

API BOOTCAMP

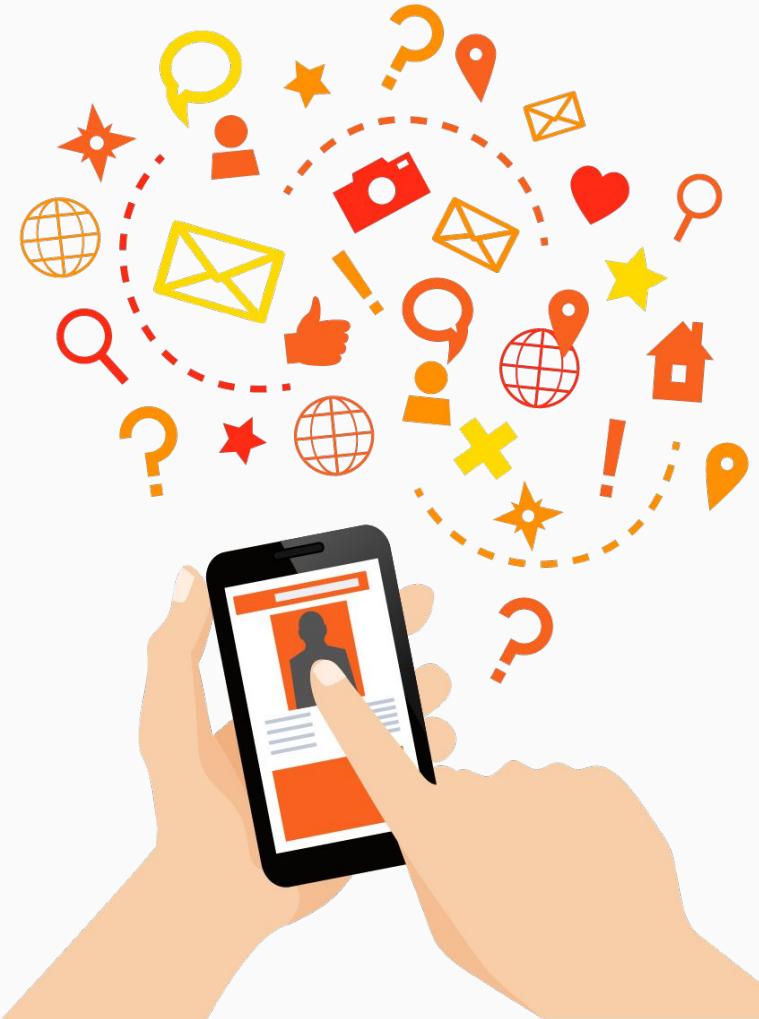
Boost Academy

#>/<
HACK
A BOSS

Noviembre 2022



Benjamín Granados
Developer Evangelist





Benjamín Granados

Developer Evangelist at  twilio

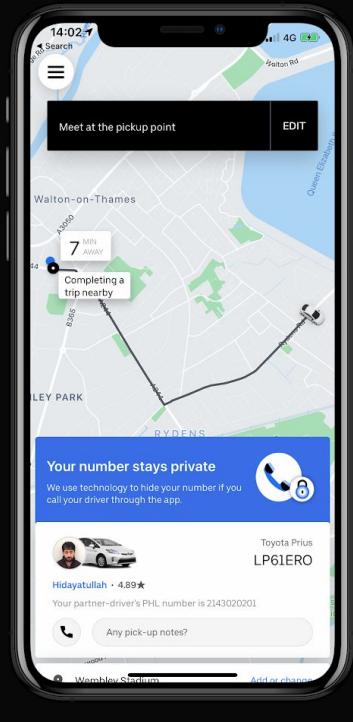
I ❤️ APIs



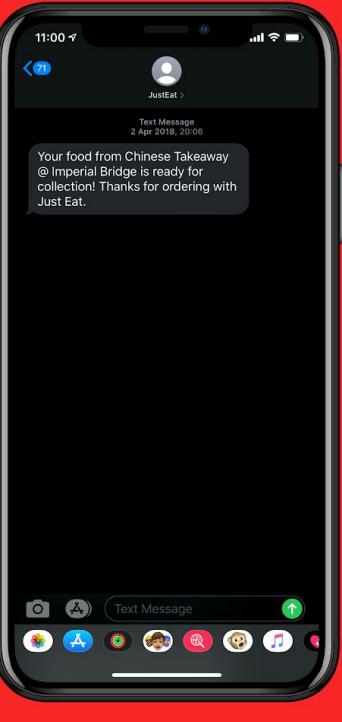
<https://www.linkedin.com/in/benjagranados>

You know Twilio, disruptors use Twilio

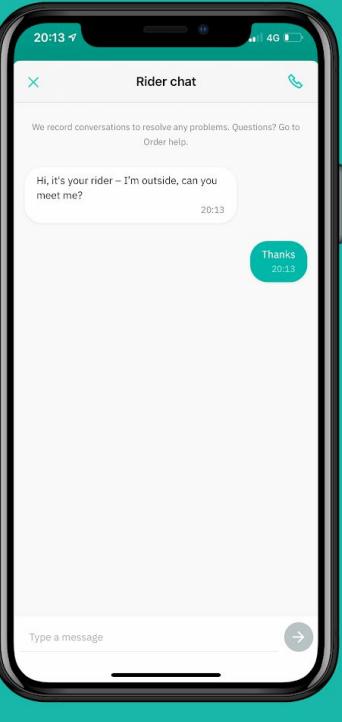
Uber - Private Communications



Just Eat - Order Notifications



Deliveroo - In App Chat

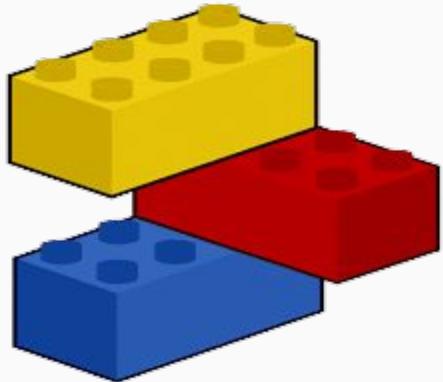


Monzo - 2FA



Airbnb - Host to Guest Email





01
INTRODUCCIÓN

02
API BASICS

03
EXPLORAR APIs

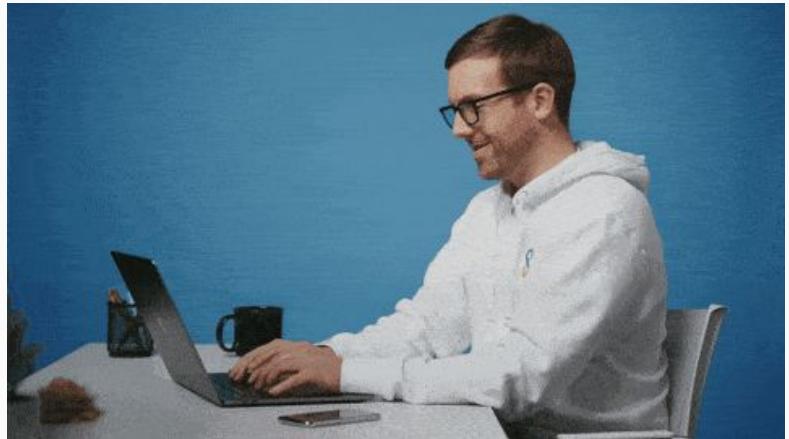
04
USAR APIs

05
DISEÑAR APIs

06
IMPLEMENTAR APIs

07
EJERCICIO FINAL

OI. PRESENTACIÓN



PRESENTACIÓN

Bienvenid@ al API Bootcamp:

- El propósito de este curso es comprender qué son las APIs, porque son tan importantes, aprender cómo explorar APIs y cómo usarlas para construir aplicaciones increíbles. Además aprenderemos a crear APIs y los aspectos principales de su ciclo de vida.
- ¿Por qué este curso? Las APIs están por todas partes y son un elemento fundamental de la mayoría de apps y plataformas que nos rodean. Como futuros desarrolladores espero que este curso os permita entender el concepto e incorporar las APIs como herramientas accesibles para vuestros futuros desarrollos.
- El Bootcamp está inspirado en el curso [APIs for Beginners](#) creado por [Craig Dennis](#) en la plataforma [freecodecamp.org](#).



ANTES DE EMPEZAR

Todo el contenido y los ejemplos empleados en este Workshop están disponibles en este repo:

<https://github.com/benjagm/api-bootcamp-boost-academy>



02. API BASICS



¿QUÉ ES UNA API?

En esta sección cubriremos estos bloques:

- Definición de Interface.
- Definición de API.
- ¿Por qué son tan importantes?
- API Styles.
- APIs REST.

APPLICATION

PROGRAMMING

INTERFACE

DEFINICIÓN DE INTERFACE



DEFINICIÓN DE API

Python String Methods

◀ Previous

Next ▶

Python has a set of built-in methods that you can use on strings.

Note: All string methods returns new values. They do not change the original string.

Method	Description
<code>capitalize()</code>	Converts the first character to upper case
<code>casefold()</code>	Converts string into lower case
<code>center()</code>	Returns a centered string
<code>count()</code>	Returns the number of times a specified value occurs in a string
<code>encode()</code>	Returns an encoded version of the string
<code>endswith()</code>	Returns true if the string ends with the specified value
<code>expandtabs()</code>	Sets the tab size of the string

Python String upper() Method

◀ String Methods

Example

Upper case the string:

```
txt = "Hello my friends"  
x = txt.upper()  
print(x)
```

Try it Yourself ▶

https://www.w3schools.com/python/python_ref_string.asp

Las APIs hacen
la vida más fácil
a los developers

DEFINICIÓN DE API



DEFINICIÓN DE API



APIS EXPOSE
SOMETHING USEFUL



DEVELOPERS WRITE APPS
WHICH CONSUMES APIS

DEFINICIÓN DE API

UNIVERSAL

Como LEGO, las APIs son bloques de construcción universales. En este caso para la creación de Apps.

CONFIGURABLE

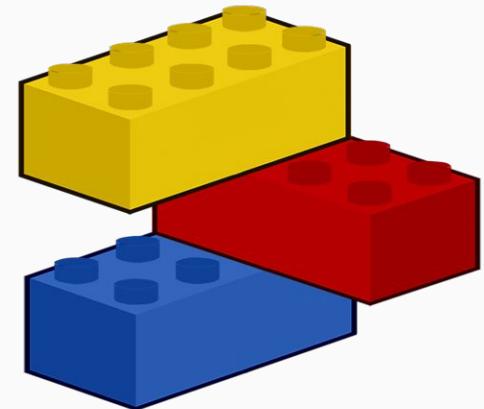
Como LEGO, las APIs dan la opción al usarlas de combinarlas para distintos casos de uso.

FÁCIL DE USAR

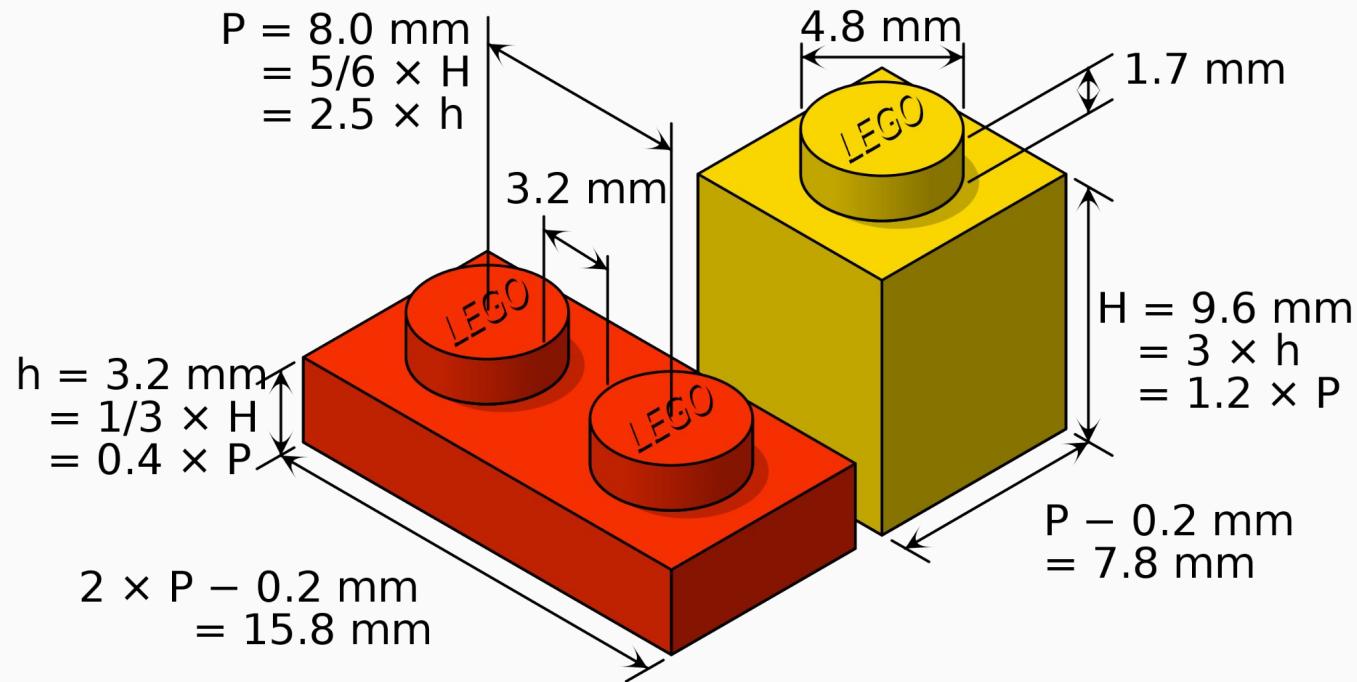
Como LEGO, las APIs tienen sus instrucciones y documentación para que sean fáciles de usar.

