

Design a Dynamic Frontend with React



Serving Data with JSON Server



Engage and Think



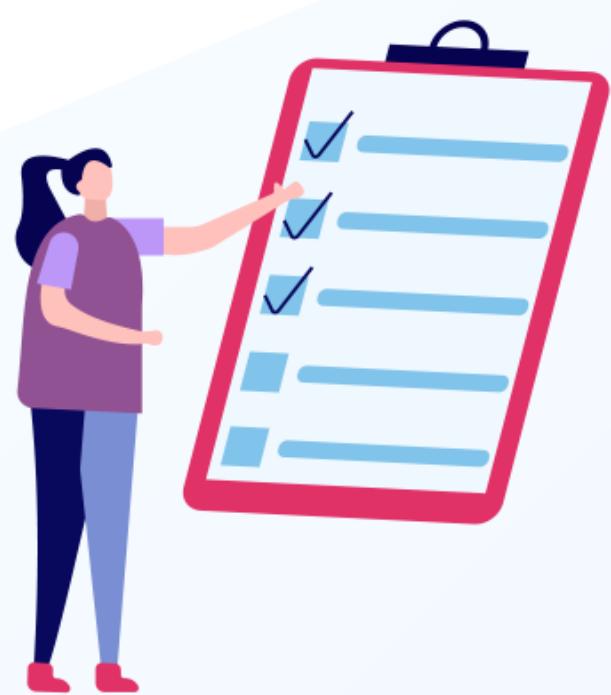
You are a MERN stack developer at a fintech startup, building a banking application. To manage customer data efficiently, you need a lightweight, structured, and scalable solution. Your team suggests using a JSON Server to store and handle data seamlessly.

How would you structure and manage data for fast retrieval, updates, and integration with your application?

Learning Objectives

By the end of this lesson, you will be able to:

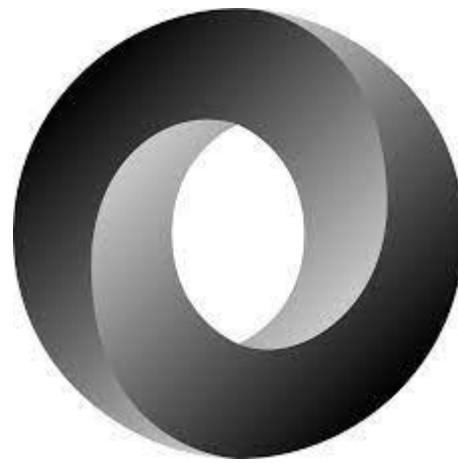
- Define the structure and functionality of JSON for effective data modeling in web applications
- Set up and configure a JSON server for managing data storage and retrieval, ensuring a robust backend for web applications
- Implement CRUD operations using JSON (GET, POST, PUT, DELETE) for dynamic data handling in web application development
- Analyze and manipulate data types, including objects and arrays in JSON for optimized data structure and management
- Integrate JSON with HTML elements for seamless data display and interaction in web interfaces



Overview of JSON

JSON: Introduction

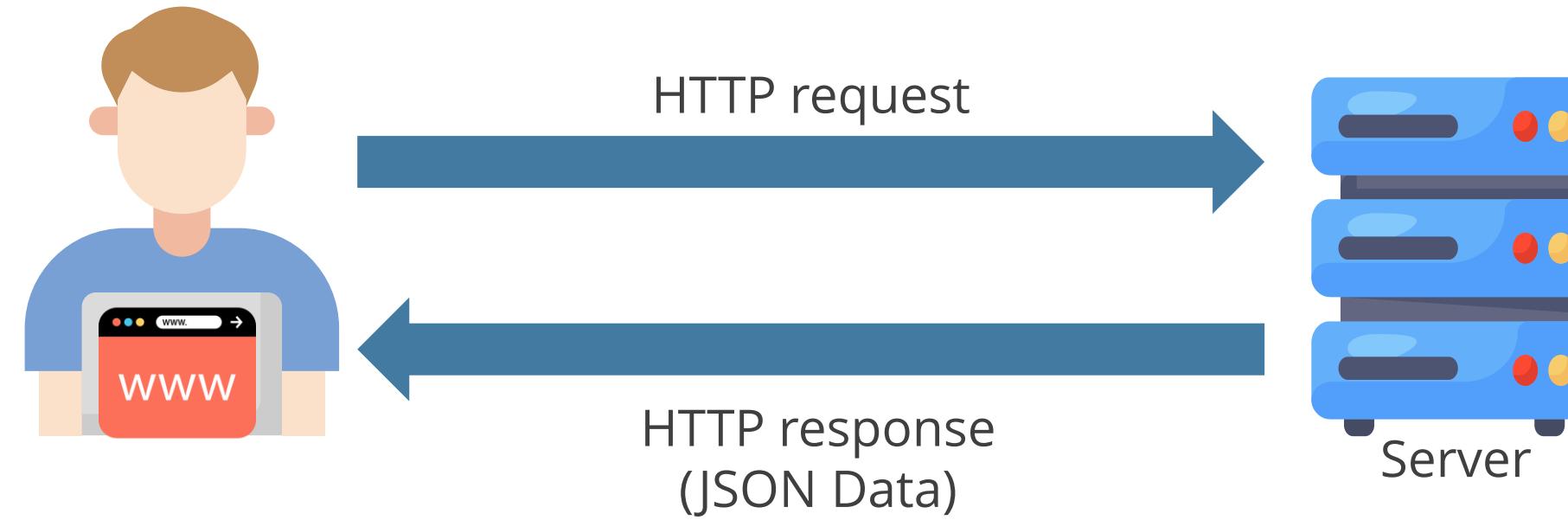
JSON stands for **JavaScript Object Notation**.



Douglas Crockford created the JSON format in the early 2000s.

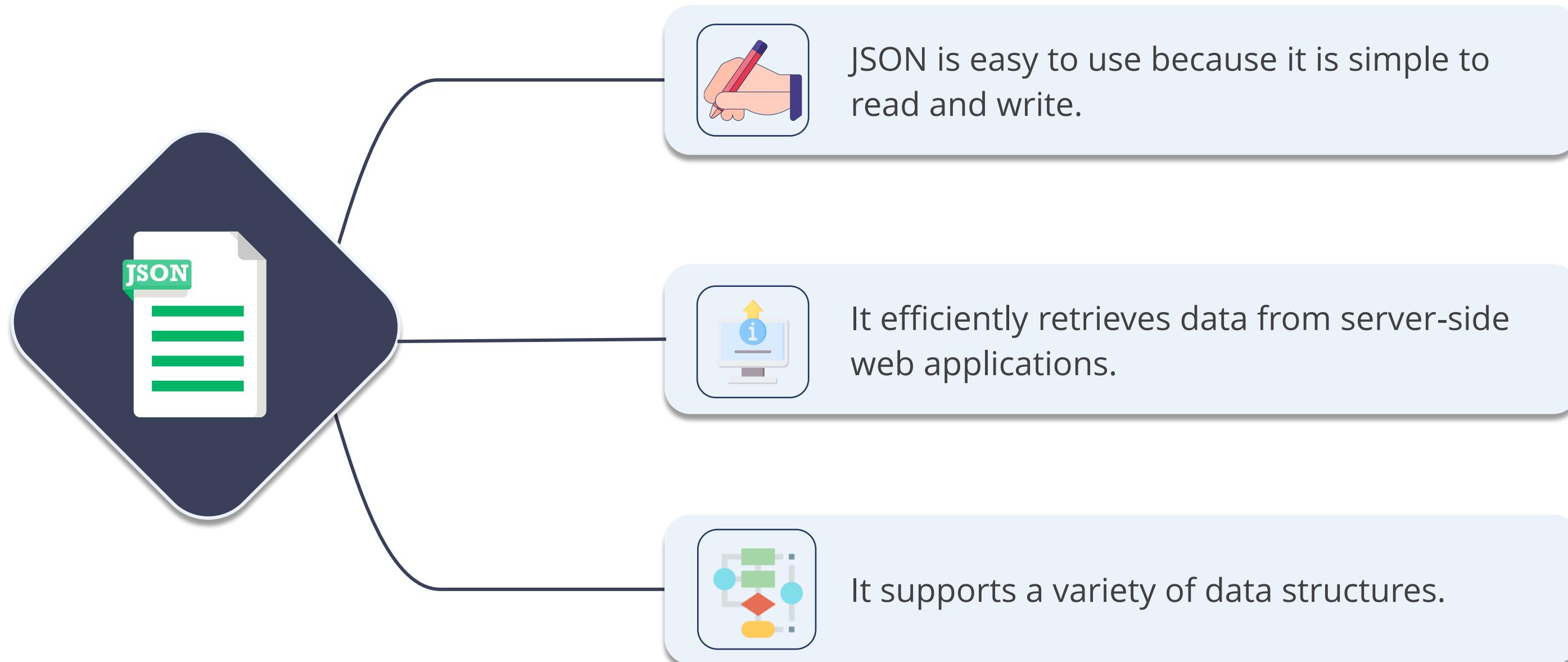
What Is JSON?

It is a format for storing and transmitting data. It facilitates data exchange between a browser and a server.



Features of JSON

JSON is easy to use as mentioned below:



JSON Supporting Languages

The JSON text format is entirely independent of programming languages:



Python



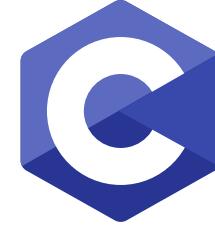
Ruby



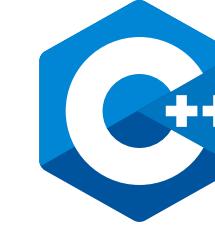
PHP



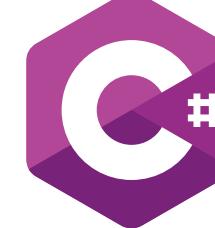
Java



C



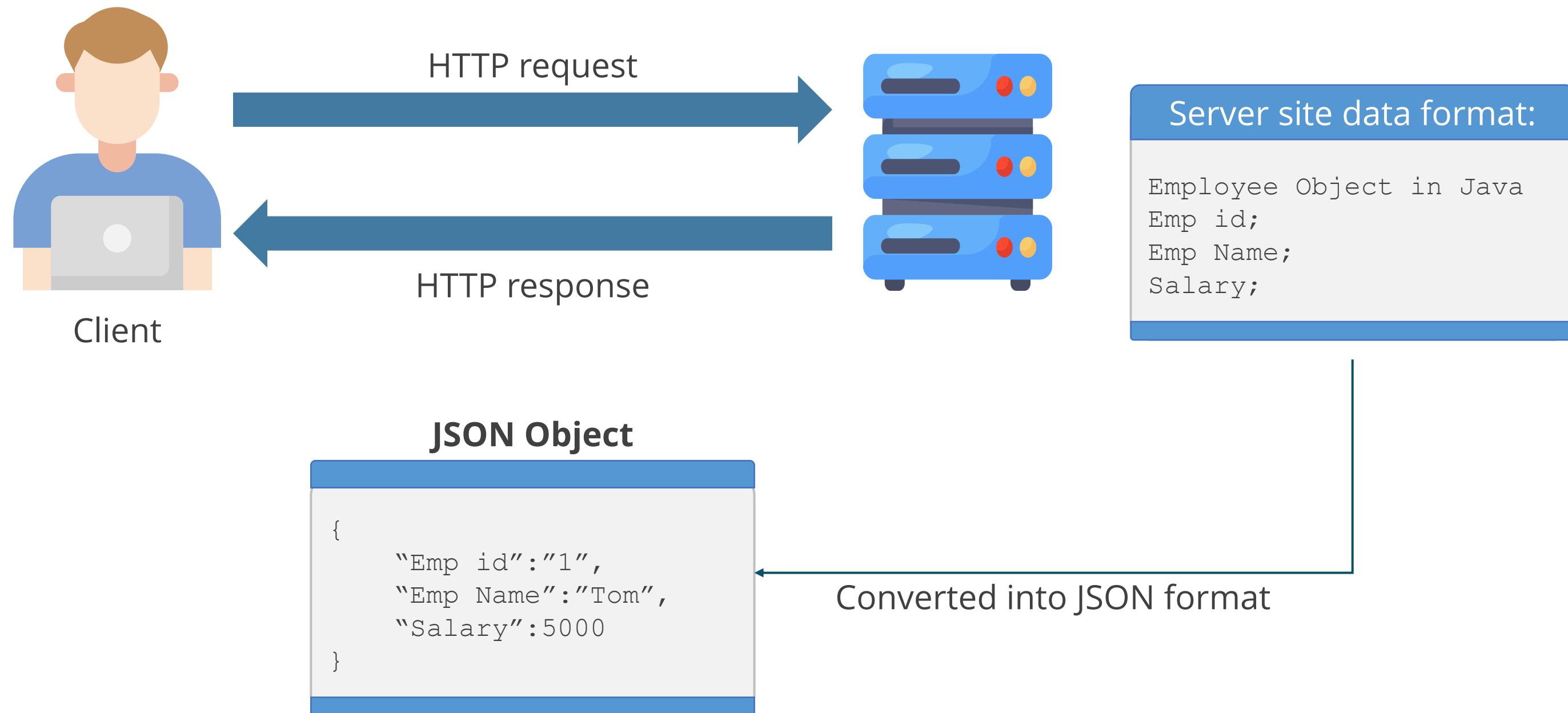
C++



C#

This independence makes it universally adaptable and usable across various programming environments.

