

# API DESIGN PRINCIPLES

Mohammad Nazari



# Contents

1. What is an API?
2. REST API architecture
3. What is important in designing an API?
4. Design Principles
  1. Don't surprise your users
  2. Focus on use cases
  3. Copy successful APIs
  4. REST is not always the best
  5. Focus on the developer experience

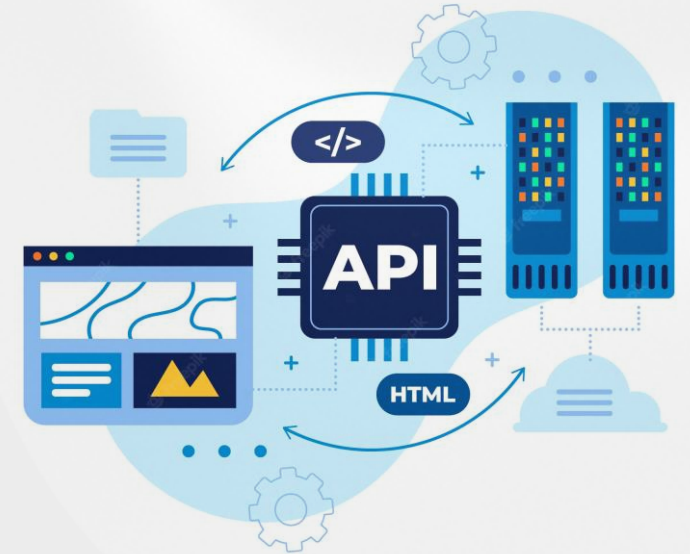
# What is an API?

- API is the acronym for Application Programming Interface.
- It allows two applications to talk to each other.

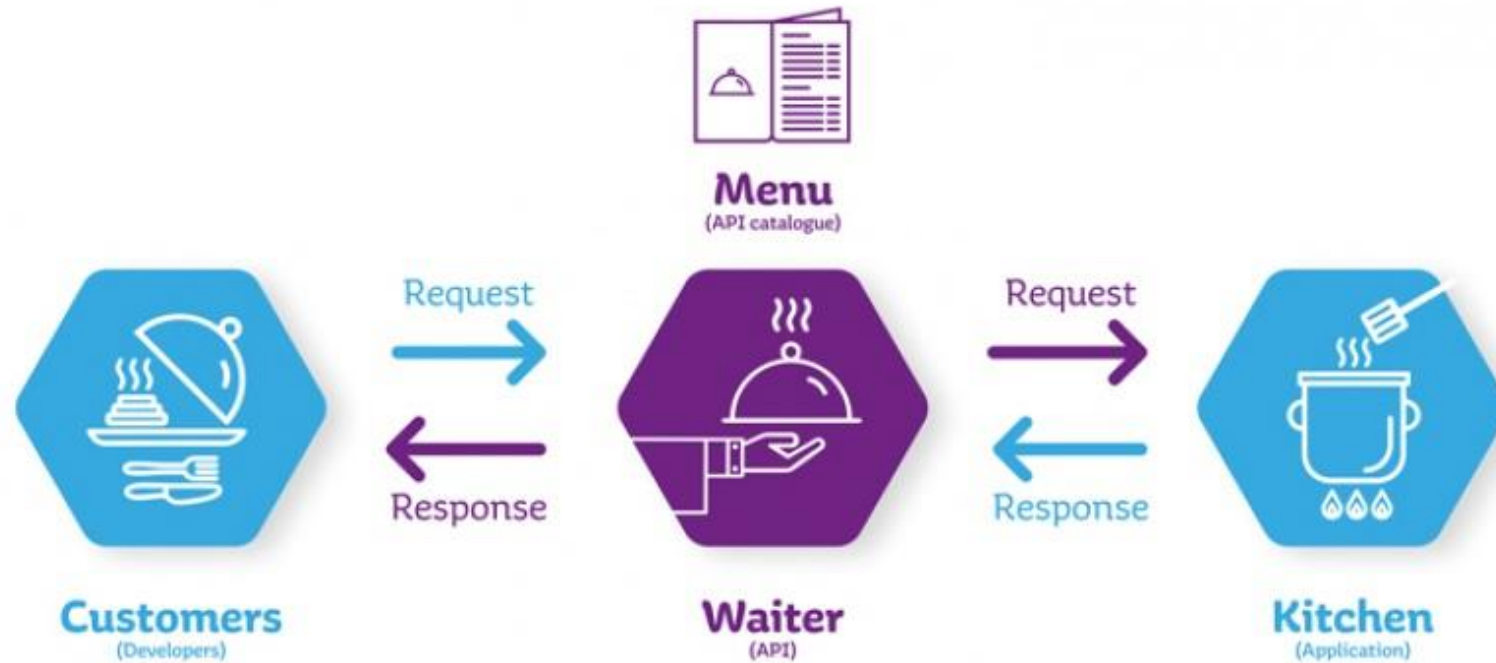


# What Is an Example of an API?

When you use an application, the application connects to the Internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way. This is what an API is - all of this happens via API.



