# API Design and Management

Mohamed Sweelam

Software Engineer

# Outline


1. Course Objectives
2. Understanding APIs
3. API Design Principles
4. RESTful API Design
5. API Documentation and Specification
6. API Security
7. API Testing and Quality Assurance
8. API Management and Lifecycle
9. Conclusion


# Course Objectives

1. Provide good Arabic content for the topic

# Course Objectives

1. Provide good Arabic content for the topic
2. Overview of API Design and Management

# Course Objectives

1. Provide good Arabic content for the topic
2. Overview of API Design and Management
3. Role and Importance of APIs in Distributed Systems

# Course Objectives

1. Provide good Arabic content for the topic
2. Overview of API Design and Management
3. Role and Importance of APIs in Distributed Systems
4. The best practices you should follow today

# Understanding APIs

## Definition *wikipedia*

Application programming interface (API) is a way for two or more computer programs or components to communicate with each other. It is a type of software interface, offering a service to other pieces of software.

## Definition *ChatGPT*

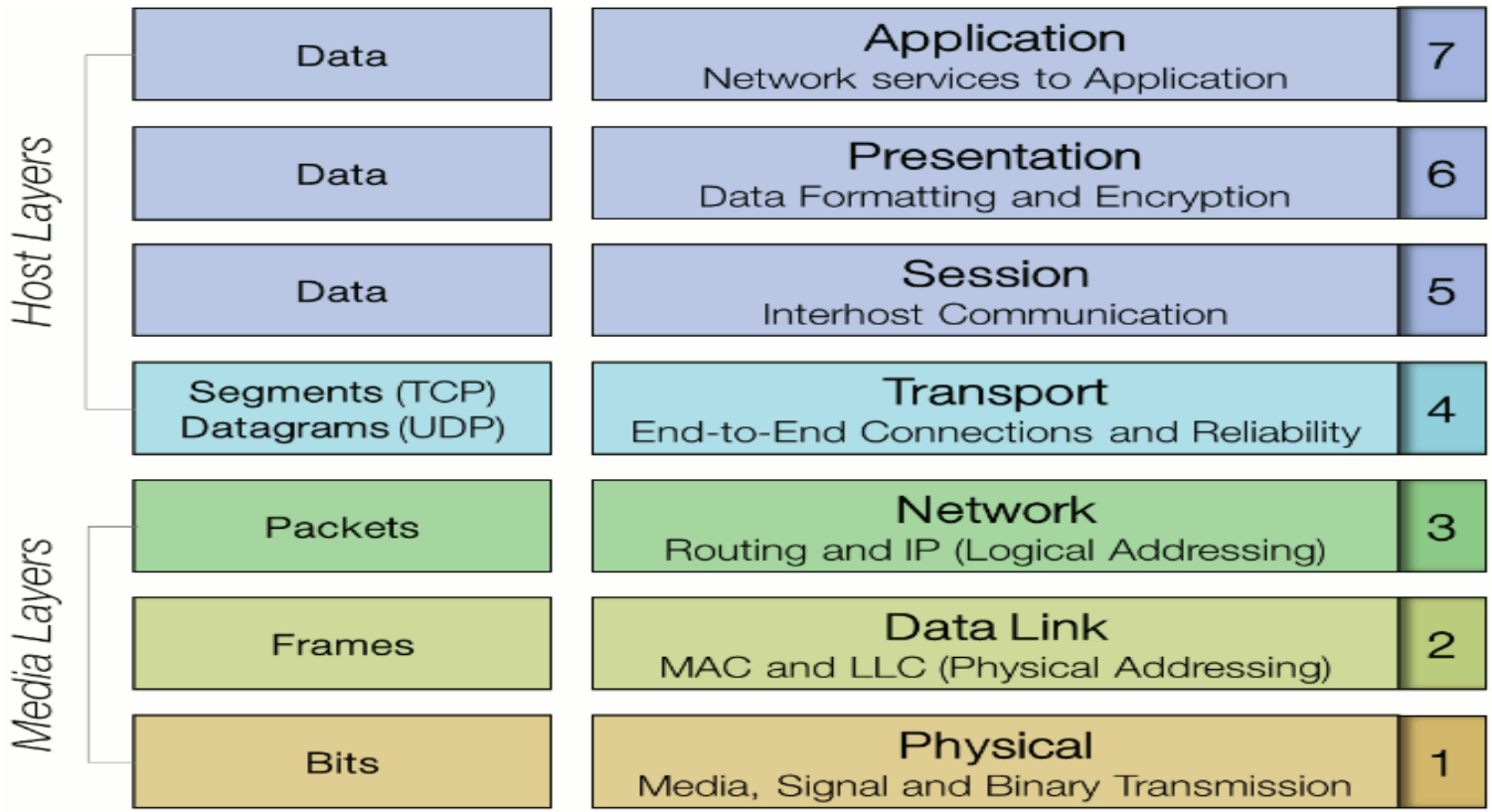
API (Application Programming Interface) is a set of rules, protocols, and tools for building software applications. It specifies how software components should interact and is used to enable the integration between different software systems.

## History *wikipedia*

The term "application program interface" is first recorded in a paper called Data structures and techniques for remote computer graphics in 1968. The authors use the term to describe the interaction of an application "Graphics Program" with the rest of the computer system.

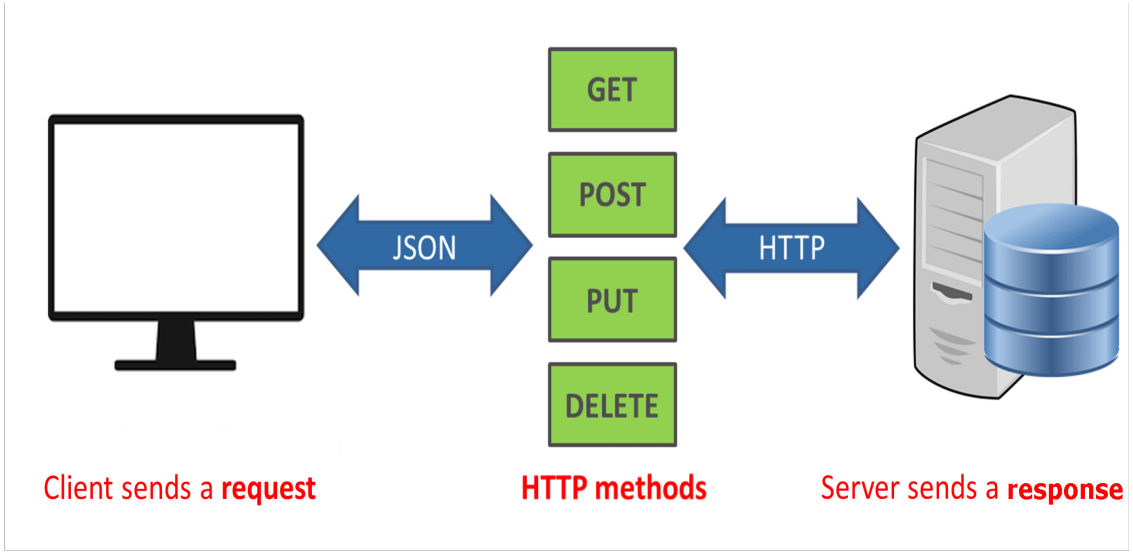
# Understanding APIs *OSI Model*

The open systems interconnection (OSI) model is a conceptual model created for Standardization which enables diverse communication systems to communicate using standard protocols.




source: coengoedegebure.com/osi-model

# Understanding APIs Example




source: phpenthusiast.com/blog/what-is-rest-api

# Understanding APIs (API Types)

There are several types of API, each one serves specific use case.

- Public APIs (Open APIs)
  The APIs are publicly available and can be designed in various ways, taking security into account. However, the main priority is to ensure they are easily consumable by as many clients as possible.

# Understanding APIs (API Types)

There are several types of API, each one serves specific use case.

- **Public APIs (Open APIs)**
  The APIs are publicly available and can be designed in various ways, taking security into account. However, the main priority is to ensure they are easily consumable by as many clients as possible.
- **Partner APIs**
  Specialized interfaces that enable organizations to access data and service offerings across businesses (B2B) in order to create unique features within their own applications or services by utilizing a partner's resources.

# Understanding APIs (API Types)

There are several types of API, each one serves specific use case.

- **Public APIs (Open APIs)**
  The APIs are publicly available and can be designed in various ways, taking security into account. However, the main priority is to ensure they are easily consumable by as many clients as possible.
- **Partner APIs**
  Specialized interfaces that enable organizations to access data and service offerings across businesses (B2B) in order to create unique features within their own applications or services by utilizing a partner's resources.
- **Internal APIs**
  Intended for use internally by the organizations own developers. These APIs facilitate the transmission of data between different components of a system, enabling process automation.

# Understanding APIs (API Types)

There are several types of API, each one serves specific use case.

- **Public APIs (Open APIs)**
  The APIs are publicly available and can be designed in various ways, taking security into account. However, the main priority is to ensure they are easily consumable by as many clients as possible.
- **Partner APIs**
  Specialized interfaces that enable organizations to access data and service offerings across businesses (B2B) in order to create unique features within their own applications or services by utilizing a partner's resources.
- **Internal APIs**
  Intended for use internally by the organizations own developers. These APIs facilitate the transmission of data between different components of a system, enabling process automation.
- **Composite APIs** Executes a series of API requests in a single call.

# Understanding APIs (API Types)

There are several types of API, each one serves specific use case.

- **Public APIs (Open APIs)**
  The APIs are publicly available and can be designed in various ways, taking security into account. However, the main priority is to ensure they are easily consumable by as many clients as possible.
- **Partner APIs**
  Specialized interfaces that enable organizations to access data and service offerings across businesses (B2B) in order to create unique features within their own applications or services by utilizing a partner's resources.
- **Internal APIs**
  Intended for use internally by the organizations own developers. These APIs facilitate the transmission of data between different components of a system, enabling process automation.
- **Composite APIs** Executes a series of API requests in a single call.

The types can be designed and developed using two ways

1. API Architectural Style,
2. API Standard Protocol.

# Understanding APIs (API Architecture vs API Protocols)

## API Architecture Style

It refers to the high-level structural design of the API. It encompasses the standards, and best practices governing how the API is developed, how it interacts with other systems, and how it exposes its functionality and data.
Example: REST

## No Restrictions

Architectural style is sensitive to change and enhancement; it relies more on human experience.

## API Protocol

It refers to a set of rules and standards used for communication between various software components. The protocol dictates how requests and responses are formatted and transmitted, and what are the restrictions of the communication.
Example: SOAP

# Understanding APIs (SOAP API)

- SOAP (Simple Object Access Protocol) is a protocol used for exchanging structured information in web services in computer networks.

# Understanding APIs (SOAP API)

- SOAP (Simple Object Access Protocol) is a protocol used for exchanging structured information in web services in computer networks.
- It's a standards-based web services access protocol that has been around for a long time.

# Understanding APIs (SOAP API)

- SOAP (Simple Object Access Protocol) is a protocol used for exchanging structured information in web services in computer networks.
- It's a standards-based web services access protocol that has been around for a long time.
- It relies on XML as its message format and usually relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

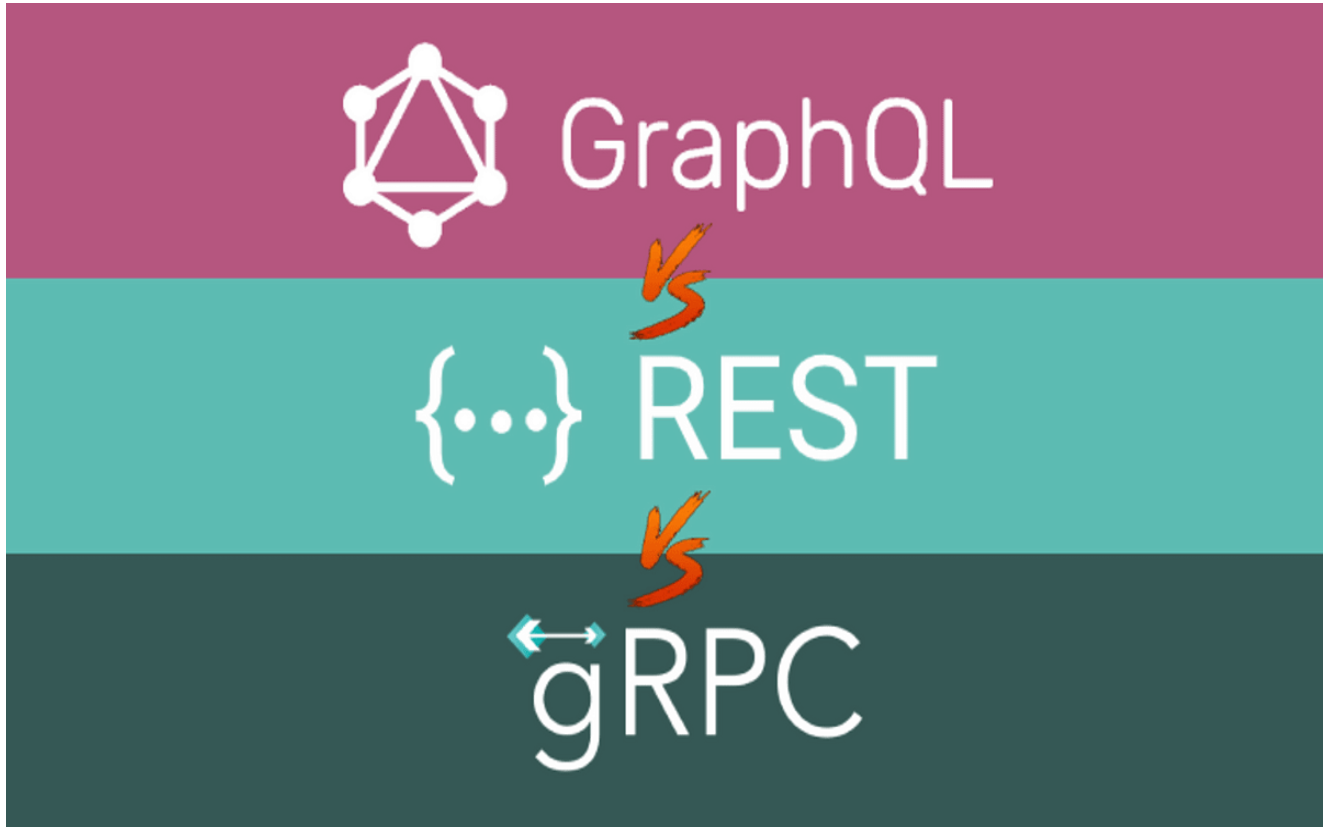
# Understanding APIs (SOAP API)