INNOVACIÓN

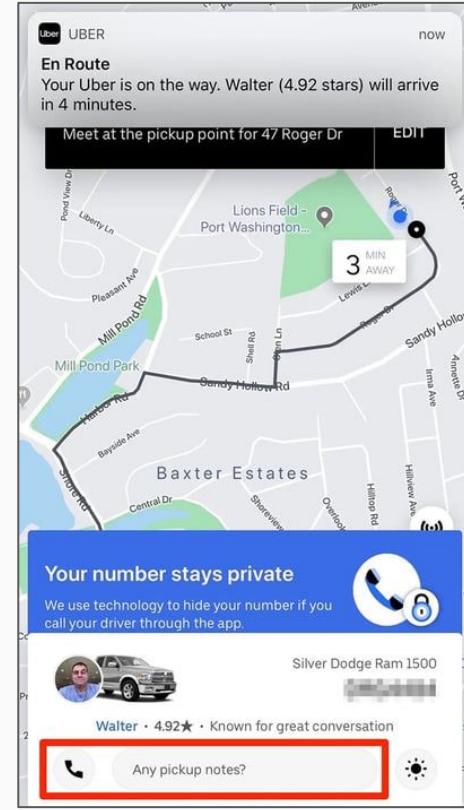
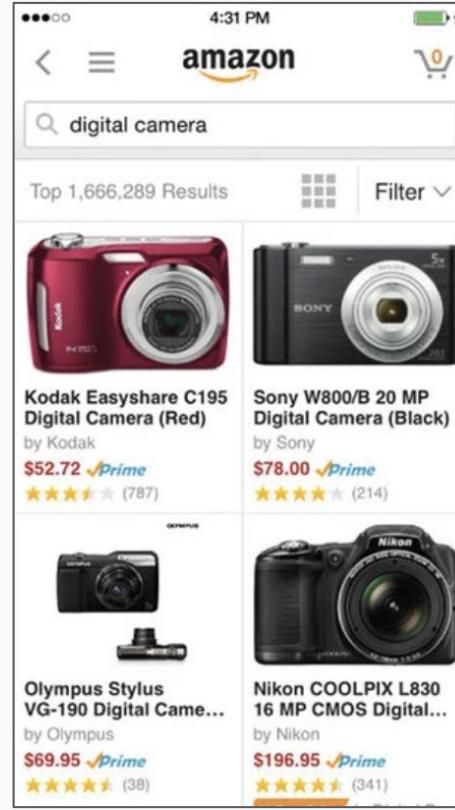
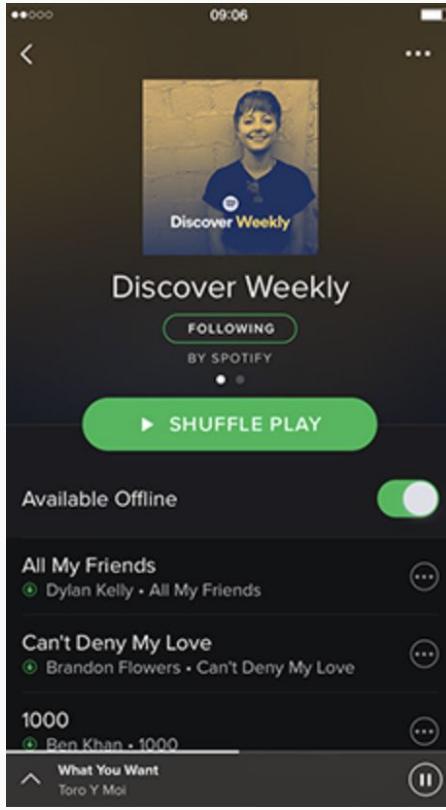
Como LEGO, las APIs permiten crear soluciones de altísimo valor ahorrando mucho tiempo.



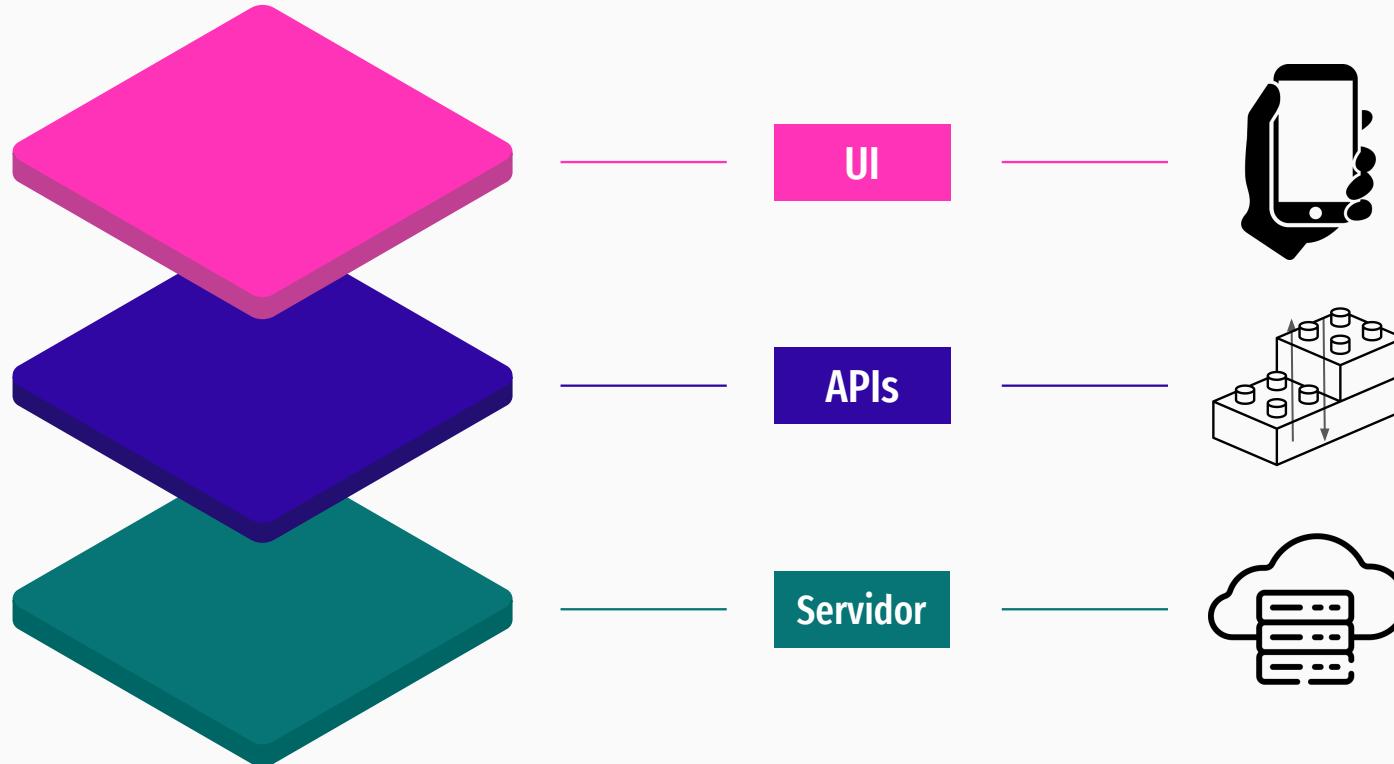
DEFINICIÓN DE API



APIS ESTÁN EN TODAS PARTES

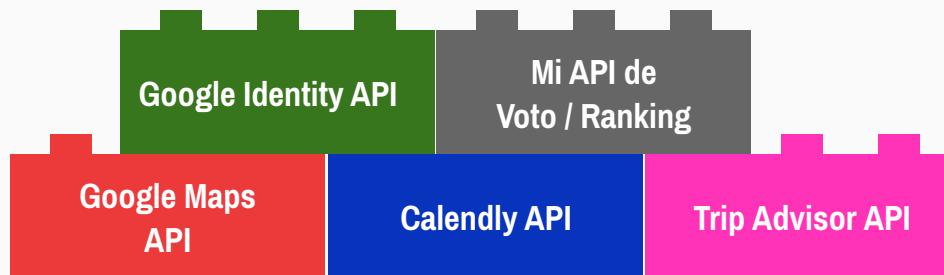


CÓMO FUNCIONAN LAS APIs

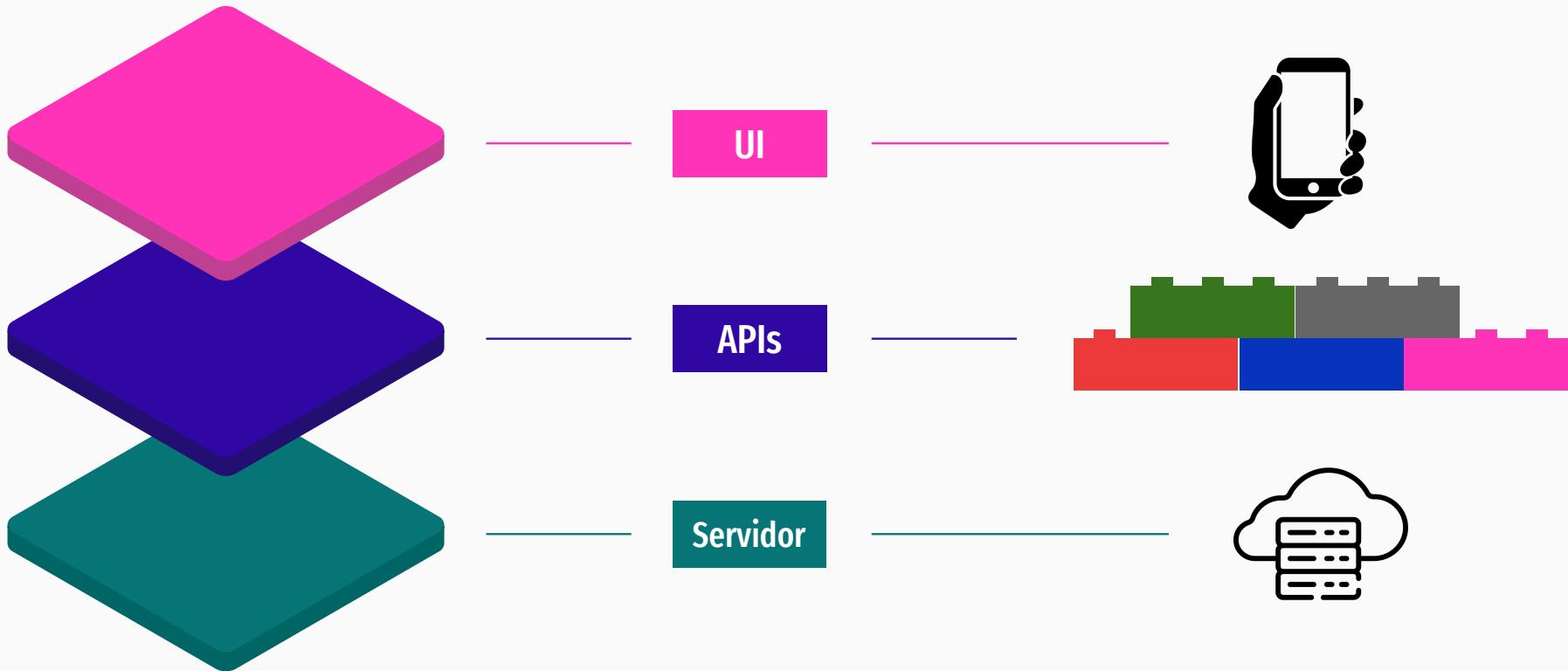


Quiero crear una app que me muestre en el mapa los mejores restaurantes para celíacos y me permita: hacer un reserva, ver reviews de otros clientes y votar para crear un ranking.

