/62670/task" \l "pre-request-script-%D0%B8-tests)  
   4.7. [Задание №4](https://edu.21-school.ru/project/62670/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-4-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-api-%D1%81-postman)

**Chapter I**

**Общая инструкция**

Методология Школы 21 может быть не похожа на тот образовательный опыт, который случался с тобой ранее. Её отличает высокий уровень автономии: у тебя есть задача, ты должен её выполнить. По большей части тебе нужно будет самому добывать знания для её решения. Второй важный момент — это peer-to-peer обучение. В образовательном процессе нет менторов и экспертов, перед которыми ты защищаешь свой результат. Ты это делаешь перед таким же учащимися, как и ты сам. У них есть чек-лист, который поможет им качественно выполнить приемку вашей работы.

Роль Школы 21 заключается в том, чтобы обеспечить через последовательность заданий и оптимальный уровень поддержки такую траекторию обучения, при которой ты не только освоишь hard skills, но и научишься самообучаться.

* Не доверяй слухам и предположениям о том, как должно быть оформлено ваше решение. Этот документ является единственным источником, к которому стоит обращаться по большинству вопросов;
* твое решение будет оцениваться другими учащимися;
* подлежат оцениванию только те файлы, которые ты выложил в GIT (ветка develop, папка src);
* в твоей папке не должно быть лишних файлов — только те, что были указаны в задании;
* не забывай, что у вас есть доступ к интернету и поисковым системам;
* обсуждение заданий можно вести и в Rocket.Chat;
* будь внимателен к примерам, указанным в этом документе — они могут иметь важные детали, которые не были оговорены другим способом;
* и да пребудет с тобой Сила!

**Chapter II**

**Балансировка нагрузки**

Представьте, что у вас разрабатывается большое веб-приложение с множеством сервисов. Вы знаете, что одного сервера для обработки запросов большого количества пользователей будет недостаточно. Как быть в этой ситуации? Например, можно подключить ещё больше серверов! В таком случае как нам распределить между ними нагрузку? В этом может помочь балансировка нагрузки.

**Балансировка нагрузки** — это метод равномерного распределения сетевого трафика по пулу ресурсов, поддерживающих приложение. Современные приложения должны обрабатывать миллионы пользователей одновременно, а также быстро и надежно возвращать каждому пользователю правильный текст, видео, изображения и другие данные. Для обработки таких значительных объемов трафика в большинстве приложений используется множество серверов ресурсов с дублированием данных между ними.

**Балансировщик нагрузки** — это устройство, которое находится между пользователем и группой серверов и действует как невидимый посредник, обеспечивая одинаковое использование всех серверов ресурсов.

Балансировщики нагрузки бывают двух типов: аппаратные и программные.

**Аппаратный балансировщик нагрузки** — это аппаратное устройство, которое может безопасно обрабатывать и перенаправлять гигабайты трафика на сотни различных серверов.

**Программные балансировщики нагрузки** — это приложения, выполняющие все функции балансировки нагрузки. Вы можете установить их на любом сервере или получить к ним доступ как к полностью управляемому стороннему сервису.

**Тестовое окружение**

Интуитивно мы понимаем, что тестовое окружение — это среда, место или машины, которые используются в ходе тестирования. Тестовых окружений, как правило, бывает много, ведь большие системы могут быть развернуты на нескольких машинах (это часто называют *кластером*). Под каждый проект должно быть своё тестовое окружение: свой *тестовый стенд*, свой способ проведения тестирования, браузер определённой версии и так далее.

Говоря про тестовые окружения, стоит рассказать про то, что такое прод (продакшн), тестовый стенд и локальное окружение.

**Боевая среда, продакшн, прод** — это реальная сеть машин, в которой работают настоящие пользователи приложения; та версия приложения, которая выложена онлайн или установлена клиенту. Продакшн не является тестовым окружением.

**Превью среда, препрод** — это среда, идентичная или максимально приближенная к той, которая будет использоваться в боевой среде: те же данные, то же аппаратно-программное окружение, та же производительность. Она используется, чтобы сделать финальную проверку ПО в условиях, максимально приближенным к "боевым".

**Тестовый стенд** — это совокупность аппаратуры, программ и определенных настроек для тестирования продукта. Часто это конкретная машина, на которой разворачивается приложение и которая используется в целях тестирования.

