
Node.js
Run JavaScript Everywhere

Agenda

01 introduction to backend and node.js

02 Working with Express.js

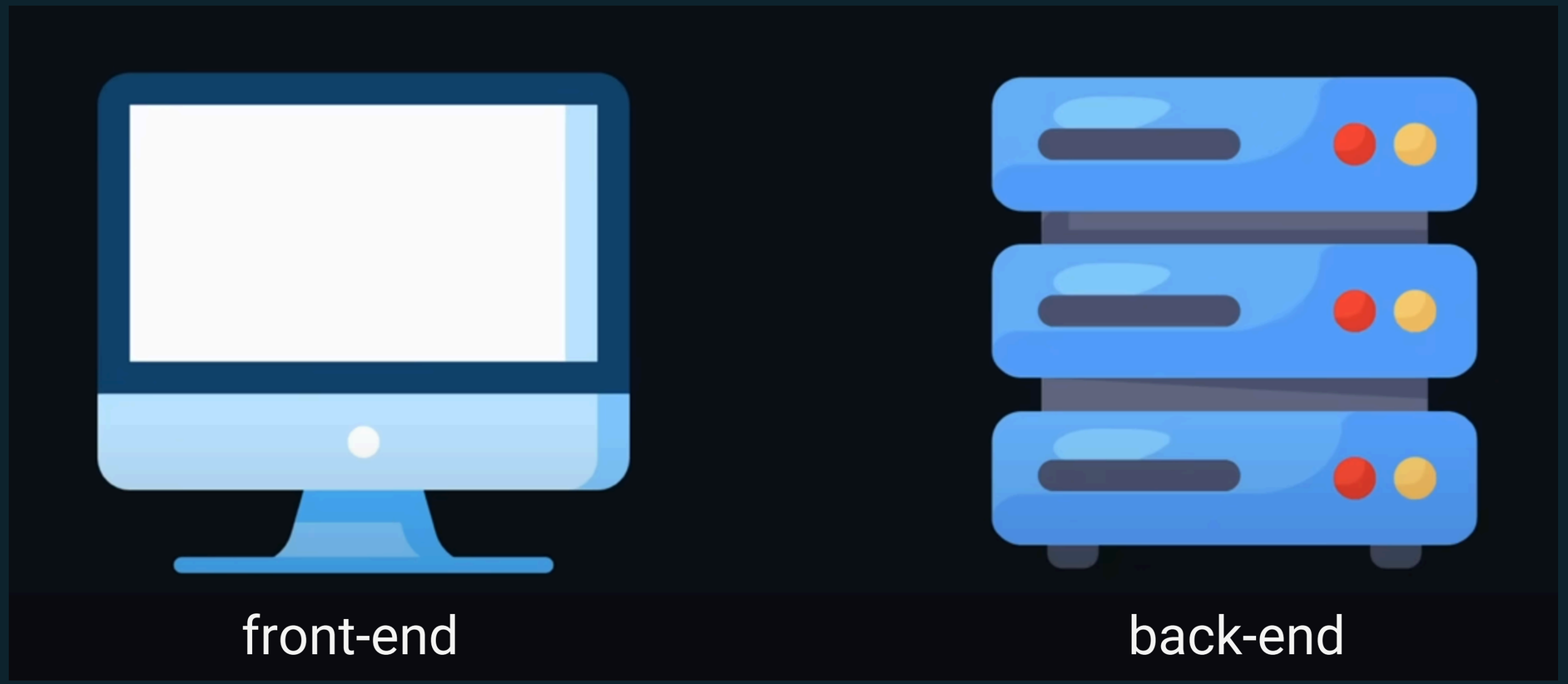
03 Database Integration & Architecture

04 Authentication, Authorization & Advanced Topics

01

introduction to backend and node.js

Web Development is Split into Two Main Components:



the frontend is all about UX with 3 key parts :

the user

the client device or web browser

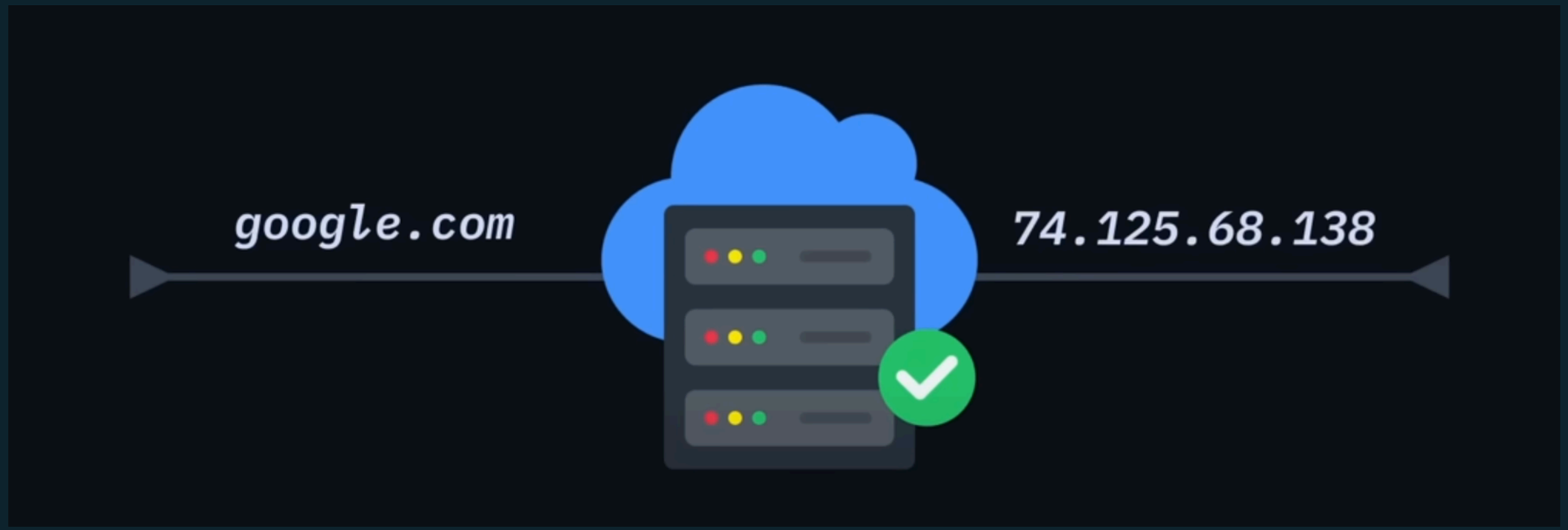
the interface built with HTML, CSS and JavaScript