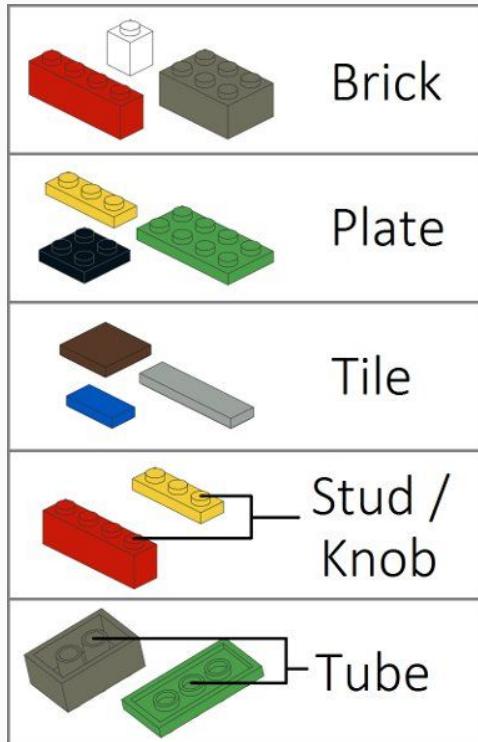
CÓMO FUNCIONAN LAS APIs



CÓMO FUNCIONAN LAS APIs



API STYLES



{REST API}

REST



AsyncAPI



GraphQL

gRPC

gRPC



webhooks

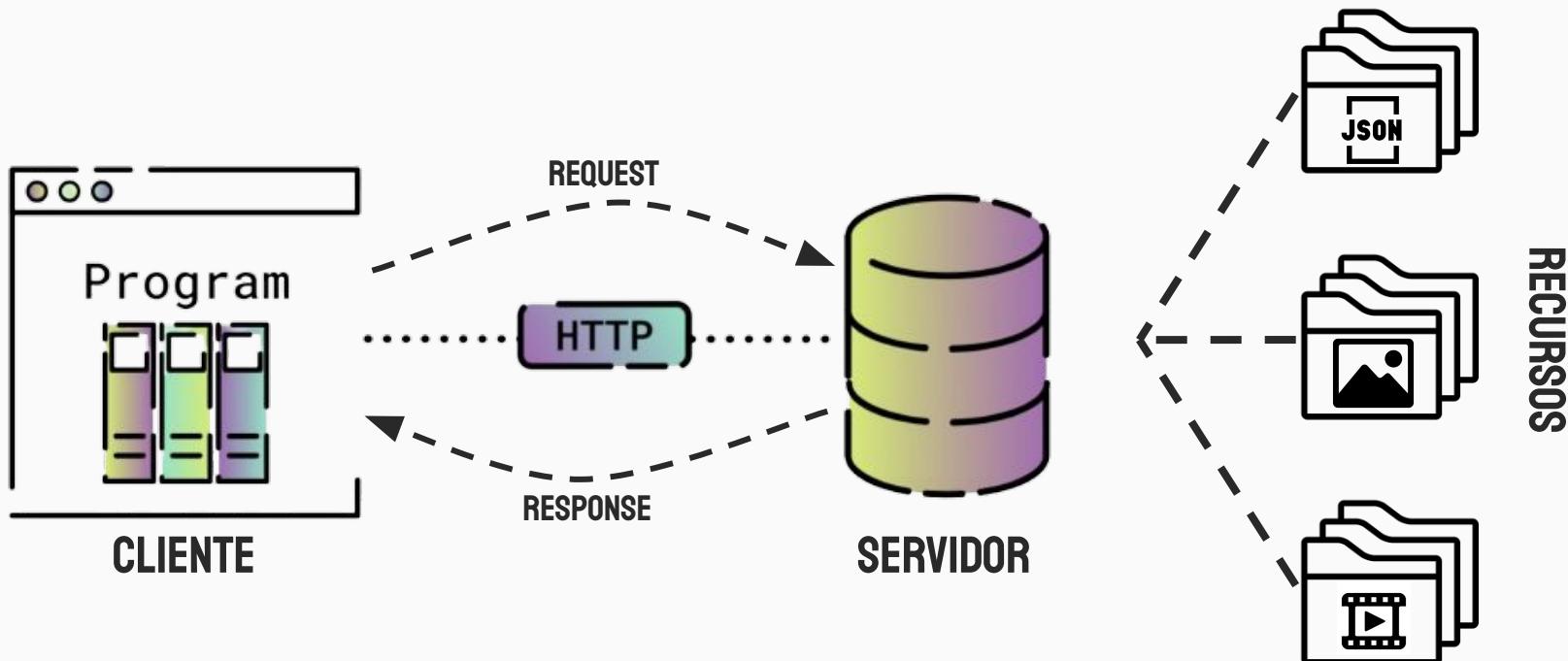
Webhooks

SOAP

Web Services

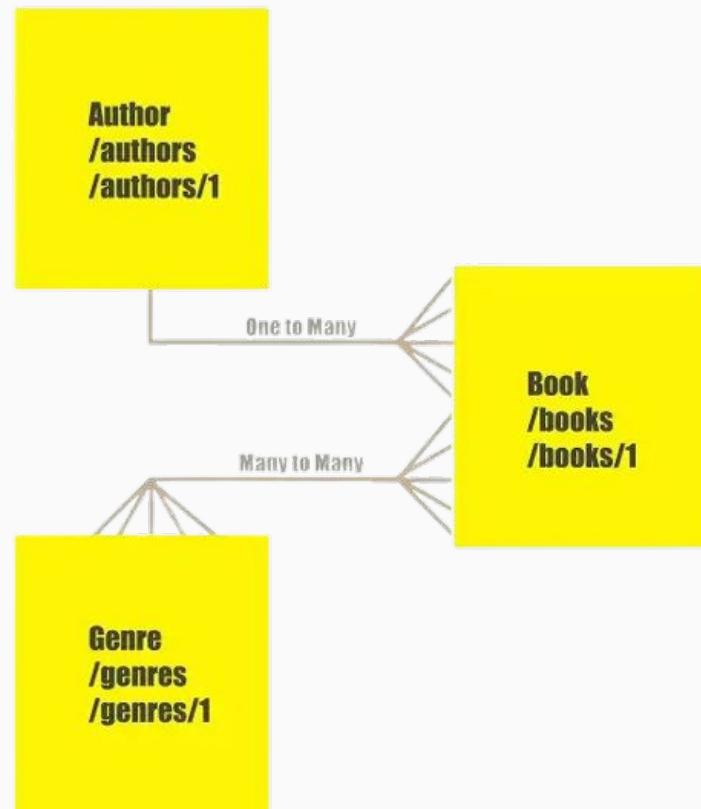


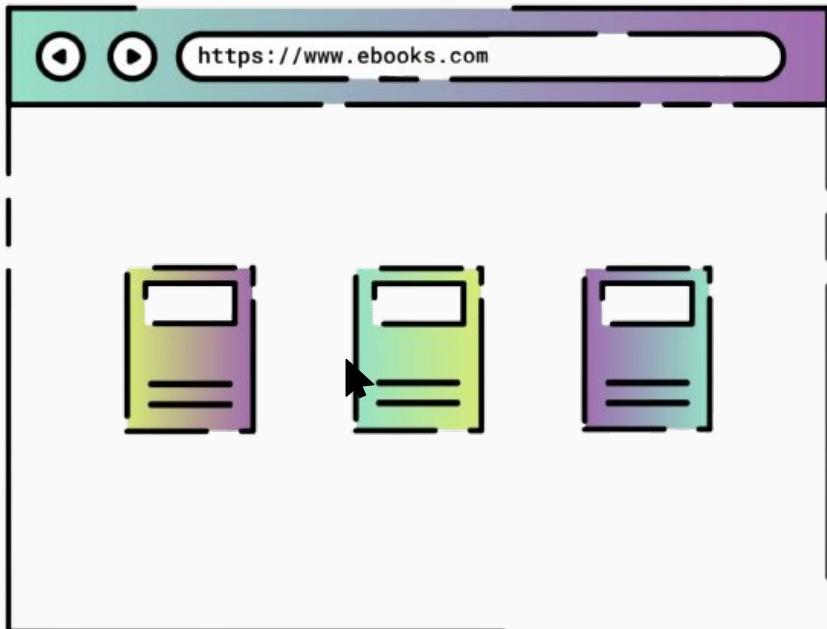
API REST



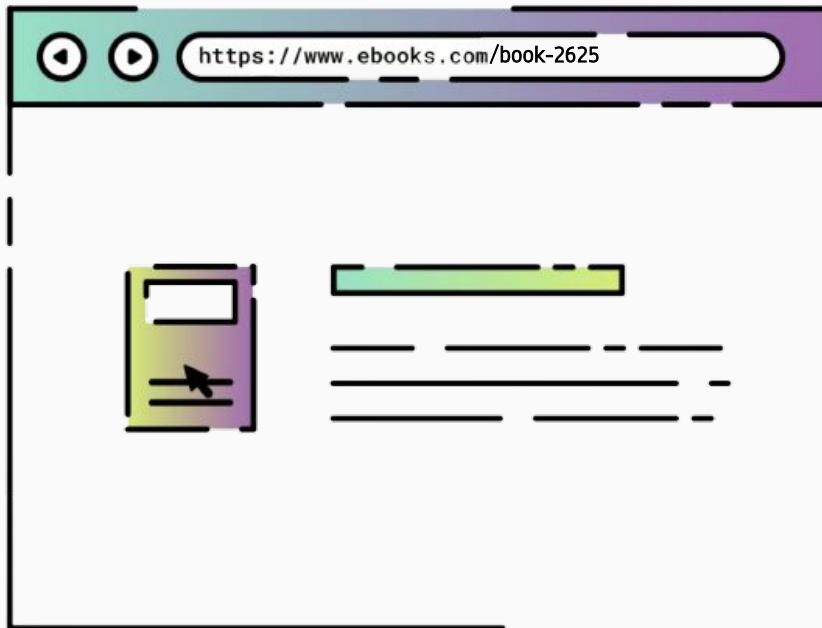


API REST

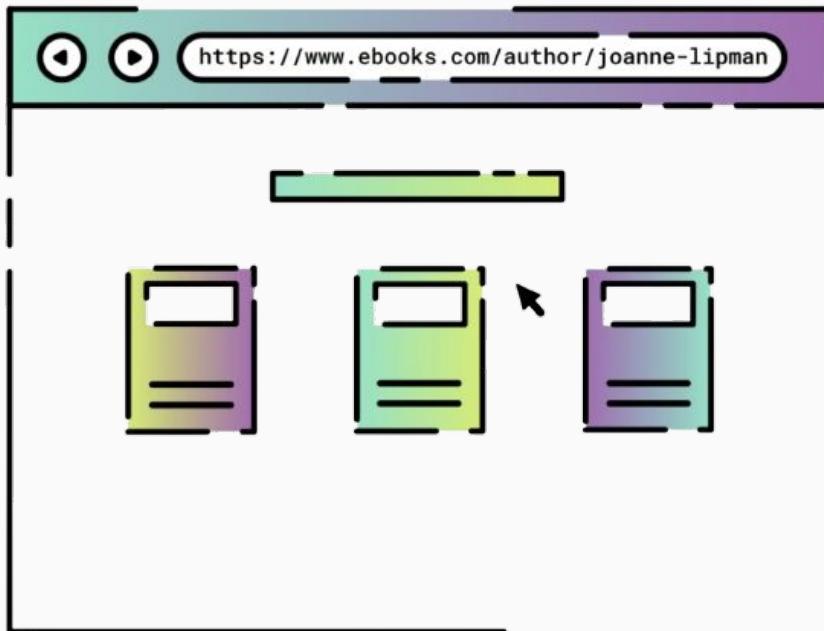




COLECCIÓN DE
RECURSOS
LIBROS



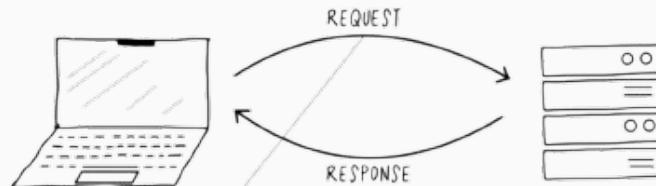
**RECURSO
LIBRO**



**RECURSO
AUTOR
COLECCION LIBROS**

REST STANDS FOR REPRESENTATIONAL STATE TRANSFER

REST APIs OPERATE ON A SIMPLE REQUEST/RESPONSE SYSTEM



CLIENT CAN MAKE A REQUEST
USING HTTP METHODS

THESE METHODS ARE:
GET, POST, PUT, PATCH, DELETE, HEAD,
TRACE, OPTIONS, CONNECT

SERVER RETURNS A RESPONSE WITH AN
HTTP STATUS CODE

POPULAR HTTP STATUS CODE:
EX, 200, 202, 403, 404, 500 ETC

HTTP REQUEST CONTAINS

REQUEST METHOD HTTP HEADERS BODY

HTTP RESPONSE CONTAINS

STATUS CODE HTTP HEADERS RESPONSE BODY

HTTP Verbs	CRUD
GET	Read
POST	Create
PUT	Update
PATCH	
DELETE	Delete



API REST

1a. Method

POST

1b. URL

https://example.com/resource.html

1c. Version

HTTP/1.1

2. Headers

Content-Type: application/json
Custom-Header: goes/here

3. <Empty Line>

4. Message Body

```
{  
    "Key": "Value"  
}
```



API REST

1a. Version

HTTP/1.1

1b. Status Code

404

1c. Status Message

Not Found

2. Headers

Content-Length: 1595

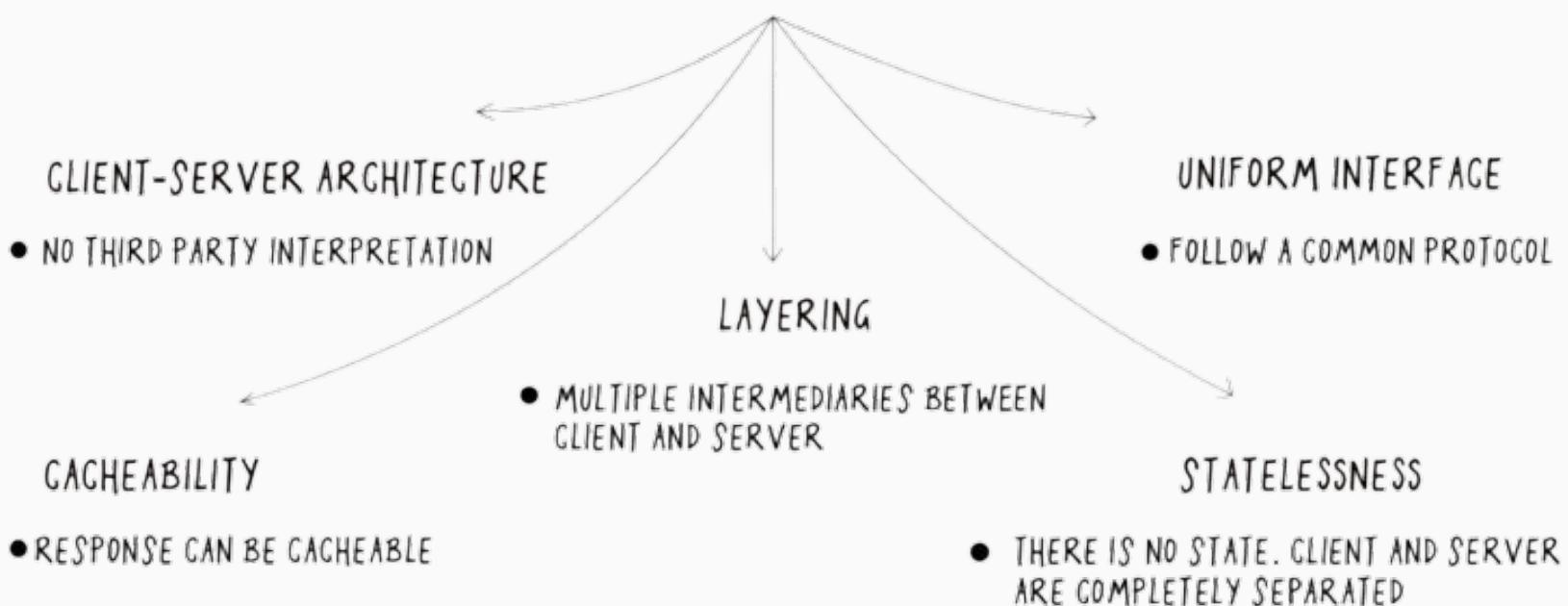
Content-Type: text/html

3. <Empty Line>

4. Message Body

```
<!DOCTYPE html>
<html lang=en>
<title>Error 404 (Not Found)!!1</title>
```

★ REST API CONSTRAINTS ★



- `GET /books`: get all the available books
- `POST /books`: create a new book 
- `GET /books/{id}`: get the details of a book with a specific ID
- `PUT /books/{id}`: update the details of a book with a specific ID 
- `DELETE /books/{id}`: delete a book with a specific ID 
- `GET /authors/{id}`: get the details of an author with a specific ID



API REST

Entrantes

Croquetas de espinacas con perla de mozzarella	10,00€
Croquetas de salmon y bacalao	10,00€
Croquetas de boletus	10,00€
Huevos rotos de coral con picadillo	12,00€
Huevos rotos de coral con vitolas de jamon ibérico	13,00€
Revolvle de garbanzos (boletus, bacalao, pinchos)	12,00€
Tintado de moljeles con verduras crujientes	16,00€
Jamon ibérico de bellota	22,00€
Tempura de verdura con variedad de aliños	12,00€
Morilla de Cantimploras frita	10,00€
Jamón serrano de Segovia	10,00€
Queso de oveja de Segovia	10,00€
Chorizo de la olla	10,00€
Alcachofas sobre crema de ajablanco y chips de bacalao	16,00€
Anchas con pimientos de piquillo	18,00€
Champión portobello relleno de secreto ibérico al PX.	14,00€
Empedrilladas de morilla, manzana y hiedrahuena con salsa aliñada	12,00€
Ferrero Rocher de cochinillo con parmentier de judojo asado	15,00€
Bolita de risotto de boletus y foie con salsa ligera de queso y tocina ibérica ahumada	14,00€
Langostinos fritos con encebollado de tomate y cilantro	16,00€

Ensaladas

Ensalada mixta	9,00€
Ensalada de La Huerta	7,00€
Milhoja lumbada de tomate y calabacin a la plancha con salsa pesto y escamas de queso de oveja	14,00€
Ensaladilla de Txangurro con langostinos y mahonesa clásica	15,00€
Ensalada de presa ibérica escabecheada con vinagreta de frambuesa	15,00€
Ensalada Thai con langostinos a la plancha con dados de mango y crujiente de patata	16,00€
Ensalada de rúcula con tomate y taco de bonito escabechedado al jerez	19,00€

Con el fin de informar a nuestros clientes sobre las posibles alergias e intolerancias de nuestros alimentos, incluimos una relación de nuestros platos y su posible manifestación alérgica. Recomendamos consultar con nuestro personal cualquier duda o intolerancia.

Sopas y plato cuchara

Sopa castellana	7,00€
Judiones	10,00€
Gazpacho (temporada)	9,00€