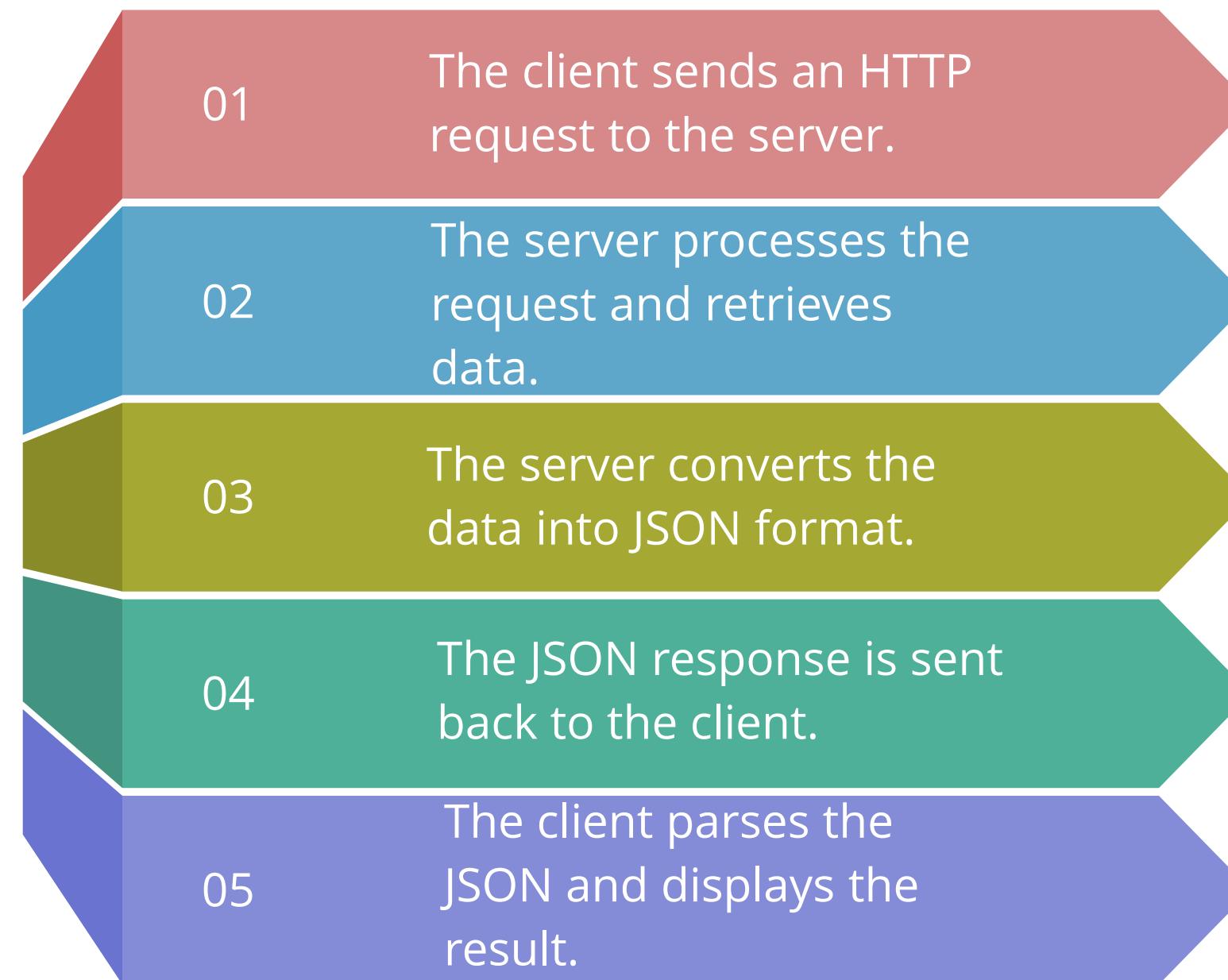
JSON: Architecture



JSON: Architecture

It enables structured data exchange between clients and servers, ensuring seamless communication.

Below mentioned is the flow of the architecture:



JSON Syntax

Syntax

```
{  
    "Emp_id" : "1",  
    "Emp_Name" : "Tom",  
    "Salary": 5000  
}
```

Data is represented as name-value pair.

Name must be in double quotes.

Data is separated by commas.



JSON files should have the file extension **.json**.

JSON: Data Types

JSON supports the following data types:



String

Represents text or Unicode double-quoted with backslash
escapement

For example:

{"First Name":"Ramesh"}

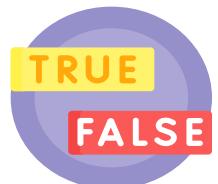


Number

Represents the actual number, integer, or floating number

For example:

{"Age":26}



Boolean

Represents the true or false value

For example:

{"Male":true}

JSON: Data Types

JSON supports the following data types:



Null

Represents the absence of a value or a deliberate nonexistence

For example:

`{“Age”: null, “Company”: null, “car”: null }`



Object

Represents a collection of key-value pairs, separated by commas and enclosed in curly brackets

For example: `{“name”: “Ricky”, “age”:25, “car”: “BMW”}`



Array

Represents a structure with zero, one, or more ordered elements, each separated by a comma

For example:

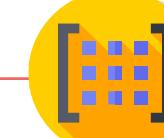
`["Jan", “Feb”, “Mar”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, “Dec”]`

JSON: Object and Arrays



Objects

An object is a collection of name or value pairs.



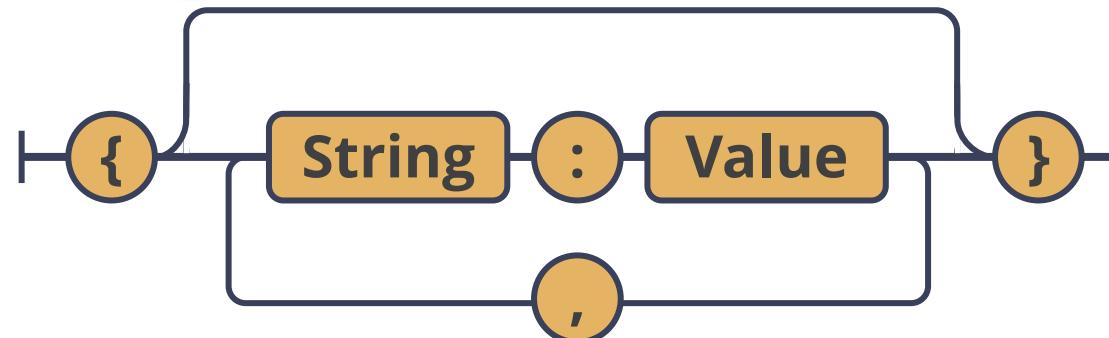
Arrays

An array is an ordered list of values.

In most programming languages, objects are known by various names, including object, record, struct, dictionary, hash table, keyed list, or associative array.

In most programming languages, arrays are known as an array, vector, list, or sequence.

JSON Object

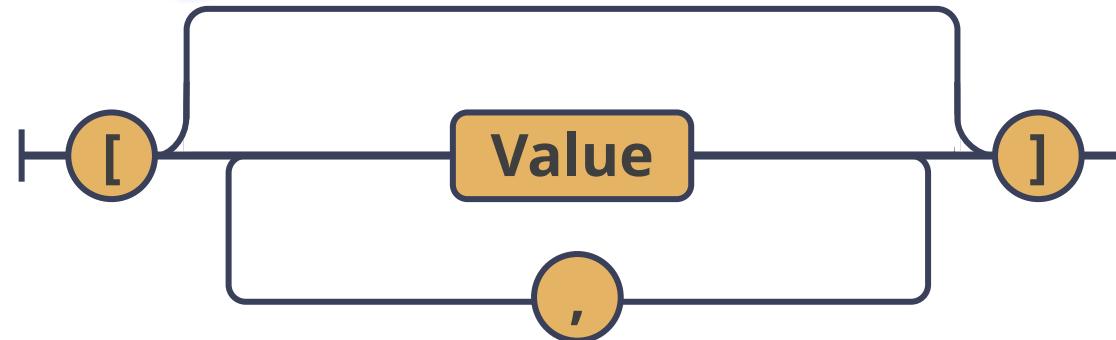


Example

```
{  
  Languages:  
  [  
    { "Name":"Python"  
      "Version":3.11.0  
    }  
    { "Name":"Java"  
      "Version":19.0  
    }  
  ]  
}
```

- JSON object starts with a left curly brace ({) and ends with a right curly brace (}).
- An object is an unordered set of key-value pairs.
- Each name is followed by a colon (:).
- Key-value pairs are separated by a comma (,).

JSON Array



Example

```
Employee:  
[  
  { "Name": "Jayesh"  
    "Age": 25  
  }  
  { "Name": "Kunal"  
    "Version": 28  
  }  
]
```

- JSON array starts with a left square brace (`[`) and ends with a right square brace (`]`).
- JSON array represents an ordered list of values.
- Values are separated by a comma (,`,`).

Object vs. Array

Feature	Object	Array
Structure type	Set of key-value pairs	List of values
Enclosing symbols	Enclosed in curly braces ({})	Enclosed in square brackets ([])
Separator used	Name-value pairs are separated by a comma (,)	Values are separated by a comma (,)
Key-value format	Each name is followed by a colon (:)	Not applicable
Value types allowed	Each value can be of the seven value types, including another object or array	Each value in an array can be of a different type, including another array or an object

Quick Check



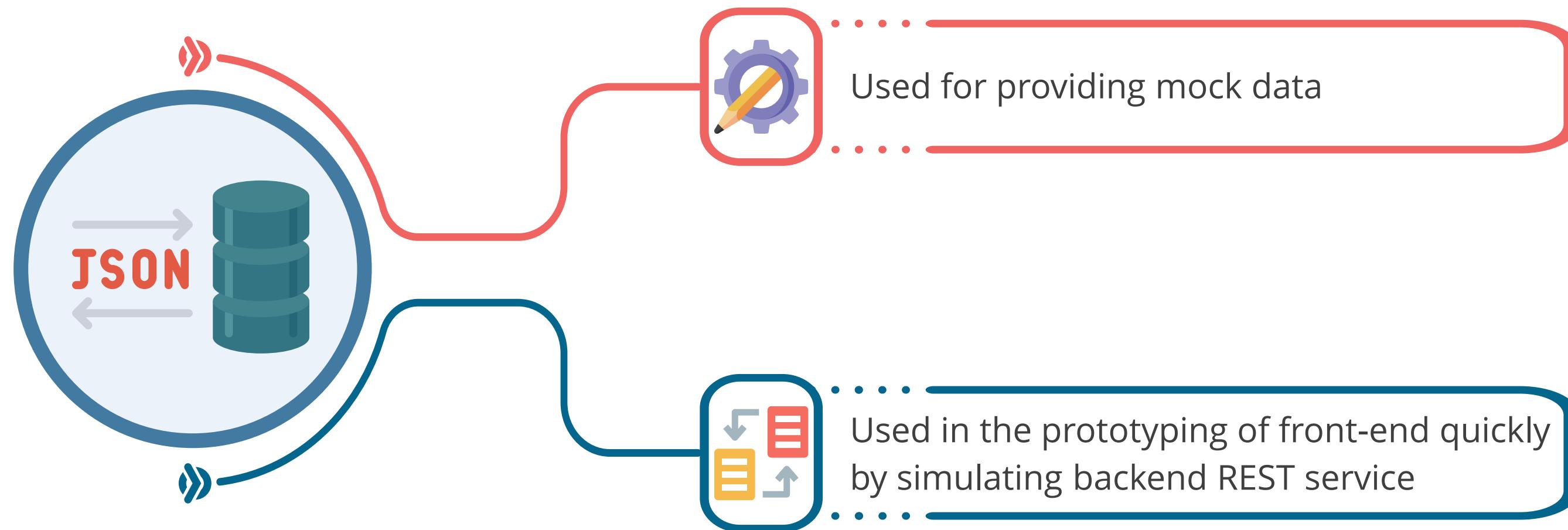
A company is developing a real-time stock market tracking application. The application must efficiently exchange structured data between the client (user interface) and the server. Which of the following explains why JSON is the best choice for this task?

- A. JSON is a lightweight and human-readable format, making data transfer efficient.
- B. JSON encrypts data automatically, ensuring high security during transmission.
- C. JSON is the only format that allows storing data in a structured way.
- D. JSON does not require parsing and can be directly used in all programming languages.

Overview of JSON Server

What Is JSON Server?

It is a npm (Node Package Manager) module used for creating fake REST API.



Benefits of Using JSON Server

It is a lightweight tool that simplifies the process of creating a mock REST API, making it useful for rapid development and testing.

