Design a Dynamic Frontend with React

Serving Data with JSON Server

A Day in the Life of a MERN Stack Developer

You are working as a MERN stack developer at a fintech startup, tasked with developing a banking application.

This project requires efficient data management, prompting you to explore the use of JSON Servers.

In this lesson, you will delve into JSON and understand its features, applications, and syntax.

You will set up and demonstrate a server, handle data types like objects and arrays, and explore the interaction of JSON with HTML.

A Day in the Life of a MERN Stack Developer

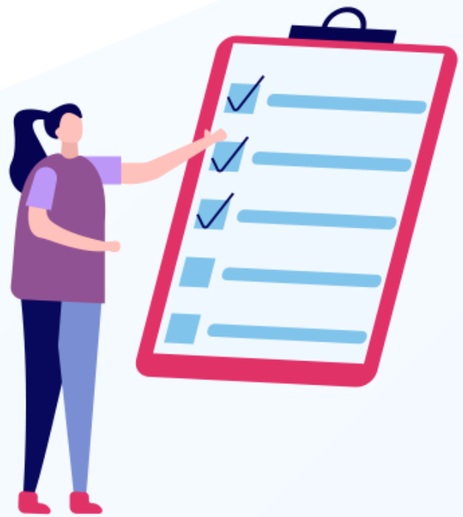
You will also learn to implement CRUD operations (GET, POST, PUT, DELETE) on your JSON server and engage in activities like sorting and filtering data.

To achieve the above goals, along with some additional features, you will be learning a few concepts in this lesson that will help you find a solution to the above scenario.

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



Introduction to JSON

What Is Model?

A model refers to the data structure in programming.

- In an application, a model serves the purpose of storing data.
- This data storage often involves creating classes and objects using Object-Oriented Programming (OOP).

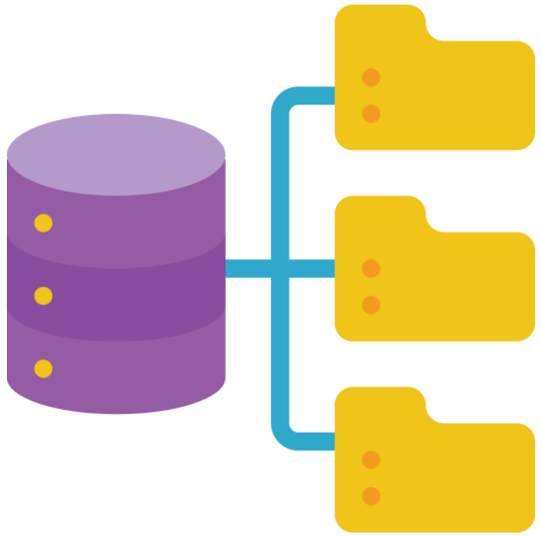
Model: Example

In a simple e-commerce application, a model might be created to represent a **Product**.

- The **Product** class will include attributes like price, name, and category.
- Each product in the application will be an object of the **Product** class, holding its unique data.

Importance of Models

Models are crucial for separating the application's data and business logic from the user interface.



This separation allows for a more organized codebase and easier maintenance.

Introduction to JSON

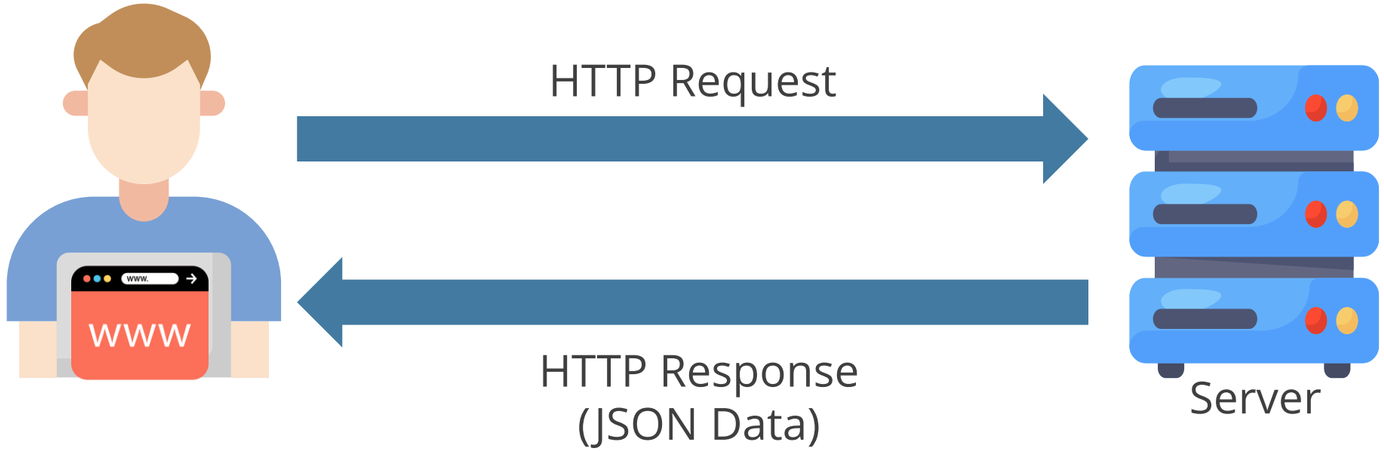
JSON stands for **J**ava**S**cript **O**bject **N**otation.

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

Source: https://www.codemotion.com/magazine/frontend/javascript/douglas-crockford-presents-his-new-book-how-javascript-works/

What Is JSON?

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



Features of JSON

JSON is user-friendly, being both easy to write and read.

It efficiently retrieves data from server-side web applications.

Additionally, JSON supports a variety of data structures.

JSON Supporting Languages

The JSON text format is entirely independent of programming languages:

Python

Ruby

PHP

Java

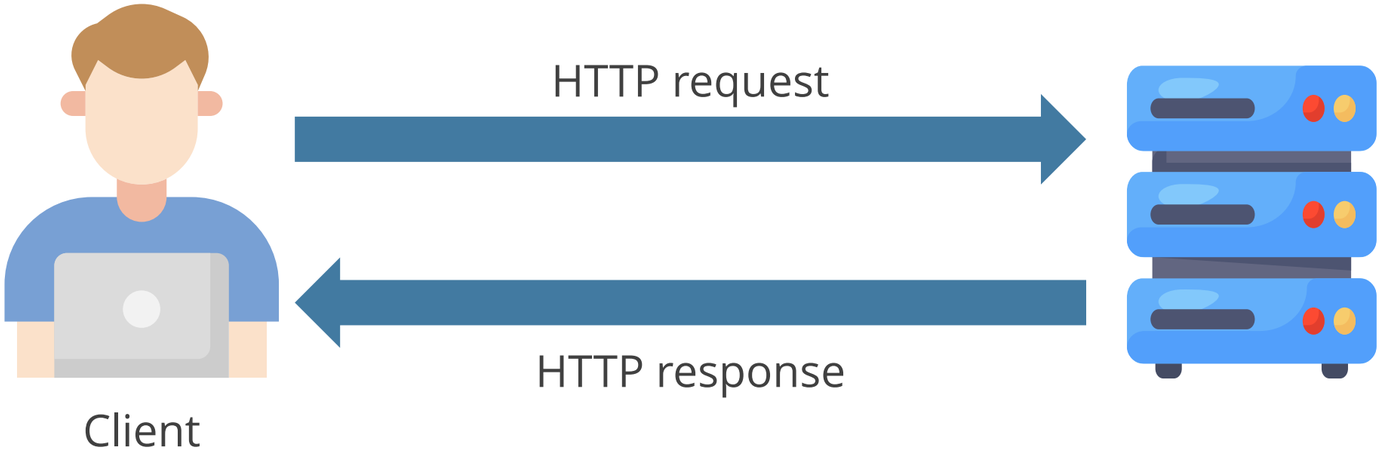
C

C++

C#

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

JSON: Architecture



Server site data format:

```
Employee Object in Java
Emp id;
Emp Name;
Salary;
```

Converted into JSON format

JSON Object

```
{
    "Emp id":"1",
    "Emp Name":"Tom",
    "Salary":5000
}
```

JSON 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 type:

Null

Represents the absence of a value or a deliberate nonexistence

For example:
‘{“Age": null, “Compant": 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.