Crema de calabaza y jengibre	9,00€

Todo Bacalao

Pimientos rellenos de bacalao al ajillo y espinacas con salsa vizcaina	12,00€
Taquitos de bacalao rebozado sobre patata revolconada y salsa suave de longaniza	16,00€
Callos de bacalao con chipirones a la marinera	16,00€
Suprema de bacalao gratinado con el olor y trufas rojas	20,00€
Lomo de bacalao con salsa de ciruelas rojas	20,00€

Pescados

Corvina a la plancha sobre lecho de verduras, algas wakame y salsa de coco	20,00€
Bolita de rodaballo sobre tiras de lechuga y cebolla frita acompañado de dados de mango y calabacín braasado	22,00€
Lubina salvaje con mahonesa de soja, cebolla morada encocinada y patata confitada	24,00€
Merluza a la cazuela con mejillones y alcachofas	20,00€

Carnes y Asados

Cochinillo asado marca de garantía	22,00€
Cordero asado 1/4 (dos personas)	44,00€
Chuletas de lechona a la plancha	20,00€
Lomo de choto a la plancha con salsa a la pimienta y guarnición	18,00€
Solomillo de choto con salsa barbacoa	24,00€
Carne de ternera con pure cremoso de patata, cebollita crujiente y zanahoria	20,00€
Secreto ibérico relleno de queso curado, pimientos al ron y salsa oporto	20,00€
Carcapio de buey con escamas de queso curado de Espíndol, pinchos y helado de chicharrón	18,00€
Lagarto ibérico a la plancha con salsa de setas de temporada y pastel de batata y bacon	20,00€
Deshuesado de cochinillo sobre manzana confitada, cebolla y parmentier de bonito	20,00€
Jarrete de cordero asado a baja temperatura con cremoso de patata	20,00€

Available endpoints

Tweets

Post, retrieve, and engage with Tweets

Get Tweet timelines

Curate a collection of Tweets

Search Tweets: 7 day*

Filter realtime Tweets

Sample realtime Tweets

Users

Manage account settings and profile

Mute, block, and report users

Follow, search, and get users

Create and manage lists

User profile images and banners

Direct Messages

Sending and receiving events

Welcome Messages

Message attachments

Quick Replies

Buttons

Typing indicator and read receipts

Conversation management

Carta

<https://restauranteCasares.com/>

API methods

<https://developer.twitter.com/en/docs/twitter-api/v1>

03. EXPLORANDO APIs



EXPLORANDO APIs

En esta sección cubriremos estos bloques:

- Explorando APIs Online.
 - Spotify for Developers
 - Spotify - Search API {BETA}
- Explorando APIs desde la línea de comandos con CURL.
 - Twilio
 - Twilio Console
- Explorando APIs con Postman.
<https://jsonplaceholder.typicode.com/>

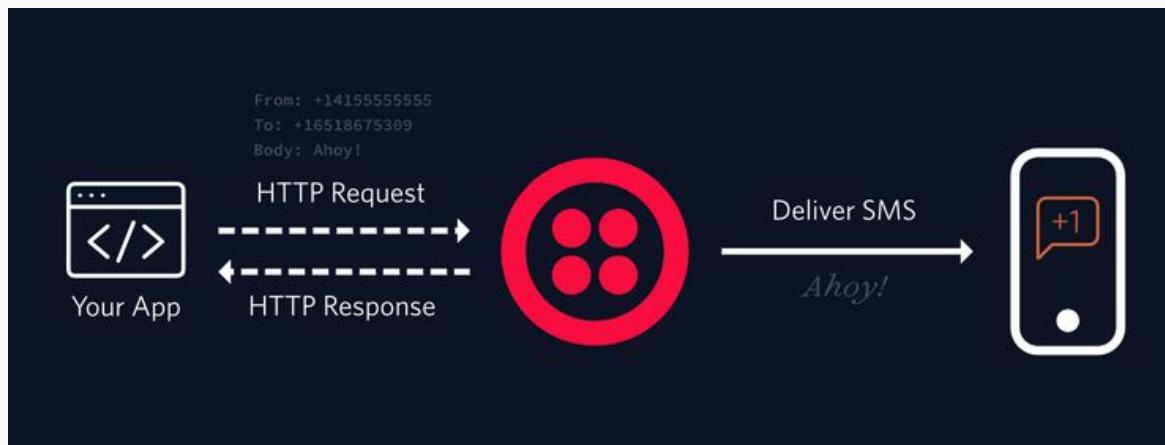
04. USANDO APIs



USANDO APIs

En esta sección cubriremos estos bloques:

- Que es un SDK
- Introducción al SDK de Twilio Messaging para Python.
- Aplicación de ejemplo en la que implementar un envío de SMS usando la API y el SDK de Twilio.
<https://www.twilio.com/docs/sms/quickstart/python>



QUE ES UN SDK

DECLARAR UN CLIENTE HTTP PARA CADA LLAMADA CON TODO LO QUE ACARREA ES CANSINO

```
import python_http_client

global_headers = {"Authorization": "Bearer XXXXXXXX"}
client = Client(host='base_url', request_headers=global_headers)
query_params = {"hello":0, "world":1}
request_headers = {"X-Test": "test"}
data = {"some": 1, "awesome": 2, "data": 3}
response = client.your.api._(param).call.post(request_body=data,
                                                query_params=query_params,
                                                request_headers=request_headers)

print(response.status_code)
print(response.headers)
print(response.body)
```



QUE ES UN SDK

```
from twilio.rest import Client

# put your own credentials here
account_sid = "ACc34dc91ef3e06c3ab9090d4f15f50948"
auth_token = "6118b5af92d39b1deb98abe05fd23725"

client = Client(account_sid, auth_token)

message= client.messages.create(
    to="use your number given when registration",
    from_="use the free number ",
    body="This is the ship that made the Kessel Run in fourteen parsecs?"
)
```

