**Тестирование API**

**API (Application Programming Interface)**расшифровывается как “интерфейс прикладного программирования” или “интерфейс программирования приложений”. Он позволяет осуществлять связь и обмениваться данными между двумя отдельными модулями программы. Система программного обеспечения, реализующая API, содержит функции/подпрограммы, которые могут быть выполнены с помощью другого программного обеспечения.

"Общение" между модулями приложения происходит с использованием стандартных форматов XML и JSON и посредством специальных протоколов REST и SOAP.

Например, некое приложение, сервис предоставления данных о прогнозе погоды - имеет API, которым могут пользоваться разработчики. То, каким образом разработчики будут пользоваться, зависит от возможностей API. Например, может ли API выдавать данные о прогнозе погоды на неделю вперед, по каким городам мира выдаются данные, возможно ли запросить такие данные, как скорость ветра, давление и т.д.

**Форматы данных**

Как и говорилось выше, основные форматы, которые используются для передачи данных в API - это JSON и XML. На изображении ниже представлена одна и та же информация в разных форматах.

В JSON существуют типы данных, которые записываются по-разному. Данные в JSON записываются парами "Ключ":"Значение". Например:

**{“name”:”JamesKirk”}**

**Имя параметра** — это строка в двойных кавычках слева от двоеточия.

**{“name”}**

**Значение**— может быть строкой в двойных кавычках, числом, логическим значением (true или false), объектом, массивом, или значением null. Эти структуры могут быть вложены друг в друга.

**{”JamesKirk”}**

**Объект**— это множество пар "Ключ":"Значение", заключённое в фигурные скобки { }. Между именем параметра и значением стоит двоеточие ":", а пары "Ключ":"Значение" разделяются запятыми “,”.

**{**

**“name”:”JamesKirk”,**

**"age":40**

**}**

**Строка**— это упорядоченное множество из нуля или более символов Unicode, заключенное в двойные кавычки.

**Массив**— это множество объектов. Массив заключается в квадратные скобки [ ], а значения отделяются запятыми (см. пример на изобрежнии выше).

В XML данные хранятся между так называемыми "тэгами".

Существуют открывающие и закрывающие тэги, а данные, в свою очередь, хранятся между ними.

Например:

<note> - открывающий тэг;

</note> - закрывающий тэг.

Примечательно то, что тэги чувствительны к регистру. Другими словами, нельзя использовать открывающий тэг <MESSAGE> и закрывающий тэг </message>. XML воспринимает это как разные тэги.

Более подробно о принципах построения XML можно изучить в официальной документации [тут](https://www.opennet.ru/docs/RUS/XML/).

XML является более громоздким форматов данных и все больше разработчиков API от него отказываются.

**Понятие HTTP**

**HTTP (Hyper Text Transfer Protocol)**– широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов. По умолчанию используется 80-ый порт.

**HTTPS (Hyper Text Transfer Protocol Secure)**— безопасный протокол передачи гипертекста. Это расширение протокола HTTP, поддерживающее шифрование посредством криптографических протоколов SSL и TLS. По умолчанию используется 443-ий порт.

Спецификация HTTP (и HTTPS) определяет то, как запросы к серверу должны быть построены, и то, как сервер должен отвечать на эти запросы.

Основные свойства HTTP:

* **Не зависит от соединения.**Для отправки запроса клиент устанавливает соединение с сервером и отсоединяется после отправки запроса. Сервер, в свою очередь, обрабатывает запрос и устанавливает соединение с клиентом для отправки ответа и отсоединяется после нее. Ни клиент, ни сервер не "знают" ничего о состоянии друг друга до начала соединения и после его окончания.
* **Не привязан к конкретному типу данных.**Можно передавать любой тип данных при условии, что и клиент, и сервер способен работать с выбранным типом данных.
* **Взаимодействует только через соединение.**Клиент и сервер могут взаимодействовать друг с другом только с помощью запроса. Из-за этой особенности ни клиент, ни сервер не могут получить информацию за пределами запроса.

**Запросы HTTP**

Клиент отправляет запрос на сервер в виде метода, URL и версии протокола, после которого идет некоторое сообщение, которое и содержит данные запроса.

Разберем подробнее каждую из частей запроса.

**Method (метод)**- это действие, которое мы хотим произвести над ресурсом на сервере. Их достаточно большое количество, но выделим основные 4:

* **GET** - предназначен для получения ресурса с сервера;
* **POST**- отправляет данные на сервер с созданием новой записи;
* **PUT**- отправляет данные на сервер с перезаписью существующей записи;
* **DELETE**- удаляет данные ресурса.

| **GET** | **POST** | **PUT** | **DELETE** |
| --- | --- | --- | --- |
| Только получает данные ресурса | Может получать и отправлять данные ресурса | Перезаписывает существующий ресурс | Удаляет указанный ресурс |
| Передает данные в URL | Передает данные в теле запроса | Передает данные в теле запроса | Данные могут передаваться в теле и в URL запроса |
| Имеет ограничение на длину 255 символов | Нет ограничений по длине | Нет ограничений по длине | Нет ограничений по длине |
| Можно использовать только символы ASCII | Можно использовать символы любой кодировки и передавать файлы | Можно использовать символы любой кодировки и передавать файлы | Можно использовать только символы ASCII |
| Не безопасен (нельзя передавать пароли) | Более безопасен | Более безопасен | Не безопасен |

**Request URI**- строка запроса, которая содержит последовательность символов к ресурсу, а также (опционально) параметры запроса, которые могут передаваться прямо в строке запроса (например, для GET).

