APPLICATION PROGRAMMING INTERFACE (API)

EGCO343 SOFTWARE DESIGN

KANAT POOLSAWASD
DEPARTMENT OF COMPUTER ENGINEERING
MAHIDOL UNIVERSITY

APPLICATION PROGRAMMING INTERFACE (API)

- An API is a set of protocols and instructions written in programming languages that determine how two software components will communicate with each other.

API EXAMPLES

- APIs are a crucial behind-the-scenes aspect of user experience (UX). Consider a few familiar examples of APIs and how a website owner or administrator might use them:
  - The YouTube API allows you to add videos to your website or app, as well as manage your playlists and subscriptions.
  - The Facebook API for conversions allows you to track page visits and conversions, as well as provide data for ad targeting and reporting.
  - The Google Maps API allows you to embed static and dynamic maps, as well as street view imagery, on your website.

TYPES OF API

- **Open APIs** - also known as external or public APIs, are available for anyone to use and integrate with their sites or apps.
- **Partner APIs** - are also considered external, but you can use them only if you have a business relationship with the companies providing them.
- **Internal APIs** - also called private APIs, are used by people within a company and help to transfer data between teams or connect different systems and apps.
- **Composite APIs** - combine multiple APIs from different servers or data sources to create a unified connection to a single system.
- **Web Service API (or Web API)** - an application interface between a web browser and a web server

API ARCHITECTURAL STYLES

- SOAP (Simple Object Access Protocol) - determines how to transmit data across networks, how messages should be sent, and what the messages should include.
- REST (REpresentational State Transfer) - It's a set of guidelines for scalable APIs that are easy to use when transferring data securely. REST APIs are also known as RESTful APIs.
- RPC (Remote Procedural Call) - execute code on remote networks.
- Etc.

SOAP (Web Services)

WEB SERVICES

- Web services refers to a collection of standards that cover interoperability.
- In fact, these standards define both the protocols that are used to communicate and the format of the interfaces that are used to specify services and service contracts.

HISTORY OF WEB SERVICES (1)

- Microsoft coined the term Web Services in 2000, to describe a set of standards that allow computers to communicate with each other via network.
- One of core standards is the eXtensible Markup Language (XML); another is HTTP.
- Together with some others, Microsoft had already begun working on a protocol called SOAP that used XML to exchange data over a native connection based on HTTP and TCP/IP.
- Then IBM jumped in, and as a result autumn 2000, two other standard were announced: the Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI).
- At the end of 2000, Oracle, HP, and Sun also announced their intention to support and deploy the Web Services in their products.

HISTORY OF WEB SERVICES (2)

- Nowadays, Web Services is a major movement based on several standards, and driven by many companies and standardisation organizations.
- The names of the standards usually start with Web Services based on about 10 low-level standards for XML and HTTP, there are more than 50 Web Services standards, plus about 10 profiles specified by different standards bodies, such as the World Wide Web Consortium (W3C), OASIS, and Web Services Interoperability (WS-I).

FUNDAMENTAL WEB SERVICES STANDARDS (1)

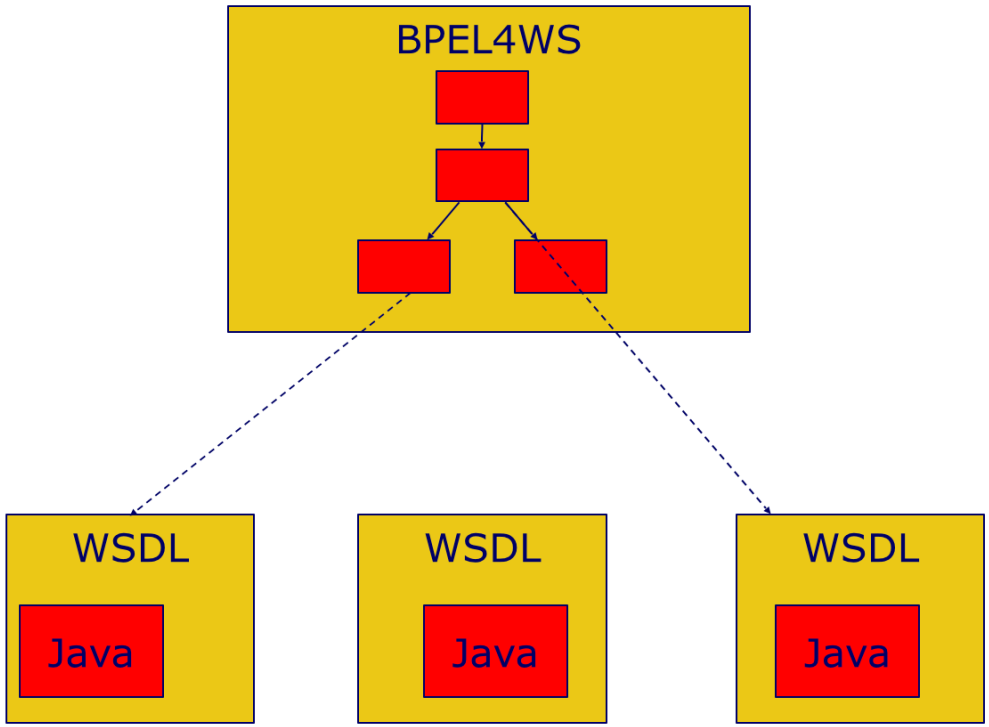
- There are 5 fundamental Web Services standards. Two of them are general standards that existed beforehand and were used to realize the Web Services approach:
  - XML is used as the general format to describe models, formats, and data types. Most other standards are XML standards. In fact, all Web Services standards are based on XML 1.0, XSD (XML Schema Definition), and XML namespaces.
  - HTTP (including HTTPS) is the low-level protocol used by the Internet.

FUNDAMENTAL WEB SERVICES STANDARDS (2)

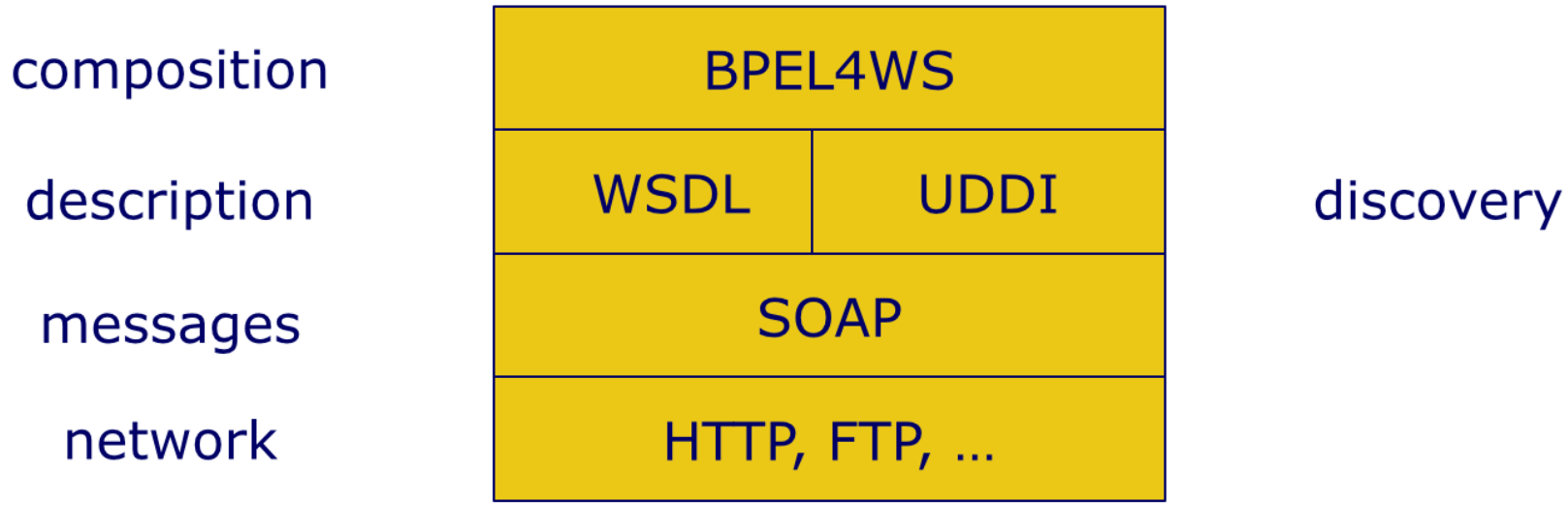
- The other three fundamental standards are specific to Web Services and were the first standards specified for them:
  - WSDL is used to define service interfaces. In fact, it can describe two different aspects of a service: its signature (name and parameters) and its binding and deployment details (protocol and location).
  - SOAP is a standard that defines the Web Services protocol. While HTTP is the low-level protocol, also used by the Internet, SOAP is the specific format for exchanging Web Services data over this protocol.
  - UDDI is a standard for managing Web Services (i.e., registering and finding services)

FUNDAMENTAL WEB SERVICES STANDARDS (3)

- BPEL4WS: Business Process Execution Language for Web Services
- Main standardisation bodies: OASIS, W3C



WEB SERVICES STACK



XML (EXTENSIBLE MARKUP LANGUAGE)

- Looks like HTML
- Language/vocabulary defined in schema: collection of trees
- Only syntax
- Semantic Web, Web 2.0: semantics as well: OWL and descendants

JSON VS. XML

- JSON Example:

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

- XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

- SOAP is message inside an envelope
- Envelop has optional header (~address), and mandatory body: actual container of data
- You can see that the SOAP message have an XML format containing a root element called <Envelope>. It might contain an optional <Header> and a mandatory <Body> element.
- While the <body> element contains the payload (request, response, or fault data), the <header> can contain additional information to help the infrastructure deal with the messages.
- SOAP message is unidirectional: it's NOT a conversation

SOAP EXAMPLE (1)

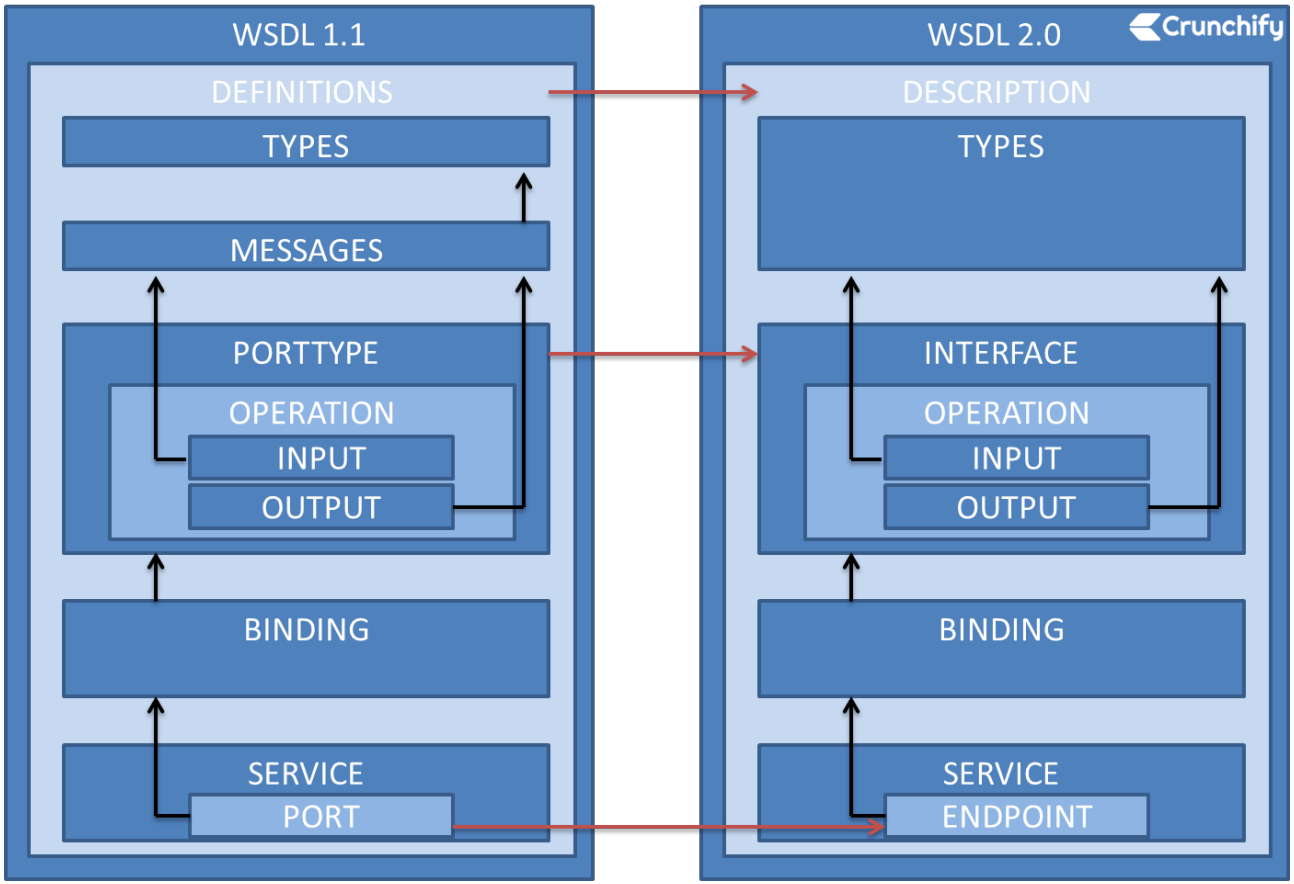
```
<?xml version='1.0' ?>
<soap:Envelope xmlns:soap="http://a.soap.org/soap/">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    <getCustomerAddress xmlns="http://soa.com/xsd">
      <customerID>123456789</customerID>
    </getCustomerAddress>
  </soap:Body>
</soap:Envelope>
```