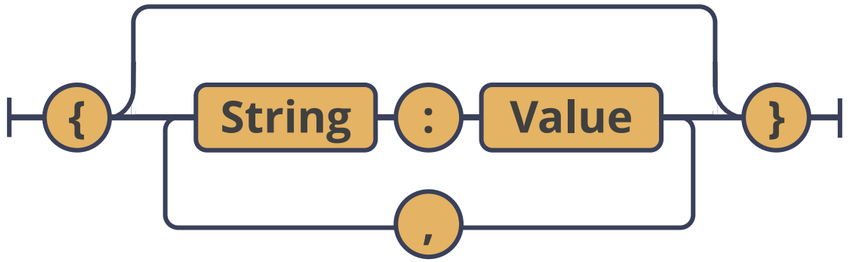
In most programming languages, objects are known by various names, including object, record, struct, dictionary, hash table, keyed list, or associative array.

Arrays

An array is an ordered list of values.

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

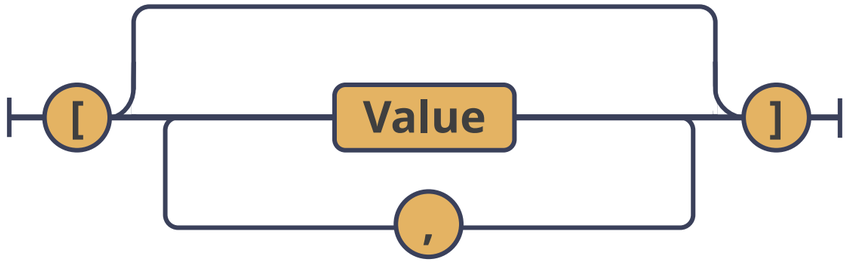
JSON Object




```
{
    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




```
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 (,).

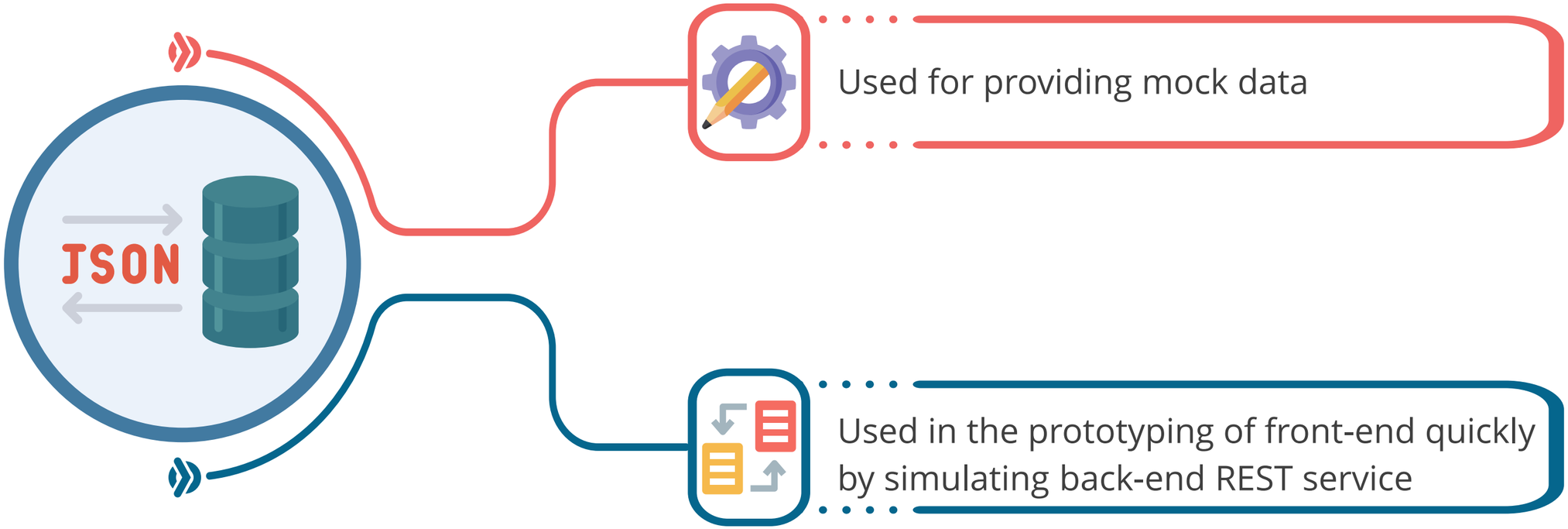
Object vs. Array

Object	Array
Set of key-value pairs	List of values
Enclosed in curly braces (**{ }**)	Enclosed in square brackets (**[ ]**)
Name-value pairs are separated by a comma (**,**)	Values are separated by a comma (**,**)
Each name is followed by a colon (**:**)	NA
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.

JSON Server

What Is JSON Server?

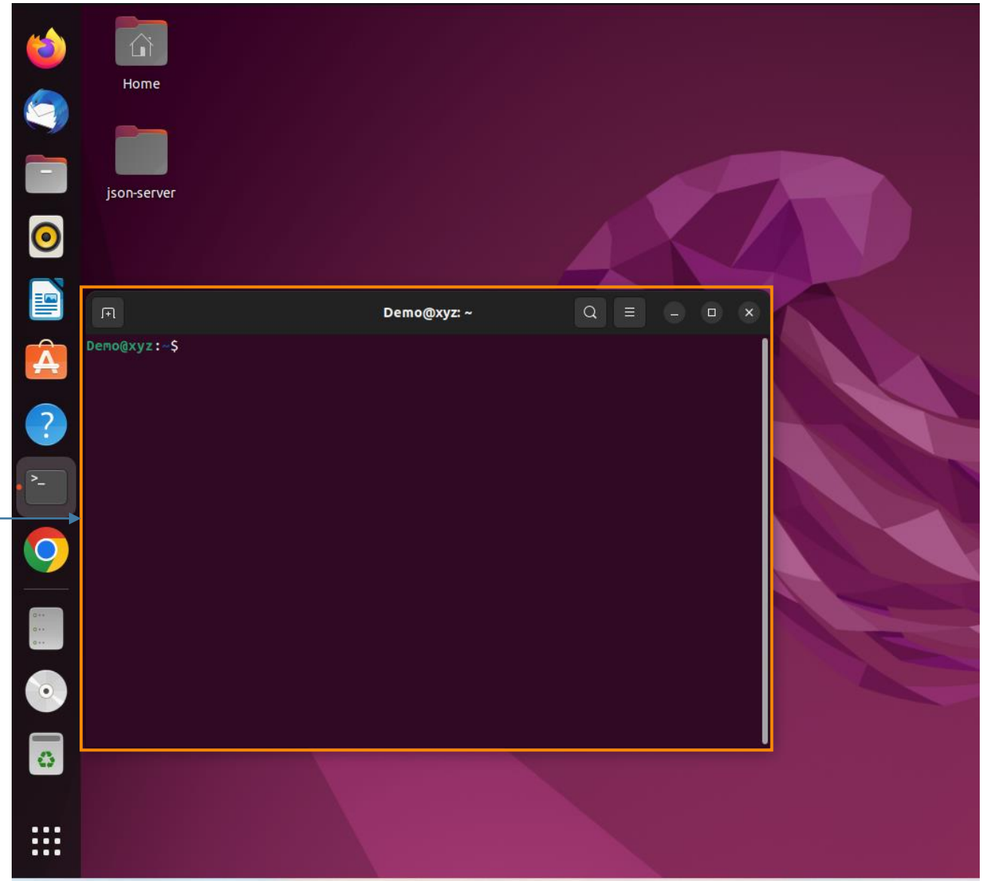
JSON Server is a npm (Node Package Manager) module used for creating fake REST API.