Easy setup

Requires minimal configuration and can be installed quickly using npm

Mock REST API

Simulates a backend, allowing frontend developers to test APIs without a real database

Support CRUD operations

Enables data manipulation using standard HTTP methods (GET, POST, PUT, DELETE)

Lightweight and fast

Runs locally with minimal resource consumption, making it ideal for prototyping

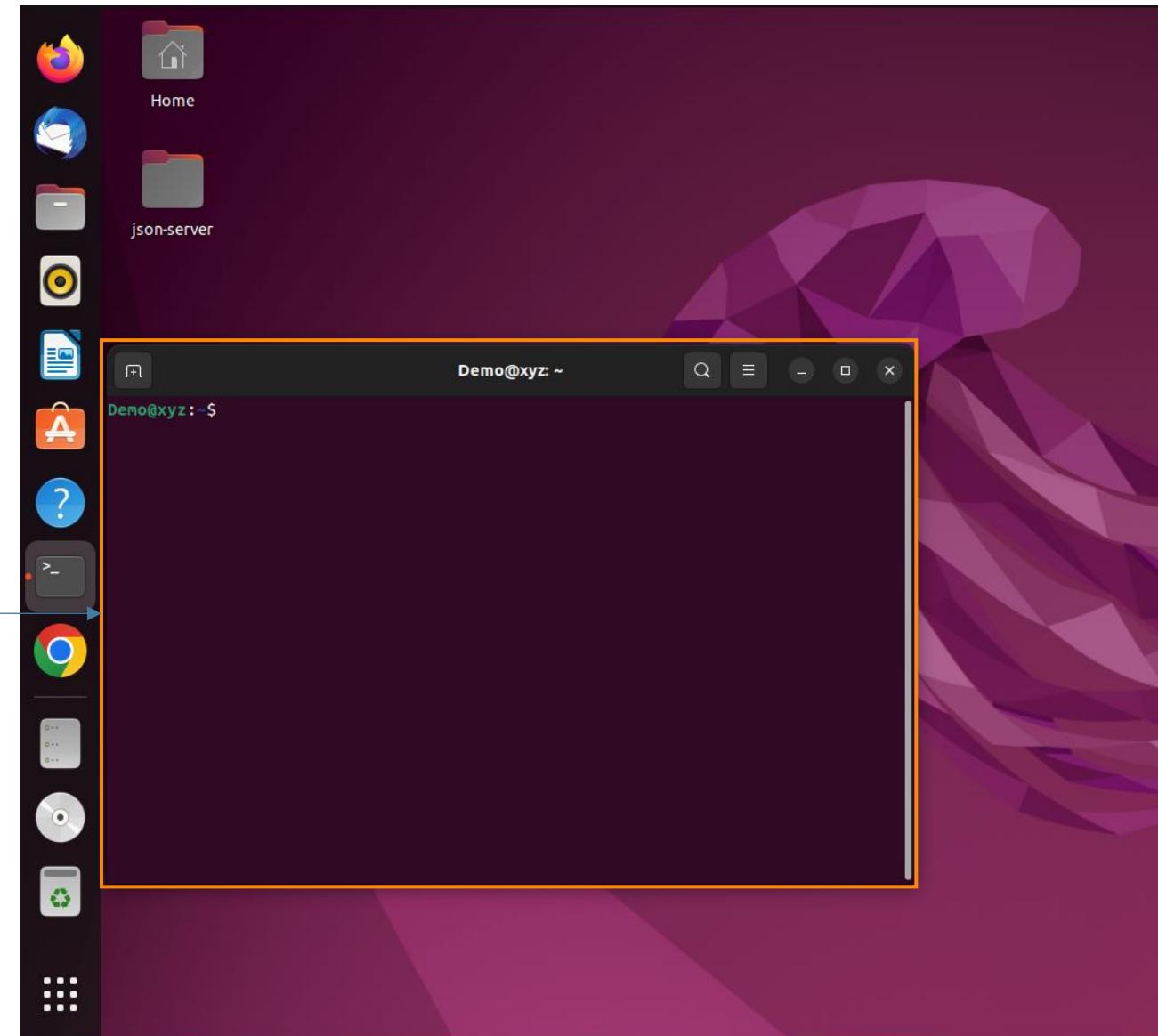
JSON-based storage

Uses a simple JSON file (db.json) as a database, eliminating the need for complex database management

Installing JSON Server

Step 1

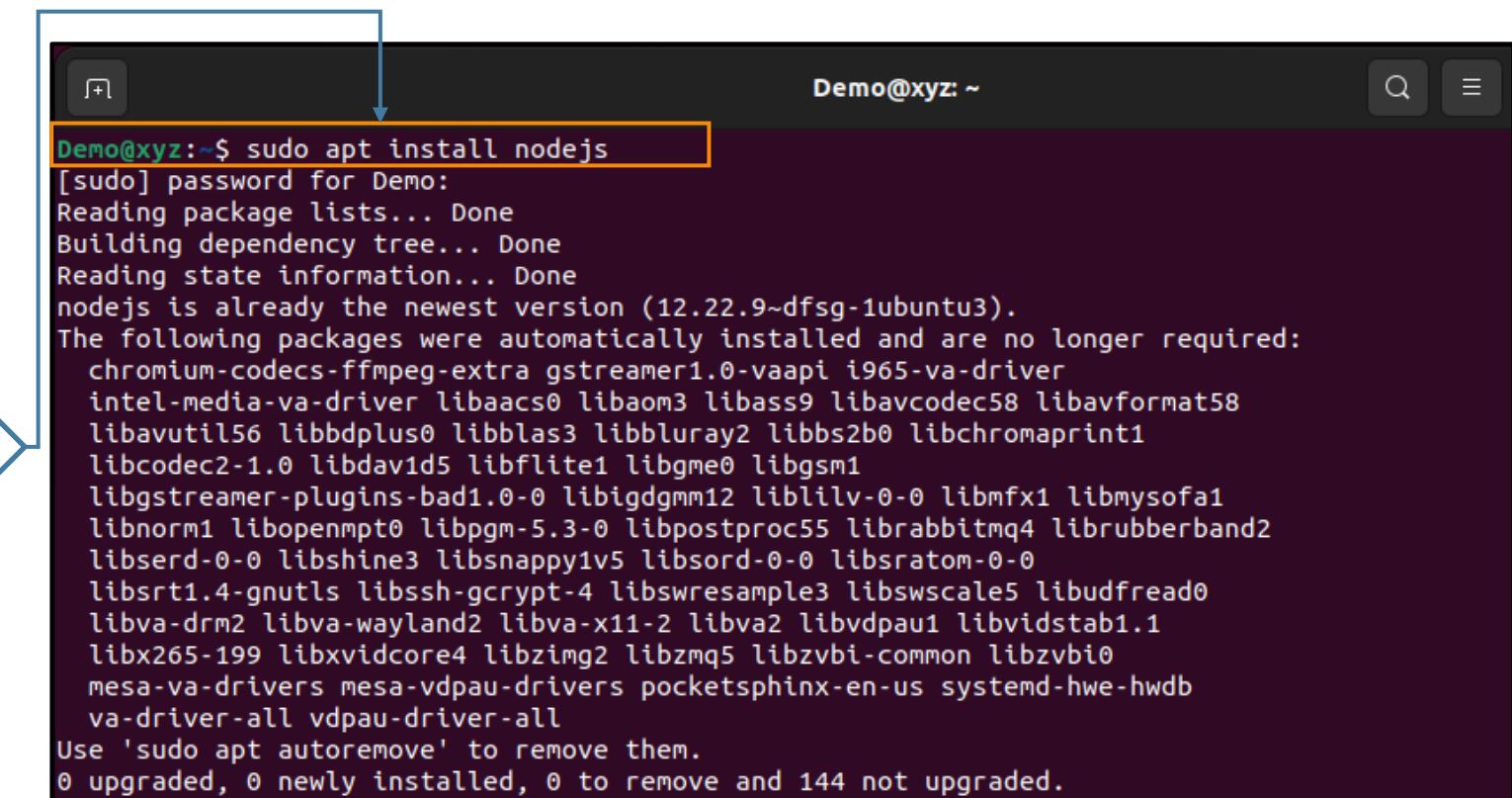
Users must open the **Terminal** on the system.



Installing JSON Server

Step 2

Type **sudo apt install nodejs** for installing nodejs

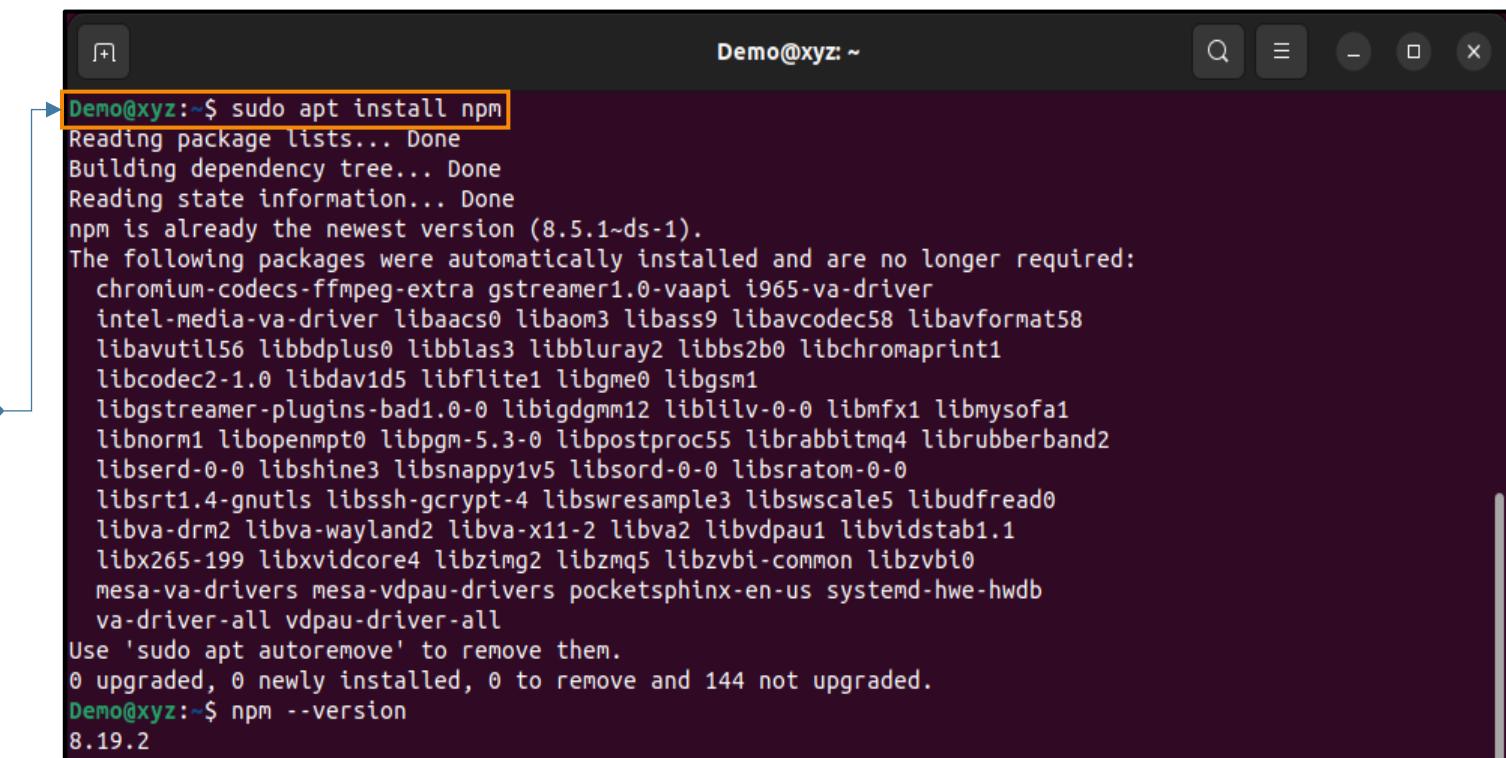


```
Demo@xyz:~$ sudo apt install nodejs
[sudo] password for Demo:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nodejs is already the newest version (12.22.9~dfsg-1ubuntu3).
The following packages were automatically installed and are no longer required:
  chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi i965-va-driver
  intel-media-va-driver libaacs0 libaom3 libass9 libavcodec58 libavformat58
  libavutil56 libbdplus0 libblas3 libbluray2 libbs2b0 libchromaprint1
  libcodec2-1.0 libdav1d5 libflite1 libgme0 libgsm1
  libgstreamer-plugins-bad1.0-0 libigdgmm12 liblilv-0-0 libmfx1 libmysofa1
  libnorm1 libopenmpt0 libpgm-5.3-0 libpostproc55 librabbitmq4 librubberband2
  libserd-0-0 libshine3 libsnappy1v5 libsord-0-0 libsratom-0-0
  libsrt1.4-gnutls libssh-gcrypt-4 libswresample3 libswscale5 libudfread0
  libva-drm2 libva-wayland2 libva-x11-2 libva2 libvdpa1 libvidstab1.1
  libx265-199 libxvidcore4 libzimg2 libzmq5 libzvbi-common libzvbi0
  mesa-va-drivers mesa-vdpau-drivers pocketsphinx-en-us systemd-hwe-hwdb
  va-driver-all vdpau-driver-all
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 144 not upgraded.
```

Installing JSON Server

Step 3

Type **sudo apt install npm** to install node package manager



```
Demo@xyz:~$ sudo apt install npm
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
npm is already the newest version (8.5.1~ds-1).
The following packages were automatically installed and are no longer required:
chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi i965-va-driver
intel-media-va-driver libaacs0 libaom3 libass9 libavcodec58 libavformat58
libavutil56 libbdplus0 libblas3 libbluray2 libbs2b0 libchromaprint1
libcodec2-1.0 libdav1d5 libflite1 libgme0 libgsml
libgstreamer-plugins-bad1.0-0 libigdgmm12 liblilv-0-0 libmfx1 libmysofa1
libnorm1 libopenmpt0 libpgm-5.3-0 libpostproc55 librabbitmq4 librubberband2
libserd-0-0 libshine3 libsnappy1v5 libsord-0-0 libsratom-0-0
libsrt1.4-gnutls libssh-gcrypt-4 libswresample3 libswscale5 libudfread0
libva-drm2 libva-wayland2 libva-x11-2 libva2 libvpdau1 libvidstab1.1
libx265-199 libxvidcore4 libzimg2 libzmq5 libzvbi-common libzvbi0
mesa-va-drivers mesa-vdpau-drivers pocketsphinx-en-us systemd-hwe-hwdb
va-driver-all vdpau-driver-all
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 144 not upgraded.
Demo@xyz:~$ npm --version
8.19.2
```

