Università di Corsica Pasquale Paoli

LABORATOIRE SCIENCES POUR L'ENVIRONNEMENT UMR 6134 SPE

cnrs

UE 1 : Développement d’applications web

Paul-Antoine Bisgambiglia

2023-2024

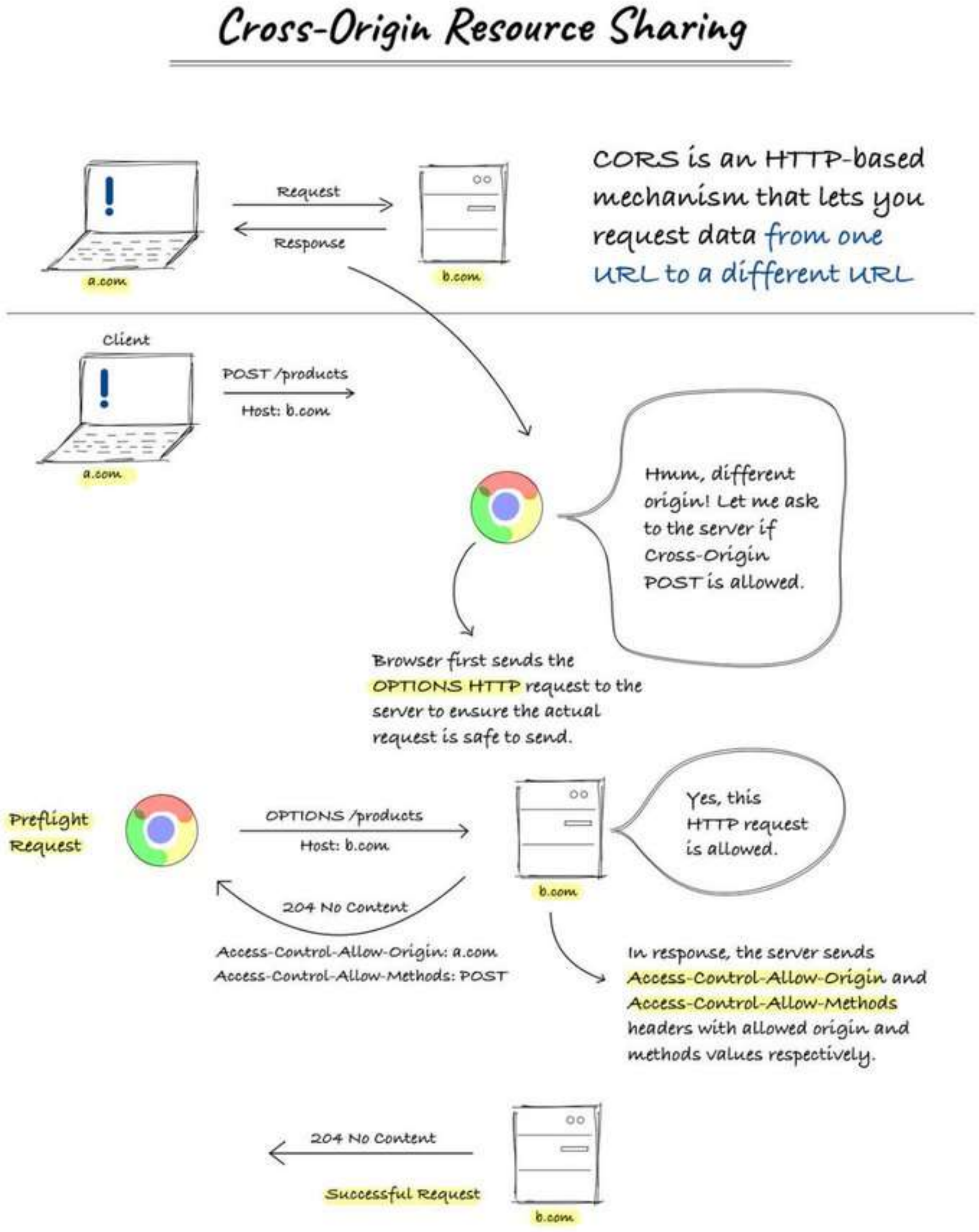
Sommaire

- **Cross-Origin Resource Sharing**
- API vs graphQL
- Patterns
- Kafka
- Redis
- Selenium
- Sécurité

What is Cross-Origin Resource Sharing (CORS)?

Browsers use CORS, a method, to prevent websites from requesting data from different URLs. A request from a browser includes an origin header in the request message. The browser allows it if it gets to the server of the exact origin; if not, the browser blocks it. We can deal with CORS issues on the backend. Cross-origin requests require that the values for origin and **Access-Control-Allow-Origin** in the response headers match and it is set by the server. When you add an origin to the backend code, the CORS middleware only permits this URL to communicate with other origins and utilize it for cross-origin resource requests.

What is Cross-Origin Resource Sharing (CORS)?



Cross-Origin Resource Sharing

CORS is an HTTP-based mechanism that lets you request data from one URL to a different URL

Hmm, different origin! Let me ask to the server if Cross-Origin POST is allowed.

Browser first sends the OPTIONS HTTP request to the server to ensure the actual request is safe to send.

Yes, this HTTP request is allowed.

In response, the server sends Access-Control-Allow-Origin and Access-Control-Allow-Methods headers with allowed origin and methods values respectively.

What is Cross-Origin Resource Sharing (**CORS**)?

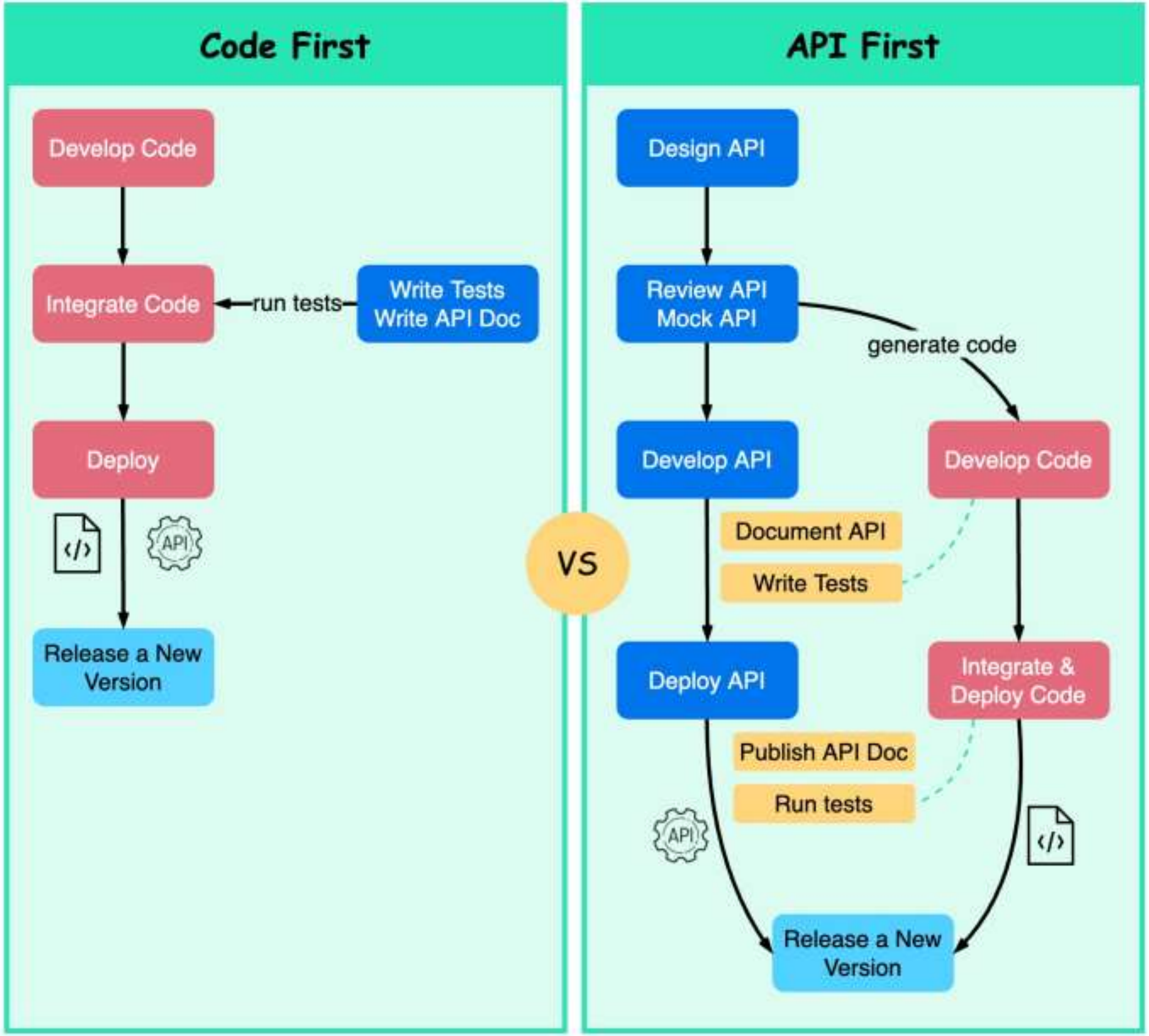
There are two ways to fix CORS issues:

1. **Configure the Backend to Allow CORS** Server can let all domains with **Access-Control-Allow-Origin**: *. This actually turns off same-origin policy, which is not recommended. Another optin would be only to allow particular domain, which is better option, e.g., **Access-Control-Allow-Origin**: **https**://**somedomain**.**com**.
2. **Use a Proxy Server** We can use a proxy server to call external API. It acts as a middleware between client and the server. If the server doesn't return proper headers defined by CORS, we can add them in the proxy. Image credits: RapidAPI

API

Code First v.s API First Development

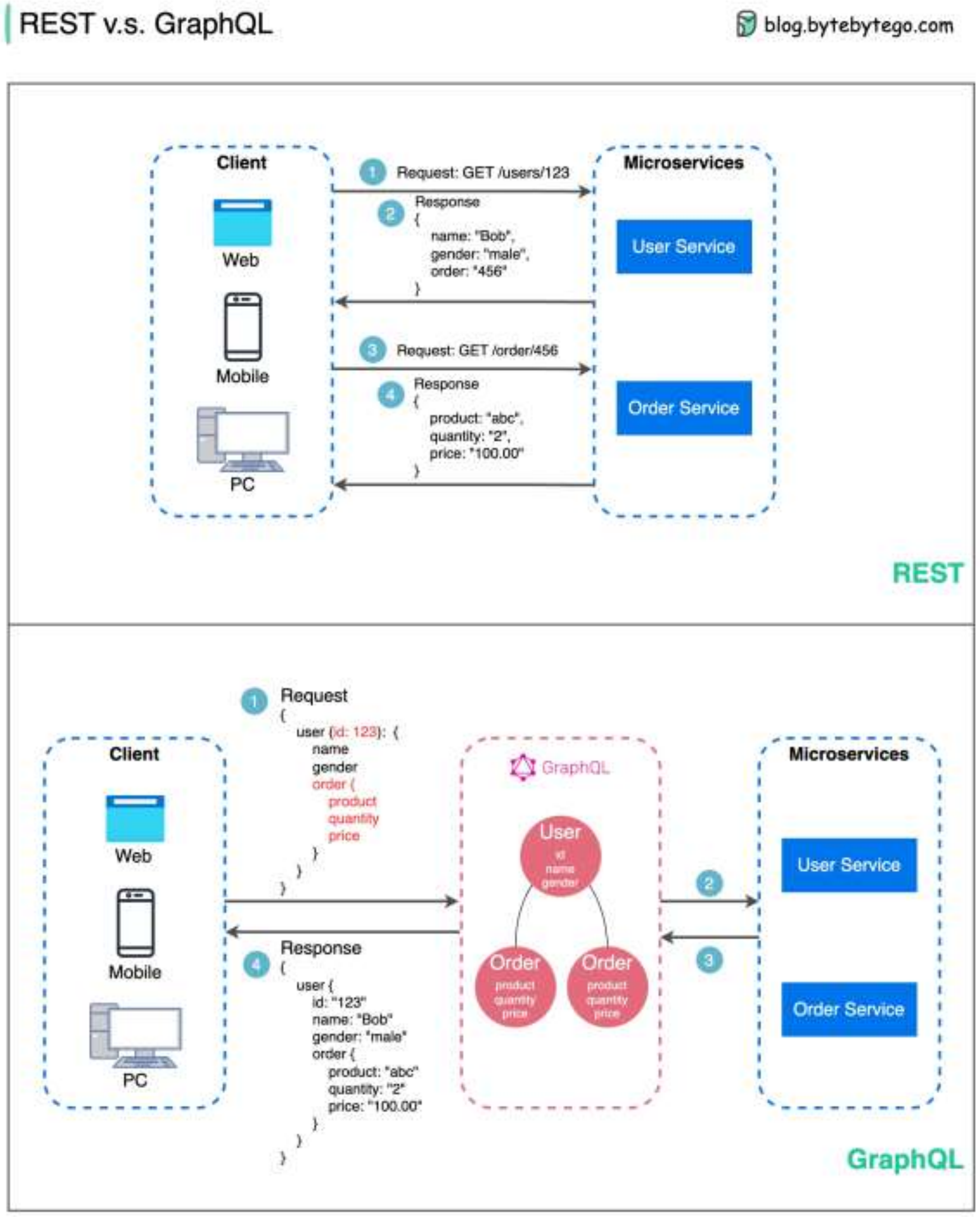
blog.bytebytego.com



API vs GraphQL

REST

- Uses standard HTTP methods like GET, POST, PUT, DELETE for CRUD operations.
- Works well when you need simple, uniform interfaces between separate services/applications.
- Caching strategies are straightforward to implement.
- The downside is it may require multiple roundtrips to assemble related data from separate endpoints.