# Real world example



# API Architectures

- An API's architecture consists of the rules that guide what information an API can share with clients and how it shares the data.
- REST
- SOAP
- RPC (JSON-RPC, XML-RPC)
- GraphQL

# What is RESTful API?



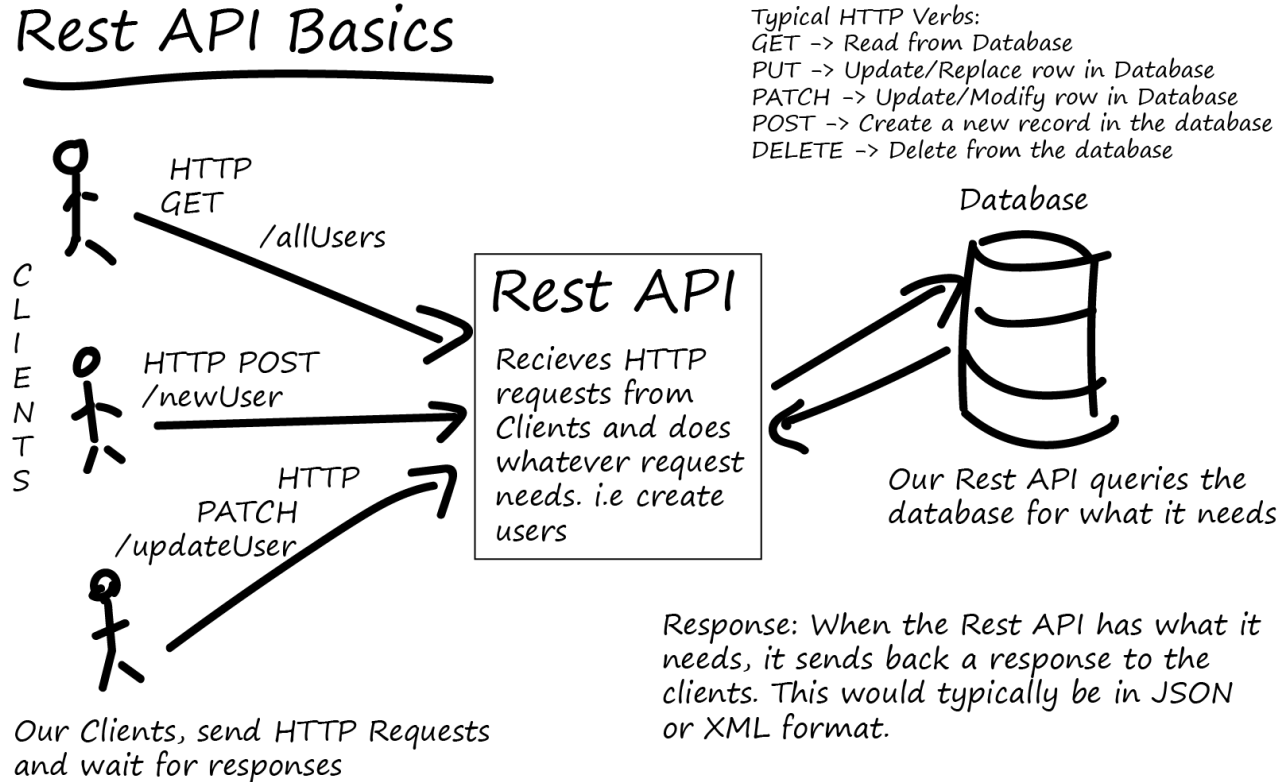
Roy Fielding

- REST stands for Representational State Transfer
- refers to APIs that are resources based-where the clients interact with the servers by requesting things rather than actions
- REST APIs are commonly used by various websites like Amazon, Google, Facebook, LinkedIn, and Twitter, allowing users to communicate with these cloud services.

**{REST:API}**

# How does REST API work?

## Rest API Basics



## persons Operations about users of the system

POST

/persons Create a new user in the database. (Note: JS

GET

/persons/ Return a list of all users in the database.