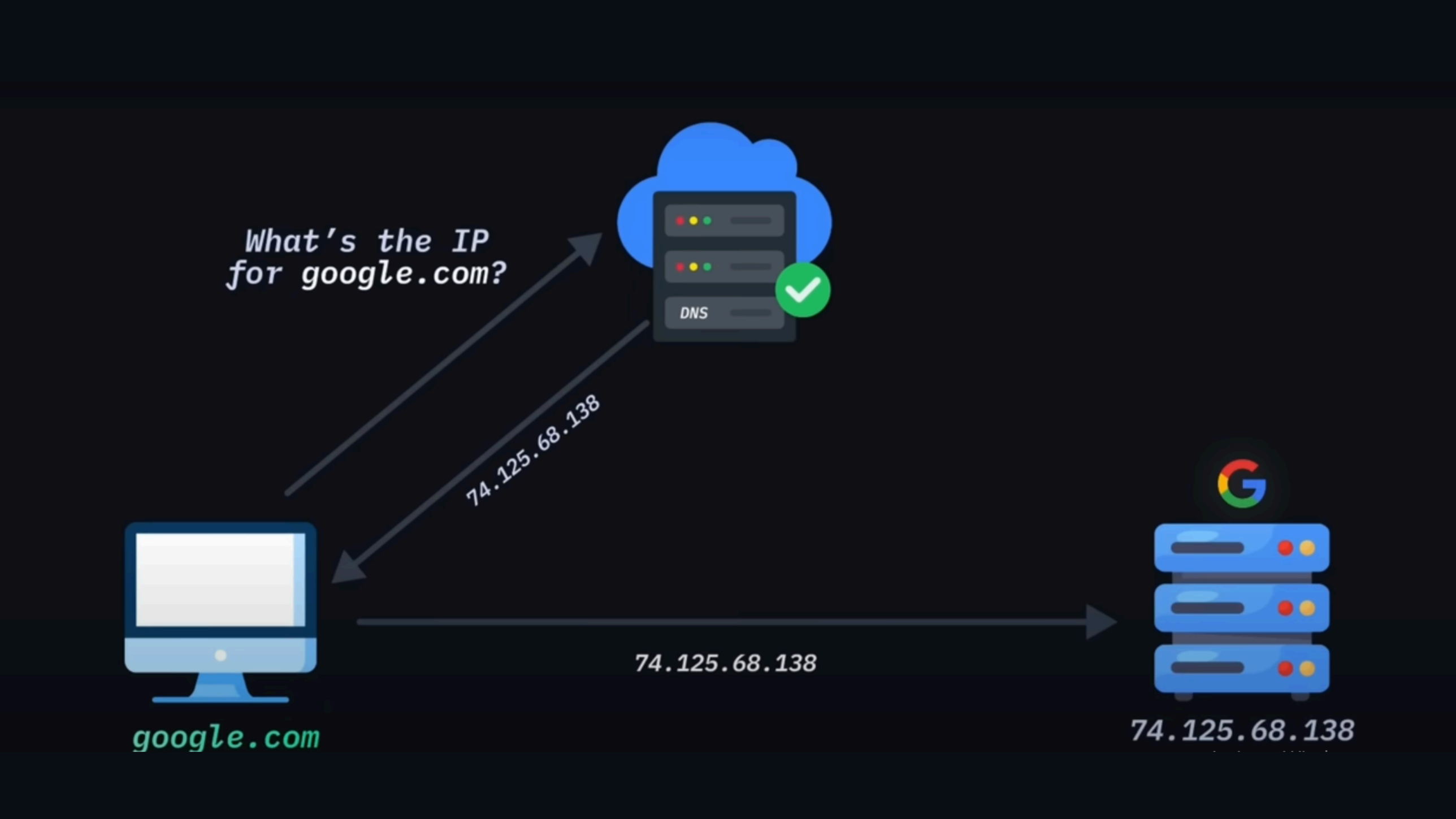
server : is a powerfull computer designed for store , process and send data

How does your device communicate with a random server to get things done ?

con..

The client and server communicate instantly using protocols, with HTTP being the most important. When you visit a website, your browser uses HTTPS for secure communication. Before sending a request, it first relies on DNS to locate the server.


google.com
74.125.68.138


What's the IP for google.com?
DNS
74.125.68.138
74.125.68.138
google.com
74.125.68.138

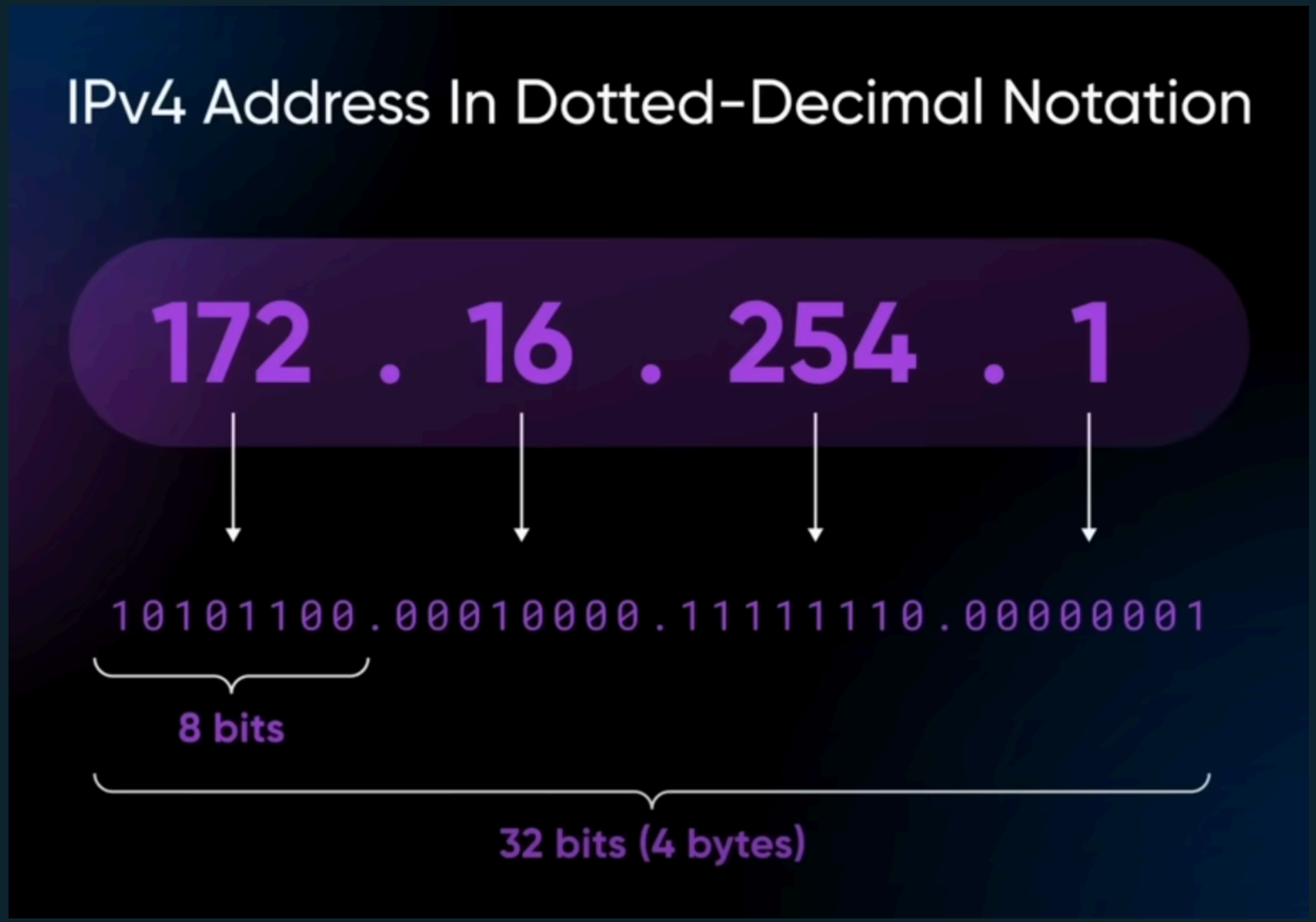
IP Address

IP address is a unique number that identifies every device on the internet , think of it as your device home address ,there are two types of IPS

