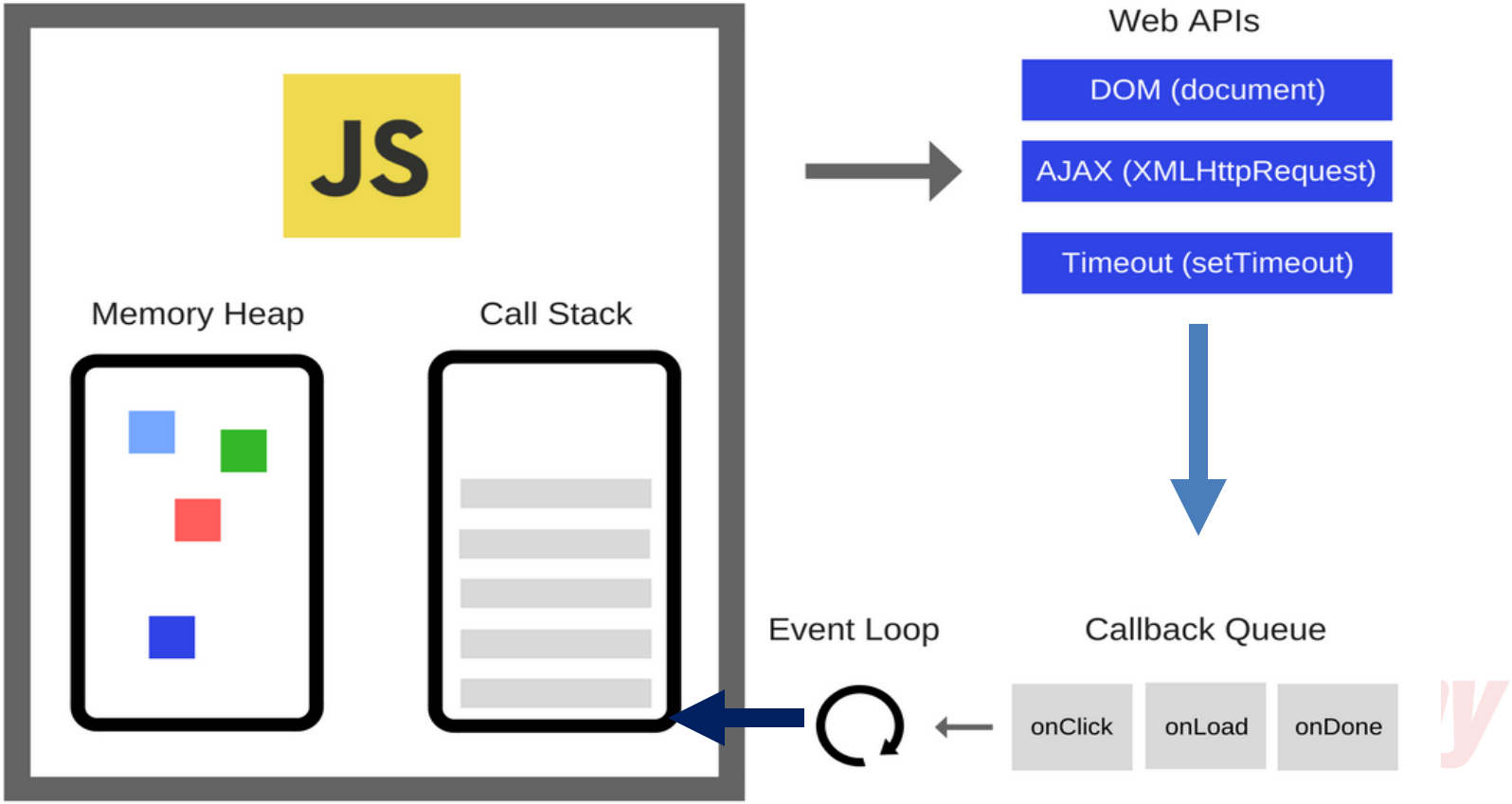
6. Website Building

Adrian Adiaconitei

LINK Academy

Obiective

- ✓ Recapitulare
- ✓ Hărți web
- ✓ Website Building
  - ✓ CvTemplate



LINKAcademy

- ✓ **Memory Heap** (Grămada de memorie)- Este memoria alocată pentru variabile, obiecte și funcții. Heap este responsabil pentru stocarea datelor noastre.
- ✓ În JavaScript, tipurile primitive (cum ar fi number, string, boolean) sunt stocate în Stack, iar tipurile de referință (cum ar fi object, array, function) sunt stocate în Heap.