Installing JSON Server

Step 4

Type **sudo npm install -g n** for installing package

```
Demo@xyz:~$ sudo npm install -g n
changed 1 package, and audited 2 packages in 957ms
found 0 vulnerabilities
Demo@xyz:~$ sudo n lts
      copying : node/18.12.1
      installed : v18.12.1 (with npm 8.19.2)
```

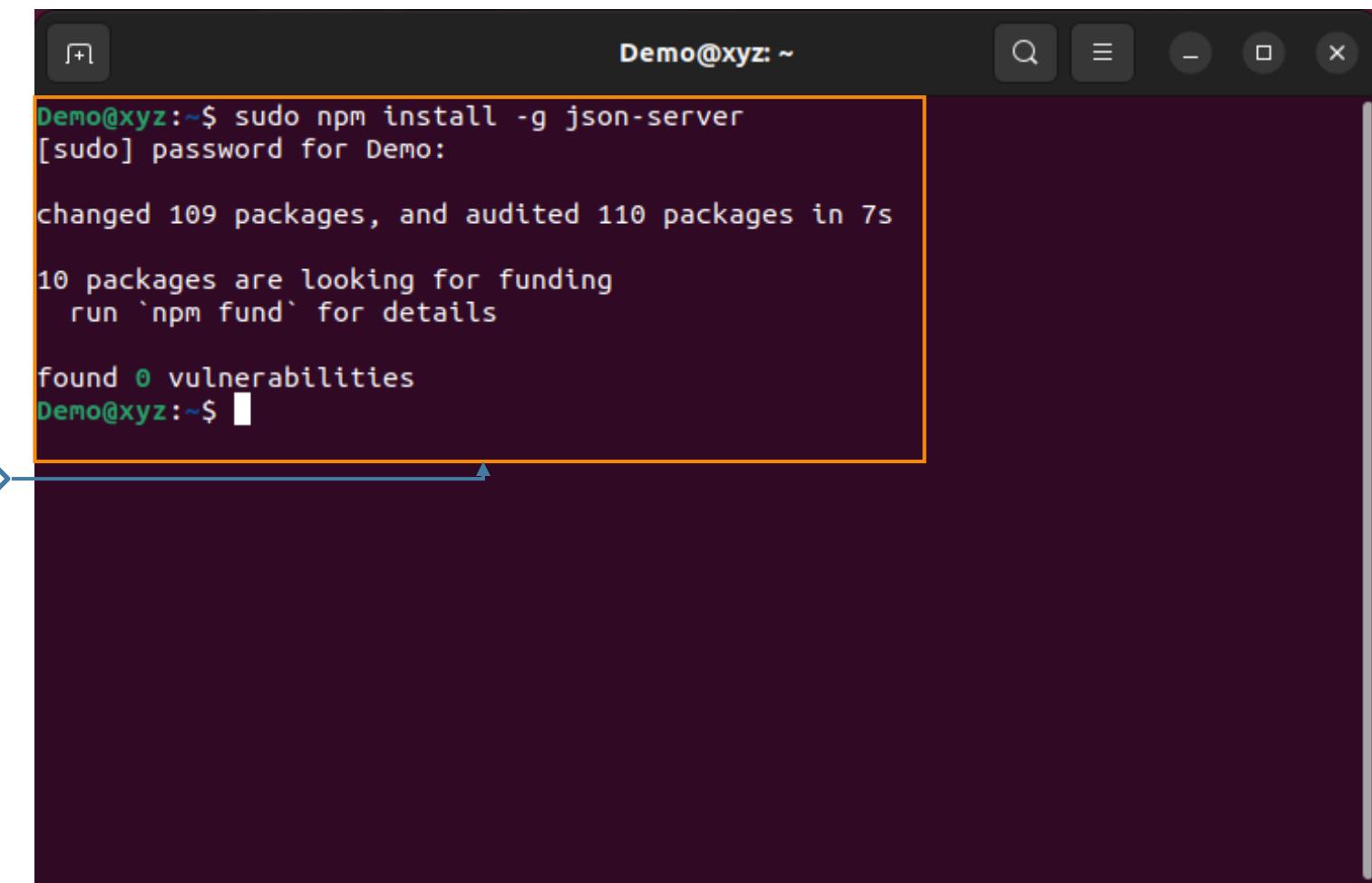
Step 5

Type **sudo n lts** for installing the latest node version

Installing JSON Server

Step 6

Type **sudo npm install -g json-server** to install the JSON Server



```
Demo@xyz:~$ sudo npm install -g json-server
[sudo] password for Demo:

changed 109 packages, and audited 110 packages in 7s

10 packages are looking for funding
  run `npm fund` for details

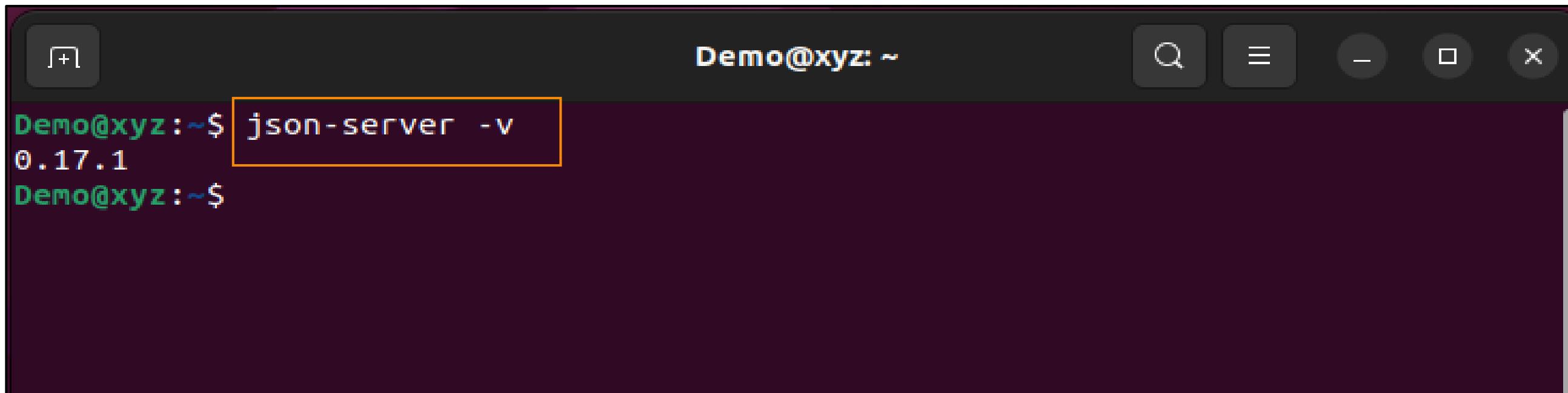
found 0 vulnerabilities
Demo@xyz:~$ █
```

A screenshot of a terminal window titled "Demo@xyz: ~". The window shows the command "sudo npm install -g json-server" being run. The output indicates that 109 packages were changed and 110 packages were audited in 7 seconds. It also mentions 10 packages looking for funding and found 0 vulnerabilities. The terminal has a dark background with light-colored text and a blue border around the main text area.

Checking JSON Server Version

json-server -v

Used to check the JSON Server version



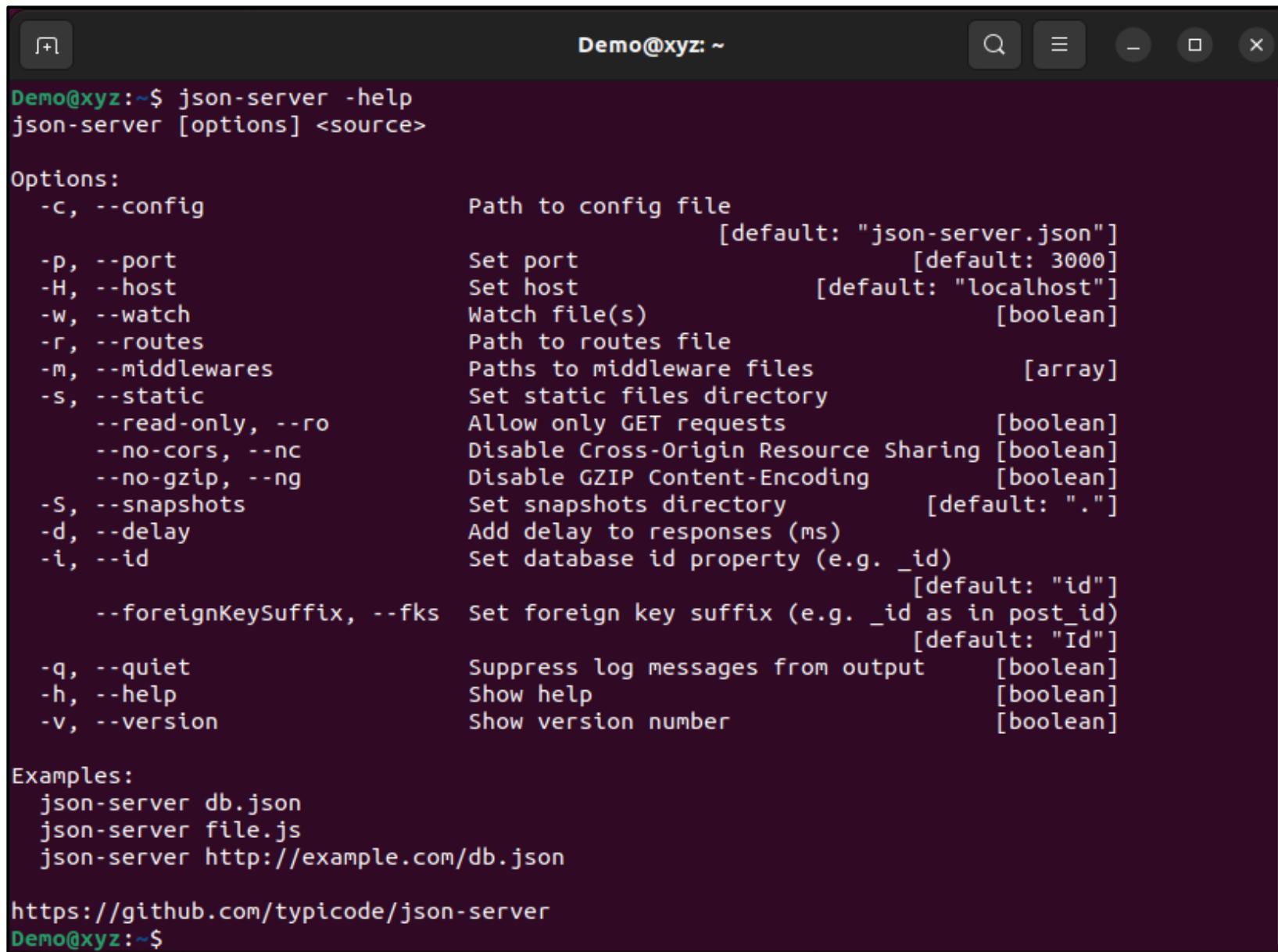
A screenshot of a terminal window titled "Demo@xyz: ~". The window has a dark theme with light-colored icons. In the terminal, the command "json-server -v" is run, resulting in the output "0.17.1". The command itself is highlighted with a yellow box.

```
Demo@xyz:~$ json-server -v
0.17.1
Demo@xyz:~$
```

Checking JSON Server Version

json-server -help

Used to show all the options of JSON Server



Demo@xyz:~\$ json-server -help
json-server [options] <source>

Options:

-c, --config	Path to config file [default: "json-server.json"]
-p, --port	Set port [default: 3000]
-H, --host	Set host [default: "localhost"]
-w, --watch	Watch file(s) [boolean]
-r, --routes	Path to routes file
-m, --middlewares	Paths to middleware files [array]
-s, --static	Set static files directory
--read-only, --ro	Allow only GET requests [boolean]
--no-cors, --nc	Disable Cross-Origin Resource Sharing [boolean]
--no-gzip, --ng	Disable GZIP Content-Encoding [boolean]
-S, --snapshots	Set snapshots directory [default: "."]
-d, --delay	Add delay to responses (ms)
-i, --id	Set database id property (e.g. _id) [default: "id"]
--foreignKeySuffix, --fks	Set foreign key suffix (e.g. _id as in post_id) [default: "Id"]
-q, --quiet	Suppress log messages from output [boolean]
-h, --help	Show help [boolean]
-v, --version	Show version number [boolean]

Examples:

- json-server db.json
- json-server file.js
- json-server http://example.com/db.json

