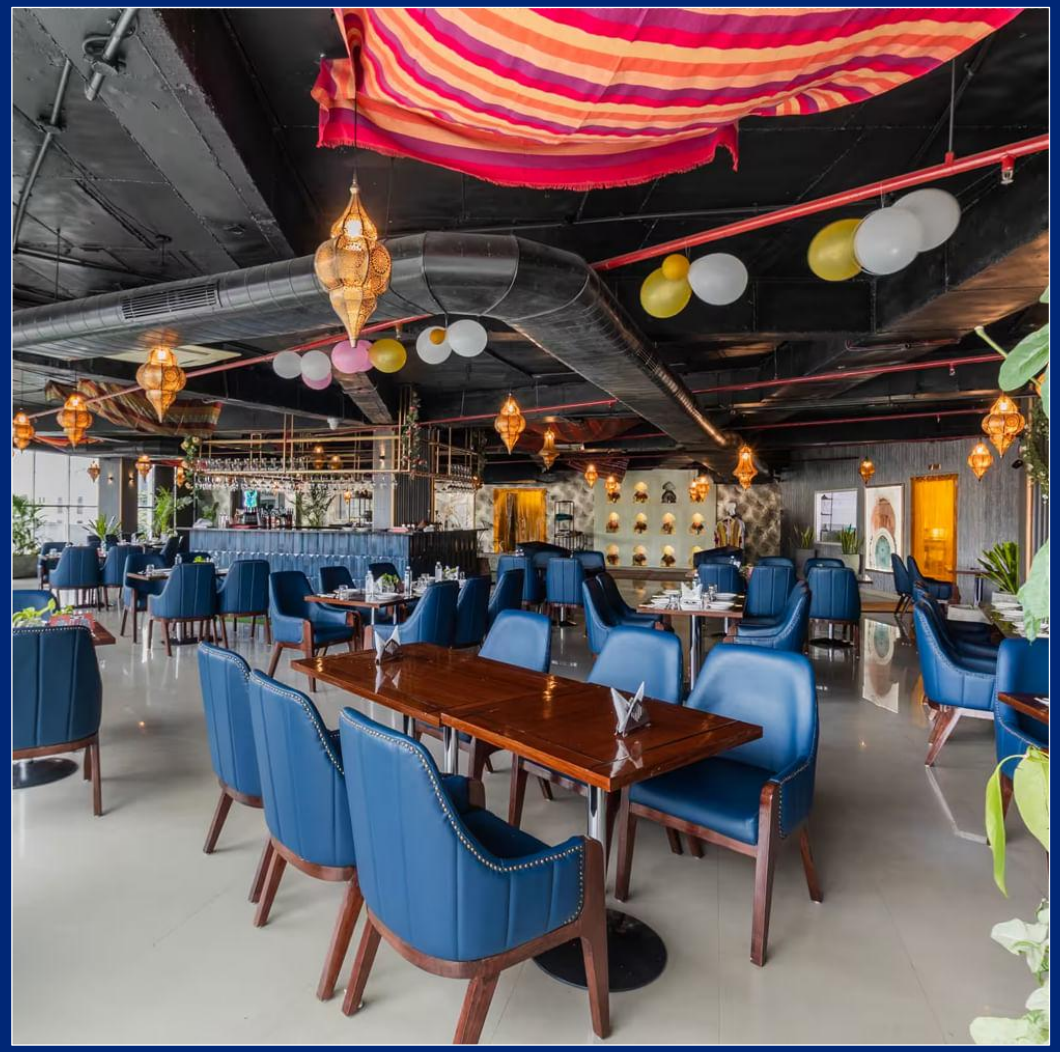
</> API

Language of Web

Real World Analogy

Restaurant



1. Customer → Client
2. Menu → API
3. Waiter → API Server (FastAPI)
4. Kitchen → Backend Logic (Server)
5. Food → Response

