HOW TO MAKE YOUR DEVELOPERS REST

(WITHOUT GIVING THEM A DAY OFF)

PayLane
way ahead

INTRODUCTION

In today's world websites aren't what they used to be. They have evolved from simple information pages to something much, much bigger and more advanced. We can easily say that in the past ten years technology changed how we gather and process information, how we get to know about certain things and how we learn. But what's more important — it changed our creative process. The role of a website is different than it was just a few years ago — now we can often call them “web applications” or “online services”. The old ways of describing websites are now in the past.

Behind any web application and any kind of service, from simple blog posts through collecting and processing data (e.g. from social media) to accepting payments, there's the architecture seen by their developers.

The developers, because we are talking about them, create new unseen things and make our online life or businesses better everyday. But even they are very much dependent on technology. Nowadays these guys are combining many, many APIs to create something new. But a common issue nowadays is that they are struggling with many problems related to these technologies. Till now there wasn't a simple explanation how APIs should be made, and every developer considered his or her way of doing things the best.

But all they really need to know and have is **REST**. We'll talk about what it really is, how to use it and whether it's really worth all the praise.

API stands for **Application Programming Interface** and describes how certain software components or layers should communicate. In practice, it's usually a library with methods that realize specific actions.

I. WHAT IS **REST**?

We'll try to explain it in simple terms: REST is basically just a new set of rules that every developer should follow. More technically, it's a web software architecture that describes how to handle API requests and its design. REST is all about handling requests and responses in a new and easy way, without the need for overly complicated documentation.

Today we are surrounded with APIs. When we click a *Like* button under a blog post, refresh tweet timeline in our favorite Twitter client, search for a new movie on Netflix or watch the said movie — we are using API calls. For instance, back in 2011, Twitter was processing roughly 13 billion API calls per day! You can only imagine that this amount probably tripled in the past two years. This is one of the reasons why the REST standard spreads across the world — developers were just simply tired of the wrong written web APIs.

During the past few years REST became a trending technology in web architecture — it is now considered a true web standard and replaced its antecessor – SOAP (Simple Object Access Protocol). You could ask why this technology is now so popular. It's simple — creating great APIs is one of the most important parts of creating web services right now. RESTful APIs design is characterized by their key principles that we are going to describe in this section.

1. KEEP SIMPLE THINGS SIMPLE, FOOL!

You don't need to complicate simple things that should be recognized in no time — like API URLs, or method names. REST is designed to be as simple as possible — keep that in mind when creating your system. Your goal is to allow developers to use your interface without them worrying about the documentation.

2. FOLLOW THE STANDARD, USE WHAT'S ALREADY THERE

The RESTful API standard was created in parallel with W3C HTTP 1.1 standard – for example you don't really need to use your own authentication protocol — you can make a good use of the *HTTP Basic Auth* instead to protect your API from strangers. All you need is to utilize what HTTP protocol has to offer.

3. STANDARDIZE YOUR OUTPUTS AND REQUESTS

The way you should think about RESTful API is that when you're giving your end customer access to certain endpoints, their requests should unambiguously describe a resource that can be further modified.

4. FOLLOW THE TREND, BUT STAY FLEXIBLE

It is very important to follow current trends — even if we're talking about easy stuff like media types used for handling requests and responses. For REST, a widely used standard is to communicate with APIs using JSON structures. But still, you should be flexible and provide your clients with a way to interact with you using other media types — for instance giving them the opportunity to respond with an XML structure.

II. WHY EVEN BOTHER WITH A **REST**FUL **API**?

Designing and creating a RESTful API may cost some time and effort, but it surely pays off in the long run. When you create your web application and want to make it open and available to third-party developers that will help you expand your functionalities, a good REST API with great documentation is simply the best way to achieve this goal and gain customers attention .

Links to exemplary API documentations

Twitter API

Dropbox API

Zendesk API

PayLane API

It not only means that you are following the biggest trend in architecture development community and you will have a better start in the eyes of programming communities, but it shows that you're open to all possible technologies — from web services to desktop and mobile applications.

Additionally, integrating with standard solutions is always easier than adapting to a new thing when starting to work with new services — with REST API you can use the very same class **wrapper** that you use to communicate with, for example, Twitter. All you need to do is extend a class that you already have with new methods that are described in their respective

Wrapper – a universal class that allows to perform, for example, REST calls. You can extend such basic class by adding additional methods (inheritance). A wrapper class is the most basic one, created to perform API requests to a specific server.

documentations. Isn't that a better way? This approach can save you and your future business a lot of time — integrating with any kind of REST API can take minutes – not hours or days.

We think that REST APIs are something that changes the face of web services. It takes away the heavy burden of integration. This is the way that you should think about REST – it's simple, and allows you to integrate **faster**. In this case you can only imagine what your future customers may say about your services when you offer them a tool like this.

Additionally, REST is typically just a faster way to accomplish some tasks. In a modern development environments there are a lot of new web technologies. Some of them might be JavaScript frameworks (like **Meteor**) or non-relational databases (**MongoDB**).

