

# API Design and Management

Mohamed Sweelam

Software Engineer



# Outline

- 1 Course Objectives
- 2 Understanding APIs
- 3 API Design Principles
- 4 RESTful API Design
- 5 Advanced API Protocols
- 6 API Documentation and Specification
- 7 API Security
- 8 API Testing and Quality Assurance
- 9 API Management and Lifecycle
- 10 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

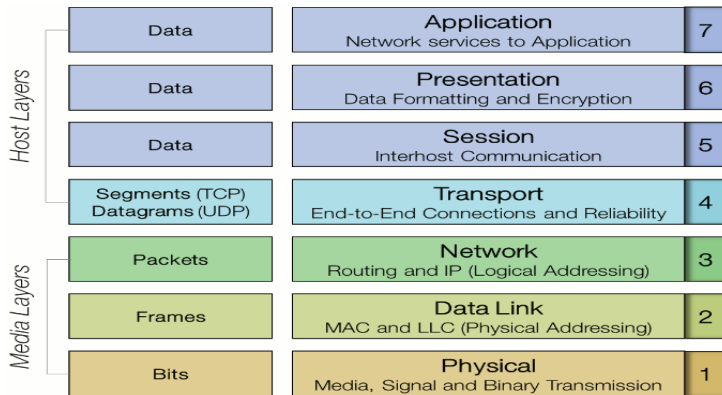
API (Application Programming Interface) is a set of rules, protocols, and tools for building software applications. It specifies how software components should interact and is used to enable the integration between different software systems.

## 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.

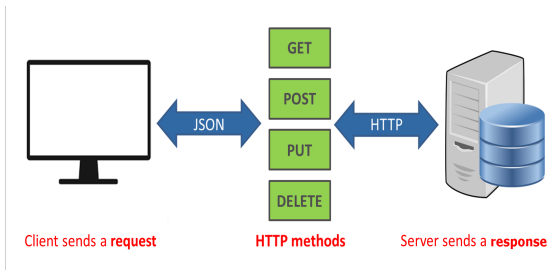


source: [coengodegebure.com/osi-model](https://coengodegebure.com/osi-model)





# Understanding APIs Example



source: [phpenthusiast.com/blog/what-is-rest-api](https://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.

```
1  <?xml version="1.0"?>
2  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
3    <soap:Header>
4      <!-- header information here -->
5    </soap:Header>
6    <soap:Body>
7      <m:GetPrice xmlns:m="http://www.example.org/stock">
8        <m:StockName>IBM</m:StockName>
9      </m:GetPrice>
10   </soap:Body>
11   <soap:Fault>
12     <!-- fault information here -->
13   </soap:Fault>
14 </soap:Envelope>
```



# Understanding APIs (Other Protocols)

- gRPC
- REST
- GraphQL

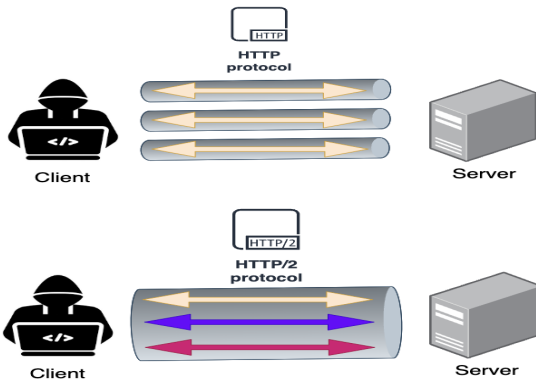


source: [REST vs GraphQL vs gRPC: Comparing Three Modern API Technologies](#)



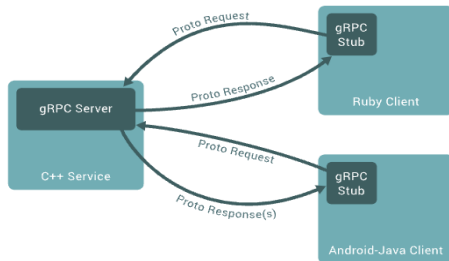
# Understanding APIs (HTTP1 vs HTTP2)

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.



