RETRIEVING SERVER DATA

REST

- All of the previous examples used React components, they communicated with each other using props and hooks.
- React also communicate with back-end servers using **RESTful** APIs.
- There are a number of different ways of requesting data from servers in React.
- `XMLHttpRequest` , `Fetch API`, `Axios`

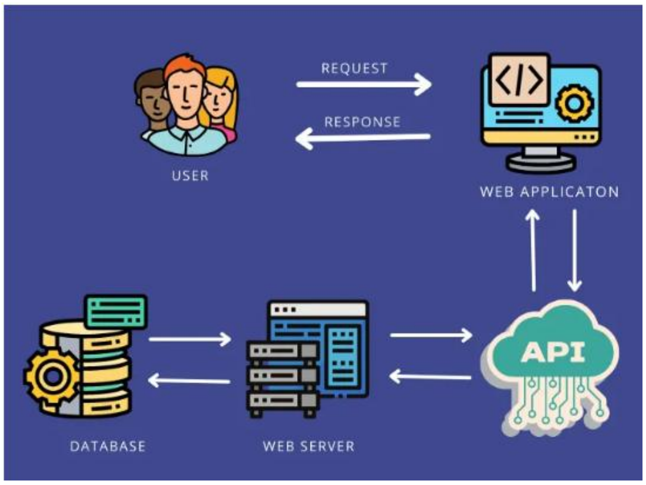
- When developing an application with React, users request data from a server to dynamically update content inside our application.
- Web services such as SOAP, WSDL, and REST are used to request data from a server.
- Most **client – server** examples focus on the RESTful API .
- REST :- Representational State Transfer
- The REST API can be defined as an API that uses the HTTP protocol GET, POST, and DELETE to request data

- Generally store data on the server
  - Allow data to be shared between site users
  - Data persistence
- But even with the data on the back-end we still have manage the data on the front-end?
- The REST API is a standardised interface for the client to manage data on the server.

REST API

- "An application programming interface (API) is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software.
- [1] A document or standard that describes how to build or use such a connection or interface is called an API specification. A computer system that meets this standard is said to implement or expose an API.
- The term API may refer either to the specification or to the implementation. Whereas a system's user interface dictates how its end-users interact with the system in question, its API dictates how to write code that takes advantage of that system's capabilities"

https://en.wikipedia.org/wiki/API



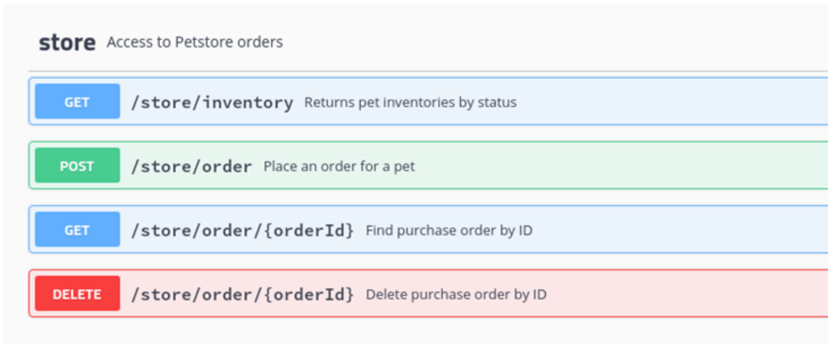
- REpresentational State Transfer abbreviated to REST.
- It is an API that follows a set of rules for an application and services to communicate with each other.

https://itznihal.medium.com/api-rest-api-and-restful-api-7767d9997854

- When a user makes a response APIs send a ‘`request`’ information from a web application or web server, it will receive a ‘response.’
- The location where APIs send requests or where resources live are endpoints.
- Examples

REST APIs

- With Full-Stack applications we want to store our data on the server side.
- The server takes care of creating, reading, updating, and deleting data.



store Access to Petstore orders

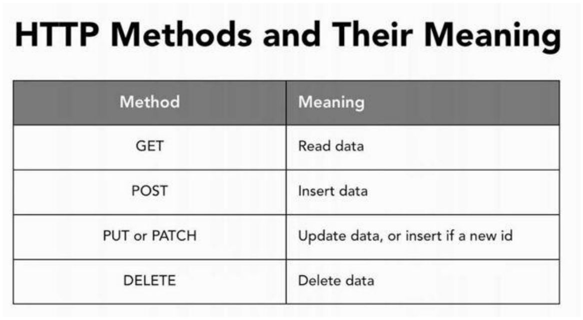
GET `/store/inventory` Returns pet inventories by status

POST `/store/order` Place an order for a pet

GET `/store/order/{orderId}` Find purchase order by ID

DELETE `/store/order/{orderId}` Delete purchase order by ID

- REST API is a standardised way for the client side and our server to interact with regards to loading and manipulating data.
- The REST APIs provide us with, ways to do common task in the Full-Stack world

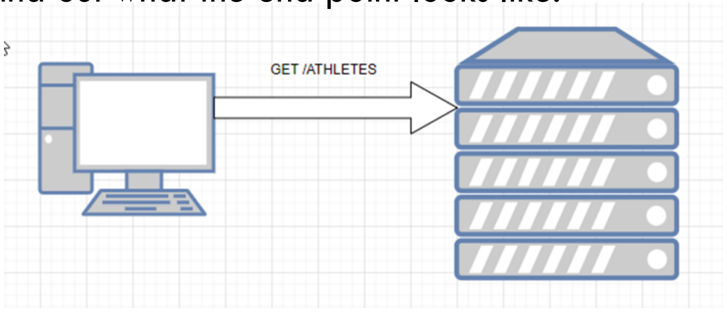


HTTP Methods and Their Meaning

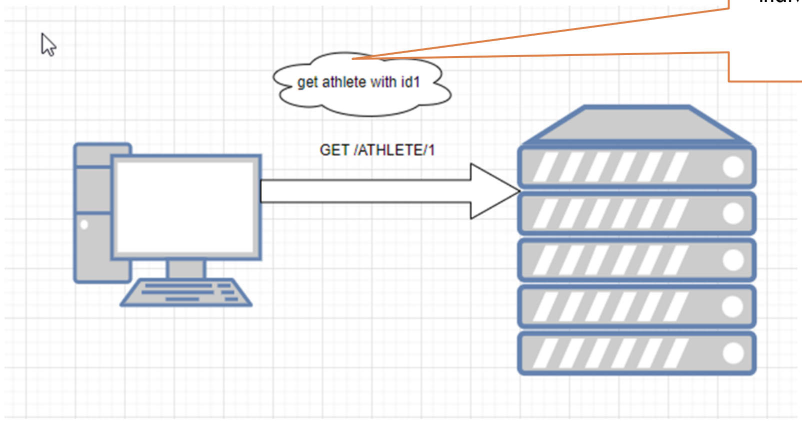
Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

https://www.freshersnow.com/rest-api-interview-questions-and-answers/

- The front-end makes a GET request to the URL /athletes on the server and then the server would send back an array containing all the athletes.
- For REST, the standardised way is to use the GET requests.
- The front-end users don't actually need to talk to the back-end people in most cases to find out what the end point looks like.

