Asynchronous Javascript

Author: Jawad Srour

Today’s Topics

1. Synchronous VS Asynchronous.
2. Importance of Asynchronous code.
3. Promises
4. APIs
5. REST APIs
6. HTTP Methods
7. Fetch API

Asynchronous VS Synchronous

Synchronous

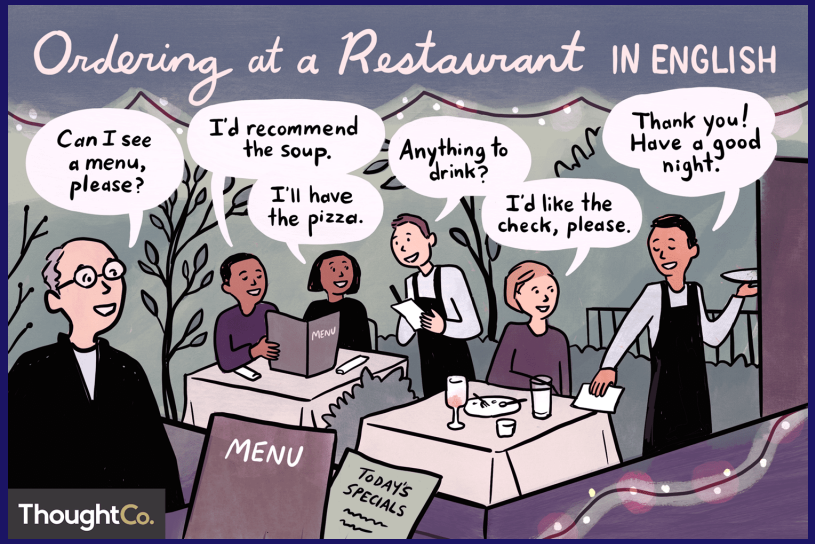
- Say you are waiting in a line to buy a ticket for a movie, you need to wait for whoever is in front of you to get done, to buy your ticket
- Queue (FIFO)
- Sequential execution of code (every statement of code gets executed one by one)
- Example



Asynchronous VS Synchronous

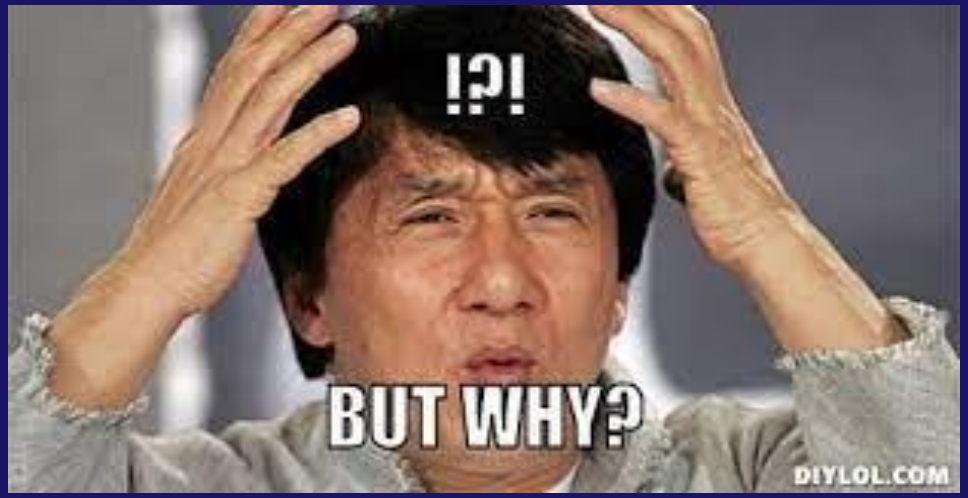
Asynchronous

- Say you are in a restaurant with many other people, you order your food while other people can order their food and don't need to wait for your food to be done.
- You can execute multiple things at the same time
- Code is not executing sequentially
- Example → setTimeOut with a delay



Importance of Asynchronous

- While users browse an application, important stuff can run in the background while not affecting the application's flow → better UX
- Example: downloading files on a website, we don't want the user's browser to freeze.



Promises



Promises

- Promises are asynchronous
- With Promises, we can defer the execution of a code block until an async request is completed.
- Represents a value that is unknown now, but may be known in the future → asynchronous value.
- Eventual completion or failure of an asynchronous operation.