Installation of JSON Server

Step 1

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



Installation of JSON Server

Step 2

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

```
Demo@xyz: ~
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 libvdpau1 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.
```

Installation of JSON Server

Step 3

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

```
Demo@xyz: ~
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 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 libvdpau1 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
```

Installation of JSON Server

Step 4

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

Step 5

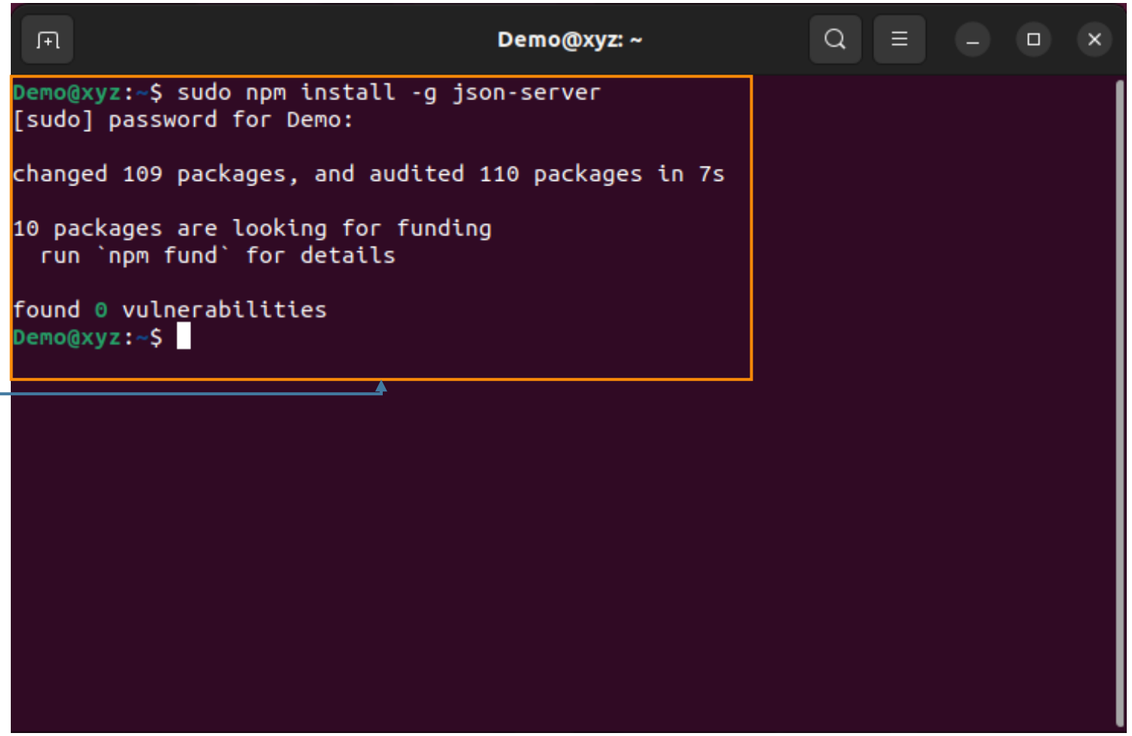
Type **sudo n lts** for installing the latest node version

```
Demo@xyz: ~
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)
```

Installation of JSON Server

Step 6

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



Checking JSON Server Version

`json-server -v`	Used to check the JSON Server version

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

Checking JSON Server Version

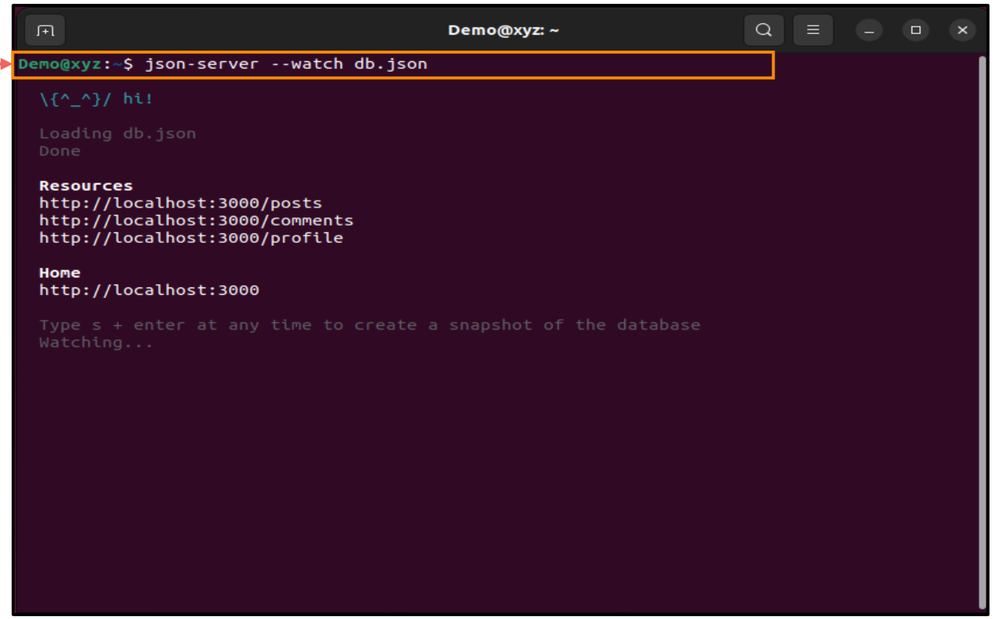
`json-server -help`	Used to show all the options of JSON Server

```
Demo@xyz: ~
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:~$
```

Run JSON Server

Step 1

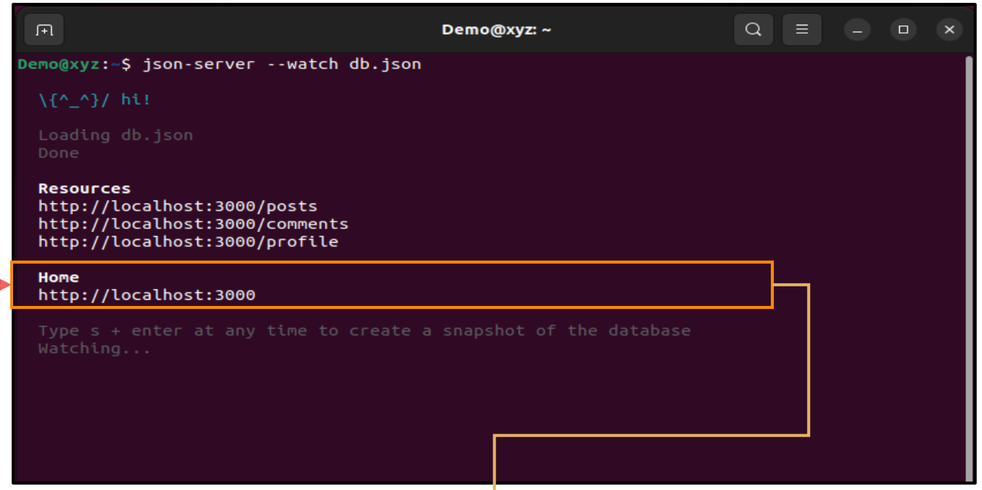
Type **json-server --watch db.json** for running the JSON Server