**Локальное окружение** — это локальная площадка, где тестируется код приложения, программного продукта. Например, разработчик может локально для себя развернуть свой код (таким образом, создав локальное окружение), проверить его и отправить на проверку тестировщикам, развернув тестовый стенд.

**Задание №1. Алгоритмы балансировки нагрузки**

Изучите существующие алгоритмы балансировки нагрузки. Создайте файл exercise1.md и опишите изученные алгоритмы, указав их особенности и недостатки.

**Chapter III**

**Проверка спецификации API**

API — это, по сути, соглашение между клиентом и сервером или между двумя приложениями. Перед тем, как начать любое тестирование функциональности, важно убедиться в правильности соглашения. Для этого нужно проверить спецификацию (или само соглашение службы, например, интерфейс Swagger) и убедиться, что:

* эндпоинты правильно именованы;
* ресурсы и их типы правильно отражают объектную модель;
* нет отсутствующей или дублирующей функциональности;
* отношения между ресурсами правильно отражаются в API.

**Этапы тестирования API**

Условно процесс тестирования API можно разделить на следующие этапы:

1. Проверка корректности кода состояния HTTP. Например, создание ресурса должно возвращать 201 CREATED, а запрещённые запросы должны возвращать 403 FORBIDDEN и т. д.
2. Проверка полезной нагрузки ответа. Проверка правильности тела JSON, имён, типов и значений полей ответа, в том числе в ответах на ошибочные запросы.
3. Проверка заголовков ответа. Заголовки HTTP-сервера влияют как на безопасность, так и на производительность.
4. Проверка правильности состояния приложения. Это необязательно, и применяется в основном, когда пользовательский интерфейс или другой интерфейс можно легко проверить.
5. Проверка базовой работоспособности. Если операция была завершена успешно, но заняла неоправданно много времени, тест вряд ли можно считать пройденным.

**Категории тестовых сценариев**

Тест-кейсы, которые используются во время проверки API, можно разделить на следующие общие группы:

* Основные позитивные тесты (прохождение успешного сценария по умолчанию);
* Расширенное позитивное тестирование с дополнительными параметрами;
* Негативное тестирование с валидными входными данными;
* Негативное тестирование с недопустимыми входными данными;
* Тесты безопасности, авторизации и доступности;
* Деструктивное тестирование – более глубокая форма негативного тестирования, когда мы намеренно пытаемся сломать API, чтобы проверить его надёжность.

**Тест-план**

Тест-план — это артефакт тестирования, описывающий действия, которые будут происходить в процессе тестирования. Тест-план является важной составляющей процесса тестирования. Он содержит в себе всю необходимую информацию, описывающую данный процесс. Иногда он играет формальную роль.

Составление тест-плана включает в себя несколько этапов:

1. Анализ программного продукта. Нужно проанализировать функциональные возможности приложения, изучить все требования к продукту. Попытаться понять пользователей и использовать возможности тестирования продукта с точки зрения пользователя.
2. Определить область действия. Хороший тест-план четко определяет область тестирования и границы. Составление списков "Проверяемые функции" и "Функции, которые проверяться не будут" сделает тест-план более конкретным. Также может потребоваться указать список результатов.
3. Разработать график. Здесь необходимо разделить работу на тестирование и оценить необходимые усилия. Также можно оценить необходимые ресурсы для каждой задачи.
4. Определить роли и обязанности. В хорошем тест-плане четко перечислены роли и обязанности команды тестирования и тимлида. Этим отчасти вы занимались, когда формировали команды и делили имеющиеся задачи.😉

Также часто составляется стратегия тестирования (о которой мы поговорим в следующих проектах), указываются ожидаемые риски и прочее. В итоге получается большой-большой документ. Разработка полноценного тест-плана требует очень много времени и немалого опыта работы в сфере тестирования и управления. Поэтому мы будем составлять его более простую версию в TestIT.

**Задание №2. Составление тест-плана**

В этом проекте вы снова поработаете с [Fake REST API](https://fakerestapi.azurewebsites.net/index.html). Создайте новое рабочее пространство в [TestIT](https://testit.software/), добавьте в него новый проект, который назовите FakeAPI.