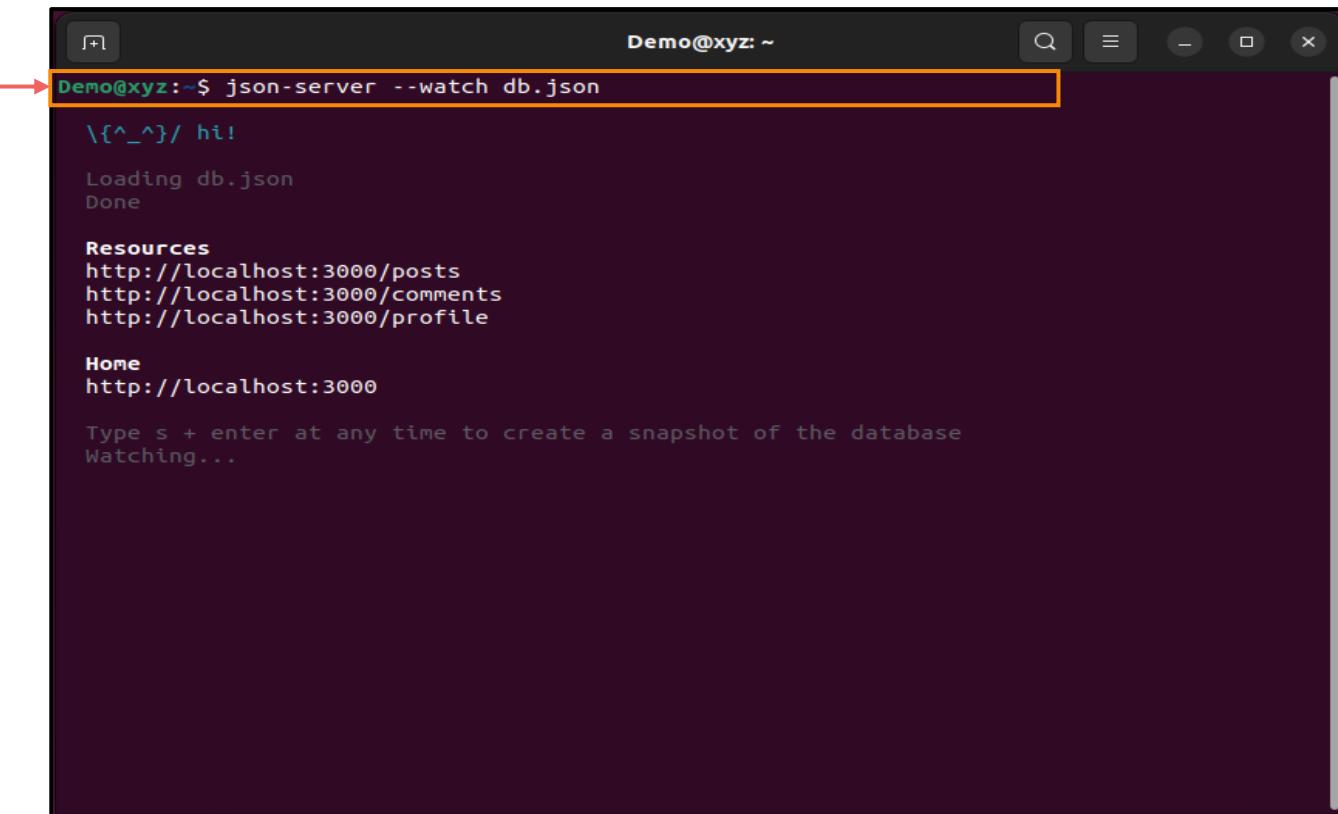
<https://github.com/typicode/json-server>

Demo@xyz:~\$

Running JSON Server

Step 1

Type **json-server --watch db.json** for running the JSON Server



Demo@xyz:~\$ json-server --watch db.json

\{^_^\}/ hi!

Loading db.json

Done

Resources

http://localhost:3000/posts
http://localhost:3000/comments
http://localhost:3000/profile

Home

http://localhost:3000

Type s + enter at any time to create a snapshot of the database

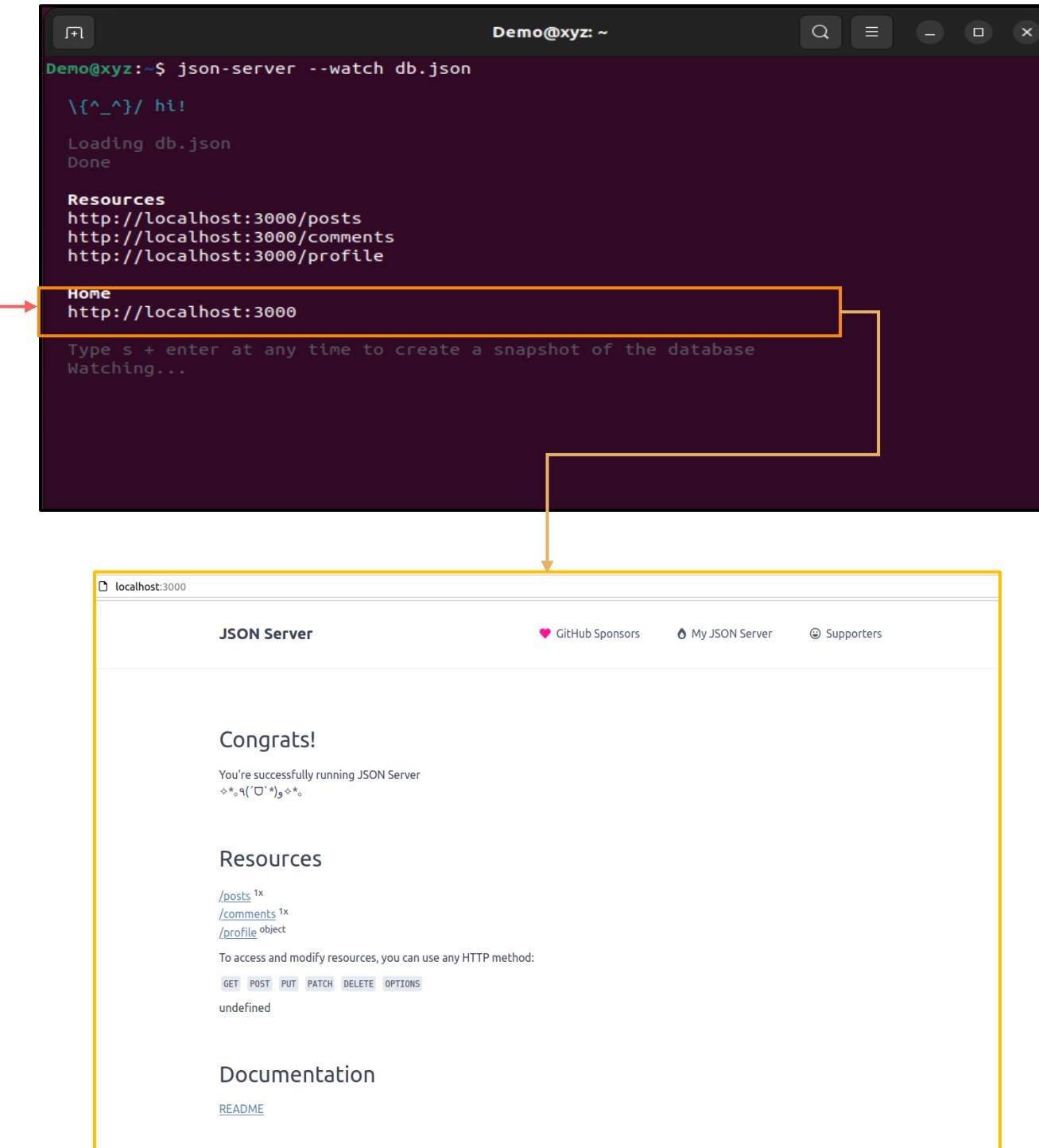
Watching...

A red arrow points from the text "Type json-server --watch db.json for running the JSON Server" to the terminal window.

Running JSON Server

Step 2

Press **ctrl + left click** on the Home URL to open the **JSON Server file**



Running JSON Server

Step 3

Click on **/post^{1x}** to open the **JSON file data**, which is automatically created

The image shows two screenshots illustrating the JSON Server interface. The top screenshot is the JSON Server dashboard, featuring a 'Congrats!' message and a list of resources: '/posts' (1x), '/comments' (1x), and '/profile' (object). Below this is a note about accessing resources via various HTTP methods (GET, POST, PUT, PATCH, DELETE, OPTIONS). The bottom screenshot shows a browser window displaying the JSON data for '/posts'. The browser's address bar shows 'localhost:3000/posts'. The JSON response is as follows:

```
id: 1
title: "json-server"
author: "typicode"
```

Using Curl to Interact with JSON Server

curl is a command-line tool used to send HTTP requests to a server, helping test API endpoints directly without needing a browser or frontend application.

Below are its key uses and benefits in the context of JSON Server:

curl supports various HTTP methods like GET, POST, PUT, and DELETE.

It helps verify if your JSON Server API endpoints are working correctly.

Ideal for quick testing of backend functionality during development.

Responses from the server are displayed directly in the console.



Quick Check



You are working on a prototype for an e-commerce website and need a lightweight backend to store and manage product data without setting up a full database. You decide to use JSON Server for quick API development. Which of the following is the correct approach to install and run JSON Server?

- A. Download and install JSON Server from the official website, then run it using the browser.
- B. Install JSON Server using npm, create a JSON file for data storage, and start the server using a command.
- C. Install JSON Server with Python's pip package manager and configure it in Django.
- D. Use a MySQL database and configure it to work with JSON Server for data storage.

Assisted Practice



Creating and Using a JSON Server File

Duration: 15 Min.

Problem statement:

You have been assigned a task to create a JSON file with sample data and use json-server and curl to retrieve and display the user data in the console.

Outcome:

By the end of this demo, you will be able to create a JSON file, simulate a REST API using json-server, and retrieve data using curl for frontend testing.

Note: Refer to the demo document for detailed steps:
[01_Creating_and_Using_a_JSON_Server_File](#)

Assisted Practice: Guidelines



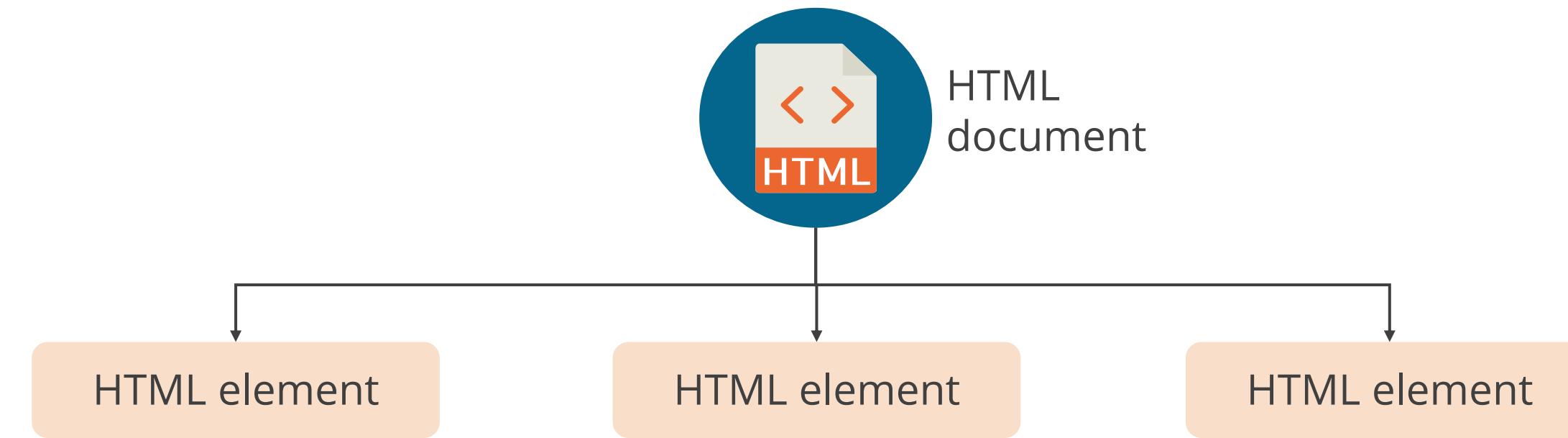
Steps to be followed:

1. Create a db.json file with user data
2. Install and run json-server using the db.json file
3. Use curl to retrieve and print the user data in the console

Understanding HTML Elements

HTML Element

It specifies how a web browser should display the related data.



Each element consists of a start tag and an end tag, with the content inserted in between.

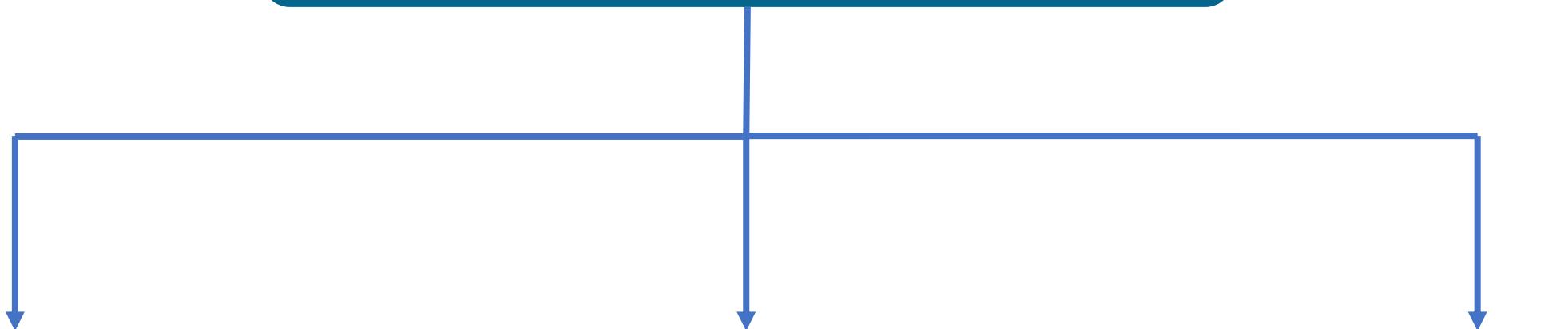
HTML Elements: Types

There are three types of HTML elements:

Normal elements

Void elements

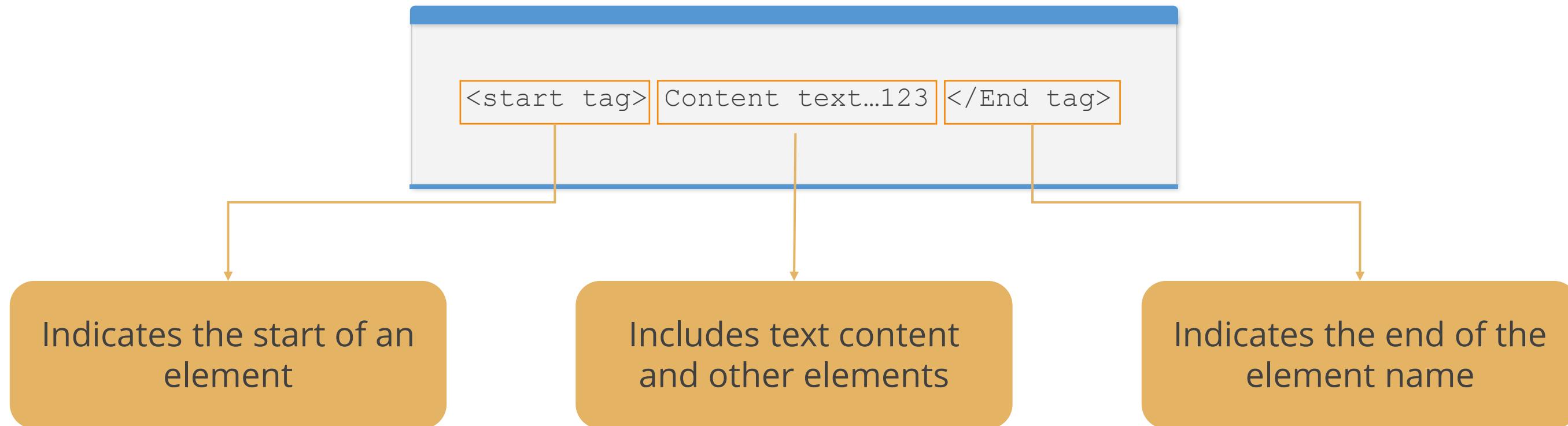
Raw text elements



Normal Elements

It contains a start and an end tag; one or both tags can be omitted for some elements.