- SOAP (Simple Object Access Protocol) is a protocol used for exchanging structured information in web services in computer networks.
- It's a standards-based web services access protocol that has been around for a long time.
- It relies on XML as its message format and usually relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!-- header information here -->
  </soap:Header>
  <soap:Body>
    <m:GetPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetPrice>
  </soap:Body>
  <soap:Fault>
    <!-- fault information here -->
  </soap:Fault>
</soap:Envelope>
```

# Understanding APIs (Other Protocols)

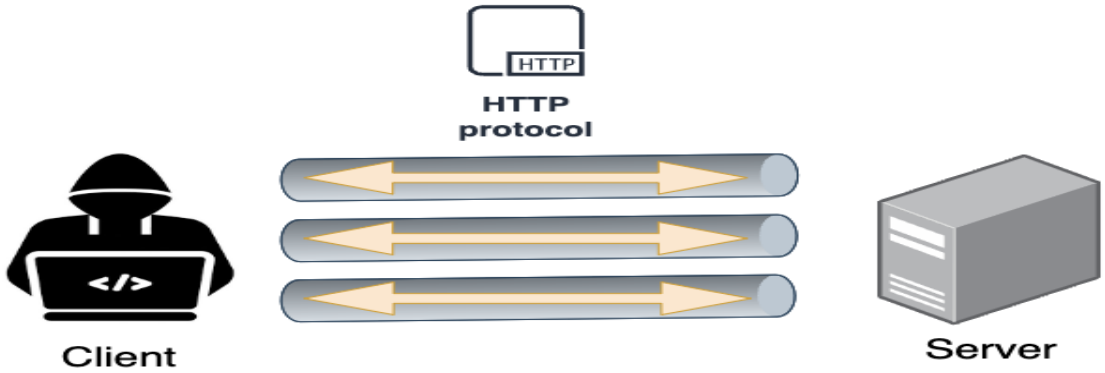
- gRPC
- REST
- GraphQL

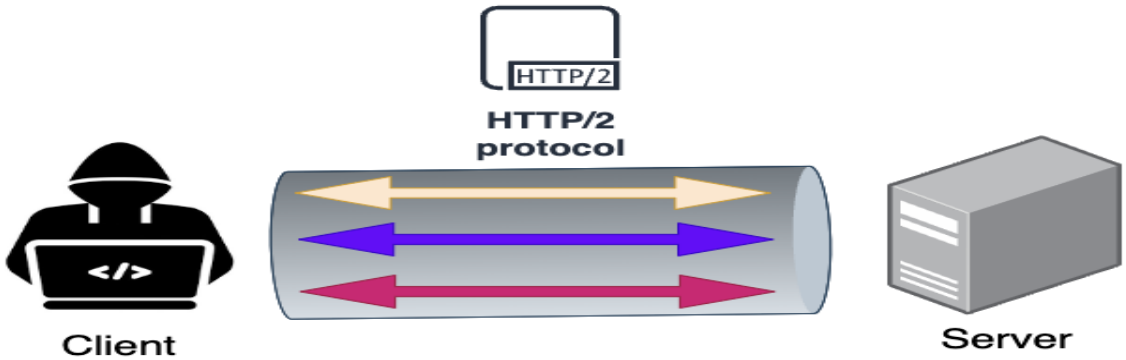



source: REST vs GraphQL vs gRPC: Comparing Three Modern API Technologies

# Understanding APIs (HTTP1 vs HTTP2)

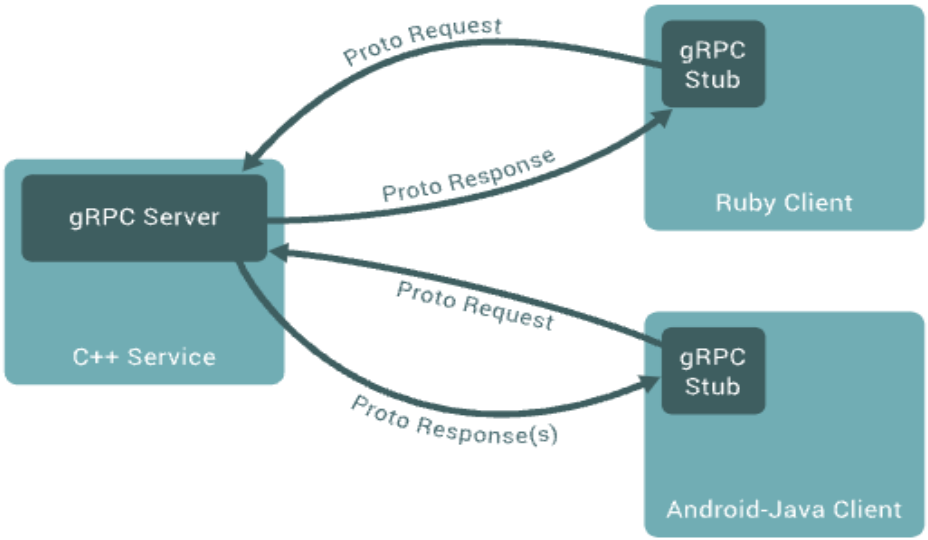
| HTTP/1.1 | HTTP/2 |
|---|---|
| Text-Based Protocol: Data is sent in a text-based format. | Binary Protocol: More efficient binary protocol, easier to parse. |
| One Request Per Connection: Each TCP connection allows only one request-response cycle at a time. | Multiplexing: Multiple requests and responses can be handled over a single TCP connection in parallel. |
| Headers Uncompressed: Headers are sent in plain text and can be quite large. | Header Compression: Uses HPACK compression to reduce overhead. |
| No Push Capabilities | Server Push |




# Understanding APIs (gRPC Protocol)

- gRPC is a modern open source high performance Remote Procedure Call (RPC) framework that was developed in Google.
- gRPC can use **protocol buffers** as its underlying message interchange format.
- gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types.
- On the server side, the server implements this interface and runs a gRPC server to handle client calls.
- On the client side, the client has a stub (referred to as just a client in some languages) that provides the same methods as the server.

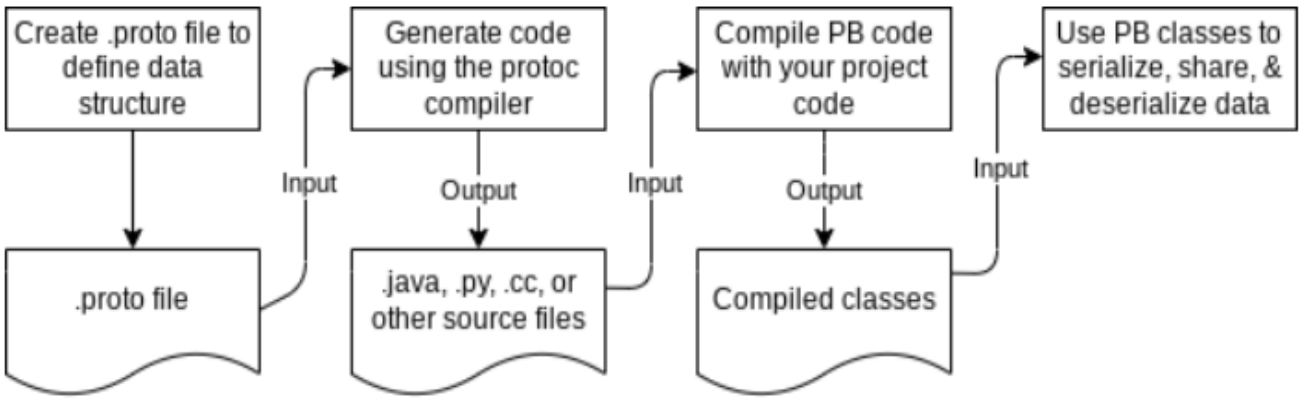



source: Introduction to gRPC

# Understanding APIs (Protocol Buffers)

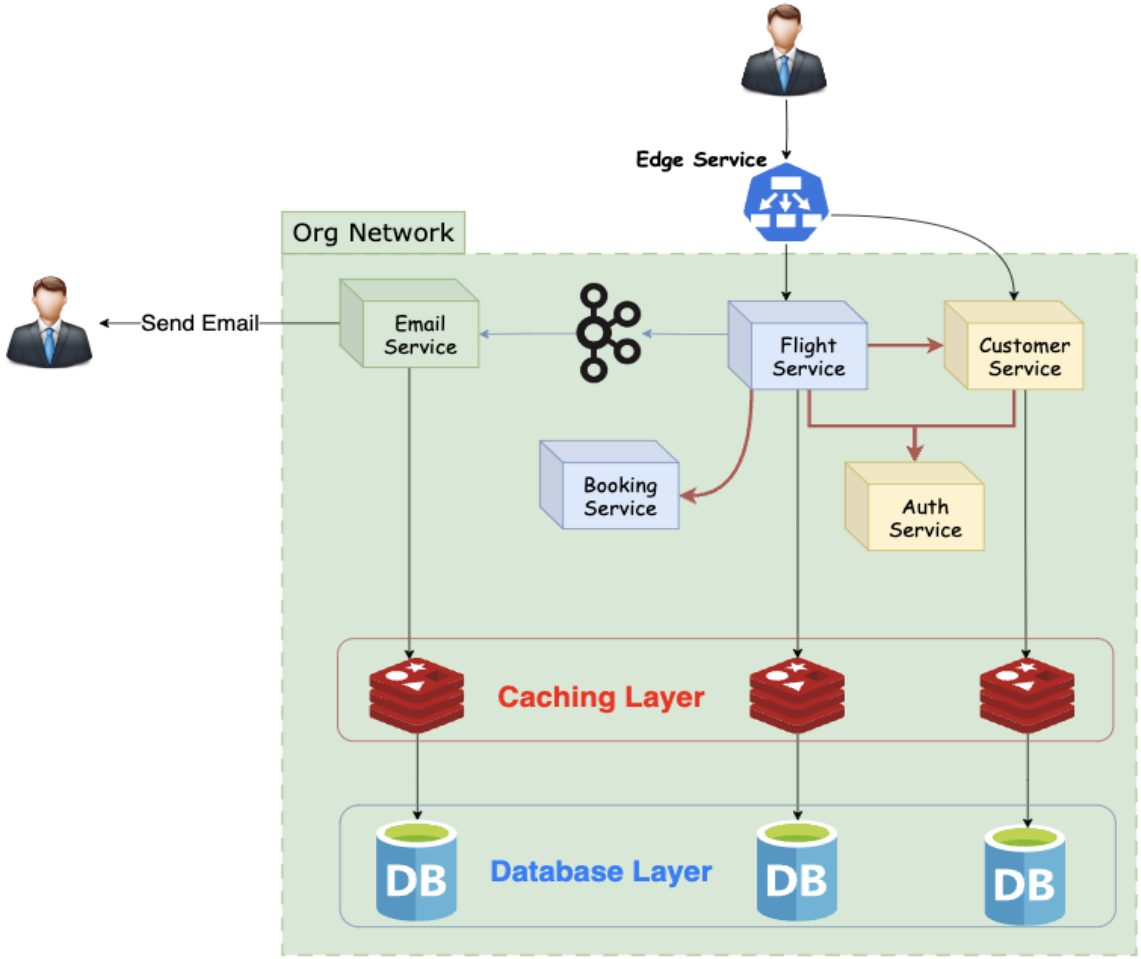
- Protocol Buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.
- The file that includes the definition is `.proto` file