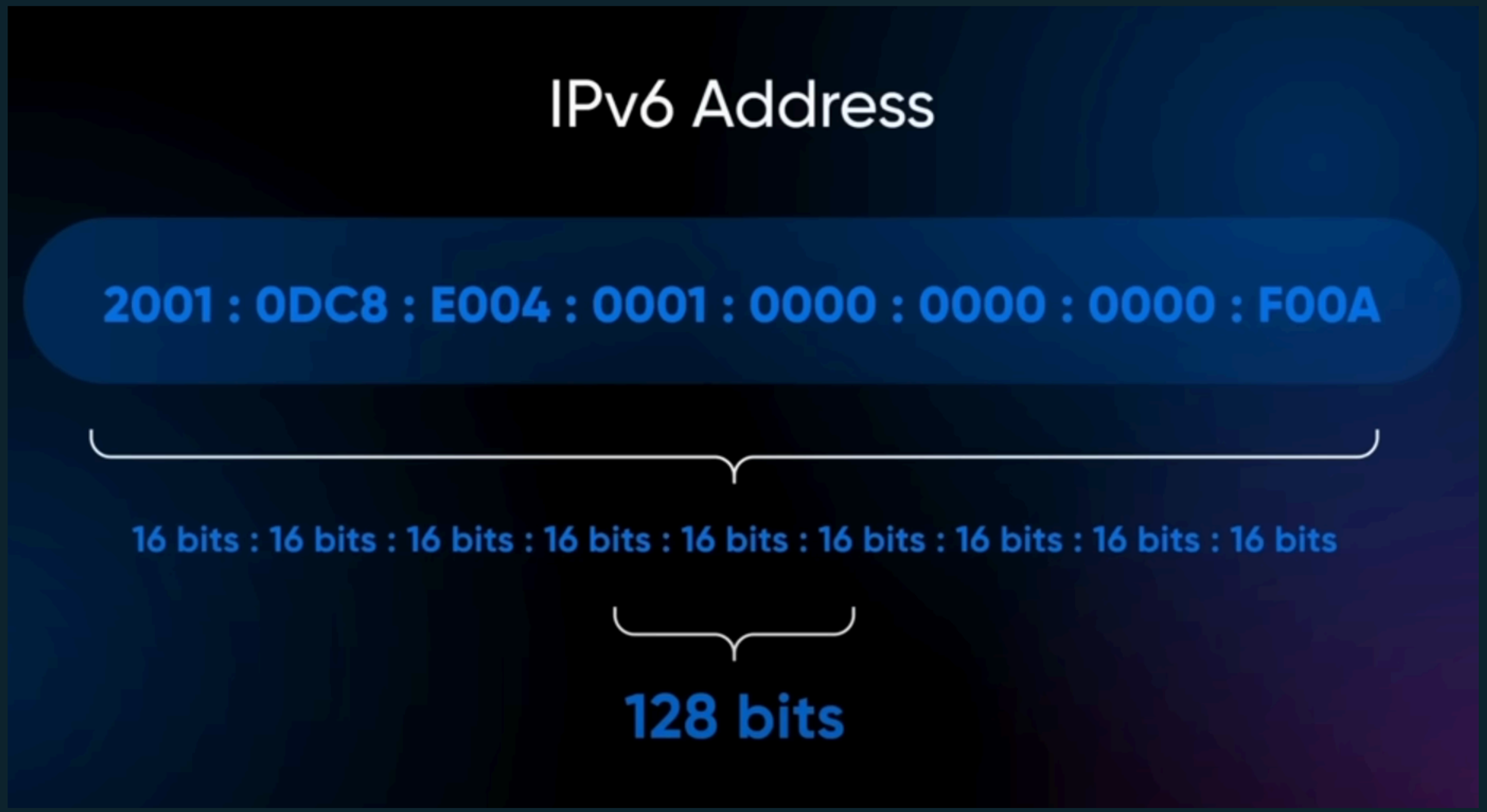
IPv4

IPv6

IPv4


IPv4 Address In Dotted-Decimal Notation
172 . 16 . 254 . 1
10101100.00010000.11111110.00000001
8 bits
32 bits (4 bytes)

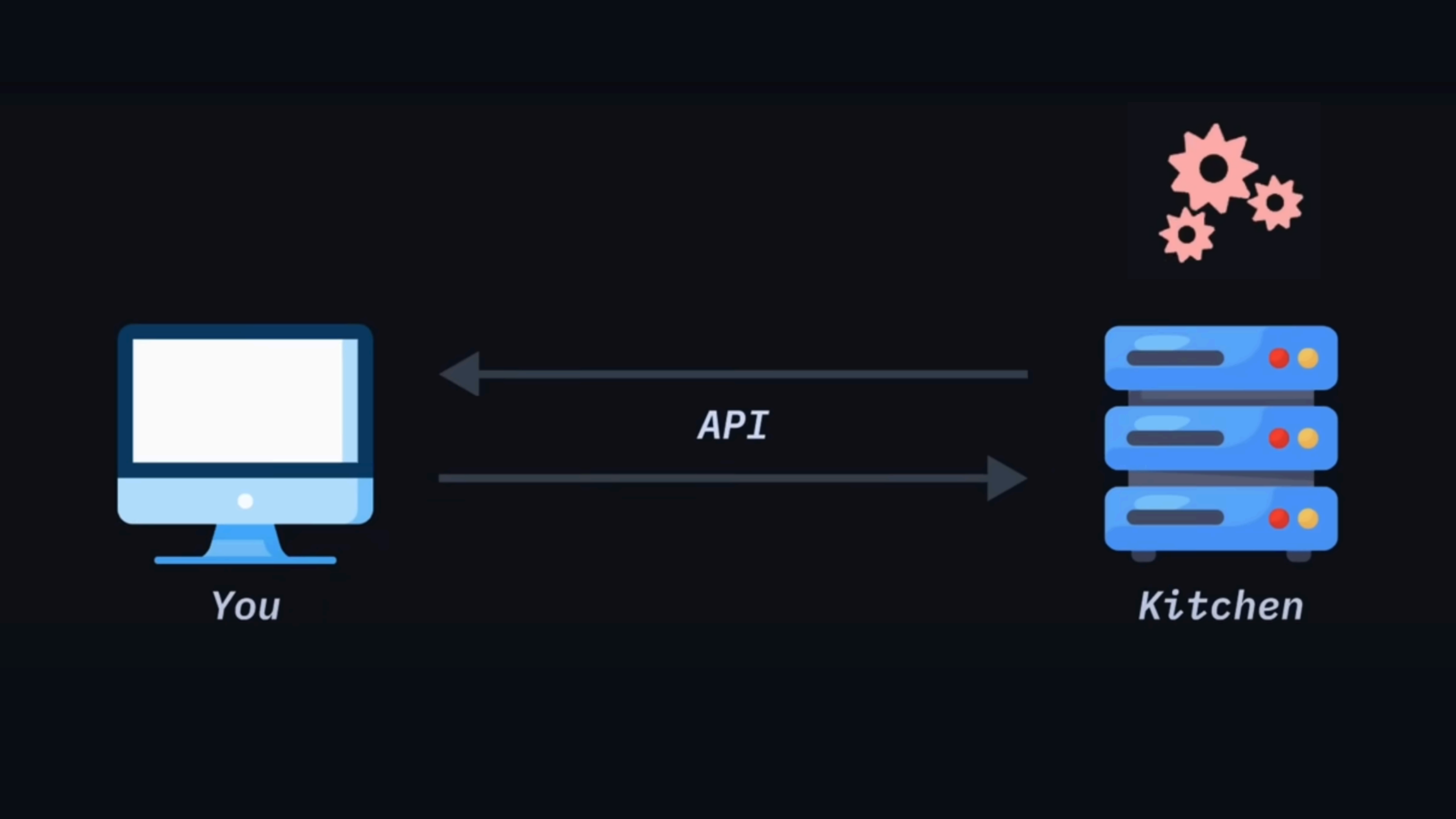
IPv6


IPv6 Address
2001 : 0DC8 : E004 : 0001 : 0000 : 0000 : 0000 : F00A
16 bits : 16 bits : 16 bits : 16 bits : 16 bits : 16 bits : 16 bits : 16 bits
128 bits

so now you know how your device connnects to a server but how does it know what to ask for that whrere “API’s” come in

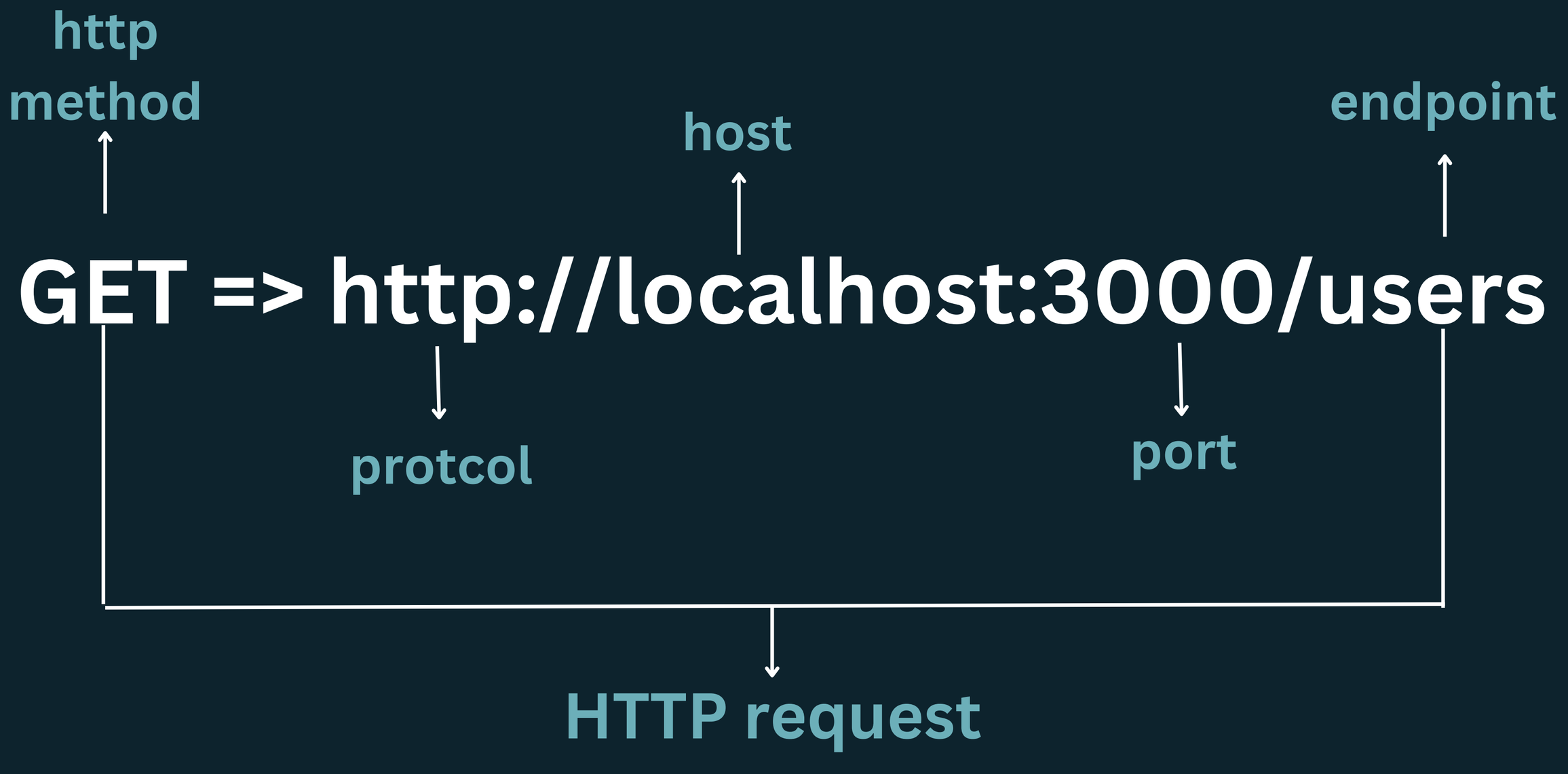
API (Application programming interface)

