TP7 Introduction to Django

Creation of an API

Luc Bertin

What is an API ?

How does data get from one place to another ?
How do different devices and applications connect to each other to allow us to place an order, make a reservation or book a flight with just a few types of things ?
The agent under the hood: **Application programming interface or API** ! It is the messenger that **take requests and tell the system what you want to do**, and then **returns the response back to you**!
Example: an API could be the waiter in a restaurant.

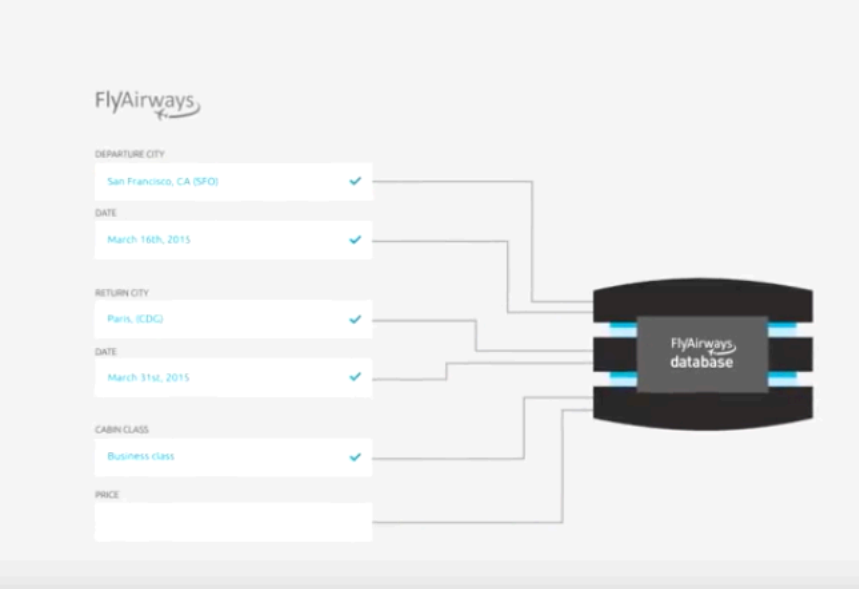

Figure 1: You request a flight in the website of FlyAirways, informing a few variables in the drop-down menus while interacting with this website. FlyAirways'API will post or get data from the FlyAirways'database with details you've provided.

Figure 2: If your using an online travel website that aggregates information from many different airlines. The service interacts with the different airlines' API. It can ask over the internet to order the seats, book a flight, etc.

From ©Wikipedia : An application programming interface (API) is an **interface or communication protocol** between **different parts of a computer program** intended to **simplify the implementation and maintenance of software**.
More recently, the term has been often used to refer to a **specific kind of interface between a client and a server**, which has been described as a "contract" between both - such that if the **client makes a request in a specific format, it will always get a response in a specific format or initiate a defined action**. This is a specialized form of API, sometimes defined as a **WEB API**

What is REST ?

REST, or **REpresentational State Transfer**, is an architectural style for providing **standards** between computer systems on the web, making **it easier for systems to communicate with each other.**
How to make an API that respects constraints implied by REST architecture style? In other words: how to make an **API RESTful** ?

What is a client ? What is a resource ?

Client: **person** or **software** who uses the API.
Example1: You, as an ESILV student could use Twitter API to retrieve tweets as part of a "PRM"® by creating a Python program.
Example2: When you go to Twitter website, your browser is the client who calls Twitter API and uses the returned data to render information on the screen.
Resource: can be any object the API can provide information about.

Example: in Instagram API: resource could be a user, a photo, a hashtag. Each resource has a unique identifier. Identifier can be a name or a number.

How to make an API “Restful”?

REST APIs brief explanation : https://www.youtube.com/watch?v=7YcW25PHnAA

RESTFul 6 constraints to respect: https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a582?

Requests from client to server ?

An API request lets you contact a server with API endpoints that you want to reach and perform some action. Those actions are HTTP methods.

The most common methods are GET, POST, PUT, and DELETE. The names of the methods are self-explanatory. For example, GET enables you to retrieve data from a server. POST enables you to add data to an existing file or resource in a server. PUT lets you replace an existing file or resource in a server. And DELETE lets you delete data from a server.

Best practices for CRUD (Create Read Update Delete): (https://www.codecademy.com/articles/what-is-crud

POSSIBLE SOLUTION - REQUESTS/RESPONSES

GET REQUESTS

Request- `GET /index.html` Accept: `text/html` Response- 200 (OK) Content-type: `text/html`

Request- `GET /style.css` Accept: `text/css` Response- 200 (OK) Content-type: `text/css`

Request- `GET /venues` Accept: `application/json` Response- 200 (OK) Content-type: `application/json`

Request- `GET /venues/:id` Accept: `application/json` Response- 200 (OK) Content-type: `application/json`

Request- `GET /venues/:id/photos/:id` Accept: `application/json` Response- 200 (OK) Content-type: `image/png`

POST REQUESTS

Request- `POST /users` Response- 201 (CREATED) Content-type: `application/json`

Request- `POST /venues` Response- 201 (CREATED) Content-type: `application/json`

Request- `POST /venues/:id/photos` Response- 201 (CREATED) Content-

Appendix: other information

When should I **use GET** or **POST method**?

HTTP/1.1 specification (RFC 2616) section 9 **Method Definitions** contains more information on `GET` and `POST` as well as the other HTTP methods, if you are interested.

In addition to explaining the intended uses of each method, the spec also provides at least one practical reason for why `GET` should only be used to retrieve data:

> Authors of services which use the HTTP protocol SHOULD NOT use GET based forms for the submission of sensitive data, because this will cause this data to be encoded in the Request-URI. Many existing servers, proxies, and user agents will log the request URI in some place where it might be visible to third parties. Servers can use POST-based form submission instead

---

Finally, an important consideration when using `GET` for AJAX requests is that some browsers - IE in particular - will cache the results of a `GET` request. So if you, for example, poll using the same `GET` request you will always get back the same results, even if the data you are querying is being updated server-side. One way to alleviate this problem is to make the URL unique for each request by appending a timestamp.

share improve this answer edited Mar 28 '13 at 19:46 answered Aug 13 '10 at 13:42

Justin Ethier
114k • 47 • 210 • 267

94

A `POST`, unlike a `GET`, typically has relevant information in the body of the request. (A `GET` should not have a body, so aside from cookies, the only place to pass info is in the URL.) Besides keeping the URL relatively cleaner, `POST` also lets you send much more information (as URLs are limited in length, for all practical purposes), and lets you send just about any type of data (file upload forms, for example, can't use `GET` -- they have to use `POST` plus a special content type/encoding).

Aside from that, a `POST` connotes that the request will change something, and shouldn't be redone willy-nilly. That's why you sometimes see your browser asking you if you want to resubmit form data when you hit the "back" button.

`GET`, on the other hand, should be *idempotent* -- meaning you could do it a million times and the server will do the same thing (and show basically the same result) each and every time.

share improve this answer edited Jun 30 '13 at 0:30 answered Aug 13 '10 at 13:50

cHao
73.2k • 16 • 124 • 160