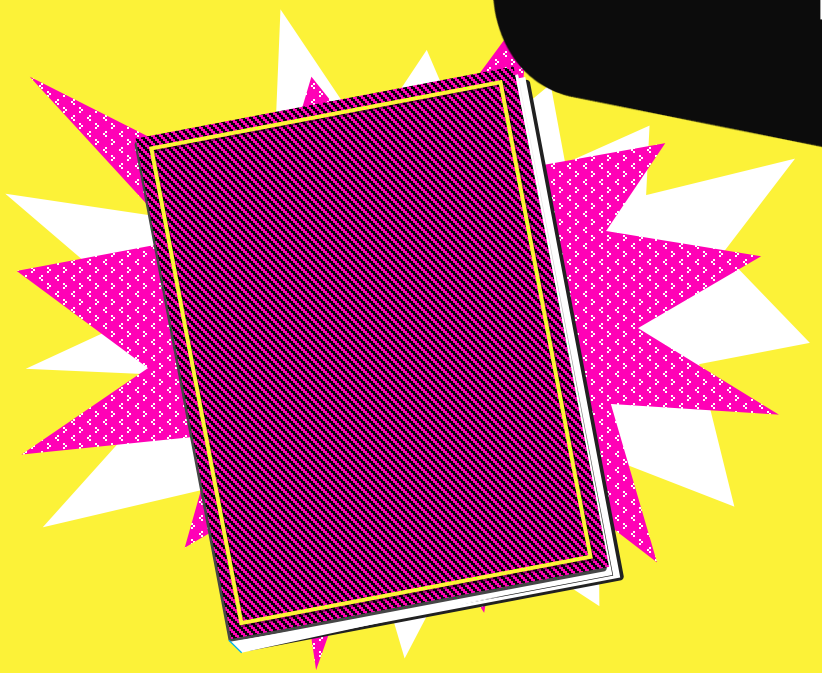
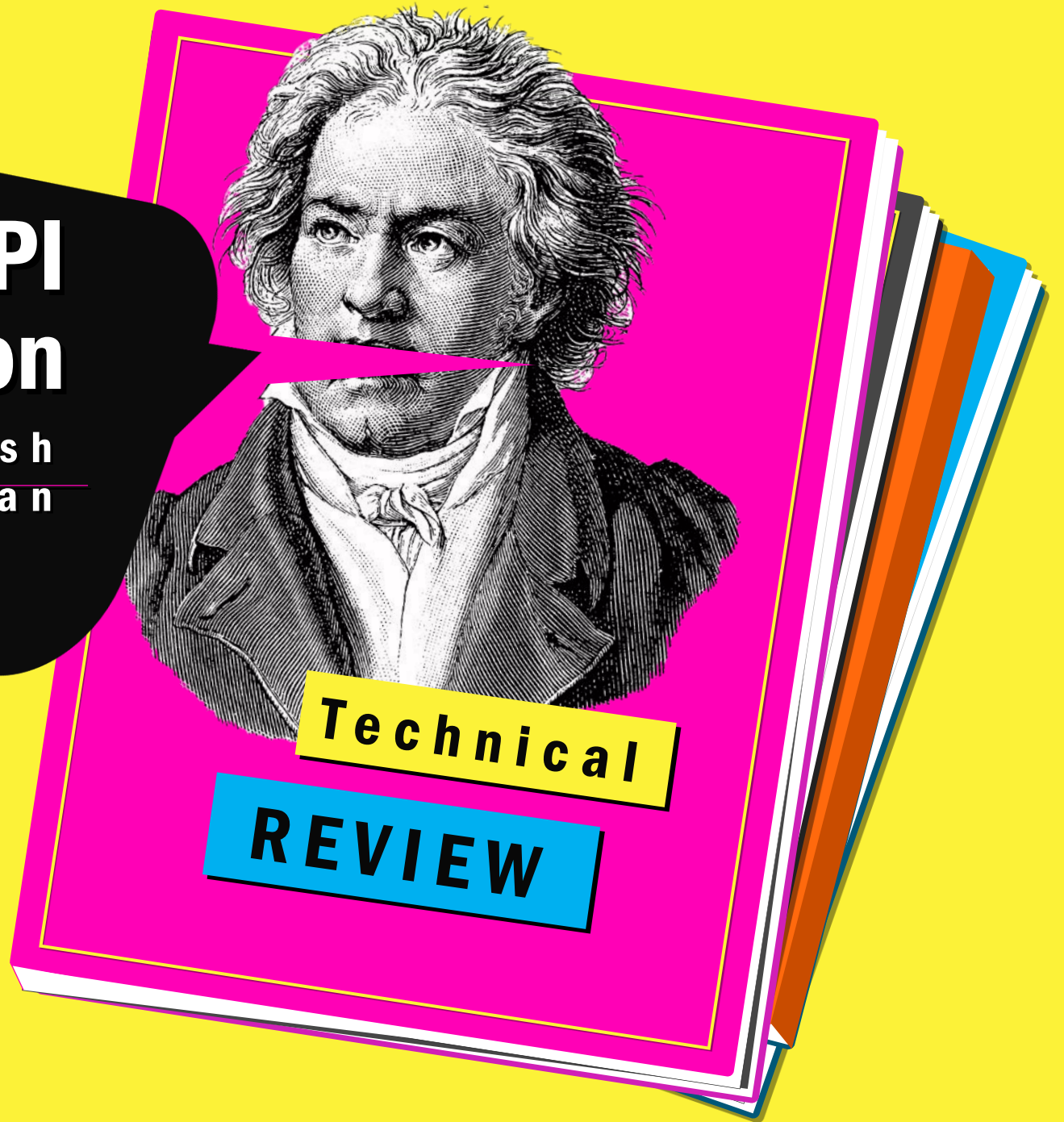
API Introduction

