

# NODE.JS

ITS 2024-25

# INSTALLA NODE.JS

## Prerequisiti

- GIT: <https://git-scm.com>

## Installazione consigliata

- Installa Volta: <https://volta.sh>
- Apri git bash:
  - `curl https://get.volta.sh | bash`
  - `volta install node`

# CREA UN NUOVO PROGETTO NODE.JS

- Crea una nuova cartella
- Apri git bash e posizionati nella cartella del progetto
  - `npm init`
    - segui le istruzioni
  - `volta pin node:` imposta la versione corrente di Node.js

# MONOREPO (NPM WORKSPACES)

Node.js permette di raggruppare diversi progetti in un unico repository git

```
repository-root
- /first-package
  - package.json <- package di una applicazine
- /second-package
  - package.json
- package.json <- package principale
```

# PACKAGE PRINCIPALE

```
{  
  ...  
  "workspaces": ["first-package", "second-package"],  
  ...  
}
```

# USARE NPM IN UN MONOREPO

```
npm run script -w first-package
```

Esegue un comando in un determinato pacchetto del monorepo

# RIPASSO DI JAVASCRIPT

# CONST

Il valore di una costante non può essere modificato

```
const invariabile = 5;  
invariabile = 6; // NO!  
// TypeError: Assignment to constant variable.
```



# CONST

Se la costante è un oggetto o un array, le sue proprietà o i suoi elementi possono essere modificati!

```
const nomeCostante = {}; // Oggetto  
  
nomeCostante.proprieta = 1; // OK:  
// ho modificato una proprietà dell'oggetto,  
// ma non l'oggetto in sè
```

# LET

Può cambiare nel tempo

```
let nomeLet = "valore";  
  
nomeLet = {  
  proprieta: "valore",  
  altra: 123  
};
```

Si può fare, ma non è consigliato modificare il tipo di dato di una variabile!

# VAR

Sintassi valida ma non più in uso

```
var nomeVariabile = "valore";
```

# OGGETTI

```
const myObject = {  
  proprieta1: "valore",  
  proprieta2: "altro valore"  
};
```

Accedere al valore di una proprietà:

```
// sintassi equivalenti  
const proprieta1 = myObject["proprieta1"];  
  
const proprieta2 = myObject.proprieta2;
```

Estrae una o più proprietà di un oggetto e le assegna a delle variabili

```
const { inputPath, url, postId, tags, ...postContent } = apiPost;
```

# ...SPREAD

```
const oldObject = {  
  "key1": "first value",  
  "key2": "second value",  
  "key3": "third value"  
};  
  
const myObject = {  
  "key0": "zeroth value",  
  ...oldObject  
};  
  
// > myObject: Object { key0: "zeroth value", key1: "first value"
```

# ARRAY

```
const myArray = ["valore 1", 123, true, "..."];
```

Accedere al valore in una determinata posizione:

```
const primoElementoDellArray = myArray[0]; // "valore 1"
```

# [DESTRUCTURING]

Estrae uno o più elementi di un array e assegnarli ad una variabile

```
const originalArray = ["primo", "secondo", 3];  
  
// destructuring  
const [primoElemento, secondoElemento, terzoElemento] = originalA  
  
// > primoElemento === "primo";  
// > secondoElemento === "secondo";  
// > terzoElemento === 3
```



# [DESTRUCTURING]

```
const originalArray = ["primo", "secondo", 3];  
  
// non devi per forza estrarre tutti gli elementi  
const [primoElemento, , terzoElemento] = originalArray;  
  
// > primoElemento === "primo"; terzoElemento === 3
```

# ...SPREAD

```
const myArray = [ 4, 5, 6 ];  
const myMergedArray = [ 1, 2, 3, ...myArray ];  
  
// > Array(6) [ 1, 2, 3, 4, 5, 6 ]
```

# METODI DEGLI ARRAY

forEach(callback)

```
myArray.forEach((el) => {  
  el = el + "stringa";  
});
```

Modifica ogni elemento dell'array, applicando la funzione di callback

```
myArray === myArray; // true:  
// vengono modificati gli elementi dell'array
```

# METODI DEGLI ARRAY

map(callback)

```
const newArray = myArray.map((el) => {  
  el = el + "stringa";  
  return el;  
})
```

Ritorna un nuovo array, con gli elementi modificati dalla callback

```
newArray === myArray; // false  
// il metodo map() genera un nuovo array
```