is a set of rules and protocols that allows different software applications to communicate with each other. It defines how requests and responses should be structured, enabling seamless interaction between systems.


API
You
Kitchen

APIs follow a specific structure that helps us manage and manipulate resources be that data or objects that can :

- create
- read
- update
- delete


http
method
host
endpoint
GET => http://localhost:3000/users
protcol
port
HTTP request

HTTP Methods

HTTP defines several methods that indicate the type of action a client wants to perform on a server. The most common HTTP methods are:

GET	Retrieves data from the server.
POST	Sends data to the server to create a new resource.
PUT	Updates or replaces an existing resource on the server.
PATCH	Partially updates an existing resource.
DELETE	Removes a resource from the server.

Endpoints

An endpoint is a specific URL in an API where a client sends requests to interact with a server. Each endpoint corresponds to a particular resource or action.

GET /users ⟶ return a list of users.

POST /users ⟶ create a new user.

PUT /users/:ID ⟶ update a specific user.

DELETE /users/:ID ⟶ delete a specific user.

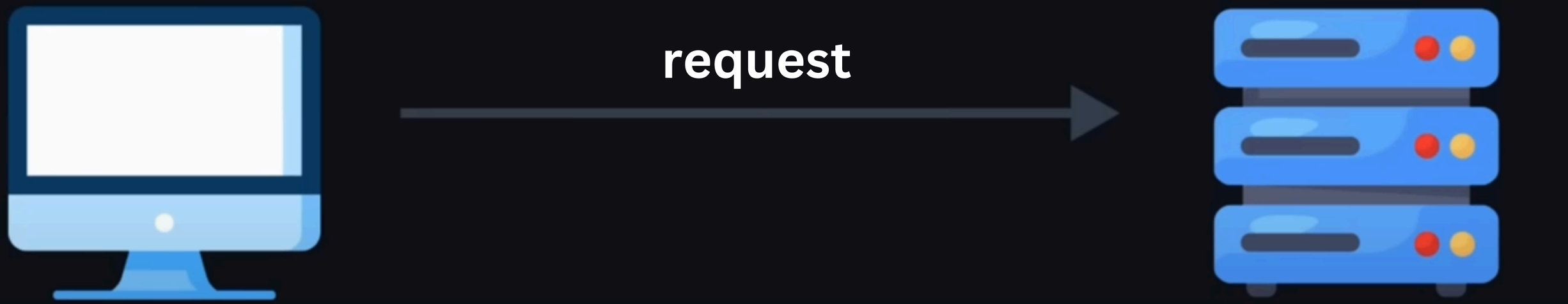
Headers

headers contain extra information like metadata about the request or a response like authentication tokens content type or caching instructions .

Request Body

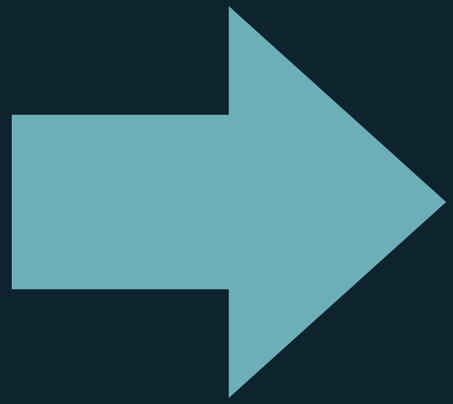
when you use a POST or a PUT/PATCH request to send the data to the server the details go into the request body usually in a JSON format.

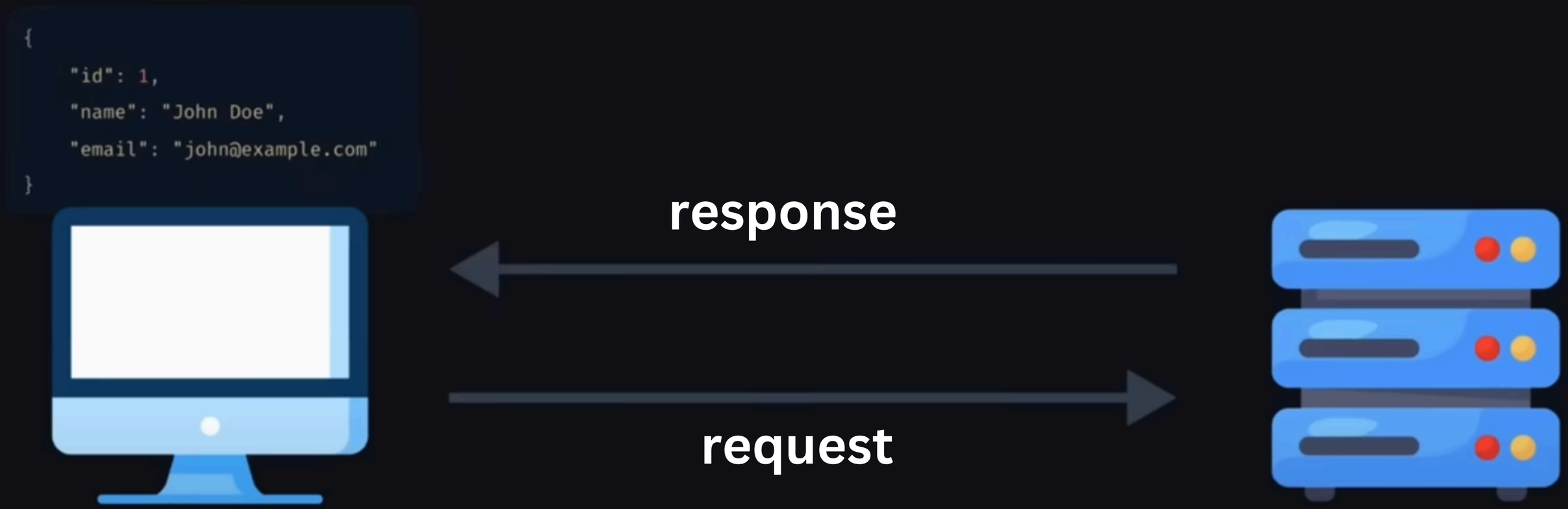
```
{
  "name": "John Doe",
  "email": "john@example.com"
}
```


{
"name": "John Doe",
"email": "john@example.com"
}
request

Response Body

The response body contains the data the server sends back after processing the request.
Also formatted in JSON in most modern APIs.




{
"id": 1,
"name": "John Doe",
"email": "john@example.com"
}
response
request

when i get all users in my system .

GET /users

Status Code

When making API requests, the server responds with a status code in addition to the response body. Status codes indicate whether the request was successful, encountered an error, or requires further action.

1. **Informational (100 – 199)**
   - These codes indicate that the request has been received and is being processed.
2. **Success (200 – 299)**
   -These codes indicate that the request was successful.
3. **Redirection (300 – 399)**
   -These codes indicate that the client must take additional action to complete the request.
4. **Client Errors (400 – 499)**
   -These codes indicate an issue with the client request.
5. **Server Errors (500 – 599)**
   -These codes indicate that the server encountered an issue while processing the request.