A Sailesh Padmanabhan



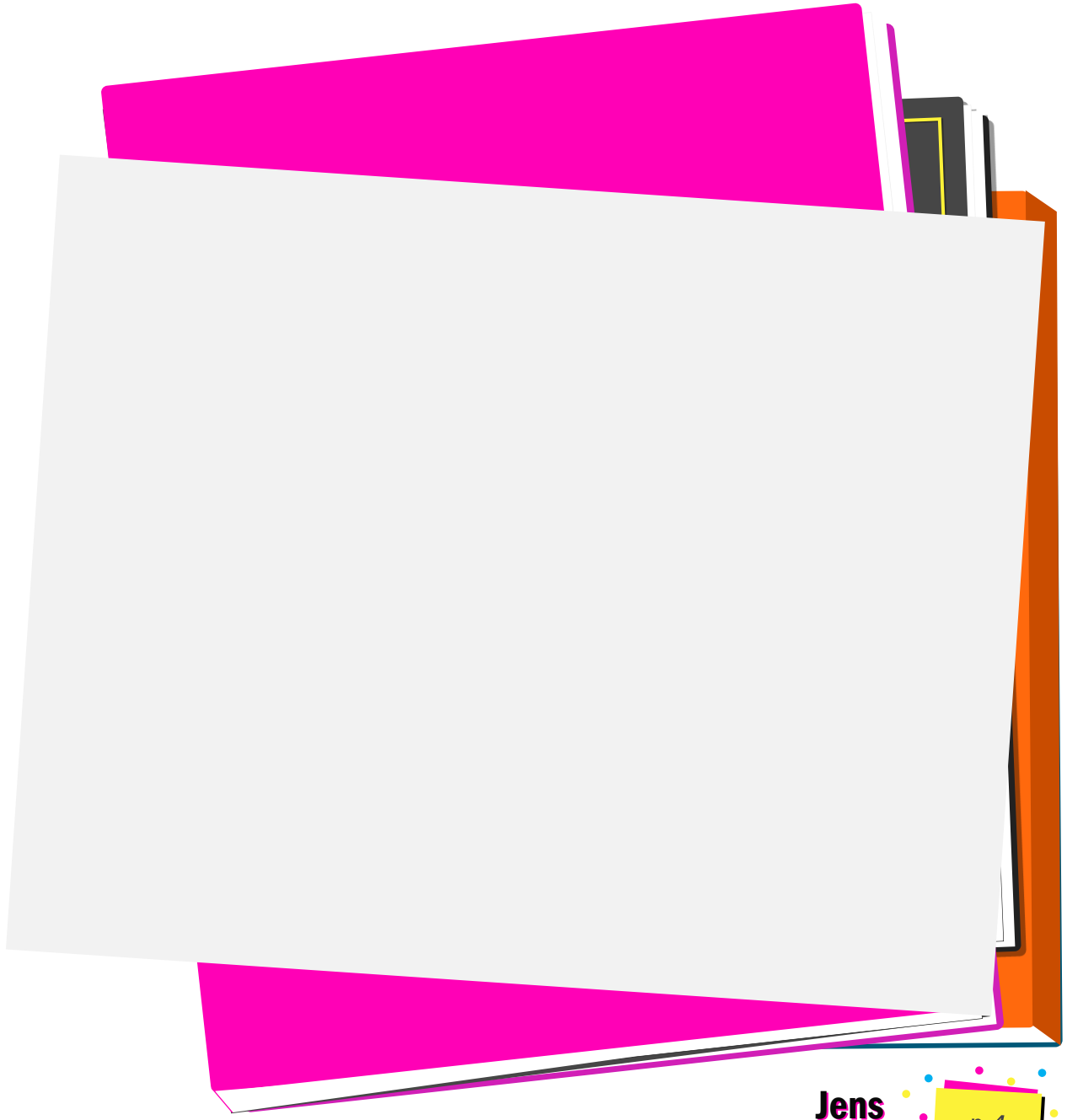


Where the story begins

What is API

- API stands for application programming interface.
- An API is basically a set of definitions and protocols that enable two applications to talk to one another.
- APIs are the connectors that enable most of the communications between the web applications, or apps, that we use today.
- For example, if you use apps on your phone to check the weather, or send a message to friends, or to find the nearest cafe, you'll be using APIs without even knowing it.

- APIs work by communicating with, and exchanging data between, other systems.
- They act as the messengers between us, the users, and the backend systems, allowing us to retrieve the data we want, when we want it.



API Basics

- In software terms, APIs are a set of functions and procedures that allow application developers to create apps that use the data and features of other services and applications. APIs give developers a toolset that they can use to create their apps.
- There are three main types of APIs:
- Open APIs, which are publicly available with minimal restrictions to access them.
- Partner APIs, which need specific access rights to be able to use them and are typically exposed via an API developer portal.
- Internal APIs, which are hidden from external users and only exposed by internal systems. Typically, these are exposed via an internal API developer portal to enable authorization to the right set of APIs.
- There are also two different ways that APIs can be implemented:
- Simple APIs, which are available separately.
- Composite APIs, where multiple data or service APIs are combined together, allowing developers to access several APIs at a time. For example, you'd implement composite APIs in a microservices architecture, where a user needs information from several endpoints to perform a single task, like the navigation app example we described previously.

API Example

- Let’s take a familiar example. Say you’re going to stay at a hotel, but you’ve not been there before, so you want to use a navigation app to get there. You input some details about the journey into the app, like the destination address, whether you’re going by car or by public transport, and any route preferences.