SOAP EXAMPLE (2)

```
<?xml version='1.0' ?>
<soap:Envelope xmlns:soap="http://a.soap.org/soap/">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    <getCustomerAddress xmlns="http://soa.com/xsd">
      <Address>
        <street>Gaussstr.29</street>
        <city>Braunschweig</city>
        <zipCode>D-38106</zipCode>
      </Address>
    </getCustomerAddress>
  </soap:Body>
</soap:Envelope>
```

WSDL (1)

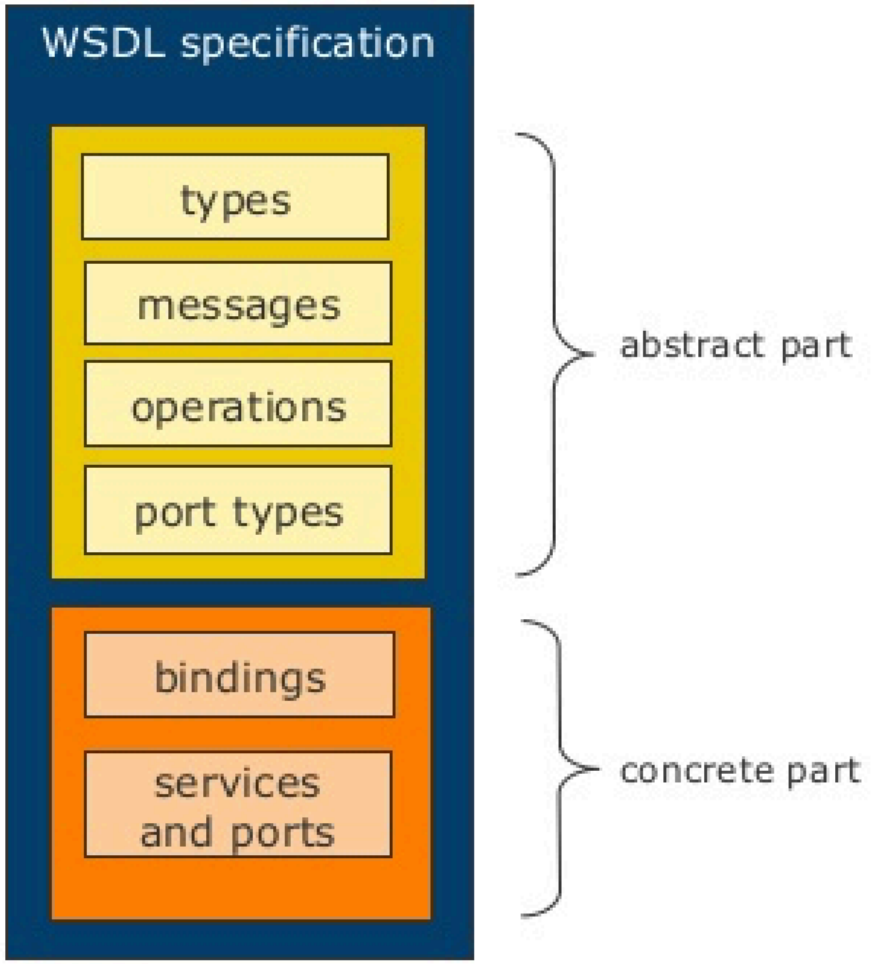
- WSDL (Web Services Description Language)
- To understand what WSDL files are all about, let's begin by looking at the general structure of a WSDL file.



WSDL (2)

- WSDL files describe services from the bottom up. That is, they start with the data types used and end with the location (address) of the service.
- In addition, they have three layers of description
  - First layer describes the interface of a service. This interface (called “port type” in WSDL 1.1) can consist of one or more operations with input and output parameters that use types specified in the first section of the WSDL file.
  - Second layer defines the “binding” of a Web Service; that is, the protocol format for which the Web Service operations are provided.
  - Third layer defines the physical location (address, URL) where the Web Services is available.

WSDL (3)




```
<definitions>

 <types>
  data type definitions
 </types>

 <message>
  definition of the data being communicated
 </message>

 <portType>
  <operation>
   set of operations
  </operation>
 </portType>

 <binding>
  protocol and data format specification
 </binding>

 <service>
  location of the service
 </service>

</definitions>
```

UDDI

- UDDI (Universal Description, Discovery and Integration)
- The original idea was to introduce all three roles of a working marketplace: providers that offer services, consumers that need services, and brokers that bring them together by locating services.
- Three (main) parts:
  - Info about organization that publishes the services
  - Descriptive info about each service
  - Technical info to link services to implementation
- Original dream: one global registry
- Reality: many registries, with different levels of visibility
  - Mapping problems

BPEL4WS

- Three main parts:
  - Partner-links: dependencies between services: who sends what to whom
  - Global variables
  - Workflow model: “program”
- BPEL4WS is an orchestration language; executable
- WS-CDL (Web Services Choreography Description Language) is a choreography language; not executable

REST / RESTful

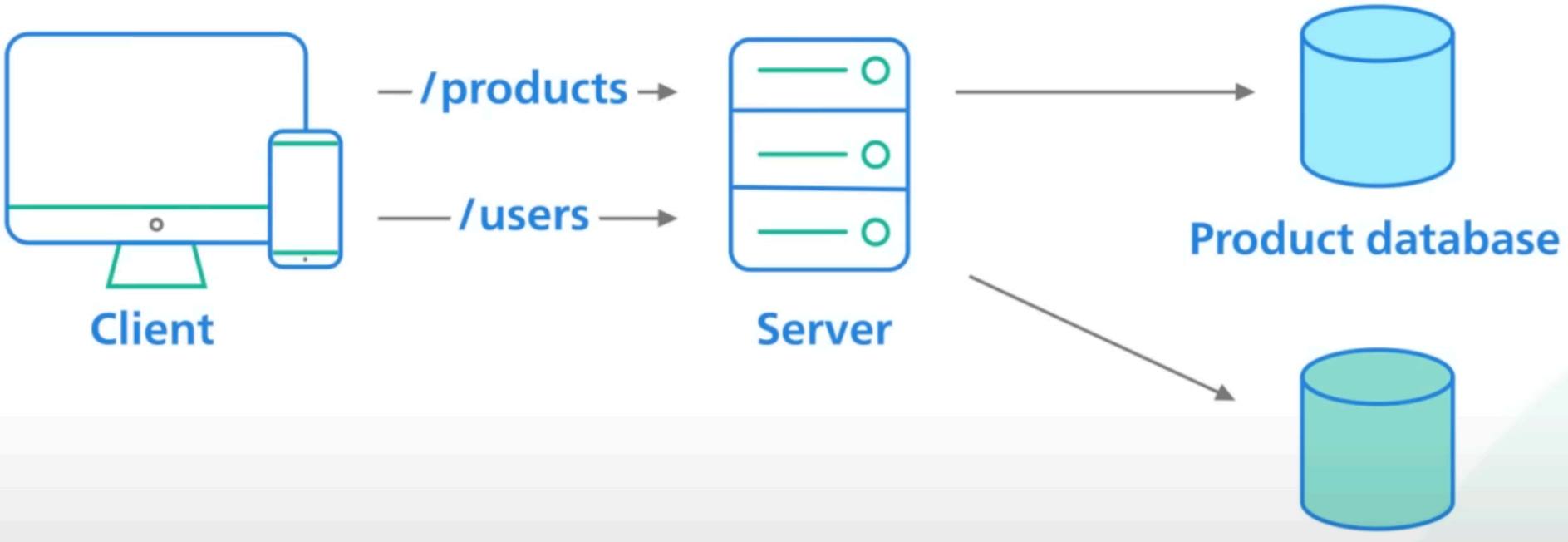
REST API (1)

- REST (REpresentational State Transfer)
- REST APIs are a type of Web Service APIs.
- REST API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources.

REST API (2)

URI
(Endpoint)

```
https://example.com/api/v3/products
https://example.com/api/v3/users
```



* URI (Uniform Resource Identifiers)

REST API (3)

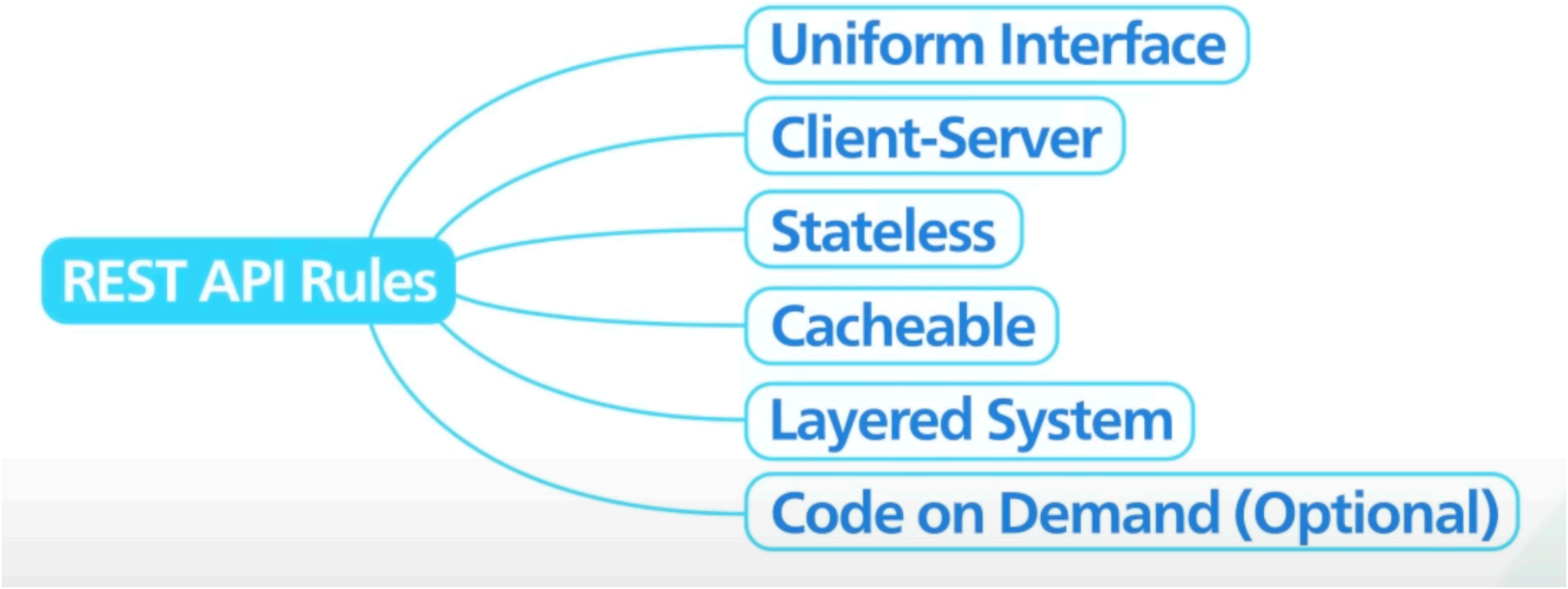

POST /products HTTP/1.1
Accept: application/json
JSON:
{
"customer": "Jason Sweet"
"quantity": 1,
"price": 18.00
}
Client
Server

