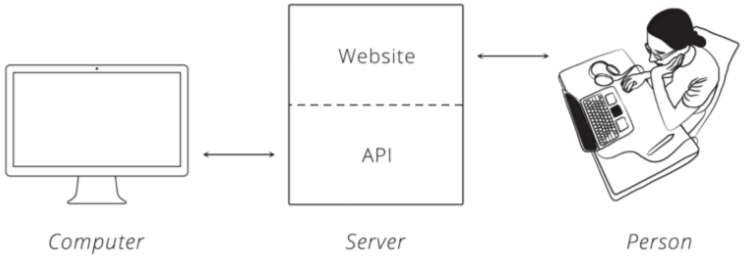
An Introduction to API's

Gonzalo Vázquez Follow
Aug 27, 2015

API stands for application programming interfaces.

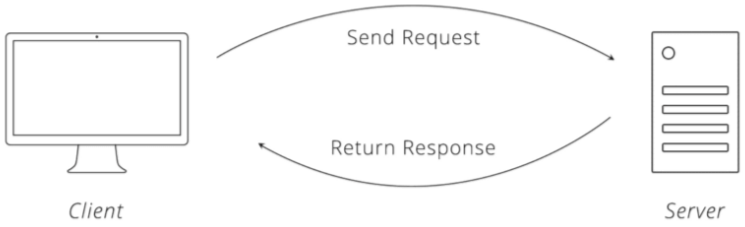
An API is the tool that makes a website's data digestible for a computer. Through it, a computer can view and edit data, just like a person can by loading pages and submitting forms.



When systems link up through an API, we say they are **integrated**. One side the server that serves the API, and the other side the client that consumer the API and can manipulate it.

HTTP Request

Communication in HTTP (Hyper Text Transfer Protocol) center around a concept called the **Request-Response Cycle**.



To make a valid request, the client needs to include four things:

URL (Uniform Resource Locator)

Method

List of Headers

Body

URL

URLs become an easy way for the client to tell the server which things it wants to interact with, called **resources**.

Method

The method request tells the server what kind of action the client wants the server to take in. The four most commonly seen in API's are:

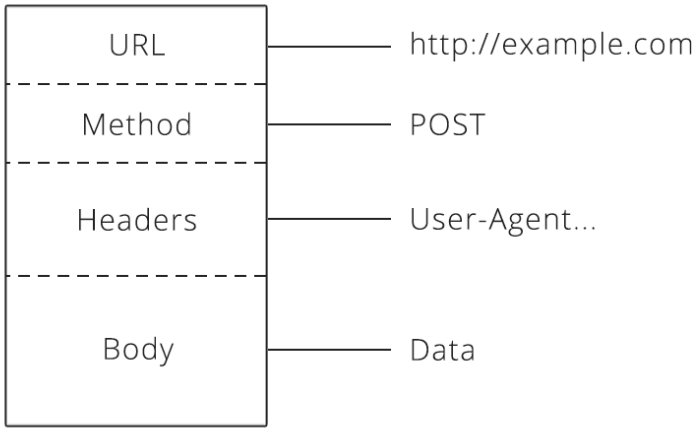
GET—Asks the server to retrieve a resource.
POST—Asks the server to create a new resource.
PUT—Asks the server to edit/update an existing resource.
DELETE—Asks the server to delete a resource.

Headers

Headers provide meta-information about a request.

Body

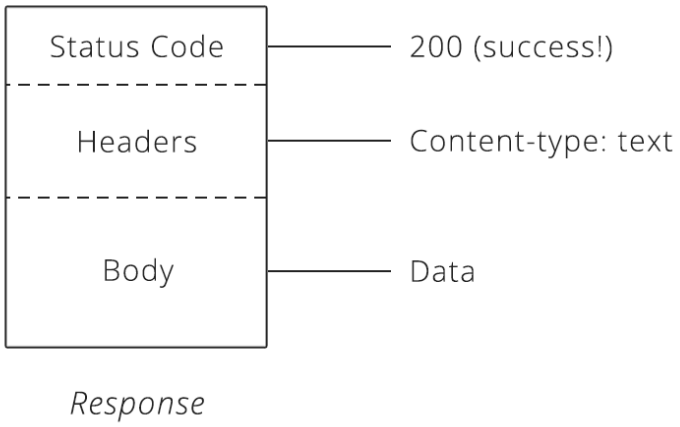
The request body contains the data the client wants to send the server.