GET

/persons/{username} Return a user by supplying

PUT

/persons/{username} Update an existing user in the body along with the

DELETE

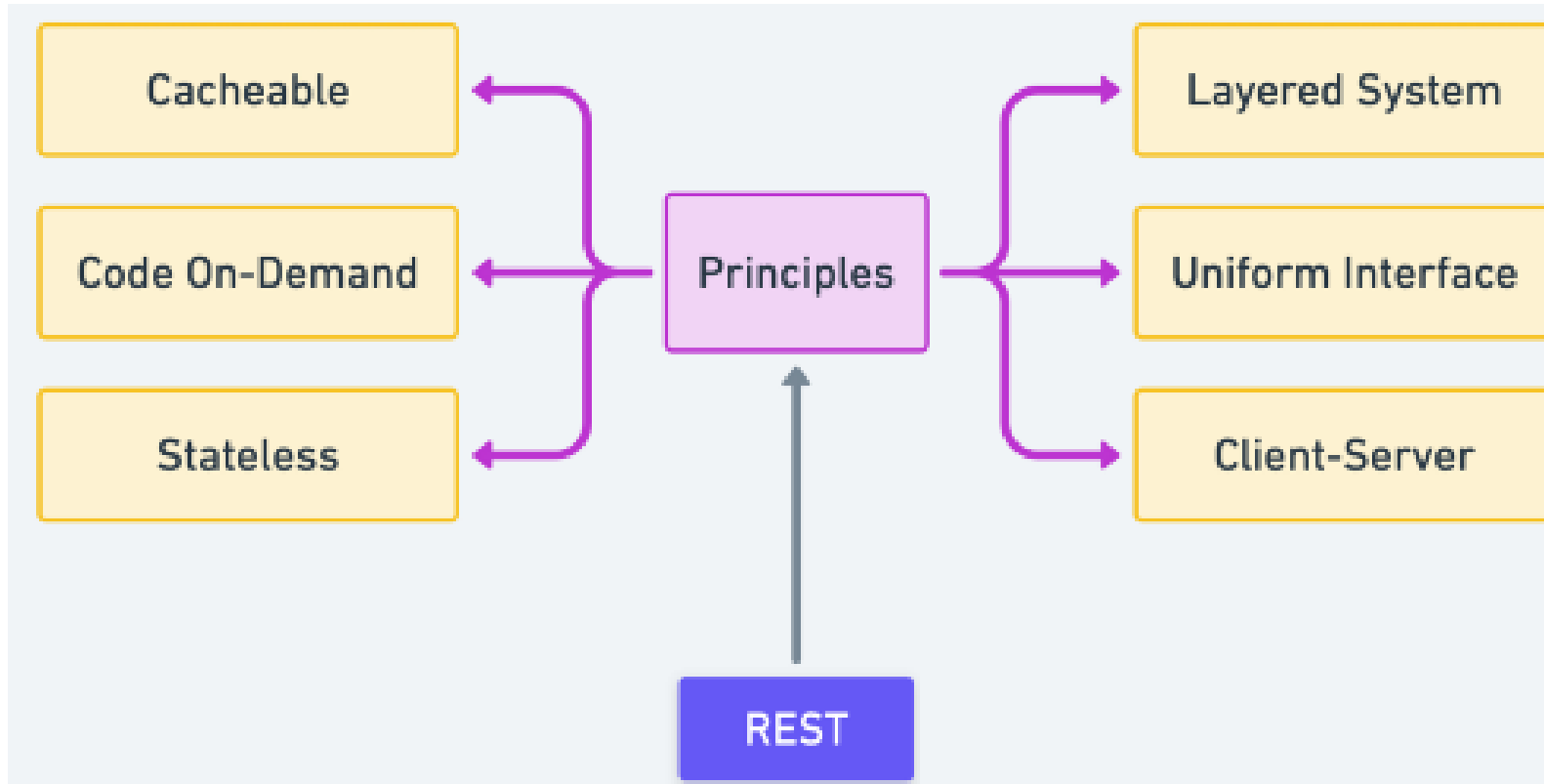
/persons/{username} Delete an existing user w



# Status codes

- REST APIs use the Status-Line part of an HTTP response message to inform
- clients of their request's overarching result
- 1xx: Informational - Communicates transfer protocol-level information.
- 2xx: Success - Indicates that the client's request was accepted successfully.
- 3xx: Redirection - Indicates that the client must take some additional action in order to complete their request.
- 4xx: Client Error - This category of error status codes points the finger at clients.
- 5xx: Server Error - The server takes responsibility for these error status codes.

# Guiding Principles of REST



# Client-server

- server should be isolated from one another and permitted to develop independently
- This way, you can improve manageability across numerous platforms and increase scalability by streamlining the server components as user interface concerns are separate from the data storage concerns.

# Stateless

- calls can be made independent of one another
- every request sent from the client to server must include all the info needed to comprehend the request.

# Cacheable

- As a stateless API can upsurge request overhead by managing huge loads of inbound and outbound calls, a REST API design should be able to store cacheable data
- In case a response is cacheable, the client cache is provided the right to recycle that response data for similar requests in the future.

# Uniform Resource Identifiers

- the app needs to know that it can hit the same URI to get a particular piece of data, every-time!
- It also needs to know that it'll get the data in the format that it expects, and that format will not change.

# Layered System

- REST API's architecture includes several layers that operate together to construct a hierarchy that helps generate a more scalable and flexible application

# Code on Demand \*

- In distributed computing, code on demand is any technology that sends executable software code from a server computer to a client computer upon request from the client's software
- Flash Player, and JavaScript

# Common pitfalls of organic APIs in big companies

- Lack of vision
- Prioritizing the developer experience
- Bad designing



# Lack of vision

- the Netflix API was first released, it was open to the developers
- The goal of this product was to "**let a thousand flowers bloom**" to enable third party developers to create amazing applications that would bring in new subscribers and make money
- but the revenue benefit never occurred!



# What was the problem?

- Guidance for developers on how to use the API was minimal and focused on what the API did rather than providing tutorials and use cases
- the company's goal for the API was unfocused
- many clients re-created the functionality of Netflix's website without adding new functionality





# Prioritizing the developer experience



- Twitter started with a single web page with a few features
- Many Twitter features were initially created by external developers as part of their products, and Twitter adopted the new features
- Twitter rewrote its Terms of Use so that developers couldn't create applications that competed directly with its product
- the developer community was, by and large, quite unhappy with Twitter, and the company's credibility suffered as a result

# What is important in designing an API?

it should be:

- straightforward to use
- well documented
- well supported

Using your API should be a joyful and engaging experience, not a slog through a frustrating and never ending series of challenges

Remember, developers are people too!



# Design Principles

- Don't surprise your users
- Focus on use cases
- Copy successful APIs
- REST is not always the best
- Focus on the developer experience





**developers are  
your customers**



**focus on  
usability**



## Don't surprise your users

---

Be mindful of the decisions you make and make sure to communicate your intent clearly and consistently.

## Flickr API example

Flickr call	RESTful call
GET /services/rest/?method=flickr.activity.userPhotos	GET /services/rest/activity/userPhotos
POST /services/rest/?method=flickr.favorites.add	POST /services/rest/favorites
POST /services/rest/?method=flickr.favorites.remove	DELETE /services/rest/favorites/:id
POST /services/rest/?method=flickr.galleries.editPhoto	PUT /services/rest/galleries/photo/:id



Don't make me think!