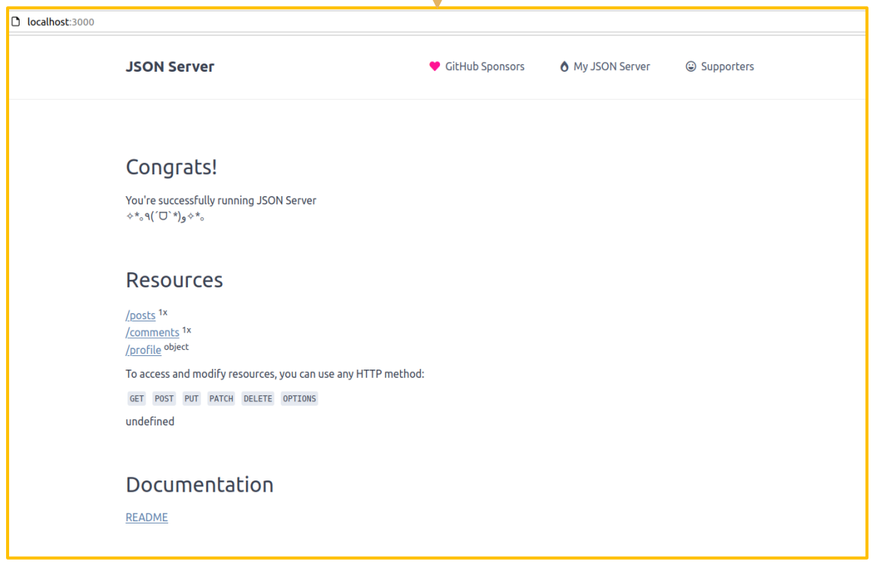


Run JSON Server

Step 2

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

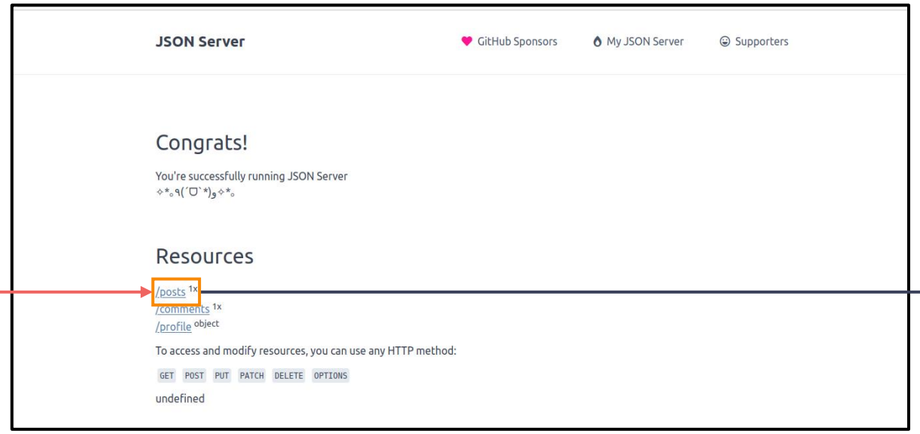


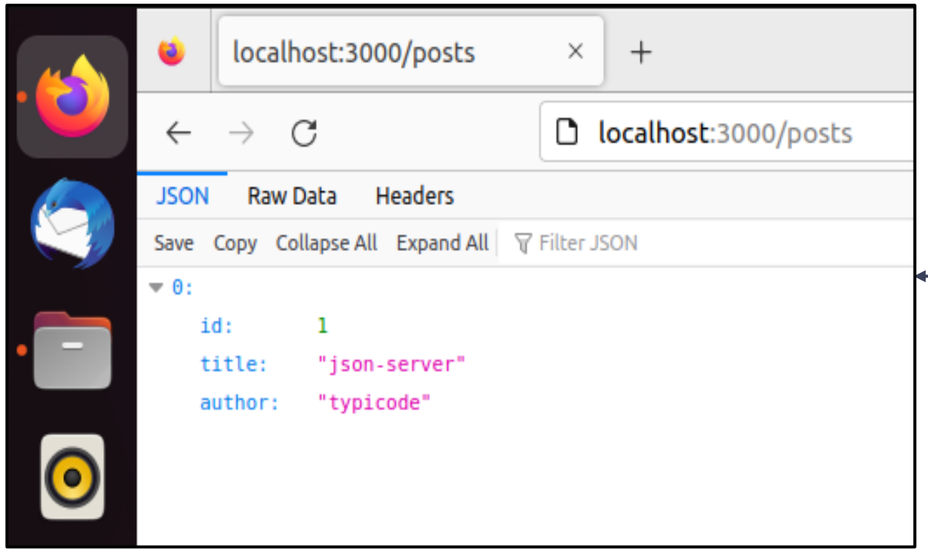


Run JSON Server

Step 3

Click on **/post**[1x] to open the **JSON file data**, which is automatically created





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.

Assisted Practice: Guidelines

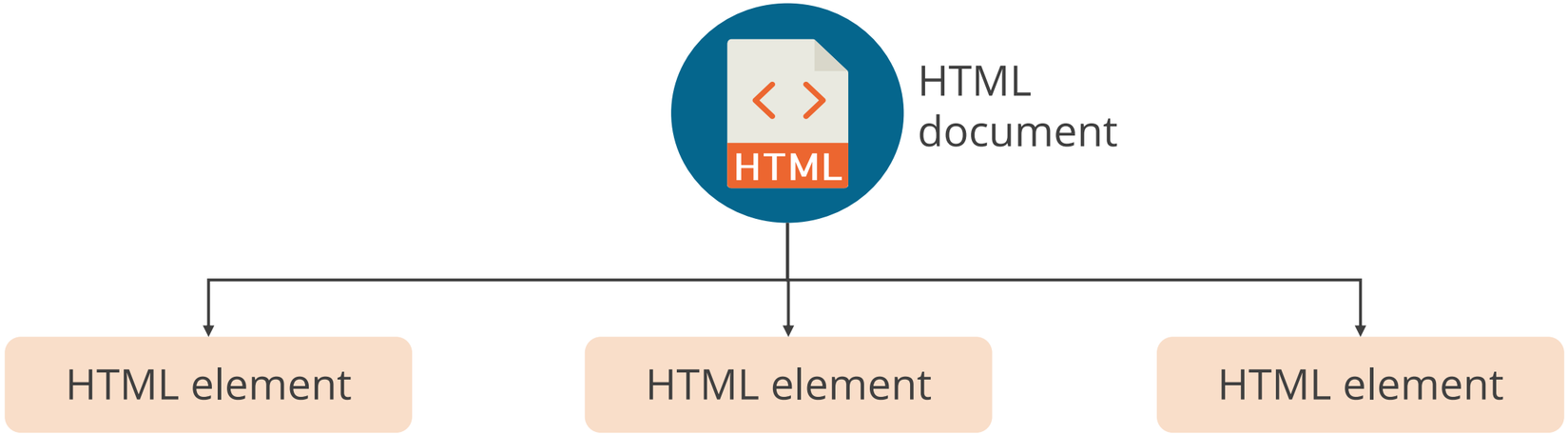
Steps to be followed:

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

HTML Elements

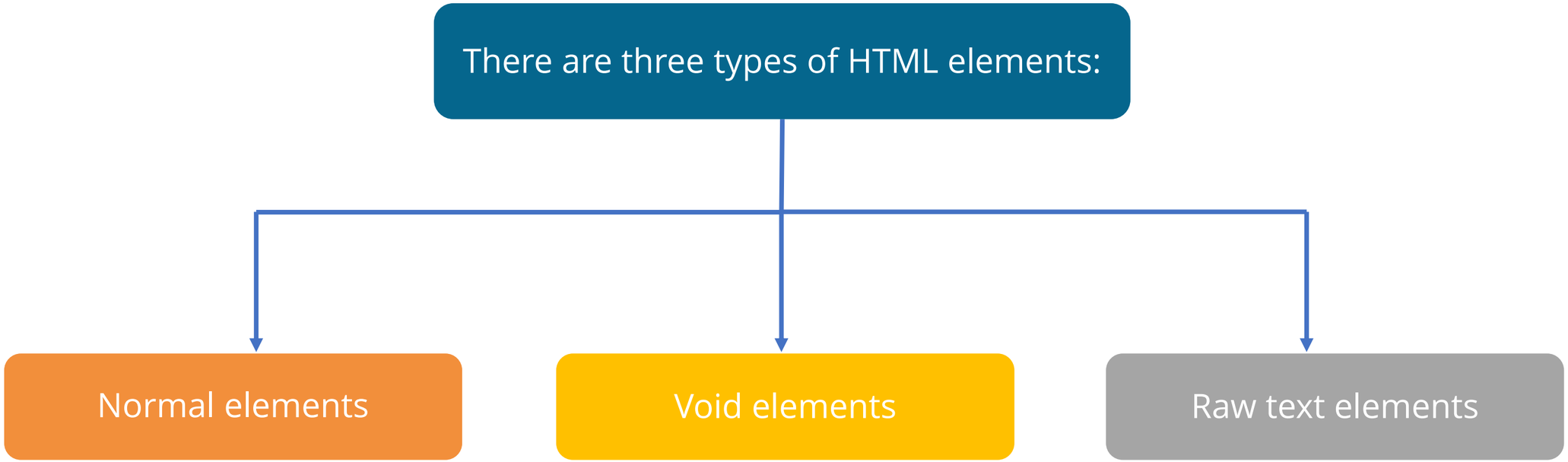
HTML Element

HTML element 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



Normal Elements

Normal elements contain a start and an end tag. One or both tags can be omitted for some elements.

They are written as follows:

```
<start tag> Content text…123 </End tag>
```

Indicates the start of an element

Includes text content and other elements

Indicates the end of the element name

Raw Text Elements

The raw text element is also referred to as a text-only element.

They are written as follows:

```
<start tag> Content text... </End tag>
```

Indicates the start of an element

Includes text content without elements

Indicates the end of the element name

Void Elements

Void element is also referred as empty or standalone element.

They are written as follows:

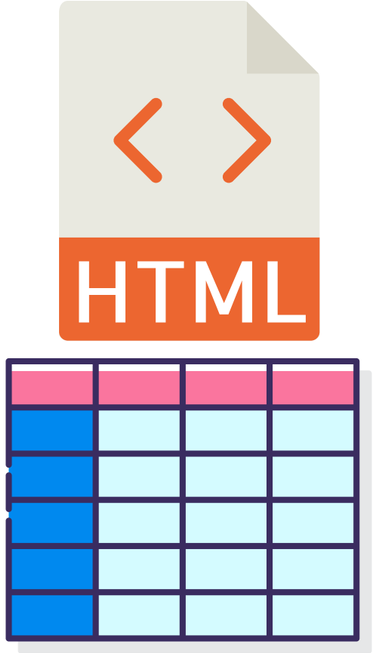
Contains a backslash before the end of the start tag, but this is entirely optional

```
<start tag/.....>
```

```
<br/>
```

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

```
<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 Elements

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

```
<!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 Elements

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

```
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 Elements

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

A dynamic HTML table changes its number of rows based on the input data.

A dynamic HTML table can be created using JavaScript.

Example of Dynamic HTML Table

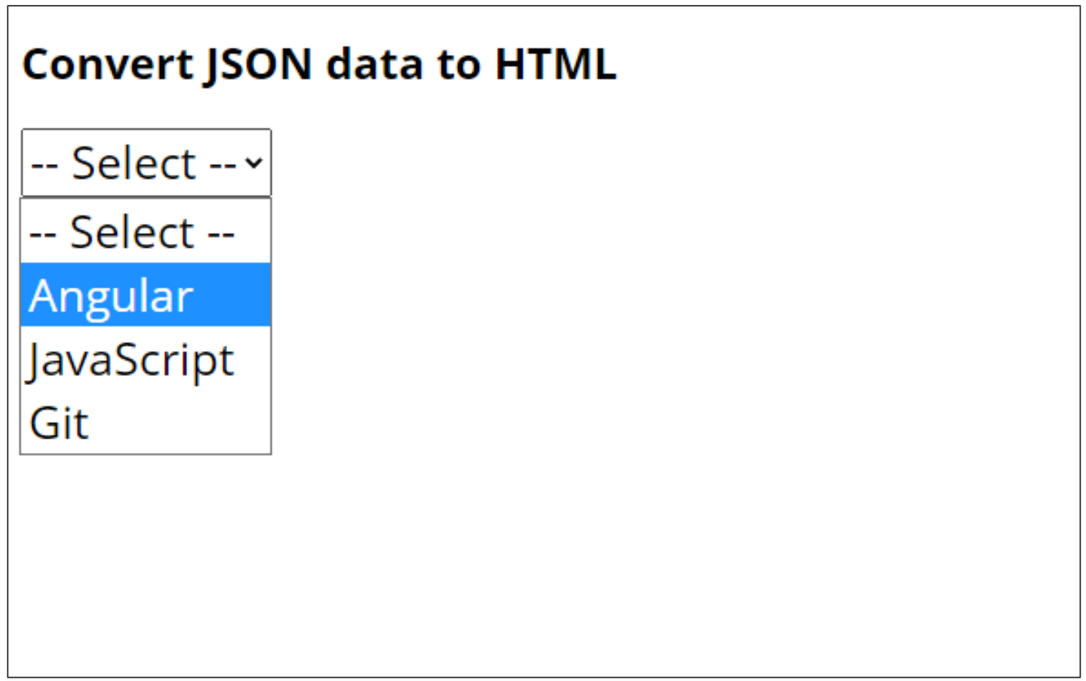
The following example shows how to create the HTML table based on the value of a drop-down menu:

```
<!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>
```

```
<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:



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.

Assisted Practice: Guidelines

Steps to be followed:

1. Create an HTML file

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.