```
message Person {
 string name = 1;
 int32 id = 2;
 bool has_kids = 3;
 optional string email = 4;
}
```




source: protobuf overview

# Understanding APIs (gRPC)




gRPC
Service to Service Integration

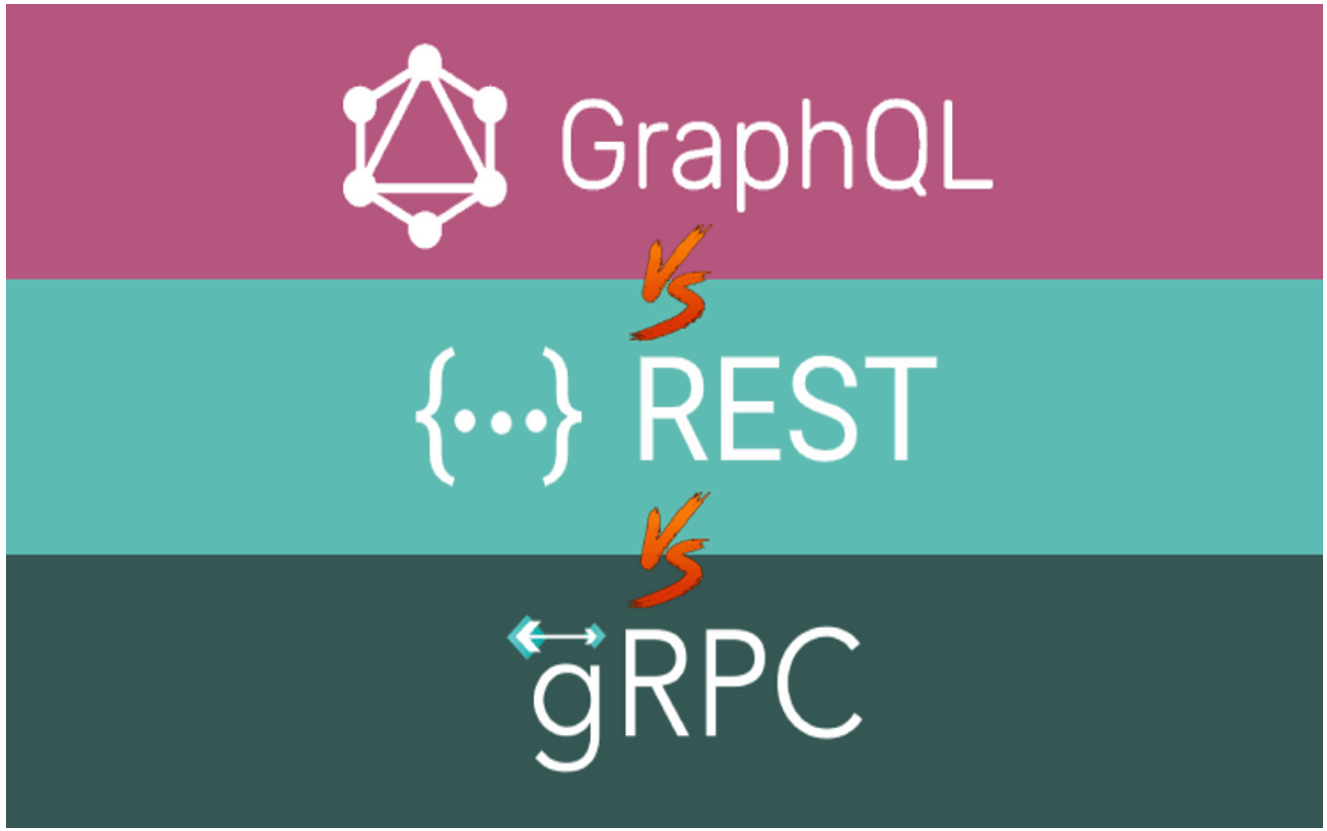
source: gRPC Offical doc

# Understanding APIs (gRPC)

```
1  syntax = "proto3";
2  package observer;
3
4  service ObserverStatusService {
5      rpc getServiceStatus (SystemStatusRequest)
6        returns (SystemStatusResponse) {}
7  }
8
9  message SystemStatusRequest {
10     optional string uuid = 1;
11     string service_name = 2;
12 }
13
14 message SystemStatusResponse {
15     string uuid = 1;
16     string status = 2;
17     string component = 3;
18     optional string service_name = 4;
19 }
```

# Understanding APIs (Other Protocols)

- gRPC
- REST
- GraphQL




source: REST vs GraphQL vs gRPC: Comparing Three Modern API Technologies

# Understanding APIs (RESTful API)

## REST API

- In 2000, Roy Fielding proposed Representational State Transfer (REST) as an architectural approach to designing web services
- REST is independent of any underlying protocol and is not necessarily tied to HTTP
- REST is an architectural style for building distributed systems based on *hypermedia*

## Primary Goal

REST API doesn't bind the implementation to any specific implementation, the API could be written in **Go** , and the client can use any language or tools.

## Example

To **GET** an order , API URI might look like https://api-design/orders/1

Client response `{"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}`

source: Microsoft: RESTful web API design

# Understanding APIs (RESTful API)

**It is all about *resources***

A resource is an entity that can be identified, named, addressed, or handled on the web.REST APIs expose data as resources and use standard HTTP methods to represent Create, Read, Update, and Delete (CRUD) transactions against these resources.

source: Designing Web APIs

# Understanding APIs (RESTful API)

**It is all about *resources***

A resource is an entity that can be identified, named, addressed, or handled on the web.REST APIs expose data as resources and use standard HTTP methods to represent Create, Read, Update, and Delete (CRUD) transactions against these resources.

- Resource is part of URL like, https://api-design/**orders**/1

source: Designing Web APIs

# Understanding APIs (RESTful API)

**It is all about *resources***

A resource is an entity that can be identified, named, addressed, or handled on the web.REST APIs expose data as resources and use standard HTTP methods to represent Create, Read, Update, and Delete (CRUD) transactions against these resources.

- Resource is part of URL like, https://api-design/**orders**/1
- Resource is always noun and plural, for each resource, two URLs are generally implemented one for collection, and one for specific element.
  like, **/users** and **/users/12**

source: Designing Web APIs

# Understanding APIs (RESTful API)

**It is all about *resources***

A resource is an entity that can be identified, named, addressed, or handled on the web.REST APIs expose data as resources and use standard HTTP methods to represent Create, Read, Update, and Delete (CRUD) transactions against these resources.

- Resource is part of URL like, https://api-design/**orders**/1
- Resource is always noun and plural, for each resource, two URLs are generally implemented one for collection, and one for specific element.
  like, **/users** and **/users/12**
- HTTP methods like GET, POST, UPDATE, and DELETE inform the server about the action to be performed

source: Designing Web APIs

# Understanding APIs (RESTful API)

**It is all about *resources***

A resource is an entity that can be identified, named, addressed, or handled on the web.REST APIs expose data as resources and use standard HTTP methods to represent Create, Read, Update, and Delete (CRUD) transactions against these resources.

- Resource is part of URL like, https://api-design/**orders**/1
- Resource is always noun and plural, for each resource, two URLs are generally implemented one for collection, and one for specific element.
  like, **/users** and **/users/12**
- HTTP methods like GET, POST, UPDATE, and DELETE inform the server about the action to be performed
- REST methods semantics "CRUD"
  - READ → GET, `never change the server state`
  - CREATE → POST
  - UPDATE → PUT or PATCH
  - DELETE → DELETE

source: Designing Web APIs

# Understanding APIs (RESTful API)

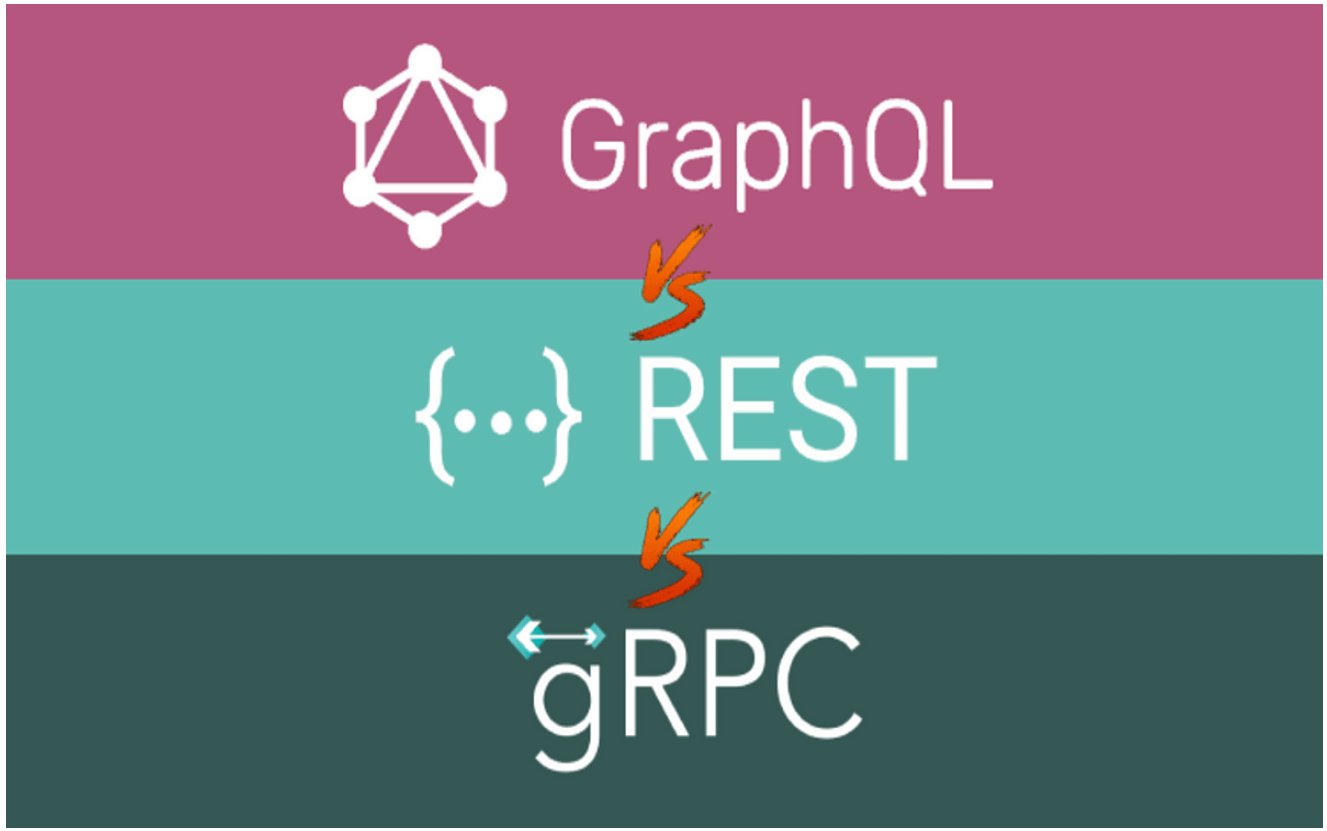
`CRUD operations, HTTP verbs, and REST conventions`

| Operation | HTTP verb | URL: /users | URL: /users/U123 |
|---|---|---|---|
| Create | POST | Create a new user | Not applicable |
| Read | GET | List all users | Retrieve user U123 |
| Update | PUT or PATCH | Batch update users | Update user U123 |
| Delete | DELETE | Delete all users | Delete user U123 |

source: Designing Web APIs

# Understanding APIs (Other Protocols)

- gRPC
- REST
- GraphQL




source: REST vs GraphQL vs gRPC: Comparing Three Modern API Technologies

# Understanding APIs (Intro to GraphQL)

## What is GraphQL?

- As shown in the name, it includes “Graph“ which means it represents a relation to some extend, you have to keep in your mind this subtle notice.

source: Introduction to GraphQL

# Understanding APIs (Intro to GraphQL)

## What is GraphQL?

- As shown in the name, it includes “Graph“ which means it represents a relation to some extend, you have to keep in your mind this subtle notice.
- GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data.

source: Introduction to GraphQL

# Understanding APIs (Intro to GraphQL)

## What is GraphQL?

- As shown in the name, it includes “Graph“ which means it represents a relation to some extend, you have to keep in your mind this subtle notice.
- GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data.
- GraphQL **isn’t tied** to any specific database or storage engine and is instead backed by your existing code and data.

source: Introduction to GraphQL

# Understanding APIs (Intro to GraphQL)

## What is GraphQL?

- As shown in the name, it includes “Graph“ which means it represents a relation to some extend, you have to keep in your mind this subtle notice.
- GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data.
- GraphQL **isn't tied** to any specific database or storage engine and is instead backed by your existing code and data.

## Example

A GraphQL service is created by defining types and fields on those types, then providing functions for each field on each type. For example, a GraphQL service that tells you who the logged in user is (me) as well as that user's name might look like this:

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

source: Introduction to GraphQL

# Understanding APIs (Intro to GraphQL "examples")

```
{
  hero {
    name
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
```