200 OK → The request was successful

201 Created → A new resource was created

400 Bad Request → The request was invalid

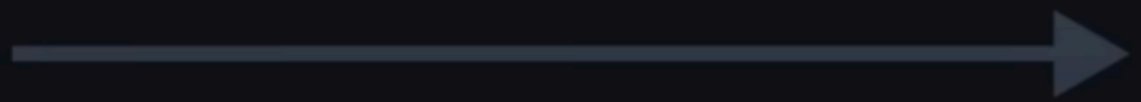
404 Not Found → The resource doesn’t exist

500 Internal Server error → Something went wrong on the server

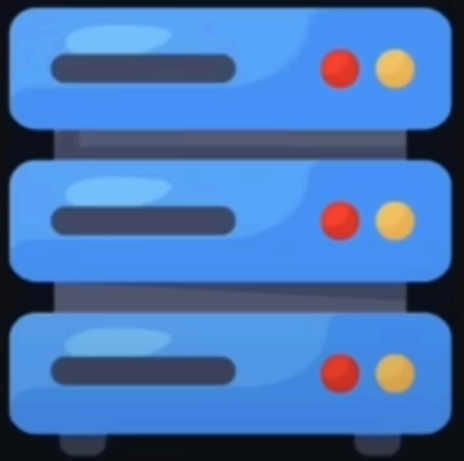
How API’s work behind the scenes

```
const requestOptions = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    Authorization: 'Bearer your_token_here',
  },
  body: JSON.stringify({
    name: 'John Doe',
    email: 'johndoe@example.com',
    age: 30,
  }),
};

fetch('https://api.example.com/users', requestOptions);
```



Request

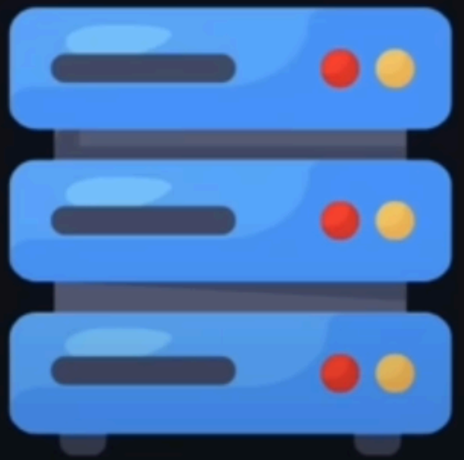


```
app.post("/users", (req, res) => {
  const userData = req.body; // Data sent by the client

  // Simulating a successful operation
  if (userData) {
    res.status(200).json({
      message: "User created successfully!",
      data: userData
    });
  } else {
    // Simulating an error
    res.status(400).json({
      message: "Failed to create user. Missing data."
    });
  }
});
```

Response

Request




API

To build your apis you could use languages like:

JavaScript Run Time:

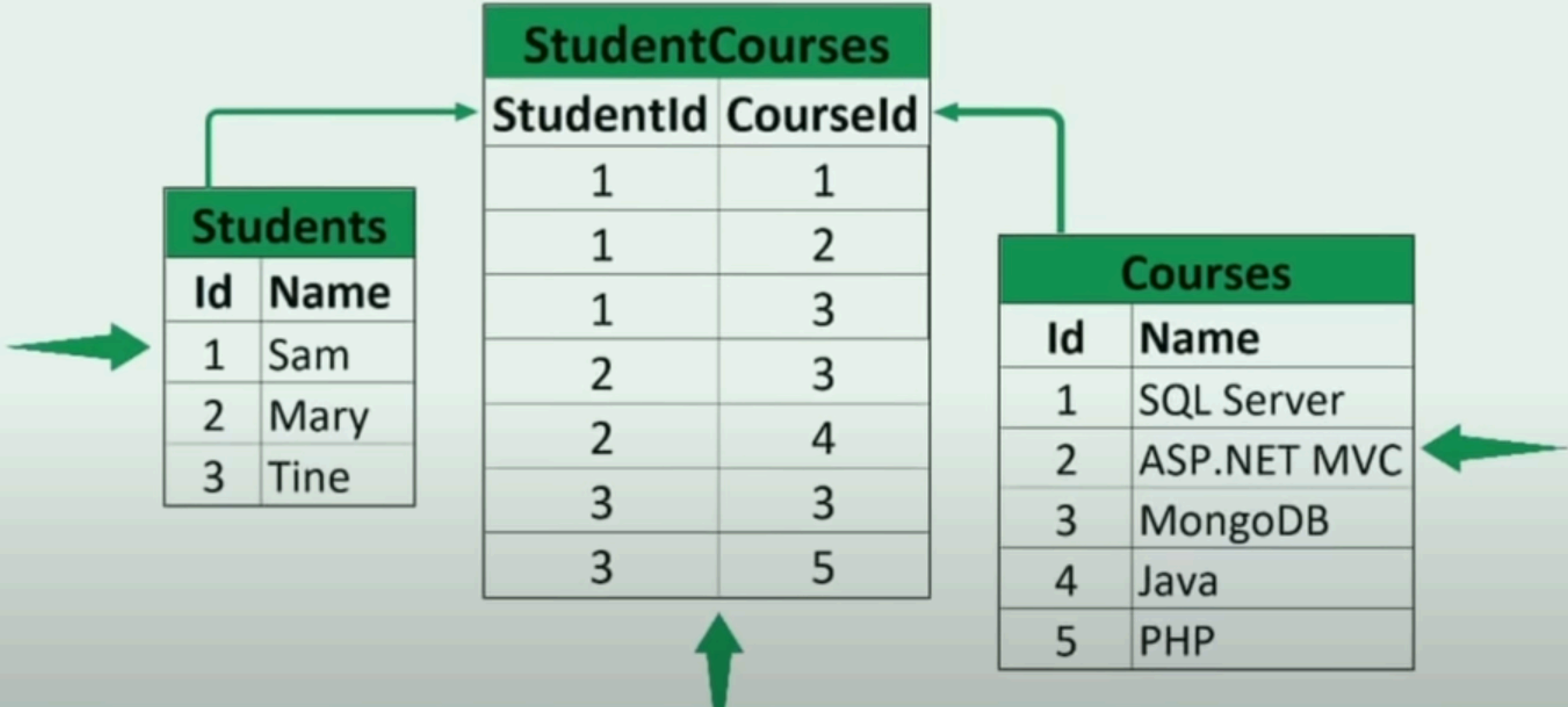
JS Frameworks

some of the most popular backend Frameworks are Express and nestjs for JavaScript

DataBase

Two Types Of Database:

Relational Database



StudentCourses

StudentId	CourseId
1	1
1	2
1	3
2	3
2	4
3	3
3	5

Students

Id	Name
1	Sam
2	Mary
3	Tine

Courses

Id	Name
1	SQL Server
2	ASP.NET MVC
3	MongoDB
4	Java
5	PHP

Non Relational DataBase

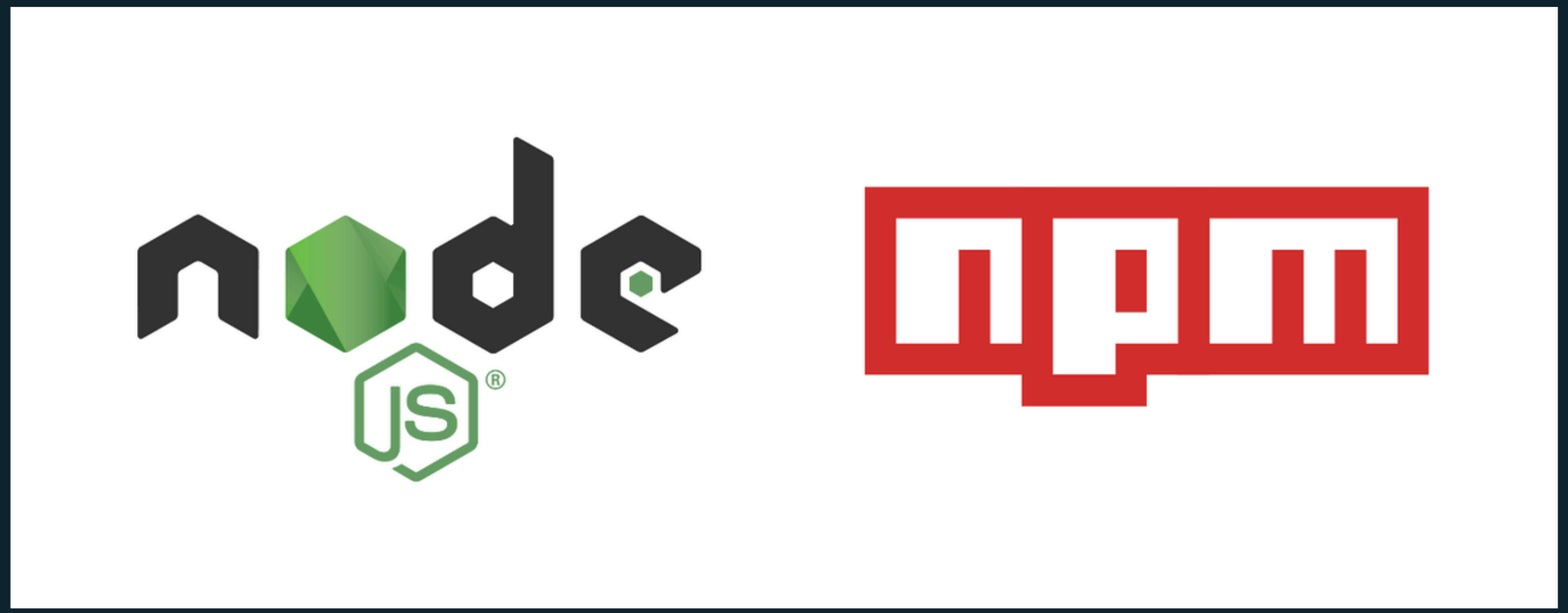

```
{
  "_id": {
    "$oid": "66d4ba1dc6c6e24e676e7b2e"
  },
  "title": "The Last of Us Part II",
  "description": "Experience a heart-wrenching story of survival and revenge in a post-apocalyptic world.",
  "publisher": "Naughty Dog",
  "releaseDate": {…},
  "platform": "PlayStation",
  "price": 49.99,
  "discount": 20,
  "quantity": 300,
  "imageCover": "https://ik.imagekit.io/shadyyd/uploads/api-1725217309002-xlc953w7j_PeQzNAOjyR.jpeg",
  "images": [],
  "category": {…},
  "createdAt": {…},
  "updatedAt": {…},
  "__v": 0
}
```

```
{
  "_id": {…},
  "title": "Cyberpunk 2077",
  "description": "Immerse yourself in a futuristic open-world RPG filled with action, adventure, and deep storylines.",
  "publisher": "CD Projekt",
  "releaseDate": {…},
  "platform": "PC",
  "price": 39.99,
  "discount": 25,
  "quantity": 500,
  "imageCover": "https://ik.imagekit.io/shadyyd/uploads/api-1725217473864-6w53gjixf_l5QE_VLTm.jpeg",
  "images": [],
  "category": {…},
  "createdAt": {…},
  "updatedAt": {…},
  "__v": 0
}
```

02 Working with Express.js

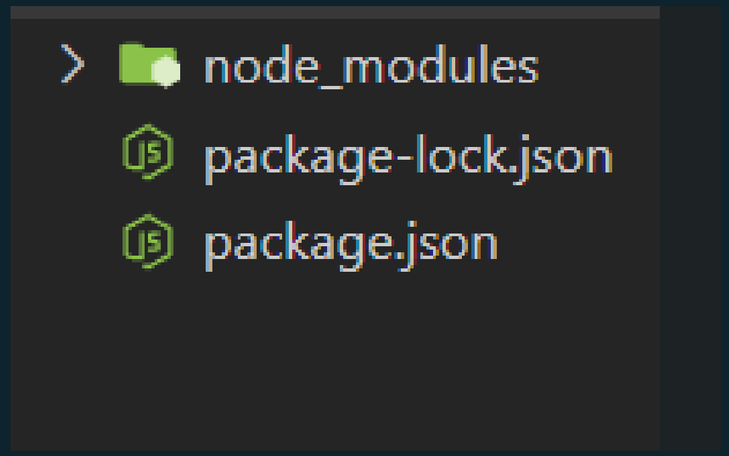
Package Mangers

A package manager is a tool that manages the libraries and dependencies used in a project. When I install Node.js, it automatically comes with npm , which is one of the most commonly used package managers for JavaScript.



Package.json & Package-lock.json & node_modules

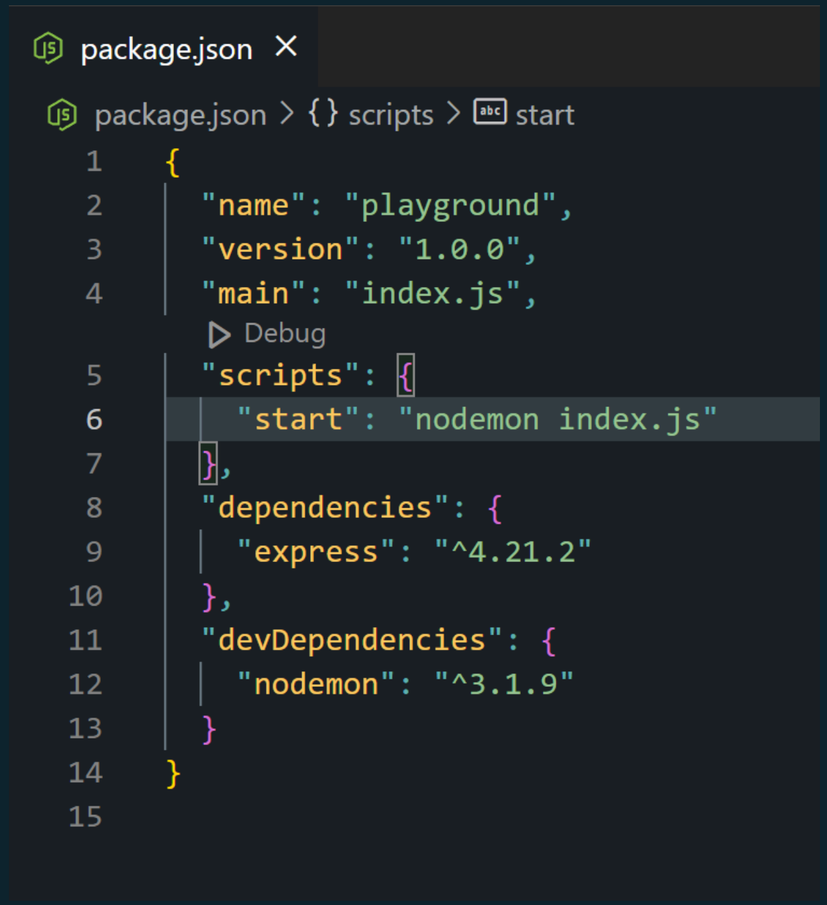
- The package.json file: metadata about the project, dependencies, scripts, etc.
- The package-lock.json file: ensures consistency by locking dependency versions.
- node_modules folder: a folder that will hold all the 3rd party packages installed



Package.json

we have two ways of installing a package :
1- install as a dependency =>> `npm i express`
2- install as a dev dependency =>> `npm i nodemon --save-dev`

Scripts: we can define any script to run later from the terminal
and then run it this way => `npm run start`



What is a RESTful API?

1. What is REST?

- REST (Representational State Transfer) is an architectural style for APIs.
- Uses standard HTTP methods (GET, POST, PUT, DELETE).
- Stateless: Each request is independent and contains all necessary information

The REST Architecture

1. Seperate API into logical **resources (Names not Verbs)**
2. Structured resource-based URLs
3. Use HTTP methods (verbs)
4. Send data as **JSON**
5. Be **stateless**

```
❌Don't                ✅DO
/addNewUser            -> POST /users
/getAllUsers           -> GET /users
/getUser               -> GET /users/3
/updateUser            -> PUT /users/5 || PATCH /users/5
/deleteUser            -> DELETE /users/7
//CRUD Operations: create,read,update,delete

/getUserPosts          -> GET /users/3/posts
/deleteUserPost        -> DELETE /users/3/posts/5
// more complex routes
```

Stateless

Stateless RESTful API: All state is handled on the client.

- This means that request must contain **all** the information necessary to process a certain request .
- The server should **not** have to remember previous requests.

Express.JS

Why Use Express?