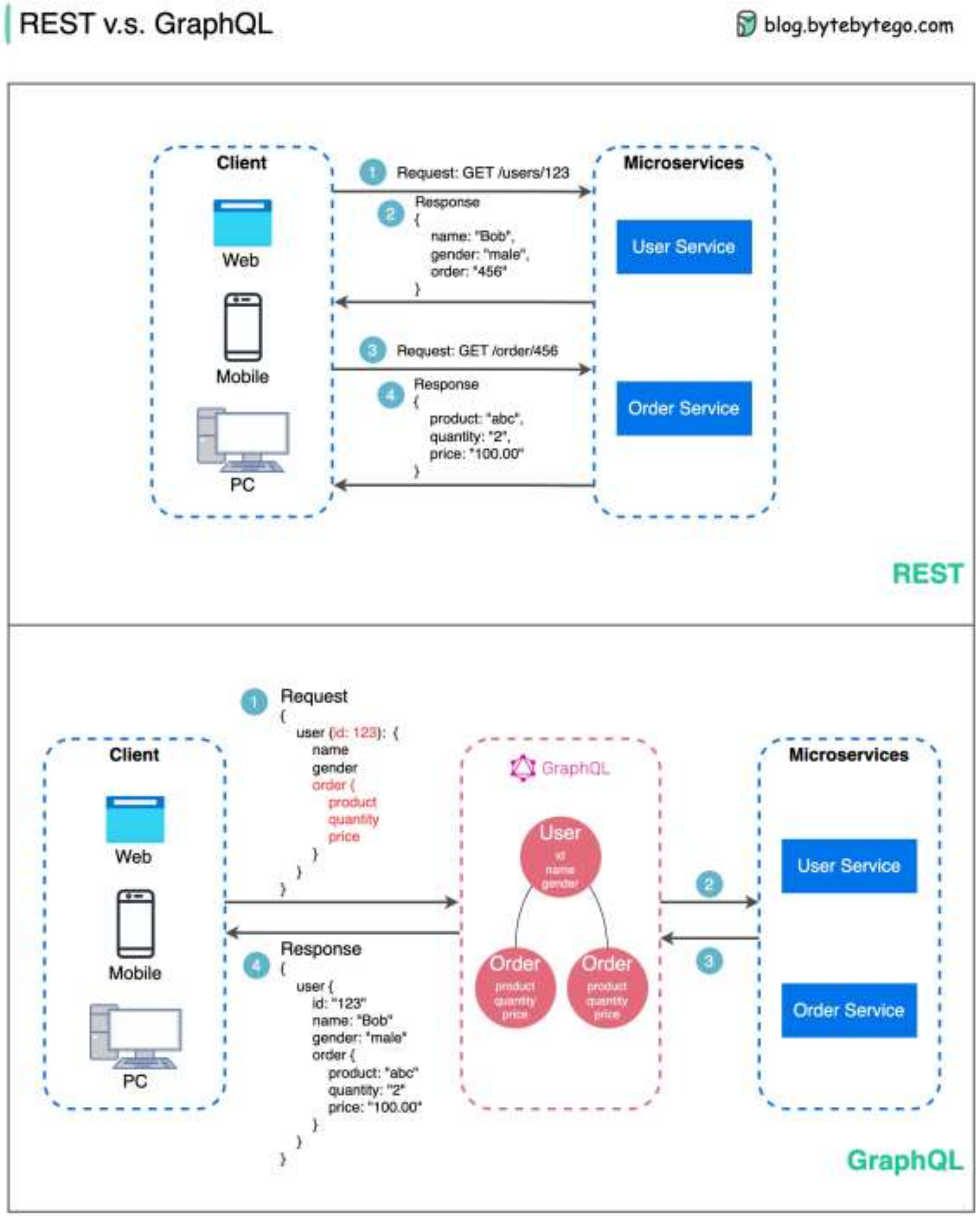
```
1 Request: GET /users/123
2 Response
{
    name: "Bob",
    gender: "male",
    order: "456"
}
3 Request: GET /order/456
4 Response
{
    product: "abc",
    quantity: "2",
    price: "100.00"
}
```

```
1 Request
{
  user (id: 123): {
    name
    gender
    order {
      product
      quantity
      price
    }
  }
}
4 Response
{
  user {
    id: "123"
    name: "Bob"
    gender: "male"
    order {
      product: "abc"
      quantity: "2"
      price: "100.00"
    }
  }
}
```

API vs GraphQL

GraphQL

- Provides a single endpoint for clients to query for precisely the data they need.
- Clients specify the exact fields required in nested queries, and the server returns optimized payloads containing just those fields.
- Supports Mutations for modifying data and Subscriptions for real-time notifications.
- Great for aggregating data from multiple sources and works well with rapidly evolving frontend requirements.
- However, it shifts complexity to the client side and can allow abusive queries if not properly safeguarded
- Caching strategies can be more complicated than REST.



1 Request: GET /users/123

2 Response
```
{
   name: "Bob",
   gender: "male",
   order: "456"
}
```

3 Request: GET /order/456

4 Response
```
{
   product: "abc",
   quantity: "2",
   price: "100.00"
}
```

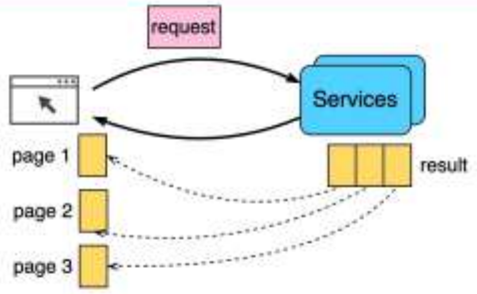
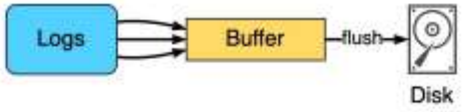
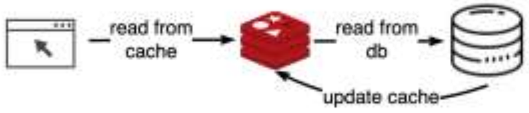
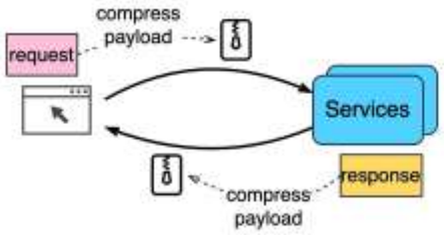
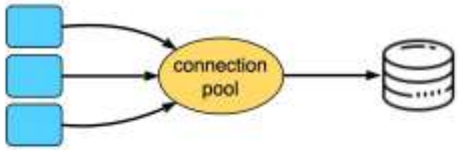
1 Request
```
{
   user (id: 123): {
      name
      gender
      order {
         product
         quantity
         price
      }
   }
}
```

4 Response
```
{
   user {
      id: "123"
      name: "Bob"
      gender: "male"
      order {
         product: "abc"
         quantity: "2"
         price: "100.00"
      }
   }
}
```

API vs GraphQL

- The best choice between REST and GraphQL depends on the specific requirements of the application and development team. GraphQL is a good fit for complex or frequently changing frontend needs, while REST suits applications where simple and consistent contracts are preferred.
- Neither API approach is a silver bullet. Carefully evaluating requirements and tradeoffs is important to pick the right style. Both REST and GraphQL are valid options for exposing data and powering modern applications.