USANDO UN SDK

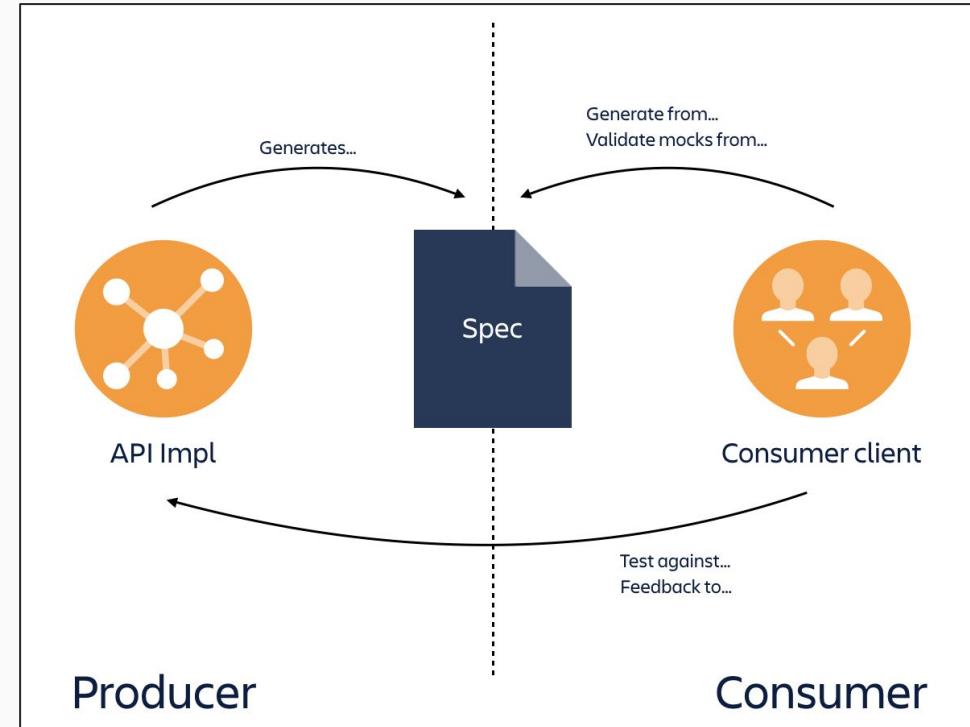
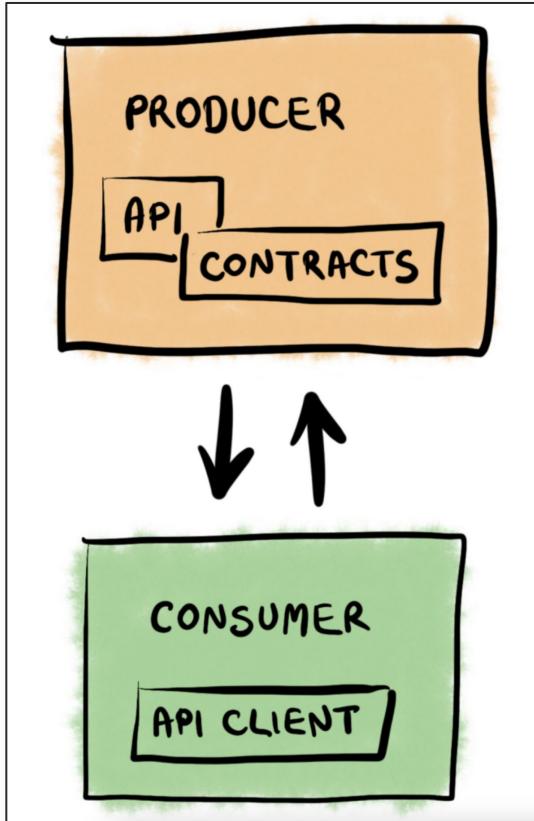
Aplicación de ejemplo en la que implementar un envío de SMS usando la API y el SDK de Twilio.



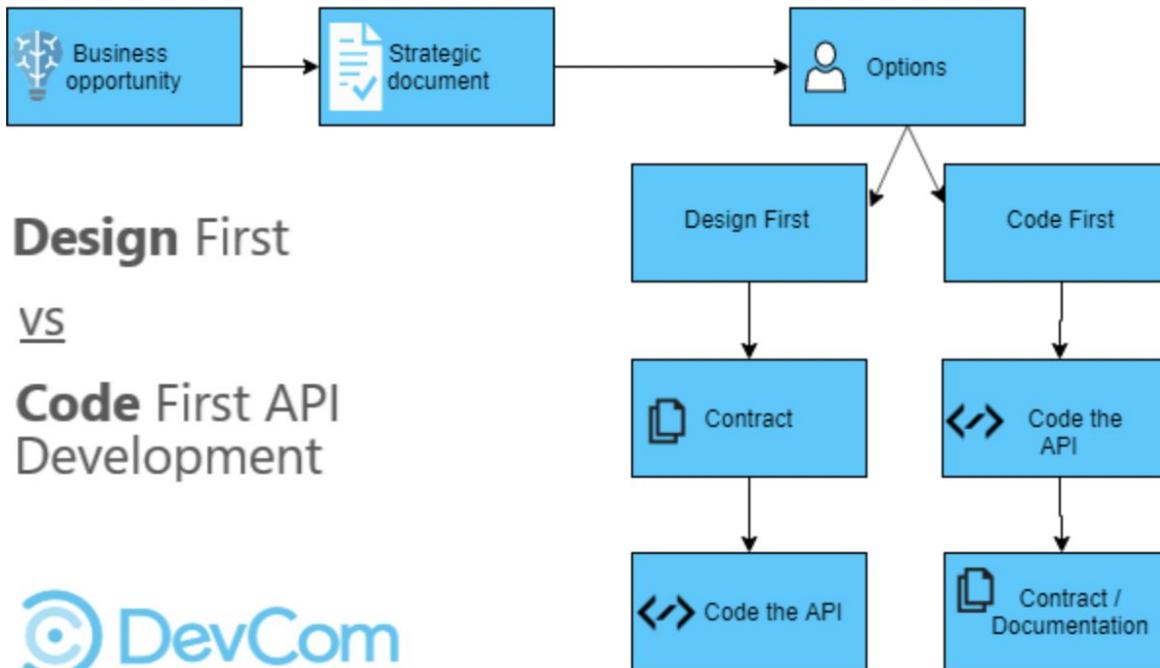
05. DISEÑAR APIS



PRODUCERS VS CONSUMERS



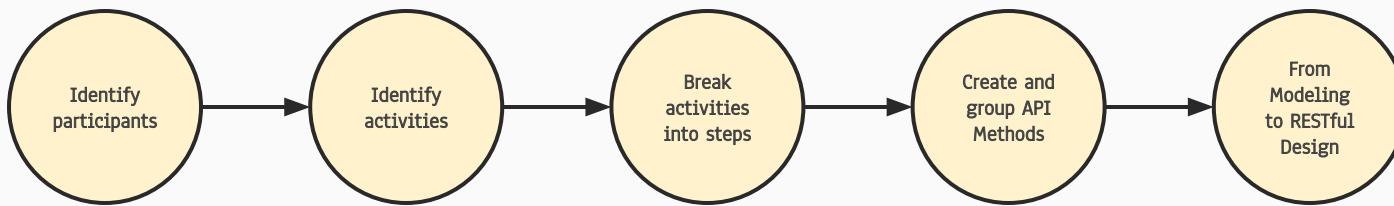
API-FIRST VS CODE-FIRST



API first 
VS
Code first 



API MODELING PROCESS



IDENTIFY PARTICIPANTS

Participant	Location	Description
System Admin	Internal	Responsible for managing accounts, gathering system-wide metrics
Customer	External	A customer account
Manager	External	A manager oversees and reports on the activity of one or more customer accounts
...

IDENTIFY ACTIVITIES

Activity	Participant(s)	Description
Place Order	Customer	A customer places and pays for an order
Lookup Order Status	Customer, Manager, Customer Support	Lookup an order status
Cancel Order	Manager, Customer Support	Cancel an order and refund payment

BREAK ACTIVITIES INTO STEPS

Activity	Step	Participant(s)	Description
Place Order	List Items	Customer	List/search for store items
Place Order	Add Item To Cart	Customer	Add an item to the customer's cart
Place Order	Remove Item From Cart	Customer	Remove an item from the customer's cart
Place Order	Clear Cart	Customer	Remove all items from the customer's cart
Place Order	View Cart	Customer	View the current cart and total
Place Order	Checkout	Customer	Create an order from the contents of the cart

Activity	Step	Participant(s)	Description
Lookup Order Status	List Orders For Customer	Customer, Manager, Customer Support	List a customer's orders from most recent
Lookup Order Status	Obtain order status	Customer, Manager, Customer Support	View the status of an order
Lookup Order Status	Locate Customer	Manager, Customer Support	Find a customer by name, ID, phone
...

CREATE AND GROUP API METHODS

Orders API

The Orders API will manage the user's orders and allow customer support staff and managers to view order status:

Method	Participant(s)	Description
List Orders For Customer	Customer, Manager, Customer Support	List a customer's orders from most recent
Obtain order status	Customer, Manager, Customer Support	View the status of an order

Customers API

The Customers API will allow customer support staff and managers to lookup a customer by name, prior to viewing an order status:

Method	Participant(s)	Description
Locate Customer	Manager, Customer Support	Find a customer by name, ID, phone

Items API

The Items API will manage what is available from the store:

Method	Participant(s)	Description
List Items	Customer	List/search for store items

Cart API

The Cart API will manage a user's cart and the checkout process:

Method	Participant(s)	Description
Add Item To Cart	Customer	Add an item to the customer's cart
Remove Item From Cart	Customer	Remove an item from the customer's cart
Clear Cart	Customer	Remove all items from the customer's cart
View Cart	Customer	View the current cart and total
Checkout	Customer	Create an order from the contents of the cart

FROM MODELING TO RESTFUL DESIGN API DESIGN PRINCIPLES

- ARQUETIPOS DE RECURSOS
- HEADERS Y PAYLOADS
- URIS
- PARAMETROS
- MÉTODOS HTTP
- CÓDIGOS DE RESPUESTA

ARQUETIPOS RECURSOS



COLLECTION
***STORE**

<http://myapi.com/books>



DOCUMENT

<http://myapi.com/books/1>



CONTROLLER

<http://myapi.com/me/unlock>

<http://uniknow.github.io/AgileDev/site/0.1.9-SNAPSHOT/parent/rest/resource-archetypes.html>