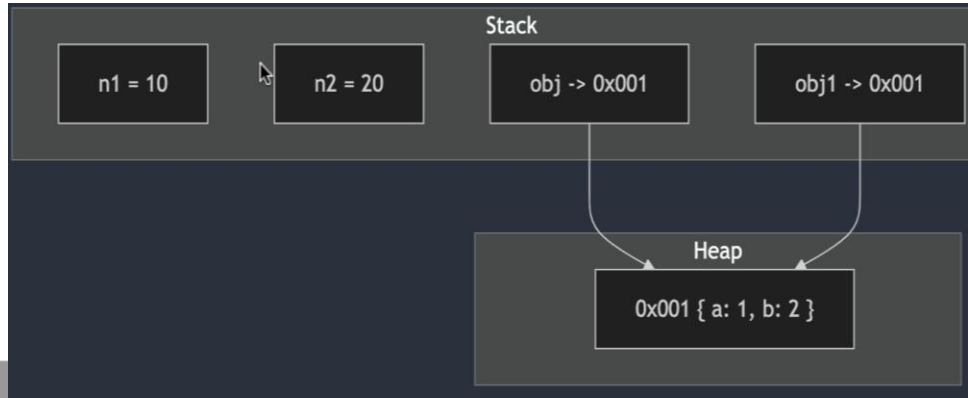
```
let n1 = 10;
let n2 = n1;
n2 = 20;
console.log(n1);
const obj = { a: 1, b: null };
let obj1 = obj;
obj1.b=2
```

Variabilele primitive (n1, n2) sunt stocate în Stack

→ n2 primește o copie a valorii lui n1.

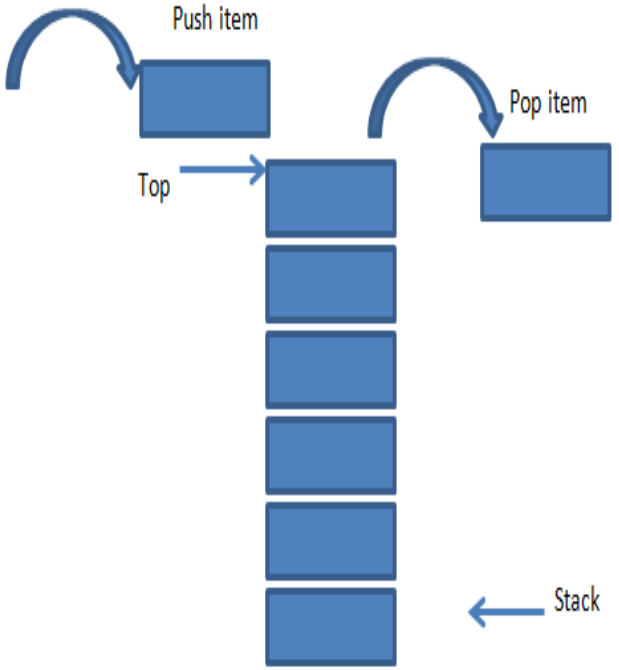
Obiectele (obj, obj1) sunt stocate în Heap

→ obj1 primește o referință către același obiect ca obj.



LINKAcademy

- ✓ **Call Stack** - Stivă de apeluri ne permite să efectuăm apeluri de funcție.
  - ✓ Ține evidența ce funcții au fost apelate și în ce ordine.
  - ✓ Când o funcție este apelată, intră în Stack. După execuție, este eliminată.
  - ✓ Este o stivă de tip LIFO (last in, first out)