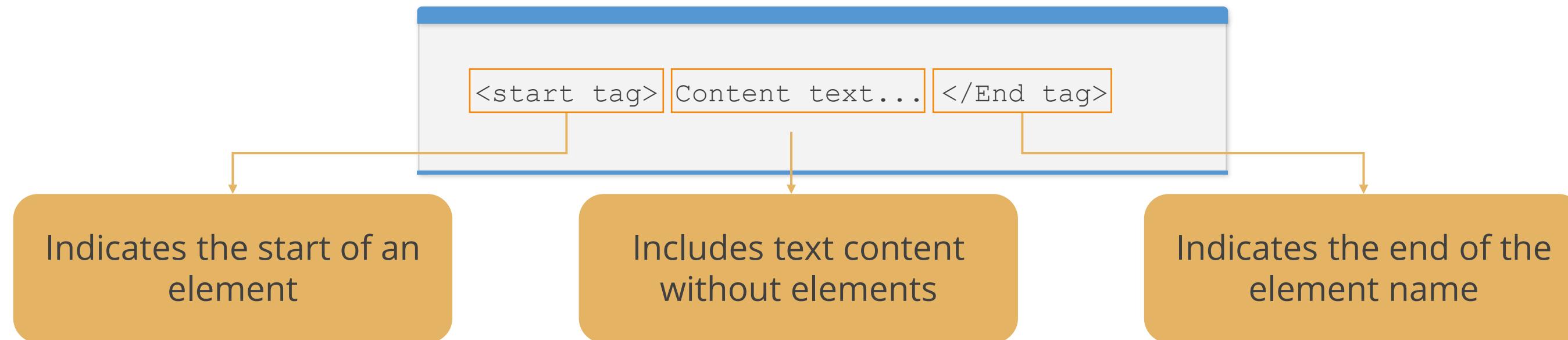
They are written as follows:



Raw Text Elements

It is also referred to as a text-only element.

They are written as follows:



Void Elements

It is also referred as empty or standalone element.

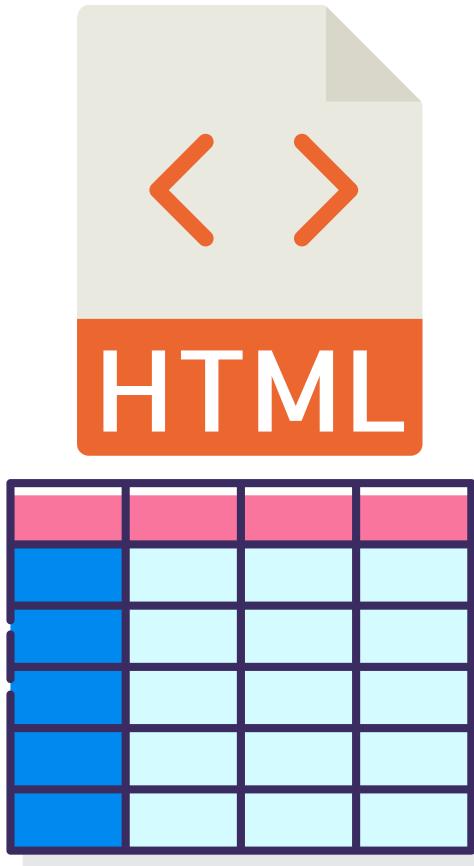
They are written as follows:

Contains a backslash before
the end of the start tag (this
is entirely optional)

<start tag/.....>

Contains only a start tag (in
the form <tag>) that
includes any HTML
attributes

Tables in HTML



01

Tables are specified by **<table>** tag.

02

Table headers are defined by **<th>** tag.

03

Table rows are defined by **<tr>** tag.

04

Table data or information of the row is defined by **<td>** tag.

Tables in HTML: Example

The following example shows how to create an HTML table:

Example

```
<table>
  <tr>
    <th>Employee ID</th>
    <th>Employee Name</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Alex</td>
  </tr>
  <tr>
    <td>2</td>
    <td>John</td>
  </tr>
</table>
```

Output:

Employee ID	Employee Name
1	Alex
2	John

Example of HTML Table with JSON Data

The following example shows how to create an HTML table with JSON data:

Example

```
<!DOCTYPE html>
<html>
<body>
<h3> HTML Table using JSON data.</h3>
<input type="button" value="Generate Table" onclick="GenerateTable()" />
<hr />
<div id="dvTable"></div>
<script type="text/javascript">
    function GenerateTable() {
        var cs = new Array();
        cs.push(["Customer Id", "Name", "Country"]);
        cs.push([1, "John", "Washington"]);
        cs.push([2, "Rohit", "India"]);
        cs.push([3, "Denis", "Paris"]);
        cs.push([4, "Eva", "Russia"]);
        var table = document.createElement("TABLE");
```

Example of HTML Table with JSON Data

The following example shows how to create an HTML table with JSON data:

Example

```
var columnCount = cs[0].length;
    var row = table.insertRow(-1);
    for (var i = 0; i < columnCount; i++) {
        var headerCell = document.createElement("TH");
        headerCell.innerHTML = cs[0][i];
        row.appendChild(headerCell);
    }
    for (var i = 1; i < cs.length; i++) {
        row = table.insertRow(-1);
        for (var j = 0; j < columnCount; j++) {
            var cell = row.insertCell(-1);
            cell.innerHTML = cs[i][j];
        }
    }
    var dvTable = document.getElementById("dvTable");
    dvTable.innerHTML = "";
    dvTable.appendChild(table);
}
</script>
```

Example of HTML Table with JSON Data

Output:

HTML Table using JSON data.

[Generate Table](#)

Customer Id Name Country

1	John	Washington
2	Rohit	India
3	Denis	Paris
4	Eva	Russia

Dynamic HTML Table

It changes the number of rows based on the input data.



A dynamic HTML table can be created using JavaScript.

Example of Dynamic HTML Table

The following example shows how to create the HTML table based on the value of a drop-down menu:

Example

```
<!DOCTYPE html>
<html>
<head>
    <h2>Convert JSON data to HTML</h2>
    <style>
        * { font-family: 'open sans' ; }
        select { font-size: 25px; } p { padding:
10px 0; }
    </style>
</head>
<body>
    <select id="sel"
onchange="selectText(this)">
        <option value="">-- Select --</option>
        <option value="Angular">Angular</option>
        <option
value="JavaScript">JavaScript</option>
        <option value="Git">Git</option>
    </select>
</body>
```

Continuation

```
<script>
    let selectText = (ele) => {
        let msg =
document.getElementById('msg');
        msg.innerHTML = 'Selected Text: <b>' +
ele.options[ele.selectedIndex].text + '</b>
<br>' +
                    'Value of the Selected Text: <b>' +
ele.value + '</b>';
    }
</script>
</html>
```

Example of Dynamic HTML Table

Output:

Convert JSON data to HTML

-- Select -- ▾

-- Select --

Angular

JavaScript

Git

Quick Check



You are creating a pricing table for a subscription-based streaming service. The table should display plan names, monthly prices, and features in a structured format. Which HTML elements should you use to create this table correctly?

- A. <table>, <tr>, <th>, and <td> to define the table structure properly
- B. <div> and to style and position each row and column
- C. and to list the plans and their features in a structured format
- D. <p> and
 to separate pricing details for better readability

Assisted Practice



Creating HTML Elements for JSON Server

Duration: 15 Min.

Problem statement:

You have been assigned a task to create an HTML file that displays a table with data from a JSON server.

Outcome:

By the end of this demo, you will be able to build an HTML page that fetches data from a JSON server and displays it in a table using CSS and JavaScript.

Note: Refer to the demo document for detailed steps:
[02_Creating_HTML_Elements_for_JSON_Server](#)

Assisted Practice: Guidelines



Steps to be followed:

1. Set up the HTML file
2. Integrate Java script for data fetching
3. Run and view the application

Assisted Practice



Creating a Form to Generate JSON Data

Duration: 15 Min.

Problem statement:

You have been assigned a task to develop an HTML form that takes user inputs and converts them into JSON format upon submission.

Outcome:

By the end of this demo, you will be able to create an HTML form that captures user input and converts it into structured JSON format for web applications.

Note: Refer to the demo document for detailed steps:
[03_Creating_a_Form_to_Generate_JSON_Data](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines



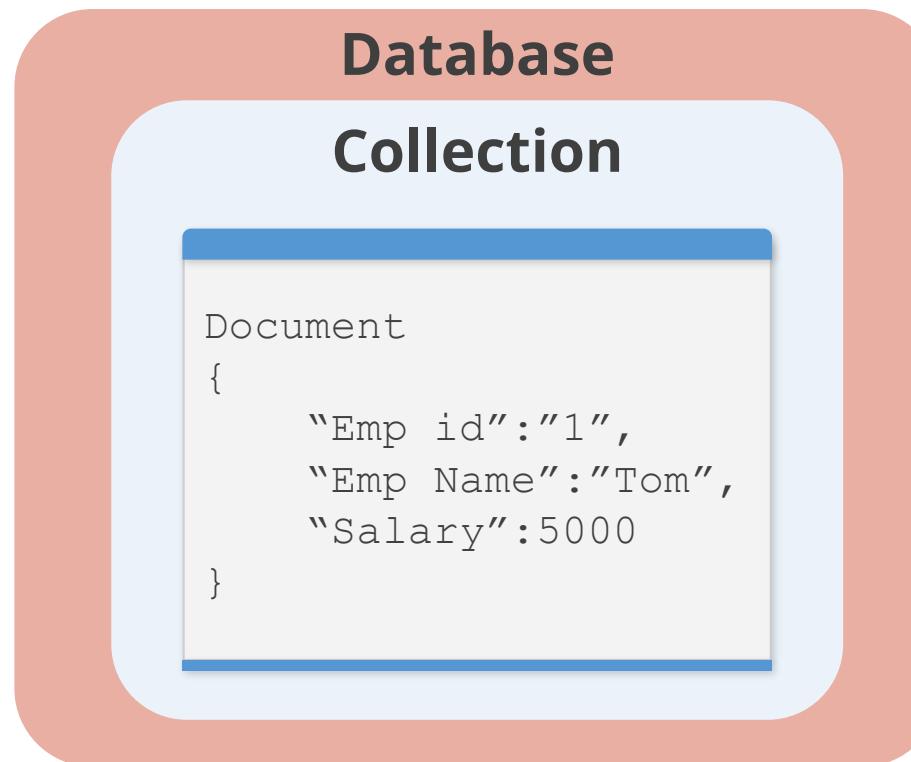
Steps to be followed:

1. Create an HTML file
2. Verify form submission output

Setting up a JSON DB

JSON DB: Introduction

It is a document type non-relational database, designed to store and query data as JSON documents.



It is a NoSQL database.

It stores semi-structured data.

It is more flexible than the SQL format.

Example

Popular JSON databases are MongoDB, Firestore, and Cosmos DB.

JSON DB: Example

Indexing and querying can make JSON data appear as a table.

Example

```
{  
  {  
    "Number": "S043659",  
    "Date": "2011-05-  
31T00.00.00"  
  
    "AccountNumber": "AW29825",  
    "Price": 59.99,  
    "Quantity": 1  
  }  
  {  
    "Number": "S043661",  
    "Date": "2011-06-  
01T00.00.00"  
  
    "AccountNumber": "AW73565",  
    "Price": 24.99,  
    "Quantity": 3  
  }  
}
```

Number	Date	Customer	Price	Quantity
S043659	2011-05-31T00.00.00	MSFT	59.99	1
S043661	2011-06-01T00.00.00	Nokia	24.99	3

Working with JSON Server

db.json

```
{  
  "employees": [  
    {  
      "Id" : "1",  
      "Emp Name" : "Tom",  
      "Salary": 5000  
    },  
    {  
      "id" : "2",  
      "Emp Name" : "John",  
      "Salary": 6000  
    }  
]  
}
```

The JSON structure contains:

- One object named employee
- Two datasets for the employee object
- Three properties for each employee object:
 - Id
 - Emp Name
 - Salary

Working with JSON Server

Type:

```
json-server --watch db.json  
for running the JSON-server
```

To access the database:

<http://localhost:3000/employees>

JSON Server will create the following end-points automatically:

- GET /employees
- GET /employees/{id}
- POST /employees
- PUT /employees/{id}
- DELETE /employees/{id}



Changes made using POST, PUT, and DELETE will be done automatically in the db.json file.