```
{
  human(id: "1000") {
    name
    height
  }
}
```

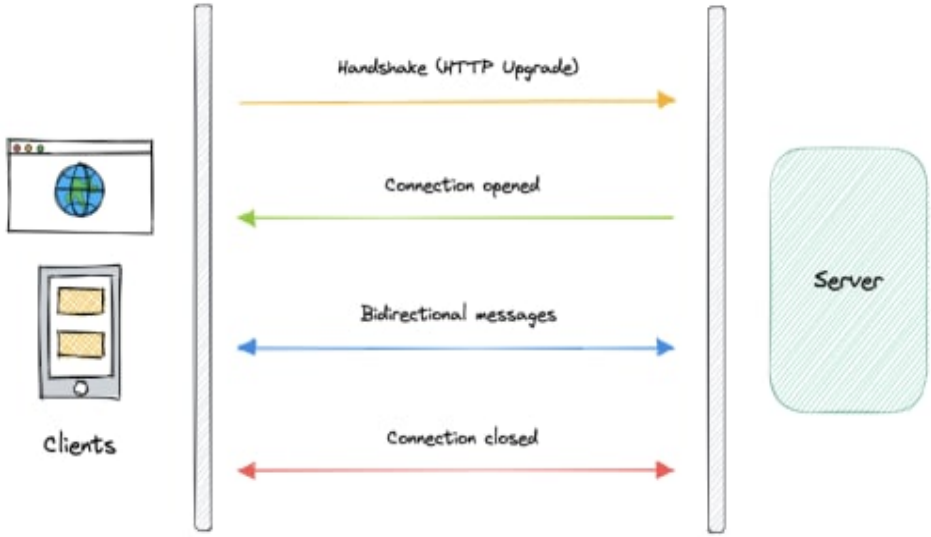
```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 1.72
    }
  }
}
```

source: Introduction to GraphQL

# Understanding APIs (Websockets API)

## What is Websocket API?

- The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server.

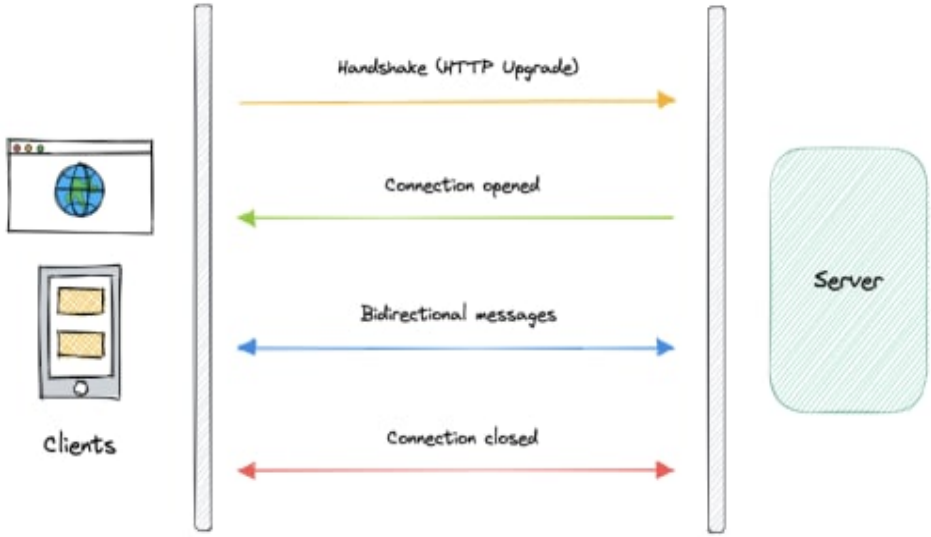



source: WebSockets, Long polling, Server-Sent Events

# Understanding APIs (Websockets API)

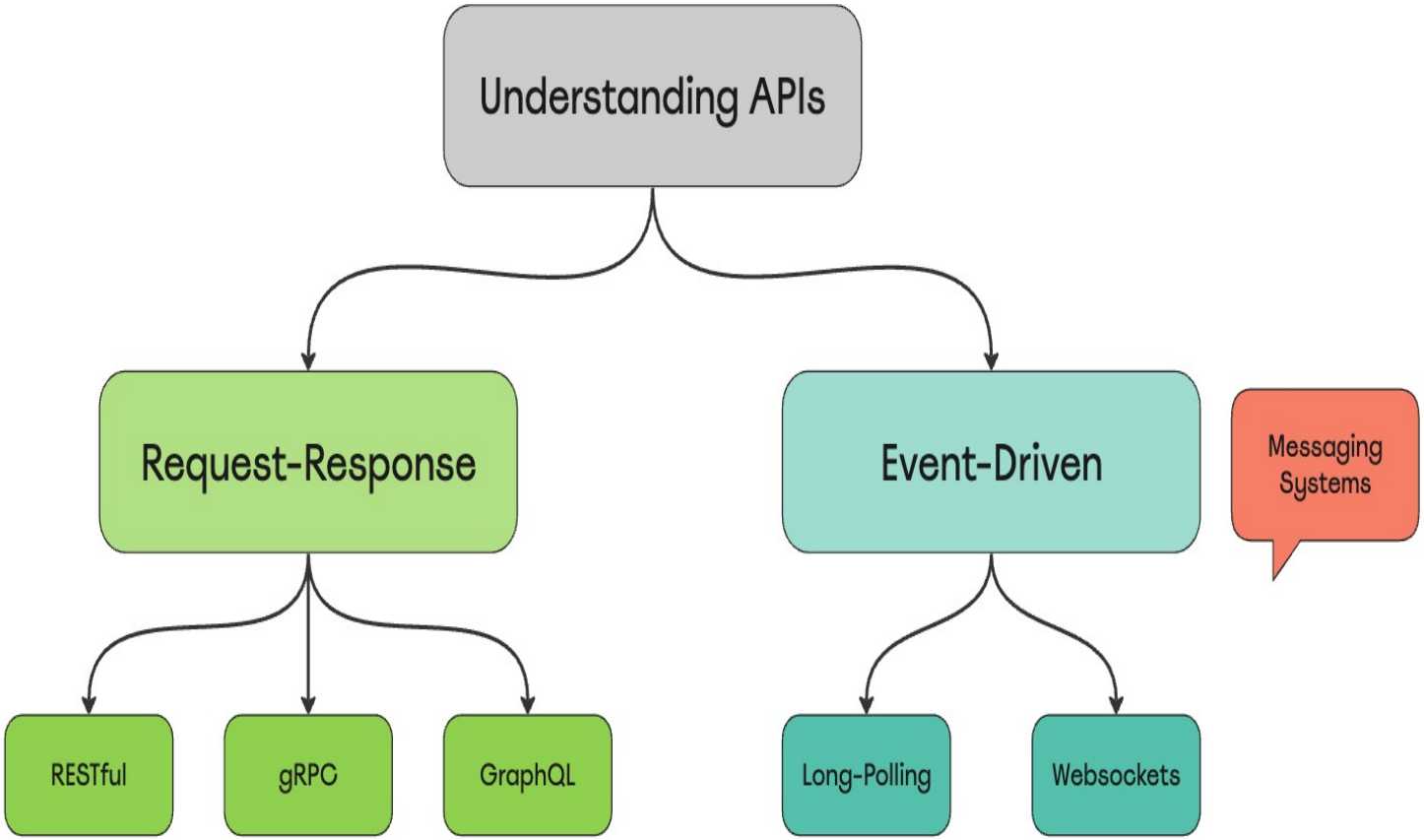
## What is Websocket API?

- The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server.
- With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.




source: WebSockets, Long polling, Server-Sent Events

# Understanding APIs




# API Design Principles (Know Your Consumer)

When designing an API, it's best to consider real-life use cases and think about the developers who will be using your API

- What tasks should they be able to complete with your API?

source: Designing Web APIs

# API Design Principles (Know Your Consumer)

When designing an API, it's best to consider real-life use cases and think about the developers who will be using your API

- What tasks should they be able to complete with your API?
- What kind of application should developer be able to build?

source: Designing Web APIs

# API Design Principles (Know Your Consumer)

When designing an API, it's best to consider real-life use cases and think about the developers who will be using your API

- What tasks should they be able to complete with your API?
- What kind of application should developer be able to build?
- How can you make the developer understand your API clearly?

source: Designing Web APIs

# API Design Principles (Know Your Consumer)

When designing an API, it's best to consider real-life use cases and think about the developers who will be using your API

- What tasks should they be able to complete with your API?
- What kind of application should developer be able to build?
- How can you make the developer understand your API clearly?

**Think Twice when Designing an API**

API you design is like a window that shows what you want outsiders to see.

source: Designing Web APIs

# API Design Principles (Know Your Consumer)

When designing an API, it's best to consider real-life use cases and think about the developers who will be using your API

- What tasks should they be able to complete with your API?
- What kind of application should developer be able to build?
- How can you make the developer understand your API clearly?

**Think Twice when Designing an API**

API you design is like a window that shows what you want outsiders to see.

**Expert Advice**

When we asked Ido Green, developer advocate at Google, what makes an API good, his top answer was *focus*:

> "The API should enable developers to do one thing really well. It's not as easy as it sounds, and you want to be clear on what the API is not going to do as well."

source: Designing Web APIs

# API Design Principles (Best Practices)

Think of your API as the UI you provide to someone, check the UX multiple times.

source: Designing Web APIs

# API Design Principles (Best Practices)

Think of your API as the UI you provide to someone, check the UX multiple times.

- Make it fast and Easy to Get Started

source: Designing Web APIs

# API Design Principles (Best Practices)

Think of your API as the UI you provide to someone, check the UX multiple times.

- Make it fast and Easy to Get Started
- Documentation that outline the specifications of an API can go a long way toward helping developers get started

source: Designing Web APIs

# API Design Principles (Best Practices)

Think of your API as the UI you provide to someone, check the UX multiple times.

- Make it fast and Easy to Get Started
- Documentation that outline the specifications of an API can go a long way toward helping developers get started
  - A tutorial is an interactive interface to teach developers about your API

source: Designing Web APIs

# API Design Principles (Best Practices)

Think of your API as the UI you provide to someone, check the UX multiple times.

- Make it fast and Easy to Get Started
- Documentation that outline the specifications of an API can go a long way toward helping developers get started
  - A tutorial is an interactive interface to teach developers about your API
  - A guide is a more contextual document than a specification. It provides information for developers at a certain point in time—typically when getting started

source: Designing Web APIs

# API Design Principles (Best Practices)

Think of your API as the UI you provide to someone, check the UX multiple times.

- Make it fast and Easy to Get Started
- Documentation that outline the specifications of an API can go a long way toward helping developers get started
  - A tutorial is an interactive interface to teach developers about your API
  - A guide is a more contextual document than a specification. It provides information for developers at a certain point in time—typically when getting started

**Expert Advice**

No matter how carefully we design and build our core API, developers continue to create products we'd never expect. We give them the freedom to build what they like.

Designing an API is much like designing a transportation network. Rather than prescribing an end state or destination, a good API expands the very notion of what's possible for developers.

—Romain Huet, head of developer relations at Stripe

source: Designing Web APIs

# API Design Principles (Request Packets)

HTTP Protocol was built on top of TCP protocol

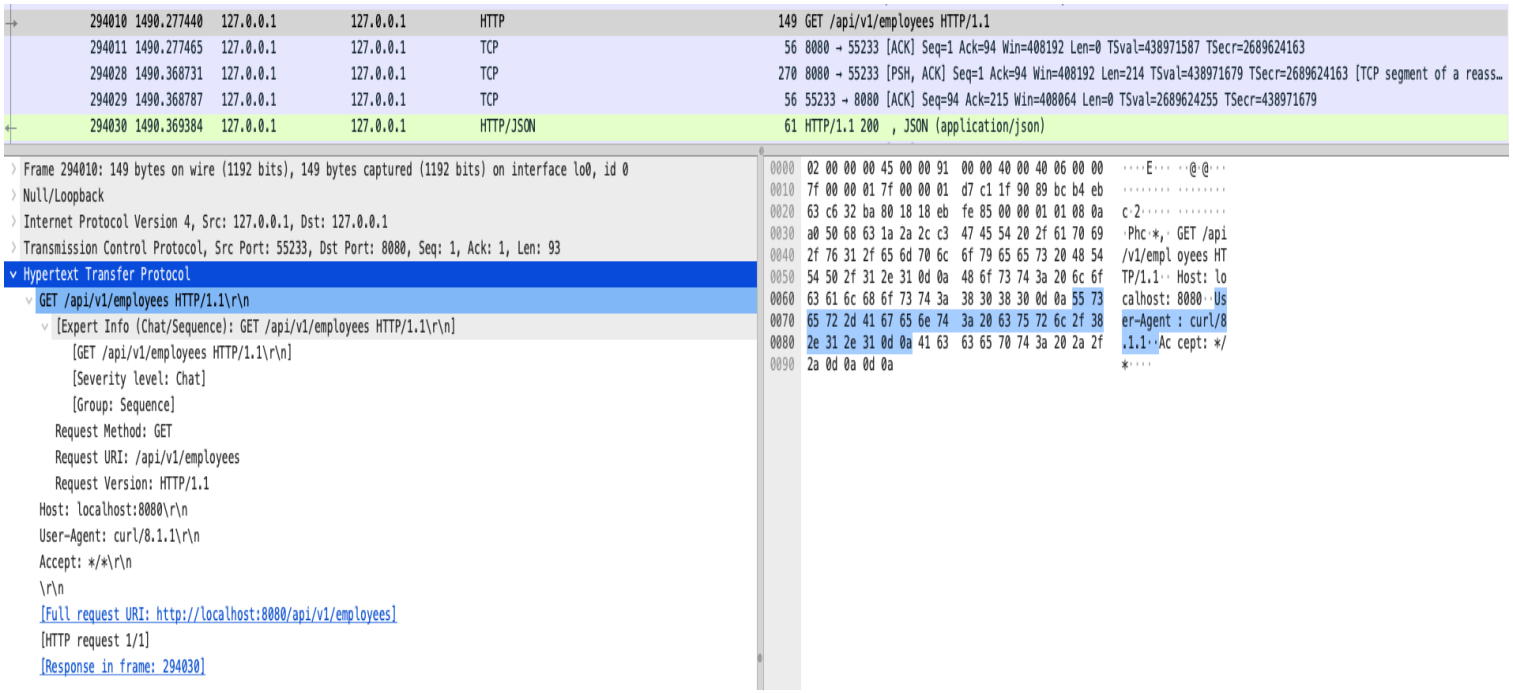
- HTTP Request is sent to server over TCP IP PACKET
- Server Reply with ACK , PSH, and provide Response in another IP PACKET
- CLIENT can follow up with ACK and close the connection