Для передачи параметров в строке запроса необходимо следовать ряду определенных правил:

* Параметры отделяются от адреса символом "?".
* Каждый параметр задается парой "Ключ" и "Значение".
* "Ключ" и "Значение" разделены между собой символом "=".
* При необходимости задать несколько параметров в одной строке запроса, они отделяются друг от друга символом "&".

Например, в строке запроса <http://example.com/path/to/page?name=ferret&color=purple>:

* <http://example.com/> - базовый адрес (base URL), с которого будут начинаться все запросы;
* /path/to/page - путь к ресурсу относительно базового адреса;
* Параметр name со значением ferret;
* Параметр color со значением purple.

Стоит отметить, что данные в строке запроса должны передаваться в специальной кодировке - URL Percent Encoding. Таким образом, чтобы передать в строке запроса, например, символы кириллицы, необходимо перевести их в этот формат. В сети существует множество инструментов, позволяющих легко перевести строку запроса в нужный формат.

**Protocol version**- версия протокола HTTP (практически всегда используется HTTP/1.1).

**Headers (заголовки или "хедеры")**- часть запроса, в которой хранится необходимая для выполнения запроса информация от клиента.

Заголовки представляют пары "Ключ":"Значение". Они содержат различную информацию о HTTP-запросе и Вашем браузере. Например, строка "User-Agent" предоставляет информацию о версии браузера и операционной системе, которую Вы используете. "Accept-Encoding" сообщает серверу, может ли Ваш браузер принимать сжатый output, например, gzip.

В свою очередь, ответ сервера так же содержат заголовки. Эти значения могут содержать информацию о софте сервера при последнем изменении страницы/файла и прочее. Опять же, большинство этих headers на самом деле являются необязательными.

Кроме этого, сервер отправляет так же код состояния (statuscode) ответа. Коды состояния делятся на 5 групп:

| **1хх** | **Информационные** |
| --- | --- |
| 2хх | Успешные |
| 3хх | Перенаправление |
| 4хх | Ошибки клиента |
| 5хх | Ошибки сервера |

Полный список кодов ошибок из каждой группы и их описание можно посмотреть [тут](http://www.restapitutorial.ru/httpstatuscodes.html).

**Body (тело запроса)**- опциональное поле, в котором передается вся необходимая информация, которую нужно передать на сервер.

**Рассмотрим выполнение запросов на примерах.**

Общим предусловием для всех примеров будет наличие сайта [https://reqres.in](https://reqres.in/), который позволяет производить различного рода действия над пользователем. В данном случае, [https://reqres.in](https://reqres.in/) - это базовый адрес (base URL), к которому будут добавляться пути к ресурсам.

**GET запрос.**Имеет функцию получения списка пользователей. Задача данной функции - отображать список пользователей по три записи на странице.

GET /api/users?page={page\_number}

Получим список пользователей для второй страницы:

Как мы видим, в поле ниже мы получили ответ в формате JSON, который содержит 3 записи о пользователях.

**POST запрос.**Функция используется для входа в приложение и возвращает ответ со значением токена авторизации.

POST /api/login

**Параметры тела запроса:**

* *email*– новое имя пользователя;
* *password* – новая профессия пользователя.

Вызов функции и ответ сервера будет выглядеть таким образом:

В данном примере строка запроса не содержит параметров. Body запроса же в свою очередь содержит два параметра - email и password. Ответ функции содержит Body с токеном авторизации пользователя.

**PUT запрос.**Функция изменения данных пользователя. Возвращает ответ с измененными данными пользователя и датой изменения.

PUT /api/users/{user\_id}

**Параметры тела запроса:**

* *name* – новое имя пользователя;
* *job* – новая профессия пользователя.

Подставив параметры в URI и тело запроса, получим:

**DELETE запрос.**Функция удаления данных пользователя. Функция возвращает пустой ответ и код 204 (No Content)

DELETE /api/users/{user\_id}

**Тело запроса не содержит данных.**

Подставив нужный идентификатор пользователя в строку запроса, удалим его данные:

**Понятие REST**

**REST (Representational state transfer)**- подход к разработке клиент-серверных приложений.

Приложения на REST архитектуре должны быть:

* Клиент-серверными.
* Взаимодействие между клиентом и сервером должно быть на HTTP.
* Все операции над ресурсами указываются в самих запросах. В архитектуре REST все данные являются "ресурсами" Все, что необходимо сделать с ресурсом в архитектуре REST, несется в самом запросе.
* Stateless – состояние клиента не сохраняется на сервере. Каждый раз, при обращении клиента к серверу, сервер воспринимает клиента как нового. Для аутентификации клиента на сервере могут использоваться cookies, например: сookies предоставляет дополнительную информацию от клиента пользователю (позиции в корзине пользователя в интернет-магазине).
* Возможность работать с любыми форматами данных (json, xml, text…).