# ALTRI METODI DEGLI ARRAY

- **Sort**: riordina gli elementi dell'array
- **Find**: trova un elemento
- **Filter**: elimina alcuni elementi
- **Reduce**: elabora tutti gli elementi in un unico risultato
- **Join**: unisce due o più array

# APPROFONDISCI

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops\\_and\\_iteration](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration)

# FUNZIONI

```
const miaFunzione = function nomeFunzione() {  
  //...  
}  
  
const miaArrowFunction = () => {  
  //...  
}  
  
const funzioneConParametri = (param1, param2) => {  
  return param1 + " " + param2;  
}
```

# FIRST CLASS FUNCTIONS

Si può assegnare una funzione a una variabile

```
const foo = () => {  
  console.log("foobar");  
};  
foo();
```



# FIRST CLASS FUNCTIONS

Si può passare una funzione come argomento di un'altra funzione

```
function sayHello() {  
    return "Hello, ";  
}  
  
function greeting(helloMessage, name) {  
    console.log(helloMessage() + name);  
}  
  
// Pass `sayHello` as an argument to `greeting` function  
greeting(sayHello, "JavaScript!");  
  
// Hello, JavaScript!
```

# FIRST CLASS FUNCTIONS

Una funzione può ritornare una nuova funzione

```
function sayHello() {  
  return () => {  
    console.log("Hello!");  
  };  
}
```

# APPROFONDISCI

[https://developer.mozilla.org/en-US/docs/Glossary/First-class\\_Function](https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function)

# CURRYING

Funzioni che ritornano altre funzioni, con una sintassi fluida

```
const chiama = (nome) => (functionCognome) => {  
  return nome + " " + cognome;  
}  
  
// posso chiamarla così:  
chiama("Mario")("Bianchi");  
  
// oppure in diversi step,  
// assegnando a una variabile la funzione generata:  
const chiamaPaolo = chiama("Paolo");  
chiamaPaolo("Rossi");
```

# APPROFONDISCI

<https://it.javascript.info/currying-partial>

# COMPOSIZIONE DI FUNZIONI

```
const famigliaRossi = (nome) => {  
  return nome + " Rossi";  
}  
  
const famigliaBianchi = (nome) => {  
  return nome + " Bianchi";  
}  
  
const chiamaConNomeCompleto = (nome, functionCognome) => {  
  return functionCognome(nome);  
}  
  
chiamaConNomeCompleto("Mario", famigliaBianchi);  
chiamaConNomeCompleto("Paolo", famigliaRossi);
```

# **FUNZIONI ASINCRONE**

```
1 function salutamiDopo() {
2   setTimeout(() => {
3     console.log("Ciao");
4   }, 1000);
5 }
6
7 const chiamaDopo = (nome) => {
8   setTimeout(() => {
9     console.log("Ehi, " + nome + "!");
10   }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
```



```
6
7 const chiamaDopo = (nome) => {
8   setTimeout(() => {
9     console.log("Ehi, " + nome + "!");
10   }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
17
18 // Console:
19 // Inizio
20 // Fine
```

```
7     console.clear(); //clear console
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
17
18 // Console:
19 // Inizio
20 // Fine
21 // Ehi, Mario!
22 // Ciao
```

```
6  
7  const chiamaDopo = (nome) => {  
8    setTimeout(() => {  
9      console.log("Ehi, " + nome + "!");  
10   }, 0); // timeout = zero  
11 }  
12  
13 console.log("Inizio");  
14 salutaMiDopo();  
15 chiamaDopo("Mario");  
16 console.log("Fine");  
17  
18 // Console:  
19 // Inizio  
20 // Fine  
21 // Ehi, Mario!
```

```
7     console.log("Ehi, " + nome + "!");
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
17
18 // Console:
19 // Inizio
20 // Fine
21 // Ehi, Mario!
22 // Ciao
```

```
7 console.timeEnd('ops') // nome, fine, fine - inizio  
8     setTimeout(() => {  
9         console.log("Ehi, " + nome + "!");  
10     }, 0); // timeout = zero  
11 }  
12  
13 console.log("Inizio");  
14 salutamiDopo();  
15 chiamaDopo("Mario");  
16 console.log("Fine");  
17  
18 // Console:  
19 // Inizio  
20 // Fine  
21 // Ehi, Mario!  
22 // Ciao
```

```
7     console.clear(); // nome, cognome, età
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
17
18 // Console:
19 // Inizio
20 // Fine
21 // Ehi, Mario!
22 // Ciao
```

```
2   setTimeout(() => {
3     console.log("Ciao");
4   }, 1000);
5 }
6
7 const chiamaDopo = (nome) => {
8   setTimeout(() => {
9     console.log("Ehi, " + nome + "!");
10   }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
17
```

```
7     console.log("Ehi, " + nome + "!");
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
17
18 // Console:
19 // Inizio
20 // Fine
21 // Ehi, Mario!
22 // Ciao
```



```
1 function salutamiDopo() {
2   setTimeout(() => {
3     console.log("Ciao");
4   }, 1000);
5 }
6
7 const chiamaDopo = (nome) => {
8   setTimeout(() => {
9     console.log("Ehi, " + nome + "!");
10   }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
```

```
7     console.clear(); //clear console
8     setTimeout(() => {
9         console.log("Ehi, " + nome + "!");
10    }, 0); // timeout = zero
11 }
12
13 console.log("Inizio");
14 salutamiDopo();
15 chiamaDopo("Mario");
16 console.log("Fine");
17
18 // Console:
19 // Inizio
20 // Fine
21 // Ehi, Mario!
22 // Ciao
```