</>API

Application Programming Interface

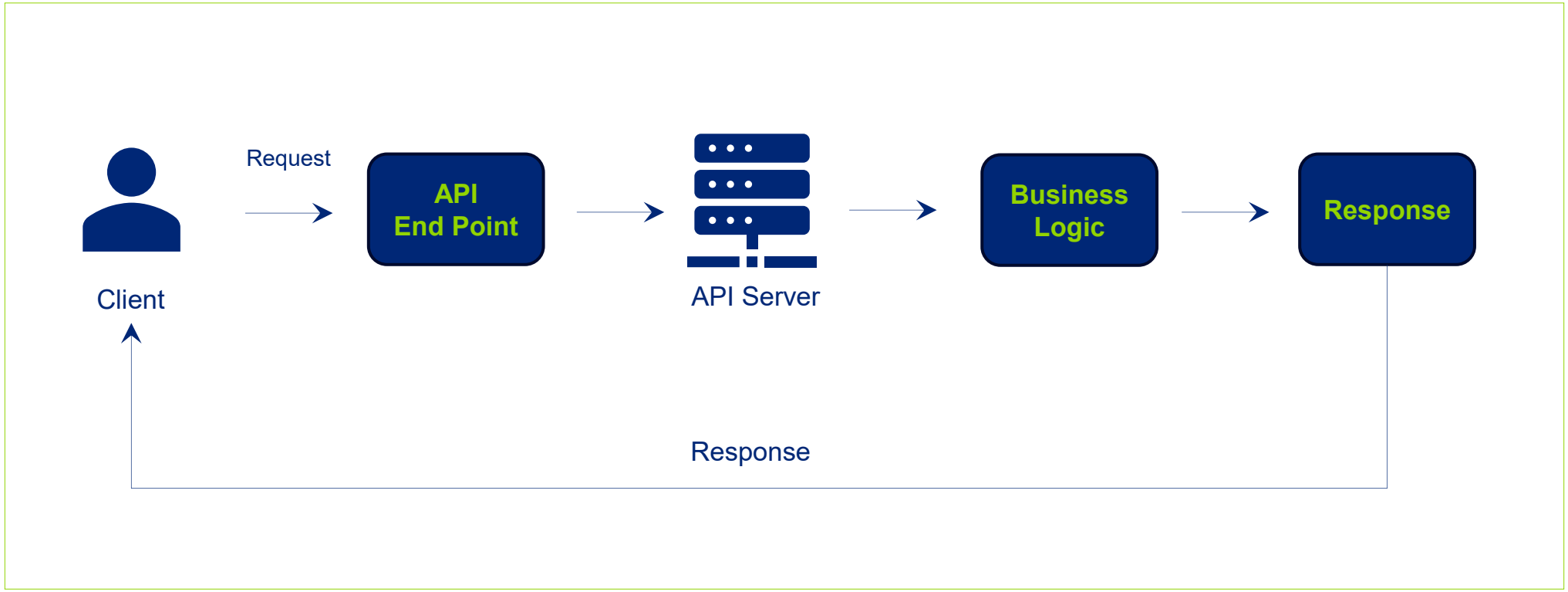
An API is simply a communication contract between **client** and **server.** And communication happens using **HTTP**

Client	Browser, Mobile App, Postman	→ Request
Server	FastAPI Application	→ Response

</>API

Application Programming Interface

- Client Browser, Mobile App, Postman → Request
- Server FastAPI Application → Response



HTTP

Hypertext Transfer Protocol

Language of Communication between client and server.

HTTP

Hypertext Transfer Protocol

- How to ask for data ?
- How to send data ?
- How to respond to errors ?

→ Protocol

HyperText Transfer Protocol

Sending data from one place to another

Text that contains links to other text

Eg:

- A web page that links to another page
- Clicking a link → jumps to another resource