Перед созданием тест-плана вам нужно создать оптимальный набор тест-кейсов для проверки всех эндпоинтов в Fake REST API. Не забудьте, что POST- и PUT-запросы, а также некоторые GET-запросы (с указанием {id}) требуют для проверки минимум два случая (чтобы ловить не только 200 статус-код). Сами тест-кейсы, чтобы не запутаться, называйте так: <Секция>: <название>, например, Activities: получение списка активностей. Используйте общие шаги, чтобы несколько раз не переписывать одни и те же условия. Также указывайте приоритет и статус тест-кейса!

*P.S.: обратите внимание, что тестировать нужно API, а не интерфейс Swagger*

После создания всех тест-кейсов и чек-листов (если они потребуются), необходимо создать тест-план. В созданном плане нужно добавить набор тест-кейсов (для удобства выгрузки всего плана в один документ создайте только один набор)

Распределите между собой тест-кейсы так, чтобы каждый смог проверить часть функциональных возможностей API, а потом проведите тестирование!😇

После проверки всех тест-кейсов, выгрузите 2 файла:

* Сам тест-план во вкладке "Планирование" и назовите файл exercise2.plan.xlsx
* Результаты прогона тестов во вкладке "Отчёт" и назовите файл exercise2.results.xlsx (при выгрузке отчёта шаги желательно не включать):

А также создайте файл exercise2.bugs.md, в котором опишите все найденные баги (если API возвращает неверный код или если вы считаете, что метод некорректно ведёт себя в каких-то случаях).

**Chapter IV**

**Инструменты тестирования API**

Тестируя API через Swagger UI, можно столкнуться с различными проблемами. Например, там отсутствуют инструменты, позволяющие как-либо автоматизировать процесс тестирования API, не сохраняется история запросов, не предусмотрена работа с большими объёмами данных (из-за ограничений, которые накладывает сам браузер). Также разработчики не всегда предоставляют документацию API, оформленную в виде Swagger. В таком случае нам нужно проверять API другими средствами, с которыми вам предстоит познакомиться в этой главе.🙂

Есть множество инструментов, с помощью которых можно быстро и удобно протестировать API. Давай остановимся на самых популярных из них!

**SoapUI**

*SoapUI* — это приложение с открытым исходным кодом для тестирования веб-сервисов, а также SOAP и REST API.

Среди *особенностей* SoapUI можно отметить:

* Лёгкое и быстрое создание тестов при помощи технологий перетаскивания объектов «мышкой» (Drag and drop) и метода «указания и щелчка» (Point-and-click);
* Быстрое создание пользовательского кода при помощи Groovy;
* Мощное тестирование на основе данных: данные загружаются из файлов, баз данных и Excel, поэтому они могут создать симуляцию взаимодействия пользователя и API;
* Создание комплексных сценариев и поддержка асинхронного тестирования;
* Повторное использование скриптов: загрузка тестов и сканирование безопасности могут повторно использоваться в случае функционального тестирования всего в несколько шагов.

**Katalon Studio**

**Katalon Studio** — это интегрированная среда для создания и выполнения тестов при работе с API, Web- и мобильными приложениями. Имеет богатый набор инструментов для тестирования и поддерживает множество платформ, включая Windows, Mac OS и Linux. Интегрируя движки от Selenium и Appium со всеми необходимыми компонентами, встроенными ключевыми словами и шаблонами, Katalon Studio предоставляет уникальную среду разработки как для тестировщиков, так и для разработчиков, занимающихся тестированием API и веб-автоматизацией.

**Postman**

**Postman** — это платформа API, позволяющая проектировать, создавать и тестировать свои API. Именно с ним мы познакомимся подробнее, так как он обладает следующими преимуществами:

* Богатый интерфейс делает этот инструмент простым и удобным в использовании;
* Postman может использоваться как для автоматизированного, так и для исследовательского тестирования;
* Postman можно установить и в качестве desktop-приложения для операционных систем Windows, Mac и Linux, и в виде расширения для браузера;
* В нём есть пакеты средств интеграции, такие как поддержка форматов Swagger и RAML;
* Не требует изучения нового языка программирования;
* Позволяет пользователям легко делиться опытом и знаниями с другими членами команды, поскольку в нём можно упаковывать все запросы и ожидаемые ответы и отправлять их коллегам, а также делиться с ними рабочим пространством.