REST API (4)

REST ARCHITECTURAL CONSTRAINTS (1)

- REST is a software architectural style that defines the set of rules to be used for creating web services.
- Web services which follow the REST architectural style are known as RESTful web services.
- Interaction in REST based systems happen through Internet's Hypertext Transfer Protocol (HTTP)
- There are six architectural constraints which makes any web service are listed next slide:

REST ARCHITECTURAL CONSTRAINTS (2)

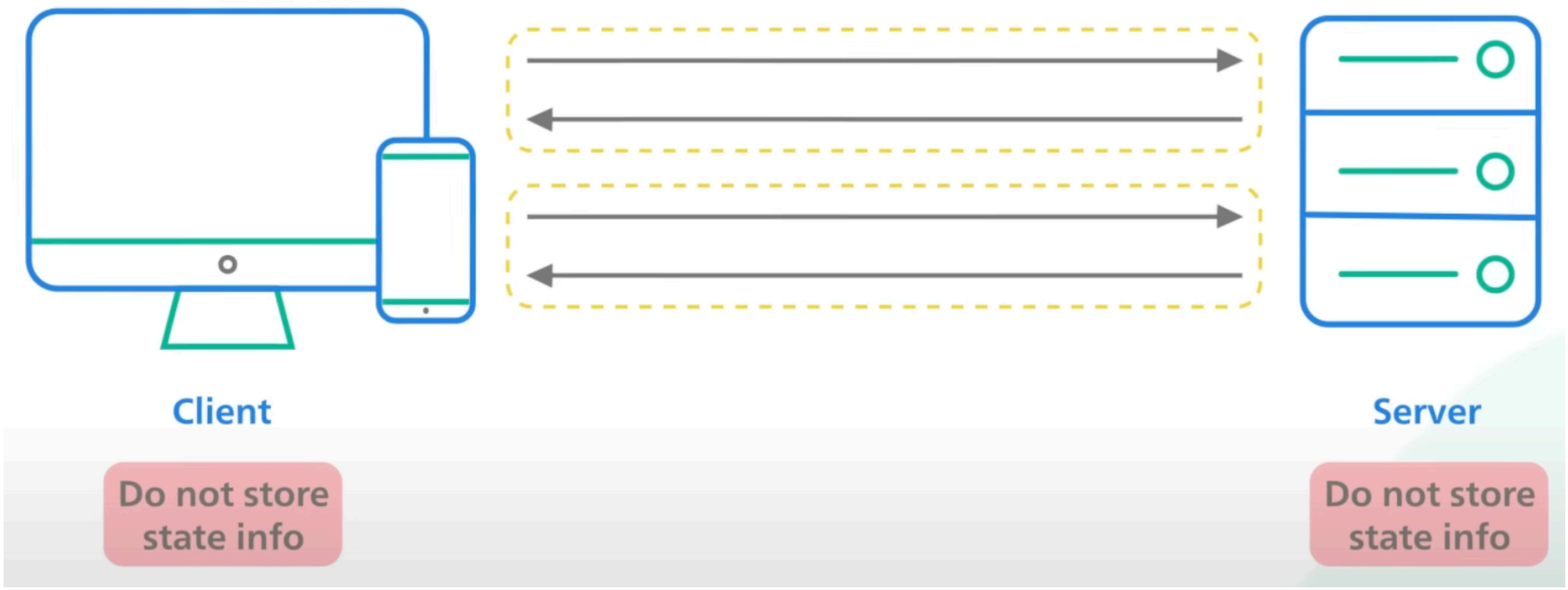


CLIENT-SERVER

- REST application should have a client-server architecture.
- A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server, and server is someone who holds the resources and are not concerned with the user interface or user state.
- They can evolve independently. Client doesn’t need to know anything about business logic and server doesn’t need to know anything about frontend UI.

STATELESS (1)

- Stateless means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session.



STATELESS (2)

- Statelessness enables greater availability since the server does not have to maintain, update or communicate that session state. There is a drawback when the client need to send too much data to the server so it reduces the scope of network optimization and requires more bandwidth.

CACHEABLE

- Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side.
- Client will return the data from its cache for any subsequent request and there would be no need to send the request again to the server.
- A well-managed caching partially or completely eliminates some client–server interactions, further improving availability and performance. But sometime there are chances that user may receive stale data.

LAYERED SYSTEM

- An application architecture needs to be composed of multiple layers.
- Each layer doesn’t know any thing about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server.
- Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.

UNIFORM INTERFACE (1)

- Uniform Interface suggests that there should be an uniform way of interacting with a given server irrespective of device or type of application.
- There are four guidelines principle of Uniform Interface are:
  - Resource-Based
  - Manipulation of Resources Through Representations
  - Self-descriptive Messages
  - Hypermedia as the Engine of Application State (HATEOAS)

UNIFORM INTERFACE (2)

GET request fetches an account resource, requesting details in a JSON representation

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
```

The response is:

```
HTTP/1.1 200 OK

{
    "account": {
        "account_number": 12345,
        "balance": {
            "currency": "usd",
            "value": 100.00
        },
        "links": {
            "deposits": "/accounts/12345/deposits",
            "withdrawals": "/accounts/12345/withdrawals",
            "transfers": "/accounts/12345/transfers",
            "close-requests": "/accounts/12345/close-requests"
        }
    }
}
```

CODE ON DEMAND (OPTIONAL)

- Code on demand is an optional feature.
- According to this, servers can also provide executable code to the client.
- The examples of code on demand may include the compiled components such as Java Servlets and Server-Side Scripts such as JavaScript.

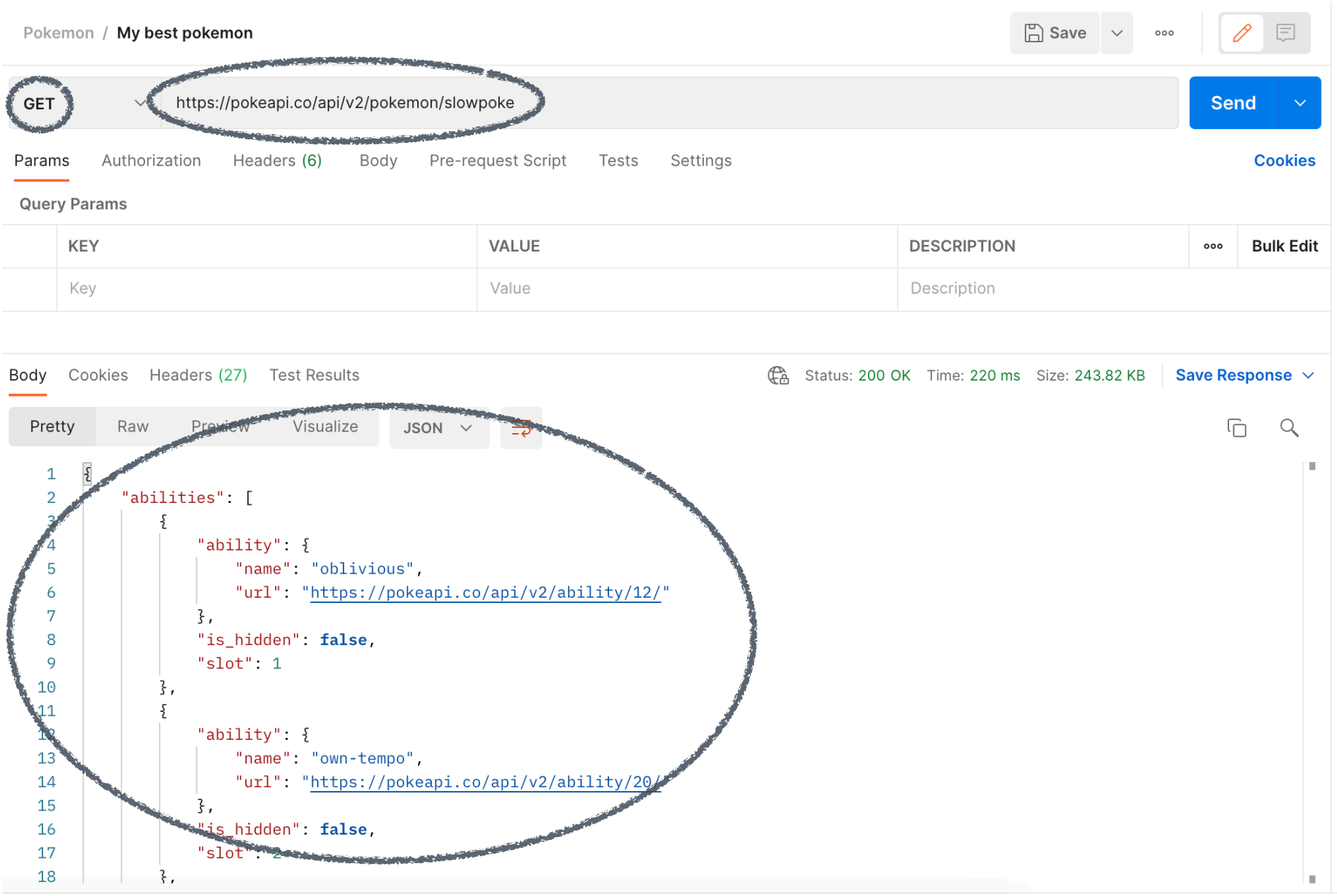
RULES OF REST API

- REST is based on the resource or noun instead of action or verb based. It means that a URI of a REST API should always end with a noun.
  - Example: /api/users is a good example, but /api?type=users is a bad example of creating a REST API.
- HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE, PATCH.
- A web application should be organized into resources like users and then uses HTTP verbs like – GET, PUT, POST, DELETE to modify those resources (CRUD).

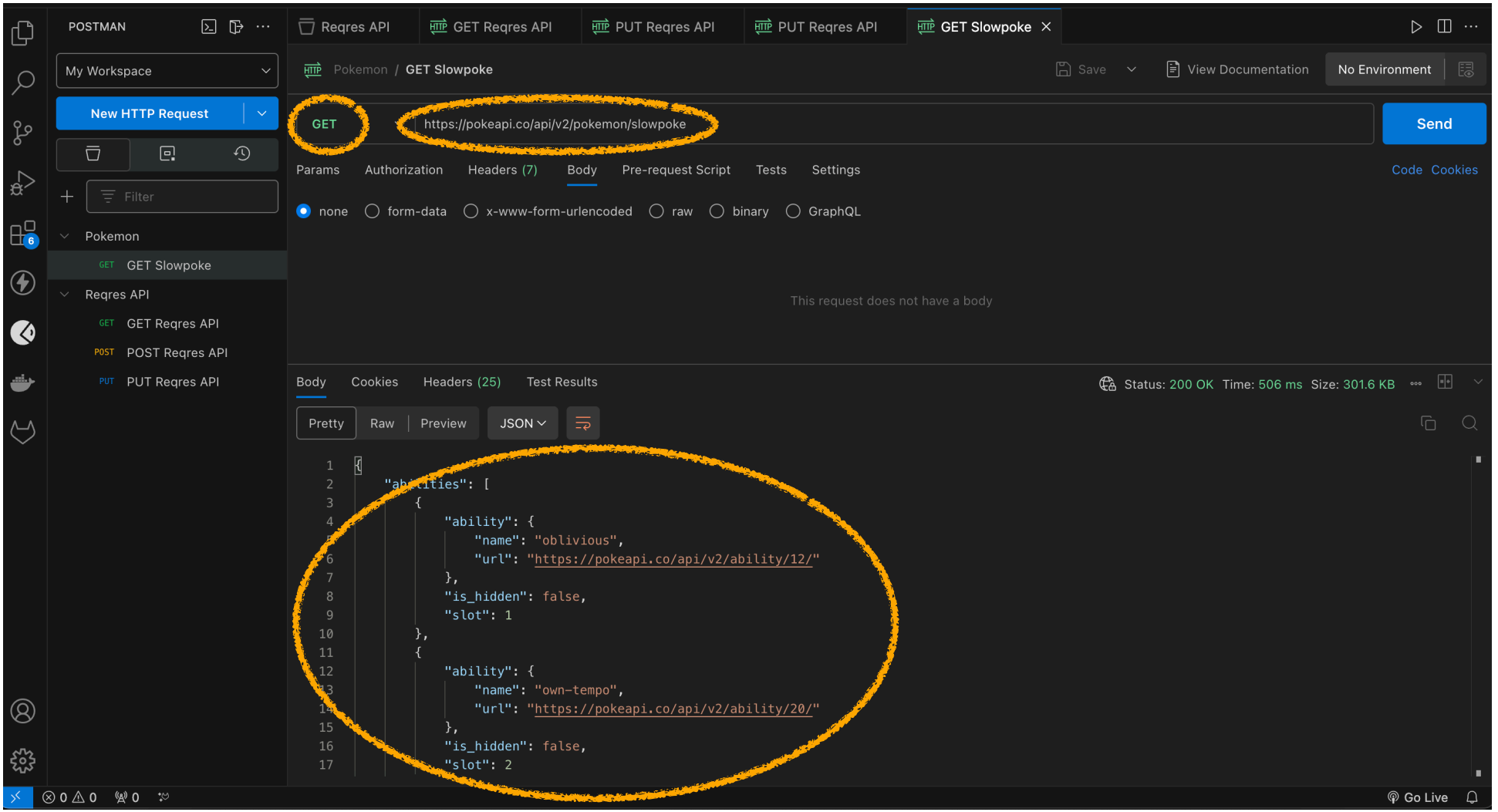
EXAMPLE

URI	HTTP Verb	Description
api/users	GET	Get all users.
api/users/new	GET	Show form for adding new user.
api/users	POST	Add a user.
api/users/1	PUT	Update a user with id = 1
api/users/1/edit	GET	Show edit form for user with id = 1
api/users/1	DELETE	Delete a user with id = 1
api/users/1	GET	Get a user with id = 1

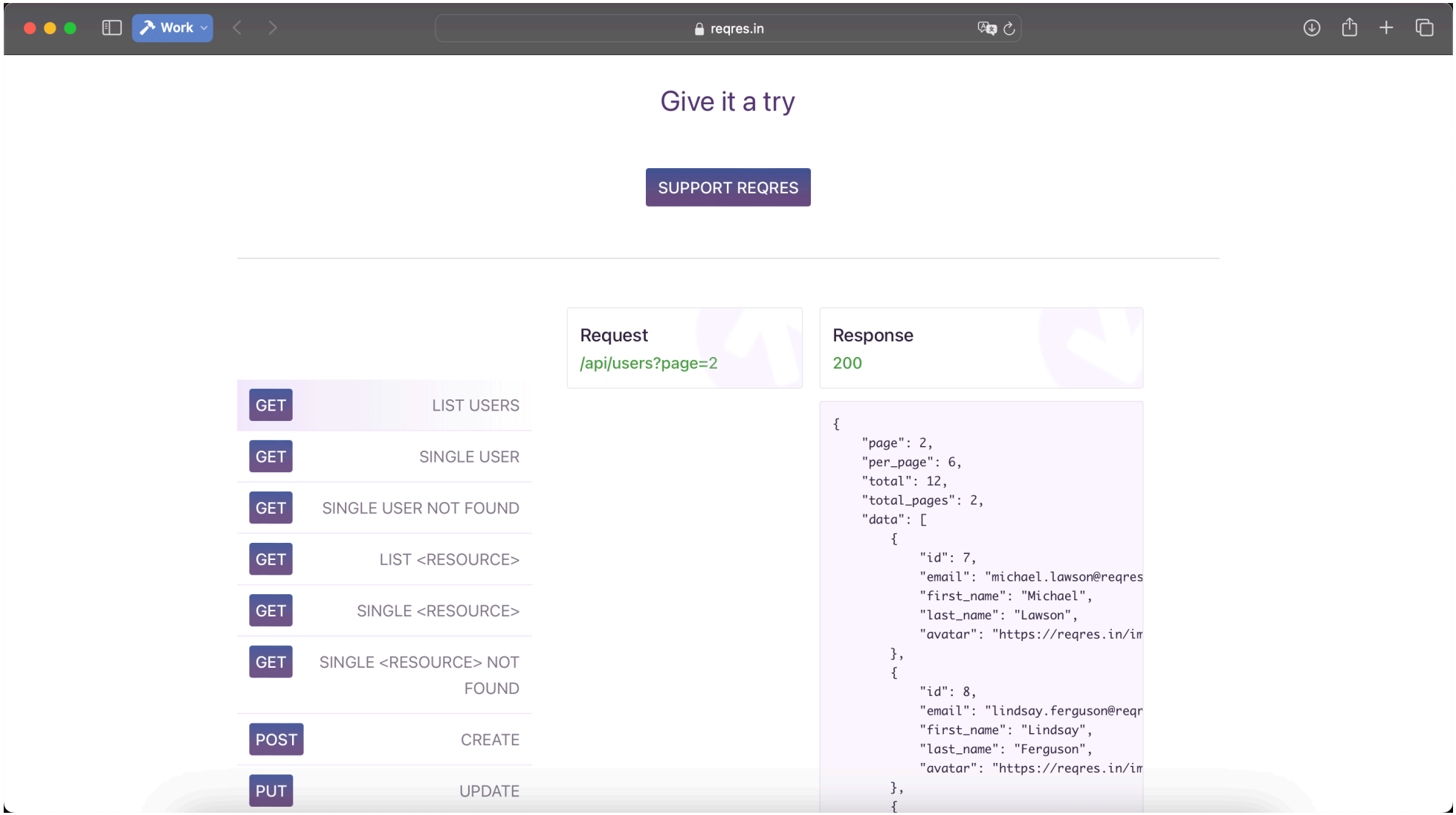
REST API EXAMPLE



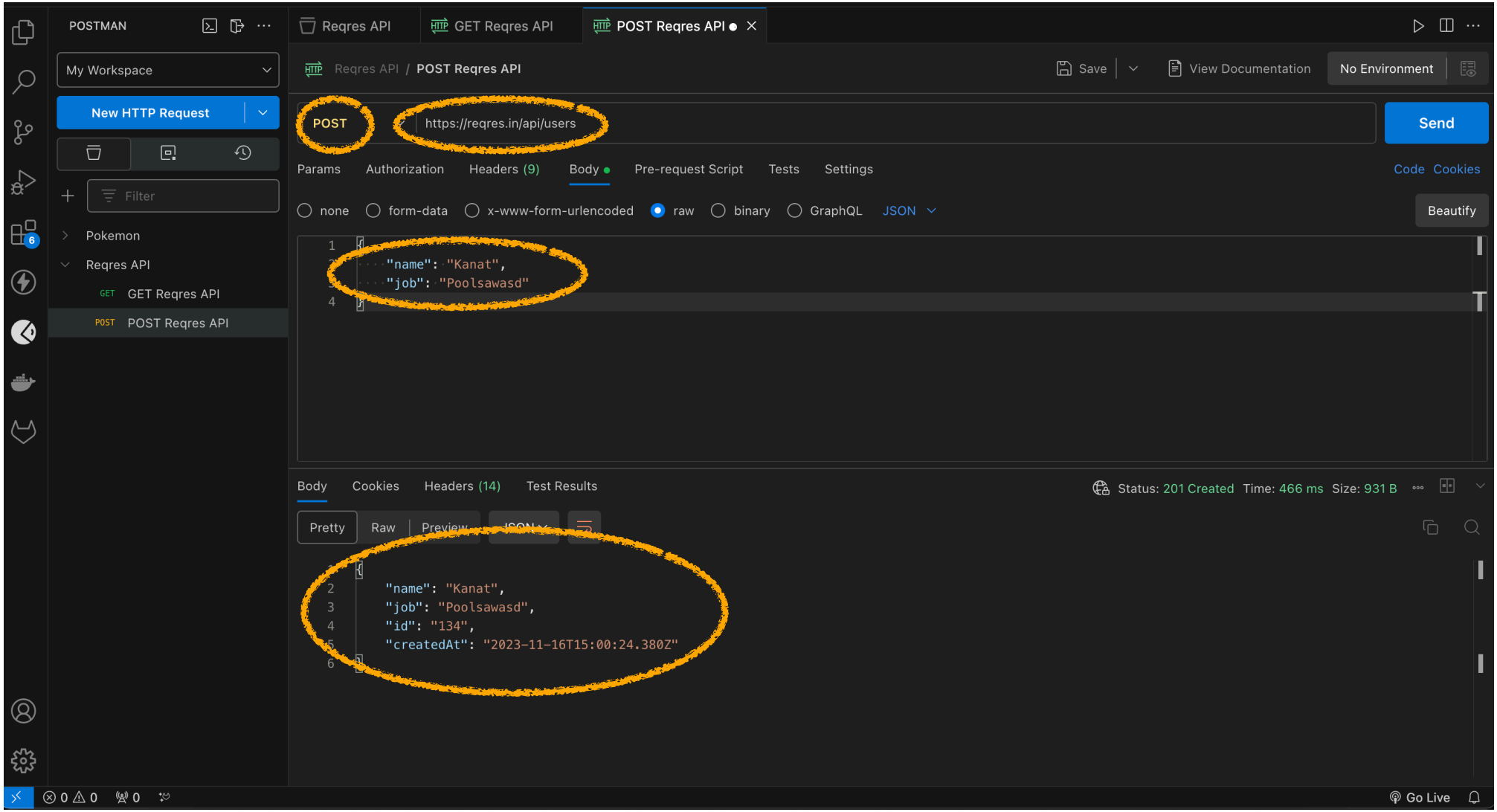
POSTMAN IN VSCOSE



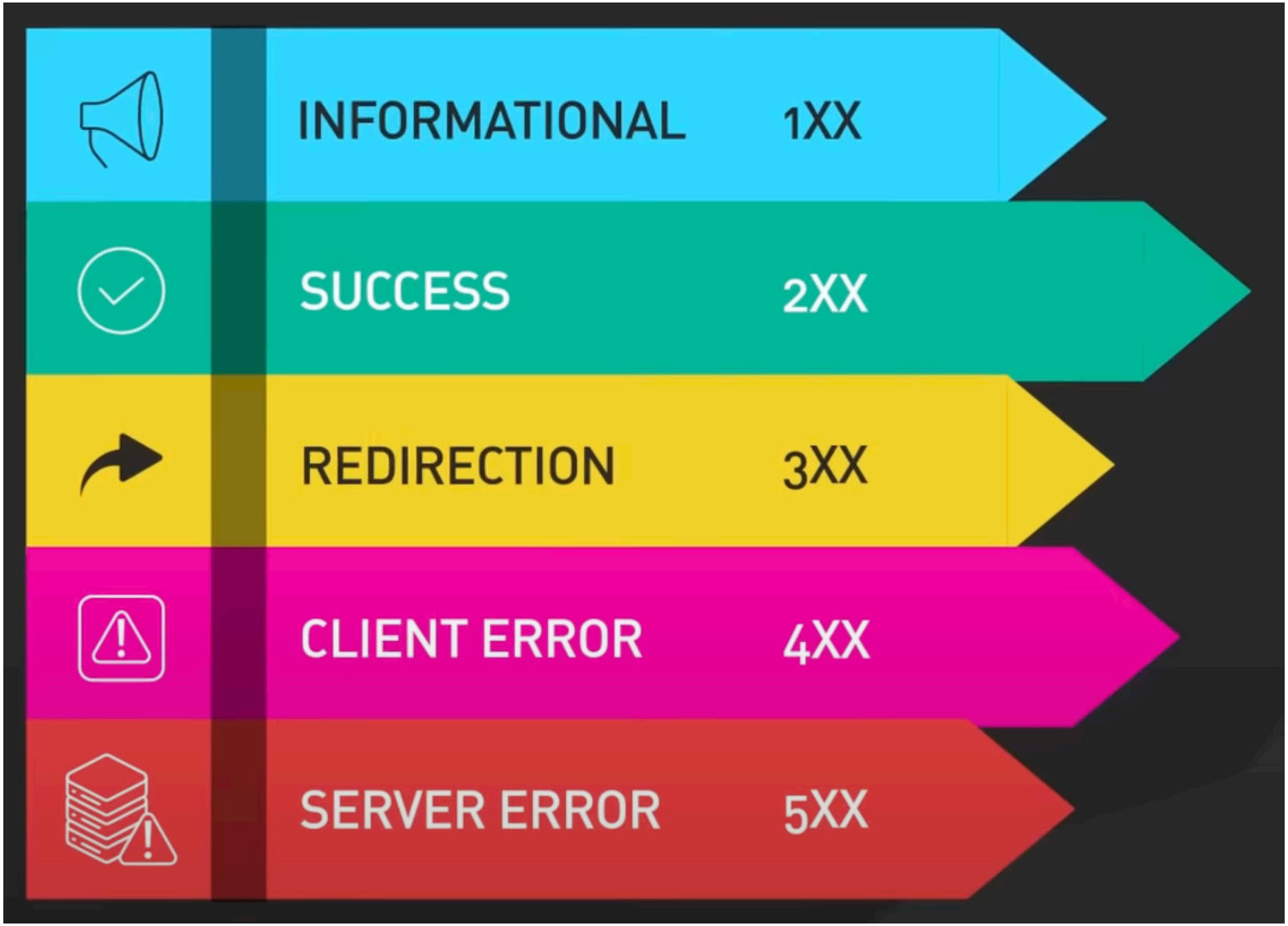
REQRES.IN (1)


Work
reqres.in
Give it a try
SUPPORT REQRES
GET LIST USERS
GET SINGLE USER
GET SINGLE USER NOT FOUND
GET LIST <RESOURCE>
GET SINGLE <RESOURCE>
GET SINGLE <RESOURCE> NOT FOUND
POST CREATE
PUT UPDATE
Request
/api/users?page=2
Response
200
{
"page": 2,
"per_page": 6,
"total": 12,
"total_pages": 2,
"data": [
{
"id": 7,
"email": "michael.lawson@reqres