- Now, for the app to provide you with a complete navigation picture, it will check several different services, such as GPS data, weather information, public transport timetables, route delays, amenities, and so on. The navigation app will interact with all of these services by using APIs.

Types of API protocols

- There are three main types of API protocols:

- **REST.** REST is short for Representational State Transfer and is a web services API. It provides a uniform interface, where a client and server communicate with one another via Hypertext Transfer Protocol (HTTP), by using Uniform Resource Identifiers (URIs), the common Create, Read, Update, and Delete (CRUD) operations, and most often JavaScript Object Notation (JSON) conventions for data exchange.
- **SOAP.** SOAP is short for Simple Object Access Protocol and is another type of web services API. SOAP APIs have been used since the 1990s, but the protocol is stricter and more heavyweight than REST, so it isn't used as much in modern API development.
- **RPC.** RPC is short for Remote Procedural Call and is the oldest and simplest type of API protocol. RPC is a request-response protocol, where a client sends a request to a remote server to execute a specific procedure, and then the client receives a response back. However, RPC APIs are much more difficult to maintain and update than REST APIs, so again RPC APIs aren't used as much in modern API development.

- REST APIs have become the preferred standard for building apps that communicate over a network. However, there is a new data query and manipulation language for APIs that is rapidly gaining ground called GraphQL. GraphQL provides a different approach to API development, where the client, or app developer, has much greater control over what data is returned.

API Basics :- Endpoint

- An API endpoint is where an API receives requests. For most services, these endpoints are URLs, just like the ones you use to navigate to a website.
- **How API endpoints work**
- Let’s take a closer look at how API endpoints work. Here’s an example of an endpoint URL:

- https://api.github.com/repos/torvalds/linux

- This endpoint belongs to the GitHub REST API and returns information about a repository as a JSON object.

- API endpoints have a base path that's usually a dedicated subdomain, like cdn.contentful.com, graphql.contentful.com, or https://collectionapi.metmuseum.org/. They can also be presented as paths within the main site, like https://mandrillapp.com/api but it's rarer.