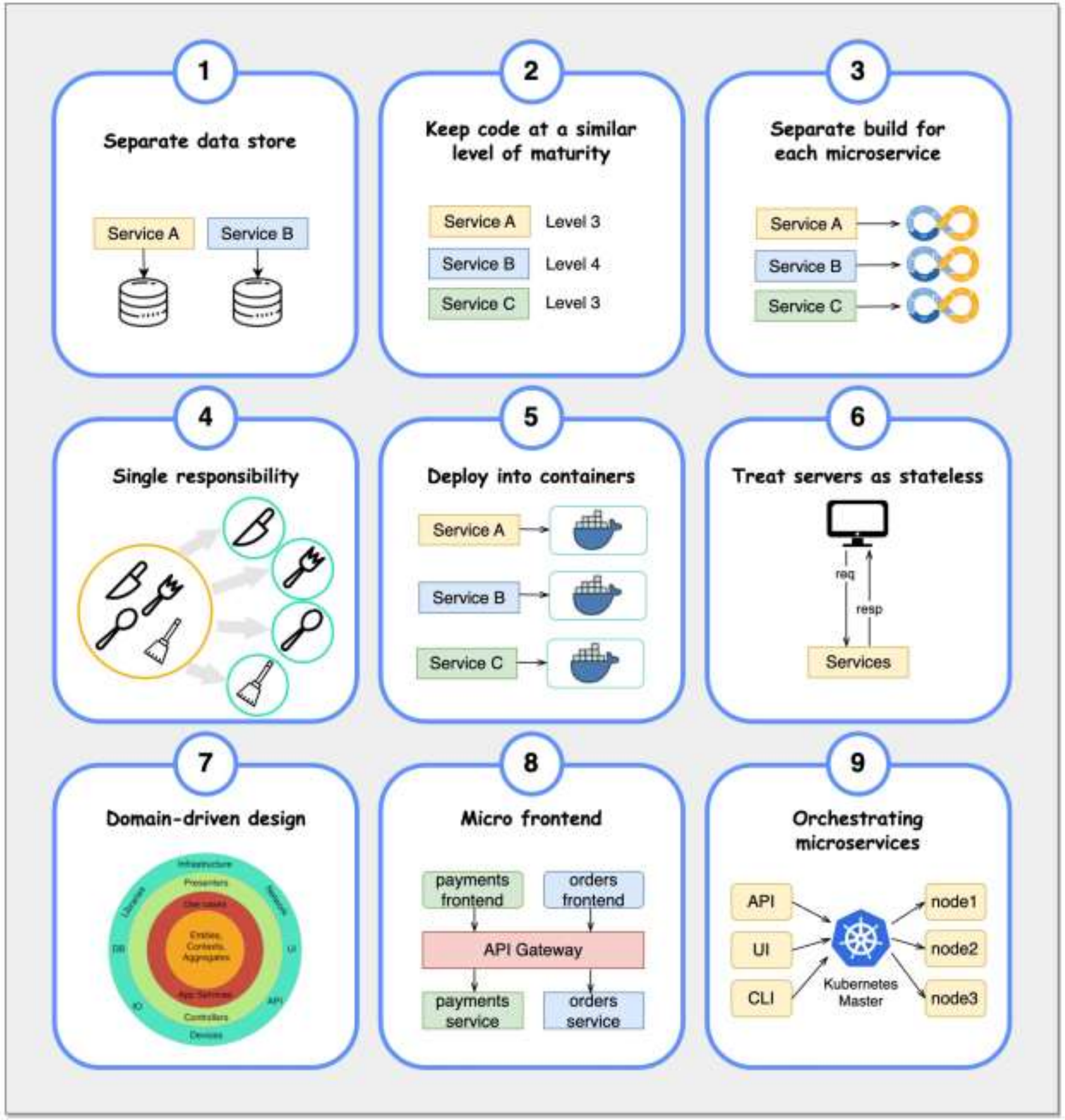
How to Improve API Performance?

PAGINATION	 	• an ordinal numbering of pages • handles a large number of results
ASYNC LOGGING	 	• send logs to a lock-free ring buffer and return • flush to the disk periodically • higher throughput and lower latency
CACHING	 	• store frequently used data in the cache instead of database • query the database when there is a cache miss
PAYLOAD COMPRESSION	 compress payload request Services response compress payload	• reduce the data size to speed up the download and upload
CONNECTION POOL	 	• opening and closing DB connections add significant overhead • a connection pool maintains a number of open connections for applications to reuse

Reference: Rapid API

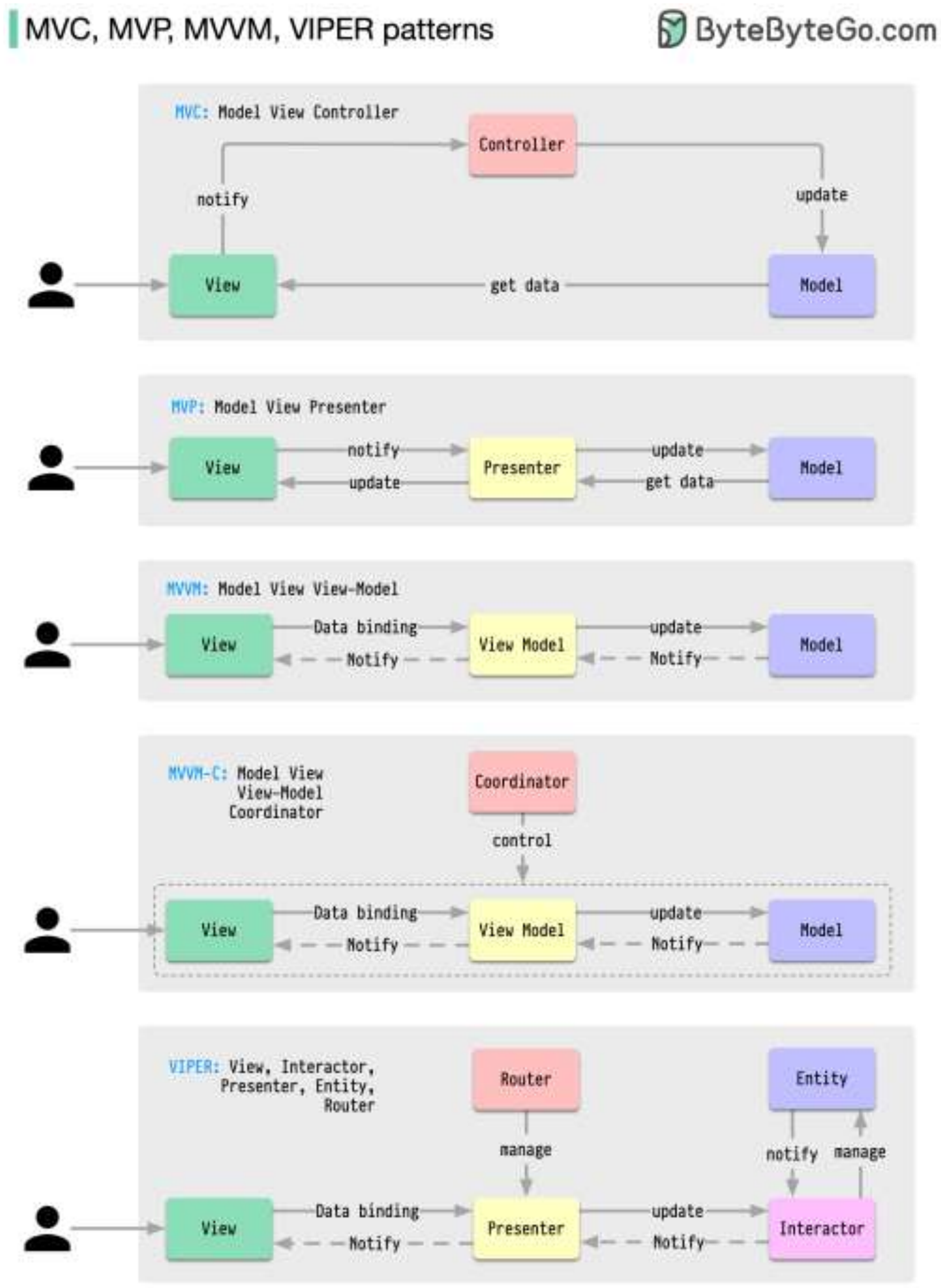
Microservice Best Practices

blog.bytebytego.com



1. Separate data store
2. Keep code at a similar level of maturity
3. Separate build for each microservice
4. Single responsibility
5. Deploy into containers
6. Treat servers as stateless
7. Domain-driven design
8. Micro frontend
9. Orchestrating microservices