1. Built on top of Node.js – Makes it easier to handle requests/responses.
2. Minimal & Flexible – Small but powerful framework.
3. Middleware Support – Ability to process requests before sending a response.
4. Routing System – Helps in handling different HTTP methods (GET, POST, etc.).
5. RESTful APIs – Express is perfect for building APIs.
6. Large Ecosystem – Supports various third-party middleware like authentication, logging, and validation.

Middlewares

-Middleware is a function that runs between the request and response cycle in an application. It processes incoming requests before they reach the final route handler and can modify the request (req) and response (res) objects.

types of middleware:

1-application-level-middleware (ex.. app.use() , every req pass on it / app.post()).

2-router-level-middleware (routes).

3-built-in-middleware (like in express ⇒ express.json()).

4-third-party-middleware (libraries ⇒ cors , morgan).

5-error-handling-middleware .

03 Database

MongoDB & Mongoose

- **MongoDB :** is a NoSQL database that stores data in flexible, **JSON**-like called **BSON** documents rather than rigid tables.

- **Mongoose** :is an ODM (Object Data Modeling) library that provides a straightforward schema-based solution for defining your application data it wraps the native MongoDB driver with features like schema validation, middleware, and simplified CRUD operations.

What is BSON?

BSON (Binary JSON) is a binary format used to store and exchange data. It is primarily used in MongoDB to efficiently store documents. BSON extends JSON by adding additional data types, such as Date, Binary Data, and ObjectId, and it is designed to be more efficient in parsing and storing.

Differences Between BSON and JSON

Feature	JSON (JavaScript Object Notation)	BSON (Binary JSON)
Format	Text-based (UTF-8)	Binary format
Data Size	Compact but can be larger due to text-based nature	Can be larger due to additional metadata, but is optimized for fast retrieval
Performance	Readable but slower to parse	Faster to parse because of binary encoding
Supports Additional Data Types	No (Only String, Number, Boolean, Array, Object, Null)	Yes (Binary, Date, ObjectId, Decimal128, etc.)
Usage	Used for APIs, web applications, and data exchange	Used for internal MongoDB storage and retrieval
Readability	Human-readable	Not human-readable (binary)

Key Advantages of BSON in MongoDB

- Faster Read & Write Operations → Since BSON is binary, MongoDB can process it more efficiently than plain JSON.
- Supports More Data Types → BSON includes additional types like ObjectId, Date, and Binary, which JSON doesn’t support.
- Efficient Indexing & Storage → MongoDB optimizes BSON storage, making queries faster.

Mongoose Schemas and Models

What is a Schema?

A Mongoose Schema defines the structure of documents in a MongoDB collection. It acts as a blueprint that enforces:

- Field types (String, Number, Boolean, Date, etc.).
- Validation rules (required, min/max length, regex patterns).
- Default values and custom methods.

Schemas help maintain data consistency and prevent unexpected values from being stored in MongoDB.

Mongoose Schema

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: {
    type: String,
    required: true,
    unique: true,
    match: /.+\@.+\..+/
  },
  password: { type: String, required: true }
}, { timestamps: true });
```

Mongoose Model

Mongoose Model is a constructor function that allows you to create and query documents based on a schema. It provides an interface to interact with MongoDB collections.

```
const User = mongoose.model('User', userSchema);
module.exports = User;
```

CRUD Operations in Mongoose

```
// CREATE
const user = new User({ name: 'John Doe', email: 'john@example.com', age: 25 });
const savedUser = await user.save();
// OR
const user = await User.create({ name: 'John Doe', email: 'john@example.com', age: 25 })

// READ
const users = await User.find(); // Fetch all users
const user = await User.findOne({ email }); // Fetch one user by email

// UPDATE
const updatedUser = await User.findOneAndUpdate(
    { email }, // search term
    { age: newAge }, //data to update
    { new: true } // options object ex: Return the updated document
 );

//DELETE
await User.deleteOne({ email });
```

Environment Variables (.env)

Why Use Environment Variables?

- Store sensitive credentials (like database URIs) outside your source code.
- Easily switch environments (development, production) by changing the variables.
- We always should ignore any file with (.env) extension when uploading our code online

04 Authentication & Authorization

Authentication vs. Authorization 🔑

Authentication (Who You Are)

✅ Verifies identity (e.g., email & password, biometrics)

✅ Ensures the user is legitimate

✅ Example: Logging into an account

Authorization (What You Can Do)

✅ Determines access to resources

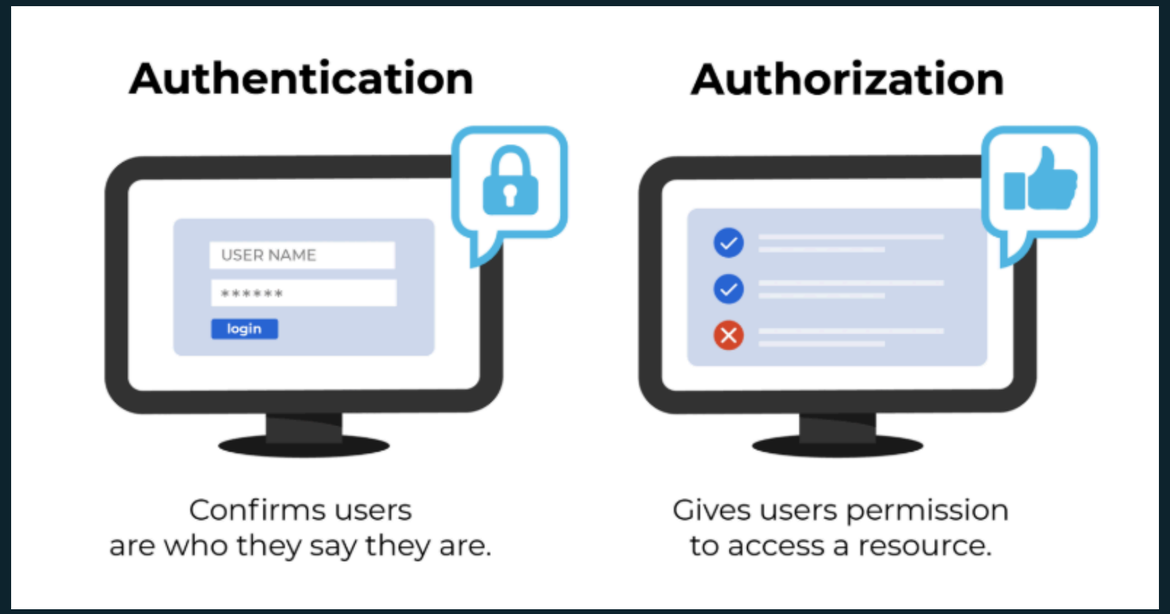
✅ Based on user roles & permissions

✅ Example: Admin vs. regular user access

Key Difference:

🔹 Authentication = Identity Verification

🔹 Authorization = Access Control



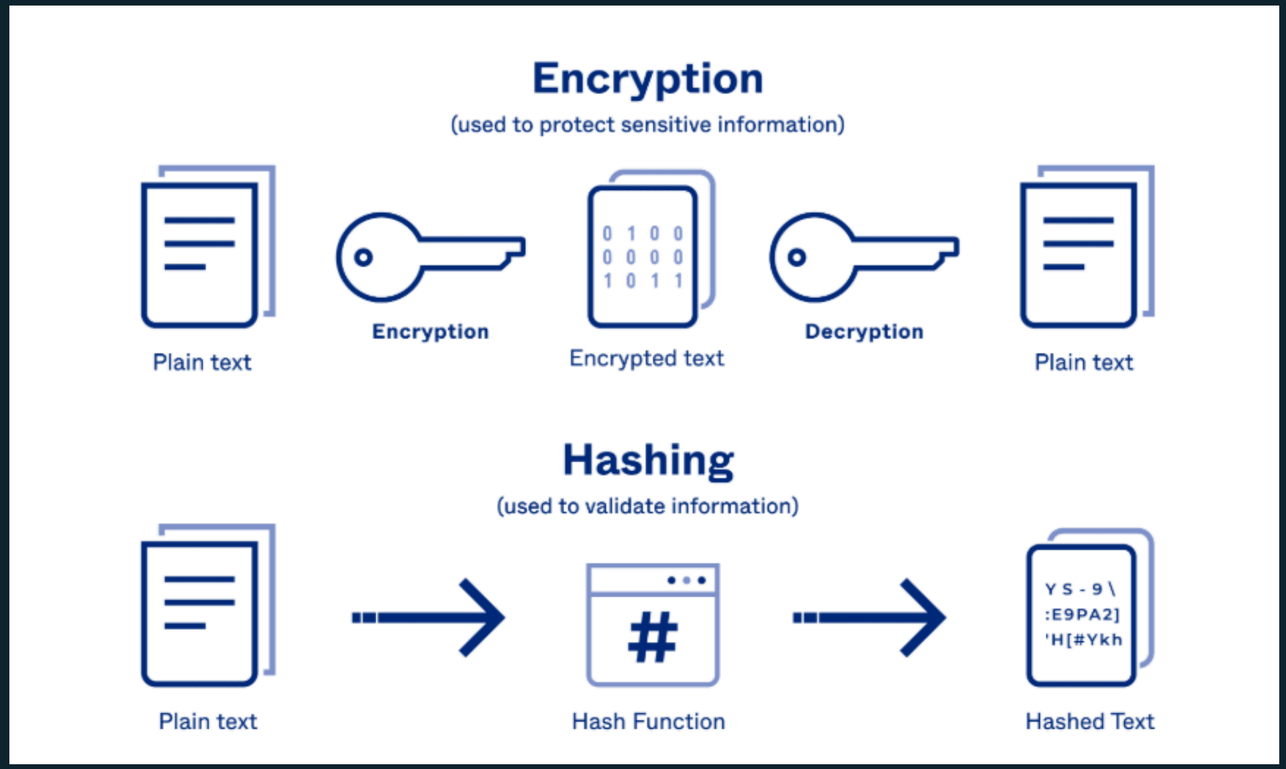
Encryption VS Hashing

- **Encryption:** Two-way process that transforms data into a different format using a key. Can be decrypted back. Used for secure communication (e.g., HTTPS).

- **Hashing:** One-way process that converts data into a fixed-length string. Cannot be reversed. Used for password storage

Key Differences :

- Hashing is irreversible; encryption is reversible.
- Hashing is used for integrity (e.g., password storage); encryption is used for confidentiality (e.g., HTTPS, JWT).
- Common Hashing Algorithms: SHA-256, bcrypt
- Common Encryption Algorithms: AES, RSA



Hashing Passwords

- Converts plain passwords into irreversible hashed values.
- Uses bcrypt: salts the password and applies multiple rounds for added security.
- Stores only the hashed result; on login, compares the hashed input.
- Mitigates brute-force and dictionary attacks.

❌ Bad Practice