- Then, to complete the endpoint, you add a suffix to that base URL that specifies what you want. In the example above, that's the `repos` in api.github.com/repos.

- From an API design perspective, this lets developers create and maintain logical groupings.

- Some APIs have a hierarchical structure, using path values to create a parent/child relationship.

API Methods

- GET
- The HTTP GET method requests a representation of the specified resource. Requests using GET should only be used to request data (they shouldn't include data).
- HEAD
- The HTTP HEAD method requests the headers that would be returned if the HEAD request's URL was instead requested with the HTTP GET method. For example, if a URL might produce a large download, a HEAD request could read its Content-Length header to check the filesize without actually downloading the file.
- If the response to a HEAD request shows that a cached URL response is now outdated, the cached copy is invalidated even if no GET request was made.

- HTTP Options
- The HTTP OPTIONS method requests permitted communication options for a given URL or server. A client can specify a URL with this method, or an asterisk (*) to refer to the entire server
- HTTP PATCH
- The HTTP PATCH request method applies partial modifications to a resource.

- PATCH is somewhat analogous to the "update" concept found in CRUD (in general, HTTP is different than CRUD, and the two should not be confused).

- A PATCH request is considered a set of instructions on how to modify a resource. Contrast this with PUT; which is a complete representation of a resource.

- The HTTP TRACE method performs a message loop-back test along the path to the target resource, providing a useful debugging mechanism.

- The final recipient of the request should reflect the message received, excluding some fields described below, back to the client as the message body of a 200 (OK) response with a Content-Type of message/http. The final recipient is either the origin server or the first server to receive a Max-Forwards value of 0 in the request.

- HTTP PUT
- The HTTP PUT request method creates a new resource or replaces a representation of the target resource with the request payload

- POST
- The HTTP POST method sends data to the server. The type of the body of the request is indicated by the Content-Type header

- API Parameters are options that can be passed with the endpoint to influence the response. In GET requests, they're found in strings at the end of the API URL path. In POST requests, they're found in the POST body.
- Types of API Parameters
- Query String Parameters
- Path Parameters
- Request Body Parameters
- Header Parameters

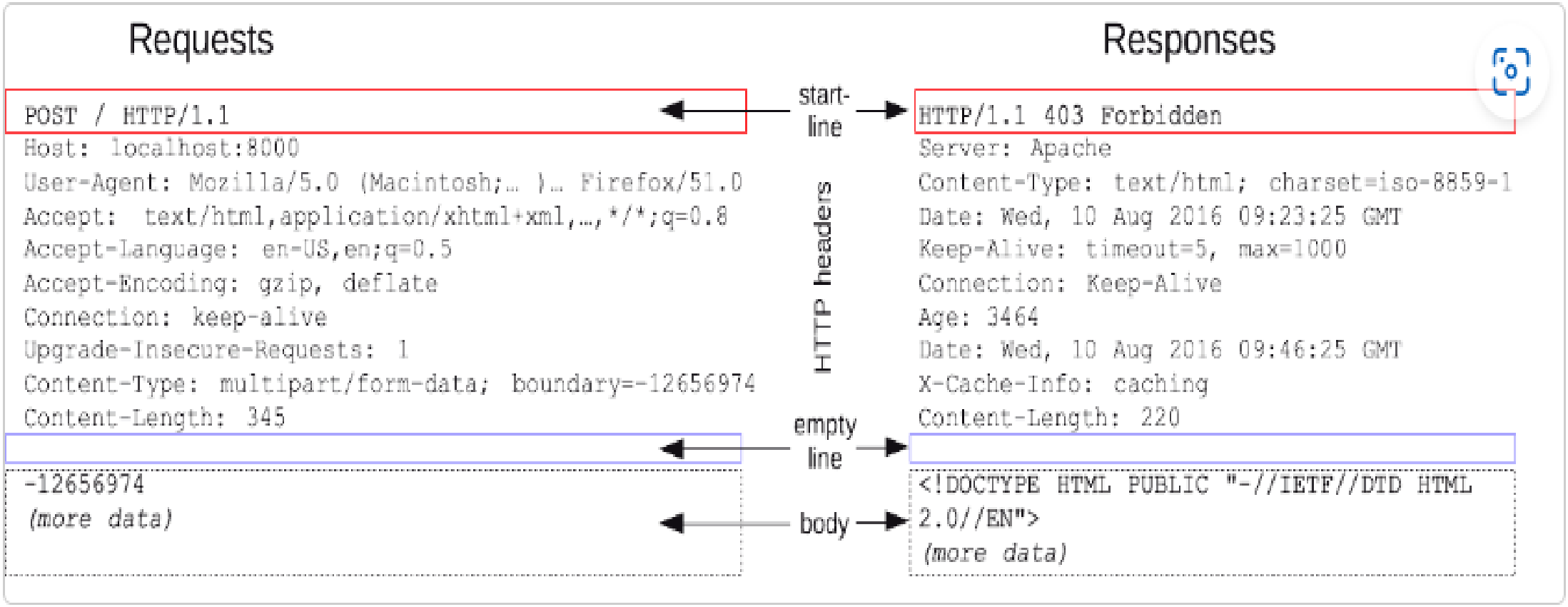
- The difference between PUT and POST is that PUT is idempotent: calling it once or several times successively has the same effect (that is no side effect), where successive identical POST may have additional effects, like passing an order several times.