Request

HTTP Response

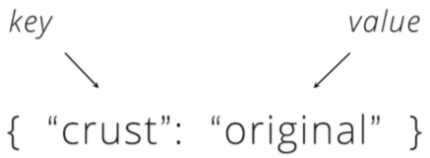
The server responds with a status code. Status code are three-digit numbers.



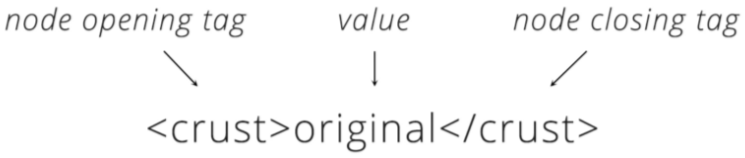
Data Formats

A well-designed format is dictated by what makes the information the easiest for the intended audience to understand. The most common formats found in APIs are JSON and XML.

JSON is very simple format that has two pieces: key and value.



XML provides a few building blocks. The main block is called a node. XML always starts with a root node, inside there are more "child" nodes. The name of the node tells us the attribute of the order and the data inside is the actual details.

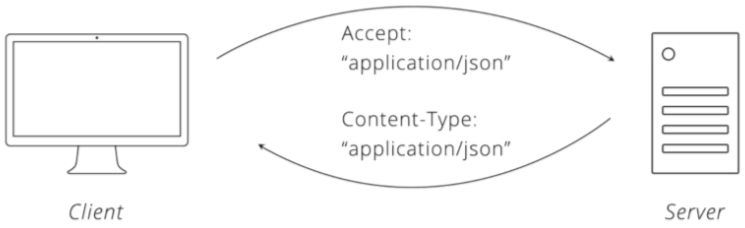


How Data Formats Are Used In HTTP

Using Headers we can tell the server what information we are sending to it and what we expect in return.

Content-type: When the clients send the Content-type its saying what format the data is.

Accept: The Accept header tells the server what data-format it is able to accept.

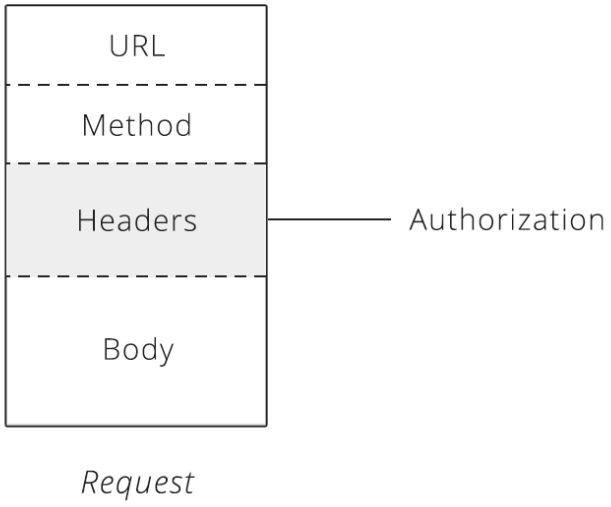


Authentication, Part 1

There are several techniques APIs use to athenticate a client. These are called **authentication schemes**.

Basic Authentication

Also referred as Basic Auth. Basic Auth only requires a user name and password. The client takes these two credentials, converts them to a single value and passes that along in the HTTP header called **Authentication**.



The server compares the Authorization header and compares it to the credential it has stored. If it matches, the server fulfills the request. If there is no match, the server returns status code 401.

API Key Authentication

API Key Authentication is a technique that overcomes the weakness of using shared credentials. by requiring the API to be accessed with a unique key. Unlike Basic Auth, API keys were conceived at multiple companies in the early days of the web. As a result, API Key Authentication has no standard and everybody has its own way of doing it.

The most common approach has been to include it onto the URL(http://example.com?api*key*=*my*secret_key).

