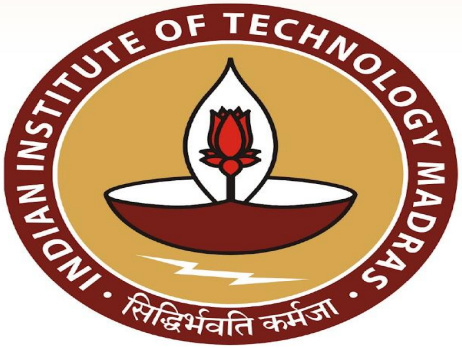

IIT Madras
BSc Degree

Using APIs

Separation of concerns

- Backend: manage data models
- Frontend: manage UI

Clean interaction mechanism to separate the two

Required

- System level design with separate backend and frontend
  - Backend should never know what UI looks like
  - No direct calls to HTML template rendering etc.
  - Data output only in neutral formats: JSON is preferred nowadays, but not essential
  - Data input through form data or URLs
- Fetch mechanism
  - How to retrieve data from a backend?
  - URL based APIs
- Rendering mechanism
  - Frontend can be rendered on server and pushed
  - Frontend implemented in browser, pulls data

Fetch - Asynchronous

- Fetching data depends on factors outside server control
  - Latency to backend
  - Network load, disruptions
- Should not make browser hang if correct data not available
- Asynchronous operation:
  - Start fetch in background
  - Wait for results, update

How?

Async

- Events and Callbacks
- Promises
- fetch API
- axios

Callbacks

- Function `doSomething` takes a long time to execute
- `let result = doSomething()`
  - Entire JS interpreter blocks till result is obtained
  - JS is a single threaded system - browser will hang
- Instead start doSomething and tell it to call us back when done

Events

- `button onclick` handler?
- This is a function
  - But never explicitly “called” - not imperative code
  - How to specify when to call?
- Event callback
  - Specify to DOM: on particular event, invoke function

JS: Event loops and call stacks

Call Stack

- Execute all operations (function calls etc) in present scope in sequence
- Go check “callback queue” to see if any new functions to be called
  - If so, execute them
  - Keep checking… events can be pushed to queue later by timeouts etc.
- https://html.spec.whatwg.org/multipage/webappapis.html#event-loops
- https://blog.sessionstack.com/how-javascript-works-event-loop-and-the-rise-of-async-programming-5-ways-to-better-coding-with-2f077c4438b5
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

Callbacks

- Higher order functions call other functions depending on some conditions
- Example:

```
doSomething(successCB, failureCB) {
    let result = doLongComputation();
    if (result) successCB();  // called as function
    else failureCB();
}
```

Promise: alternative syntax?

```
doSomething().then(successCB, failureCB);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

- More than just syntax
  - Additional guarantees on behaviour
  - Easier chaining

Concurrency vs Parallelism

- Concurrent: multiple operations can be in process at the same time
  - But maybe they only execute in time-multiplexed manner and not literally at same time instant
- Parallel: multiple concurrent operations are actually physically executing at the same time
- Parallel requires concurrent, not vice versa

Async operations bring in notion of concurrency - whether this is actually implemented in parallel or time multiplexed is up to run time

- Web workers, timers - parallel execution

Async op: fetch

- Fetching a URL must be async
  - No guarantee on network speeds
  - Server load may result in slow responses
  - Broken connection or other network failures can happen
- JS API since ES6: `fetch()`
  - Implemented using `Promise`
  - Built into most browsers - “polyfills” available for backward compatibility

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

Axios

Custom API library with similar functionality to fetch

- Can be used on most browsers - provides good backward compatibility
- Also works on nodejs

https://blog.logrocket.com/axios-or-fetch-api/

Existing APIs

Building a frontend

- Many public and useful APIs already exist
- Significant development possible with just API access
- Examples
  - Open Weathermap
  - HackerNews
  - Wikipedia
  - Github