Working with JSON Server

The following methods are used while working on the JSON server:

GET

/employees+{Id} finds the employee by ID.

PUT

/employees updates an existing employee.

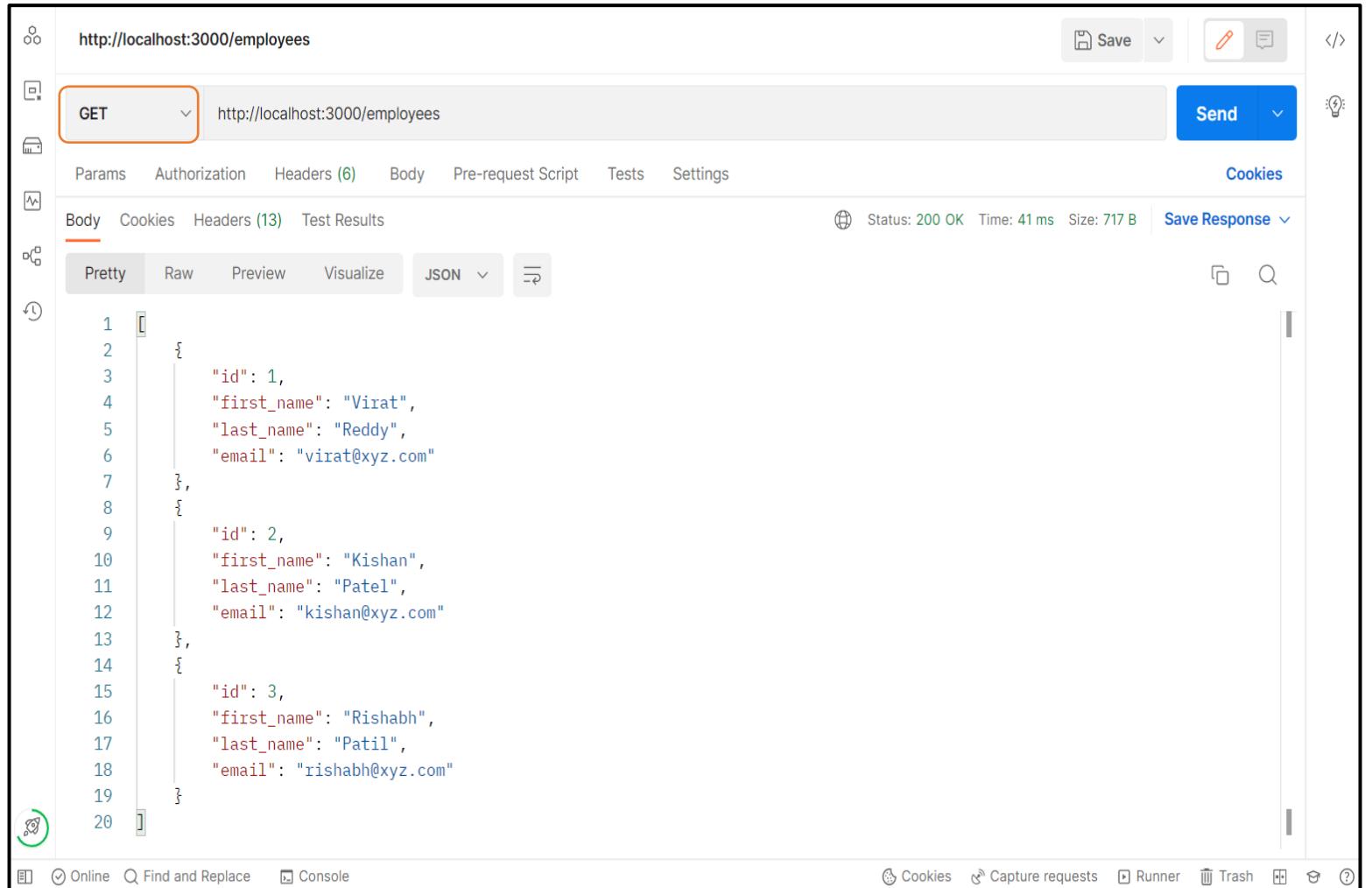
DELETE

/employees/{Id} deletes an employee by ID.

POST

/ employees /{Id}/uploadImage uploads an image.

JSON Server: GET



The screenshot shows the Postman application interface. A GET request is made to `http://localhost:3000/employees`. The response status is 200 OK, with a time of 41 ms and a size of 717 B. The response body is a JSON array containing three employees:

```
1
2 [
3   {
4     "id": 1,
5     "first_name": "Virat",
6     "last_name": "Reddy",
7     "email": "virat@xyz.com"
8   },
9   {
10    "id": 2,
11    "first_name": "Kishan",
12    "last_name": "Patel",
13    "email": "kishan@xyz.com"
14  },
15  {
16    "id": 3,
17    "first_name": "Rishabh",
18    "last_name": "Patil",
19    "email": "rishabh@xyz.com"
20 }
```

GET

Requests remain in the browser history.

GET

Requests can be bookmarked.

GET

Requests have length restrictions.

GET

Requests are only used to request data.

JSON Server: POST

The screenshot shows the Postman application interface. A POST request is being made to the URL `http://localhost:3000/employees`. The request method is selected in the dropdown. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "first_name": "Rohit",  
3   "last_name": "Shetty",  
4   "email": "rohit@xyz.com"  
5 }
```

The response status is 201 Created, and the response body is:

```
1 {  
2   "first_name": "Rohit",  
3   "last_name": "Shetty",  
4   "email": "rohit@xyz.com",  
5   "id": 4  
6 }
```

POST

Requests do not survive in the browser history.

POST

Requests cannot be bookmarked.

POST

Requests have no restrictions on data length.

JSON Server: PUT

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:3000/employees`. Below the URL, the method is selected as `PUT`. The request body is set to `http://localhost:3000/employees`. The `Body` tab is active, showing the following JSON data:

```
1 {  
2   "first_name": "Virat",  
3   "last_name": "Reddy",  
4   "email": "virat@xyz.com"  
5 }
```

Below the body, the response status is shown as `201 Created`. The response body contains the following JSON data:

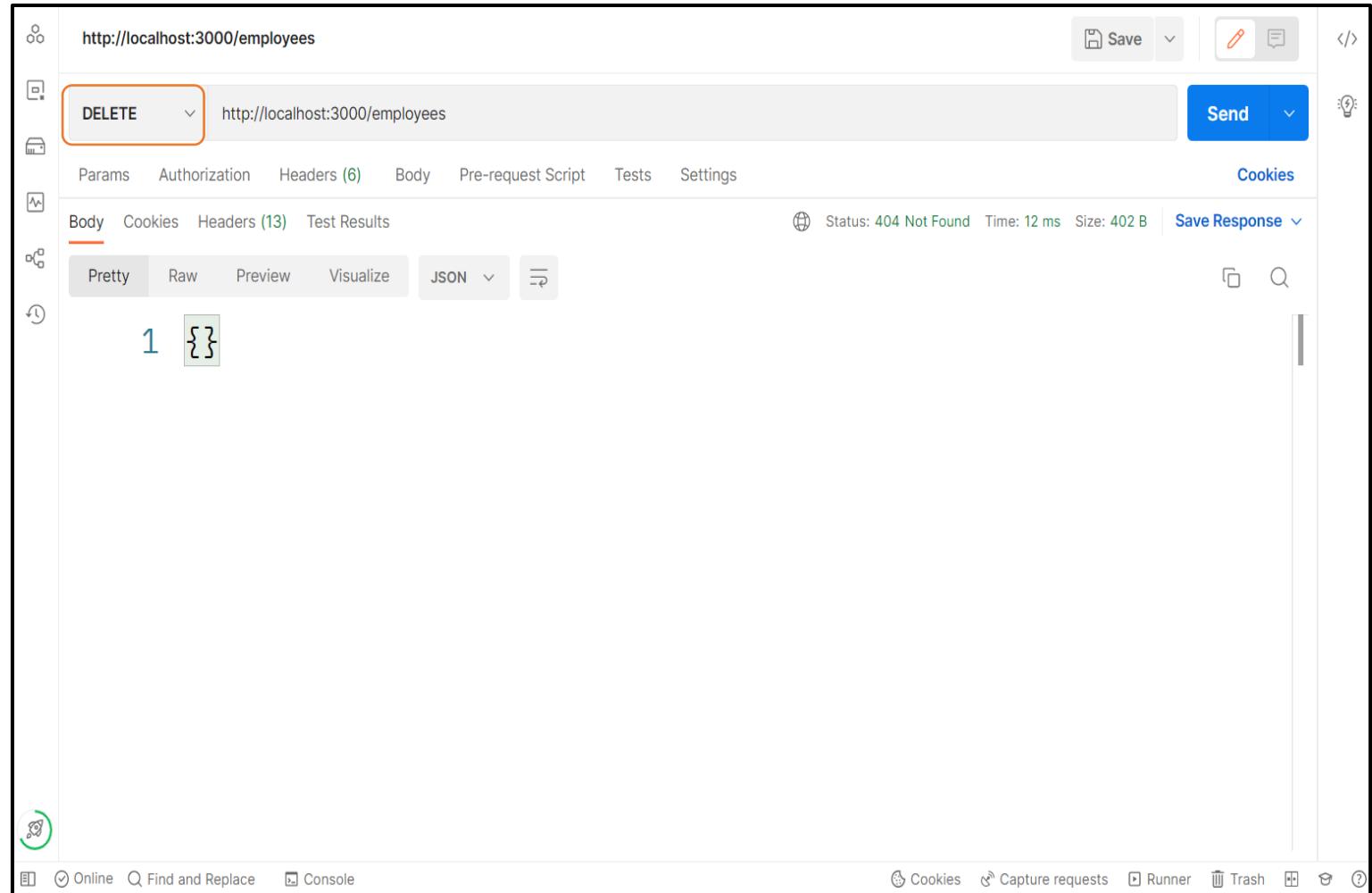
```
1 {  
2   "first_name": "Virat",  
3   "last_name": "Reddy",  
4   "email": "virat@xyz.com",  
5   "id": 5  
6 }
```

At the bottom of the interface, there are various navigation and settings buttons.

PUT

Relays data to a server to create or update a resource

JSON Server: DELETE



DELETE Deletes a resource from the server

Quick Check



You are developing a student management system using JSON DB. The system should allow users to add, update, retrieve, and modify student records through an API. Which HTTP method should you use to update an existing student's details in JSON DB?

- A. GET: Retrieve student details from the database
- B. POST: Add a new student record to the database
- C. PUSH: Send an update request to the database
- D. PUT: Update the existing student record with new details

Assisted Practice



Performing CRUD Methods in JSON DB

Duration: 15 Min.

Problem statement:

You have been assigned a task to perform CREATE, READ, UPDATE, and DELETE operations on the JSON server using different HTTP methods and endpoints via curl.

Outcome:

By the end of this demo, you will be able to perform CRUD operations on a JSON database using curl to interact with a REST API efficiently.

Note: Refer to the demo document for detailed steps:
[04_Performing CRUD Methods in a JSON Database](#)

Assisted Practice: Guidelines



Steps to be followed:

1. Create a new user in the users collection
2. Retrieve data for a specific user by their ID
3. Update data for a specific user by their ID
4. Delete a specific user by their ID

Advanced Data Handling and Configuration in JSON

Advanced Data Handling in JSON: Introduction

It refers to managing complex JSON data structures and performing dynamic operations such as filtering, merging, and transformation for real-world applications.

There are five key techniques used in advanced data handling:

1

Large datasets often use deeply nested structures to organize data, which has long been the industry standard.

2

Advanced operations include filtering keys, transforming values, and flattening structures.

3

Data merging enables combining multiple JSON files or API responses into one.

4

Conditional data access helps extract only the required subset of information.

5

Libraries and tools like jq, jsonpath, and language-specific parsers simplify processing.

Fetching Data Using JSON API

Users can fetch the JSON file's data.

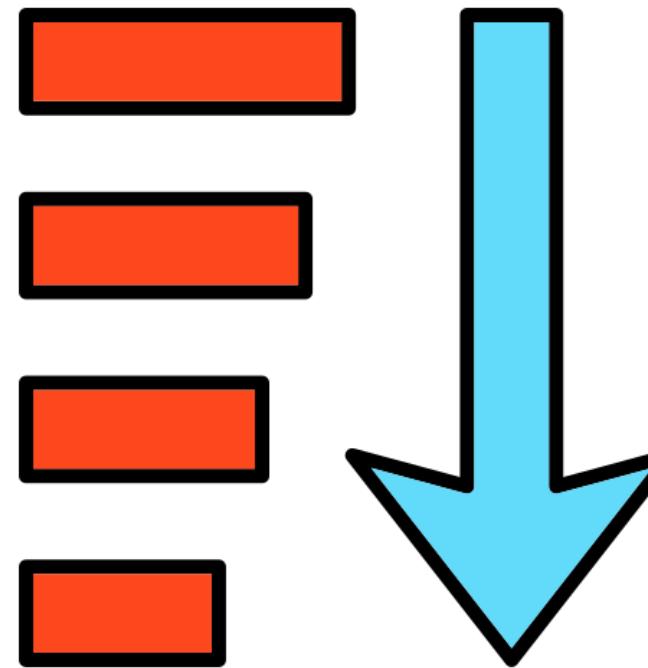
Fetch

- The **fetch()** method takes the path to the resource to be fetched and returns a promise that resolves with a **response** object.
- The response object is a representation of the HTTP response. The **json()** method is used to extract the JSON body content from the response object.