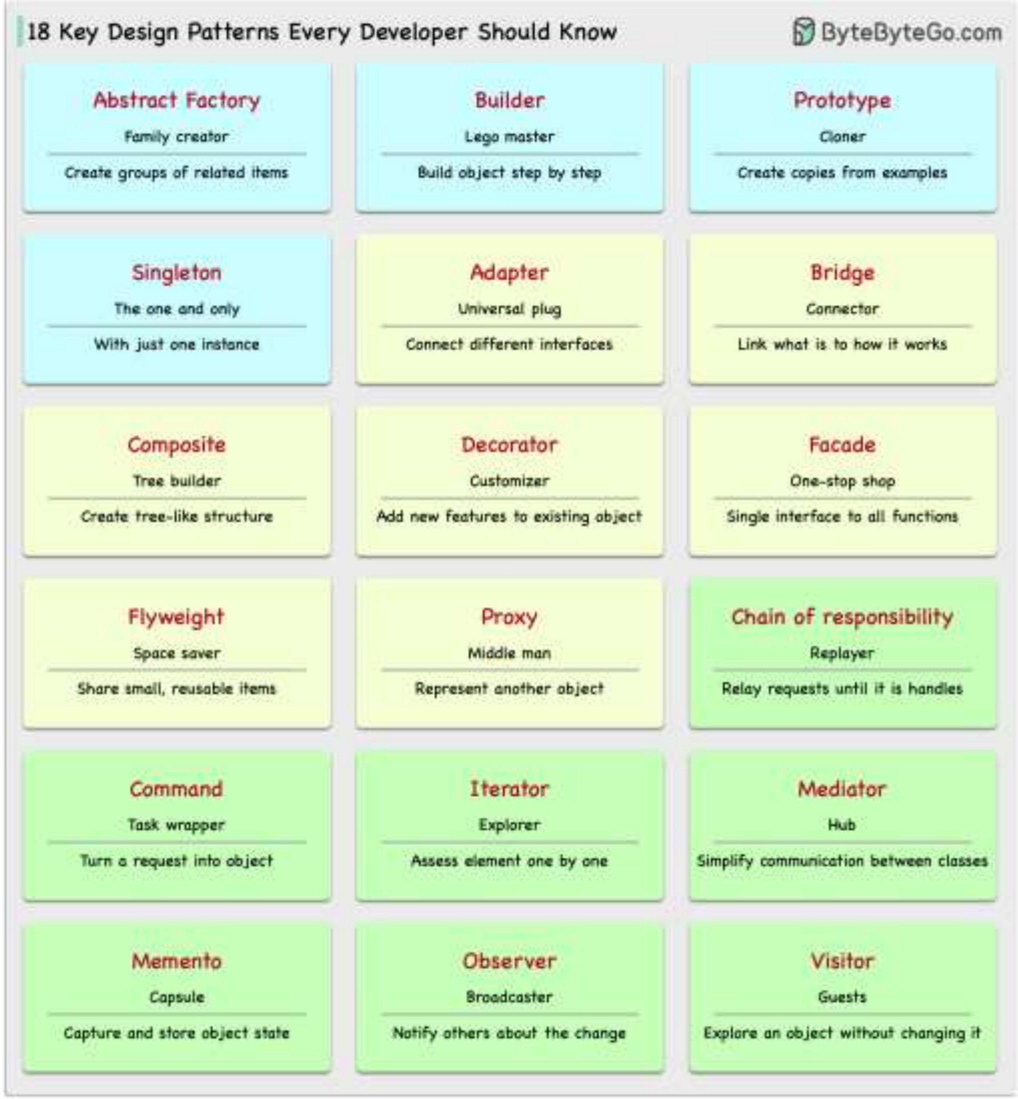
Patterns



Patterns

- MVC, the oldest pattern, dates back almost 50 years
- Every pattern has a "view" (V) responsible for displaying content and receiving user input
- Most patterns include a "model" (M) to manage business data
- "Controller," "presenter," and "view-model" are translators that mediate between the view and the model ("entity" in the VIPER pattern)

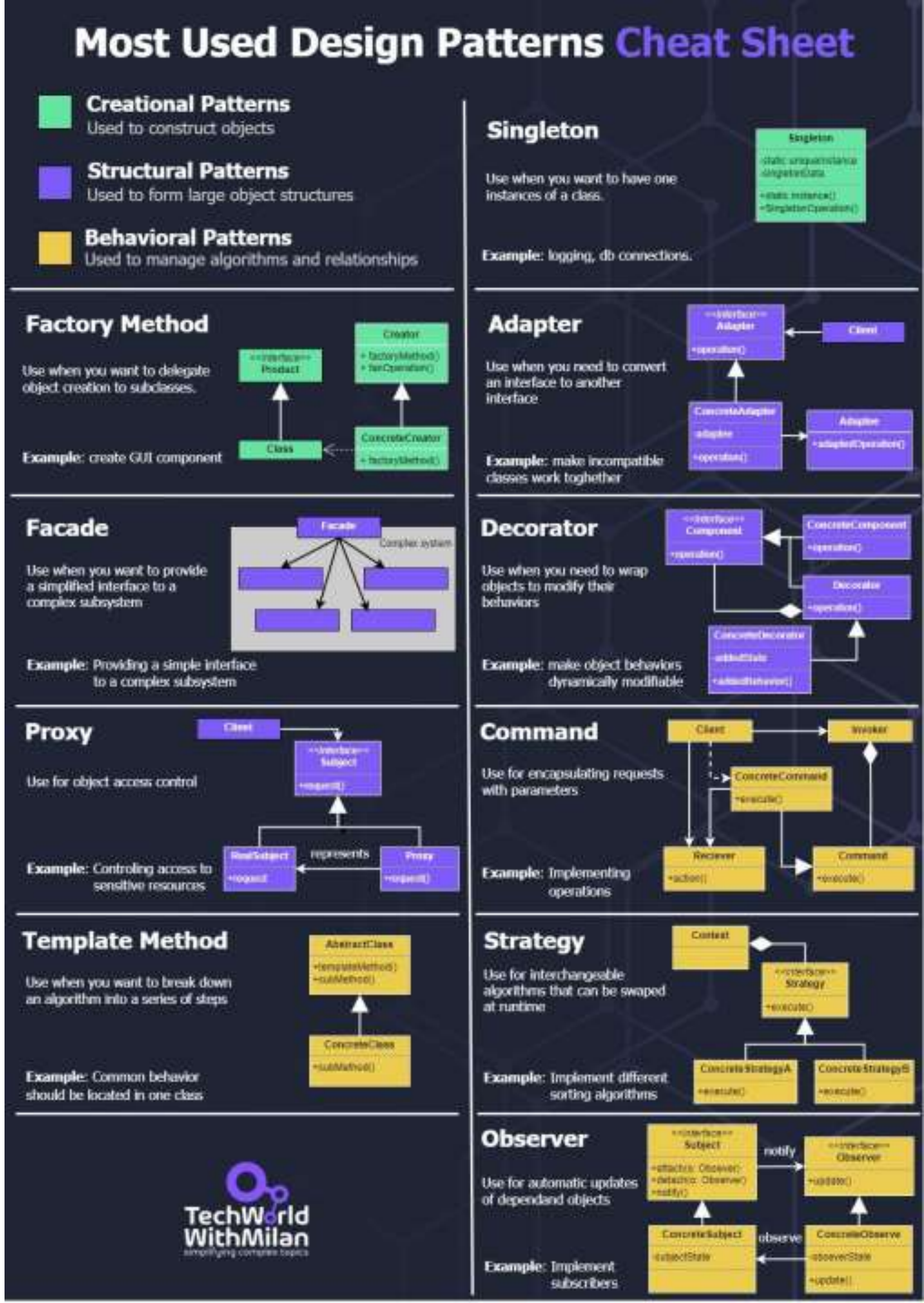
Patterns



18 Key Design Patterns Every Developer Should Know

ByteByteGo.com

Abstract Factory Family creator Create groups of related items	**Builder** Lego master Build object step by step	**Prototype** Cloner Create copies from examples
Singleton The one and only With just one instance	**Adapter** Universal plug Connect different interfaces	**Bridge** Connector Link what is to how it works
Composite Tree builder Create tree-like structure	**Decorator** Customizer Add new features to existing object	**Facade** One-stop shop Single interface to all functions
Flyweight Space saver Share small, reusable items	**Proxy** Middle man Represent another object	**Chain of responsibility** Replayer Relay requests until it is handles
Command Task wrapper Turn a request into object	**Iterator** Explorer Assess element one by one	**Mediator** Hub Simplify communication between classes
Memento Capsule Capture and store object state	**Observer** Broadcaster Notify others about the change	**Visitor** Guests Explore an object without changing it