## Focus on use cases

---

If you can't describe what you want developers to do with your API, they won't know what you're expecting, and you won't have any guiding vision to drive the development of the API.

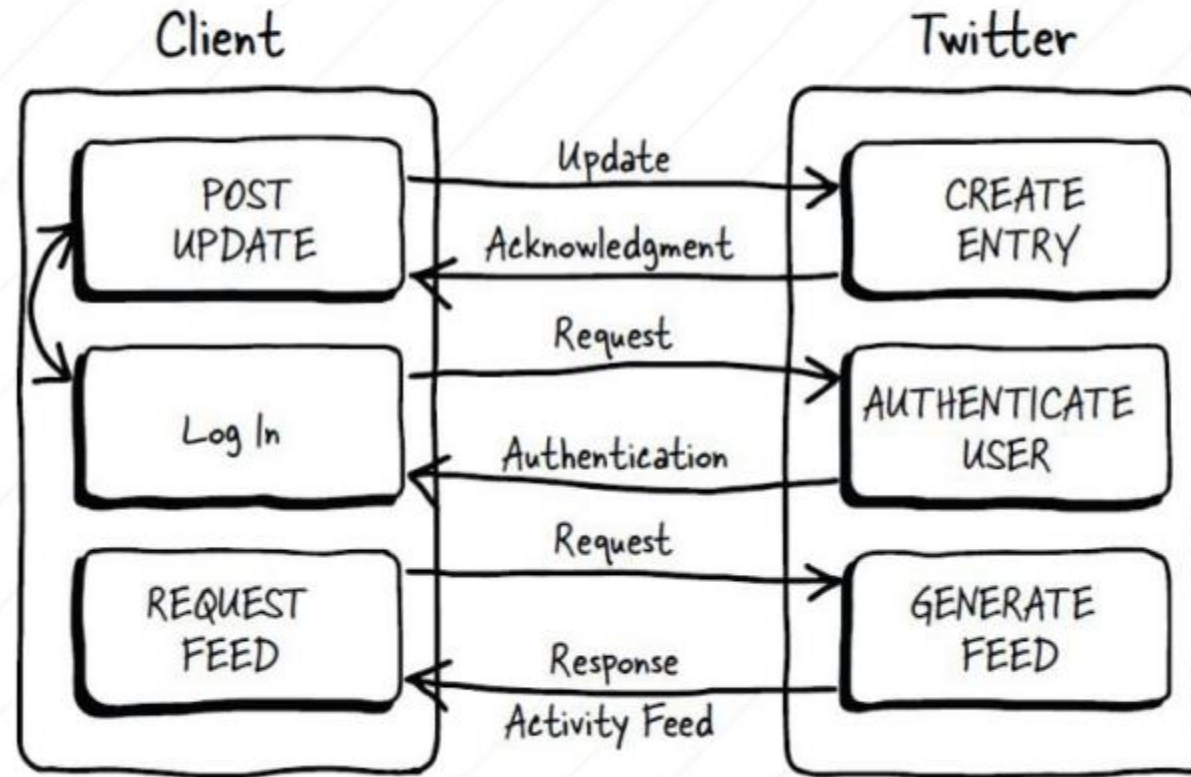


# Twitter example

- Log in with the platform.
- Post a message to the user's stream.
- Read the user's message stream to display the new content in context.



# Twitter example





## Copy successful APIs

---

Stand on the shoulders of giants. There's no shame in cribbing from a successful API to make the experience of the application developers that much more consistent and easy.

# {A} API COMMONS

[HOME](#)[BLOG](#)[ABOUT](#)[FORMAT](#)[LICENSES](#)[APIS](#)[ADD API](#)[BRANDING](#)[FAQ](#)[CONTACT](#)

## API SPECIFICATIONS AND DATA MODELS

A common place to publish and share your own API specifications / Data Models in any format such as Swagger, API Blueprint or RAML, as well as explore and discover the API designs of others.



## DECLARING AN API IN THE COMMONS

Licensing your API for others to reuse and remix is easy using Creative Commons licenses - find out more [here](#).



## JOIN THE CONVERSATION

API Commons is just getting started and there are plenty of interesting challenges (technical and non-technical), so join [the conversation](#) on how best to proceed - join the mailing list or follow on twitter or github.