Request and Response Format

- HTTP requests, and responses, share similar structure and are composed of:

1. A *start-line* describing the requests to be implemented, or its status of whether successful or a failure. This start-line is always a single line.
2. An optional set of *HTTP headers* specifying the request, or describing the body included in the message.
3. A blank line indicating all meta-information for the request has been sent.
4. An optional *body* containing data associated with the request (like content of an HTML form), or the document associated with a response. The presence of the body and its size is specified by the start-line and HTTP headers.

- The start-line and HTTP headers of the HTTP message are collectively known as the *head* of the requests, whereas its payload is known as the *body*.


Requests
Responses
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;…)… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345
-12656974
(more data)
start-line
HTTP headers
empty line
body
HTTP/1.1 403 Forbidden
Server: Apache
Content-Type: text/html; charset=iso-8859-1
Date: Wed, 10 Aug 2016 09:23:25 GMT
Keep-Alive: timeout=5, max=1000
Connection: Keep-Alive
Age: 3464
Date: Wed, 10 Aug 2016 09:46:25 GMT
X-Cache-Info: caching
Content-Length: 220
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
(more data)

- HTTP Requests
- Start line
- HTTP requests are messages sent by the client to initiate an action on the server. Their start-line contain three elements:

- An HTTP method, a verb (like GET, PUT or POST) or a noun (like HEAD or OPTIONS), that describes the action to be performed. For example, GET indicates that a resource should be fetched or POST means that data is pushed to the server (creating or modifying a resource, or generating a temporary document to send back).
- The request target, usually a URL, or the absolute path of the protocol, port, and domain are usually characterized by the request context. The format of this request target varies between different HTTP methods. It can be
- An absolute path, ultimately followed by a '?' and query string. This is the most common form, known as the origin form, and is used with GET, POST, HEAD, and OPTIONS methods.
- POST / HTTP/1.1
- GET /background.png HTTP/1.0
- HEAD /test.html?query=alibaba HTTP/1.1
- OPTIONS /anypage.html HTTP/1.0

- A complete URL, known as the absolute form, is mostly used with GET when connected to a proxy. GET https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages HTTP/1.1
- The authority component of a URL, consisting of the domain name and optionally the port (prefixed by a ':'), is called the authority form. It is only used with CONNECT when setting up an HTTP tunnel. CONNECT developer.mozilla.org:80 HTTP/1.1
- The asterisk form, a simple asterisk ('*') is used with OPTIONS, representing the server as a whole. OPTIONS * HTTP/1.1
- The HTTP version, which defines the structure of the remaining message, acting as an indicator of the expected version to use for the response.

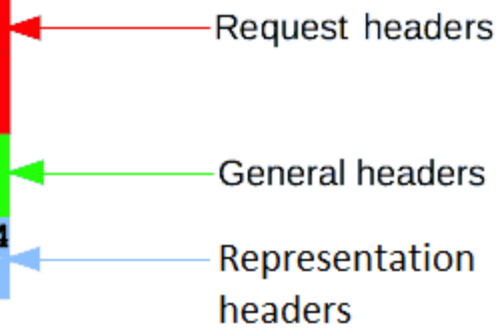
- HTTP headers from a request follow the same basic structure of an HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consists of one single line, which can be quite long.