Assisted Practice: Guidelines

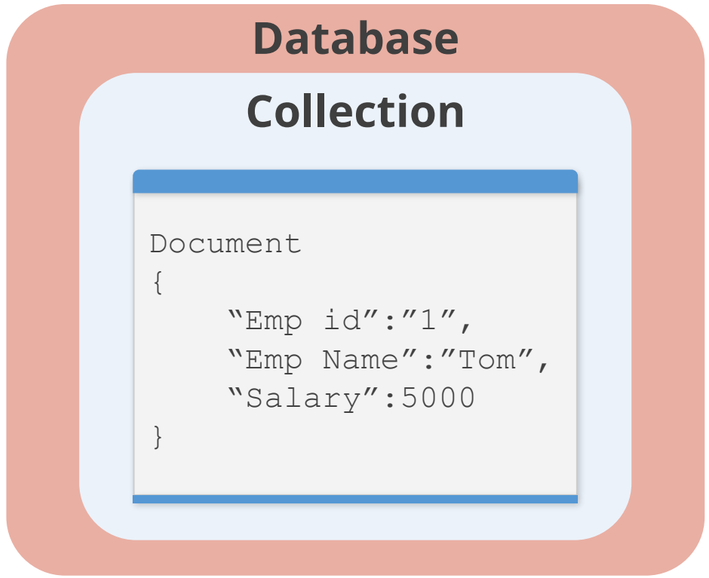
Steps to be followed:

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

Set up a JSON DB

JSON DB

JSON database 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.

```
{
    {
        "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 data sets 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}

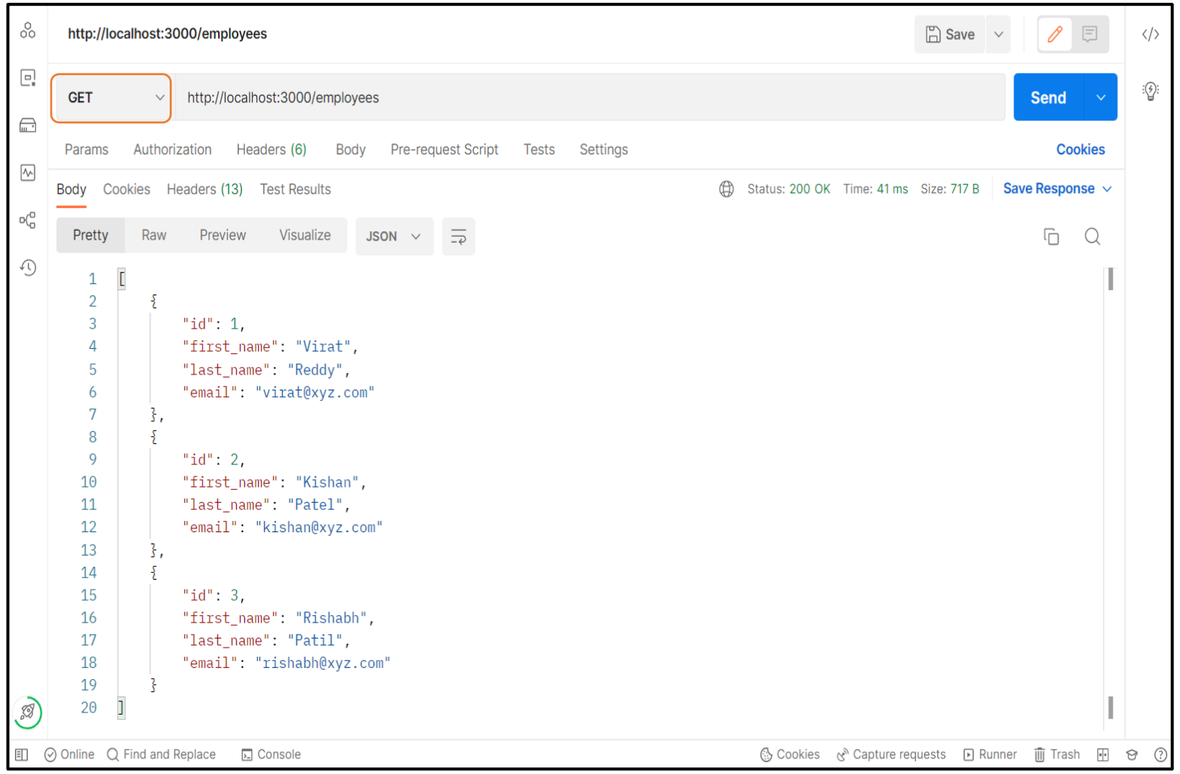
Changes made using POST, PUT, and DELETE will be done automatically in the db.json file.

Working with JSON Server

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

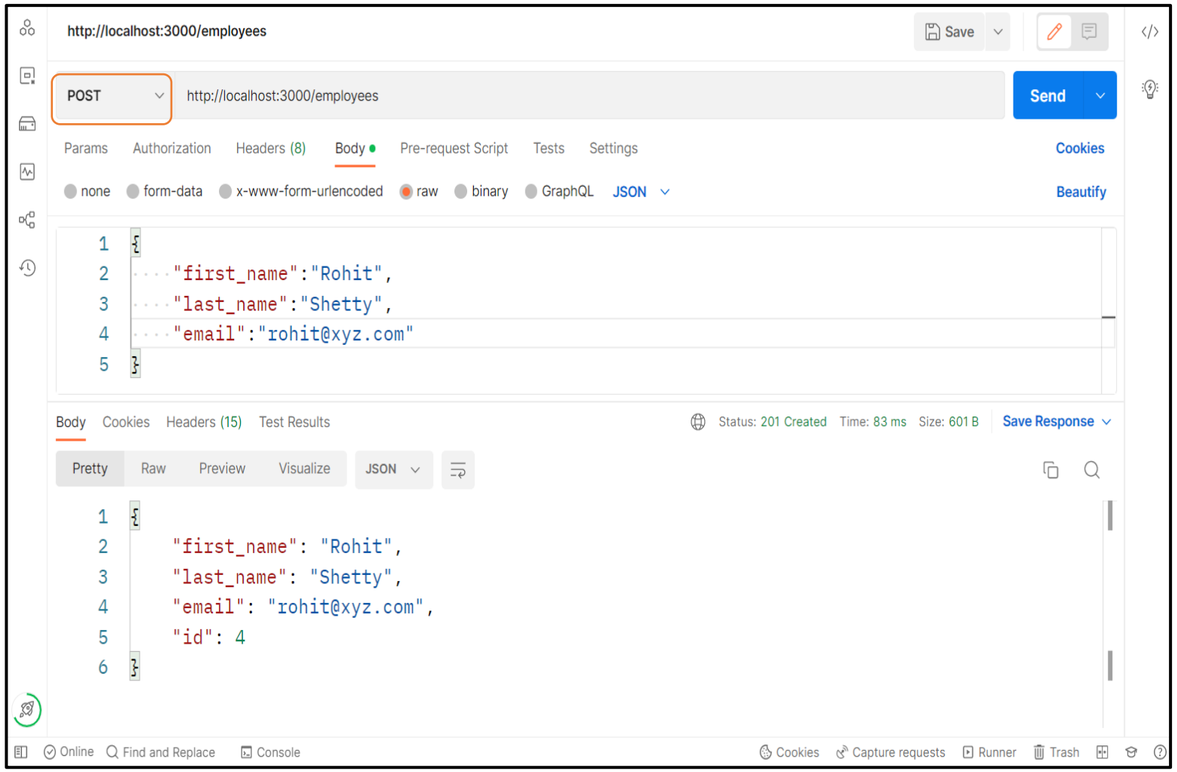
GET	**/employees+{Id}** finds the employee by ID.
PUT	**/employees** updates an existing employee.
DELETE	**/employees/{Id}** deletes an employee.
POST	**/ employees /{Id}/uploadImage** uploads an image.

JSON Server: GET



GET	Requests abide 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

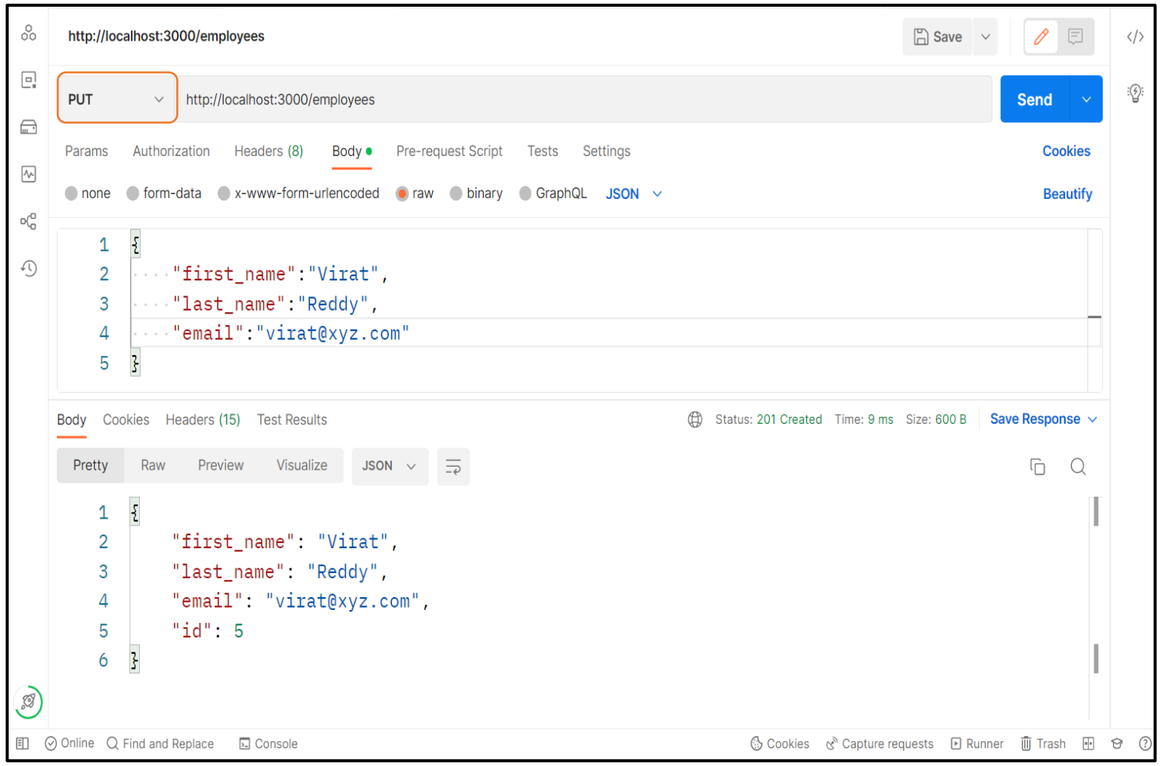


POST — Requests do not survive in the browser history

POST — Requests cannot be bookmarked

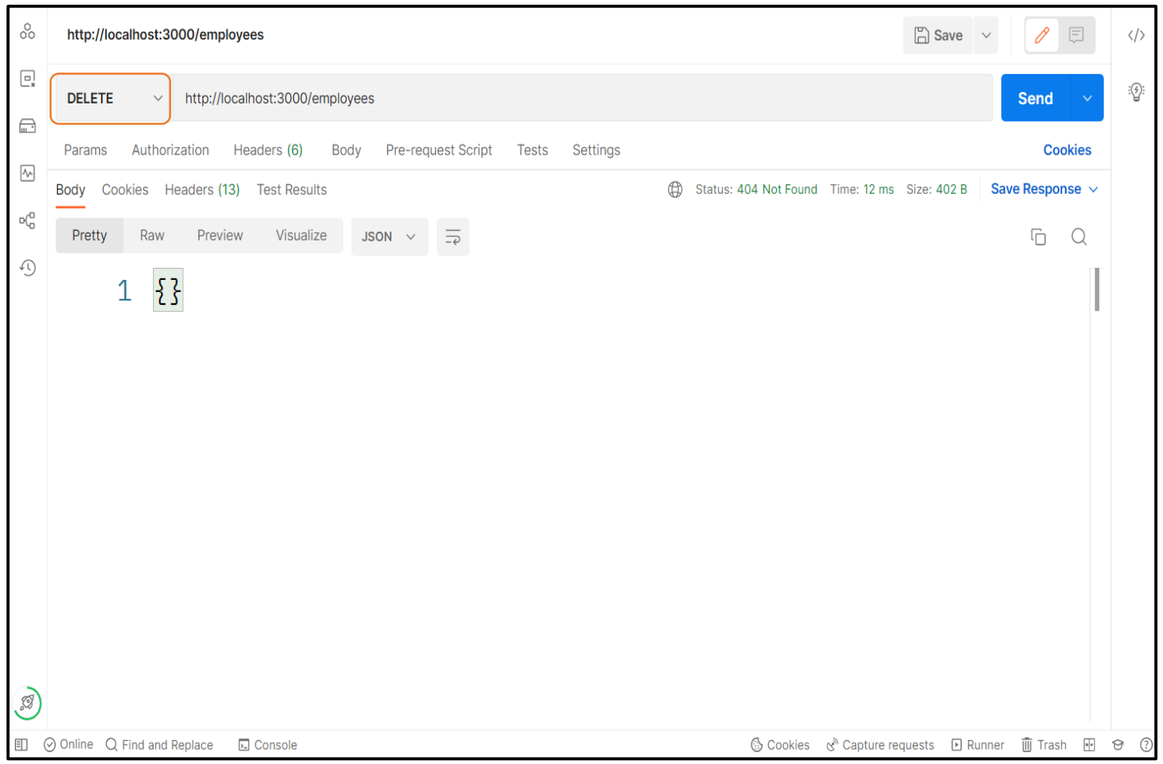
POST — Requests have no restrictions on data length

JSON Server: PUT



PUT	Relays data to a server to create or update a resource

JSON Server: DELETE



DELETE Deletes a resource from the server

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.

Assisted Practice: Guidelines

Steps to be followed:

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

Fetching Data

Fetching Data

Users can fetch the JSON file's data.

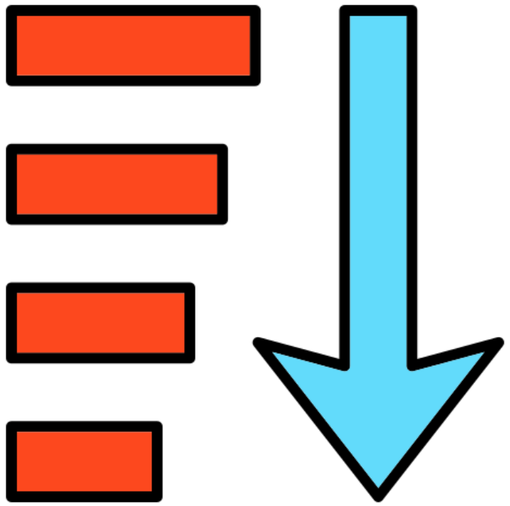
Fetch

- 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

The 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

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

Filter query parameter family

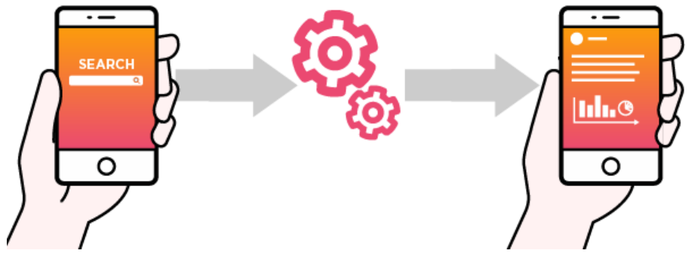