Authentication, Part 2

Open Authorization (OAuth) automates the key exchange. OAuth only requires user credentials, then behind the scenes, the client and server are chatting back and forth to get the client a valid key.

Currently there are two versions of OAuth, OAuth1 and OAuth2.

OAuth2

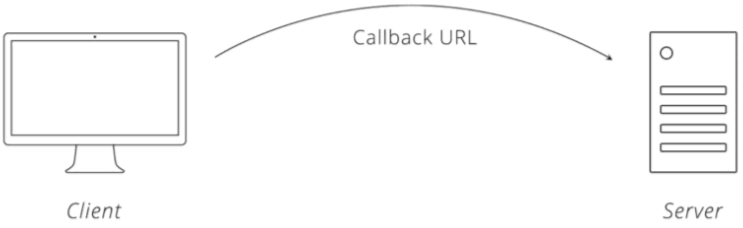
The players involved are:

```
The User - A person that wants to connect to the website
 The Client - The website that will be grated the access to the user's data
 The Server - The website that has the user's data
```

Step 1—User Tells Client to Connect to Server

Step 2—Client Directs User to Server

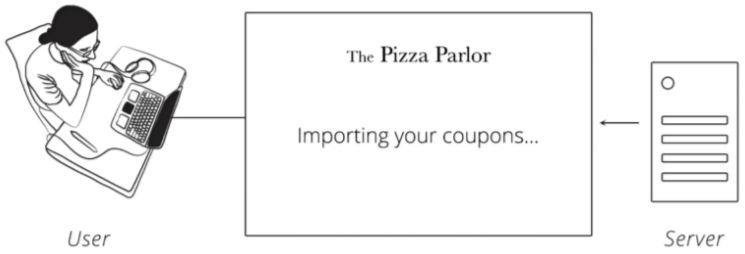
The client sends the user over to the server's website, along with a URL that the server will send the user back to once the user authenticates, called the **callback URL**.



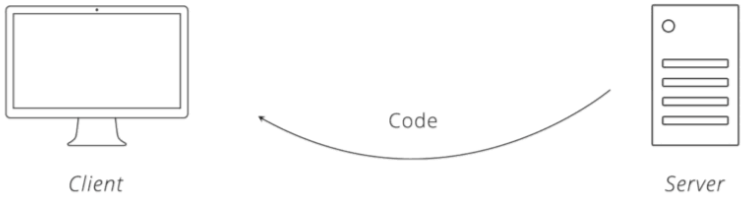
Step 3—User Logs-in to Server and Grants Client Access

With their normal user name and password, the user authenticates with the server.

Step 4—Server Sends User Back to Client, Along with Code

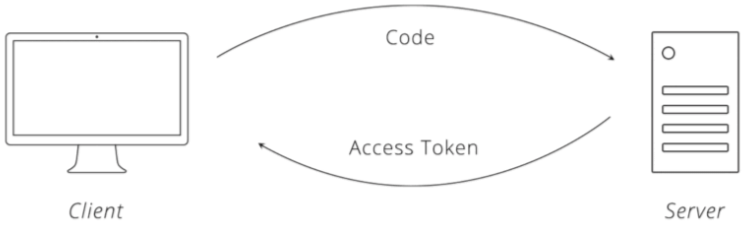


The server sends the user back to the client using the **callback URL**. Hidden in the response is a unique **authorization code** for the client.



Step 5—Client Exchange Code + Secret Key for Access Token

The client takes the authorization code it receives and makes another request to the server. This request includes the client's **secret key**. When the server sees a valid authorization code and a trusted client secret key, it is certain that the client is who it claims. The server responds back with an **access token**.



Step 6—Client Fetches Data from Server.

The access token from Step 5 is essentially another password into the user's account on the server. The client includes the access token with every request so it can authenticate directly with the server.

Client Refresh Token (Optional)

A feature in OAuth 2 is the option to have access tokens expire. The lifespan of a token is set by the server.

Authorization

Authorization is the concept of limiting access. In Step 2, when the user allows the client access, buried in the fine print are the exact permissions the client is asking for. Those permission are called scope.