- Many different headers can appear in requests. They can be divided in several groups:

- General headers, like Via, apply to the message as a whole.
- Request headers, like User-Agent or Accept, modify the request by specifying it further (like Accept-Language), by giving context (like Referer), or by conditionally restricting it (like If-None).
- Representation headers like Content-Type that describe the original format of the message data and any encoding applied (only present if the message has a body).

```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```



- Body
- The final part of the request is its body. Not all requests have one: requests fetching resources, like GET, HEAD, DELETE, or OPTIONS, usually don't need one. Some requests send data to the server in order to update it: as often the case with POST requests (containing HTML form data).

- Bodies can be broadly divided into two categories:

- Single-resource bodies, consisting of one single file, defined by the two headers: Content-Type and Content-Length.
- Multiple-resource bodies, consisting of a multipart body, each containing a different bit of information. This is typically associated with HTML Forms.

HTTP Responses

- Status line
- The start line of an HTTP response, called the status line, contains the following information:
- The protocol version, usually HTTP/1.1.
- A status code, indicating success or failure of the request. Common status codes are 200, 404, or 302
- A status text. A brief, purely informational, textual description of the status code to help a human understand the HTTP message.
- A typical status line looks like: HTTP/1.1 404 Not Found.

- Headers
- HTTP headers for responses follow the same structure as any other header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the type of the header. The whole header, including its value, presents as a single line.

- Many different headers can appear in responses. These can be divided into several groups:

- General headers, like Via, apply to the whole message.
- Response headers, like Vary and Accept-Ranges, give additional information about the server which doesn't fit in the status line.
- Representation headers like Content-Type that describe the original format of the message data and any encoding applied (only present if the message has a body).

BODY

- The last part of a response is the body. Not all responses have one: responses with a status code that sufficiently answers the request without the need for corresponding payload (like 201 Created or 204 No Content) usually don't.

- Bodies can be broadly divided into three categories:

- Single-resource bodies, consisting of a single file of known length, defined by the two headers: Content-Type and Content-Length.
- Single-resource bodies, consisting of a single file of unknown length, encoded by chunks with Transfer-Encoding set to chunked.
- Multiple-resource bodies, consisting of a multipart body, each containing a different section of information. These are relatively rare.

API Principals

- Some of the guidelines and basic principles described below are subjective but they are essential in today's API development. They provide fundamental benefits and help to stay at par with industry-wide adoption of best practices:

- Vocabulary

- This refers to the standard naming conventions one should following while naming each API endpoint. They should be human-readable, easy to understand and follow the HTTP standards.

- Versioning

- By versioning, you are allowing various consumers to access your published APIs in two different variations. Though Version management adds complexity to the existing APIs, they do however help in better management of API endpoints, thereby serving various consumers through different mediums. There are two different ways to implement this:

- URL — For e.g., api.myorg.com/v1/users
- Accept Header — requesting for specific version via request/accept header
- Support Multiple Media Types

- At any point in time, a given object or resource can have multiple representations. This is necessary so that various consumers can request the content or resource in the manner that they would like. Having said that, it is not necessary to support all media types, only the ones that are required based on specific use cases.

- Caching and Concurrency Control

- Caching improves performance, thereby providing faster access to frequently accessed resources and eliminating the load on backend services. However, caching comes the challenge of managing concurrent access. Therefore, it is essential to manage the caching in a better way using HTTP standards such as:

- ETag — Entity tagging. Equivalent to versioning each entity for updates
- Last-Modified — Contains the last modified timestamp
- Standard Response Codes

- This responsibility lies with business owners as it affects the business needs of consumers of your APIs. The contract definition should contain all possible error codes that could occur with each API.

- Adhere to the standard HTTP response codes
- Include both business and developer messages. Developer messages should be optional and contain technical messages that guide debugging and troubleshooting techniques.
- Due to security reasons, do not reveal too much about the request (to avoid Cross-Site Request Forgery).
- The best practice is to limit the list of potential error codes, as too many error codes lead to chaos.
- Security Considerations

- This does not require much explanation, as security requirements are the basic needs of any application or an API. Keep in mind that your API's are mostly public, so invest the effort required to secure them. API management platforms (explained in the later section) provides security mechanisms; however, as an API developer, you should be aware of the current trends and industry best practices adopted in addressing security requirements.

- Always use SSL
- APIs are stateless, so avoid session/cookie management — authenticate each request
- Authorize based on resource, not on URL
- HTTP status code 401 vs. 403: Some may prefer to use code 401 rather than 403 to indicate that either authentication or authorization failed
- Follow the guidelines defined by the Open Web Application Security Project (OWASP) Threat Protection