"first_name": "Michael",
"last_name": "Lawson",
"avatar": "https://reqres.in/im
},
{
"id": 8,
"email": "lindsay.ferguson@reqr
"first_name": "Lindsay",
"last_name": "Ferguson",
"avatar": "https://reqres.in/im
},
{

REQRES.IN (2)


POSTMAN
My Workspace
New HTTP Request
Filter
Pokemon
Reqres API
GET GET Reqres API
POST POST Reqres API
Reqres API
GET Reqres API
POST Reqres API
Reqres API / POST Reqres API
Save
View Documentation
No Environment
POST
https://reqres.in/api/users
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Code
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Kanat",
"job": "Poolsawasd"
}
Body
Cookies
Headers (14)
Test Results
Status: 201 Created
Time: 466 ms
Size: 931 B
Pretty
Raw
Preview
JSON
{
"name": "Kanat",
"job": "Poolsawasd",
"id": "134",
"createdAt": "2023-11-16T15:00:24.380Z"
}
Go Live

HTTP STATUS CODES (1)



HTTP STATUS CODES (2)

1XX Information	
100	Continue
101	Switching Protocols
102	Processing
103	Early Hints

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
205	Reset Content
206	Partial Content
207	Multi-status (WebDAV)
208	Already Reported (WebDAV)
226	IM Used (HTTP Delta Encoding)

3XX Redirection	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
306	Unused
307	Temporary Redirect
308	Permanent Redirect

HTTP STATUS CODES (3)

4XX Client Error	
400	Bad Request
401	Unauthorised
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	URI Too Large
415	Unsupported Media Type
416	Range Not Satisfiable
417	Exception Failed
418	I'm a Teapot
421	Misdirected Request
422	Unprocessable Entity (WebDAV)
423	Locked (WebDAV)
424	Failed Dependency (WebDAV)
425	Too Early
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields too Large
451	Unavailable for Legal Reasons
499	Client Closed Request

HTTP STATUS CODES (4)

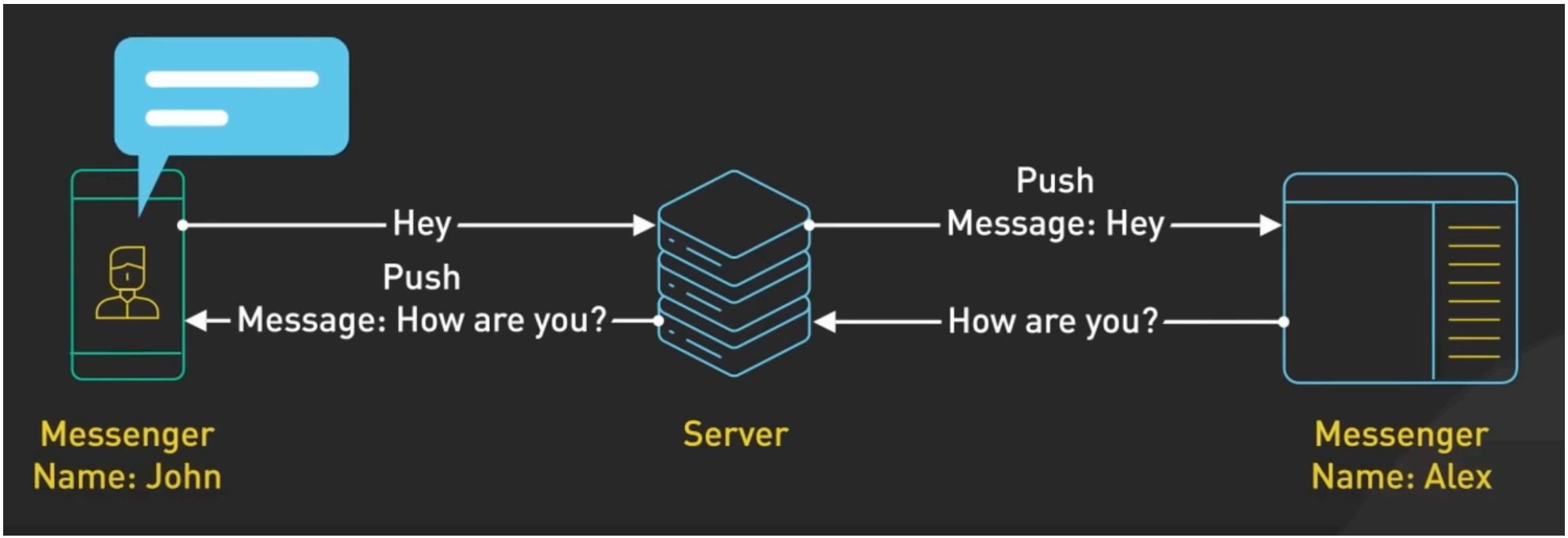
5XX Server Error Responses	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unvailable
504	Gateway Timeout
505	HTTP Version Not Supported
507	Insufficient Storage (WebDAV)
508	Loop Detected (WebDAV)
510	Not Extended
511	Network Authentication Required
599	Network Connection Timeout Error