What those technologies have in common nowadays is the JSON format — you can pass the gathered from a REST endpoint directly to your service or (in secure cases) directly to your database. It all became so popular that businesses that missed all the said became industry dinosaurs. And when you use new technologies, you usually use REST as well.

This technology is designed to break the conventions of separating web, mobile and desktops. It allows you to connect them all with simple HTTP calls (we will talk about them in a while), gives you the possibility to operate on every platform and helps exchange information between them. This is a pure new business opportunity for your current and future customers.

Summarizing this section, having a good REST API could lead to gaining new users and making more money. From the developer's point of view — using a good API allows more fluent and productive work.

III. HOW TO USE A **REST**FUL **API** – IMPLEMENTATION

In this section we'll be a little bit more technical to show you how REST works and how it should be implemented. As we said, integration with REST is relatively simple and easy to achieve. If you want basic stuff, all you need is cURL or a similar tool that allows you to perform simple HTTP 1.1 requests. For the purpose of this article, we'll stay with simple HTTP requests — because of that we'll stick as close as it can be to the standard — REST should not be directly related to any programming language and should be kept as simple as possible.

But first we need to learn some basics. REST API takes a good use of typical HTTP request methods such as **POST, GET, PUT, DELETE**. Also, we have more complex methods: OPTIONS, HEAD, TRACE, CONNECT. We'll focus on the first four, because they are the most widely used in current APIs around the globe. If you are afraid of them, we need to clear up your mind a little. First of all, let's take a look at these four methods and try to compare them with CRUD operations. Not a clue? Well, we were prepared for that.

- POST is commonly used for creating new things, it's similar to *Create* from CRUD terminology;
- GET is just a simple query with parameters (*Read*);
- PUT updates or replaces data for us, treat it like an SQL *update/replace into* statement or *Update* (CRUD);
- DELETE speaks for itself — stands for the same purpose as *Delete* in CRUD operations.

Let's see how it works on a simple example – imagine for a moment that we have a car factory API and we want to make it more RESTful. In this simple example we want to show you the basic concept of REST HTTP methods while we manipulate simple URL endpoints. Take a look at the table below:

Request URL	**POST**	**GET**	**PUT**	**DELETE**
/factory	Create a new car	Get all cars	Replace/ update all cars	Delete all cars
/factory/ford	Create a new ford car	Get all fords	Replace/ update all fords	Delete all **fords**

OK, now let's try to learn how REST API works on particular examples — for the sake of this article we will be using **Postman**, a REST Client for Google Chrome. We strongly recommend you to do the same — it's a great tool, very easy to learn. Right now we are going to talk briefly about these four methods.

GET

To get our first results and feel the REST magic, we'll prepare our first request. For this purpose we'll use a very simple REST endpoint that has only one job to do — return our IP number.

To do that, simply provide this address: http://call.jsonlib.com/ip into a request URL field in the Postman tool. Set your method to GET and click Preview. You'll see something similar to this:

```
GET /ip HTTP/1.1
Host: call.jsonlib.com
Cache-Control: no-cache
```

What you are seeing right now is a simple GET request via HTTP that is ready to be sent to the server. After hitting the Send button, you'll get a response that will include a status code 200 (OK) and a JSON object with your IP number — in our case it looks like this:

```
{
    "ip" : "89.206.36.193"
}
```

Congratulations, you just performed your first REST operation! Now, when we know the basics, it's time for something a little more complex.

POST

This time, in Postman, you need to switch the request method to POST. Put a new URL (http://call.jsonlib.com/echo) into the address bar. When you change the method from GET to POST a new form will show – it's a simple representation of a JSON metadata object that we will send to the server. Fill it with some data. You can check your request by pressing *Preview* to see how the server will see your request and then press *Send*. Our data is a little bit different from what you have posted, but the server should respond with the data you have typed into Postman. In our case the response looks like this:

```
{
    "lastname" : "Banks",
    "id"       : "1",
    "name"     : "Tom"
}
```

PUT & DELETE

There's no simple way to test these two methods in the real world, but let's assume we have a blog post in our system and want to change its title. For this case you want to use the PUT method that is designed to handle update/replace statements. To give you an example, we have prepared a JSON object that you might use in your project:

```
{
    "title" : "This is a brand new title of my post."
}
```

We are assuming that your REST endpoint is named *blogpost/{post_id}*. When you would send a PUT request with this data, your backend server should recognize it as a blog post title change in a post that is identified by `post_id` that you've provided in resource URL.

The same idea can be used to handle DELETE requests — you should be able to send a request to your REST endpoint without any metadata, just by manipulating the resource URL. It might look like this: *https://api.yourbusiness.com/blogpost/{post_id}* with `{post_id}` being a variable. Requesting this URL should lead to deleting blog post identified with the provided ID number.

