API

An API (AppAn API (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate with each other. It defines the methods and data formats that applications can use to request and exchange information.

HTTP Status Codes

•2xx (Success): 200 OK, 201 Created, 204 No Content

•3xx (Redirection): 301 Moved Permanently, 304 Not Modified

•4xx (Client Error): 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found

•5xx (Server Error): 500 Internal Server Error, 503 Service Unavailable

Response Headers

•Content-Type, Cache-Control, Authorization, etc.

2.**Response Body**

•Data format (JSON schema validation)

•Data accuracy

•Error messages

3.**Response Time**

•Performance benchmarks

Aspect	REST	SOAP
Protocol	Uses HTTP/HTTPS	Can use HTTP, SMTP, TCP, etc.
Format	JSON (mostly), XML, plain text	XML only
Standards	Architectural style (no official standard)	Strict protocol with WSDL specification
Performance	Lightweight, faster	More overhead due to XML
State	Stateless	Can be stateful or stateless
Caching	Supports caching	No built-in caching
Security	Relies on HTTPS/OAuth	WS-Security (more robust)
Flexibility	More flexible, easy to implement	Rigid structure

Key Notes:

•REST is preferred for public APIs, mobile apps, and web services where performance matters
•SOAP is used in enterprise environments (banking, healthcare) where security and reliability are critical
•REST uses standard HTTP methods, SOAP uses custom XML envelopes

Explain HTTP status codes you've worked with.

Answer:
HTTP status codes are grouped into 5 classes:

1xx (Informational)

•100 Continue
•102 Processing

2xx (Success)

•200 OK (Standard success)
•201 Created (Resource created)
•204 No Content (Success but no body)

3xx (Redirection)

•301 Moved Permanently
•304 Not Modified (Cached version valid)

4xx (Client Errors)

•400 Bad Request (Malformed syntax)
•401 Unauthorized (Missing auth)
•403 Forbidden (No permission)
•404 Not Found
•429 Too Many Requests (Rate limiting)

5xx (Server Errors)

•500 Internal Server Error

•502 Bad Gateway

•503 Service Unavailable

•504 Gateway Timeout

PUT	PATCH
Replaces entire resource	Partially updates resource
Idempotent (same call = same result)	Not always idempotent
Must send complete representation	Only send fields to update
Can create if resource doesn't exist	Typically doesn't create resources

REST

Principle	Meaning
1. **Client-Server**	The client and server are separate and communicate via requests and responses.
2. **Stateless**	Each request contains all the data needed (no session or memory stored between calls).
3. **Cacheable**	Responses can be cached (e.g., GET results) to improve performance.
4. **Uniform Interface**	API follows consistent rules using HTTP (GET, POST, PUT, DELETE).
5. **Layered System**	A client cannot tell whether it's connected to a proxy, load balancer, or server.
6. **Code on Demand (optional)**	The server can send executable code (e.g., JavaScript) to the client.

What Problem Does JWT Solve?

🛑 Traditional Session Auth:

- After login, server stores a session ID in memory (RAM or Redis).
- Client stores this session ID in a cookie.
- **Drawbacks:**
    - Server has to **maintain state** (violates statelessness).
    - Hard to scale in distributed systems (need shared session storage).

With JWT (Stateless Auth):

- After login, the server **generates a token** that encodes user info (e.g., user ID).
- The token is sent to the client.
- On every request, the **client sends this token** in the `Authorization` header.
- The server **verifies the token** using a **secret key** (no database/session lookup needed).

✅ **No server-side session needed** → fully **stateless**, **scalable**, and **secure**.

Part	Name	Purpose
Header	`xxxxx`	Type of token + algorithm used
Payload	`yyyyy`	Data (user ID, role, expiry)
Signature	`zzzzz`	Verifies token integrity

Signature

This part **verifies that the token was not tampered with**.

```
HMACSHA256(
  base64urlEncode(header) + "." + base64urlEncode(payload),
  secret
)
```

The server signs the header + payload using a **secret key**.

- On each request, the server **recalculates the signature** from the token.
- If it doesn't match: token is invalid or tampered with → reject.

```
from jose import jwt

# Encode
data = {"sub": "user123"}
token = jwt.encode(data, secret_key, algorithm="HS256")

# Decode + Verify
decoded = jwt.decode(token, secret_key, algorithms=["HS256"])
```

How the Signature Is Actually Verified
Let's say the token parts:
Header: {"alg":"HS256","typ":"JWT"}
Payload: {"sub":"user123","exp":1696345600}
These are base64-encoded and joined:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.
eyJzdWIiOiJ1c2VyMTIzIiwiZXhwIjoxNjk2MzQ1NjAwfQ

Then the server does:
signature = HMAC-SHA256(base64(header.payload), secret_key)
****If that signature matches the one in the token, it's valid.**

URL Encoding

•Special characters must be encoded
•Space becomes `%20`
•`/` becomes `%2F`
•Example: `search?q=API%20Testing`

Verify HTTP status codes (200, 201, 400, 401, 404, 500, etc.).
•**Validate response payload** (JSON/XML schema, data accuracy).
•**Test CRUD operations** (POST for create, GET for read, PUT/PATCH for update, DELETE for remove).
•**Check error handling** (invalid inputs, edge cases).
•**Validate headers** (Content-Type, CORS, caching).
•**Test authentication/authorization** (API keys, OAuth, JWT).
•**Performance testing** (latency, throughput under load).