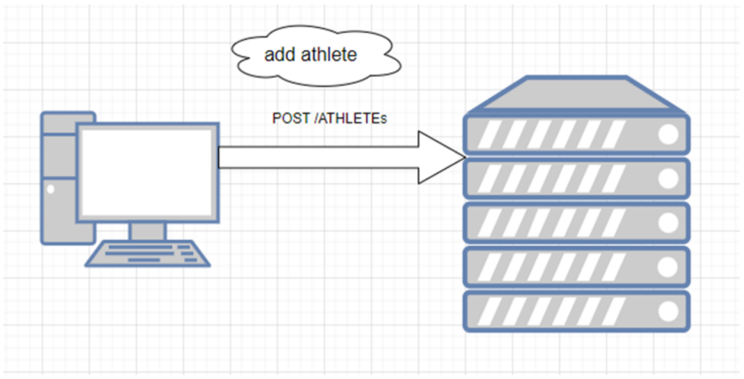


- Loading a single instance

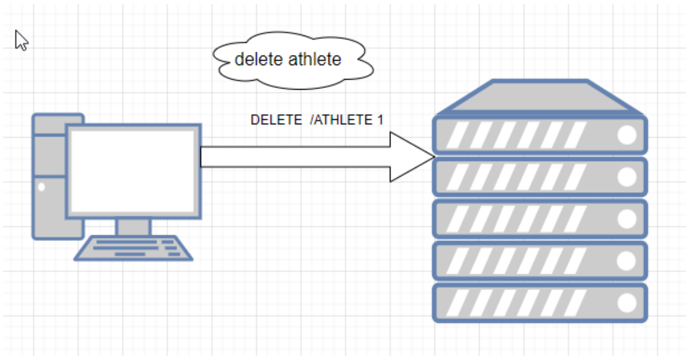


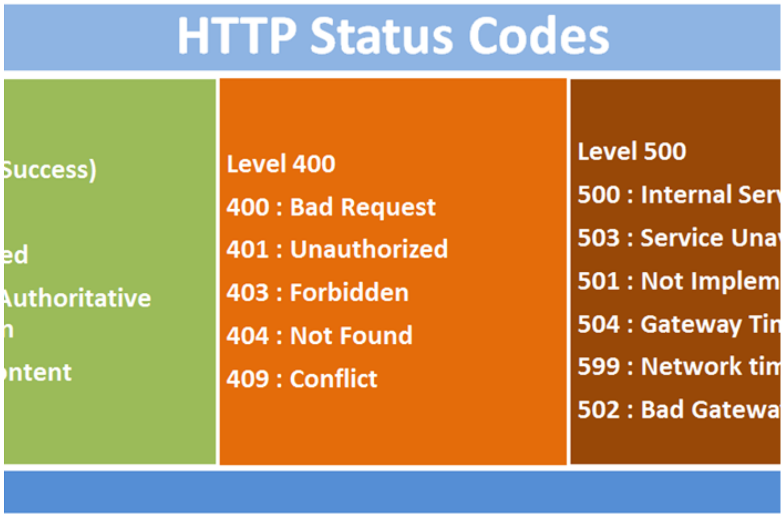
standard way of loading an individual element by its unique id.

- To create new data on the server send a POST request to the back-end



- To delete an instance of a given resource send a DELETE request to the URL /athlete/1





HTTP Status Codes

uccess)

ed

uthoritative

n

ntent

Level 400

400 : Bad Request

401 : Unauthorized

403 : Forbidden

404 : Not Found

409 : Conflict

Level 500

500 : Internal Ser

503 : Service Una

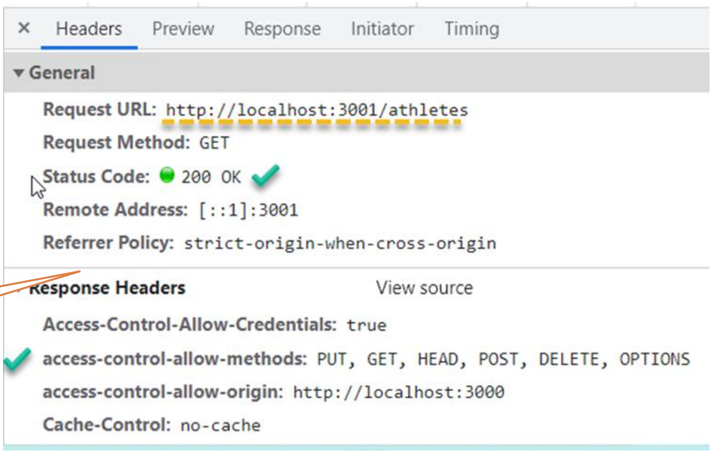
501 : Not Implem

504 : Gateway Ti

599 : Network ti

502 : Bad Gatewa

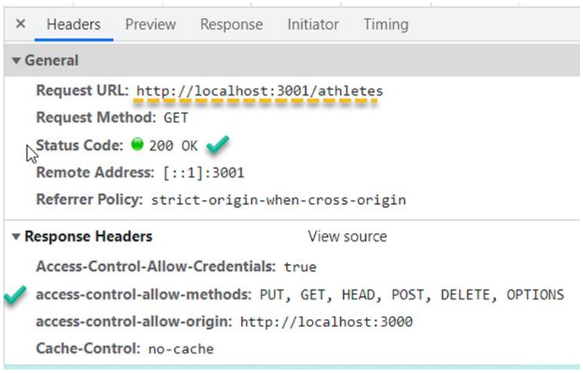
- When we request data from a server, we include HTTP headers in a REST request to tell the server the format of data the client is expecting.



```
× Headers  Preview  Response  Initiator  Timing
▼ General
   Request URL: http://localhost:3001/athletes
   Request Method: GET
   Status Code: ● 200 OK ✓
   Remote Address: [::1]:3001
   Referrer Policy: strict-origin-when-cross-origin
▼ Response Headers                      View source
   Access-Control-Allow-Credentials: true
 ✓ access-control-allow-methods: PUT, GET, HEAD, POST, DELETE, OPTIONS
   access-control-allow-origin: http://localhost:3000
   Cache-Control: no-cache
```

To review details of headers, open the Chrome DevTools (F12), go to the Network tab, and refresh the page to request the data.