Of course the obvious aspect for your system is user authentication (HTTP 1.1 Basic Auth) for that kind of tasks — you really wouldn't want to let anyone who finds out an API endpoint to delete your favorite blog post about cats, would you?

IV. **REST API** – PRACTICAL EXAMPLE

Congratulations for reaching that far in our whitepaper! Now, that you get what REST is really all about, we have some examples that should convince you to use this technology in your projects. All we need in this exercise is a command line and a cURL library.

First thing that we need to do is to create a test account at PayLane.com and collect API credentials. Then, we take a look into the REST API Function Reference (available to everyone at Developer Zone) to pick one of the methods and try to process a simple request. In this particular example, we'll use our most popular method — cards/sale.

No worries, no actual money will be processed here – it's just a test example.

POST cards/sale

Resource URL: `POST https://direct.paylane.com/rest/cards/sale`

The `cards/sale` method allows to perform a single transaction using a card

(credit, debit, pre-paid etc.). It includes additional security features, such as *AVS* or *fraud check*.

Now, when we know what it does, we need to prepare a request structure that will be compatible with structure reference. Let's just be pretty straight-forward and copy a sale structure from the cards/sale method example.

After we do that, we need to prepare a cURL query that will be sent to PayLane's system. In this case we have prepared a sample query for you. If you still want to use the Postman application that we were using last time — do not hesitate. The only reason why we are using cURL is that it makes it much easier for us to demonstrate a single API call. At this point you should really see that REST technology is very much independent of any program or library that you use to communicate.

As we said, our metadata JSON structure is taken from example request and looks like this:

```
{
    "sale"      :   {
                    "amount"          :   100.00,
                    "currency"        :   "EUR",
                    "description"     :   "A brilliant product, #52704",
                    "fraud_check_on"  :   true,
                    "avs_check_level" :   2
    },
    "customer"  :   {
                    "name"    :   "John Doe",
                    "email"   :   "john@doe.com",
                    "ip"      :   "123.456.78.90",
                    "address" :   {
                                "street_house"  :   "1600 Pennsylvania Avenue Northwest",
                                "city"          :   "Washington",
                                "state"         :   "DC",
                                "zip"           :   "20500",
                                "country_code"  :   "US"
                    }
    },
    "card"      :   {
                    "card_number"       :   "4111111111111111",
                    "expiration_month"  :   "03",
                    "expiration_year"   :   "2017",
                    "name_on_card"      :   "John Doe",
                    "card_code"         :   "123"
    }
}
```

Now, all we need to do is to prepare a valid cURL request, using API credentials and execute it in the Terminal.

```
curl --request POST 'https://test_login:test_password@direct.paylane.com/rest/cards/sale' \
     --data '{"sale":{"amount":19.99,"currency":"EUR","description":"Product
#1"},"customer":{"name":"John Doe","email":"john@doe.
com","ip":"127.0.0.1","address":{"street_house":"1600 Pennsylvania Avenue
Northwest","city":"Washington","state":"DC","zip":"500","country_code":"US"}},"card":{"card_
number":"4111111111111111","expiration_month":"03","expiration_year":"2017","name_on_
card":"John Doe","card_code":"123"}}' \
     -i \
     --header 'Content-type:application/json'
```

The last parameter — *header* — is optional and it's provided to make sure that data which we are sending to our API endpoint is in a valid JSON format.

When we execute this prepared query, a POST HTTP request will be sent to the API System and create a valid sale — in this the case server response is pretty clear:

```
{
    "success"     :   true,
    "id_sale"     :   1234567,
    "avs_result"  :   "M",
    "fraud_score" :   10
}
```

This time we have succeed and we processed some fictional money from a customer. We received a sale identifier that we can store in our system and use in other operations via API, like resale or getting sale information for further use.

CONCLUSION

We hope that thanks to this short document your knowledge of REST APIs and services that use it is much better. The basic conclusion might be that REST is designed to be a simple web standard that utilizes basic HTTP requests and helps developers to communicate better with present and future web services. Thanks to REST API precursors like Twitter, Facebook or Netflix we're finally getting to a point where implementing new services isn't a problem anymore and our productivity is enhanced.

Happy **REST**-ing! :)
PayLane Team

SOURCES & RESOURCES

API calls stats:
http://blog.programmableweb.com/2011/05/25/who-belongs-to-the-api-billionaires-club/

Postman for Chrome:
https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjo-ooidkmcomcm

Teach a dog to REST:
https://blog.apigee.com/detail/restful_api_design

Beginners guide to creating a REST API:
http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api

call.jsonlib:
http://call.jsonlib.com

PayLane documentation:
http://devzone.paylane.com

Creating a test account for the practical example:
http://paylane.com/create-account

COMPILED BY MACIEJ KOŁEK

Maciej is a **Web Application Developer** at PayLane.
Apart from being a programmer, he has a huge interest in all things related to mobile development and business aspects of that subject. He is always eager to learn about technologies that make it easy to build for the web. You can follow him and ask him any questions on Twitter: @ferusinfo.