Web Socket

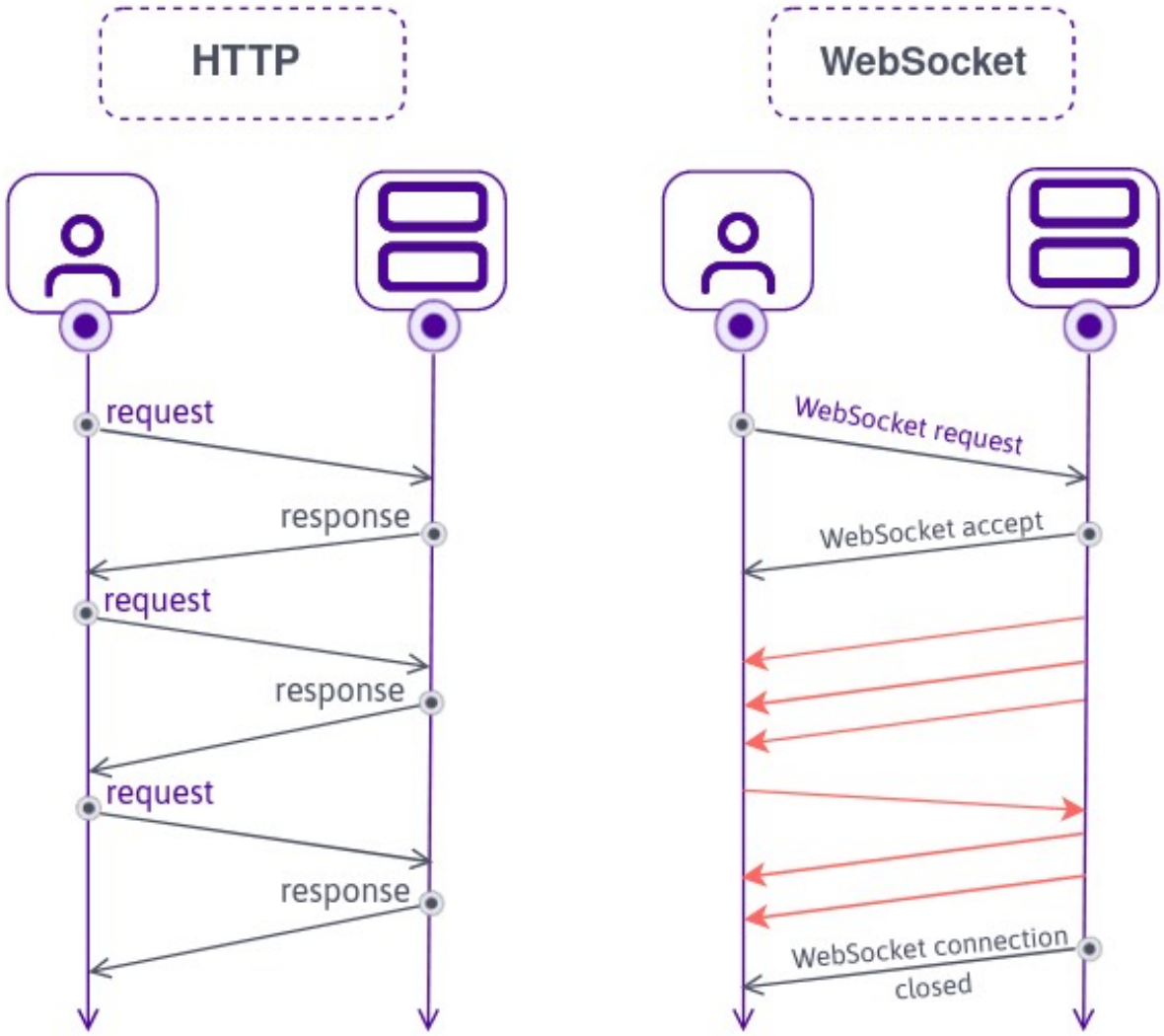
WEB SOCKET (1)

- WebSocket is bidirectional, a full-duplex protocol that is used in the same scenario of client-server communication, unlike HTTP it starts from ws:// or wss://.
- It is a stateful protocol, which means the connection between client and server will keep alive until it is terminated by either party (client or server).
- After closing the connection by either of the client and server, the connection is terminated from both ends.

WEB SOCKET (2)


Hey
Push
Message: How are you?
Push
Message: Hey
How are you?
Messenger
Name: John
Server
Messenger
Name: Alex

WEB SOCKET (3)


HTTP
WebSocket
request
response
request
response
request
response
WebSocket request
WebSocket accept
WebSocket connection
closed

WHEN CAN A WEB SOCKET BE USED (1)

- **Real-time web application** uses a web socket to show the data at the client end, which is continuously being sent by the backend server. In WebSocket, data is continuously pushed/transmitted into the same connection which is already open, that is why WebSocket is faster and improves the application performance.
- For example, in a trading website or bitcoin trading, for displaying the price fluctuation and movement data is continuously pushed by the backend server to the client end by using a WebSocket channel.

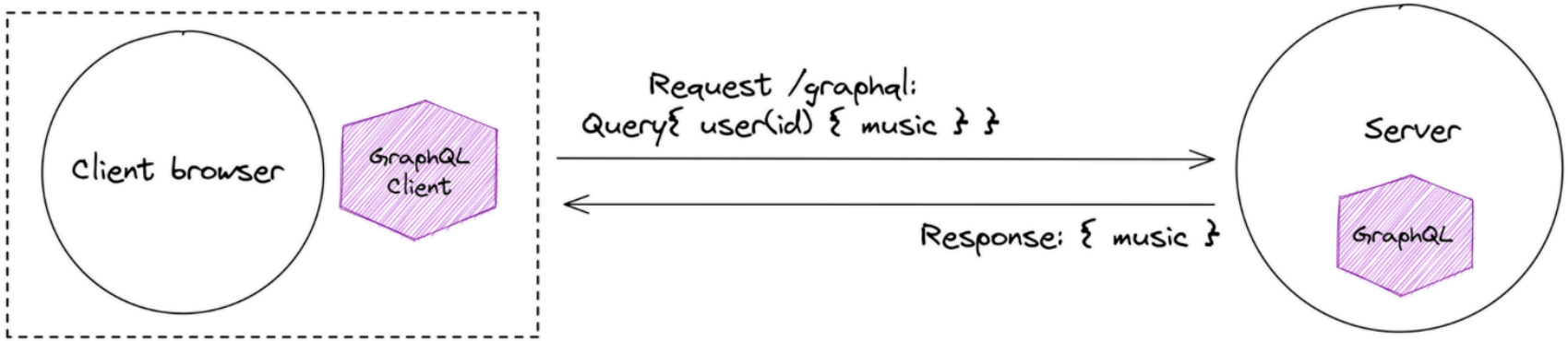
WHEN CAN A WEB SOCKET BE USED (2)

- **Gaming application**, you might focus on that, data is continuously received by the server, and without refreshing the UI, it will take effect on the screen, UI gets automatically refreshed without even establishing the new connection, so it is very helpful in a Gaming application.
- **Chat applications** use WebSockets to establish the connection only once for exchange, publishing, and broadcasting the message among the subscribers. It reuses the same WebSocket connection, for sending and receiving the message and for one-to-one message transfer.

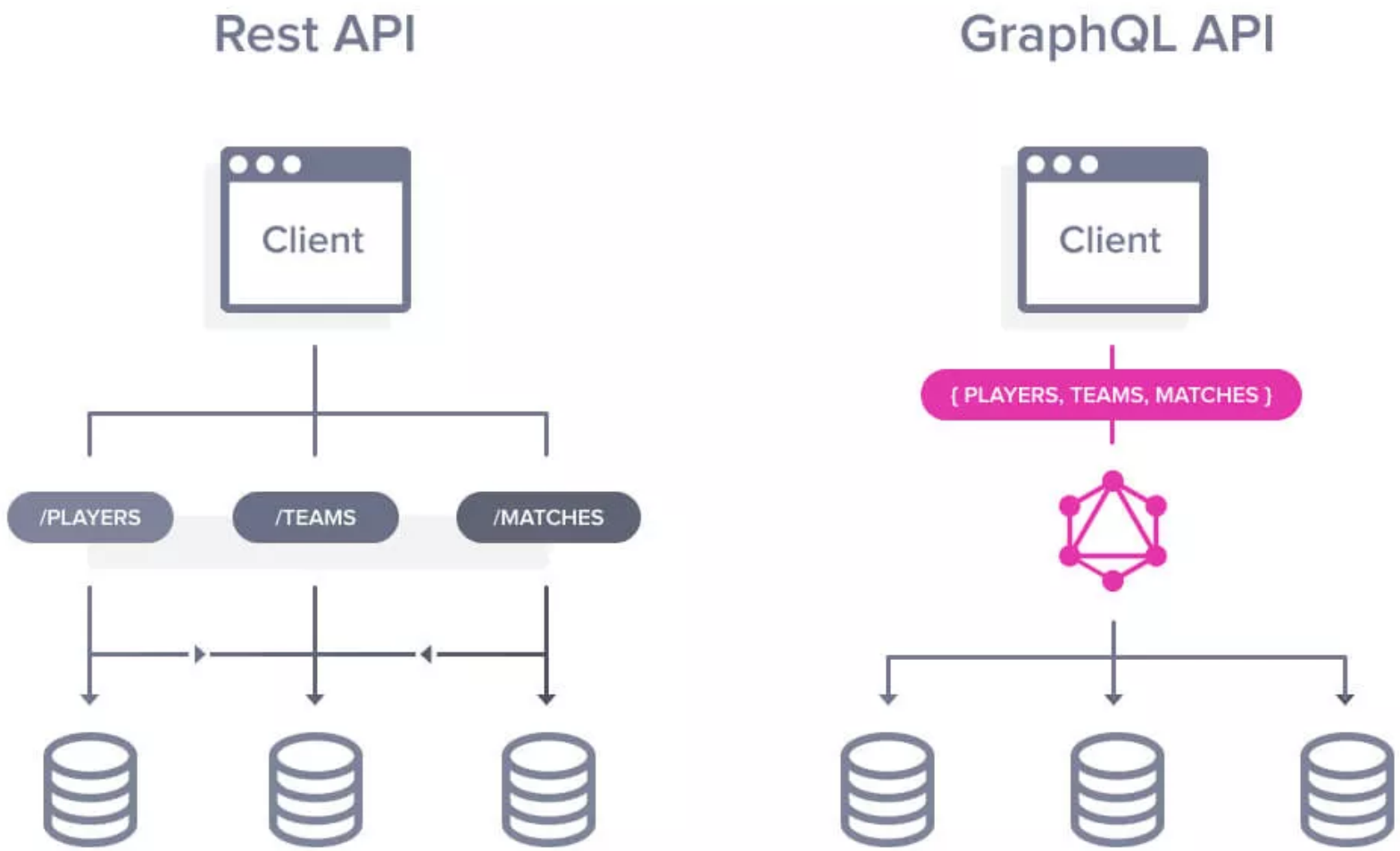
GraphQL

GRAPHQL

- GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.
- GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.



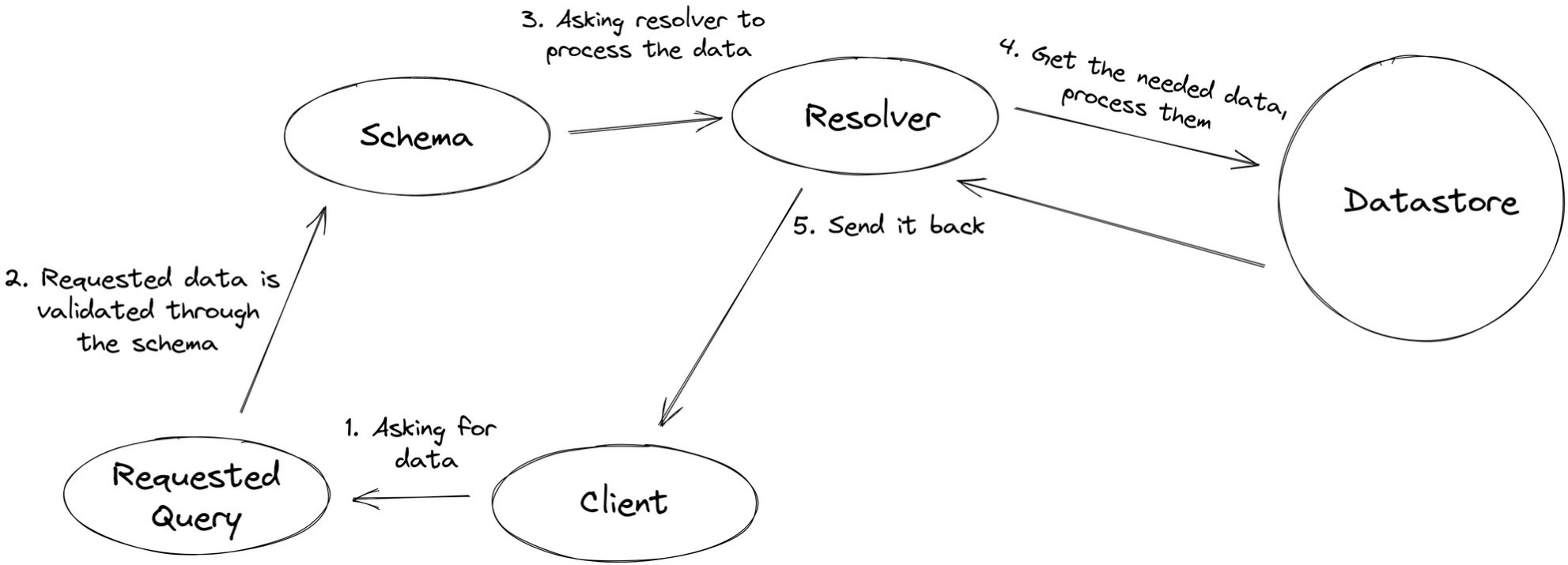
GRAPHQL VS REST (1)



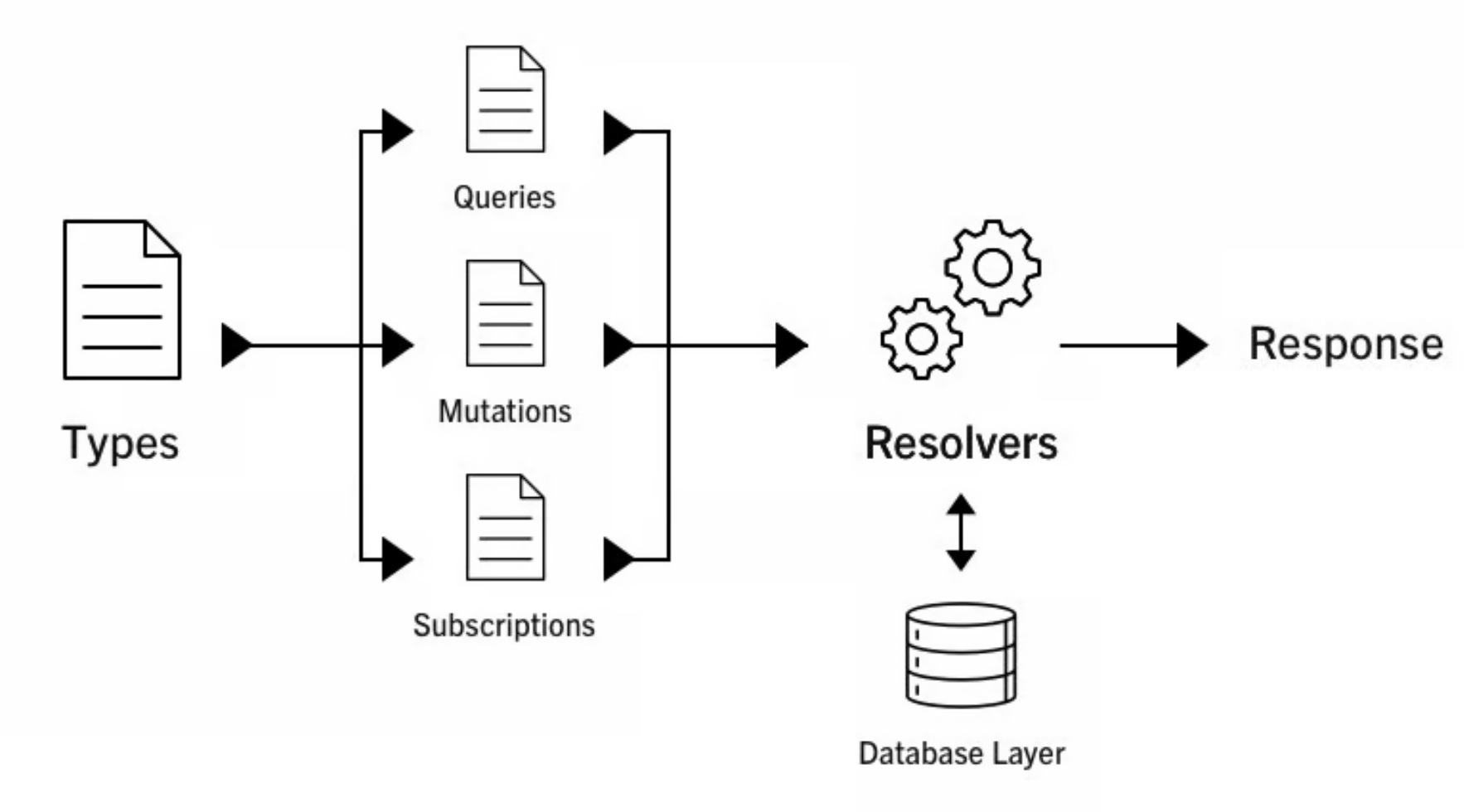
GRAPHQL VS REST (2)

	GraphQL	**REST**
Architecture	Client-driven	Server-driven
Organized	Schema & Type System	Endpoints
Data Fetching	Specific data with single call	Fixed data and multiple call
Performance	Fast	Take up more time
Dev. Speed	Rapid	Slower
Learning Curve	Difficult	Moderate
Use-Cases	Microservices / Mobile Apps	Simple Apps / Resource Driven Apps

HOW DOES GRAPHQL WORK ?



HOW DOES GRAPHQL WORK ?



QUERIES AND MUTATION

- A GraphQL operation can either be
  - **Read** operation, known as **Query**, is used to read or fetch values
  - **Write** operation, known as **Mutation**, is used to write or delete values.
- In both cases, the operation is a simple string that a GraphQL server can parse and respond to with specifically formatted data. JSON is usually the popular response format for mobile and web applications.

SCHEMA

- A schema is a collection of GraphQL types.
- GraphQL uses it to describe the shape of your available data.
- Schema is one of the key concepts when working with GraphQL API. It defines how the client can request the data and specifies the capabilities of the API.
- It’s like a contract between the server and the client.

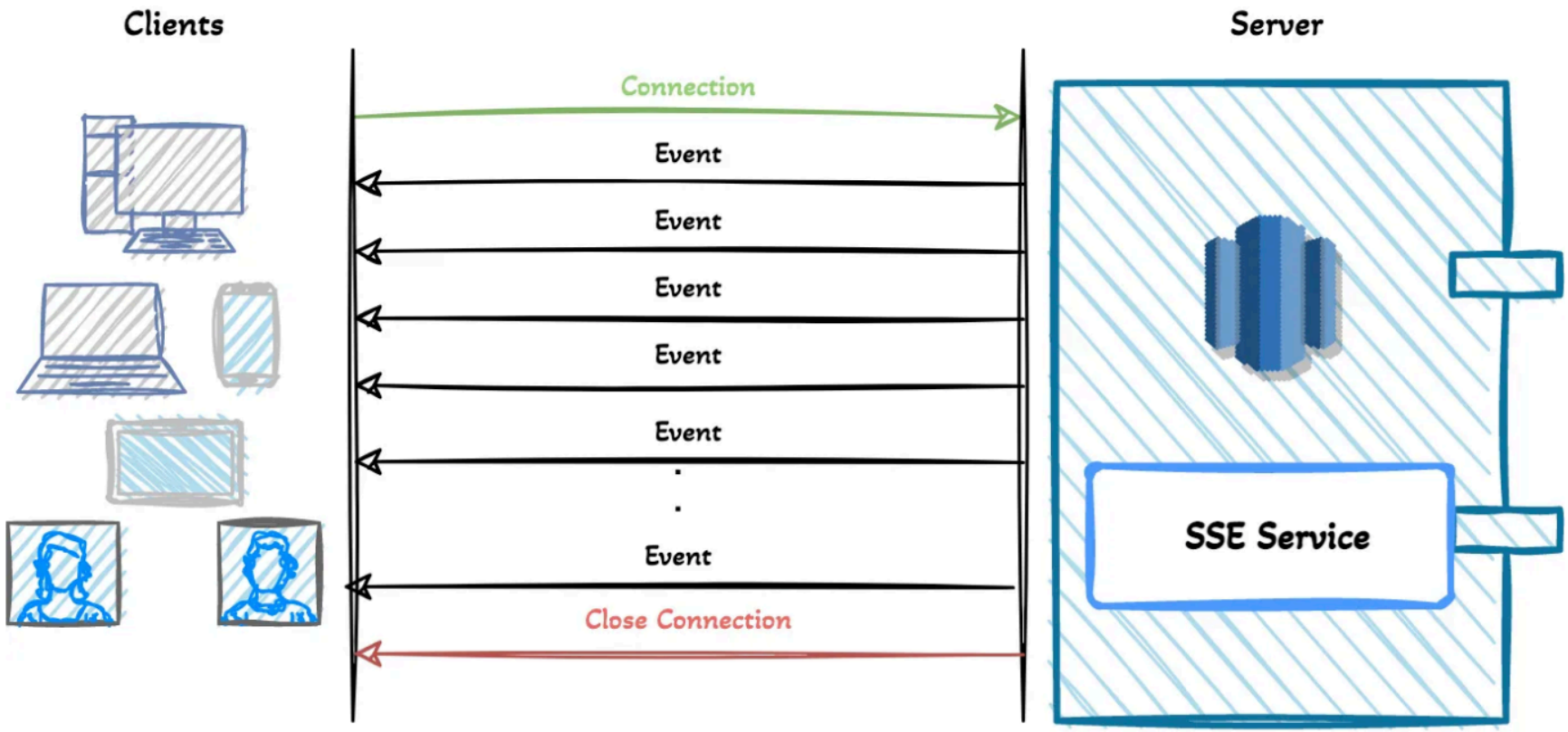
RESOLVERS

- Resolver is a function that generates responses for GraphQL queries & mutations by acting as a GraphQL query handler.
- It's defined within the GraphQL schema that a particular query will use the resolver, so it is responsible for fetching data, processing data, then transforming the fetched and processed data into a GraphQL array format.
- Then, the resolver returns the results wrapped by a callable function.

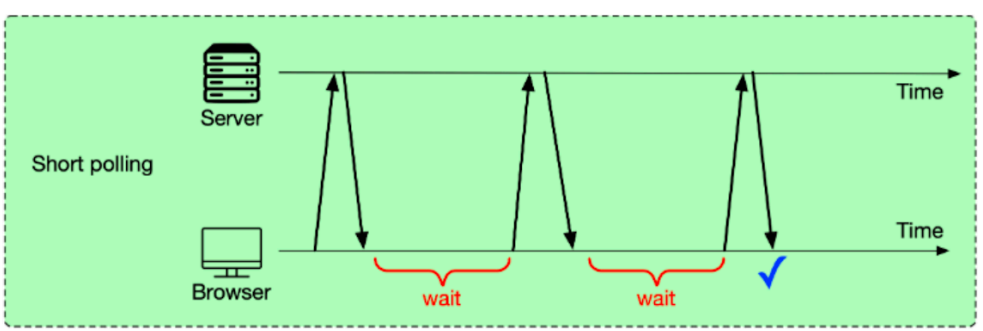
SERVER-SENT EVENTS (SSE)

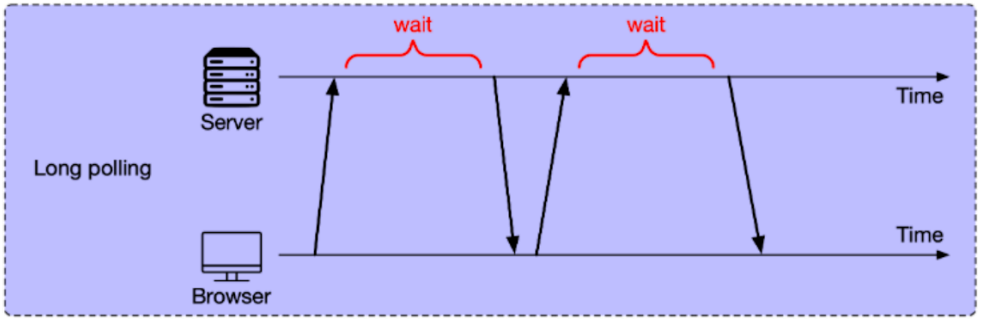
- Server-sent events (SSE) is a technology for transmitting data from a server to a web client in real-time. It was introduced as part of the HTML5 specification and has been supported by modern web browsers since around 2011.