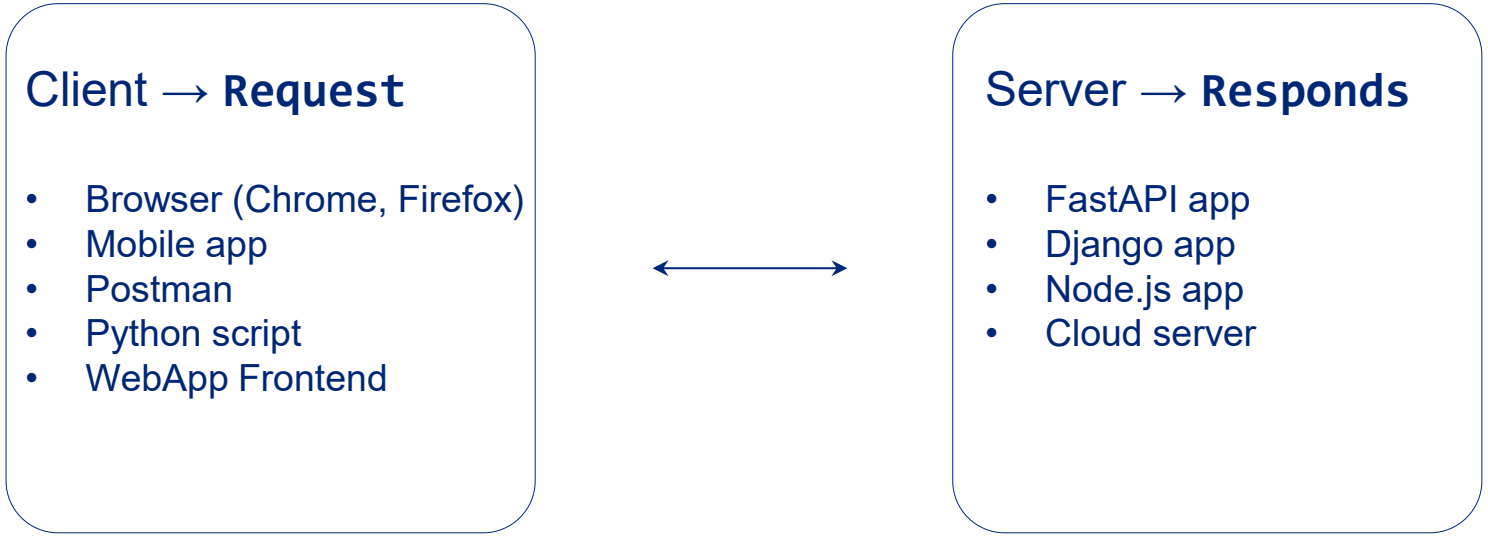
Rules

- Message format
- Order of communication
- Error handling
- Status codes

HTTP

Rulebook for transferring data between a client and a server over the web

Client – Server Model



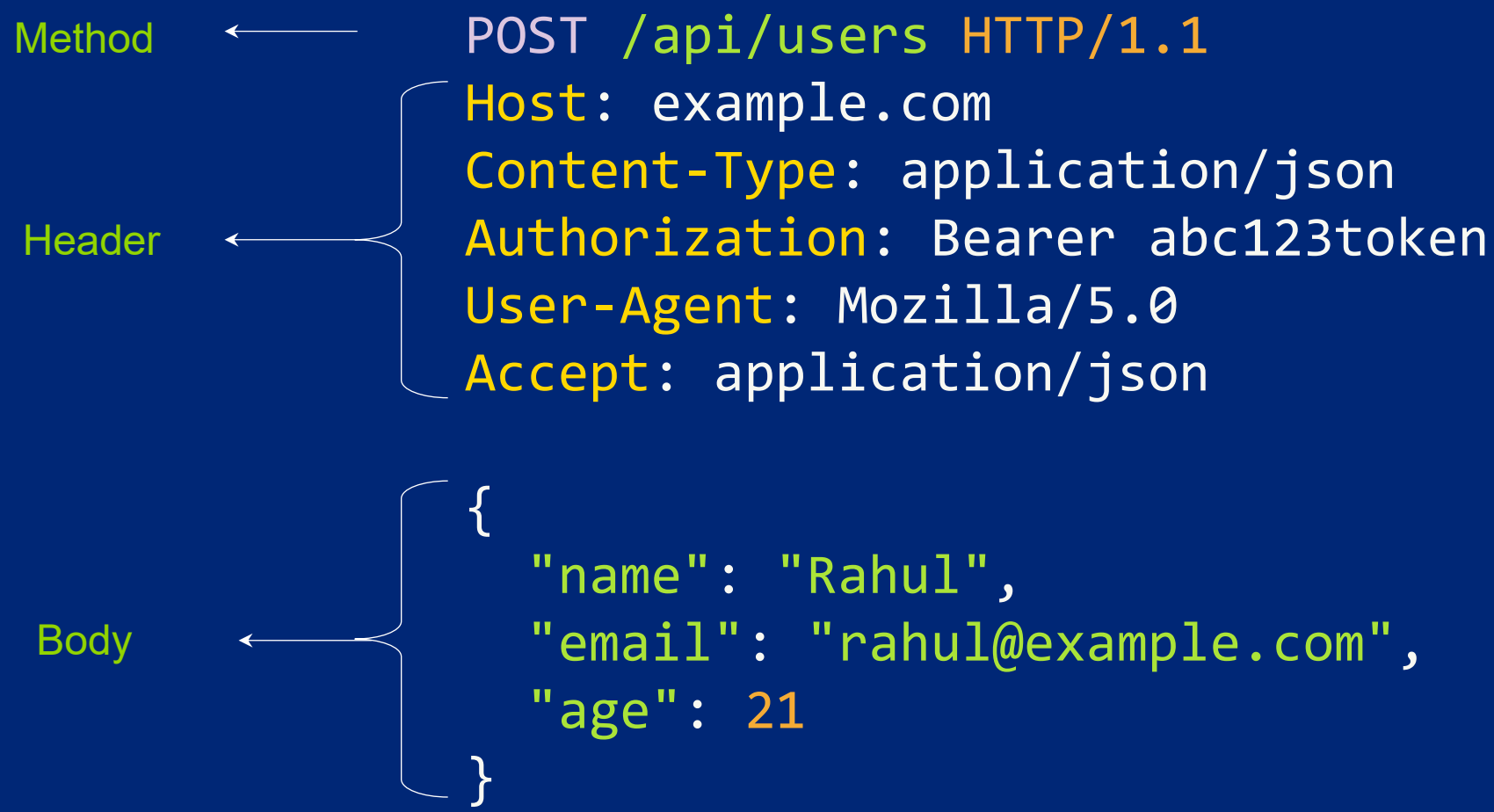
Notes:

- Client always initiates the communication
- Server never talks first

Next | HTTP Request

HTTP Request

HTTP Request



HTTP Request

Rulebook for transferring data between a client and a server over the web

A HTTP request has **4 major parts.**

HTTP Request	
Method	GET, POST, PUT, DELETE
URL	Address of resource
Headers	Metadata
Body	Data

HTTP

Rulebook for transferring data between a client and a server over the web

A HTTP request has **4 major parts.**

HTTP Request	
Method	**GET, POST, PUT, DELETE**
URL	Address of resource
Headers	Metadata
Body	Data

HTTP Method

Specific action a client want to perform on the server

Most common methods: (Usually maps to CRUD operations **C**reate **R**ead **U**pdate and **D**elete)

Method	Meaning
GET	Read data
POST	Create or Update data
PUT	Create or Replace data
DELETE	Remove data

HTTP

Rulebook for transferring data between a client and a server over the web

A HTTP request has **4 major parts.**

HTTP Request	
Method	GET, POST, PUT, DELETE
URL	**Address of resource**
Headers	Metadata
Body	Data

URL

Uniform Resource Locator/Identifier
→ Address or Location of the resource.

`http://example.com:8000/users/10?active=true`

Part	Meaning
`http://`	Schemes/Protocol
`example.com`	Unique name that identifies a specific website or Server
`8000`	Port
`/user/10`	Path
`?active=true`	Query

Schemes

URI scheme is the first part of the URL

Schemes	Full Form
http://	HyperText Transfer Protocol
https://	HyperText Transfer Protocol Secure
ftp://	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol.
rtsp://	Real Time Streaming Protocol
s3://	Amazon Simple Storage Service
gs://	Google Cloud Storage
mongodb://	Mongodb Wire Protocol
spotify://	Spotify app to play a specific track or playlist

URL

Uniform Resource Locator/Identifier
→ Address or Location of the resource.

`http://example.com:8000/users/10?active=true`

Part	Meaning
`http://`	Schemes/Protocol
`example.com`	Unique name that identifies a specific website or Server
`8000`	Port (default 80 for HTTP , 443 for HTTPS)
`/user/10`	Path
`?active=true`	Query

HTTP

Rulebook for transferring data between a client and a server over the web

A HTTP request has **4 major parts.**

HTTP Request	
Method	GET, POST, PUT, DELETE
URL	Address of resource
Headers	**Metadata**
Body	Data

</> Header

<Metadata> | Headers are **not data**, they are **instructions**

key : value

Headers provide extra information about the request.

- Content-Type
- Authorization
- Accept
- User-Agent

```
POST /api/users HTTP/1.1
Host: example.com
Content-Type: application/json
Authorization: Bearer abc123token
User-Agent: Mozilla/5.0
Accept: application/json

{
  "name": "Rahul",
  "email": "rahul@example.com",
  "age": 21
}
```

</> Header

<Metadata> | Headers are **not data**, they are **instructions**

Content-Type:

Question: How does the browser know whether the data is HTML, JSON, an image, or something else?
Answer: From the Content-Type header.

Example:

- `Content-Type: text/html; charset=UTF-8` → This means: 'The body is an HTML page.'
- `Content-Type: application/json` → This means: 'The body is JSON data, usually used in APIs.'
- `Content-Type: image/png` → This means: 'The body is a PNG image.'

</> Header

<Metadata> | Headers are **not data**, they are **instructions**

Authorization Header:

Question: How does a server know who you are, and whether you are allowed to access something?.

Example:

- `Authorization: Basic dXNlcjpwYXNzd29yZA==`
  - Basic authentication. → base64-encoded username : password
- `Authorization: Bearer <token>`

</> Header

<Metadata> | Headers are **not data**, they are **instructions**

Accept Header:

Question: What if a client wants data in a particular format, like JSON instead of HTML?

Example:

- `Accept: text/html`
  - I want HTML
- `Accept: application/json`
  - Prefer JSON data

</> Header

<Metadata> | Headers are **not data**, they are **instructions**

User-Agent Header:

Question: How can a website know which browser or device you are using?

Answer:

- It identifies the client application, such as Chrome, Firefox, Postman, or curl.
- OS details like Windows, Android, or iOS

Example:

- `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/122.0 Safari/537.36`
- `User-Agent: PostmanRuntime/7.39.0`
- `User-Agent: curl/8.1.0`

</> Header

<Metadata> | Headers are **not data**, they are **instructions**

Headers provide extra information about the request.