- When we request data from a server, we include HTTP headers in a REST request to tell the server the format of data the client is expecting.



```
× Headers  Preview  Response  Initiator  Timing
▼ General
   Request URL: http://localhost:3001/athletes
   Request Method: GET
   Status Code: ● 200 OK ✓
   Remote Address: [::1]:3001
   Referrer Policy: strict-origin-when-cross-origin
▼ Response Headers                      View source
   Access-Control-Allow-Credentials: true
 ✓ access-control-allow-methods: PUT, GET, HEAD, POST, DELETE, OPTIONS
   access-control-allow-origin: http://localhost:3000
   Cache-Control: no-cache
```

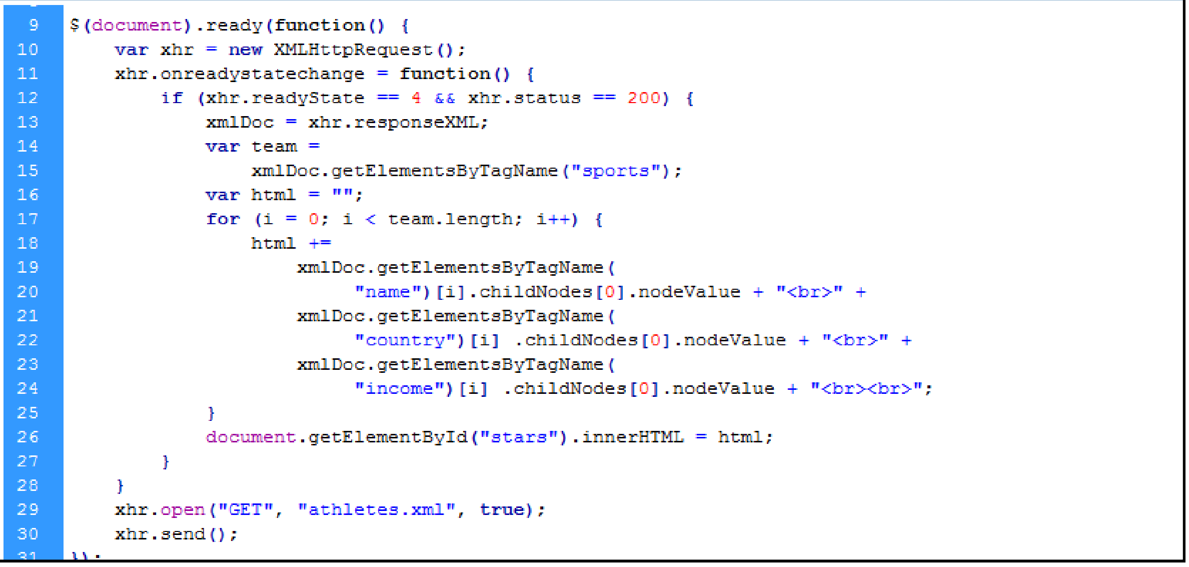
`XMLHttpRequest` object

- This object is behind every Ajax request.
- The open method is used to open a connection for a request.
- The parameters for the open method, this method specifies whether the request is a GET or POST request and it provides the URL for the request.
- The `readyState` property indicates the state of the request, the `status` and `statusText` properties provide the status code and status message that's returned by the server, and the `responseText` and `responseXML` properties provide the returned data in plain or XML format.

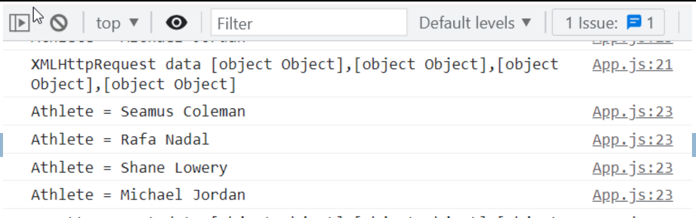
`XMLHttpRequest` (old and dated)

- **Methods**
  - `abort()`
  - `getAllResponseHeaders()`
  - `getResponseHeader(`*`name`*`)`
  - `open(`*`method`*`,`*`url`*`[,`*`async`*`][,`*`user`*`][,`*`pass`*`])`
  - `send([`*`data`*`])`
  - `setRequestHeader(`*`name`*`,`*`value`*`)`
- **Properties**
  - `readyState`
  - `responseText`
  - `responseXml`
  - `status`
  - `statusText`
- **Event**
  - `onreadystatechange`

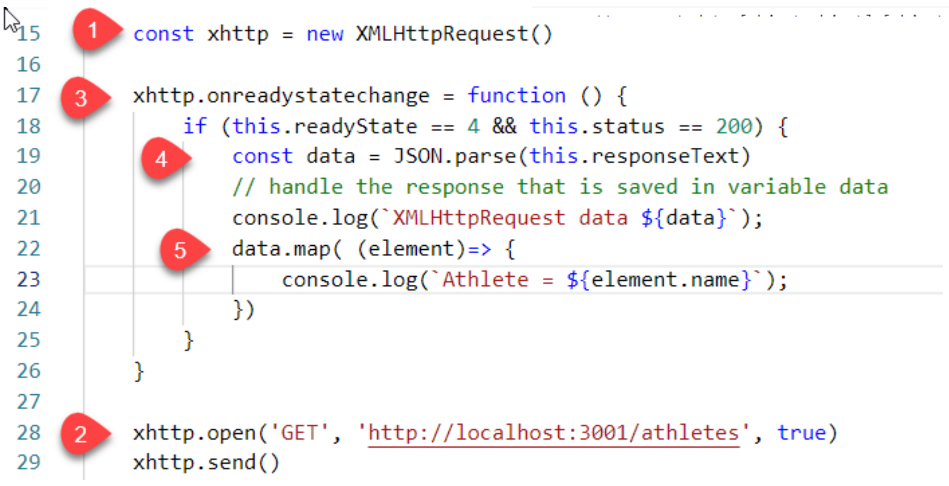
Recap



```
$(document).ready(function() {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            xmlDoc = xhr.responseXML;
            var team =
                xmlDoc.getElementsByTagName("sports");
            var html = "";
            for (i = 0; i < team.length; i++) {
                html +=
                    xmlDoc.getElementsByTagName(
                        "name")[i].childNodes[0].nodeValue + "<br>" +
                    xmlDoc.getElementsByTagName(
                        "country")[i] .childNodes[0].nodeValue + "<br>" +
                    xmlDoc.getElementsByTagName(
                        "income")[i] .childNodes[0].nodeValue + "<br><br>";
            }
            document.getElementById("stars").innerHTML = html;
        }
    }
    xhr.open("GET", "athletes.xml", true);
    xhr.send();
});
```



```
XMLHttpRequest data [object Object],[object Object],[object    App.js:21
Object],[object Object]
Athlete = Seamus Coleman                                        App.js:23
Athlete = Rafa Nadal                                            App.js:23
Athlete = Shane Lowery                                          App.js:23
Athlete = Michael Jordan                                        App.js:23
```