**Коллекции и запросы**

Как мы уже говорили ранее, всё тестирование API в этом курсе будет показано на примере Postman. Начнём его изучение с базовых понятий.🙂

Главными понятиями, которыми оперирует Postman, являются **Collection** на верхнем уровне и **Request** на нижнем. Вся работа начинается с коллекции и сводится к описанию API с помощью запросов. Рассмотрим это подробнее.

**Collection** (коллекция) — отправная точка для нового API. Можно рассматривать коллекцию как файл проекта. Коллекция объединяет в себе все связанные запросы. Обычно все методы одного API описываются в одной коллекции, но если вы захотите сделать по-другому, то никаких ограничений на это нет.

**Folder** (папка) — используется для объединения запросов в одну группу внутри коллекции. К примеру, вы можете создать папку, где будут находиться запросы, проверяющие первую версию API, или использовать папки для того, чтобы разграничить запросы по функционалу, который они проверяют.

**Request** (запрос) — основная составляющая коллекции. Запрос создаётся в *конструкторе*. Конструктор запросов — это главное пространство, с которым вы будете работать.

Стоит также сказать, что Postman умеет выполнять запросы с помощью всех стандартных HTTP методов, поддерживает множество типов передаваемых данных (JSON и XML форматы, файлы, GraphQL и другие), позволяет настраивать авторизацию, выполнять небольшие скрипты до и после отправки запроса.

**Задание №3. Начало работы с Postman**

В этом задании вам предстоит создать свой первый запрос в Postman. С чего начать? Сейчас расскажем!

1. Прежде чем приступить к работе с запросами, руководителю нужно создать рабочую область для команды (team workspace) и пригласить в него всех участников команды.
2. Теперь создайте коллекцию, переименуйте её в "Fake REST API". В коллекции создайте папку, которой дайте название "Activities". В этой папке будем сохранять запросы, которые работают с блоком методов "Activities".
3. В папке создайте первый запрос на эндпоинт GET /api/v1/Activities и отправьте его.

Сделайте скрин результата работы (нужно уместить запрос, часть ответа, и левую шторку с коллекцией). Назовите его exercise3.png.

**Основные вкладки конструктора запросов**

Postman предоставляет нам удобный интерфейс для добавления заголовков, параметров и тела к запросу.

Чтобы добавить параметр в запрос, не нужно его вписывать прямо в сам URL — это сделает за нас Postman, нам лишь нужно добавить пару "ключ-значение" во вкладку "Params".

Аналогично и с заголовками запроса — они добавляются во вкладке "Headers".

Если в запрос нужно добавить данные, необходимые для авторизации (например, bearer token), то это можно сделать во вкладке "Authorization".

Тело запроса добавляется во вкладке "Body". К примеру, если нам нужно добавить данные в формате JSON, то необходимо выбрать тип "raw" и формат "JSON".

**Переменные окружения**

Часто мы сталкиваемся с тем, что нужно передать какой-нибудь GUID (или, например, просто идентификатор объекта передать в URL) из одного запроса в другой — для этого его нужно скопировать с одной вкладки, а затем вставить в другую. А если речь идёт о необходимости постоянной авторизации? Всё это будет занимать наше время, которого в реальных условиях разработки ПО и так достаточно мало. Оптимизировать подобные процессы нам помогут **переменные окружения**. Они нужны для того, чтобы передавать данные из запроса в запрос внутри созданного окружения.

Отличный пример использования переменных окружения — передача id объекта из запроса POST в запрос GET или DELETE. Когда у нас много запросов к одному объекту, удобно создать одну переменную, которая будет хранить в себе значение id, и уже пользоваться непосредственно одной переменной, редактируя её сразу во всех запросах.

Прежде чем добавлять переменную, нужно создать окружение. Это можно сделать в выпадающем окне в правой верхней части рабочей области Postman:

Создаём окружение, затем добавляем новую переменную activity\_id и присваиваем ей значение:

Теперь для того, чтобы использовать нашу переменную, надо ввести в необходимом нам месте {{название переменной}}, в нашем случае — это {{activity\_id}}. Postman автоматически подставит значение переменной из окружения в запрос. Переменную можно вставлять куда угодно: в тело запроса, в его URL, заголовки, параметры и пр.