```
GET /api/v1/employees HTTP/1.1
Host: localhost:8080
User-Agent: curl/8.1.1
Accept: */*

HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 16 Mar 2024 23:37:09 GMT

[{"id":1,"name":"Mohamed Ahmed"},{"id":2,"name":"Alaa Mohamed"},{"id":3,"name":"Heba Masoud"}]
```

# API Design Principles (Request Packets)

HTTP Protocol was built on top of TCP protocol

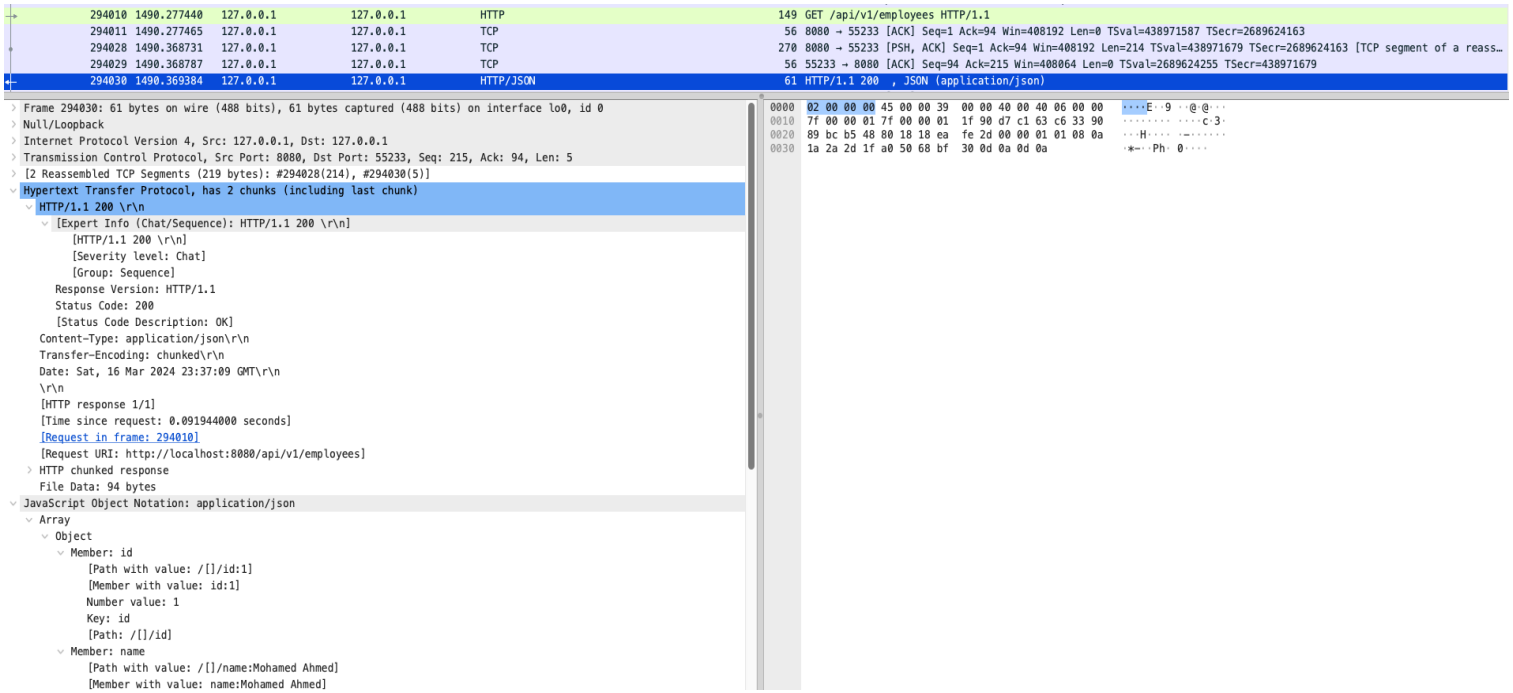
- HTTP Request is sent to server over TCP IP PACKET
- Server Reply with ACK , PSH, and provide Response in another IP PACKET
- CLIENT can follow up with ACK and close the connection




# API Design Principles (Request Packets)

HTTP Protocol was built on top of TCP protocol

- HTTP Request is sent to server over TCP IP PACKET
- Server Reply with ACK , PSH, and provide Response in another IP PACKET
- CLIENT can follow up with ACK and close the connection




# RESTful API Design

- RESTful Architecture Principles
- Designing RESTful Services (HTTP Methods, Status Codes)
- Best Practices in RESTful API

# RESTful Architecture Principles

In late 1993, Roy Fielding, co-founder of the Apache HTTP Server Project recognized that the Web's scalability was governed by a set of key constraints which were grouped under six categories:

- Client-server
- Uniform interface
- Layered system
- Stateless
- Code-on-demand 'Optional'
- Cache

# RESTful Architecture Principles

In late 1993, Roy Fielding, co-founder of the Apache HTTP Server Project recognized that the Web's scalability was governed by a set of key constraints which were grouped under six categories:

- Client-server
- Uniform interface
- Layered system

  The layered system constraints enable network-based intermediaries such as proxies and gateways to be transparently deployed between a client and server using the Web's uniform interface.
- Stateless
- Code-on-demand 'Optional'
- Cache

# RESTful Architecture Principles

In late 1993, Roy Fielding, co-founder of the Apache HTTP Server Project recognized that the Web's scalability was governed by a set of key constraints which were grouped under six categories:

- Client-server
- Uniform interface
- Layered system

  The layered system constraints enable network-based intermediaries such as proxies and gateways to be transparently deployed between a client and server using the Web's uniform interface.

- Stateless

  The stateless constraint dictates that a web server is not required to memorize the state of its client applications. As a result, each client must include all of the contextual information that it considers relevant in each interaction with the web server.

- Code-on-demand 'Optional'
- Cache

# RESTful Architecture Principles

In late 1993, Roy Fielding, co-founder of the Apache HTTP Server Project recognized that the Web's scalability was governed by a set of key constraints which were grouped under six categories:

- Client-server
- Uniform interface
- Layered system

  The layered system constraints enable network-based intermediaries such as proxies and gateways to be transparently deployed between a client and server using the Web's uniform interface.
- Stateless

  The stateless constraint dictates that a web server is not required to memorize the state of its client applications. As a result, each client must include all of the contextual information that it considers relevant in each interaction with the web server.
- Code-on-demand 'Optional'

  Code-on-demand enables web servers to temporarily transfer executable programs, such as scripts or plug-ins, to clients.
- Cache

# Restful API Design Semantics

HTTP semantics include the intentions defined by each request method

## HTTP Core Semantics

- In request header fields, status codes that describe the response
- **Representation** metadata describe how content is intended to be interpreted by a recipient
- Request header fields that might influence content selection, and the various selection algorithms that are collectively referred to as "content negotiation"

## Representation Definition

A "representation" is information that is intended to reflect a past, current, or desired state of a given resource. A representation consists of a set of representation metadata and a potentially unbounded stream of representation data.

```
➜  ~ curl --location 'http://universities.hipolabs.com/search?name=middle' -i
HTTP/1.1 200 OK
Server: nginx/1.14.0
Date: Tue, 09 Apr 2024 12:45:37 GMT
Content-Type: application/json
Content-Length: 1873
Connection: keep-alive
Access-Control-Allow-Origin: *
```

source: RFC 9110

# Restful API Design Semantics "Content Type"

- Content-Type: The indicated media type defines both the data format and how that data is intended to be processed by a recipient.
  [Format]
  Content-Type: [media type];[subtype]

```
Content-Type:  text/html ; Charset="utf-8"
```

**Examples**

| | |
|---|---|
| application/json | image/png |
| application/x-www-form-urlencoded | multipart/form-data |

- Understanding the Media type helps you handling the request correctly with minimal processing

```
➜  ~ curl -X POST http://localhost:3000/api/orders -i
HTTP/1.1 415 Unsupported Media Type
X-Powered-By: Express
Content-Type: text/plain; charset=utf-8
Content-Length: 22
ETag: W/"16-WZ8/jdYUt/AP1zhoOBkR1qtJ4vE"
Date: Wed, 10 Apr 2024 19:52:39 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

source: RFC 9110

# Restful API Design "HTTP Methods"

| Method Name | Description | Section |
|---|---|---|
| GET | Transfer a current representation of the target resource. | 9.3.1 |
| HEAD | Same as GET, but do not transfer the response content. | 9.3.2 |
| POST | Perform resource-specific processing on the request content. | 9.3.3 |
| PUT | Replace all current representations of the target resource with the request content. | 9.3.4 |
| DELETE | Remove all current representations of the target resource. | 9.3.5 |
| CONNECT | Establish a tunnel to the server identified by the target resource. | 9.3.6 |
| OPTIONS | Describe the communication options for the target resource. | 9.3.7 |
| TRACE | Perform a message loop-back test along the path to the target resource. | 9.3.8 |

source: RFC 9110

# Restful API Design Semantics "Common Properties"

## Safe Methods

An HTTP method is safe if a request using this method doesn't alter the state of the server. In other words, it leads to a read-only operation.

- Safe methods: `GET, HEAD, OPTIONS`
- Unsafe methods: `POST, PUT, DELETE, CONNECT, PATCH`

source: Common Properties

# Restful API Design Semantics "Common Properties"

## Safe Methods

An HTTP method is safe if a request using this method doesn't alter the state of the server. In other words, it leads to a read-only operation.

## Idempotent Methods

An HTTP method is idempotent if multiple identical requests using this method will have the same effect on the server as that of a single request of that same method.

- Idempotent methods: `GET, HEAD, PUT, DELETE, OPTIONS, TRACE`
- Non-idempotent methods: `POST, CONNECT, PATCH`

source: Common Properties

# Restful API Design Semantics "Common Properties"

## Safe Methods

An HTTP method is safe if a request using this method doesn't alter the state of the server. In other words, it leads to a read-only operation.

## Idempotent Methods

An HTTP method is idempotent if multiple identical requests using this method will have the same effect on the server as that of a single request of that same method.

## Method Caching

An HTTP method is cacheable if the response to a request using this method is allowed to be stored for future use.

- In general safe methods are defined as cacheable like `GET` and `HEAD`.
- Non-cacheable methods: `POST, PUT, DELETE, CONNECT, OPTIONS, PATCH`.

source: Common Properties

# Hypermedia

User uses APIs according to his understanding with the help of communications support, calling APIs using *resources*. Eventually the user will need to follow some more resources' calls. You might be think of:

- How can the user/client know which request he needs to make next?

source: Restful Web APIs

# Hypermedia

User uses APIs according to his understanding with the help of communications support, calling APIs using *resources*. Eventually the user will need to follow some more resources' calls. You might be think of:

- How can the user/client know which request he needs to make next?
- How the request should look like?

source: Restful Web APIs

# Hypermedia

User uses APIs according to his understanding with the help of communications support, calling APIs using *resources*. Eventually the user will need to follow some more resources' calls. You might be think of:

- How can the user/client know which request he needs to make next?
- How the request should look like?
- Which types of error should the client expect?

source: Restful Web APIs

# Hypermedia

User uses APIs according to his understanding with the help of communications support, calling APIs using *resources*. Eventually the user will need to follow some more resources' calls. You might be think of:

- How can the user/client know which request he needs to make next?
- How the request should look like?
- Which types of error should the client expect?

The answer to this is *"hpermedia"*

source: Restful Web APIs

# Hypermedia

User uses APIs according to his understanding with the help of communications support, calling APIs using *resources*. Eventually the user will need to follow some more resources' calls. You might be think of:

- How can the user/client know which request he needs to make next?
- How the request should look like?
- Which types of error should the client expect?