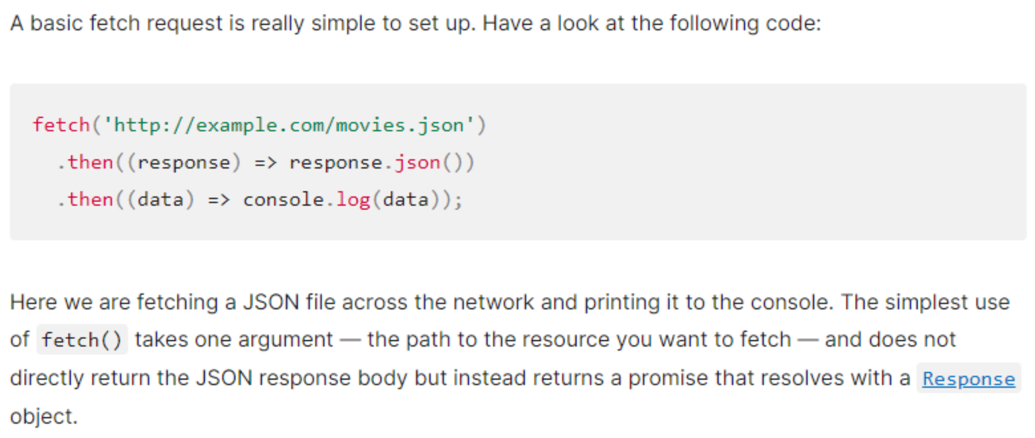


```
const xhttp = new XMLHttpRequest()

xhttp.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200) {
        const data = JSON.parse(this.responseText)
        // handle the response that is saved in variable data
        console.log(`XMLHttpRequest data ${data}`);
        data.map( (element)=> {
            console.log(`Athlete = ${element.name}`);
        })
    }
}

xhttp.open('GET', 'http://localhost:3001/athletes', true)
xhttp.send()
```

- The newer `fetch` API provides an simpler interface for fetching resources.
- Familiar to anyone who has used `XMLHttpRequest`, but the new API provides a more powerful and flexible feature set.

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

- `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch`



A basic fetch request is really simple to set up. Have a look at the following code:

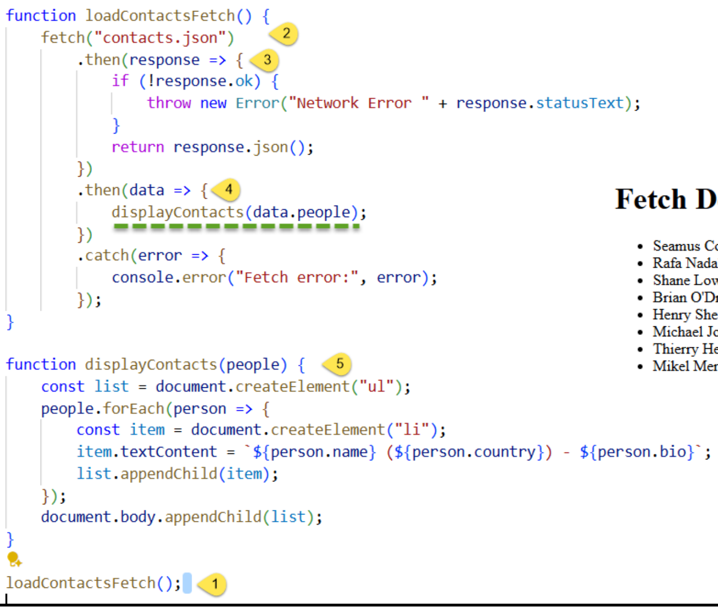
```
fetch('http://example.com/movies.json')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

Here we are fetching a JSON file across the network and printing it to the console. The simplest use of `fetch()` takes one argument — the path to the resource you want to fetch — and does not directly return the JSON response body but instead returns a promise that resolves with a `Response` object.

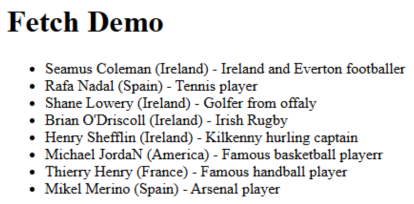
fetch API

```
fetch(url)
    .then(function() {
      // handle the response
    })
    .catch(function() {
      // handle the error
    });
```

fetch demo



```
function loadContactsFetch() {
    fetch("contacts.json")
        .then(response => {
            if (!response.ok) {
                throw new Error("Network Error " + response.statusText);
            }
            return response.json();
        })
        .then(data => {
            displayContacts(data.people);
        })
        .catch(error => {
            console.error("Fetch error:", error);
        });
}

function displayContacts(people) {
    const list = document.createElement("ul");
    people.forEach(person => {
        const item = document.createElement("li");
        item.textContent = `${person.name} (${person.country}) - ${person.bio}`;
        list.appendChild(item);
    });
    document.body.appendChild(list);
}

loadContactsFetch();
```



Fetch Demo

- Seamus Coleman (Ireland) - Ireland and Everton footballer
- Rafa Nadal (Spain) - Tennis player
- Shane Lowery (Ireland) - Golfer from offaly
- Brian O'Driscoll (Ireland) - Irish Rugby
- Henry Shefflin (Ireland) - Kilkenny hurling captain
- Michael JordaN (America) - Famous basketball playerr
- Thierry Henry (France) - Famous handball player
- Mikel Merino (Spain) - Arsenal player

Fetch Vs jQuery Ajax

- The **Promise** returned from `fetch()` won't reject on HTTP error status even if the response is an `HTTP 404` or `500`. Instead, it will resolve normally (with ok status set to false), and it will only reject on network failure or if anything prevented the request from completing.
- `fetch()` won't can receive cross-site cookies; you can't can establish a cross site session using fetch. Set-Cookie headers from other sites are silently ignored.
- `fetch()` won't send cookies, unless you set credentials: 'same-origin'.

Axios

- Axios provides a cleaner, promise-based syntax that reduces boilerplate, automatically parses JSON responses, and includes built-in error handling for HTTP status codes.
- promise-based HTTP client for the browser and node.js.
- Takes advantage of async and await for more readable and asynchronous code.
- Need
  - `npm install axios`

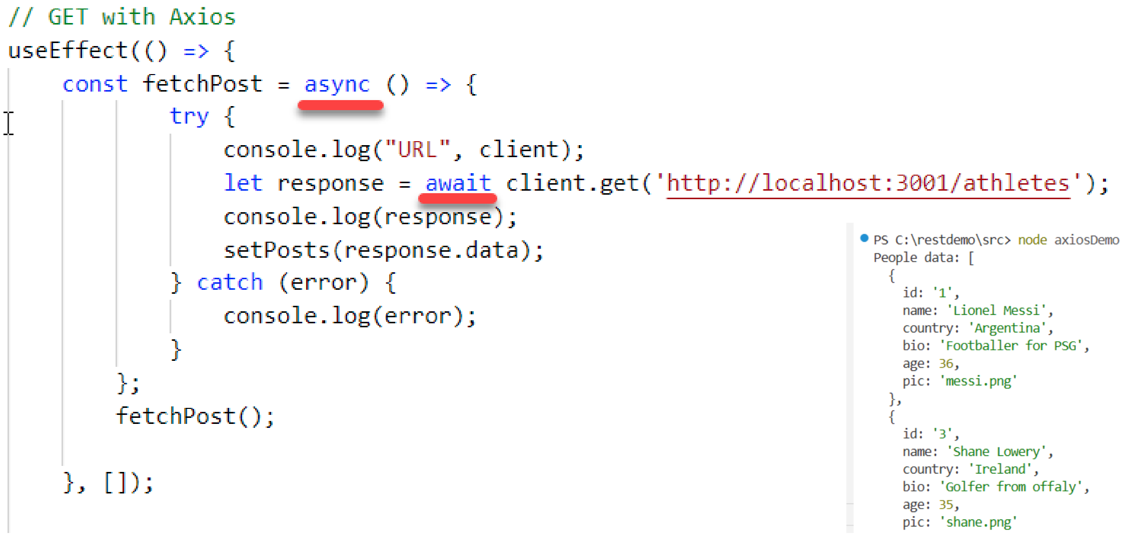
Axios HTTP requests

- `axios.request(config)`
- `axios.get(url[, config])`
- `axios.delete(url[, config])`
- `axios.head(url[, config])`
- `axios.options(url[, config])`
- `axios.post(url[, data[, config]])`
- `axios.put(url[, data[, config]])`
- `axios.patch(url[, data[, config]])`