- Content-Type → What kind of data is in the body (content)
- Authorization → Carries login or token information to prove identity
- Accept → Tells what data formats the client can accept
- User-Agent → Tells which browser or tool is making the request.

HTTP

Rulebook for transferring data between a client and a server over the web

A HTTP request has **4 major parts.**

HTTP Request	
Method	GET, POST, PUT, DELETE
URL	Address of resource
Headers	Metadata
Body	**Data**

{...} Body

Data sent to the server.

Body carries main data you are sending to the server like form-data, JSON, files etc.

Usually present in methods like

- POST
- PUT
- PATCH

```json
{
“name”: “Rahul”
“age”: 21
}
```

{...} Body

Data sent to the server.

Not all requests have a body.

GET → do not have a body; they mostly ask for data.

POST, PUT and PATCH → They have a body, because they send data to be stored or updated on the server.

Simple body examples

- Form data (like a login form):
  - Headers say: `Content-Type: application/x-www-form-urlencoded`
  - Body might be: `username=ram&password=12345`

- JSON data (for an API):
  - Headers say: `Content-Type: application/json`
  - Body might be: `{"name": "Rahul", "age": 20}`

- File upload:
  - Headers say something like: `Content-Type: multipart/form-data`
  - Body contains multiple parts: `text fields plus the file bytes.`

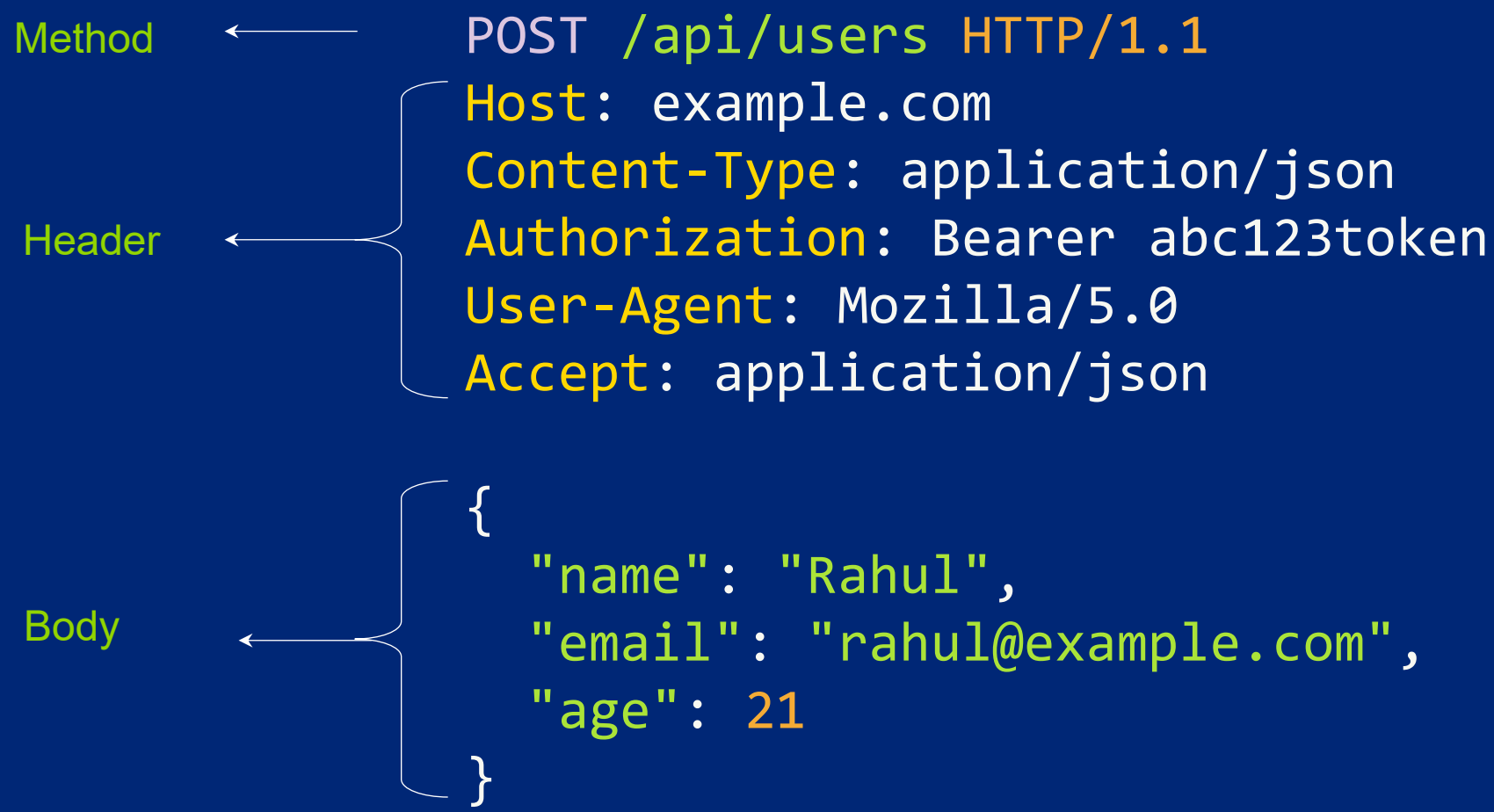
HTTP

Rulebook for transferring data between a client and a server over the web

A HTTP request has **4 major parts.**

HTTP Request	
Method	GET, POST, PUT, DELETE
URL	Address of resource
Headers	Metadata
Body	Data