**Pre-request script и Tests**

Для полноценной "картины" не хватает лишь одного — автоматического обновления переменных окружения, чтобы нам не приходилось проставлять значения вручную. Для работы с JavaScript-кодом в конструкторе запроса Postman существуют отдельные вкладки: [Pre-request script](https://learning.postman.com/docs/writing-scripts/pre-request-scripts/) и [Tests (test scripts)](https://learning.postman.com/docs/writing-scripts/test-scripts/). Ключевое различие этих скриптов заключается в том, что одни выполняются до отправки запроса, а вторые — после получения ответа.

Весь код далее мы будем писать во вкладке "Tests", так как нам необходимо работать с ответом на запрос. Если потребуется, то вы можете также использовать и pre-request скрипты. Более подробно ознакомиться с этими двумя вкладками вы можете на официальном сайте Postman (гиперссылки приложены выше).😊

Если вы не знакомы с языком программирования JavaScript, то всегда можете воспользоваться одним из готовых шаблонов скриптов в правой части экрана (сниппетами). К примеру, сниппет "Set an environment variable" обратится к переменной окружения — нам лишь нужно вписать пару ключ-значение в соответствующие поля.

Основным объектом, который позволяет работать с запросом, ответом, переменными окружения и тестовыми методами, является pm.

Чтобы получить тело ответа в JSON-формате, нам нужно вписать одну строчку:

const res = pm.response.json();

После этого мы можем работать с полученным ответом. Например, вытащить из него id (в случае если он возвращается в качестве поля "id" в объекте, который мы только что создали) и записать в переменную окружения:

const res = pm.response.json();

const activity\_id = res.id;

pm.environment.set('activity\_id', activity\_id);

Этот код можно вставить во вкладку "Tests" при создании новой активности, чтобы после каждого нажатия на кнопку "Send" у нас обновлялся id объекта. А чтобы обезопасить себя от любой непредвиденной ситуации, можно дополнить код условными операторами:

if (pm.response.code == 201) { // то есть, если код ответа = 201, то:

const res = pm.response.json(); // получаем ответ

const activity\_id = res.id; // получаем id объекта

if (activity\_id) // и если этот id существует (если мы его получили)

pm.environment.set('activity\_id', activity\_id); // записываем в окружение новое значение переменной activity\_id

}

**Задание №4. Тестирование API с Postman**

А теперь давайте применим полученные знания на практике! Используя тест-план, полученный в TestIT, снова проведите тестирование API (проверьте все имеющиеся тест-кейсы). Пользуйтесь переменными окружения, чтобы упростить себе задачу!

В файле exercise4.md вам нужно будет фиксировать все запросы и ответы. Отчёт к каждому запросу нужно будет оформить по следующему шаблону:

* URL (вставляйте его прямо со скобками, например, "[https://fakerestapi.azurewebsites.net/api/v1/Activities/{{activity\_id}}](https://fakerestapi.azurewebsites.net/api/v1/Activities/%7B%7Bactivity_id%7D%7D)")
* Ожидаемый результат (своими словами)
* Заголовки запроса (можно посмотреть через консоль в левом нижнем углу)
* Тело запроса (если тело слишком большое, то только его часть)
* Заголовки ответа
* Тело ответа (если тело слишком большое, то только его часть)

*P.S.: Тело запроса и тело ответа нужно приводить, только если они существуют и не являются пустыми.*

Также сделайте следующие скриншоты:

* Скриншот левой шторки Postman, назовите его exercise4.1.png
* Скриншот переменных окружения и глобальных переменных Postman, назовите его exercise4.2.png

**Double-check**

Перед загрузкой выполненного проекта в репозиторий перепроверь наличие всех необходимых файлов, которые требовалось создать во время его выполнения:

exercise1.md

exercise2.plan.xlsx

exercise2.results.xlsx

exercise2.md (если были найдены баги)

exercise3.png

exercise4.md

exercise4.1.png

exercise4.2.png

💡 [Нажми здесь](https://forms.gle/CbLTkfMkLLcR3L5t8) **чтобы отправить обратную связь по проекту**.