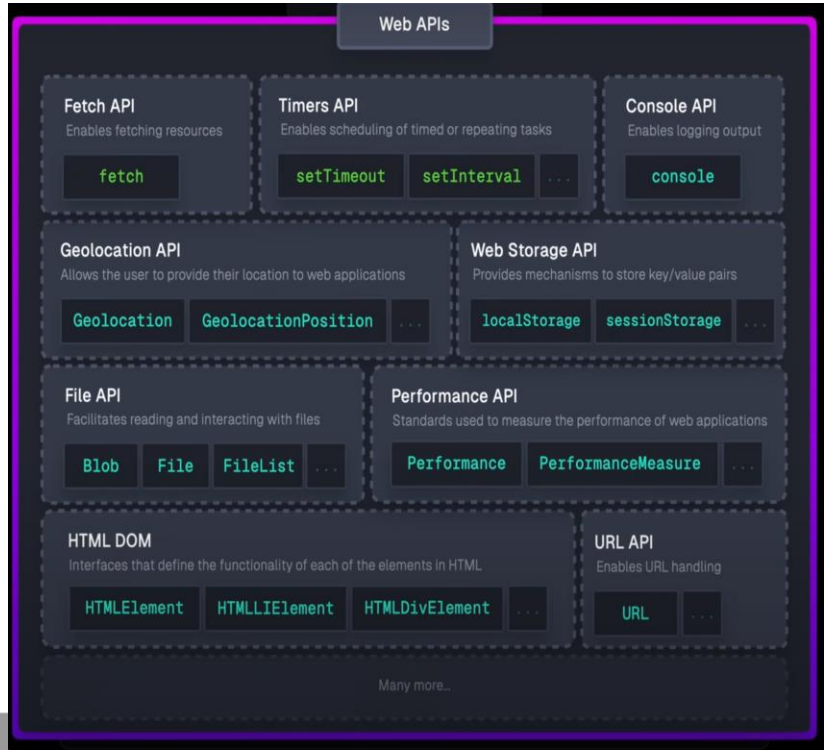
1. greet() intră în Stack
2. hello() intră în Stack
3. console.log("Hello") rulează → hello() iese din Stack
4. console.log("World") rulează → greet() iese din Stack

```
function hello() { console.log("Hello"); }
function greet() { hello(); console.log("World");
greet();
```

LINKAcademy

- ✓ **Web APIs** - sunt funcționalități oferite de browser (sau de mediul de execuție, cum ar fi Node.js) care permit interacțiunea cu sistemul, rețeaua și interfața utilizatorului. Aceste API-uri **nu sunt parte din JavaScript**, ci sunt oferite de browser prin intermediul motorului JavaScript (ex: V8, SpiderMonkey, JavaScriptCore).

1. Când apelezi o funcție asincronă (ex: setTimeout, fetch, addEventListener), browser-ul o gestionează în **Web APIs**.
2. După finalizare, rezultatul funcției este trimis în **Macrotask Queue** sau **Microtask Queue**, în funcție de tipul operației.
3. **Event Loop** verifică dacă **Call Stack** este gol și mută sarcinile finalizate din cozi în **Call Stack** pentru execuție.



✓ **Event Loop** - Buclă de evenimente procesează Sarcini și Microsarcini

1. **Evaluare Script**: Execută sincron scriptul

2. **Task Queue** - Rulează sarcina cea mai veche din coada, până când coada de apeluri este goală. Task Queue este de tip FIFO. Dacă **Call Stack** ține evidența funcțiilor care se execută chiar acum, atunci Task Queue ține evidența funcțiilor care vor fi executate în viitor.

3. **Microtask Queue** Rulează toate Microsarcinile: cel mai veche Microsarcină din coada,până când coda este goală.

Microtask Queue este de tip FIFO (first in, first out) . Gestiona apelurile de tip Promise.(ES6)

4. **Macrotask Queue** Rulează toate Macrosarcinile: cel mai veche Macrosarcină din coada. Gestiona apelurile de setTimeout