HEADERS Y PAYLOADS

1a. Method

POST

1b. URL

https://example.com/resource.html

1c. Version

HTTP/1.1

1a. Version

HTTP/1.1

1b. Status Code

404

1c. Status Message

Not Found

2. Headers

Content-Type: application/json
Custom-Header: goes/here

2. Headers

Content-Length: 1595
Content-Type: text/html

3. <Empty Line>

3. <Empty Line>

4. Message Body

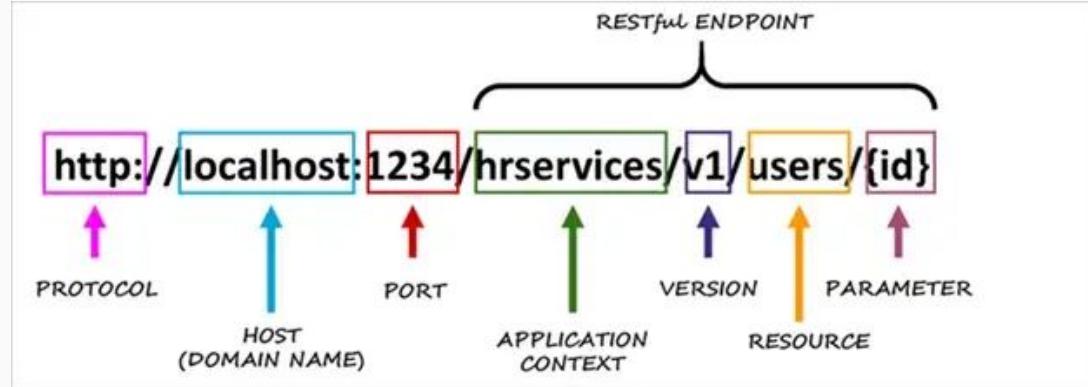
```
{  
  "Key": "Value"  
}
```

4. Message Body

```
<!DOCTYPE html>  
<html lang=en>  
<title>Error 404 (Not Found)!!1</title>
```

URIS

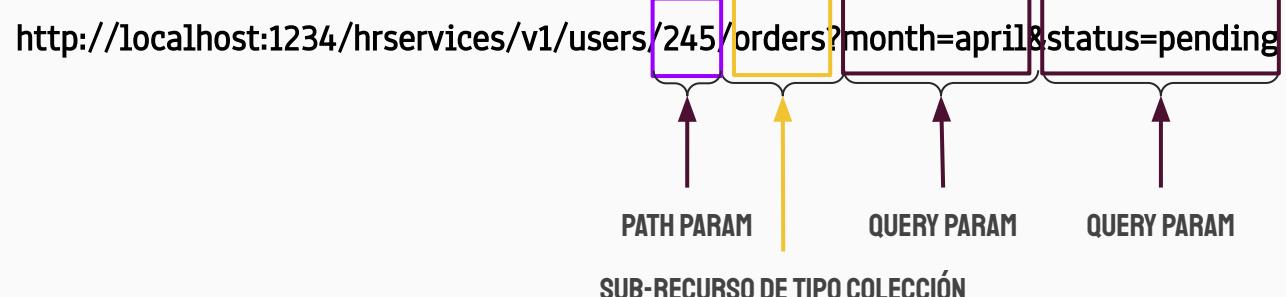
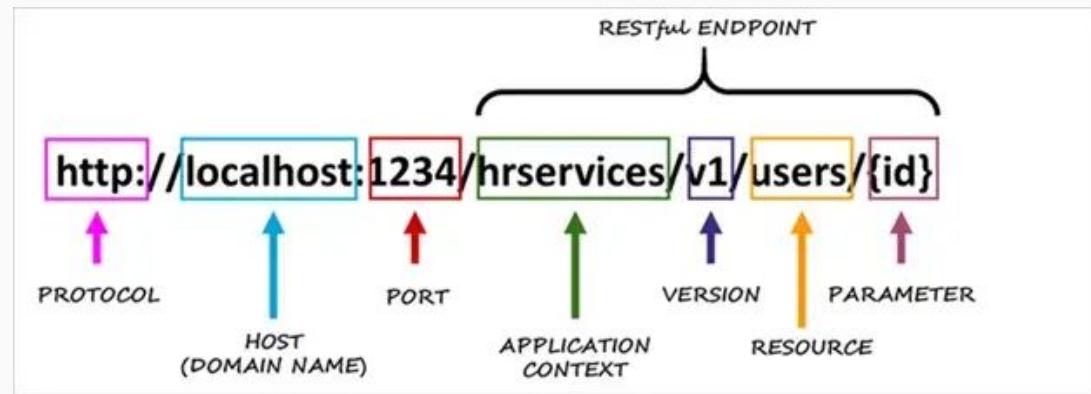
RFC 3986



PATH PARAMETERS VS QUERY PARAMETERS

Los parámetros PATH se usan para identificar recursos o sub-recursos.

Los parámetros de QUERY son parámetros de entrada que se pasan a las operaciones para filtrar, ordenar o informar otras opciones.



METODOS HTTP

HTTP Verbs	CRUD
GET	Read
* POST	Create
PUT	Update
PATCH	
DELETE	Delete

* POST también se usa para los endpoints de tipo **controller**:

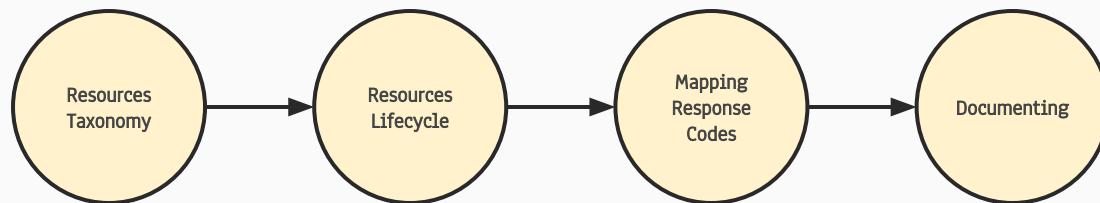
POST <http://api.example.com/song-management/users/{id}/playlist/play>

CÓDIGOS DE RESPUESTA

HTTP Status Codes

Success	Client error	Server error
Level 200 (Success)		
200 : OK	Level 400	Level 500
201 : Created	400 : Bad Request	500 : Internal Server Error
203 : Non-Authoritative Information	401 : Unauthorized	503 : Service Unavailable
204 : No Content	403 : Forbidden	501 : Not Implemented
	404 : Not Found	504 : Gateway Timeout
	409 : Conflict	599 : Network timeout
		502 : Bad Gateway

FROM MODELING TO RESTFUL DESIGN



RESOURCES TAXONOMY

Relationship	Description
Independent	The resources exist stand alone without the other's existence, but may reference each other
Dependent	One resource cannot exist without the existence of the parent resource
Associative	The resources exist independently, however their relationship contains or requires additional properties to describe it

Resource Name	Related Resources
Items	-
Cart	<i>has Items</i>
Cart Items	<i>belongs to a Cart, has Items</i>
Orders	<i>has Items</i>
Customers	<i>has Orders</i>

RESOURCES LIFECYCLE

Modeling Actions	Typical HTTP Verb
"List", "Search", "Match", "View All"	GET resource collection
"Show", "Retrieve", "View"	GET resource instance
"Create", "Add"	POST create a new resource
"Replace"	PUT update a resource collection
"Update"	PUT update a resource instance
"Delete All", "Remove All", "Clear", "Reset"	DELETE delete a resource collection
"Delete", "Remove"	DELETE delete a resource instance
<other verbs>	POST custom action on a resource collection or instance
Capability	HTTP Verb
List resources	GET /{collection name}
Get a resource	GET /{collection name}/{id}
Create a resource	POST /{collection name}
Update a resource	PUT /{collection name}/{id}
Delete a resource	DELETE /{collection name}/{id}
Replace all resources	PUT /{collection name}
Delete all resources	DELETE /{collection name}

Items API

Endpoint	Description
GET /items	List all available items from the store

Cart API

Endpoint	Description
GET /cart	List details about the cart, including total cost and all items/quantities in our cart
POST /cart/items	Add an item to the cart with a specific quantity
PUT /cart/items/{itemId}	Update an item quantity already in the cart
DELETE /cart/items	Clear the contents of the cart
DELETE /cart/items/{itemId}	Remove an item from the cart

Orders API

Endpoint	Description
GET /orders	List recent orders for the current customer
GET /orders/{orderId}	Order details, including the current status

Customers API

Endpoint	Description
GET /customers{?customerId,name}	List customers with optional filters

RESPONSE CODES

Common Response Codes For All Requests

There are a variety of response codes that are common to most requests, no matter what kind of request being issued:

Type	Condition	Recommended Response Codes
Success	Request was successful	200 OK
Error	User not authenticated	401 Unauthorized
Error	User not authorized for endpoint request, often due to missing security privileges	403 Forbidden

Common Error Response Codes For GET

Since a GET request for a collection or resource is straight-forward, there aren't too many status codes that handle other conditions. If you find yourself using other status codes, it might be an indication that you are using a GET request for some other kind of processing.

Type	Condition	Recommended Response Codes
Error	Resource not found	404 Not Found
Error	Filter parameters provided were not valid	400 Bad Request

RESPONSE CODES

Common Response Codes For POST

Type	Condition	Recommended Response Codes
Success	A custom action or processing request was successful	200 OK
Success	Created resource successfully	201 Created
Success	Long-running request has been accepted	202 Accepted
Error	Fields were not valid or required fields were missing	400 Bad Request

Common Error Response Codes For DELETE

Type	Condition	Recommended Response Codes
Success	Resource deleted	204 No Content
Error	Resource not found	404 Not Found

Common Error Response Codes For PUT

Type	Condition	Recommended Response Codes
Success	Updated resource successfully	200 OK or 204 No Content
Error	Fields were not valid or required fields were missing	400 Bad Request
Error	Resource not found	404 Not Found

RESPONSE CODES

Orders API

Endpoint	Description	Response Codes
GET /orders	List recent orders for the current customer (or by customer id if permitted)	200 OK 404 Not Found (Customer not found by id)
GET /orders/{orderId}	Order details, including the current status	200 OK 404 Not Found (Order not found by id)

Customers API

Endpoint	Description	Response Codes
GET /customers	List customers with optional filters	200 OK 404 Not Found (Customer not found by id)

Items API

Endpoint	Description	Response Codes
GET /items	List all available items from the store	200 OK

Cart API

Endpoint	Description	Response Codes
GET /cart	List details about the cart, including total cost and all items/quantities in our cart	200 OK
POST /cart/items	Add an item to the cart with a specific quantity	201 Created 401 Unauthorized 400 Bad Request (item already in cart)
PUT /cart/items/{itemId}	Update an item quantity already in the cart	200 OK 401 Unauthorized 404 Not found (item not found in cart)
DELETE /cart/items	Clear the contents of the cart	204 No Content 401 Unauthorized
DELETE /cart/items/{itemId}	Remove an item from the cart	204 No Content 401 Unauthorized 404 Not found (item not found in cart)

EJEMPLOS

CREAR RECURSOS

```
POST /projects
Content-Type: application/json
Accept: application/json

{
  "name": "My cool project",
  "description": "Bla bla .."
}
```

```
HTTP/1.1 201 Created
Location: /projects/123
Content-Type: application/json

{
  "id" : 123,
  "name": "My cool project",
  "description": "Bla bla .."
}
```

RECUPERAR RECURSOS

```
GET /paintings
Accept: application/json
```

```
HTTP/1.1 200 (Ok)
Content-Type: application/json

[
  {
    "id": 1,
    "name": "Mona Lisa"
  }, {
    "id": 2
    "name": "The Starry Night"
  }
]
```

EJEMPLOS

MODIFICAR RECURSOS

```
PUT /products/345
Content-Type: application/json

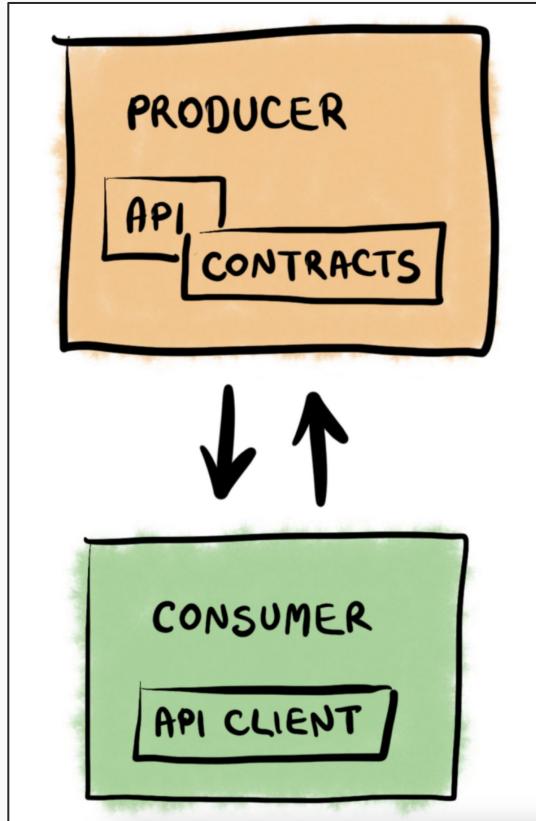
{
  "name": "Cool Gadget",
  "description": "Looks cool",
  "price": "24.99 USD"
}
```

BORRAR RECURSOS

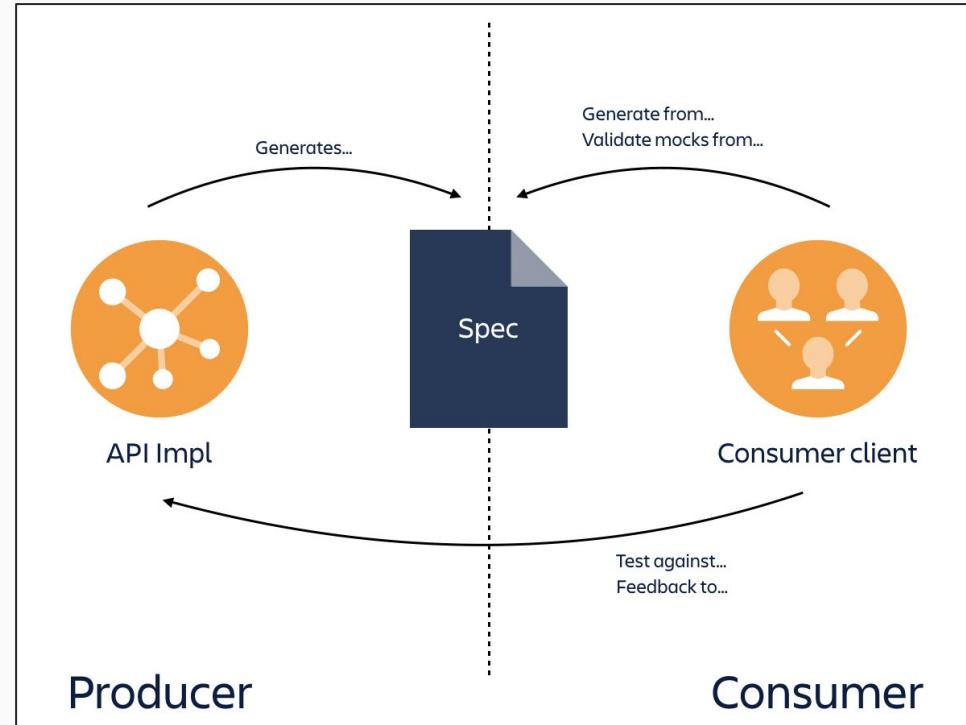
```
DELETE /artists/123
```

```
HTTP/1.1 204 (No content)
```

```
HTTP/1.1 204 (No content)
```



API CONTRACT / SPECIFICATION





OPENAPI SPECIFICATION

OpenAPI es un formato para describir web APIs, por lo tanto está totalmente ligado al protocolo HTTP.