**Понятие SOAP**

**SOAP (Simple Object Access Protocol)** является стандартизированным протоколом передачи сообщений между клиентом и сервером. Обычно он используется совместно с HTTP(S), но может работать и с другими протоколами прикладного уровня (например, SMTP и FTP).

В отличие от REST, который может использовать любые форматы данных, SOAP работает только с XML форматом. При работе всегда удобно иметь стандартизированное описание возможных XML-документов и проверять их на корректность заполнения. Для этого существует **XML Schema Definition** (или сокращенно XSD). Две главные функции XSD для тестировщика – это описание типов данных и наложение ограничений на возможные значения. Например, некоторые элементы ответов сервера можно сделать необязательным для заполнения или ограничить его размер 255 символами с помощью XSD. Чем подробнее описан XSD, тем меньше головной боли доставит Вам тестирование сервиса. С помощью выстроенной схемы сервис сам сможет валидировать полученные данные и возвращать пользователю ошибку. Подробнее прочитать про XSD можно на [w3schools](https://www.w3schools.com/xml/schema_intro.asp) и [codenet (по-русски)](http://www.codenet.ru/webmast/xml/part6.php).

**WSDL  
(Web Services Description Language)** – это язык на основе XML, который используется для описания веб-сервисов. В WSDL-документе содержится информация о местонахождении сервиса и доступных методах (операциях). Для каждого метода определяются параметры отправляемого и получаемого сообщения. Обратите внимание на то, что XSD может быть «встроен» внутрь WSDL-документа (например, у Yandex Speller API).

**Типы тестов, применимые к тестированию API**

В целом, к тестированию API применимы следующие типы тестов:

* *Функциональное тестирование* – тесты должны выполнить набор вызовов, задекларированных в API, чтобы проверить общую работоспособность системы.
* *Usability-тестирование* – проверяет, является ли API функциональным и обладает ли удобным интерфейсом, также проверяется интеграция с другими.
* *Тестирование безопасности* – проверяет используемый тип аутентификации и шифрование данных с помощью HTTP.
* *Автоматизированное тестирование* – создание скриптов, программ или настройка приложений, которые смогут тестировать API на регулярной основе.
* *Тестирование документации*– проверяется полнота описаний функций API, её понятность.

**При тестировании API необходимо проверять следующие моменты:**

* Правильный ли метод используется для того или иного запроса?
* Проверяйте, что клик по одной и той же кнопке вызывает один и тот же запрос.
* Вникайте в отправляемые запросы. Анализ запросов – это возможность обнаружить спрятавшийся дефект гораздо быстрее, чем осуществляя его поиск в интерфейсе.
* Мониторьте трафик на предмет запросов на другие сервера.
* Внимательно следите за кодами состояний

**С помощью тестирования API можно обнаружить следующие типы ошибок:**

* Сбой обработки ошибочных условий при передаче корректных и некорректных данных в запросах.
* Неиспользуемые флаги в параметрах запросов.
* Отсутствующая или дублирующаяся функциональность.
* Вопросы надежности: трудности при подключении и получении ответа от API.
* Проблемы с безопасностью.
* Проблемы многопоточности.

**Лучшие практики тестирования API:**

* Тест-кейсы должны быть сгруппированы по тестовым категориям.
* Каждый тест должен включать декларацию тестируемой функции.
* Выбор параметров должен быть явно упомянут в самом тесте.
* Установка приоритетов вызова функций API.
* Каждый тест должен быть самодостаточным и независимым друг от друга.
* Особую осторожность следует соблюдать при обращении к функциям удаления, закрытия окна и прочим.
* Вызов последовательности действий должен быть хорошо спланирован и выполнен.
* Для обеспечения полного тестового покрытия создавайте тестовые случаи для всех возможных комбинаций входных данных.

Так же мы можем использовать такие общепринятые техники, как анализ граничных значений и разбиение на классы эквивалентности. В API запросах в явном виде могут передаваться значения параметров. Это отличный повод выделить границы входных и выходных значений и проверить их. Даже у небольшого API есть множество вариантов использования и множество комбинаций входных и выходных переменных. Поэтому мы можем лишний раз использовать наши навыки выделения эквивалентных классов.

Т**естирование API обладает рядом преимуществ перед обычным тестированием через UI:**

* Точное понимание, где происходит ошибка и чем она вызвана.
* Тратится меньше времени на подготовку тестовых данных.
* Возможно выполнение тестов на больших объемах данных с приемлемой скоростью.
* Можно начать тестирование на ранних этапах, когда еще нет интерфейса

**Проблемы, с которыми сталкиваются тестировщики при работе с API:**

* Комбинация и выбор параметров.
* Отсутствие графического интерфейса.
* Валидация и верификация выходных данных в разных системах.
* Обязательная проверка обработки исключений.
* Тестировщикам необходимы знания в программировании.