5. Redați continutul în interfața de utilizare: console / browser.

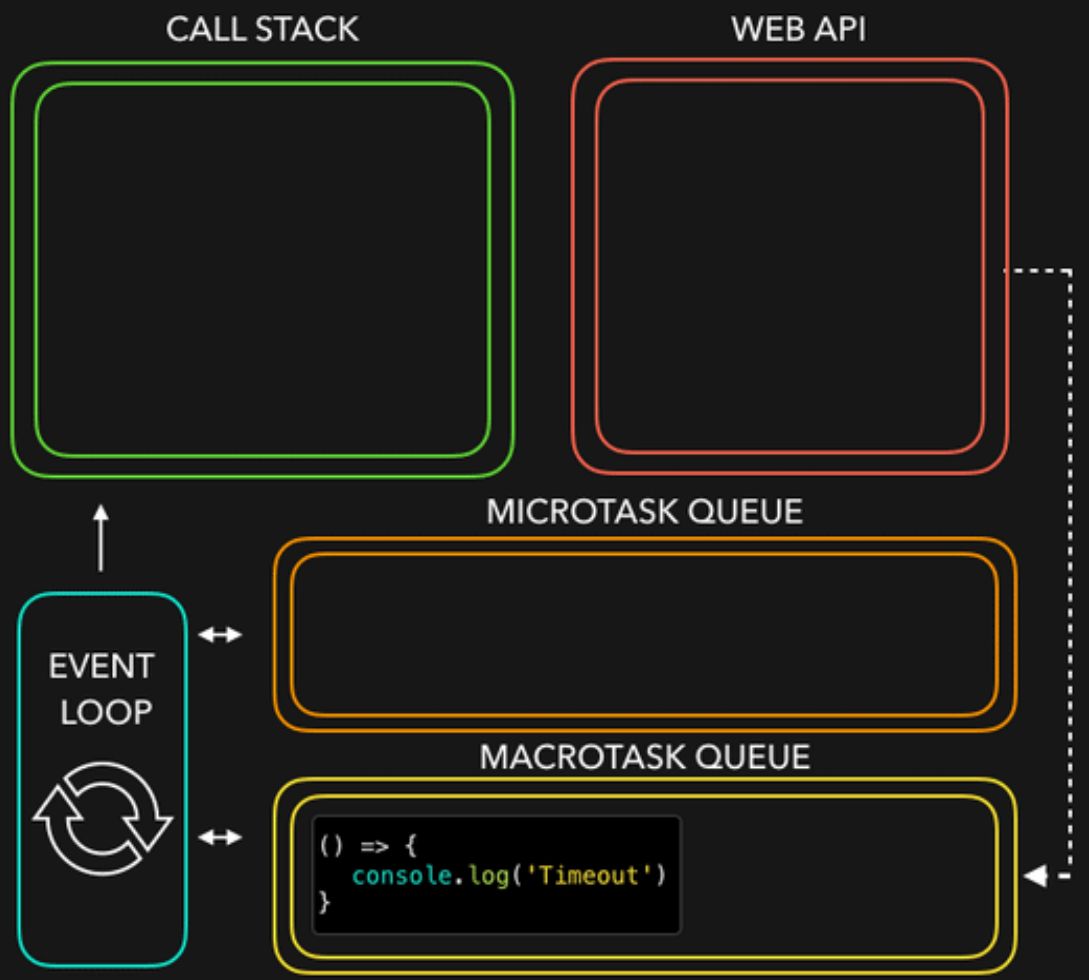
Apoi, reveniți la pasul 2

LINKAcademy

```
console.log('Start!')

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
   .then(res => console.log(res))

console.log('End!')
```

node

```
Start!
End!
Promise!
```



- ✓ https://deepu.tech/memory-management-in-v8/
- ✓ https://felixgerschau.com/javascript-memory-management/
- ✓ https://dev.to/lydiahallie/javascript-visualized-promises-async-await-5gke

https://www.jsv9000.app/

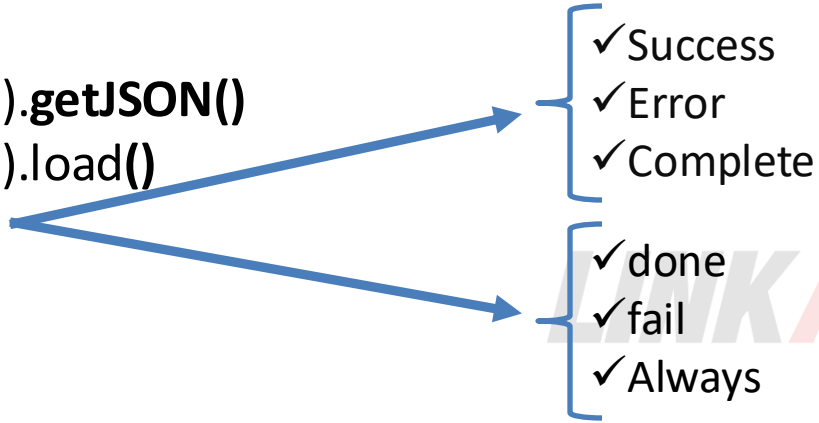
- ✓ **Call Stack** → Execută funcții sincrone (LIFO).
- ✓ **Web APIs** → Rulează operațiuni asincrone.
- ✓ **Microtask Queue** → Execută Promise.then() , async/await , fetch(), XMLHttpRequest() **înainte** de setTimeout().
- ✓ **Macrotask Queue** → Execută setTimeout(), setInterval(), addEventListener etc.
- ✓ **Event Loop** → Coordonează execuția task-urilor.
- ✓ **JavaScript rulează rapid codul sincronic, apoi procesează task-urile asincrone pas cu pas!**

Ordinea execuției în JavaScript

1. Codul **sincron** rulează prima dată.
2. Apoi, se rulează **Microtask Queue** (Promise.then()).
3. La final, **Macrotask Queue** (setTimeout, setInterval).