```
const client = axios.create({
    baseURL: 'http://localhost:3000/athletes'
});
```

Simple axios post example

```
axios.post('/athlete', {
        id: 5,
        name: 'Pele',
        country: 'Brazil',
        income: 234234
      })
.then((response) => {
    console.log(response);
  }, (error) => {
    console.log(error);
  });
```

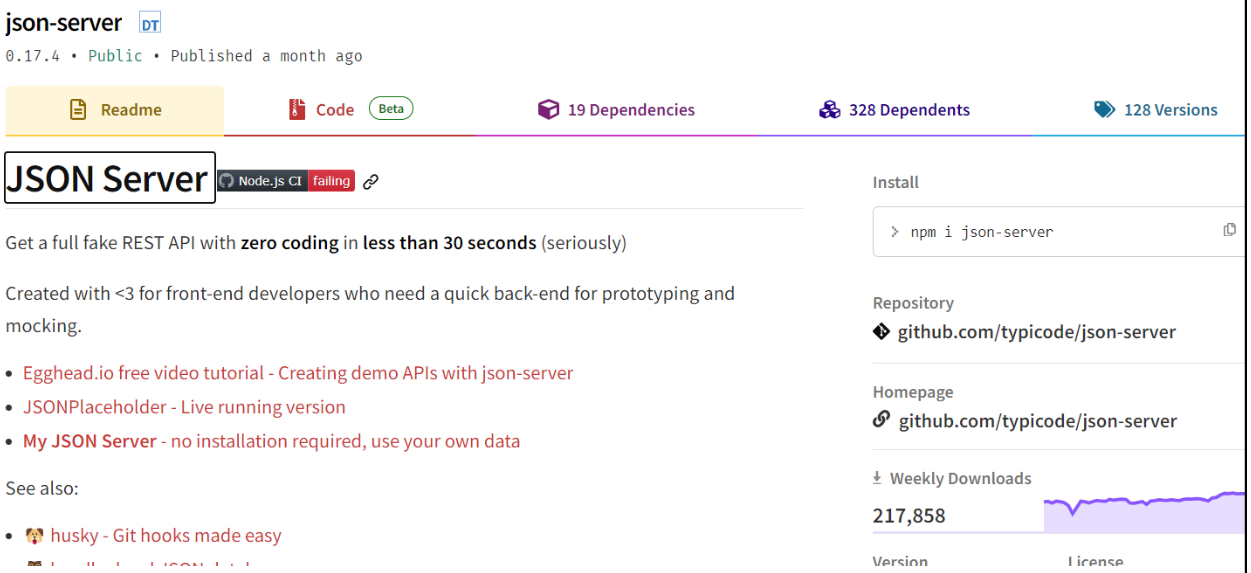


```
// GET with Axios
useEffect(() => {
    const fetchPost = async () => {
        try {
            console.log("URL", client);
            let response = await client.get('http://localhost:3001/athletes');
            console.log(response);
            setPosts(response.data);
        } catch (error) {
            console.log(error);
        }
    };
    fetchPost();

}, []);
```

```
PS C:\restdemo\src> node axiosDemo
People data: [
  {
    id: '1',
    name: 'Lionel Messi',
    country: 'Argentina',
    bio: 'Footballer for PSG',
    age: 36,
    pic: 'messi.png'
  },
  {
    id: '3',
    name: 'Shane Lowery',
    country: 'Ireland',
    bio: 'Golfer from offaly',
    age: 35,
    pic: 'shane.png'
```

JSON Server

- To use the REST API with React we need to create our own API in frontend.
- To create a fake API, with some sort of links, we need to have some sort of response that can test our frontend.
- This is where **JSON Server** come into play



json-server DT
0.17.4 • Public • Published a month ago

JSON Server Node.js CI failing

Get a full fake REST API with **zero coding** in **less than 30 seconds** (seriously)

Created with <3 for front-end developers who need a quick back-end for prototyping and mocking.

- Egghead.io free video tutorial - Creating demo APIs with json-server
- JSONPlaceholder - Live running version
- **My JSON Server** - no installation required, use your own data

See also:

- husky - Git hooks made easy

Install