What makes scope powerful is that is client-based restrictions. OAuth scope allows one client to have permission X and another to have permission X and Y.

API Design

Start with an Architectural Style

The two most common architectures for web-based APIs are **SOAP**, which is an XML-based design that has a standardized structures for requests and responses and REST (Representation State Transfer), is a more open approach, providing lots of conventions, but leaving many decisions to the person designing the API.

Our First Resource

Resources are the nouns of APIs (customers and pizza). These are things we want the world to interact with.

Let's try to design an API for a pizza parlour. For the client to be able to talk pizza with us, we need to do several things:

> Decide what resource(s) need to be available
>
> Assign URLs to the resources
>
> Decide what actions the client should be allowed to perform on those resources
>
> Figure out what pieces of data are required for each action and what format they should be in.

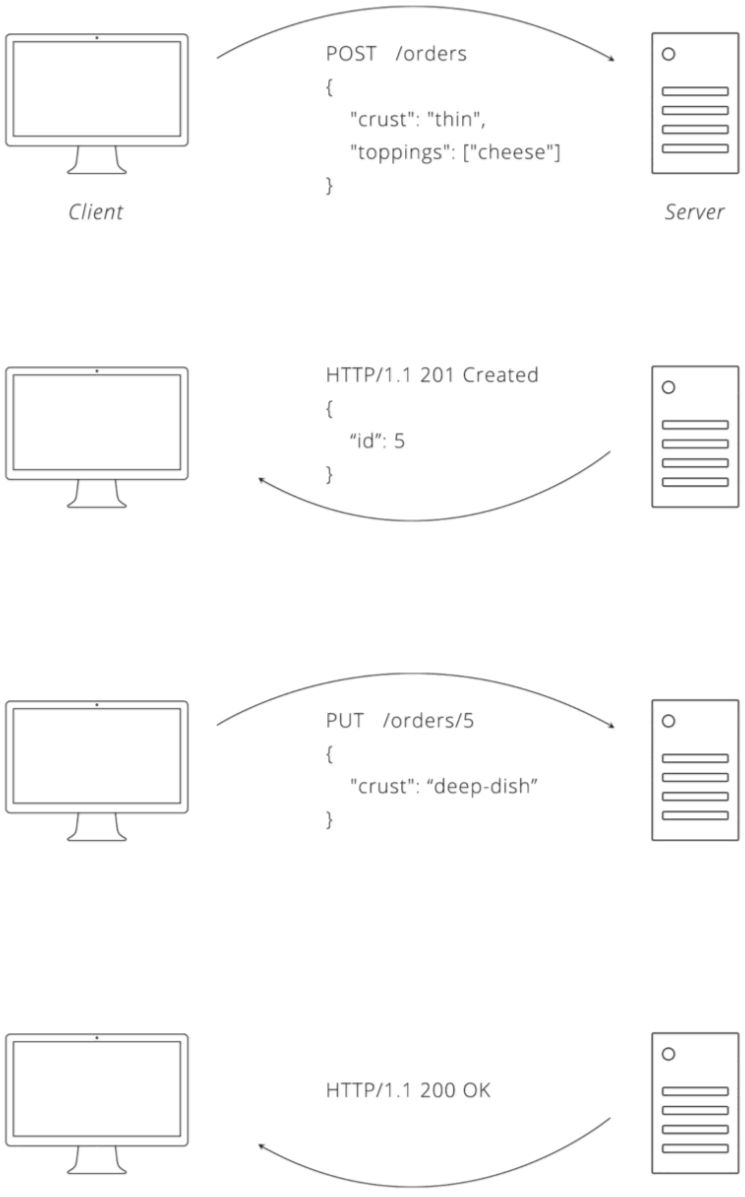
Lets get started with orders. The next step is assigning URLs to the resources. In a typical REST API, a resource will have two URL patterns assigned to it. The first is the plural of the resource name, like **orders/**. The second is the plural of the resources name plus a unique identifier to specify a single resource, like **orders/**, where is the unique identifier for an order. These two parameters make up the first **endpoints** that our API will support. These are called endpoints simply because they go at the end of the URL, as in http://example.com/

Now that we picked our resources and assigned it URLs, we need to decide what actions the client can perform. Following REST conventions, we say that the plural endpoint **(orders/)** is for listing existing orders and creating new ones. The plural with a unique identifier endpoint **(orders/)**, is for retrieving, updating, or canceling a specific order. The client tells the server which action to perform by passing the appropriate HTTP verb (GET, POST, PUT or DELETE) in the request.