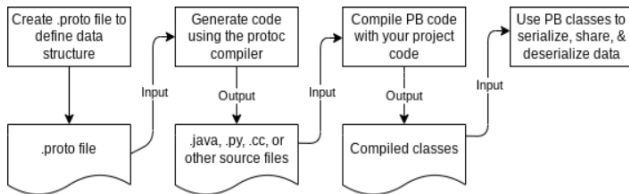
source: [Introduction to gRPC](#)



# 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

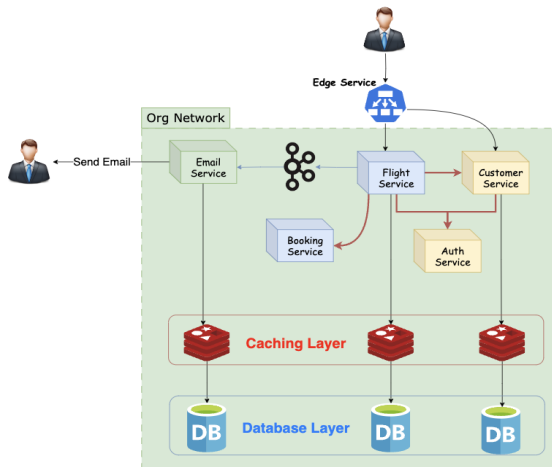
```
1  message Person {  
2    string name = 1;  
3    int32 id = 2;  
4    bool has_kids = 3;  
5    optional string email = 4;  
6  }
```



source: [protobuf overview](#)



# Understanding APIs (gRPC)



source: [gRPC Official 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”)

<pre>{   hero {     name   } }</pre>	<pre>{   "data": {     "hero": {       "name": "R2-D2"     }   } }</pre>
--------------------------------------	--

<pre>{   human(id: "1000") {     name     height   } }</pre>	<pre>{   "data": {     "human": {       "name": "Luke Skywalker",       "height": 1.72     }   } }</pre>
--	--

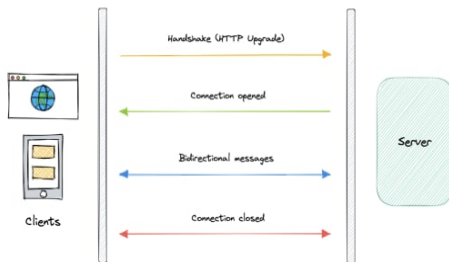
source: [Introduction to GraphQL](#)



# 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](#)

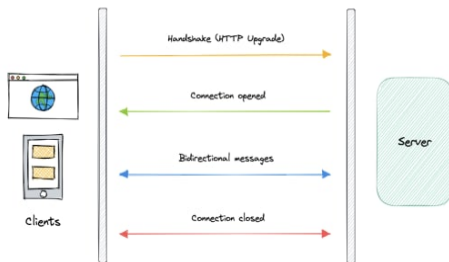




# Understanding APIs (Websockets API)

## What is WebSocket API?

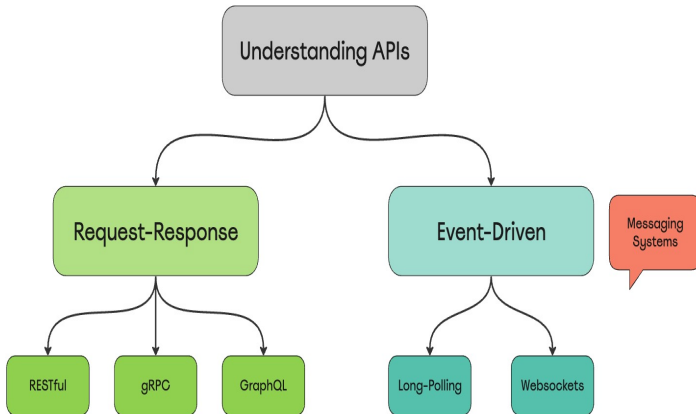
- The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server.
- With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.



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

294010	3490,277440	127.0.0.1	127.0.0.1	HTTP	349 GET /api/v1/employees HTTP/1.1
294011	3490,277465	127.0.0.1	127.0.0.1	TCP	56 0800 → 55233 [ACK] Seq=1 Ack=94 Win=408192 Len=0 TSval=438971587 TSecr=2689624163
294020	3490,368731	127.0.0.1	127.0.0.1	TCP	276 0800 → 55233 [PSH, ACK] Seq=1 Ack=94 Win=408192 Len=214 TSval=438971679 TSecr=2689624163 [TCP segment of a reas..
294029	3490,368787	127.0.0.1	127.0.0.1	TCP	56 55233 → 0800 [ACK] Seq=94 Ack=215 Win=408064 Len=0 TSval=268962455 TSecr=438971679
294030	3490,369394	127.0.0.1	127.0.0.1	HTTP/JSON	61 HTTP/1.1 200 , JSON (appLocation/json)