HTTP Request

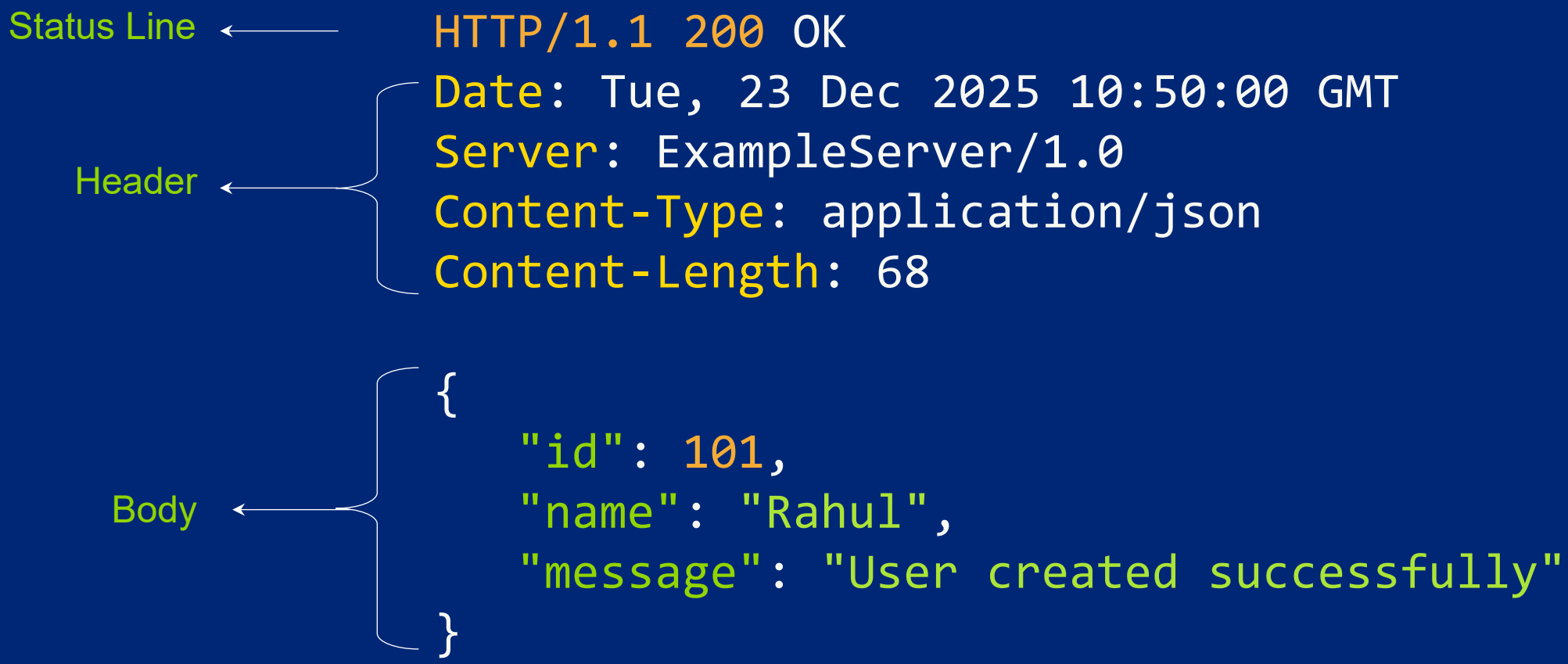


Next | HTTP Response

HTTP Response

Message sent by the server back to the client

HTTP Response



HTTP Response

Transferring data from server back to client

A HTTP response has **3 main parts.**

HTTP Response	
Status Line	Status of the request
Headers	Metadata
Body	Data

HTTP Response

Transferring data from server back to client

A HTTP response has **3 main parts.**

HTTP Response	
Status Line	**Status of the request**
Headers	Metadata
Body	Data

HTTP Response

Server replies with an HTTP response

```
HTTP/1.1 200 OK
```

Status Line = `HTTP version + status code + short message`

Code	Short Message
200	OK → Success
201	Created → New resource created
400	Bad Request → Client sent something invalid
401	Unauthorized → Needs Authentication
403	Forbidden → Not Allowed
404	Not Found → Resource not found
500	Internal Server Error → Server crashed or failed