Most Used Design Patterns Cheat Sheet

🟩 **Creational Patterns**
Used to construct objects

🟪 **Structural Patterns**
Used to form large object structures

🟨 **Behavioral Patterns**
Used to manage algorithms and relationships

Factory Method

Use when you want to delegate object creation to subclasses.

Example: create GUI component

Facade

Use when you want to provide a simplified interface to a complex subsystem

Example: Providing a simple interface to a complex subsystem

Proxy

Use for object access control

Example: Controling access to sensitive resources

Template Method

Use when you want to break down an algorithm into a series of steps

Example: Common behavior should be located in one class

TechWorld WithMilan
simplifying complex topics

Singleton

Use when you want to have one instances of a class.

Example: logging, db connections.

Adapter

Use when you need to convert an interface to another interface

Example: make incompatible classes work toghether

Decorator

Use when you need to wrap objects to modify their behaviors

Example: make object behaviors dynamically modifiable

Command

Use for encapsulating requests with parameters

Example: Implementing operations

Strategy

Use for interchangeable algorithms that can be swaped at runtime

Example: Implement different sorting algorithms

Observer

Use for automatic updates of dependand objects

Example: Implement subscribers

Creational Patterns

These design patterns deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. Important patterns in this group are:

Factory:

This pattern allows delegating the instantiation logic to factory classes. The Factory Method creates objects without exposing the instantiation logic to the client.

Singleton

The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. It's useful when exactly one object is needed to coordinate actions across the system.

Structural Patterns

These patterns deal with the composition of classes and objects that form larger structures. Important patterns in this group are:

Adapter:

This pattern works as a bridge between two incompatible interfaces. It wraps an existing class with a new interface to become compatible with the client's interface.

Structural Patterns

Facade:

The Façade pattern provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.

Decorator:

This pattern dynamically adds/overrides behavior in an existing method of an object. This pattern provides a flexible alternative to subclassing for extending functionality.

Proxy:

The Proxy pattern provides a surrogate or placeholder for another object to control access to it. In its most general form, a proxy is a class functioning as an interface to something else.

Behavioral Patterns

These patterns are specifically concerned with communication between objects and how they interact and distribute work. Important patterns in this group are:

Command:

The Command pattern encapsulates a request as an object, thus allowing users to parameterize clients with queues, requests, and operations.

Behavioral Patterns

Template Method:

This pattern defines the program skeleton of an algorithm in a method called template method, which defers some steps to subclasses.

Strategy:

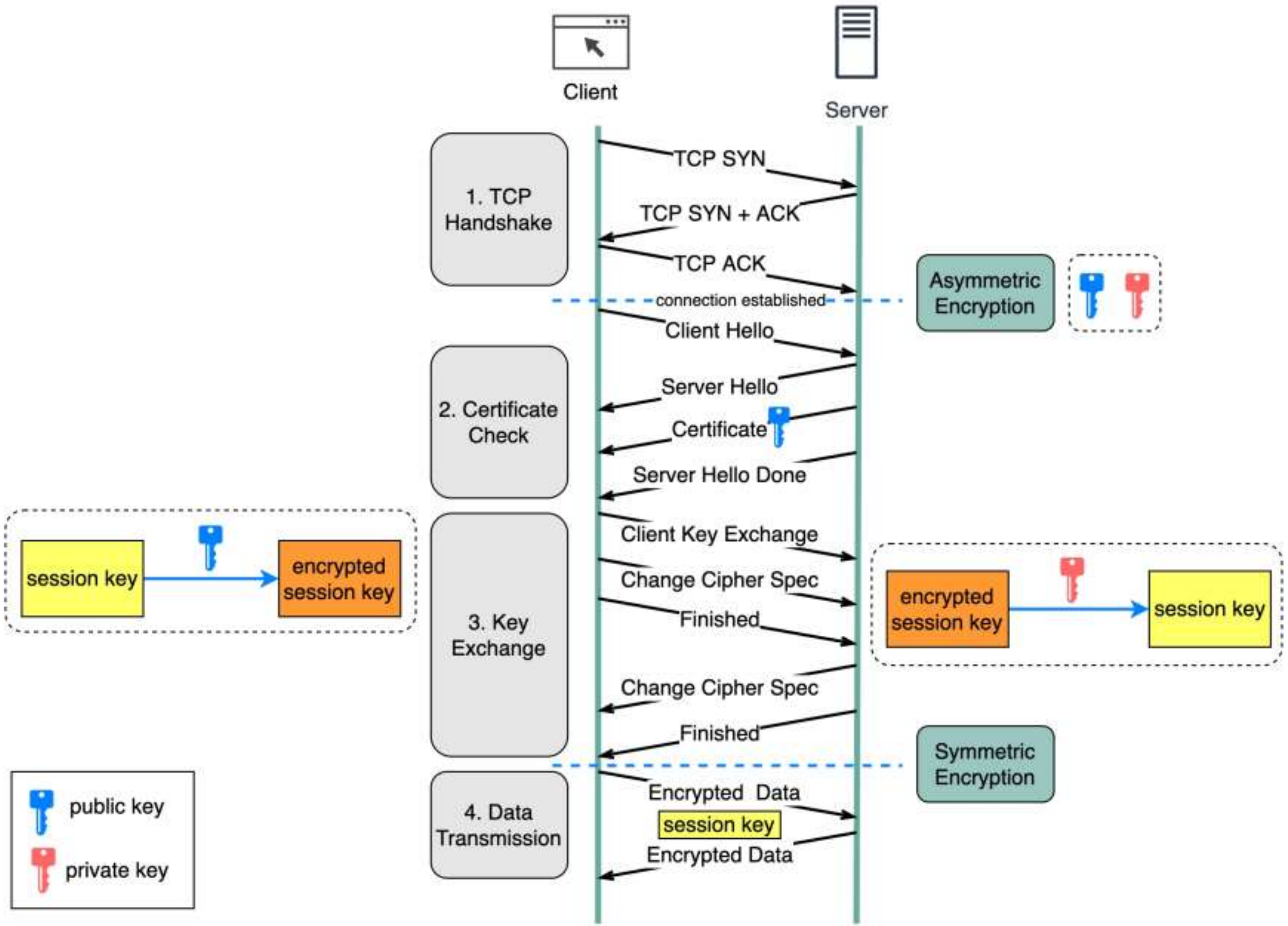
The Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it. **Observer**: This pattern defines a one-to-many dependency between objects so that all its dependents are notified and updated automatically when one object changes state.

Sécurité web

https://github.com/ByteByteGoHq/system-design-101/tree/main#security

How does HTTPS Work?