Example

- Say you are ordering an uber, and your request is pending
- The driver either accepts your request and the promise is fulfilled
- Or rejects your request and the promise is rejected
- Either ways, your original promise/request got settled
- DEMO



The Fancy Word: API

API

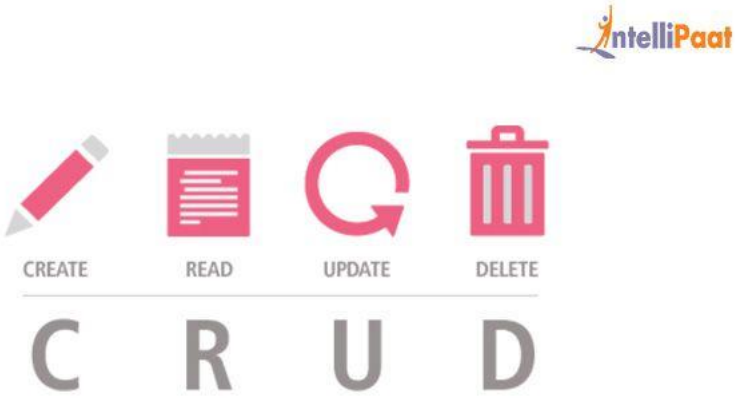
- Stands for application programming interface.
- Think of it as a butler (messenger).
- Delivers your request and gets the response back to you.
- It’s an interface offering some type of service.



REST APIs

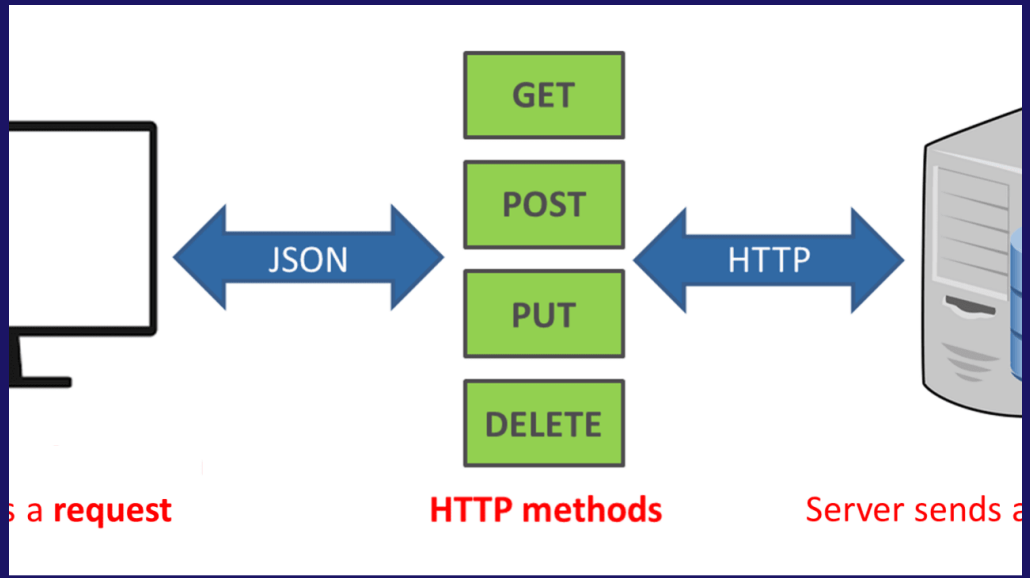
- REST stands for Representational state transfer.
- Allows for interacting with a server through CRUD operations
- CRUD Stands for: create, read, update, delete.
- Despite, the name note that REST is STATELESS, meaning it does not maintain state. (we use JWT instead of cookies commonly... future presentation!!)





HTTP Methods/Verbs

- GET → fetching data from server (example: checking my account information on a website)
- POST → storing new information on the back-end (example: registration for an account)
- PUT → updating a record (example: changing my password)
- DELETE → deleting a record (example: deleting my account)
- MAPPING CRUD TO HTTP VERBS:
  - CREATE = POST
  - READ = GET
  - UPDATE = PUT
  - DELETE = DELETE
- Server gets a request, and response is in JSON format (previously XML was used, but JSON is more lightweight, faster, and simpler to use).



POST

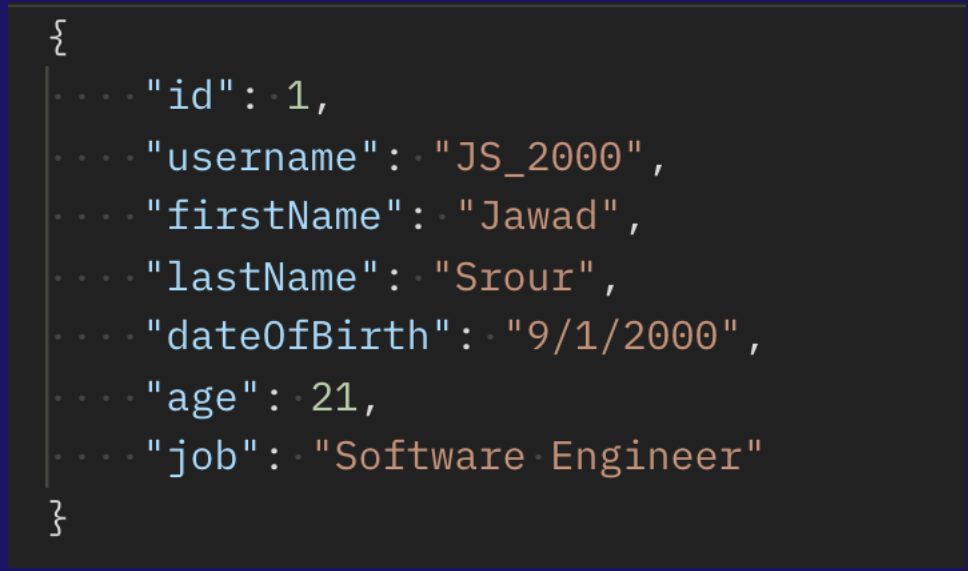
- The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.
- POST http://myserver.com/employees
- Request JSON body containing form data to be submitted.

```
{
    "username": "JS_2000",
    "password": "MY_SECRET_PW",
    "firstName": "Jawad",
    "lastName": "Srour",
    "dateOfBirth": "9/1/2000",
    "age": 21,
    "job": "Software Engineer"

}
```