Our API now looks like this:

HTTP Header	Endpoint	Action
GET	/orders	List existing orders
POST	/orders	Place a new order
GET	/orders/1	Get details for order #1
GET	/orders/2	Get details for order #2
PUT	/orders/1	Update order #1
DELETE	/orders/1	Cancel order #1

Now we decide what data needs to be exchanged. Coming from our pizza parlour, we can say an order of pizza needs a crust and toppings. We also need to select a data format, lets go with JSON. Here is what an interaction between the client and server might look like using this API:



Linking Resources Together

We want add a new customer resource to track orders by customer. Just like with orders, our customer resource needs some endpoint. Following convention, **/customer** and **/customer/**.

How do we associate orders with customers?

REST practitioners are split on how to solve the problem of associating resources. Some say that the hierarchy should continue to grow, giving endpoints like **/customers/5/orders** for all of customer #5's orders and**/customers/5/orders/3** for customer #5's third order. Others argue to keep things flat by including associated details in the data for a resource. Under this paradigm, creating an order requires a **customer_id** field to be sent with the order details. Both solutions are used by REST APIs in the wild, so it is worth knowing about each.

```
POST /customers/5/orders/3
{
  "crust": "thin",
  "toppings": ["cheese"]
}
```

VS

```
POST /orders/3
{
  "crust": "thin",
  "toppings": ["cheese"],
  "customer_id": 5
}

GET /customers/5
{
  "name": "Brian"
}
```

Searching Data

URLs have another component called the **query string**. For example:http://example.com/orders?key=value. REST APIs use the query string to define details of a search. These details are called **query parameters**. The API dictates what parameters it will accept, and the exact names of those parameters need to be used for them to effect the search. In our case key can allow search by topping and/or curst and you can concatenate the search with an ampersand**(&)**.

For example: http://example.com/orders?topping=pepperoni&crust=thin

Another use of the query string is to limit the amount of data returned in each request. The process of splitting data is called **pagination**. If the client makes a request like **GET /orders?page=2&size=200**, we know they want the second page of results, with 200 results per page, so order 201–400.

Glossary

Server: A powerful computer that runs an API

API: The “hidden” portion of a website that is meant for computer consumption

Client: A program that exchanges data with a server through an API

Request—consists of a URL (http://...), a method (GET, POST, PUT, DELETE), a list of headers (User-Agent...), and a body (data).

Response—consists of a status code (200, 404...), a list of headers, and a body.

JSON: JavaScript Object Notation

Object: a thing or noun (person, pizza order...)

Key: an attribute about an object (color, toppings...)

Value: the value of an attribute (blue, pepperoni...)

Associative array: a nested object

XML: Extensible Markup Language

Authentication: process of the client proving its identity to the server

Credentials: secret pieces of info used to prove the client's identity (username, password...)

Basic Auth: scheme that uses an encoded username and password for credentials

API Key Auth: scheme that uses a unique key for credentials

Authorization Header: the HTTP header used to hold credentials

OAuth: an authentication scheme that automates the key exchange between client and server.

Access Token: a secret that the client obtains upon successfully completing the OAuth process.

Scope: permissions that determine what access the client has to user's data.

SOAP: API architecture known for standardized message formats

REST: API architecture that centers around manipulating resources

Resource: API term for a business noun like customer or order

Endpoint: A URL that makes up part of an API. In REST, each resource gets its own endpoints

Query String: A portion of the URL that is used to pass data to the server

Query Parameters: A key-value pair found in the query string (topping=cheese)

Pagination: Process of splitting up results into manageable chunks

References:

https://zapier.com/learn/apis/chapter-1-introduction-to-apis/