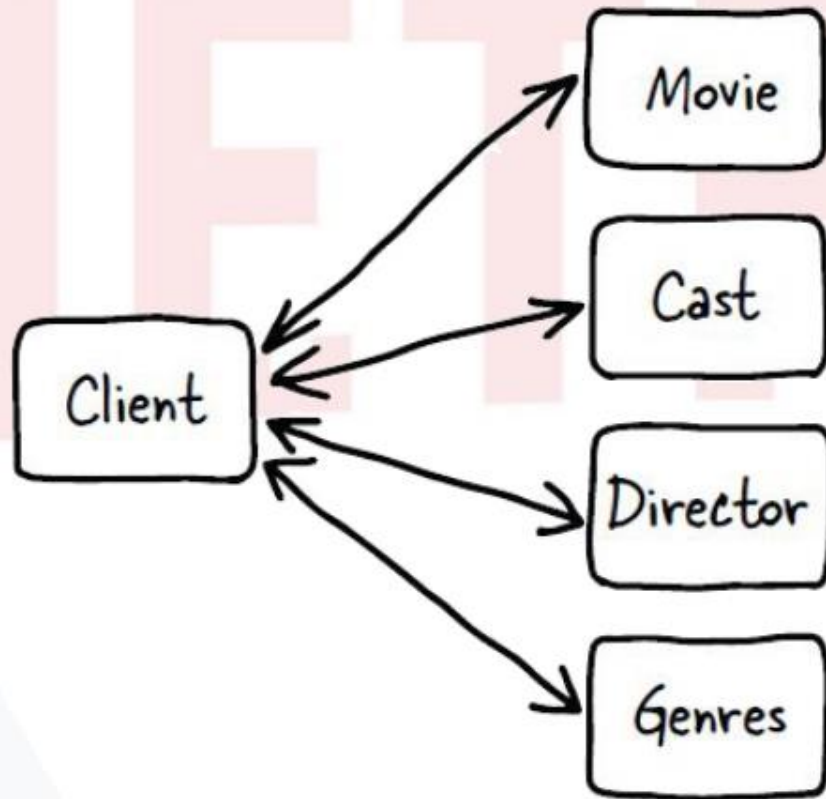


REST is not always best

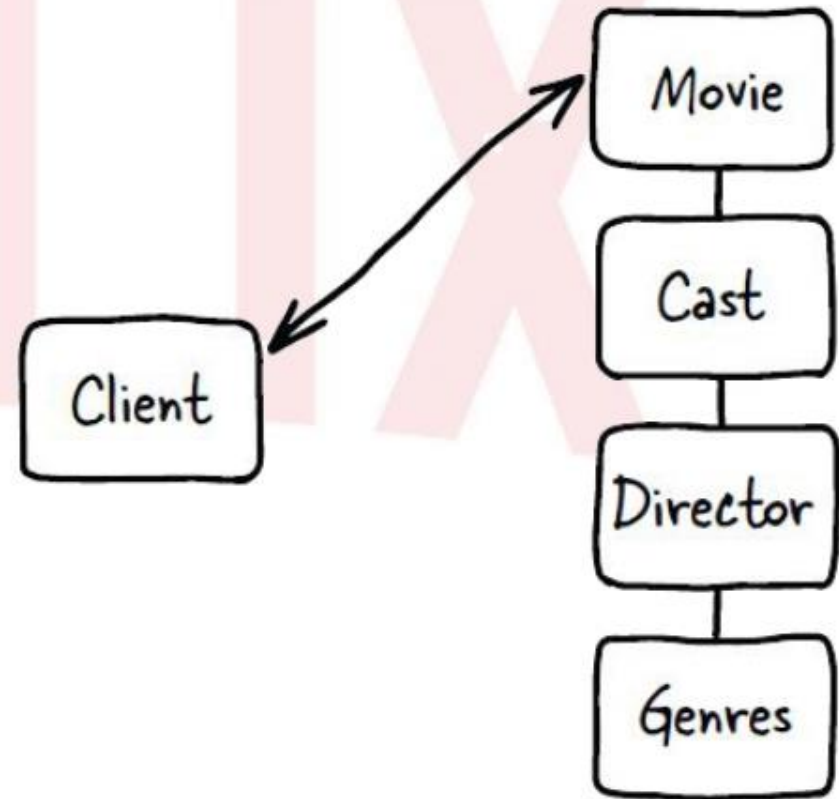
---

# Netflix example

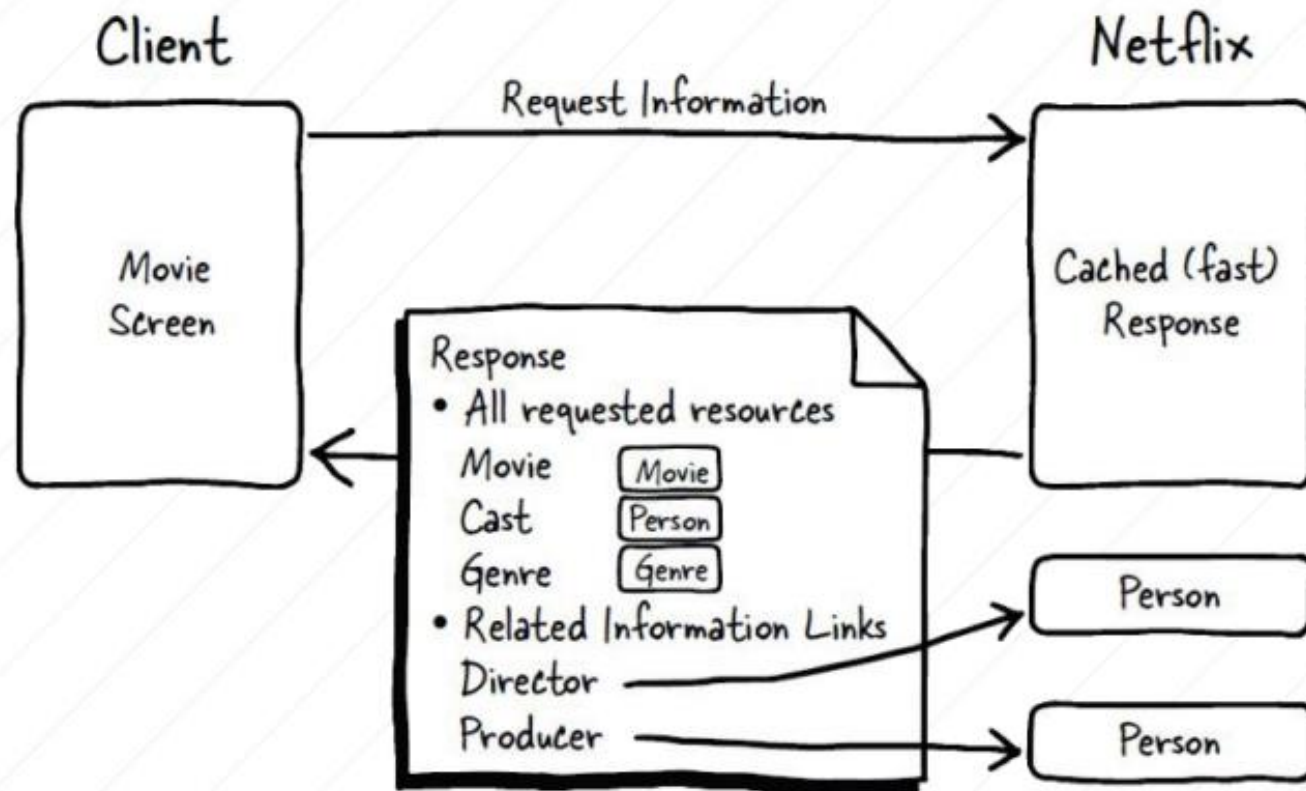
Strict REST



Expanded API