```
fetch('http://sample.com/movies.json')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

Sorting Data in JSON API

JSON API has the **sort** query parameter for sorting resources.



Sort

Users can specify how resources should be returned
in responses.

```
GET /employees?sort=id HTTP/1.1
```

Filtering Data in JSON API

JSON API has the filter query parameter family for filtering the retrieved data.

Filter query parameter family

filter, filter[x], filter[],
filter[x][], filter[], filter[x][y]



Filter

- Allows clients to search for resources and filter the number of resources returned in a response
- Is highly coupled with the application's logic
- Is exclusive of data storage

When Filtering Is Not Enough

Standard filtering works well for structured queries, but more dynamic needs call for full-text search.

Here are a few situations where filtering may fall short:

01

Filtering works with
exact field-value
matches.

03

Full-text search
handles natural
language queries.

02

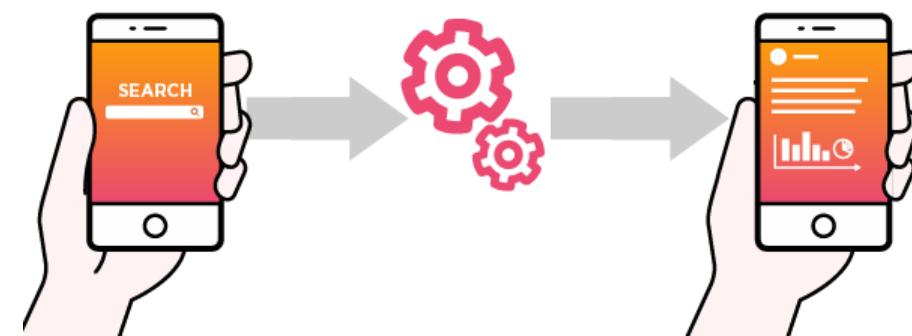
Keyword and partial
text searches need
more flexibility.

04

Improves search in
blogs, forums, and
apps.

Introduction to Full-Text Search

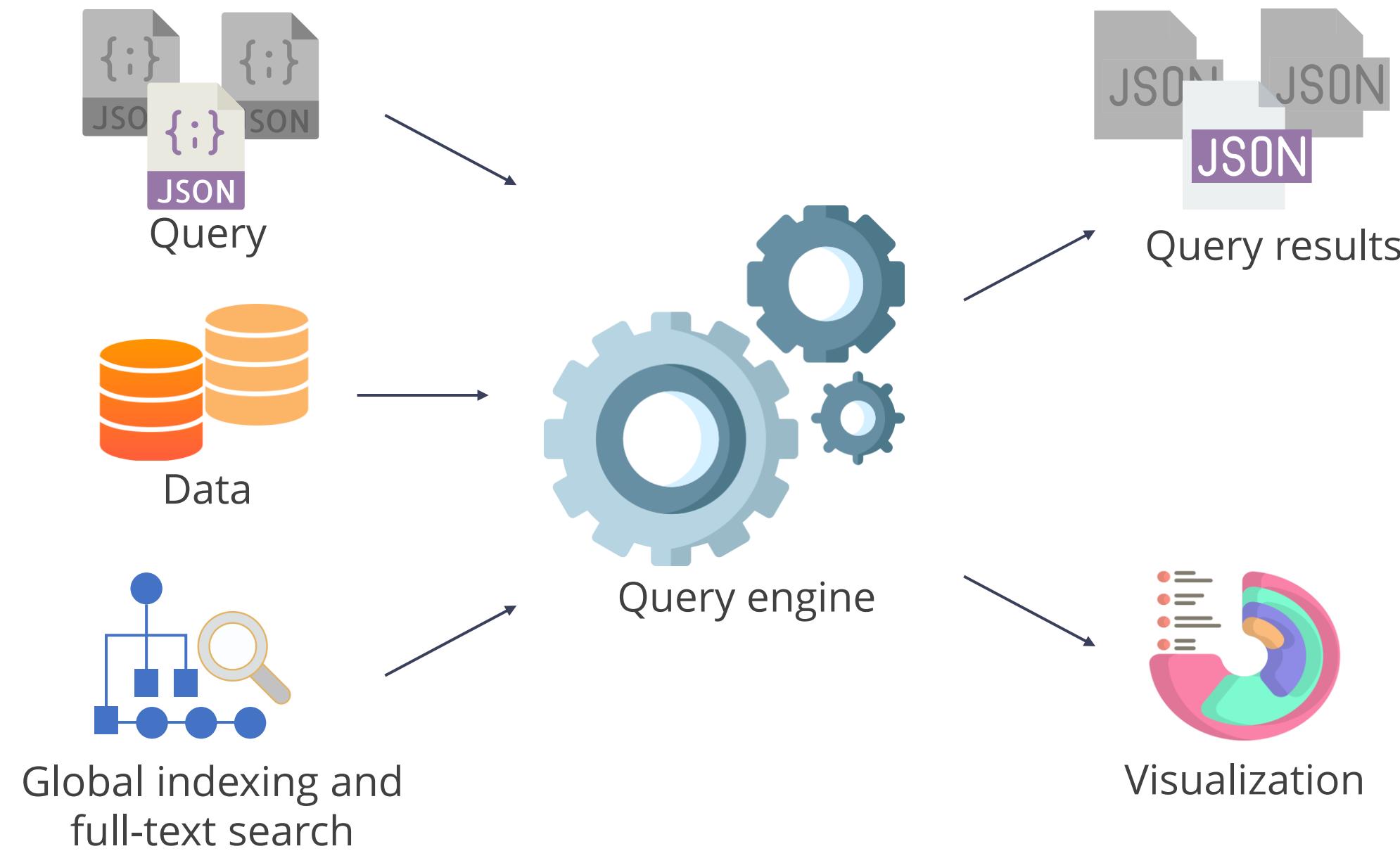
Full-text engine compiles full-text queries that can search a table for particular words or combinations of words.



- Stores information regarding significant words and their location within one or more columns of a database table.
- Uses contextual queries to simply search the entire content of the project

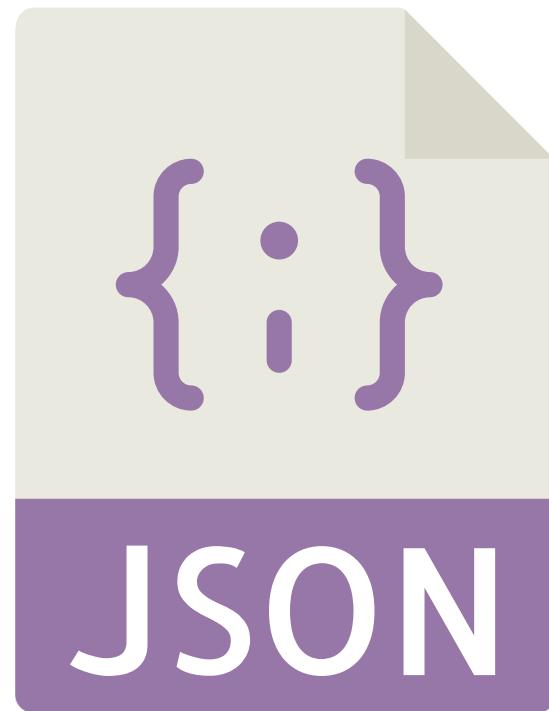
Architecture of Full-Text Search

The architecture of full-text search consists of:



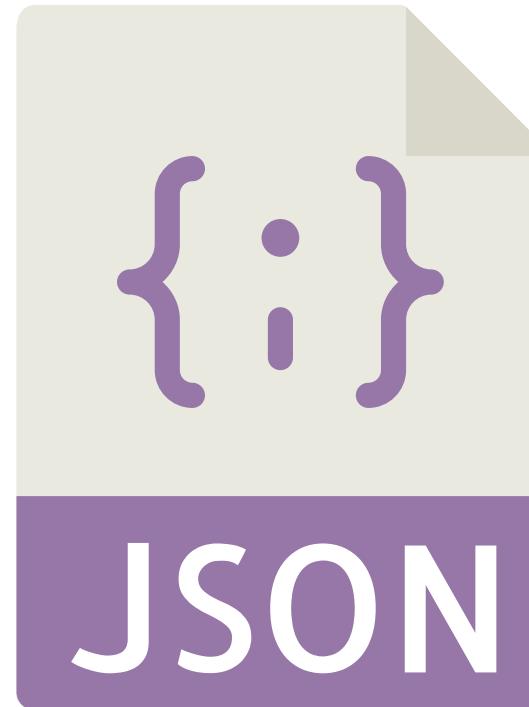
Understanding JSON Configuration

Configuration management is used for maintaining computer systems, servers, and software in a desired, consistent state.



- Data from config.json is used to configure virtual machines.

Creating a JSON Configuration File for Deployment



- 1 Create a file named **app.json**
- 2 Create deployment section in **app.json** file and define all the resources that need to be deployed

Configuration File: Example

Example

```
{  
  "deployment": {  
    "files": {  
      "example-resource-file1": {  
        "sourceUrl": "https://storage.googleapis.com/[MY_BUCKET_ID]/example-  
application/example- resource-file1"  
      },  
      "images/example-resource-file2": {  
        "sourceUrl": "https://storage.googleapis.com/[MY_BUCKET_ID]/example-  
application/images/example-resource-file2"  
      },  
    }  
  },  
}
```

Configuration File: Example

Example

```
"id": "v1",
  "handlers": [
    {
      "urlRegex": "/.*",
      "script": {
        "scriptPath": "example-python-app.py"
      }
    },
  ],
  "runtime": "python27",
  "threadsafe": true,
}
```

Building a Smart Appointment Scheduler Using JSON Server and JavaScript

Duration: 25 Mins.



Project agenda: To design and implement a real-time Smart Appointment Scheduler using HTML, CSS, JavaScript, and JSON Server, enabling users to book, reschedule, and cancel appointments dynamically while ensuring real-time updates, preventing double bookings, and improving communication between customers and service providers

Description: This project involves developing a real-time Smart Appointment Scheduler to streamline booking management for service-based businesses. The process includes setting up a mock backend using JSON Server, creating a responsive user interface, and implementing JavaScript logic to handle appointment booking, rescheduling, and cancellation. The system ensures dynamic updates and prevents double bookings. Tools such as VS Code, HTML, CSS, JavaScript, and JSON Server will be used to build and test the application.

Building a Smart Appointment Scheduler Using JSON Server and JavaScript

Duration: 25 Mins.

Perform the following:

1. Set up the JSON server
2. Install and run json-server
3. Build the user interface
4. Implement appointment logic in JavaScript
5. Test and run the application



Expected deliverables: The final output will include a fully responsive HTML interface, a mock backend powered by JSON Server, and JavaScript logic for dynamic appointment management. The deliverables will demonstrate real-time updates, user interaction handling, and basic CRUD operations, providing a foundation for scalable and efficient appointment systems.

Quick Check

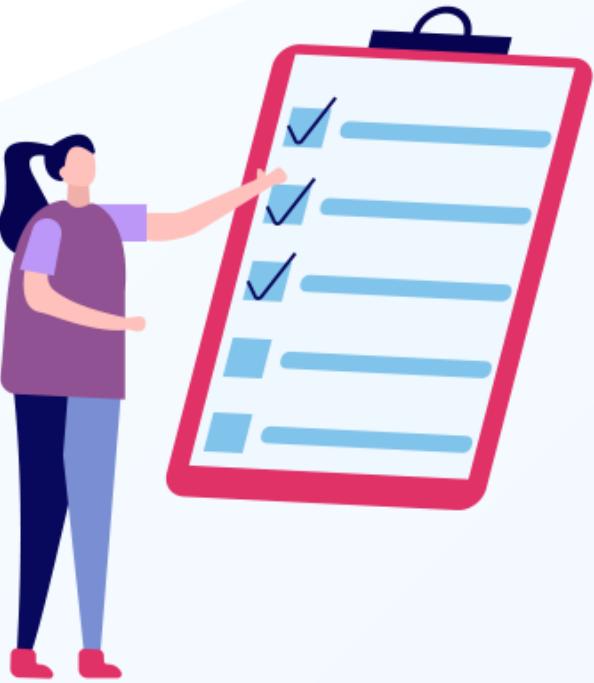


You are developing a bookstore web application that stores book details in a JSON database. Users should be able to search for books by title, author, or keywords within the description. Which approach will allow you to implement full-text search efficiently in JSON data?

- A. Use a loop in JavaScript to check if the search term exists in each book's title, author, or description
- B. Use a database with full-text indexing, like MongoDB, to perform efficient search queries
- C. Convert JSON data into an array of objects and manually compare each field with the search term
- D. Use the DELETE method to remove books that do not match the search term and display the rest

Key Takeaways

- Structured formats like JSON support readable data exchange between systems.
- Efficient communication is enabled between browsers and servers using lightweight data.
- Frontend development can be quickly simulated using tools like JSON Server.
- Document-oriented databases store and manage data using flexible JSON structures.
- Data filtering and sorting features help customize information based on user requirements.



Thank You