# APPROFONDISCI



JavaScript Event Loop

**COME POSSO CONTROLLARE IL FLUSSO DI  
UNA SEQUENZA DI FUNZIONI ASINCRONE?**

# CALLBACK

```
1 function saluta(cb) {
2   setTimeout(() => {
3     console.log("Ciao");
4     cb(); // <- esegue la callback
5   }, 1000);
6 }
7
8 const chiama = (nome) => {
9   setTimeout(() => {
10     console.log("Ehi, " + nome + "!");
11   }, 0);
12 }
13
14 saluta(() => chiama("Paolo"));
15
16 // console.log("Ciao");
```

# CALLBACK

```
3     console.log('Ciao',  
4     cb()); // <- esegue la callback  
5     }, 1000);  
6 }  
7  
8 const chiama = (nome) => {  
9     setTimeout(() => {  
10         console.log("Ehi, " + nome + "!");  
11     }, 0);  
12 }  
13  
14 saluta(() => chiama("Paolo"));  
15  
16 // Console:  
17 // Ciao  
18 // Ehi, Paolo!
```

# CALLBACK

```
1 function saluta(cb) {
2   setTimeout(() => {
3     console.log("Ciao");
4     cb(); // <- esegue la callback
5   }, 1000);
6 }
7
8 const chiama = (nome) => {
9   setTimeout(() => {
10     console.log("Ehi, " + nome + "!");
11   }, 0);
12 }
13
14 saluta(() => chiama("Paolo"));
15
16 // console.log("Ciao");
```

# CALLBACK

```
3     console.log('Ciao',  
4         cb(); // <- esegue la callback  
5     }, 1000);  
6 }  
7  
8 const chiama = (nome) => {  
9     setTimeout(() => {  
10         console.log("Ehi, " + nome + "!");  
11     }, 0);  
12 }  
13  
14 saluta(() => chiama("Paolo"));  
15  
16 // Console:  
17 // Ciao  
18 // Ehi, Paolo!
```



# CALLBACK

```
1 function saluta(cb) {
2   setTimeout(() => {
3     console.log("Ciao");
4     cb(); // <- esegue la callback
5   }, 1000);
6 }
7
8 const chiama = (nome) => {
9   setTimeout(() => {
10     console.log("Ehi, " + nome + "!");
11   }, 0);
12 }
13
14 saluta(() => chiama("Paolo"));
15
16 // console.log("Ciao");
```

# CALLBACK

```
1  setTimeout(() => {  
2    console.log("Ciao");  
3    cb(); // <- esegue la callback  
4  }, 1000);  
5  
6 }  
7  
8 const chiama = (nome) => {  
9   setTimeout(() => {  
10     console.log("Ehi, " + nome + "!");  
11   }, 0);  
12 }  
13  
14 saluta(() => chiama("Paolo"));  
15  
16 // Console:  
17 // Ciao
```

# CALLBACK

```
3     console.log('Ciao',  
4         cb(); // <- esegue la callback  
5     }, 1000);  
6 }  
7  
8 const chiama = (nome) => {  
9     setTimeout(() => {  
10         console.log("Ehi, " + nome + "!");  
11     }, 0);  
12 }  
13  
14 saluta(() => chiama("Paolo"));  
15  
16 // Console:  
17 // Ciao  
18 // Ehi, Paolo!
```

# ATTENZIONE

La funzione `chiama()` si aspetta un parametro.