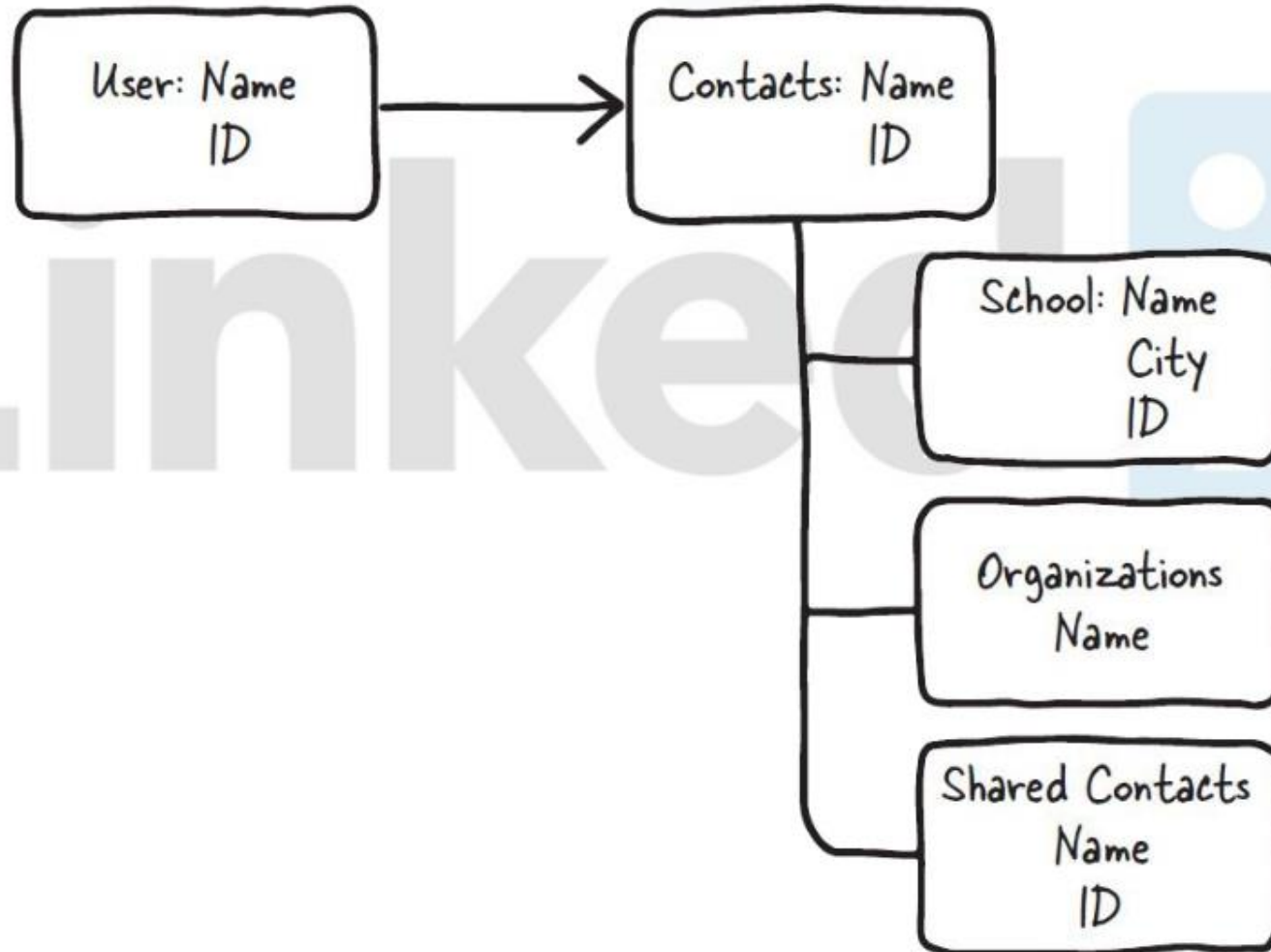
# Netflix example





# LinkedIn example

## Complex LinkedIn Request



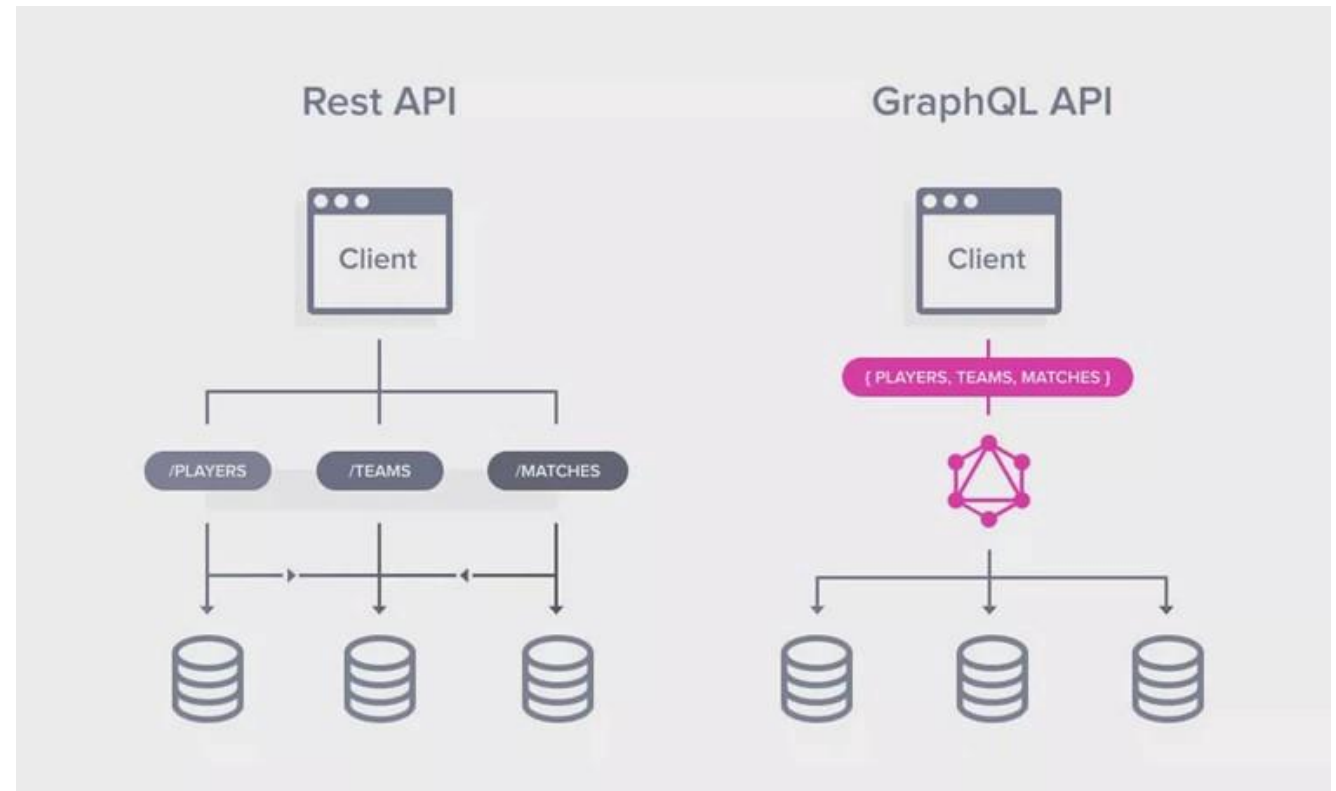


# GraphQL

- GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data

- High flexibility
- No over-fetching
- Latest version not required

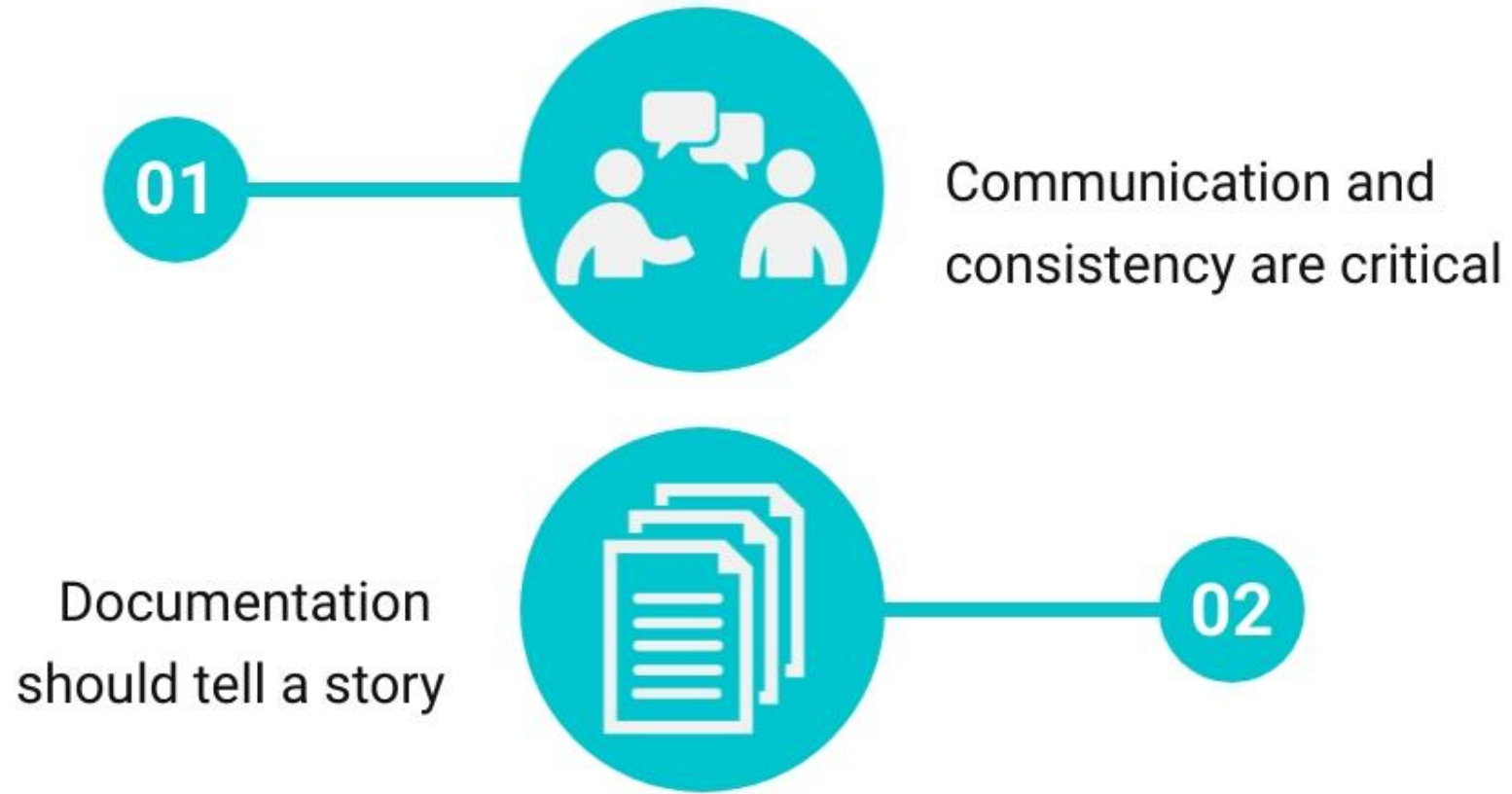
- Security
- GraphQL Caching
- Complexity