The answer to this is *"hpermedia"*

**Hypermedia**

Hypermedia is a way for the server to tell the client what HTTP requests the client might want to make in the future. It's a menu, provided by the server, from which the client is free to choose. The server knows what might happen, but the client decides what actually happens.

source: Restful Web APIs

# Hypermedia

- Application Navigation Control

```
<a href="http://www.tyi.com/messages/"> See the latest messages </a>
```

source: Restful Web APIs

# Hypermedia

- Application Navigation Control

```
<a href="http://www.tyi.com/messages/"> See the latest messages </a>
```

- Application Embdedded Control

```
<img rel="icon" src="http://www.example.com/logo.png" />
```

source: Restful Web APIs

# Hypermedia

- Application Navigation Control

```
<a href="http://www.tyi.com/messages/"> See the latest messages </a>
```

- Application Embdedded Control

```
<img rel="icon" src="http://www.example.com/logo.png" />
```

- Application Complex Control

```
<form action="http://www.youtypeitwepostit.com/messages" method="post">
 <input type="text" name="message" value="" required="true" />
</form>
```

source: Restful Web APIs

# Hypermedia

- Application Navigation Control

```
<a href="http://www.tyi.com/messages/"> See the latest messages </a>
```

- Application Embdedded Control

```
<img rel="icon" src="http://www.example.com/logo.png" />
```

- Application Complex Control

```
<form action="http://www.youtypeitwepostit.com/messages" method="post">
 <input type="text" name="message" value="" required="true" />
</form>
```

**Hypermedia Formal Definition**

Hypermedia is defined by the presence of application control information embedded within, or as a layer above, the presentation of information.

source: Restful Web APIs

# API Documentation and Specification

- Importance of Comprehensive API Documentation
- Tools for API Documentation (Swagger, OpenAPI Specification)
- Maintaining and Versioning API Documentation

# API Documentations

- API Docs is like the manual of using something, instructions to follow "remember IKEA"

# API Documentations

- API Docs is like the manual of using something, instructions to follow "remember IKEA"
- Consumers/Users always prefer agreements to feel safe

# API Documentations

- API Docs is like the manual of using something, instructions to follow "remember IKEA"
- Consumers/Users always prefer agreements to feel safe
- API docs explain and answer consumers' questions, why you'd use this API? with examples

# API Documentations

- API Docs is like the manual of using something, instructions to follow "remember IKEA"
- Consumers/Users always prefer agreements to feel safe
- API docs explain and answer consumers' questions, why you'd use this API? with examples
- API docs should mention carefully the restrictions like Authentication, Rate Limiting, and Terms of use

# API Documentations

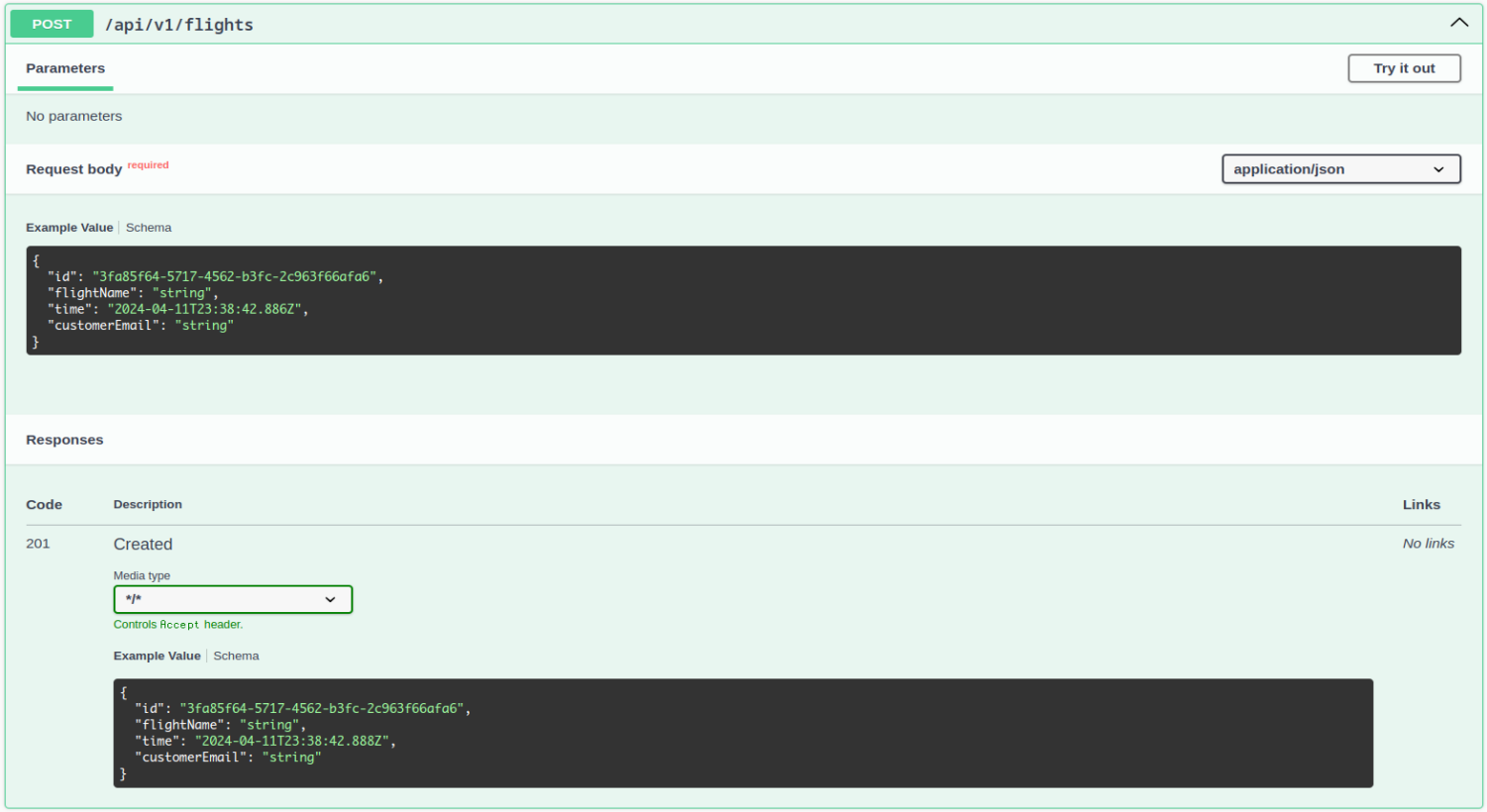
- API Docs is like the manual of using something, instructions to follow "remember IKEA"
- Consumers/Users always prefer agreements to feel safe
- API docs explain and answer consumers' questions, why you'd use this API? with examples
- API docs should mention carefully the restrictions like Authentication, Rate Limiting, and Terms of use
- Consumers love to try, they always look for *"Getting Started"*; Interactive API platforms are what they will love more.

# API Documentations

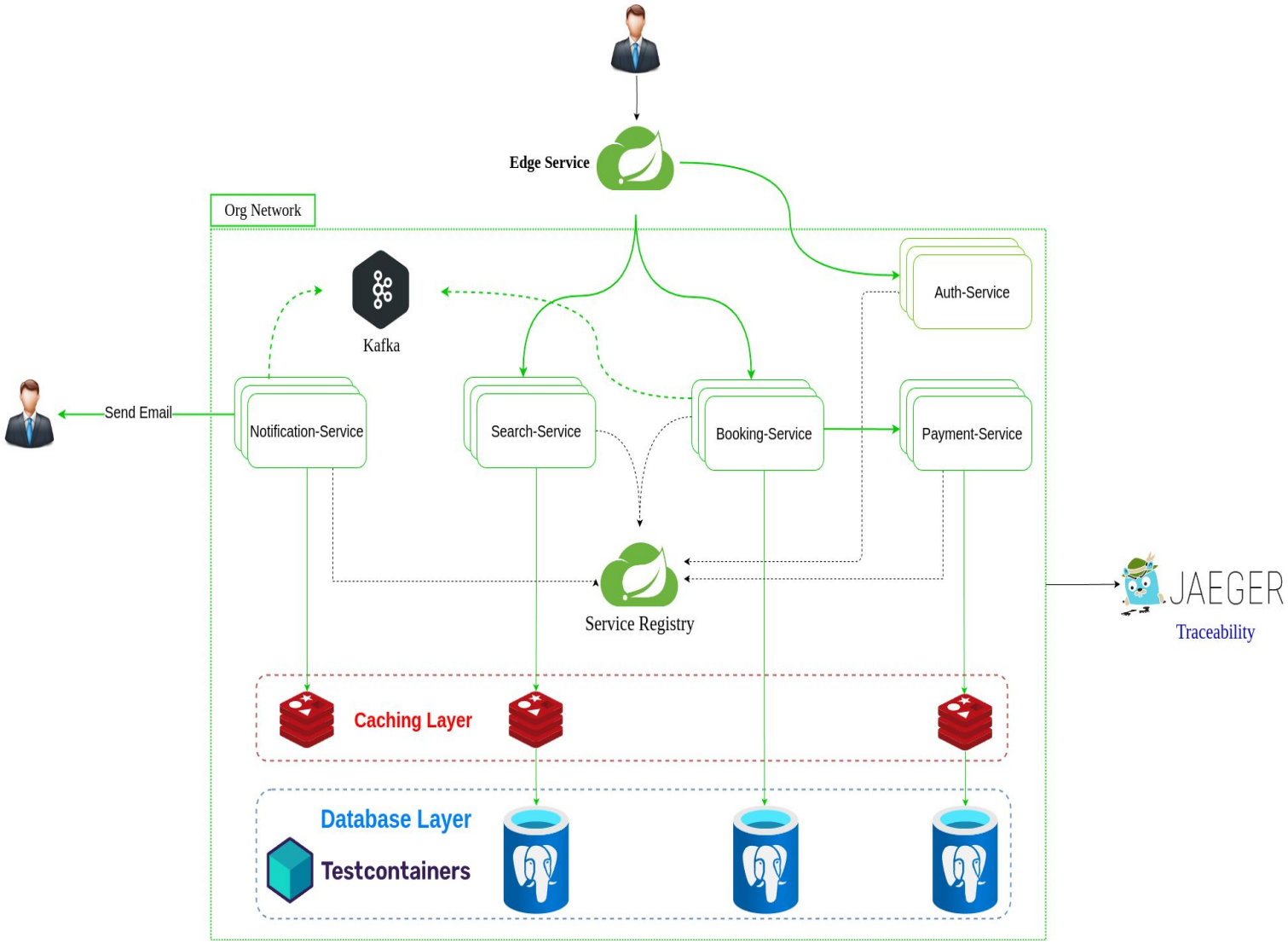
- API Docs is like the manual of using something, instructions to follow "remember IKEA"
- Consumers/Users always prefer agreements to feel safe
- API docs explain and answer consumers' questions, why you'd use this API? with examples
- API docs should mention carefully the restrictions like Authentication, Rate Limiting, and Terms of use
- Consumers love to try, they always look for *"Getting Started"*; Interactive API platforms are what they will love more.
- API Docs need to be up to date to avoid any service usage interruptions.

# Restful API Design "Communications"

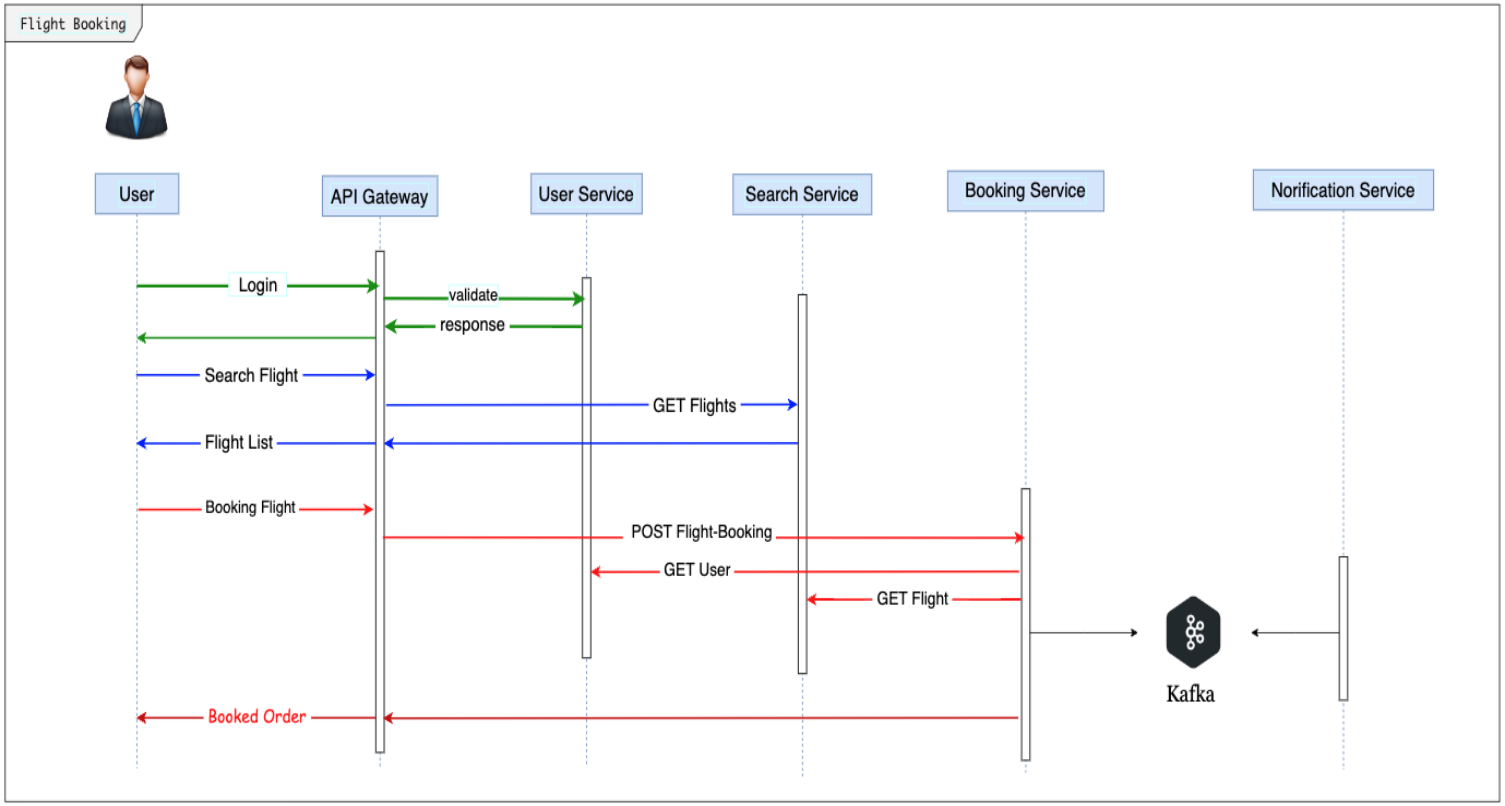
How to communicate the specifications to the client?