```
> npm i json-server
```

Repository
github.com/typicode/json-server

Homepage
github.com/typicode/json-server

Weekly Downloads
217,858

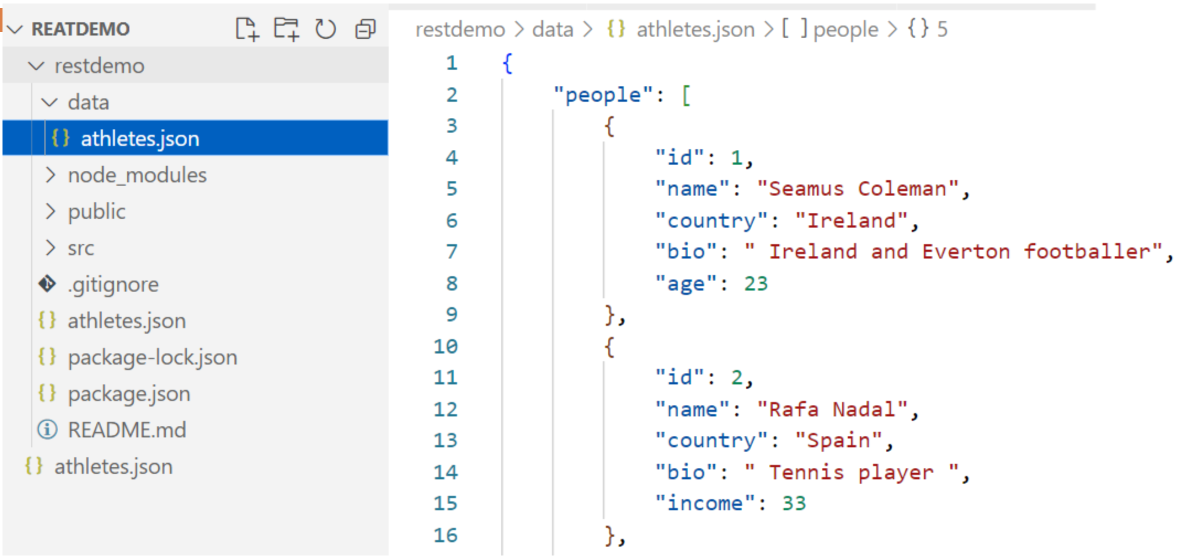
Version License

Install JSON Server

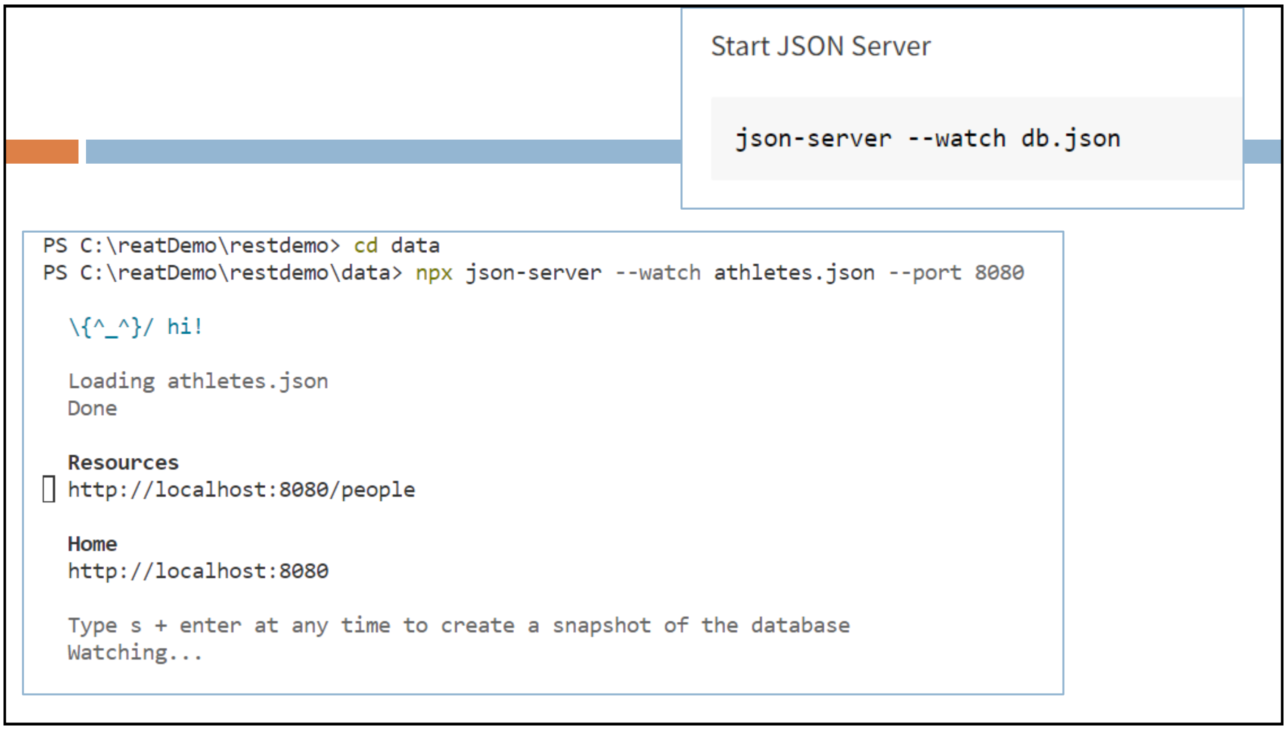
```
npm install -g json-server
```

Create a `db.json` file with some data

```
{
  "posts": [
    { "id": 1, "title": "json-server", "author": "typicode" }
  ],
  "comments": [
    { "id": 1, "body": "some comment", "postId": 1 }
  ],
  "profile": { "name": "typicode" }
}
```



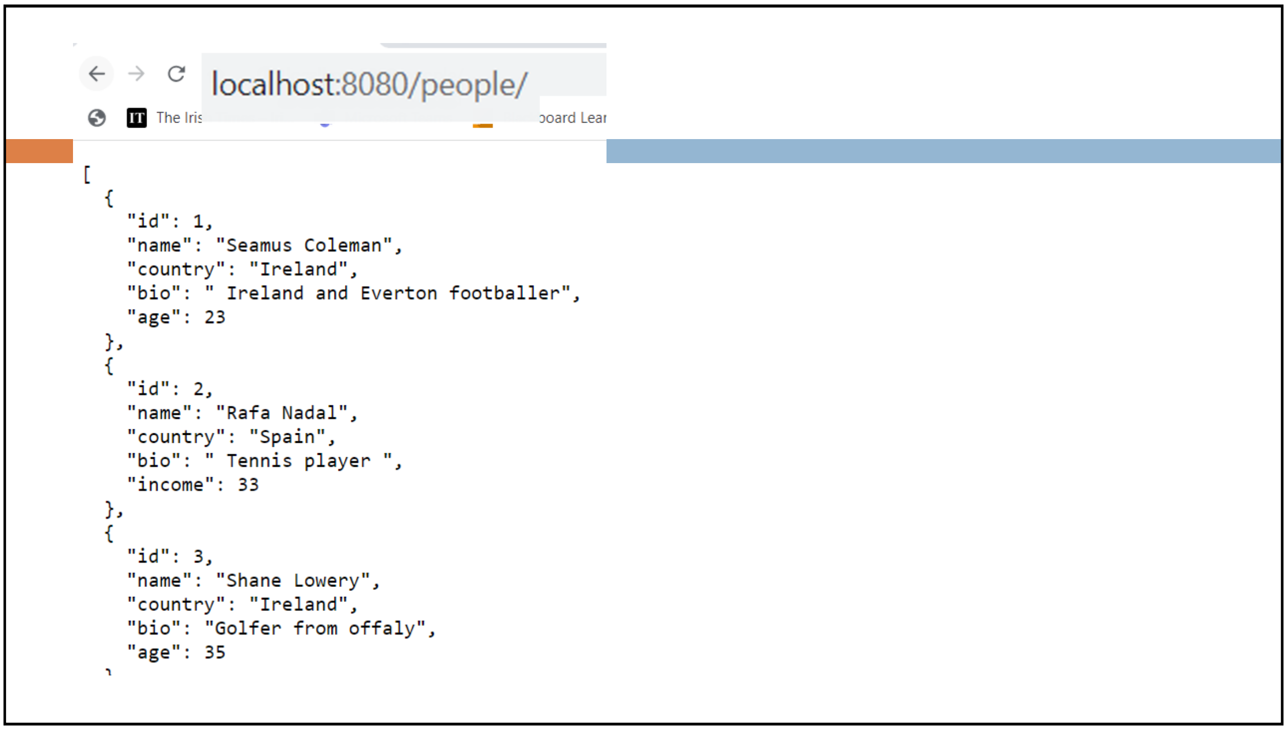
```
{
    "people": [
        {
            "id": 1,
            "name": "Seamus Coleman",
            "country": "Ireland",
            "bio": " Ireland and Everton footballer",
            "age": 23
        },
        {
            "id": 2,
            "name": "Rafa Nadal",
            "country": "Spain",
            "bio": " Tennis player ",
            "income": 33
        },
```



Start JSON Server

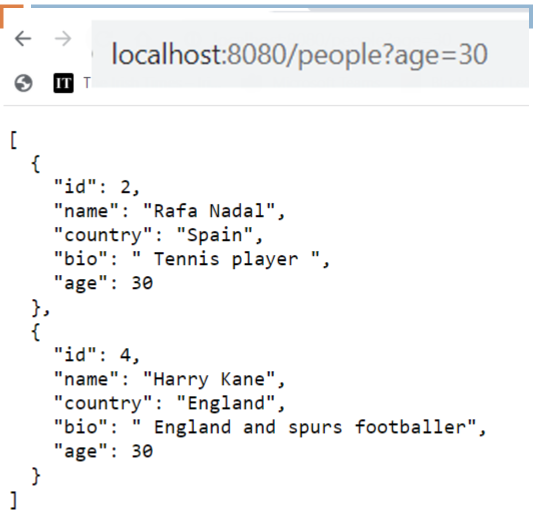
```
json-server --watch db.json
```

```
PS C:\reatDemo\restdemo> cd data
PS C:\reatDemo\restdemo\data> npx json-server --watch athletes.json --port 8080

  \{^_^}/ hi!