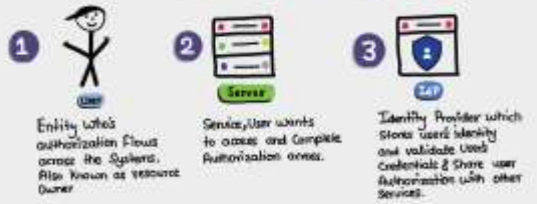
blog.bytebytego.com



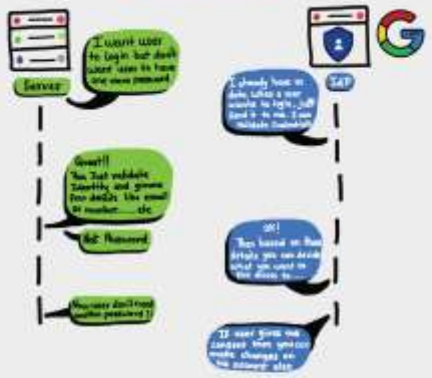
What is OAuth?

* Protocol for sharing user Authorization across systems.

Involves 3 entities


1
User
Entity who's authorization Flows across the Systems. Also known as resource Owner
2
Server
Service, User wants to access and Complete Authorization access.
3
IdP
Identity Provider which Stores user's identity and validate User's Credentials & Share user Authorization with other Services.

OAuth


Server
IdP
I want user to login but don't want user to have one more password
I already have all data, when a user wants to login, just send it to me. I can validate Credentials
Great!! You Just validate Identity and give me few details like email Id number .... etc
Not Password
OK! Then based on these details you can decide what you want to do ....
Now users don't need another password !!
If user gives me consent then you can make changes on his account also

1.0 protocal designed for web browser only

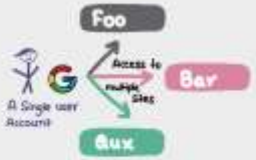
2.0 protocal upgraded for
browser App
non browser App
Windows App
mobile App
APIs

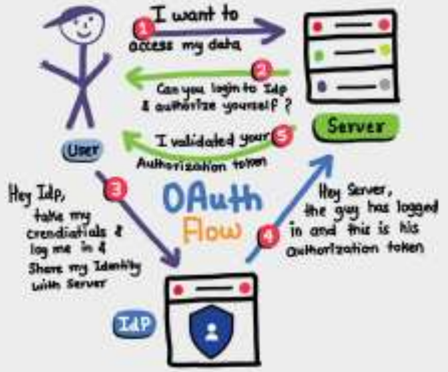
OAUTH

@ Sec_10

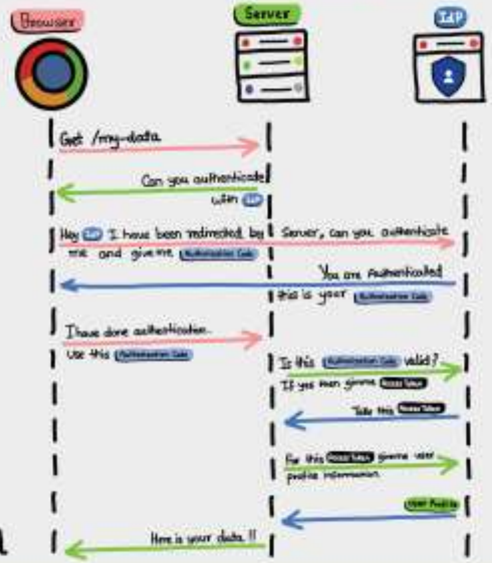
ByteByteGo

Open Authorization 2.0




1 I want to access my data
2 Can you login to Idp & authorize yourself ?
Server
User
5 I validated your Authorization token
3 Hey Idp, take my crendiatials & log me in & Share my Identity with Server
OAuth Flow
4 Hey Server, the guy has logged in and this is his authorization token
IdP

Authorization Code Flow

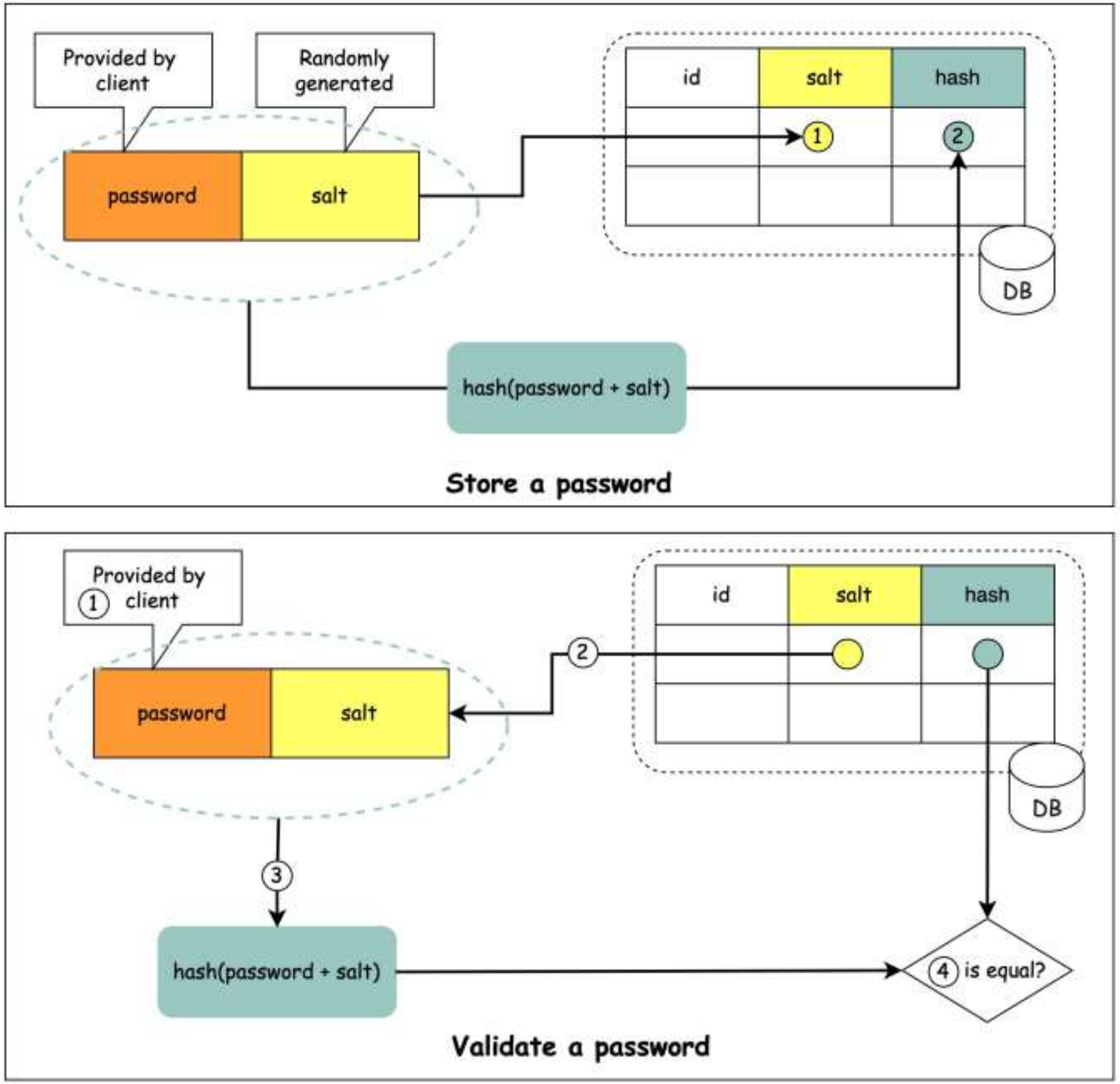

Browser
Server
IdP
Get /my-data
Can you authenticate with IdP
Hey IdP I have been redirected by Server, can you authenticate me and give me Authorization Code
You are Authenticated this is your Authorization Code
I have done authentication. Use this Authorization Code
Is this Authorization Code valid? If yes then give me Access Token
Take this Access Token
For this Access Token give me user profile information
User Profile
Here is your data !!