Poiché la funzione `saluta()` non passa nessun parametro alla callback, `chiama("Paolo")` viene eseguita in una callback che non ha parametri!

```
saluta(/* nessun parametro */) => chiama("Paolo");
```

# PROMISE

```
1 function saluta() {  
2   return new Promise((resolve) => {  
3     setTimeout(() => {  
4       console.log("Ciao");  
5       resolve();  
6     }, 1000);  
7   });  
8 }  
9  
10 const chiama = (nome) => {  
11   return new Promise((resolve) => {  
12     setTimeout(() => {  
13       console.log("Ehi, " + nome + "!");  
14       resolve();  
15     }, 0);  
16   });  
17 }
```

# PROMISE

```
4     console.log("Ehi, " + nome + "!");  
5     resolve();  
6   }, 1000);  
7 });  
8 }  
9  
10 const chiama = (nome) => {  
11   return new Promise((resolve) => {  
12     setTimeout(() => {  
13       console.log("Ehi, " + nome + "!");  
14       resolve();  
15     }, 0);  
16   });  
17 }  
18  
19 saluta().then(() => chiama("Paolo"));
```

# PROMISE

```
1 function saluta() {
2   return new Promise((resolve) => {
3     setTimeout(() => {
4       console.log("Ciao");
5       resolve();
6     }, 1000);
7   });
8 }
9
10 const chiama = (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
```

# PROMISE

```
1 function saluta() {
2   return new Promise((resolve) => {
3     setTimeout(() => {
4       console.log("Ciao");
5       resolve();
6     }, 1000);
7   });
8 }
9
10 const chiama = (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
```



# PROMISE

```
3     setTimeout(() => {
4         console.log("Ciao");
5         resolve();
6     }, 1000);
7 });
8 }
9
10 const chiama = (nome) => {
11     return new Promise((resolve) => {
12         setTimeout(() => {
13             console.log("Ehi, " + nome + "!");
14             resolve();
15         }, 0);
16     });
17 }
18
```

# PROMISE

```
4     console.log("Ehi, " + nome + "!");  
5     resolve();  
6   }, 1000);  
7 });  
8 }  
9  
10 const chiama = (nome) => {  
11   return new Promise((resolve) => {  
12     setTimeout(() => {  
13       console.log("Ehi, " + nome + "!");  
14       resolve();  
15     }, 0);  
16   });  
17 }  
18  
19 saluta().then(() => chiama("Paolo"));
```

# APPROFONDISCI

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Pro](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Pro)

# ASYNC/AWAIT

```
1  async function saluta() {
2    return new Promise((resolve) => {
3      setTimeout(() => {
4        console.log("Ciao");
5        resolve();
6      }, 1000);
7    });
8  }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
```

# ASYNC/AWAIT

```
12     setTimeout(() => {  
13         console.log("Ehi, " + nome + "!");  
14         resolve();  
15     }, 0);  
16 });  
17 }  
18  
19 const main = async () => {  
20     console.log("Inizio");  
21     await saluta();  
22     await chiama("Paolo");  
23     console.log("Fine");  
24 }  
25  
26 main();
```

# ASYNC/AWAIT

```
12     setTimeout(() => {  
13         console.log("Ehi, " + nome + "!");  
14         resolve();  
15     }, 0);  
16 });  
17 }  
18  
19 const main = async () => {  
20     console.log("Inizio");  
21     await saluta();  
22     await chiama("Paolo");  
23     console.log("Fine");  
24 }  
25  
26 main();
```

# ASYNC/AWAIT

```
12     setTimeout(() => {  
13         console.log("Ehi, " + nome + "!");  
14         resolve();  
15     }, 0);  
16 });  
17 }  
18  
19 const main = async () => {  
20     console.log("Inizio");  
21     await saluta();  
22     await chiama("Paolo");  
23     console.log("Fine");  
24 }  
25  
26 main();
```