  Loading athletes.json
  Done

  Resources
  http://localhost:8080/people

  Home
  http://localhost:8080

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



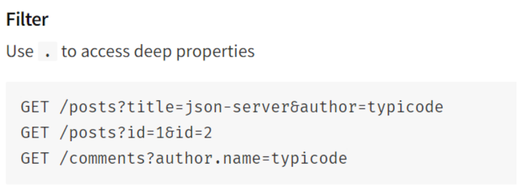
localhost:8080/people/

```
[
  {
    "id": 1,
    "name": "Seamus Coleman",
    "country": "Ireland",
    "bio": " Ireland and Everton footballer",
    "age": 23
  },
  {
    "id": 2,
    "name": "Rafa Nadal",
    "country": "Spain",
    "bio": " Tennis player ",
    "income": 33
  },
  {
    "id": 3,
    "name": "Shane Lowery",
    "country": "Ireland",
    "bio": "Golfer from offaly",
    "age": 35
```



localhost:8080/people?age=30

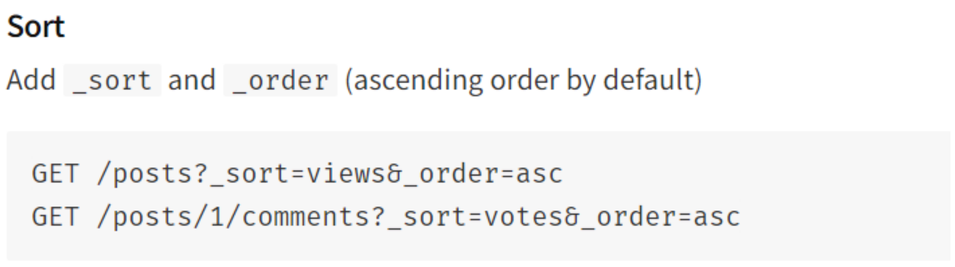
```
[
  {
    "id": 2,
    "name": "Rafa Nadal",
    "country": "Spain",
    "bio": " Tennis player ",
    "age": 30
  },
  {
    "id": 4,
    "name": "Harry Kane",
    "country": "England",
    "bio": " England and spurs footballer",
    "age": 30
  }
]
```



Filter

Use `.` to access deep properties

```
GET /posts?title=json-server&author=typicode
GET /posts?id=1&id=2
GET /comments?author.name=typicode
```



Sort

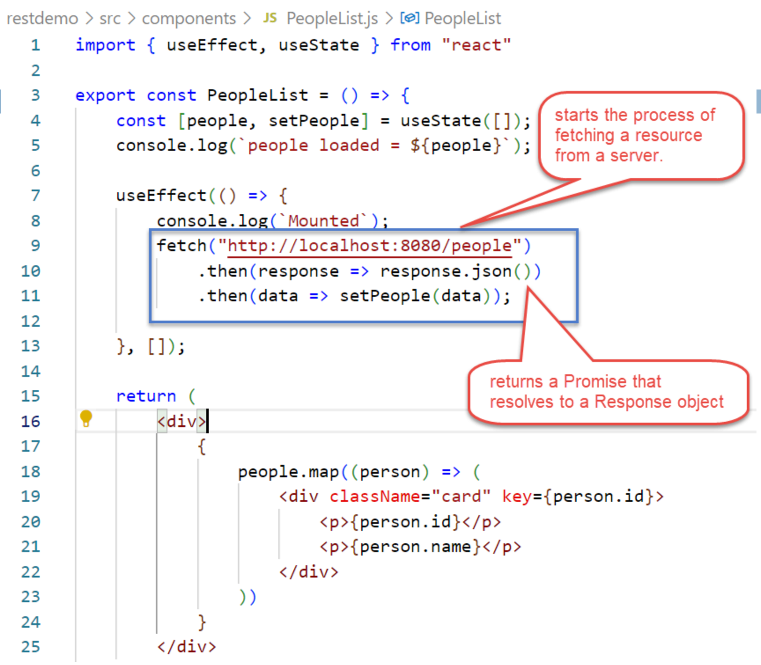
Add `_sort` and `_order` (ascending order by default)

```
GET /posts?_sort=views&_order=asc
GET /posts/1/comments?_sort=votes&_order=asc
```

Fetch API

```
fetch(url)
.then(function() {
  // handle the response
})
.catch(function() {
  // handle the error
});
```

- To test react application we need to run two servers.