filter, filter[x], filter[],
filter[x][], filter[][], filter[x][y]

Filter

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

Full-Text Search

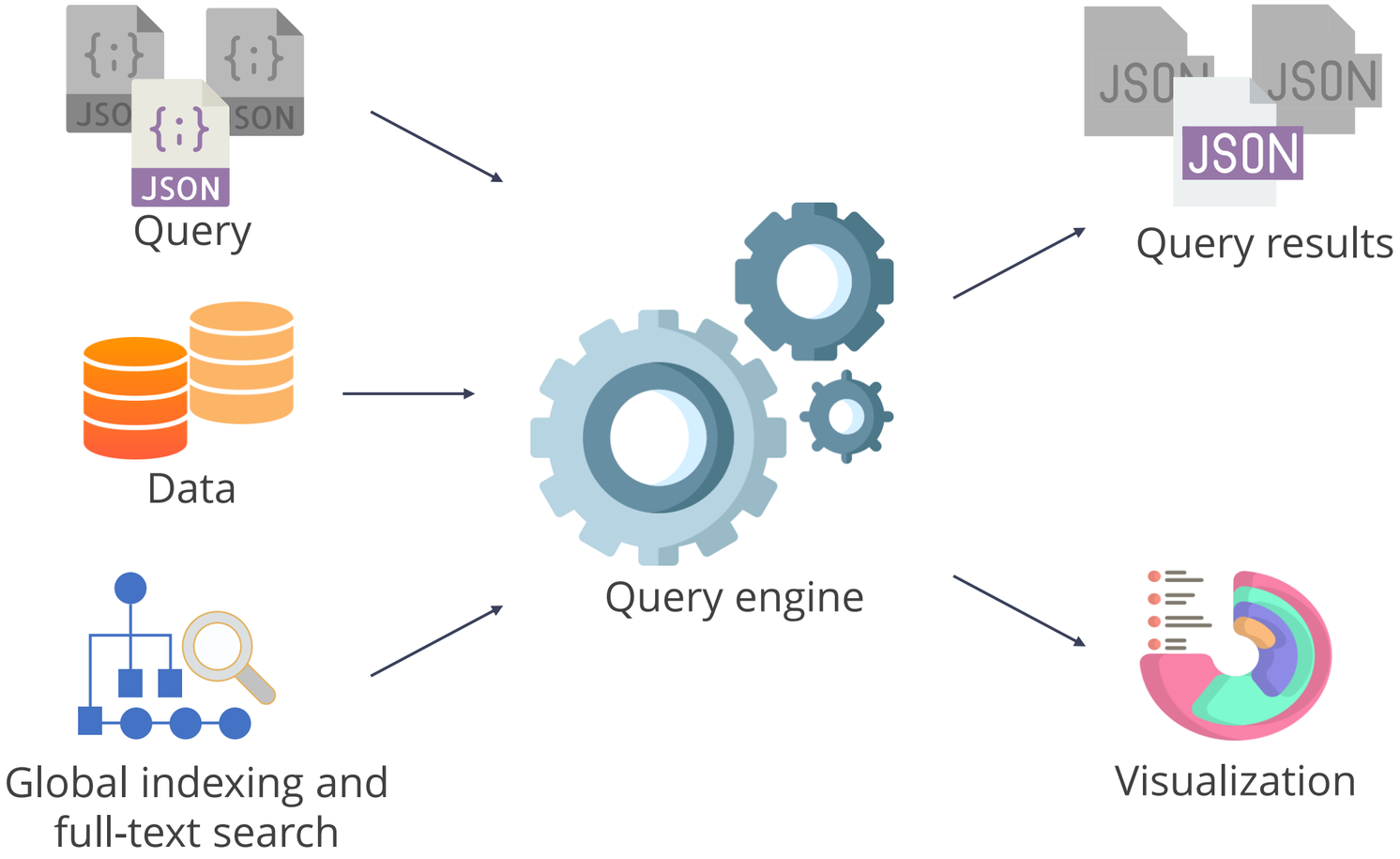
Full-text engine compiles full-text queries that can search a table for particular words or combinations of words.



- 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

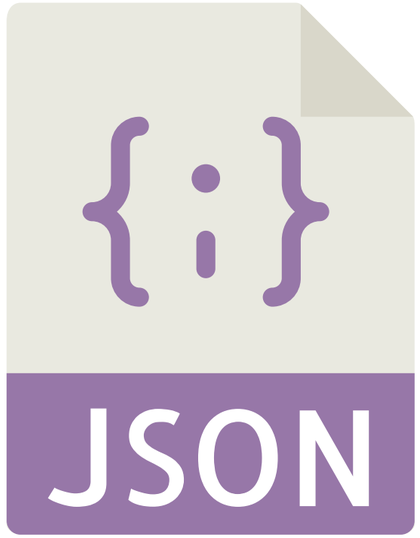
Full-Text Search

The architecture of full-text search consists of:



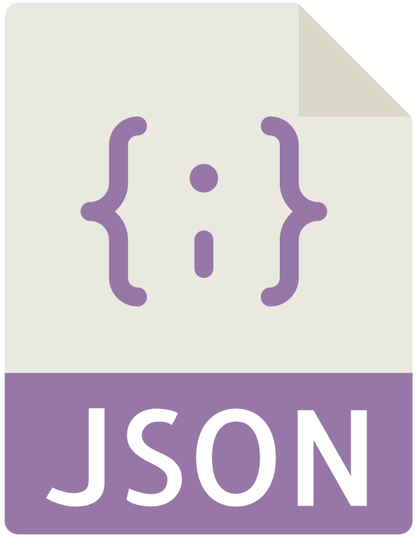
Configuration

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.

Create Configuration File for Deployment



1. Create a file named **app.json**

2. Create deployment section in **app.json** file and define all the resources which needs to be deployed

Configuration File : Example

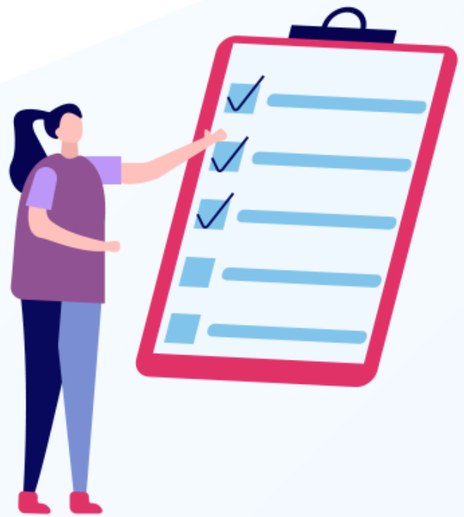
```
{
  "deployment": {
    "files": {
      "example-resource-file1": {
        "sourceUrl": "https://storage.googleapis.com/[MY_BUCKET_ID]/example-
application/example-   resource-file1"
      },
      "images/example-resource-file2": {
        "sourceUrl": "https://storage.googleapis.com/[MY_BUCKET_ID]/example-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

Configuration File : Example

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

Key Takeaways

- JSON is a data representation format like YAML or XML.
- JSON is used for transmitting data between web browsers and servers.
- JSON Server is a npm module used to quickly prototype the front-end by simulating back-end REST services.
- JSON document database is a non-relational database that stores and queries data as JSON documents.
- JSON data can be sorted and filtered based on user’s need.



Thank You