recapitulare

- ✓ JavaScript este un limbaj **single-threaded**, ceea ce înseamnă că execută **un singur task la un moment dat** în Call Stack. Însă, datorită **Event Loop-ului**, poate gestiona și operațiuni asincrone eficient.
- ✓ JavaScript
  - ✓ XMLHttpRequest
  - ✓ **Fetch**
- ✓ jQuery
  - ✓ $(”selector").**getJSON()**
  - ✓ $(”selector").load**()**
  - ✓ **$.ajax()**
  - ✓ $.get()
  - ✓ $.post()



recapitulare

1. Ce afișează următorul cod?

```
function fifth() { console.log('fifth')}
function fourth() { fifth();  console.log('fourth')
}
function third() { fourth(); console.log('third')
}
function second() { third(); console.log('second')
}
function first() { second(); console.log('first')
}
first();
```

recapitulare

2. Ce afișează următorul cod?

```
setTimeout(function a() {console.log('sunt in a()');}, 1000);

setTimeout(function b() {console.log('sunt in b()');}, 500);

setTimeout(function c() {console.log('sunt in c()');}, 0);

function d() {console.log('sunt in d()');}
d();
```

recapitulare

3. Ce afișează următorul cod?

```
fetch('https://www.google.ro')
.then(function a() {console.log('google succes')
}).catch(() => {console.log('google reject')
});
Promise.resolve()
.then(function b() {console.log('promisiunea 1 rezolvata')
});
Promise.reject()
.catch(function c() {console.log('promisiunea 2 respinsa')
});
```

recapitulare

4. Ce afișează următorul cod?

```
setTimeout(function a() {
console.log('Set timeout este executat');
}, 0);

Promise.resolve().then(function b() {
console.log('Promisiunea este executata');
});
```

recapitulare

5. Ce afișează următorul cod?

```
Promise.resolve()
.then(function a() {
Promise.resolve().then(function c() { console.log('Promisiunea 1 este executata');
});
})
.then(function b() {
Promise.resolve().then(function d() {console.log('Promisiunea 2 este executata');
});
});
```

recapitulare

6. Ce afișează următorul cod?

```
console.log("1. Start");
// Web API: setTimeout (Macrotask Queue)
setTimeout(() => console.log("2. Timeout"), 0);
// Web API: Promise (Microtask Queue)
Promise.resolve().then(() => console.log("3. Promise"));
// Web API: Event Listener (Macrotask Queue)
document.body.addEventListener("click", () => console.log("4. Click event"));
// Web API: setTimeout (Macrotask Queue)
setTimeout(() => console.log("5. Timeout 2"), 0);
console.log("6. End");
```

app1

Hărți web: Google

- ✓ https://developers.google.com/maps/documentation/javascript/adding-a-google-map#key
- ✓ https://developers.google.com/maps/documentation/javascript/markers#maps_marker_animations-javascript

Hărți web: Google

- ✓ https://codepen.io/artux/pen/PoPEMvW
- ✓ https://codepen.io/dylanvann/pen/yNWdxJ

Hărți web: OpenStreetMap + Leaflet

https://www.openstreetmap.org

https://leafletjs.com/

https://leafletjs.com/download.html

app2

Hărți web: OpenStreetMap + Leaflet

https://codepen.io/Tomik23/pen/MWKvwzz

https://codepen.io/ryancui/pen/VXLQba?editors=1111

https://codepen.io/bocko/pen/OworMY

Harta.html

Magazin online

Provocare: Folosind acest API

https://makeup-api.herokuapp.com/

https://makeup-api.herokuapp.com/api/v1/products.json?brand=maybelline

Afișați infomrațiile primite

Bootstrap

Aplicație: CvTemplate - folosind exemplele din curs să facem o aplicație de tip onepage: index.html

- ✓ Navbar/Scrollspy
- ✓ Contact: Form Validation
  - ✓ Google captcha
- ✓ Carousel –
  - ✓ Optimizare imagini încărcate în funcție de dispozitiv
- ✓ Modal și Tooltip/Popovers
- ✓ Darck mode și Go to top
- ✓ Folosim.scss Optimizăm(CSS, JS, Imagini) folosind: webpack/gulp
- ✓ [Git API Repo]

Resurse

https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/

https://makeup-api.herokuapp.com/

https://www.jsv9000.app/

http://latentflip.com/loupe/

https://262.ecma-international.org/9.0/#sec-intro

https://graceydev.hashnode.dev/enabling-https-for-live-server-visual-studio-code-extension

https://github.com/webisora/vscode-liveserver-https