# ASYNC/AWAIT

```
12     setTimeout(() => {
13         console.log("Ehi, " + nome + "!");
14         resolve();
15     }, 0);
16 });
17 }
18
19 const main = async () => {
20     console.log("Inizio");
21     await saluta();
22     await chiama("Paolo");
23     console.log("Fine");
24 }
25
26 main();
```



# ASYNC/AWAIT

```
1  async function saluta() {
2    return new Promise((resolve) => {
3      setTimeout(() => {
4        console.log("Ciao");
5        resolve();
6      }, 1000);
7    });
8  }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
```

# ASYNC/AWAIT

```
1  async function saluta() {
2    return new Promise((resolve) => {
3      setTimeout(() => {
4        console.log("Ciao");
5        resolve();
6      }, 1000);
7    });
8  }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
```

# ASYNC/AWAIT

```
1  async function saluta() {
2    return new Promise((resolve) => {
3      setTimeout(() => {
4        console.log("Ciao");
5        resolve();
6      }, 1000);
7    });
8  }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
```

# ASYNC/AWAIT

```
12     setTimeout(() => {  
13         console.log("Ehi, " + nome + "!");  
14         resolve();  
15     }, 0);  
16 });  
17 }  
18  
19 const main = async () => {  
20     console.log("Inizio");  
21     await saluta();  
22     await chiama("Paolo");  
23     console.log("Fine");  
24 }  
25  
26 main();
```

# ASYNC/AWAIT

```
6     }, 1000);
7   });
8 }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
18
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
```

# ASYNC/AWAIT

```
6      }, 1000);
7    });
8  }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
18
19 const main = async () => {
20   console.log("Inizio");
21 }
```

# ASYNC/AWAIT

```
6 // ...
7 });
8 }
9
10 const chiama = async (nome) => {
11   return new Promise((resolve) => {
12     setTimeout(() => {
13       console.log("Ehi, " + nome + "!");
14       resolve();
15     }, 0);
16   });
17 }
18
19 const main = async () => {
20   console.log("Inizio");
21   await saluta();
22 }
```

# ASYNC/AWAIT

```
12     setTimeout(() => {  
13         console.log("Ehi, " + nome + "!");  
14         resolve();  
15     }, 0);  
16 });  
17 }  
18  
19 const main = async () => {  
20     console.log("Inizio");  
21     await saluta();  
22     await chiama("Paolo");  
23     console.log("Fine");  
24 }  
25  
26 main();
```



# TYPESCRIPT

E' un **superset** di javascript, cioè “arricchisce” javascript. In particolare, permette di descrivere dettagliatamente ogni tipo di dato e funzione.

- è integrato nell'editor e permette di segnalare errori nel codice
- può fare il **parsing** del codice per segnalare errori prima della sua esecuzione
- trasforma il codice typescript in javascript eseguibile nel browser o in node.js (**transpiling**)

Tutto il codice javascript è anche typescript valido.  
Typescript, invece deve essere compilato in javascript,  
per diventare codice javascript valido.

# ESEMPIO

```
type User = {  
  name: string,  
  id: number,  
}  
  
const user: User = {  
  username: "Hayes", // Error: it doesn't match the type  
  id: 0,  
};
```

# VANTAGGI PRATICI DI TYPESCRIPT

- fornisce suggerimenti durante la scrittura del codice
- permette di fare refactoring in modo sicuro
- permette di individuare errori di spelling
- aggiunge un controllo strict sui quirks&flaws di javascript
- in fase di compilazione, segnala errori che altrimenti non sarebbero rilevabili in javascript

*Typescript è uno standard per la  
realizzazione di applicazioni con un  
minimo di complessità.*

# LAVORARE CON MOLTI FILE



- Visually explained: Bundling in 60 seconds
- Documentazione di export
- Documentazione di `require()` in node.js

# IMPORT / EXPORT

```
// a.js
```

```
export const myFunction = () => { /* */ }  
export const myVariable = "value";
```

```
// b.js
```

```
// posso importare una o più variabili esportate da un file con e  
import { myFunction, myVariable } from "./a.js"
```

```
myFunction(myVariable);
```

# EXPORT DEFAULT

```
// a.js

export default myFunction = () => {
  // ...
}
```

```
// b.js
// posso assegnare il default alla variabile che voglio
import myRENAMEDFunction from "./a.js"

myRENAMEDFunction();
```