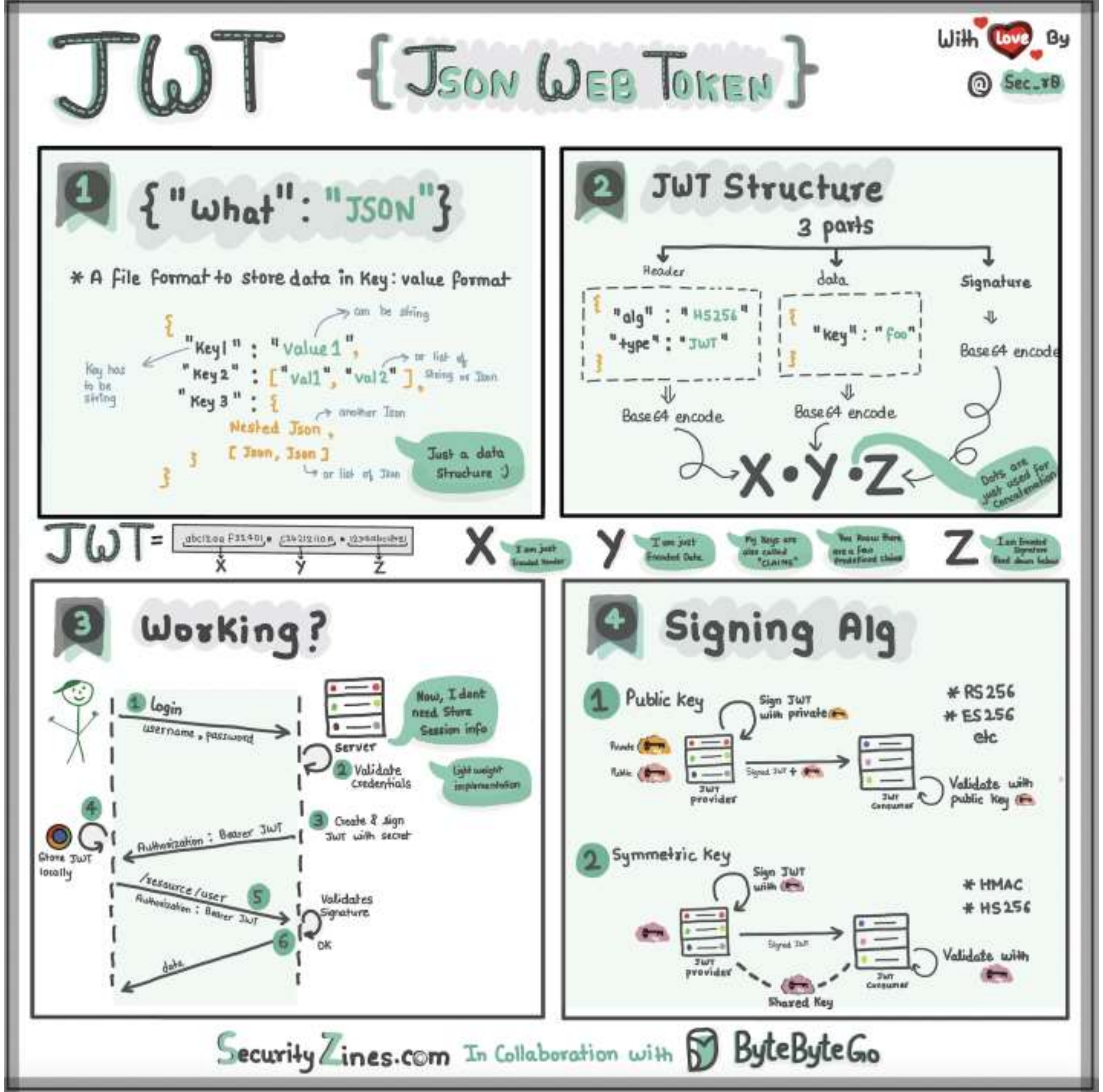
4 Types of OAuth Flows

1. Authorization code
2. Client Credentials
3. Implicit Code
4. Resourse owner Password

How to store passwords in DB?

blog.bytebytego.com




JWT
{ JSON WEB TOKEN }
With love By
@ Sec_RB
1
{ "what" : "JSON" }
* A file format to store data in Key : value format
{
"Key1" : "Value1",
can be string
"Key2" : ["Val1", "Val2"],
or list of String or Json
"Key3" : {
Nested Json,
another Json
[Json, Json]
or list of Json
}
}
Key has to be string
Just a data Structure :)
2
JWT Structure
3 parts
Header
data
Signature
{ "alg" : "HS256" "type" : "JWT" }
{ "key" : "foo" }
Base64 encode
Base64 encode
Base64 encode
X • Y • Z
Dots are just used for Concatenation
JWT = abc120a F3540 . c3421210a . 1234abcde
X
Y
Z
X
I am just Encoded Header
Y
I am just Encoded Data
My keys are also called "CLAIMS"
You know there are a few predefined claims
Z
I am Encoded Signature Read down below
3
Working?
1 login
username, password
Server
Now, I dont need Store Session info
2 Validate credentials
light weight implementation
3 Create & sign JWT with secret
Authorization : Bearer JWT
4
Store JWT locally
/resource/user
Authorization : Bearer JWT
5
Validates Signature
OK
6
data
4
Signing Alg
1 Public Key
Sign JWT with private
Private
Public
JWT provider
Signed JWT +
JWT Consumer
Validate with public Key
* RS256
* ES256
etc
2 Symmetric Key
Sign JWT with
JWT provider
Signed JWT
Shared Key
JWT Consumer
Validate with
* HMAC
* HS256
SecurityZines.com In Collaboration with ByteByteGo

Apache Kafka, qu'est-ce que c'est ?

- Apache Kafka est une plateforme distribuée de diffusion de données en continu, capable de publier, stocker, traiter et souscrire à des flux d'enregistrement en temps réel.

- https://www.redhat.com/fr/topics/integration/what-is-apache-kafka

Redis

- Redis est une technologie sous license BSD qui est notamment utilisée en tant que cache, agent de message (broker) et pour l'enregistrement de structures complexes pouvant persister sur disque.

- https://linuxembedded.fr/2020/10/introduction-a-redis-une-base-de-donnees-in-memory-cle-valeur
- https://grafikart.fr/tutoriels/redis-bases-783

Selenium

- Selenium WebDriver est un framework web qui vous permet d'exécuter des tests multi-navigateurs. Cet outil est utilisé pour automatiser les tests d'applications Web pour vérifier qu'ils fonctionnent correctement

- https://www.all4test.fr/blog-du-testeur/commencer-avec-selenium-webdriver/
- https://api.pkstate.com/phunit_4.2/fr/selenium.html
- https://www.youtube.com/watch?v=_JnNoalyvLQ

Links

- https://github.com/ByteByteGoHq/system-design-101#mvc-mvp-mvvm-mvvm-c-and-viper
- https://www.redhat.com/fr/topics/integration/what-is-apache-kafka
- https://redis.com/fr/