- Permite definir un contrato común y público entre servicios : proveedor - consumidor.
- ES Human and machine readable.
- Independiente del lenguaje/framework/tecnología.
- Permite usar los formatos YAML o JSON.

YAML

```
openapi: 3.0.0
info:
  title: Sample API
  description: Multiline/single-line description in Common Mark or HTML.
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Optional description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional description, e.g. Internal staging server

paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description (Common Mark/HTML).
      responses:
        '200': # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```



OPENAPI SPECIFICATION

- La sección **info** contiene información de alto nivel de la API como: Titulo descripción y versión.
- El objeto **servers** proporciona información de conectividad.
- El objeto **security** declara qué mecanismos de seguridad se pueden utilizar.
- El objeto **paths** describe los métodos a los que se puede acceder
- El objeto **components**, que contiene un conjunto de objetos/estructuras de datos reutilizables.
- **externalDocs**: Documentación externa adicional.

OpenAPI v3.0

info

servers

security

paths

tags

externalDocs

components



COMPONENTS

OpenAPI v3.0

info

servers

security

paths

tags

externalDocs

components

```
openapi: 3.0.0
info:
  title: Swagger Petstore
paths:
  /pets/{petId}:
    get:
      summary: Info for a specific pet
      parameters:
        - name: petId
          in: path
          description: The id of the pet
          schema:
            type: string
      responses:
        200:
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Pet"
components:
  schemas:
    Pet:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
```



INFO

The `info` object contains information about the API you're documenting. This includes things like the `title`, `version` of the API, the API's `description`, links to its knowledge base (`documentation`), and its terms-of-service (`tos`).

According to the specification, this is what an [info object](#) should look like:

Name	Type	Description
<code>title</code>	<code>string</code>	<i>REQUIRED.</i> The title of the API.
<code>description</code>	<code>string</code>	A short description of the API. Markdown can be used here.
<code>termsOfService</code>	<code>string</code>	A URL to the Terms of Service for the API.
<code>contact</code>	<u>Contact Object</u>	The contact information for the exposed API.
<code>license</code>	<u>License Object</u>	The license information for the exposed API.
<code>version</code>	<code>string</code>	<i>REQUIRED.</i> The version of the documentation, not the OpenAPI spec.

```
info:  
  title: JSONPlaceholder  
  description: Free fake API for testing and prototyping.  
  version: 0.1.0
```

Copy



EXTERNALDOCS AND SERVER

According to the specification, here's what the `externalDocs object` should look like:

Field Name	Type	Description
<code>description</code>	<code>string</code>	A short description of the target documentation. Markdown can be used.
<code>url</code>	<code>string</code>	<i>REQUIRED</i> . The URL for the target documentation.

In your YAML document, add an `externalDocs` object that points to JSONPlaceholder's guide:

```
externalDocs:  
  description: "JSONPlaceholder's guide"  
  url: https://jsonplaceholder.typicode.com/guide
```

To fix this, OpenAPI provides a `servers` field. Add the following lines to your YAML document:

```
servers:  
- url: https://jsonplaceholder.typicode.com  
  description: JSONPlaceholder
```

Copy



PATH

According to the OpenAPI specification, this is what the [Path Item object](#) will look like:

Name	Type	Description
summary	string	An optional summary of this route.
description	string	An optional description of what the route can do.
get / post / put / patch / delete / etc	Operation Object	A definition of an operation (method) on this route.
servers	Array of Server Objects	An alternative <code>server</code> array to service all operations in this path.
parameters	An array of Parameter Object	Parameters that are applicable for all operations on this path. These parameters can be on the querystring, header, cookie, or the path itself.

```
paths:  
  "/posts":  
    # ...  
    <$>post<$>:  
      tags: ["posts"]  
      summary: Create a new post  
      responses:  
        "200":  
          description: A post was created  
          content:  
            application/json:  
              schema:  
                $ref: "#/components/schemas/post"
```

Copy



OPERATION

The `operation` object has [many items](#), but for this tutorial, you'll focus on a smaller set:

Name	Type	Description
<code>tags</code>	Array of strings	A list of tags for API documentation control. Tags can be used for grouping similar routes.
<code>summary</code>	string	A short summary of what the operation does.
<code>description</code>	string	A description of the operation. Markdown can be used here.
<code>externalDocs</code>	External Documentation Object	Additional external documentation for this operation. Same as <code>externalDocs</code> on the main object.
<code>parameters</code>	Array of Parameter Objects	Same as <code>parameters</code> in the Path Item object.
<code>requestBody</code>	Request Body Object	The body of the request. This can NOT be used when the method <code>GET</code> or <code>DELETE</code> .
<code>responses</code>	Responses Object	<i>REQUIRED.</i> The list of possible responses returned by the API for this operation.

```
paths:  
  "/posts":  
    # ...  
    <$>post<$>:  
      tags: ["posts"]  
      summary: Create a new post  
      responses:  
        "200":  
          description: A post was created  
          content:  
            application/json:  
              schema:  
                $ref: "#/components/schemas/post"
```

Copy



SCHEMA

```
components:
schemas:
post:
  type: object
  properties:
    id:
      type: number
      description: ID of the post
    title:
      type: string
      description: Title of the post
    body:
      type: string
      description: Body of the post
    userId:
      type: number
      description: ID of the user who created the post
```

Copy

The above schema is an `object`, denoted by `type: object`. It has four `properties: id`, `title`, `body`, and `userId`. That is how a schema is defined. Now you can use `$ref` in any object to reference this schema. This is defined as per the specification for URI syntax, [RFC3986](#).



SPEC IN MULTIPLE FILES

Why

Be able to explain the reasons for using a multi-file format:

1. Easier to contribute.
2. Easier to review contributions.
3. Better supports a docs-like-code workflow.
4. Enforces better re-use of objects to avoid duplication and divergence issues.
5. Supported by Redocly toolchain including the free open-source [Redocly CLI](#).

```
├── README.md
├── code_samples
│   ├── C#
│   │   └── echo
│   │       └── post.cs
│   ├── PHP
│   │   └── echo
│   │       └── post.php
│   └── README.md
└── components
    ├── README.md
    ├── headers
    │   └── ExpiresAfter.yaml
    ├── schemas
    │   ├── Email.yaml
    │   └── User.yaml
    └── securitySchemes
        ├── api_key.yaml
        ├── basic_auth.yaml
        └── main_auth.yaml
└── openapi.yaml
└── paths
    ├── README.md
    ├── echo.yaml
    └── users@{username}.yaml
```

EJEMPLO OPENAPI

Ahora revisemos un par de ejemplos usando Swagger editor, Mermade Openapi-gui, Insomnia y Postman como editores.

API DESIGN BEST PRACTICES

- NOMENCLATURA DE RECURSOS
- FORMATOS DE MENSAJE
- CABECERAS
- VERSIONADO
- EJEMPLOS

2. Best Practices

2.1. Use nouns to represent resources

RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb) because nouns have properties that verbs do not have – similarly, resources have attributes. Some examples of a resource are:

- Users of the system
- User Accounts
- Network Devices etc.

and their resource URIs can be designed as below:

```
http://api.example.com/device-management/managed-devices  
http://api.example.com/device-management/managed-devices/{device-id}  
http://api.example.com/user-management/users  
http://api.example.com/user-management/users/{id}
```

RESOURCES NAMING

2.1.1. document

A document resource is a singular concept that is akin to an object instance or database record.

In REST, you can view it as a single resource inside resource collection. A document's state representation typically includes both fields with values and links to other related resources.

Use "singular" name to denote document resource archetype.

```
http://api.example.com/device-management/managed-devices/{device-id}  
http://api.example.com/user-management/users/{id}  
http://api.example.com/user-management/users/admin
```

RESOURCES NAMING

2.1.2. collection

A collection resource is a server-managed directory of resources.

Clients may propose new resources to be added to a collection. However, it is up to the collection resource to choose to create a new resource or not.

A collection resource chooses what it wants to contain and also decides the URIs of each contained resource.

Use the "plural" name to denote the collection resource archetype.

```
http://api.example.com/device-management/managed-devices  
http://api.example.com/user-management/users  
http://api.example.com/user-management/users/{id}/accounts
```

RESOURCES NAMING

2.1.3. store

A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them.

A store never generates new URLs. Instead, each stored resource has a URI. The URI was chosen by a client when the resource initially put it into the store.

Use "plural" name to denote store resource archetype.

```
http://api.example.com/song-management/users/lidl/playlists
```

RESOURCES NAMING

2.1.4. controller

A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values, inputs, and outputs.

Use "verb" to denote controller archetype.

```
http://api.example.com/cart-management/users/{id}/cart/checkout http://api.example.com/song-  
management/users/{id}/playlist/play
```

RESOURCES NAMING

2.1.4. controller

A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values, inputs, and outputs.

Use "verb" to denote controller archetype.

```
http://api.example.com/cart-management/users/{id}/cart/checkout http://api.example.com/song-  
management/users/{id}/playlist/play
```

2.2. Consistency is the key

Use consistent resource naming conventions and URI formatting for minimum ambiguity and maximum readability and maintainability. You may implement the below design hints to achieve consistency:

2.2.1. Use forward slash (/) to indicate hierarchical relationships

The forward-slash (/) character is used in the path portion of the URI to indicate a hierarchical relationship between resources. e.g.

```
http://api.example.com/device-management  
http://api.example.com/device-management/managed-devices  
http://api.example.com/device-management/managed-devices/lidl  
http://api.example.com/device-management/managed-devices/lidl/scripts  
http://api.example.com/device-management/managed-devices/lidl/scripts/{lid}
```

RESOURCES NAMING

2.2.2. Do not use trailing forward slash (/) in URIs

As the last character within a URI's path, a forward slash (/) adds no semantic value and may confuse. It's better to drop it from the URI.

```
http://api.example.com/device-management/managed-devices/ http://api.example.com/device-  
management/managed-devices /*This is much better version*/
```

2.2.3. Use hyphens (-) to improve the readability of URIs

To make your URIs easy for people to scan and interpret, use the hyphen (-) character to improve the readability of names in long path segments.

```
http://api.example.com/device-management/managed-devices/  
http://api.example.com/device-management/managed-devices /*This is much better version*/
```

2.3. Do not use file extensions

File extensions look bad and do not add any advantage. Removing them decreases the length of URIs as well. No reason to keep them.

Apart from the above reason, if you want to highlight the media type of API using file extension, then you should rely on the media type, as communicated through the **Content-Type** header, to determine how to process the body's content.

```
http://api.example.com/device-management/managed-devices.xml /*Do not use it*/
```

```
http://api.example.com/device-management/managed-devices /*This is correct URI*/
```

RESOURCES NAMING

2.4. Never use CRUD function names in URLs

We should not use URLs to indicate a CRUD function. URLs should only be used to uniquely identify the resources and not any action upon them.

We should use HTTP request methods to indicate which CRUD function is performed.

```
HTTP GET http://api.example.com/device-management/managed-devices //Get all devices  
HTTP POST http://api.example.com/device-management/managed-devices //Create new Device  
  
HTTP GET http://api.example.com/device-management/managed-devices/lidl //Get device for given  
Id  
HTTP PUT http://api.example.com/device-management/managed-devices/lidl //Update device for  
given Id  
HTTP DELETE http://api.example.com/device-management/managed-devices/lidl //Delete device for  
given Id
```

2.5. Use query component to filter URI collection

Often, you will encounter requirements where you will need a collection of resources sorted, filtered, or limited based on some specific resource attribute.

For this requirement, do not create new APIs – instead, enable sorting, filtering, and pagination capabilities in resource collection API and pass the input parameters as query parameters. e.g.

```
http://api.example.com/device-management/managed-devices  
http://api.example.com/device-management/managed-devices?region=USA  
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ  
http://api.example.com/device-management/managed-devices?  
region=USA&brand=XYZ&sort=installation-date
```

RESOURCES NAMING

2.2.4. Do not use underscores (_)

It's possible to use an underscore in place of a hyphen to be used as a separator – But depending on the application's font, it is possible that the underscore (_) character can either get partially obscured or completely hidden in some browsers or screens.

To avoid this confusion, use hyphens (-) instead of underscores (_).

```
http://api.example.com/inventory-management/managed-entities/lidl/install-script-location //More  
readable
```

```
http://api.example.com/inventory-management/managedEntities/lidl/installScriptLocation //Less  
readable
```

2.2.5. Use lowercase letters in URIs

When convenient, lowercase letters should be consistently preferred in URI paths.