## Focus on the developer experience

---



# Documentation should tell a story

 **swagger**

Authorize Explore

## Swagger Petstore

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [irc.freenode.net, #swagger](irc://freenode.net/#swagger). For this sample, you can use the api key `special-key` to test the authorization filters.

Find out more about Swagger

<http://swagger.io>  
[Contact the developer](#)  
[Apache 2.0](#)

### pet : Everything about your Pets

Show/Hide | List Operations | Expand Operations

POST	/pet	Add a new pet to the store
PUT	/pet	Update an existing pet
GET	/pet/findByStatus	Finds Pets by status
GET	/pet/findByTags	Finds Pets by tags
DELETE	/pet/{petId}	Deletes a pet
GET	/pet/{petId}	Find pet by ID
POST	/pet/{petId}	Updates a pet in the store with form data
POST	/pet/{petId}/uploadImage	uploads an image


### store : Access to Petstore orders

Show/Hide | List Operations | Expand Operations

### user : Operations about user

Show/Hide | List Operations | Expand Operations

[ BASE URL: /v2 , API VERSION: 1.0.0 ]

VALID 

# Resources

Irresistible APIs



Practical API Design

API Architecture



RESTful API Design

[www.restfulapi.net](http://www.restfulapi.net)



[www.medium.com](https://www.medium.com)



Thanks for attention

# Terms

- Resource
- Collection
- URL (Uniform Resource Locator)

# API endpoint

- `/addNewEmployee`
- `/updateEmployee`
- `/deleteEmployee`
- `/deleteAllEmployees`
- `/promoteEmployee`
- `/promoteAllEmployees`

It should only specify the location of a resource, not the current operation on it.



# API endpoint

- -GET /companies/3/employees
  - -GET /companies/3/employees/45
  - -DELETE /companies/3/employees/45
  - -POST /companies
- URL contains a resource (nouns) and resource should be plural.
  - The HTTP methods (verbs) should define the action
  - Depict Resources Hierarchy through URIs

# HTTP methods

- GET /users fetch the list of all posts
  - POST /users create a new user
  - PUT /users/1 this update all payload of a resource
  - PATCH /users/1 you send only specified parameter for an update in request
  - DELETE /users/1 this delete the user with an id of 1
- Don't misuse safe methods

# HTTP Status Code

- Informational responses (100 – 199)
- Successful responses (200 – 299)
  - 200 OK
  - 201 Created
  - 204 No Content
- Redirection messages (300 – 399)
  - 304 Not Modified
- Client error responses (400 – 499)
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- Server error responses (500 – 599)
  - 500 Internal Server Error
  - 503 Service Unavailable

# HTTP headers

Client and server need to know which format is used for communication. Specify the format in the HTTP-headers

- *Content-type* define request format
- *Accept* define lists of response format

# Versioning your APIs

Removing any part of API for some reasons.

A change in the format of response data for one or more calls.

Change in response type (i.e. changing an integer to float).

- `https://example.com/api/v1/posts/1` this for a version one
- `https://example.com/api/v2/posts/1` this is for version two

# Pagination, Searching, Sort, filtering

## Pagination

- `GET /posts?limit=5&offset=10` This query would return the 5 rows starting with the 10th row.

## Sorting

- `GET /posts?sort=-created_at`

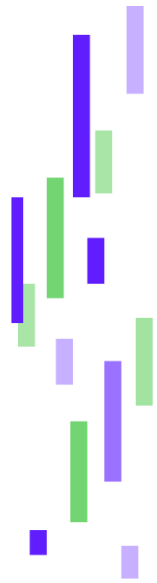
## Searching

- `GET /posts?status=viewed&tag=technology`

## Filtering

- `GET /users/123/posts?status=viewed`

# JSON Naming Convention



*Explained*

camelCase  
snake\_case  
PascalCase



camelCase



kebab-case



snake\_case



```
{  
  "thisPropertyIsAnIdentifier": "identifier value"  
}
```