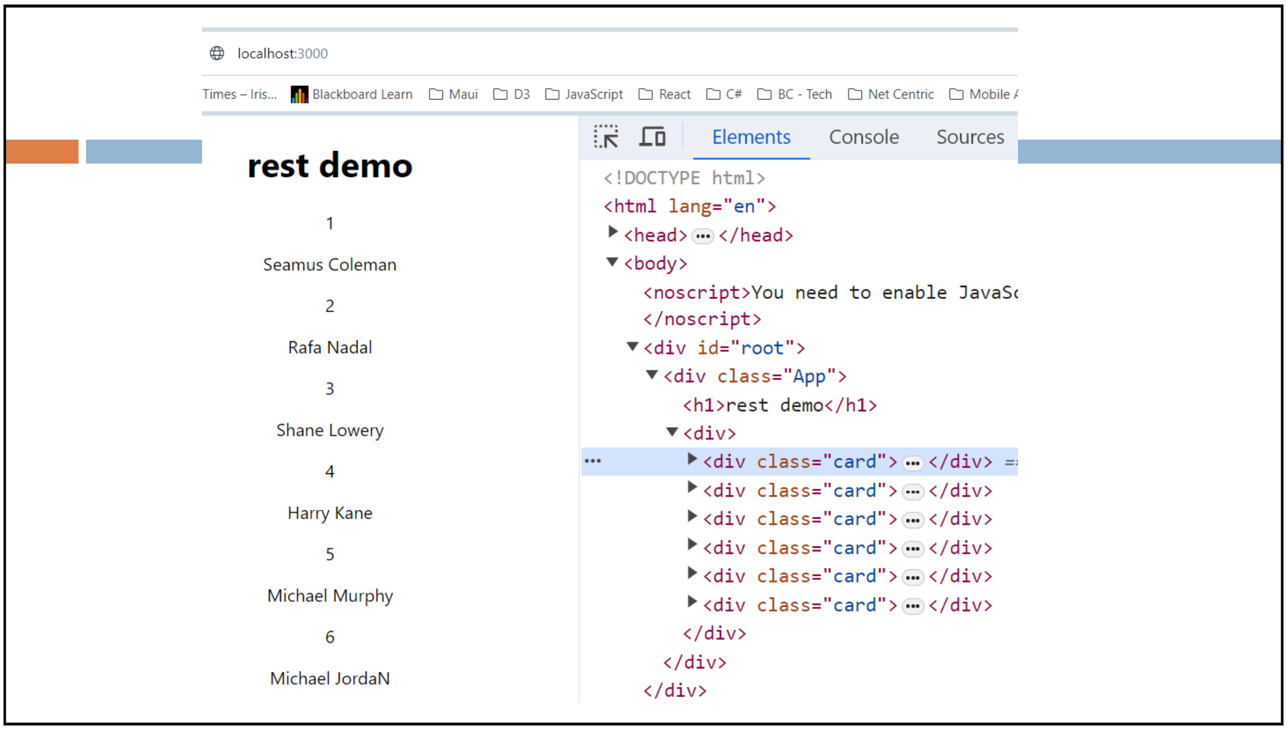


restdemo > src > components > PeopleList.js > PeopleList

```
import { useEffect, useState } from "react"

export const PeopleList = () => {
    const [people, setPeople] = useState([]);
    console.log(`people loaded = ${people}`);

    useEffect(() => {
        console.log(`Mounted`);
        fetch("http://localhost:8080/people")
            .then(response => response.json())
            .then(data => setPeople(data));

    }, []);

    return (
        <div>
            {
                people.map((person) => (
                    <div className="card" key={person.id}>
                        <p>{person.id}</p>
                        <p>{person.name}</p>
                    </div>
                ))
            }
        </div>
```

starts the process of fetching a resource from a server.

returns a Promise that resolves to a Response object


localhost:3000
Times – Iris... Blackboard Learn Maui D3 JavaScript React C# BC - Tech Net Centric Mobile
Elements Console Sources

rest demo

1

Seamus Coleman

2

Rafa Nadal

3

Shane Lowery

4

Harry Kane

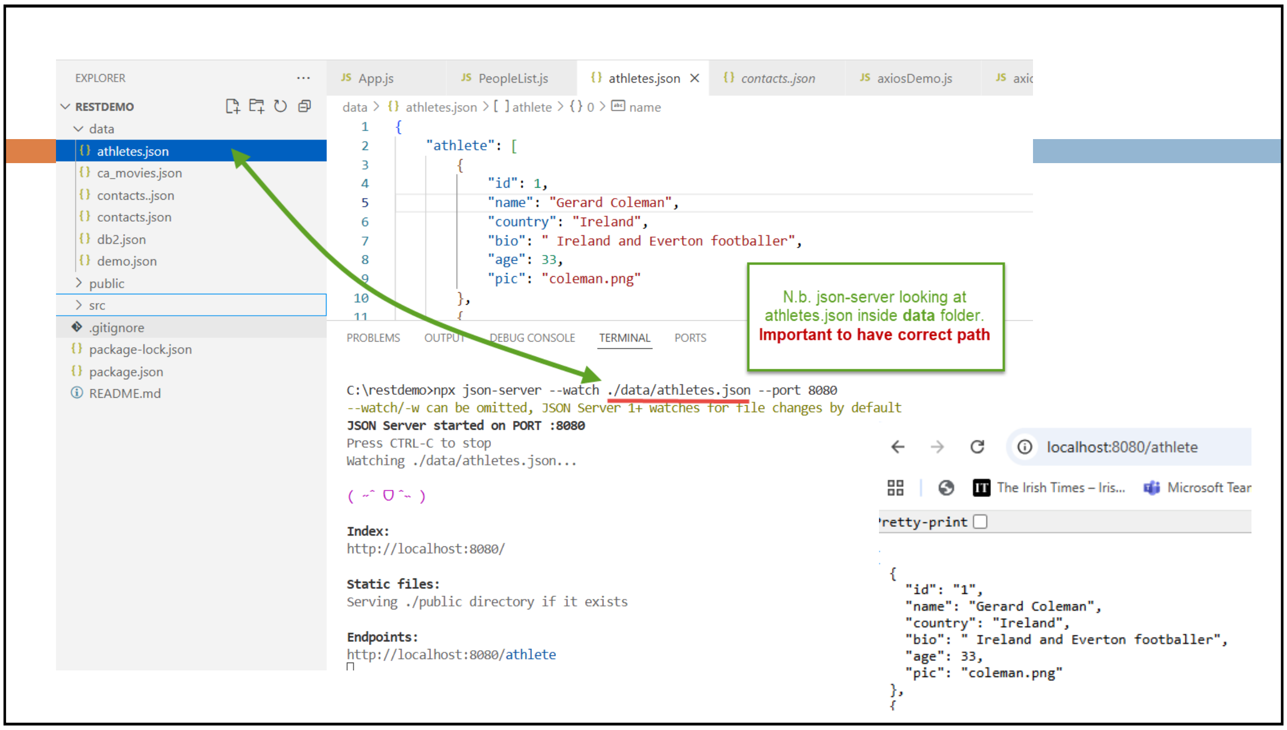
5

Michael Murphy

6

Michael JordaN

```
<!DOCTYPE html>
<html lang="en">
▶<head> ••• </head>
▼<body>
   <noscript>You need to enable JavaSc
   </noscript>
  ▼<div id="root">
    ▼<div class="App">
       <h1>rest demo</h1>
      ▼<div>
        ▶<div class="card"> ••• </div> ==
        ▶<div class="card"> ••• </div>
        ▶<div class="card"> ••• </div>
        ▶<div class="card"> ••• </div>
        ▶<div class="card"> ••• </div>
        ▶<div class="card"> ••• </div>
       </div>
     </div>
   </div>
```



```
{
    "athlete": [
        {
            "id": 1,
            "name": "Gerard Coleman",
            "country": "Ireland",
            "bio": " Ireland and Everton footballer",
            "age": 33,
            "pic": "coleman.png"
        },
        {
```

N.b. json-server looking at athletes.json inside **data** folder.
Important to have correct path

```
C:\restdemo>npx json-server --watch ./data/athletes.json --port 8080
--watch/-w can be omitted, JSON Server 1+ watches for file changes by default
JSON Server started on PORT :8080
Press CTRL-C to stop
Watching ./data/athletes.json...

( ˶ˆ ᗜ ˆ˵ )

Index:
http://localhost:8080/

Static files:
Serving ./public directory if it exists

Endpoints:
http://localhost:8080/athlete
```

```
{
  "id": "1",
  "name": "Gerard Coleman",
  "country": "Ireland",
  "bio": " Ireland and Everton footballer",
  "age": 33,
  "pic": "coleman.png"
},
{
```