- SSE is based on the concept of long-lived HTTP connections, where the client establishes a persistent connection to the server and the server can send data to the client at any time.

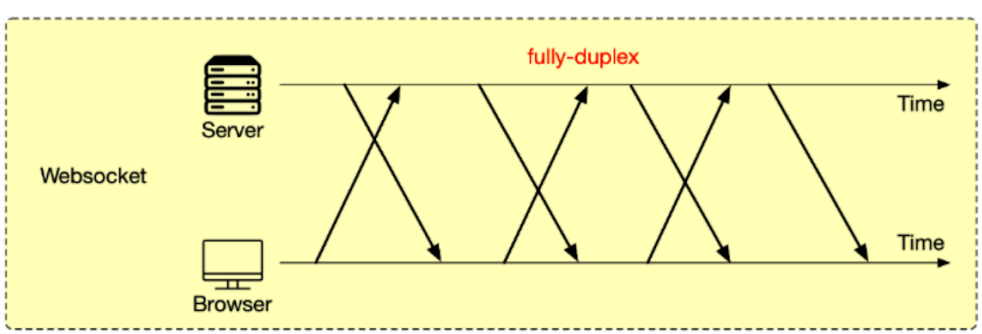
SERVER-SENT EVENTS (SSE)

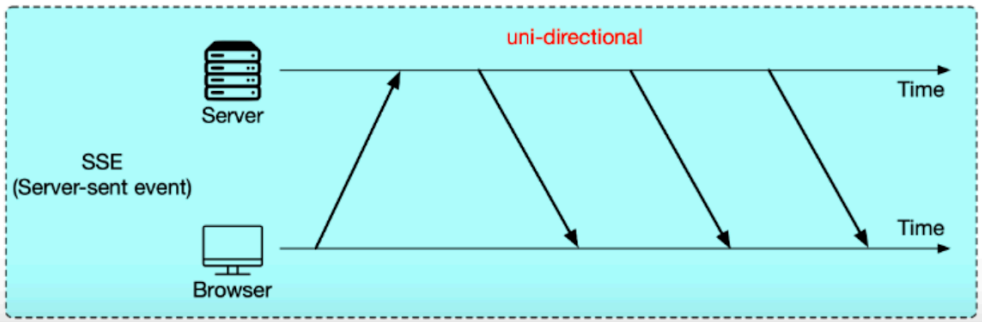


POLLING / WS / SSE









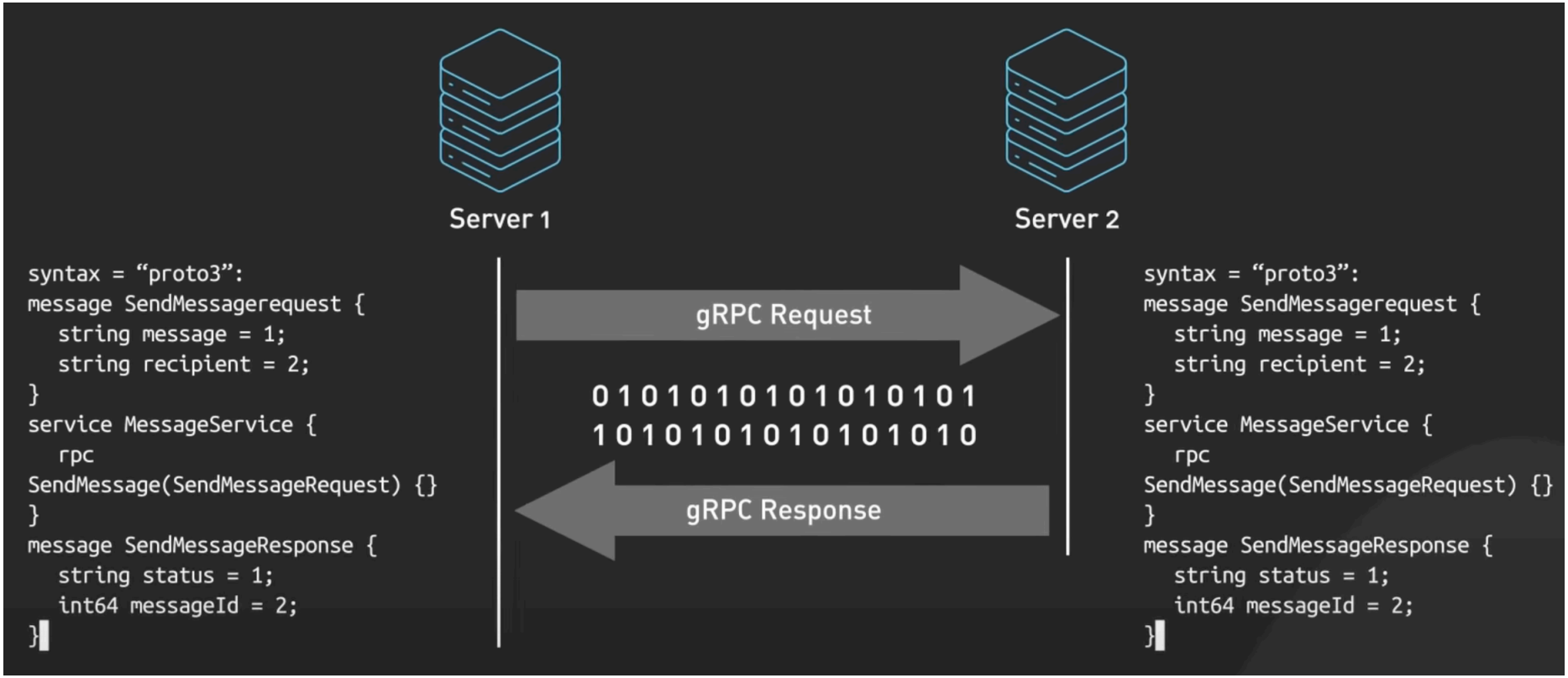
gRPC

GRPC (1)

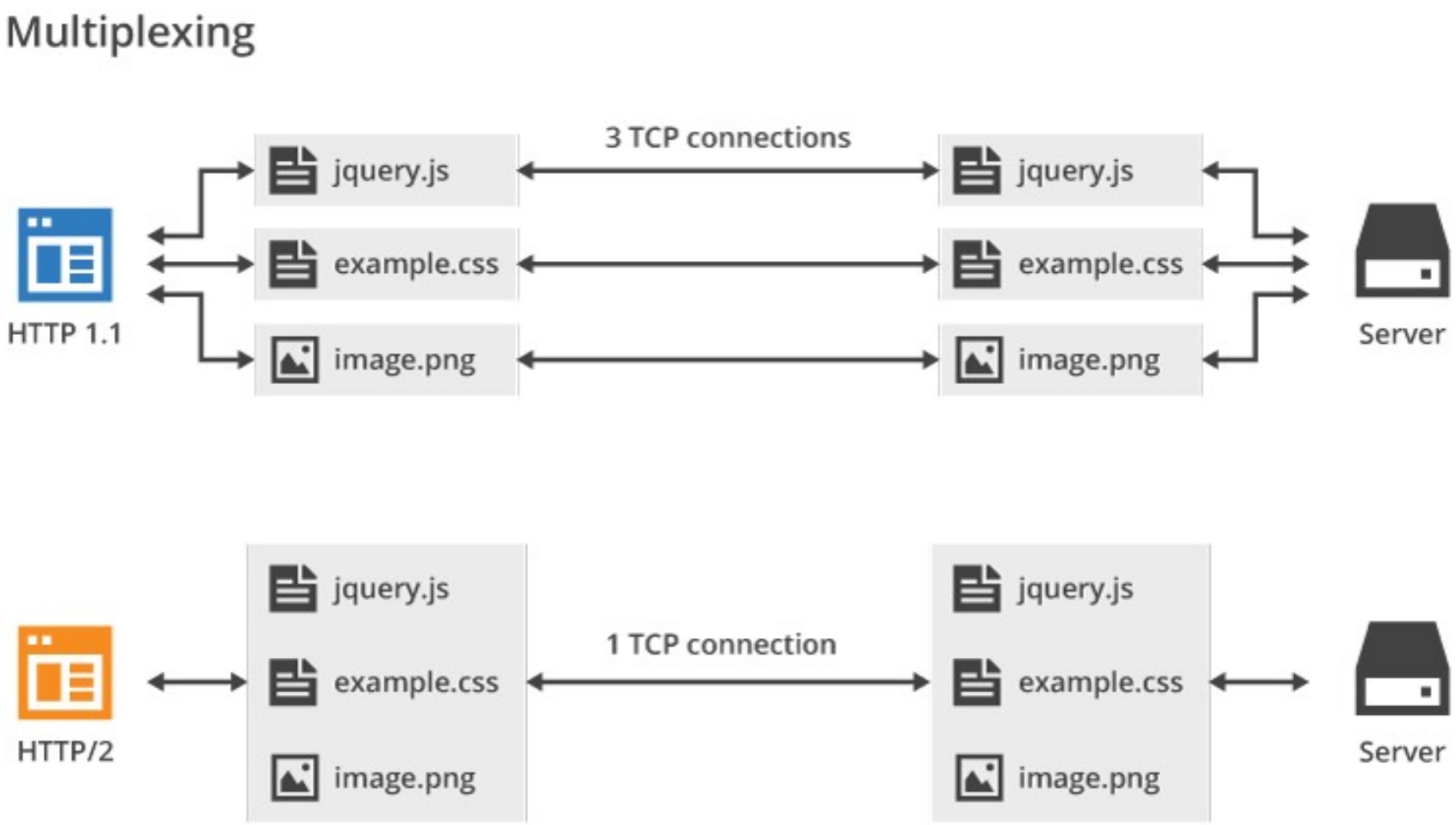
- The “RPC” in gRPC stands for “Remote Procedure Protocol.” RPC was first introduced in the late 1970s and 1980s, and it allows clients and servers to interact with one another as if they were both on the same machine.
- gRPC is an implementation of RPC that was developed and open-sourced by Google in 2015.
- gRPC use of Protocol Buffers (Protobuf) as its interface definition language (IDL) provides strong typing and facilitates code generation in multiple languages.

GRPC (2)

- Additionally, its use of HTTP/2 for transport improves network efficiency and supports real-time communication.



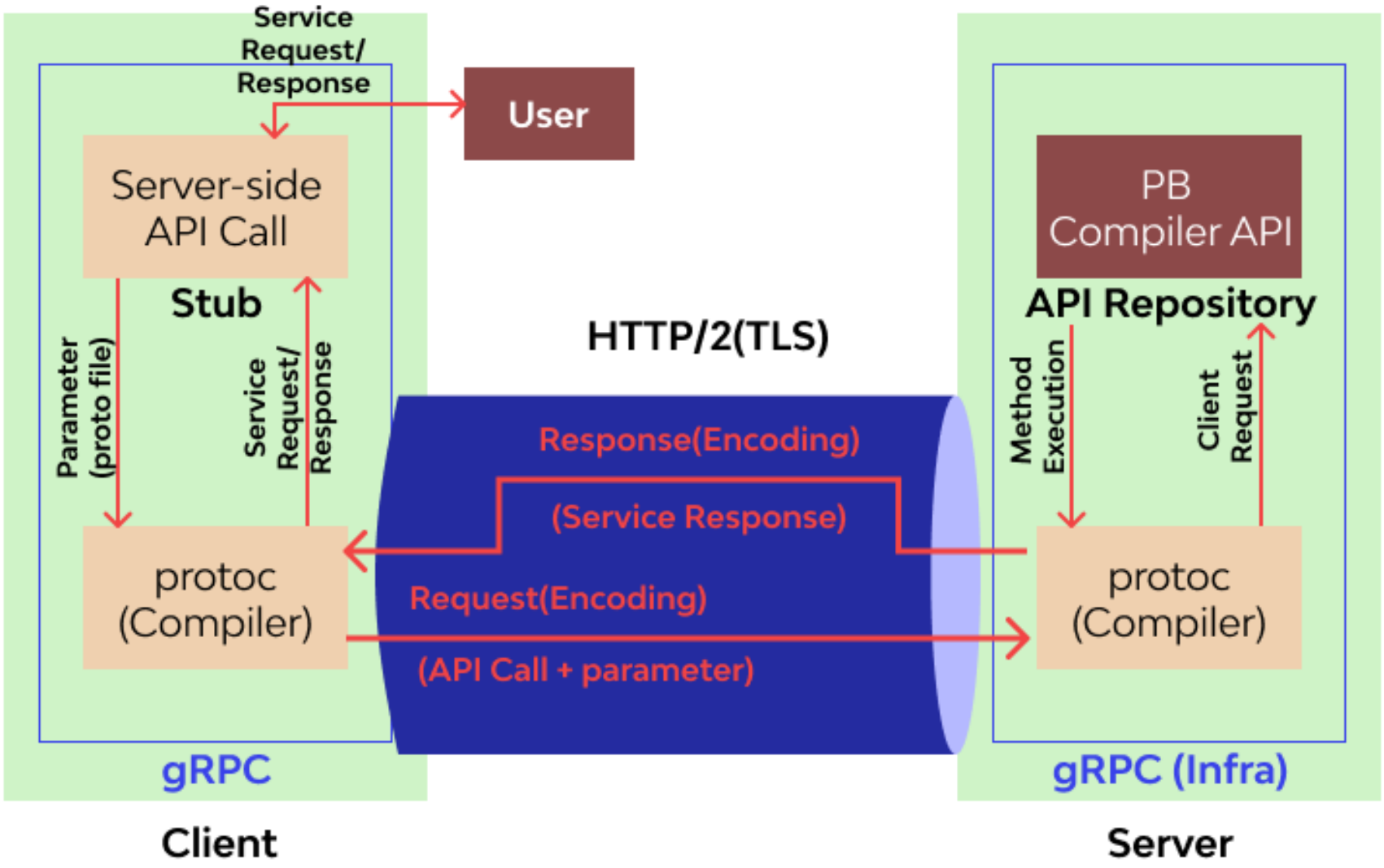
HTTP 1.1 VS. HTTP/2



PROTOCOL BUFFER

- Protocol buffers, or **Protobuf**, is Google's serialization/deserialization protocol that enables the easy definition of services and auto-generation of client libraries. gRPC uses this protocol as their Interface Definition Language (IDL) and serialization toolset.
- gRPC services and messages between clients and servers are defined in *proto files*. The Protobuf compiler (protoc) generates client and server code that loads the *.proto* file into the memory at runtime and uses the in-memory schema to serialize/deserialize the binary message.
- After code generation, each message is exchanged between the client and remote service.

GRPC ARCHITECTURE



ASSIGNMENT 7

- ให้แต่ละกลุ่มออกแบบ System Architecture ของ “ร้านก๋วยเตี๋ยว b&B (bit & Byte)” ว่าต้องประกอบไปด้วยระบบย่อย (Sub-System) อะไรบ้าง
- เลือกรูปแบบของ System Architecture (Client/Server, P2P etc.) และ Application Architecture (MVC, MVP etc.) โดยครอบคลุมตั้งแต่ UI จนถึง DBMS
- เลือกรูปแบบ API ที่ใช้ในการสื่อสารกันระหว่างแต่ละส่วน