# Flight System

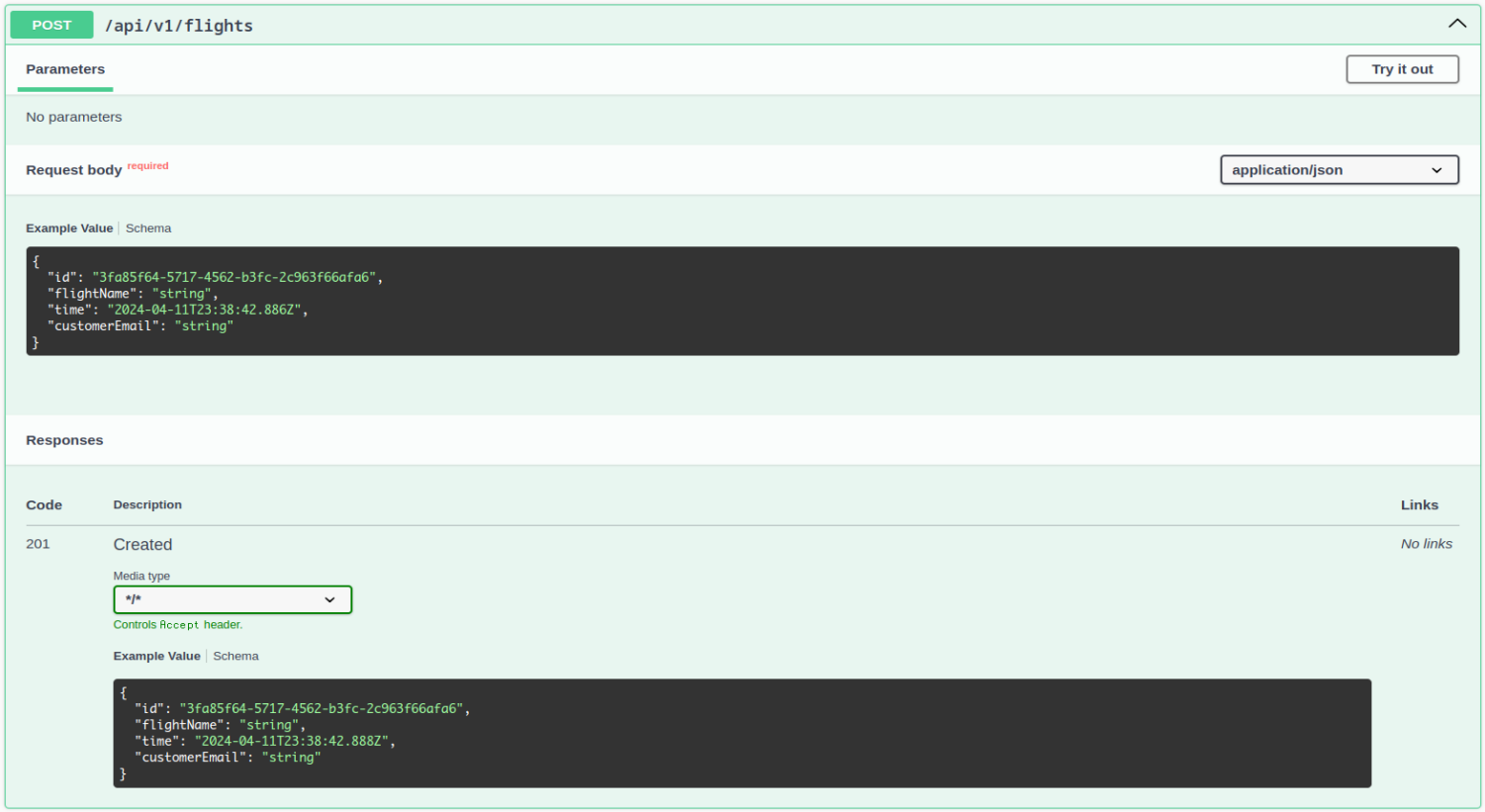


# Flight Booking Sequence Diagram




# Restful API Design "Communications"

How to communicate the specifications to the client?




# API Security

- Authentication and Authorization Mechanisms (OAuth, JWT)
- Securing API Endpoints
- Handling Sensitive Data and Privacy Concerns

# Authentication vs Authorization

- Authentication: Who is using the system? Is it human or machine?
- Authorization: What can user do inside the system? *Roles & Permissions*

# Authentication vs Authorization

- Authentication: Who is using the system? Is it human or machine?
- Authorization: What can user do inside the system? *Roles & Permissions*

## Sensitive Data

- Data is an asset, and you should keep an extreme focus on it. Differentiate between normal and sensitive data.
- It's not just about passwords or credit card numbers; Personally Identifiable Information (PII) is also sensitive information:
  Email, Biometrics like fingerprints, Driver's license number, Social Security number (SSN)
- System Configurations may also contain sensitive information that need to be secured.

# Authentication vs Authorization

- Authentication: Who is using the system? Is it human or machine?
- Authorization: What can user do inside the system? *Roles & Permissions*

## Sensitive Data

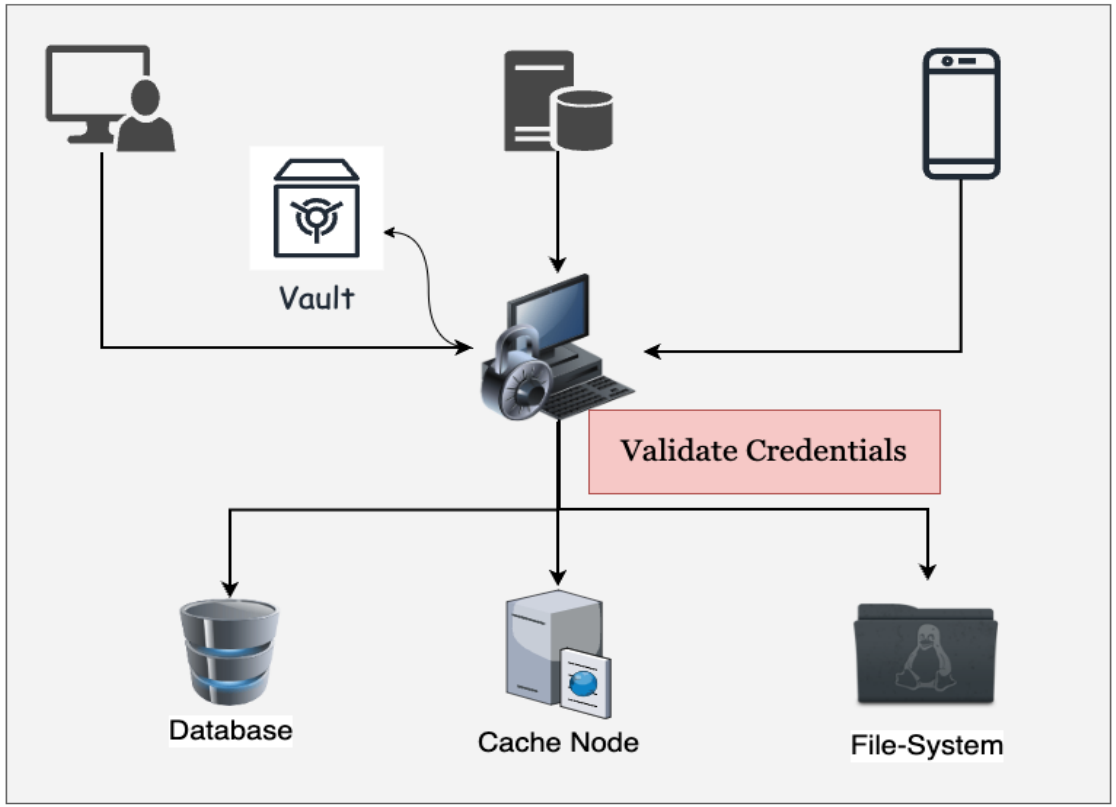
- Data is an asset, and you should keep an extreme focus on it. Differentiate between normal and sensitive data.
- It's not just about passwords or credit card numbers; Personally Identifiable Information (PII) is also sensitive information:
  Email, Biometrics like fingerprints, Driver's license number, Social Security number (SSN)
- System Configurations may also contain sensitive information that need to be secured.

Some common techniques to secure your information

- Encryption: a two-way function where information is scrambled in such a way that it can be unscrambled later
- Hashing: a one-way function where data is mapped to a fixed-length value.

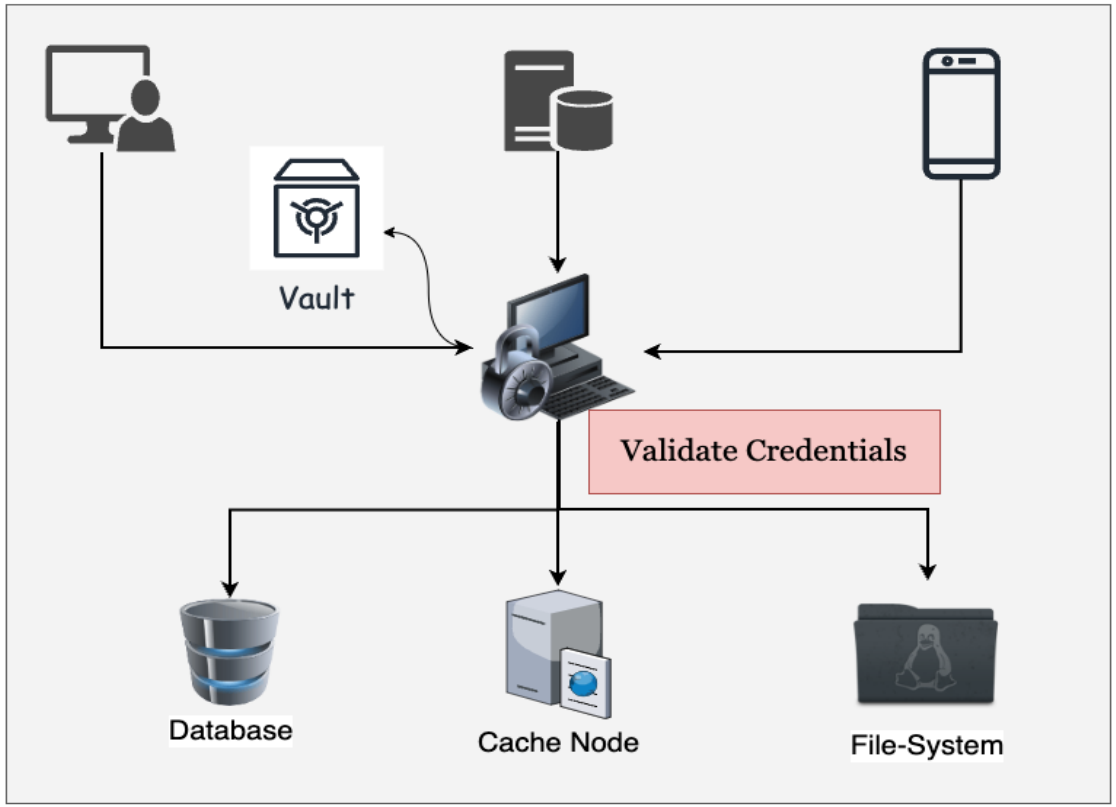
# Authentication

- Validating the user identity using credentials in **data stores** such as database, cache-node, or even file-system