- 1xx – Informational
- 2xx – Success
- 3xx – Redirection
- 4xx – Client error
- 5xx – Server error

HTTP Response

Transferring data from server back to client

A HTTP response has **3 main parts.**

HTTP Response	
Status Line	Status of the request
Headers	**Metadata**
Body	Data

HTTP Response

Transferring data from server back to client

key : value

Headers

```
Date: Tue, 23 Dec 2025 10:50:00 GMT
Server: ExampleServer/1.0
Content-Type: application/json
Content-Length: 68
```

HTTP Response

Transferring data from server back to client

key : value

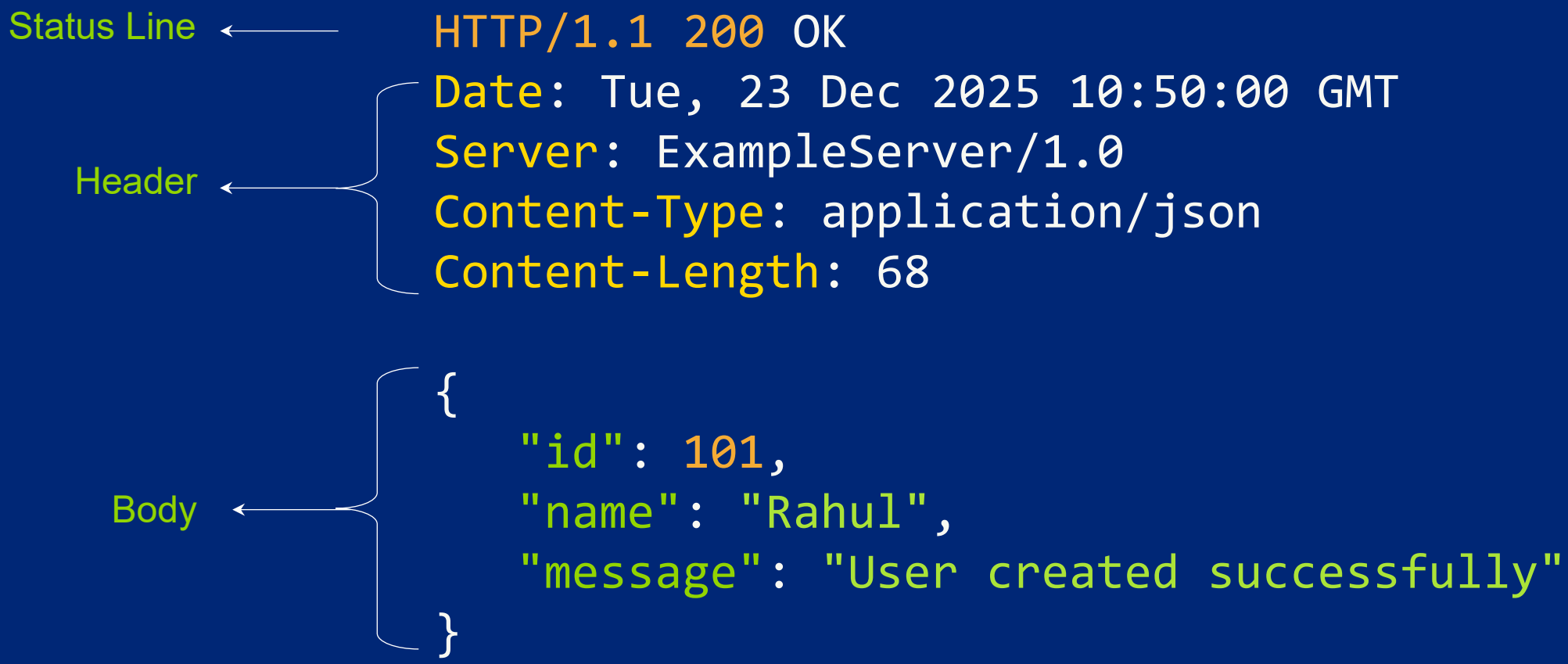
Body

```
{
   "id": 101,
   "name": "Rahul",
   "message": "User created successfully"
}
```

This is the actual content the server is sending back.

- For a web page, the body might be HTML.
- For an API, often JSON or XML.
- It can also be binary data like images, PDFs, videos, etc.

HTTP Response



Next |  Knowledge Check

Knowledge Check

Knowledge Check

Is HTTP only for browsers ?