```
_id: ObjectId('67cc2163579b9778166bc69b')
name : "rrrrrr radwan"
email : "shady@email.com"
password : "12345678"
role : "user"
createdAt : 2025-03-08T10:52:19.602+00:00
updatedAt : 2025-03-08T10:52:19.602+00:00
__v : 0
```

✅ Good Practice

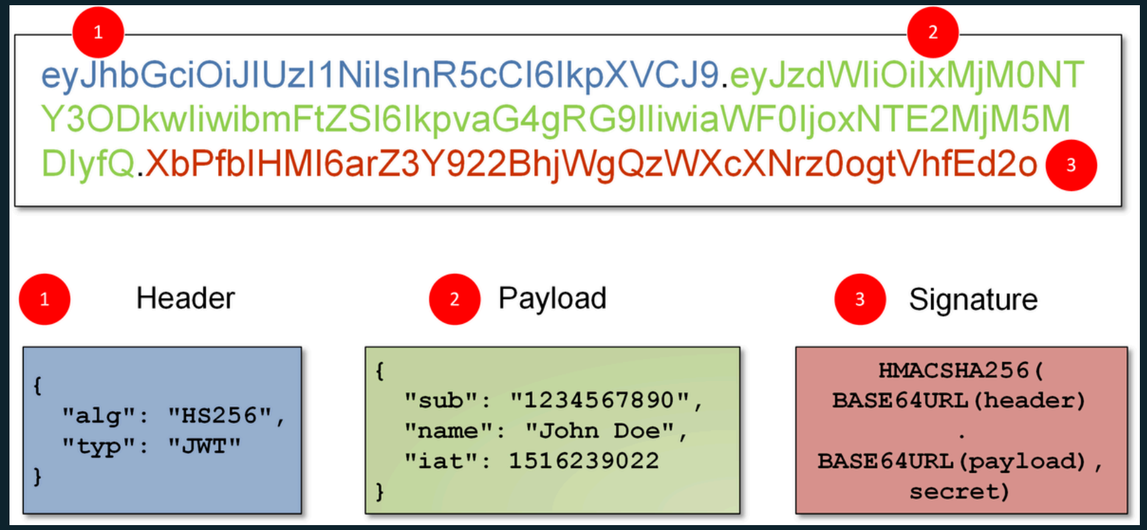
```
_id: ObjectId('67cc9ba8041334e57598728b')
name : "rrrrrr radwan"
email : "shady12433@email.com"
password : "$2b$10$MAw88YcqifwLSvQpR8sf.O6rJ9USB/RTuPkjdWNh9rDS3zYHdt.fm"
role : "user"
createdAt : 2025-03-08T19:34:00.236+00:00
updatedAt : 2025-03-08T19:34:00.236+00:00
__v : 0
```

JWT

JWT (JSON Web Token) is a compact, self-contained token used for authentication and authorization

A JWT consists of three parts:

- **Header** – Algorithm & type
- **Payload** – User claims (ID, role, expiration)
- **Signature** – Ensures integrity



JWT

How JWT Works?

User logs in → Server verifies credentials
Server generates JWT & sends it to client
Client stores & sends JWT in requests (Authorization: Bearer <token>)
Server validates JWT & grants access

Best Practices:

- (Client Side) Store JWT in HTTP cookies (avoid localStorage)
- Set expiration (exp) to limit token lifetime
- Use strong secret keys (at least 32+ characters) & validate signatures
- Implement refresh tokens for extended sessions

Protected Routes with JWT 🛡️

Why Protect Routes?

✅ Ensures only authenticated users can access certain endpoints.

✅ Prevents unauthorized access to sensitive data.

How It Works?

User logs in → Receives a JWT.

Client sends JWT in requests (Authorization: Bearer <token>).

Middleware verifies the token before granting access and fetches the user with the data decrypted from JWT then put this data on the request object (ex: req.user) for the next middleware or handler to have access to the user data.

Role-Based Access Control

What is RBAC?

✅ Restricts access based on user roles (e.g., Admin, User).

✅ Ensures users only access what they’re authorized for.

How It Works?

User logs in → JWT contains role (role: "admin" | "user").

Middleware checks role before allowing access.

Security Middleware for Node.js 🛡️

🛑 Helmet – Secure HTTP Headers

- Adds essential security headers to HTTP responses.
- Protects against clickjacking, MIME sniffing, and XSS attacks.
- Helps mitigate man-in-the-middle (MITM) attacks.

🔒 Express-Mongo-Sanitize – NoSQL Injection Prevention

- Removes harmful query operators like $ and . from user input.
- Prevents NoSQL injection attacks in MongoDB.
- Ensures data integrity by sanitizing incoming requests.

⚠️ XSS-Clean – Cross-Site Scripting (XSS) Protection

- Prevents injection of malicious JavaScript in user input.
- Protects against session hijacking and data theft.
- Removes harmful scripts from request bodies, URLs, and headers.

Security Middleware for Node.js 🛡️

🚧 HPP – HTTP Parameter Pollution Protection

- Blocks repeated query parameters that can be exploited.
- Prevents attackers from overriding API parameters.
- Protects APIs and databases from unexpected behavior.

⏳ Express-Rate-Limit – Rate Limiting for APIs

- Limits the number of requests per user/IP to prevent DoS attacks.
- Helps mitigate brute-force attacks on authentication routes.
- Improves API security by restricting excessive requests.

Question?

Advanced Backend Topics 🚀

If you want to grow as a backend developer, here are key topics to explore

🛠️ Backend Architecture
- Monolithic vs. Microservices
- Serverless Architecture

🐳 Deployment & DevOps
- Docker & Containerization
- CI/CD (Continuous Integration & Deployment)

⚡ Scalability & Performance
- Caching (Redis, CDN)
- Load Balancing & API Gateways

🌍 Distributed Systems & System Design
- Horizontal vs. Vertical Scaling

📨 Communication Between Services
- Message Queues (RabbitMQ, Kafka)
- Event-Driven Systems

📌 Best Practices & Code Quality
- SOLID Principles & Design Patterns
- Clean Code & Code Reviews

🗃️ Databases & Optimization
- SQL vs. NoSQL Databases
- Indexing & Query Optimization

Thank You

LinkedIn Profiles:

Shady Radwan