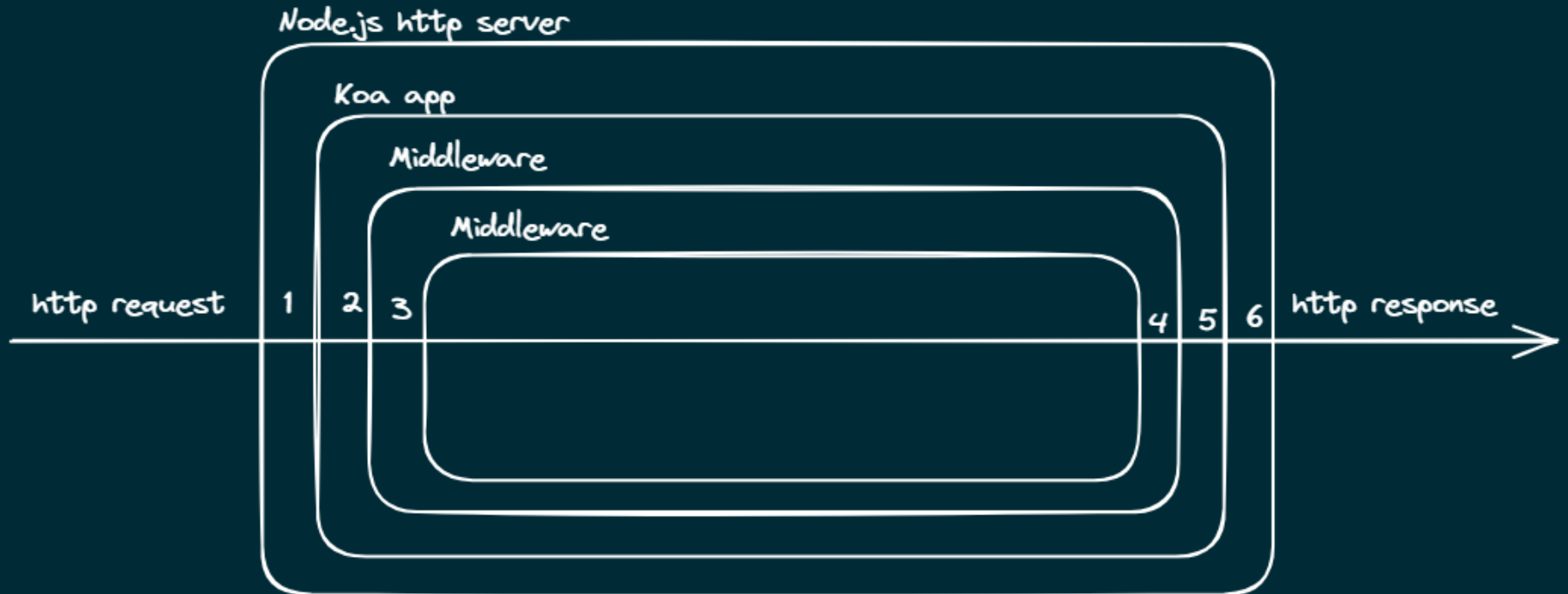
# FRAMEWORK NODE.JS PER APPLICAZIONI SERVER

- Express.js
- **Koa** (*useremo questo*)
- Fastify
- Nestjs
- ...molti altri

# KOA

```
import Koa from "koa";  
  
const app = new Koa();  
  
app.listen(3000); // fa partire il server
```

# MIDDLEWARE



```
async function Middleware(ctx, next) {  
  // ... fai qualcosa e modifica ctx (context)  
  await next(); // esegue il prossimo middleware  
  // ... esegue altro codice
```

```
// dopo l'esecuzione dei middleware innestati  
}
```

# MIDDLEWARE COME LIBRERIE

```
import Koa from "koa";
import serveStatic from "koa-static";
import cors from "@koa/cors";

const app = new Koa();

app.use(cors());
app.use(serveStatic(`./public`, {}));

app.use(async (ctx, next) => {
  console.log("Incoming HTTP request");
  await next();
});
```

# ROUTING

Il server deve comprendere l'url richiesto dall'utente e rispondere in modo appropriato per ciascun url

```
import Koa from "koa";
import Router from "@koa/router";

const app = new Koa();
const router = new Router();

// pagina html
router.get("/", (ctx) => {
  ctx.body = "<p>Home page</p>";
});

// pagina html
router.get("/come-funziona", (ctx) => {
  ctx.body = "<p>Una pagina interna</p>";
});
```