A. Yes

B. No

Answer: No
APIs, mobile apps, IoT, Postman etc all uses HTTP

Knowledge Check

What best explains the relationship between HyperText and HTTP

A. HyperText encrypts HTTP communication

B. HTTP is used only for transferring images

C. HTTP transfers HyperText documents and send data between client and server

D. HyperText and HTTP are unrelated concepts

Answer: C
HTTP was originally designed to transfer **HyperText (HTML)** documents.
Today, it also transfers JSON, images, and files, but the **navigation idea of HyperText remains central**.

Knowledge Check

When a user clicks a clickable link on a webpage, what happens internally?

A. The server initiates a request to the browser

B. The browser edits the page locally

C. The browser (client) sends an HTTP request to the server

D. The database directly returns data

Answer: C
HyperText links are clickable text that cause the **client (browser)** to send an **HTTP request** to fetch another resource.

Knowledge Check

Which Statement correctly describes a HTTP request ?

A. It is sent only by the server

B. It contains only the URL

C. It is the same as an HTTP response

D. It includes method, URL, headers, and an optional body

Answer: D
An HTTP request tells the server:
What action to perform (method), **Which resource** (URL), **How to interpret data** (headers), **Actual data** (body, if present)

Knowledge Check

Which option correctly matches **HTTP methods** with their purpose?

A. GET → Create data, POST → Read data

B. GET → Read data, POST → Send data to server

C. DELETE → Fetch data, PUT → Read data

D. POST → Delete data, GET → Update data

Answer: B

- **GET** is used to **retrieve data**
- **POST** is used to **send data** (like JSON or form input) to the server

Knowledge Check

Which part of an HTTP request line conveys the operation the client wants to perform?

A. Header Field

B. HTTP version

C. Method

D. URL

Answer: C

- The method specifies the desired action on the target resource, such as GET, POST, PUT, or DELETE

Knowledge Check

What does the status code in an HTTP response’s status line primarily indicate?

A. The outcome of request processing

B. The server software version

C. The size of the response body

D. Whether the request had a body

Answer: A

- The status code conveys the server’s assessment of the request outcome across classes 1xx (informational), 2xx (success), 3xx (redirection), 4xx (client error), and 5xx (server error)

Hands on

HTTP Request & HTTP Response

Recap

API → Communication Contract between a **client** and a **server**
HTTP → Defines how **communication** happens

HTTP Request

```
POST /api/users HTTP/1.1
Host: example.com
Content-Type: application/json
Authorization: Bearer abc123token
User-Agent: Mozilla/5.0
Accept: application/json

{
  "name": "Rahul",
  "email": "rahul@example.com",
  "age": 21
}
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Tue, 23 Dec 2025 10:50:00 GMT
Server: ExampleServer/1.0
Content-Type: application/json
Content-Length: 68

{
   "id": 101,
   "name": "Rahul",
   “email": "rahul@example.com",
   "age": 21,
   "message": "User created successfully"
}
```

Mockup endpoint

https://69578e1cf7ea690182d25b1b.mockapi.io/myendpoint/countries

This API return list of countries and cities.

httpie

Make sure we added httpie library in poetry

```
poetry add httpie
or
pip install httpie
```

- Step – 1 : Active your **poetry** virtual environment

- Step – 2: Usage

Note: Default method is GET

Example:

```
http GET https://69578e1cf7ea690182d25b1b.mockapi.io/myendpoint/countries
```

httpie

flags

Output Options	Separator	Example
Only the final response is shown	`--all`	`http –all <URL>`
Print the whole request as well as the response	`--verbose, -v`	`http -v <URL>`
Print only response body	`--body, -b`	`http –b <URL>`
Print specifying what the output should contain → 'H' request headers → 'B' request body → 'h' response headers → 'b' response body → 'm' response metadata	`--print, -p WHAT`	`HTTP –p H <URL>`
Note: Type **`http –help`**, *for more output options*		

httpie

Seperators

Request Item	Separator	Example
URL query parameter	==	`id==5`
HTTP headers	:	`myheader:value`
Body or Data fields into JSON object	=	`city=paris country=france`
Non-String JSON fields	:=	`amount:=42 colours:=[“red”, “green”, “blue”]`
Form file fields	@	`cv@~/documents/cv.pdf;type=application/pdf`
Flat files (.txt, .csv)	=@	`data=@documents/data.txt`
Raw JSON file	:=@	`package:=@./package.json`

https://beeceptor.com

httpie

Seperators

Request Item	Separator	Example
URL query parameter	==	id==5
HTTP headers	:	myheader:value
Body or Data fields into JSON object	=	city=paris country=france
Non-String JSON fields	:=	amount:=42 colours:=[“red”, “green”, “blue”]
Form file fields	@	cv@~/documents/cv.pdf;type=application/pdf
Flat files (.txt, .csv)	=@	data=@documents/data.txt
Raw JSON file	:=@	package:=@./package.json

Next

FastAPI

Asynchronous Server Gateway Interface