```
http://api.example.org/my-folder/my-doc //1  
HTTP://API.EXAMPLE.ORG/my-folder/my-doc //2  
http://api.example.org/My-Folder/my-doc //3
```

FORMATOS DE MENSAJE DE LLAMADA Y RESPUESTA

Los formatos más habituales de mensaje son:

- **Plain:** application/json, application/json
- **Vendor specific:** e.g. application/vnd.github.v3+json
- **Standard/Hipermedia:** e.g. HAL, Collection+JSON or JSON-API, Siren, etc.

Nomenclatura de atributos en mensajes de tipo JSON: Es clave la consistencia. Para toda la API debe usarse el mismo formato:

- **myIdentifier : Camel case (e.g. in java variable names)**
- **MyIdentifier : Capital camel case (e.g. in java class names)**
- **my_identifier : Snake case (e.g. in python variable names)**
- **my-identifier : Kebab case (e.g. in racket names)**
- **myidentifier : Flat case (e.g. in java package names)**

Mi recomendación es usar Camel case.

PLAIN MEDIA TYPES

Type application:

application/json
application/xml
application/zip

Type audio:

audio/mpeg

Type image:

image/gif
image/jpeg
image/png

Type multipart:

multipart/mixed
multipart/form-data

Type text:

text/csv
text/xml

Type video:

video/mpeg
video/mp4

FORMATOS DE MENSAJE DE LLAMADA Y RESPUESTA

```
{  
  "item": {  
    "id": "1234",  
    "title": "Table made from unicorns",  
    "description": "White table made of happiness: width 75 cm,  
height: 73 cm, Max weight: 50 kg",  
    "link": "https://www.example.com/tables/unicorn/fancy",  
    "availability": "in stock",  
    "price": {  
      "value": "399.67",  
      "currency": "EUR"  
    },  
    "color": "rainbow"  
  }  
}
```

VERSIONADO

BEST PRACTICES

Backward Compatibility

It is an excellent practice to have your API backward compatible if possible.

URI Versioning

While creating new versions of your API, you should add the version number in your API URI.

Use Semantic Versioning

You use semantic versioning to version your API, i.e., major.minor.patch. If you have introduced some major changes that would break the previous version, you should update the major version like 2.0.0. You update the minor one when you make some new changes, but they are not breaking the previous release. And lastly, if you are patching some bug, update the patch number.

Restful Versioning Techniques

1. URI Path Change

<http://api.mydomain.com/entities>

Becomes

<http://api.mydomain.com/1/entities>

2. Add Query Parameter

<http://api.mydomain.com/entities>

Becomes

<http://api.mydomain.com/entities?version=1>

3. Use Custom Header

[My_Custom_Accept: vesion1](#)

4. Content Negotiation via Accept Header

[Accept: application/vnd.myentity+json;version=1](#)

5. Entity Change

<http://api.mydomain.com/entities>

Becomes

http://api.mydomain.com/new_entities

Headers

The REST headers and parameters contain a wealth of information that can help you track down issues when you encounter them. HTTP Headers are an important part of the API request and response as they represent the meta-data associated with the API request and response. Headers carry information for:

1. Request and Response Body
2. Request Authorization
3. Response Caching
4. Response Cookies

Other than the above categories HTTP headers also carry a lot of other information around HTTP connection types, proxies etc. Most of these headers are for management of connections between client, server and proxies and do not require explicit validation through testing.

Headers are mostly classified as request headers and response headers, know the major request and response headers. You will have to set the request headers when you are sending the request for testing an API and you will have to set the assertion against the response headers to ensure that right headers are being returned.

The headers that you will encounter the most during API testing are the following, you may need to set values for these or set assertions against these headers to ensure that they convey the right information and everything works fine in the API:

Authorization: Carries credentials containing the authentication information of the client for the resource being requested.

WWW-Authenticate: This is sent by the server if it needs a form of authentication before it can respond with the actual resource being requested. Often sent along with a response code of 401, which means 'unauthorized'.

Accept-Charset: This is a header which is set with the request and tells the server about which character sets are acceptable by the client.

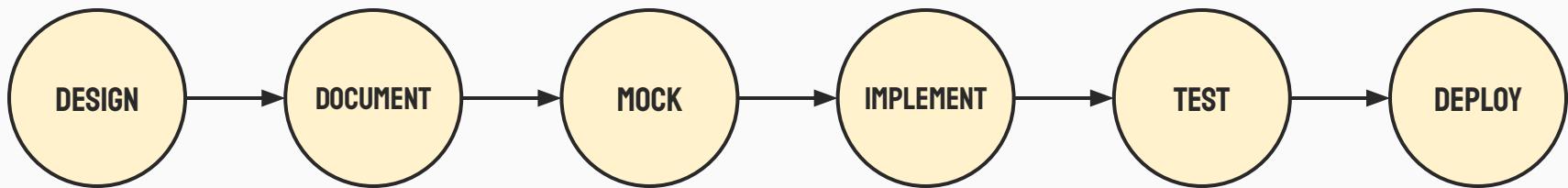
Content-Type: Indicates the media type (text/html or text/JSON) of the response sent to the client by the server, this will help the client in processing the response body correctly.

Cache-Control: This is the cache policy defined by the server for this response, a cached response can be stored by the client and re-used till the time defined by the Cache-Control header.

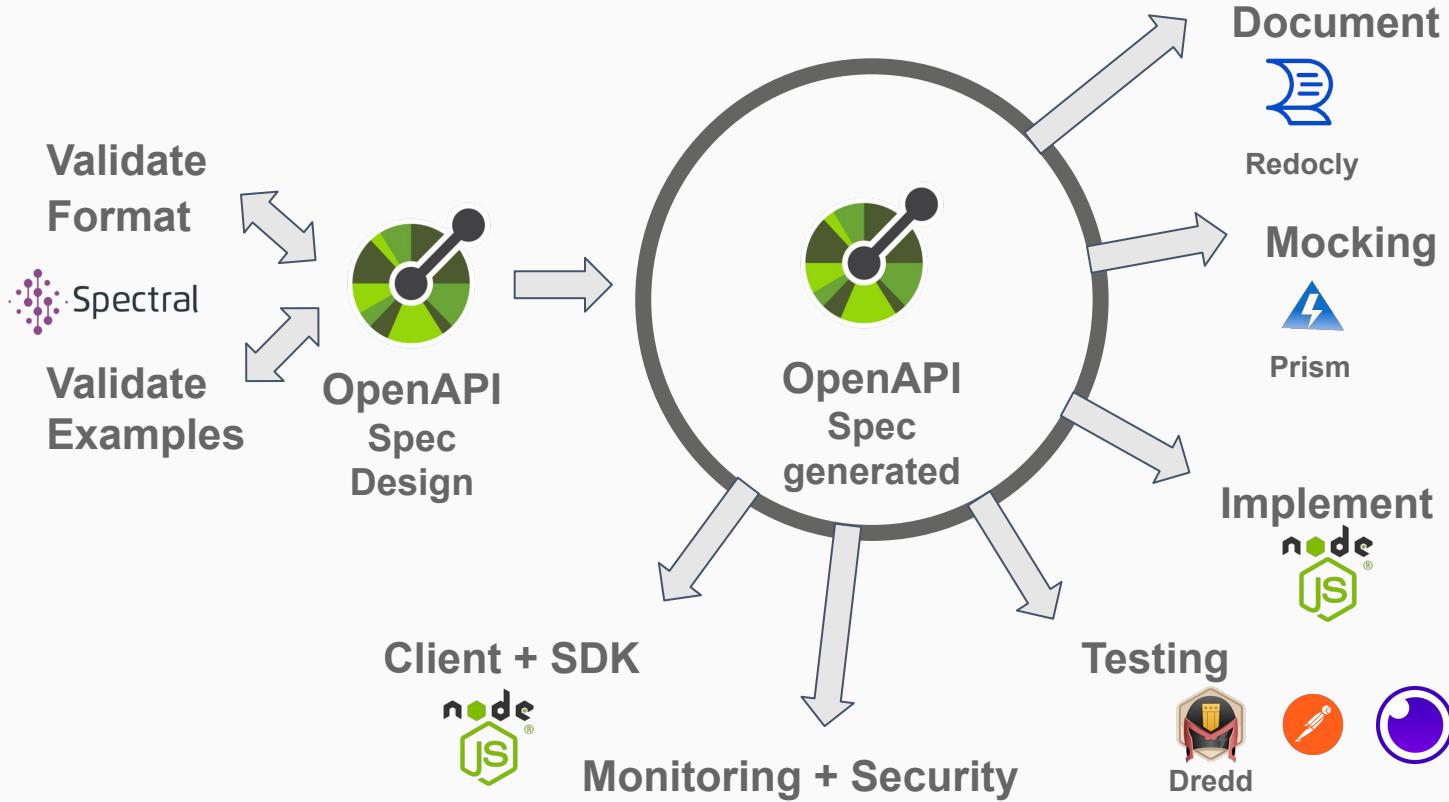
06. CICLO DE VIDA



API LIFECYCLE



API LIFECYCLE



<https://openapi.tools/>

Tool Types

- **Auto Generators:** Tools that will take your code and turn it into an OpenAPI Specification document.
- **Converters:** Various tools to convert to and from OpenAPI and other API description formats.
- **Data Validators:** Check to see if API requests and responses are lining up with the API description.
- **Description Validators:** Check your API description to see if it is valid OpenAPI.
- **Documentation:** Render API Description as HTML (or maybe a PDF) so slightly less technical people can figure out how to work with the API.
- **DSL:** Domain Specific Language to write OpenAPI in your language of choice.
- **GUI Editors:** Visual editors help you design APIs without needing to memorize the entire OpenAPI specification.
- **Miscellaneous:** Anything else that does stuff with OpenAPI but hasn't quite got enough to warrant its own category.
- **Mock Servers:** Fake servers that take description document as input, then route incoming HTTP requests to example responses.
- **Parsers:** Loads and read OpenAPI descriptions, so you can work with them programmatically.
- **SDK Generators:** Generate code to give to consumers, to help them avoid interacting at a HTTP level.
- **Security:** By poking around your OpenAPI description, some tools can look out for attack vectors you might not have noticed.
- **Server Implementations:** Easily create and implement resources and routes for your APIs.
- **Testing:** Quickly execute API requests and validate responses on the fly through command line or GUI interfaces.
- **Text Editors:** Text editors give you visual feedback whilst you write OpenAPI, so you can see what docs might look like.
- **Learning:** Whether you're trying to get documentation for a third party API based on traffic, or are trying to switch to design-first at an organization with no OpenAPI at all, learning can help you move your API spec forward and keep it up to date



API LIFECYCLE COMMANDS

Validar especificación:

```
$ spectral lint openapi.yaml
```

Generar Documentación:

```
$ redocly preview-docs openapi.yaml
```

Generar Mocks:

```
$ prism mock -d openapi.yaml # Dynamic examples  
$ prism mock openapi.yaml # Static examples
```

Generar código:

```
$ openapi-generator-cli generate -g nodejs-express-server -i openapi.yaml -o  
api_server  
$ openapi-generator-cli generate -g javascript -i openapi.yaml -o api_client
```

Pruebas:

```
$ st run openapi.yaml
```

RECURSOS.



RECURSOS

- Curso APIs for Beginners en freecodecamp.org por Craig Dennis.
 - <https://www.freecodecamp.org/news/apis-for-beginners-full-course/>
 - [Github](#) del Curso.
- [Presentación del curso.](#)
- Recursos Twilio
 - [Docs Twilio](#)
 - <https://www.twilio.com/docs/sms/quickstart/python>
- Postman: <https://www.postman.com/>
- Insomnia: <https://insomnia.rest/>
- Mermade Openapi-gui Openapi-ui: <https://github.com/Mermade/openapi-gui>
- CURL: <https://curl.se/>
- Ejemplo de crear una api con Insomnia:
<https://www.digitalocean.com/community/tutorials/how-to-create-documentation-for-your-rest-api-with-insomnia>
- OpenAPI Specification: <https://spec.openapis.org/oas/latest.html>
- Rest API Design: <https://www.mscharhag.com/p/rest-api-design>
- Open API Tools: <https://openapi.tools/>
- A Practical Approach to API Design: <https://leanpub.com/restful-api-design>
- Repo con la presentación: <https://github.com/benjagm/api-bootcamp-boost-academy>

THANKS

Does anyone have any questions?

bgranados@twilio.com
+34 661 709 879
twilio.com



CREDITS: This presentation template was created by [Slidesgo](#), including icons by [Flaticon](#), and infographics & images by [Fleepik](#) and illustrations by [Storyset](#)

Please keep this slide for attribution.

