Authentication

It's about proving WHO you are.

The system checks your identity using credentials
like username/password, OTP, fingerprint, or API key.

If credentials are valid → you're authenticated.

Real-Life Example

At the entrance of airport, security asks for your ID card + ticket.

They check if the details match → if yes, you are allowed inside the airport.
You are now authenticated

Authorization

It's about what you are allowed to do AFTER authentication

- Once identity is confirmed, the system checks your permissions/roles.
- Decides which resources you can access and which actions you can perform.

Real-Life Example (continuing airport)

- if you have economy class ticket, you can't go to business class.

Authentication in APIs (AuthN)

Checks WHO is calling the API

- The API needs to know the caller's identity.
- You provide credentials (API key, username/password, OAuth token, etc.).
- If credentials are valid → API issues you a session/token.
- If authentication fails, server returns: 401 Unauthorized

Common Authentication Methods

1. API Key Authentication

The simplest form. A unique key is generated for each developer/user.

Sent in header or query string with every request.

The server checks if the key is valid.

Example:

```
GET /weather
x-api-key: abc123xyz
```

Pros: Easy to use, widely supported.
Cons: Not very secure (can be leaked if logged or exposed).
So, almost never used for user facing api.

2. Basic Authentication

-> Uses a username + password encoded in Base64.

-> Sent with every request in the Authorization header

Example:

```
GET /profile
Authorization: Basic dGVzdCBwYXNzd29yZA==
```

Pros: Simple to implement.
Cons: Insecure unless used with HTTPS (password exposed in every request).

3. Session-Based Authentication (Cookies)

Classic web approach:

-> User logs in → server creates a session and stores it.

-> Server sends a cookie back to the client.

-> Client(browser) sends cookie with every request.

Example: `Cookie: sessionId=abc123xyz`

Pros: Common for websites.
Cons: Less common for modern APIs (stateless REST prefers tokens).

4. Bearer Token / JWT (JSON Web Token)

-> User logs in once → server issues a token.

-> Token is sent in Authorization header with every request.

-> JWT can contain user info, roles, expiry time, etc.

Example:

```
GET /orders
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6...
```

Pros: Secure, stateless (no need to store sessions).
Cons: If token is stolen, attacker has full access until expiry.

5. OAuth 2.0

-> OAuth = Open Authorization.

-> It's a delegated authorization framework (not just authentication).

- > It allows a user to grant limited access to
their resources (data, account, files) on one service to another app,
without sharing their password.

Example:

You log into an app using "Login with Google".

The app doesn't get your Google password.

Instead, it gets a token that says:
"This app can see this user's email and profile, nothing else."

Pros: Very secure, supports 3rd party apps.
Cons: More complex to implement.

6. Mutual TLS (mTLS)

Both client and server present certificates.

Ensures not only the server is trusted, but the client is also verified.

Often used in banking, financial, and enterprise APIs.

Authorization in APIs (AuthZ)

-> Checks WHAT the caller can do.

-> After authentication, the API checks your roles/permissions.

-> Authorization decides whether your identity has
access to the requested resource.

Example:

Authenticated user with role=user → can access /profile.

Authenticated user with role=admin → can also access /admin/reports.

If you try to access something
outside your permissions:

403 Forbidden

Practical API Flow

Step 1: Authentication (Login)

```
POST /login
Host: api.company.com
Content-Type: application/json

{
  "username": "aniket",
  "password": "mypassword"
}
```

Response (token):

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR...ffhnjnks3fh"
}
```

You are authenticated. ✔

Step 2: Authorization (Access Control)

- Now use token to request data:

```
GET /admin/reports
Host: api.company.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...
```

If token has role=admin → success. ✔

If token has role=user → forbidden.

Error Codes to Remember

401 Unauthorized → You are not authenticated (missing/invalid credentials).

403 Forbidden → You are authenticated, but not authorized (no permission).

DEMO

API Headers

Headers are key–value pairs sent along with an API request (or response).

They give metadata (extra info) about the request/response —
like what format the data is in, who's sending it, how to cache it, etc.

Without headers, servers/clients wouldn't know how to handle the request properly.

Types of API Headers

General Headers

Apply to both request and response.

Example:

Date: Sat, 13 Sep 2025 12:00:00 GMT → tells when the message was sent.

Cache-Control: no-cache → tells the client/server not to cache.

Request Headers (client → server)

Provide details about what the client wants.

-> Authorization → carries credentials or tokens.

```
Authorization: Bearer <token>
```

-> Content-Type → tells the server what format the body is in.

```
Content-Type: application/json
```

-> Accept → tells the server what kind of response format is expected.

```
Accept: application/json
```

-> User-Agent → info about the client (browser, app, version).

```
User-Agent: Mozilla/5.0
```

-> Host → specifies the domain name of the server.

```
Host: api.example.com
```

Response Headers (server → client)

- Give metadata about the response.

-> Content-Type → tells client format of response body.

```
Content-Type: application/json
```

-> Set-Cookie → server sets cookies on client.

```
Set-Cookie: sessionId=xyz123; HttpOnly
```

-> Access-Control-Allow-Origin → used in CORS (Cross-Origin Resource Sharing).

```
Access-Control-Allow-Origin: *
```

-> Content-Length → size of response body in bytes.

```
Content-Length: 340
```

Most Common Headers in Real APIs

Authorization → auth token / API key.

Content-Type → format of request body.

Accept → expected response format.

Cache-Control → caching rules.

User-Agent → who's making request.