GET

- The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- GET http://myserver.com/employees/1
- We are getting the employee of ID 1



PUT

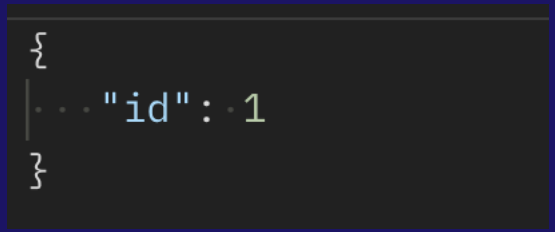
- The PUT method replaces all current representations of the target resource with the request payload.

1. PUT http://myserver.com/employees/1
2. Send JSON with new job description.
3. Server updates user of id 1 with the new job description.

```
{
    "id": 1,
    "username": "JS_2000",
    "firstName": "Jawad",
    "lastName": "Srour",
    "dateOfBirth": "9/1/2000",
    "age": 21,
    "job": "Senior Software Engineer"
}
```

DELETE

- The DELETE method deletes the specified resource.
- DELETE http://myserver.com/employees/1
- Delete user of id 1
- Deletes employee from database



Fetch API

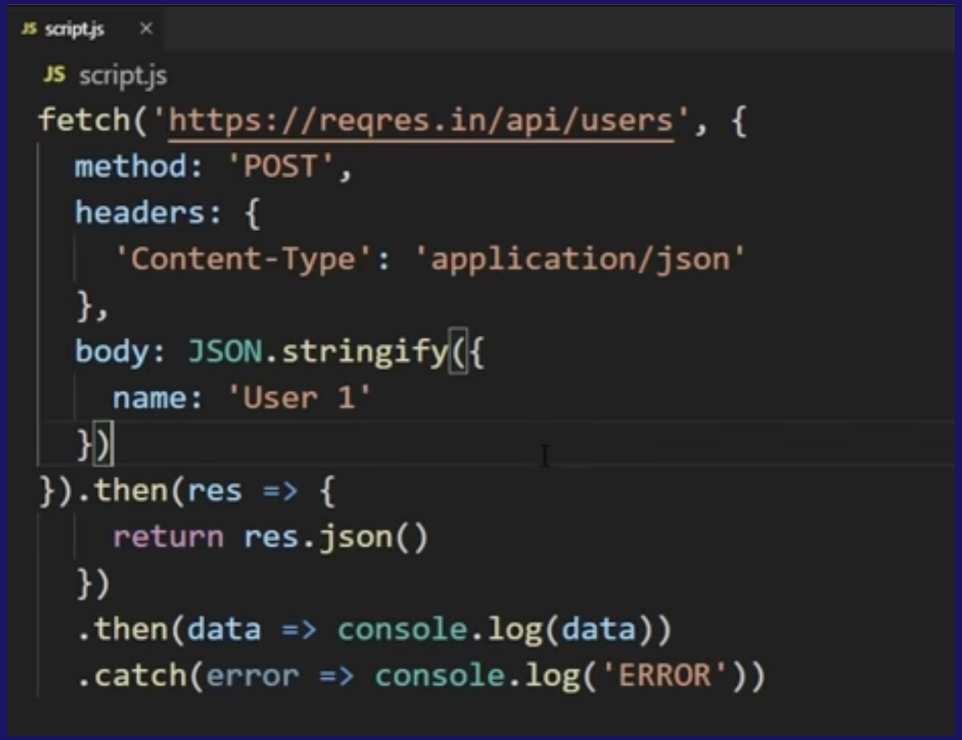
- The Fetch API provides an interface for fetching resources (including across the network)
- Fetch() method that works asynchronously to fetch data across the network
- Returns a promise object



DEMO



Fetch POST


script.js
script.js
fetch('https://reqres.in/api/users', {
method: 'POST',
headers: {
'Content-Type': 'application/json'
},
body: JSON.stringify({
name: 'User 1'
})
}).then(res => {
return res.json()
})
.then(data => console.log(data))
.catch(error => console.log('ERROR'))

References

1. https://stackoverflow.com/questions/748175/asynchronous-vs-synchronous-execution-what-is-the-main-difference
2. https://www.geeksforgeeks.org/synchronous-and-asynchronous-in-javascript/
3. https://www.youtube.com/watch?v=cuEtnrL9-H0