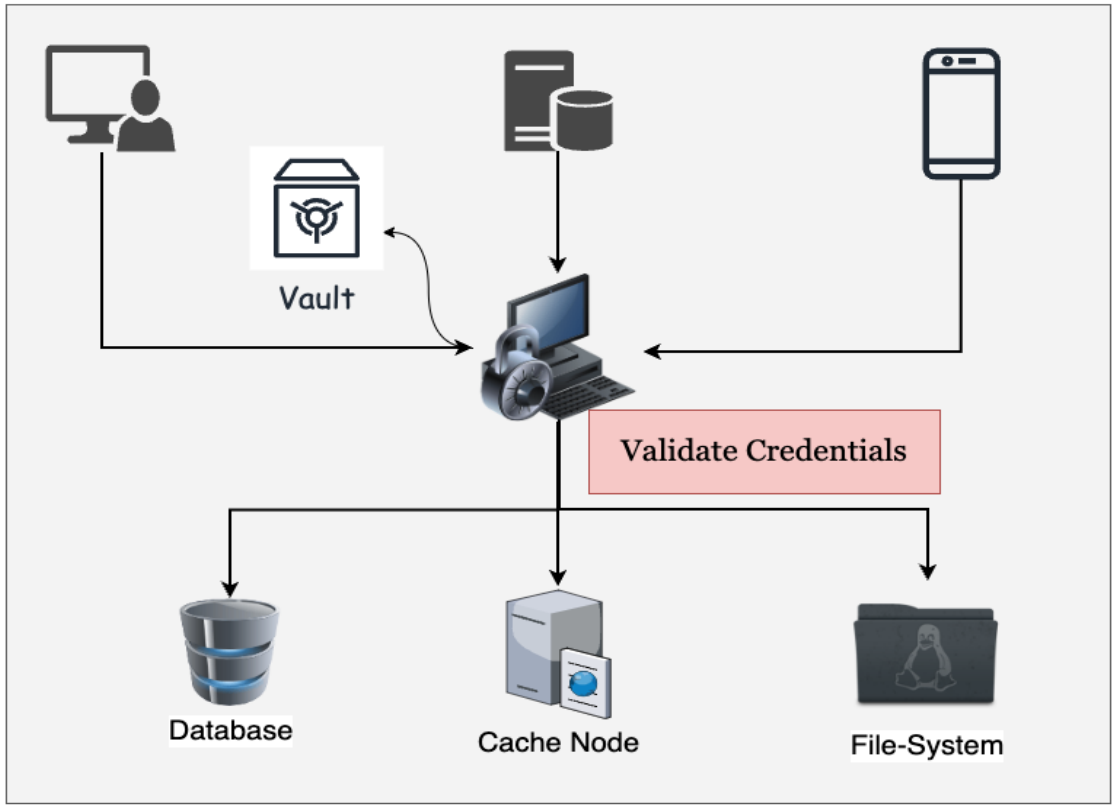
# Authentication

- Validating the user identity using credentials in **data stores** such as database, cache-node, or even file-system
- The most simple form of user credentials is username & password



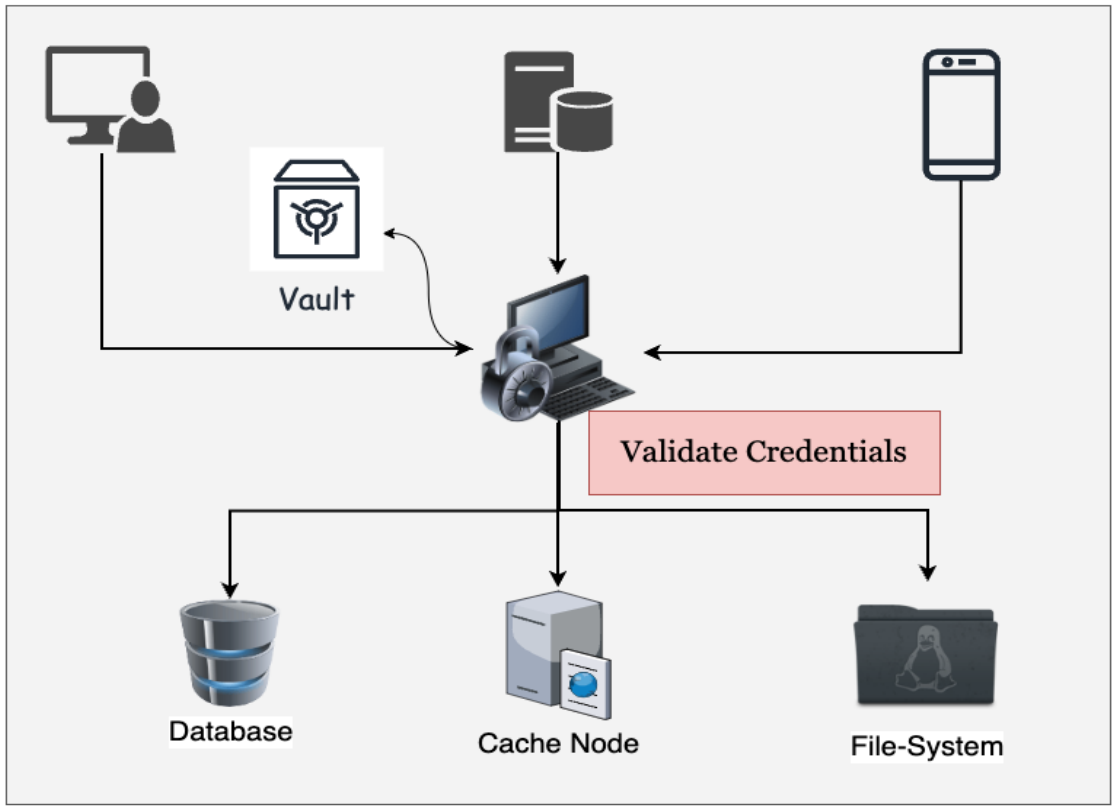
# Authentication

- Validating the user identity using credentials in **data stores** such as database, cache-node, or even file-system
- The most simple form of user credentials is username & password
- User or System can send those credentials in request-header for example , but this is still not secure!




# Authentication

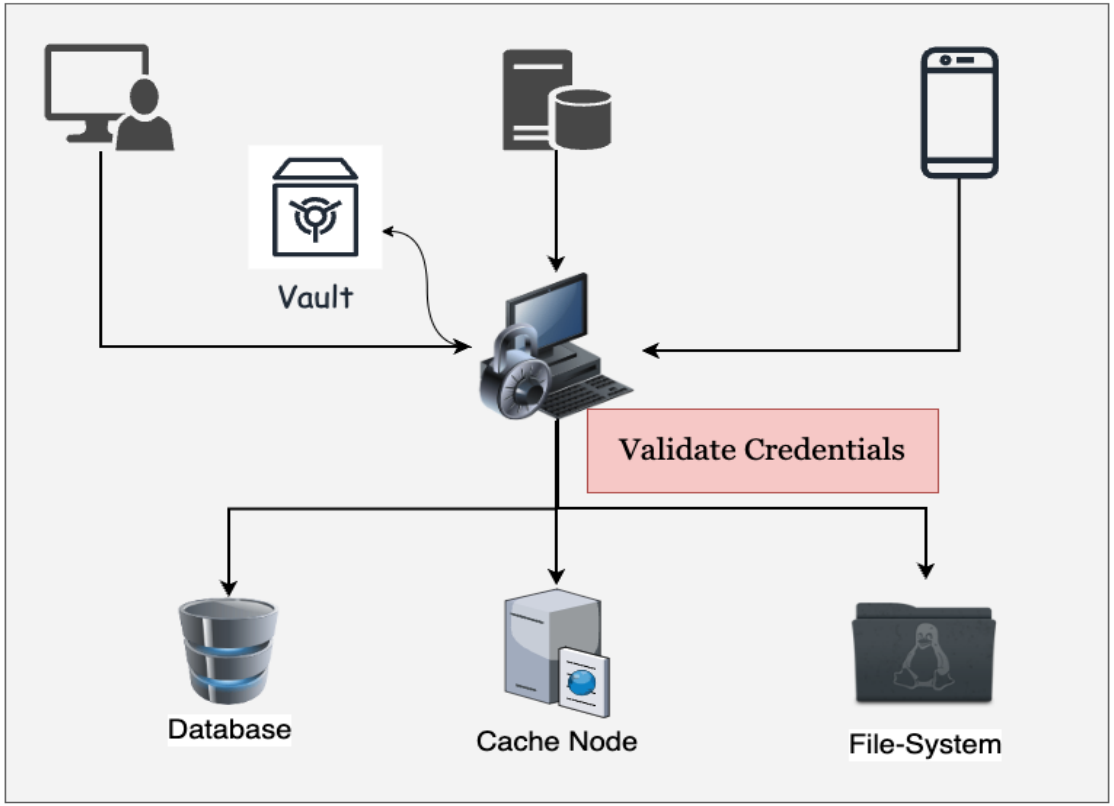
- Validating the user identity using credentials in **data stores** such as database, cache-node, or even file-system
- The most simple form of user credentials is username & password
- User or System can send those credentials in request-header for example , but this is still not secure!
- Another simple way is to use `basic-auth` which is `Basic username:password` encoded with base64



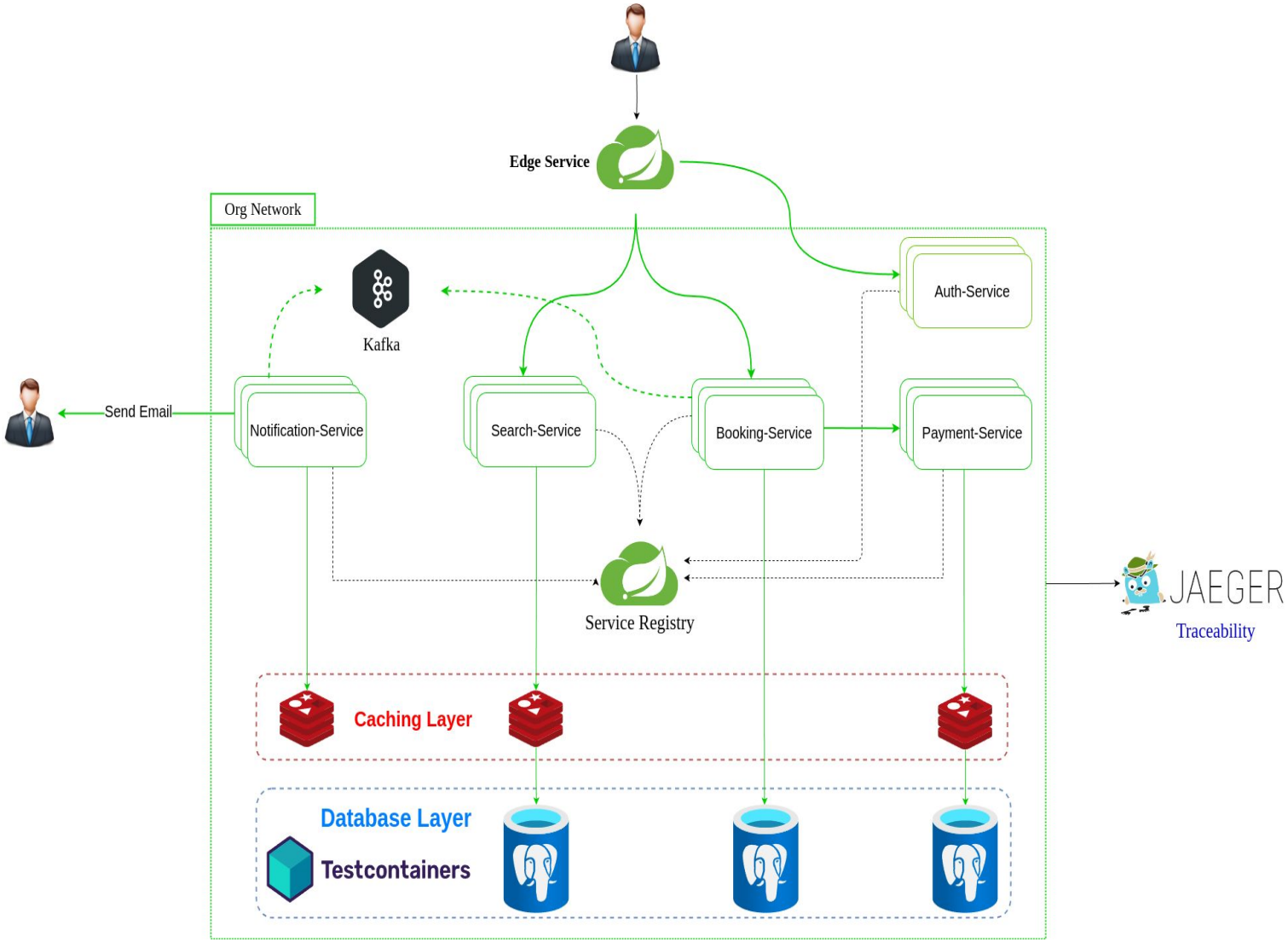

# Authentication

**Issues with this flow**

- Clients should keep Credentials in a secure place, it might be stolen! think of users who are using the same credentials across multiple systems
- Data Store will be overloaded
- Changing such information requires changing all places of client applications
- Not scalable!




# Flight System




# API Testing and Quality Assurance

- Writing Effective API Tests
- Tools and Frameworks for API Testing
- Performance Testing and Load Testing for APIs

# API Management and Lifecycle

- The Lifecycle of API Development
- API Deployment Strategies
- Monitoring and Analytics for APIs

# Conclusion

- Recap of Key Learnings
- Emerging Trends in API Development