<pre> &gt; Frame 294030: 149 bytes on wire (1192 bits), 149 bytes captured (1192 bits) on interface lo0, id 0 &gt; Null/Loopback &gt; Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 &gt; Transmission Control Protocol, Src Port: 55233, Dst Port: 8080, Seq: 1, Ack: 1, Len: 93 v Hypertext Transfer Protocol   GET /api/v1/employees HTTP/1.1/r/n     [Expert Info (Chat/Sequence): GET /api/v1/employees HTTP/1.1/r/n]       [GET /api/v1/employees HTTP/1.1/r/n]       [Severity Level: Chat]       [Group: Sequence]       Request Method: GET       Request URI: /api/v1/employees       Request Version: HTTP/1.1       Host: localhost:8080/r/n       User-Agent: curl/8.1.1/r/n       Accept: */*/r/n       /r/n       [Full request URI: http://localhost:8080/api/v1/employees]       [HTTP request 1/1]       [Response in frame: 294030] </pre>	<pre> 0000 02 00 00 00 45 00 00 31 00 00 00 00 00 00 00 00 .....E...@... 0001 7f 00 00 01 7f 00 00 01 07 c1 1f 50 09 bc b4 db ..... 0002 63 c5 32 ba 00 18 18 db fe 65 00 00 01 01 00 0a c-2----- 0003 a0 50 68 63 1a 2a 2c c3 47 45 54 20 2f 61 70 69 -Phc w, GET /api 0004 2f 76 31 2f 65 6d 70 6c 6f 75 65 65 73 20 48 54 /v1/empl oyes HT 0005 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 6c 6f TP/1.1 - Host: lo 0006 63 61 6c 68 6f 73 74 3a 30 38 30 30 0a 55 73 calhost: 8080 - 0007 65 72 24 41 67 65 6e 74 3a 20 63 75 72 6c 2f 38 er-Agent : curl/8 0008 2e 31 2e 31 0d 0a 41 63 63 65 70 74 3a 20 2a 2f 1.1 -Ac cept: */ 0009 2a 0d 0a 0d 0a ..... </pre>
--	--



# 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

The image shows a Wireshark packet capture of an HTTP transaction. The top section displays a list of packets. Packet 149 is a GET request for /api/v1/employees. Packet 150 is the corresponding 200 OK response. The packet details pane for packet 150 is expanded, showing the structure of the HTTP response, including the status bar, headers, and the JSON body.

```

149 GET /api/v1/employees HTTP/1.1
150 200 OK (application/json)

Frame 150: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface lo0, id 0
  Ethernet II, Src: Intel(R) Ethernet Controller (3:9:3d:3:9:3d), Dst: Intel(R) Ethernet Controller (3:9:3d:3:9:3d)
  Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  Transmission Control Protocol, Src Port: 8080, Dst Port: 55233, Seq: 215, Ack: 94, Len: 5
  [2 Resembled TCP Segments (215 bytes): #294820(124), #294830(51)]
  Hypertext Transfer Protocol, has 2 chunks (including last chunk)
    HTTP/1.1 200 OK
      -> Export Info (Chat/Sequence): HTTP/1.1 200 OK
      -> HTTP/1.1 200 OK
      -> [Severity level: Chat]
      -> [Group: Sequence]
      -> Response Version: HTTP/1.1
      -> Status Code: 200
      -> [Status Code Description: OK]
      -> Content-Type: application/json
      -> Transfer-Encoding: chunked
      -> Date: Sat, 16 Mar 2024 23:37:09 GMT
      -> HTTP response 1/1
      -> [Time since request: 0.095440000 seconds]
      -> [Request in frame: 294818]
      -> [Request URI: https://localhost:8080/api/v1/employees]
      -> HTTP chunked response
      -> File Data: 94 bytes
      -> JavaScript Object Notation: application/json
      -> Array
      -> Object
      -> Member: id
      -> [Path with value: /1/id]
      -> [Member with value: id:1]
      -> Number value: 1
      -> Key: id
      -> [Path: /1/id]
      -> Member: name
      -> [Path with value: /1/name]
      -> [Member with value: name: Mohamed Ahmed]
  
```





# API Design Principles

- Fundamentals of Good API Design
- Designing for Scalability and Performance ?
- API Versioning Strategies



# 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

Content-Type: application/json

Content-Type: image/png

Content-Type: application/x-www-form-urlencoded

Content-Type: multipart/form-data

Content-Type: video/mp4

source: [RFC 9110](#)





# Advanced API Protocols

- Introduction to GraphQL and Its Advantages
- Implementing gRPC for Microservices
- Comparison of Different API Styles



# API Documentation and Specification

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



# API Security

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



# 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

