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 raggrupare 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;
```

{DESTRUCTURING}

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

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

...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 val
```

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] = origin

// > 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

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

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-partials

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

```
function salutamiDopo() {
  setTimeout(() => {
    console.log("Ciao");
  }, 1000);
}

const chiamaDopo = (nome) => {
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0); // timeout = zero
}

console.log("Inizio");
salutamiDopo();
chiamaDopo("Mario");
```

```
 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
```

```
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0); // timeout = zero
}

console.log("Inizio");
salutamiDopo();
chiamaDopo("Mario");
console.log("Fine");

// Console:
// Inizio
// Fine
// Ehi, Mario!
// Ciao
```

```
const chiamaDopo = (nome) => {
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0); // timeout = zero
}

console.log("Inizio");
salutamiDopo();
chiamaDopo("Mario");
console.log("Fine");

// Console:
// Inizio
// Fine
// Ehi, Mario!
```

```
 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
```

```
 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
```

```
 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
```

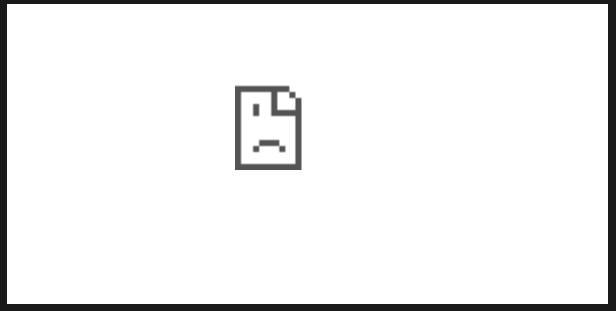
```
  setTimeout(() => {
    console.log("Ciao");
  }, 1000);
}

const chiamaDopo = (nome) => {
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0); // timeout = zero
}

console.log("Inizio");
salutamiDopo();
chiamaDopo("Mario");
console.log("Fine");
```

```
 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
```

```
function salutamiDopo() {
  setTimeout(() => {
    console.log("Ciao");
  }, 1000);
}

const chiamaDopo = (nome) => {
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0); // timeout = zero
}

console.log("Inizio");
salutamiDopo();
chiamaDopo("Mario");
```

```
 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

```
function saluta(cb) {
  setTimeout(() => {
    console.log("Ciao");
    cb(); // <- esegue la callback
  }, 1000);
}

const chiama = (nome) => {
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0);
}

saluta(() => chiama("Paolo"));
```

CALLBACK

```
    cb(); // <- esegue la callback
  }, 1000);
}

const chiama = (nome) => {
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0);
}

saluta(() => chiama("Paolo"));

// Console:
// Ciao
// 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
```

CALLBACK

```
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
```

CALLBACK

```
    console.log("Ciao");
    cb(); // <- esegue la callback
  }, 1000);
}

const chiama = (nome) => {
  setTimeout(() => {
    console.log("Ehi, " + nome + "!");
  }, 0);
}

saluta(() => chiama("Paolo"));

// Console:
// Ciao
```

CALLBACK

```
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

```
function saluta() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ciao");
      resolve();
    }, 1000);
  });
}

const chiama = (nome) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ehi, " + nome + "!");
      resolve();
    }, 0);
```

PROMISE

```
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

```
function saluta() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ciao");
      resolve();
    }, 1000);
  });
}

const chiama = (nome) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ehi, " + nome + "!");
      resolve();
    }, 0);
```

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);
```

PROMISE

```
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

```
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

ASYNC/AWAIT

```
async function saluta() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ciao");
      resolve();
    }, 1000);
  });
}

const chiama = async (nome) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ehi, " + nome + "!");
      resolve();
    }, 0);
```

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

```
async function saluta() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ciao");
      resolve();
    }, 1000);
  });
}

const chiama = async (nome) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ehi, " + nome + "!");
      resolve();
    }, 0);
```

ASYNC/AWAIT

```
async function saluta() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ciao");
      resolve();
    }, 1000);
  });
}

const chiama = async (nome) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ehi, " + nome + "!");
      resolve();
    }, 0);
```

ASYNC/AWAIT

```
async function saluta() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ciao");
      resolve();
    }, 1000);
  });
}

const chiama = async (nome) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ehi, " + nome + "!");
      resolve();
    }, 0);
```

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

```
  });
}

const chiama = async (nome) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Ehi, " + nome + "!");
      resolve();
    }, 0);
  });
}

const main = async () => {
  console.log("Inizio");
```

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");
```

ASYNC/AWAIT

```
 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

```
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

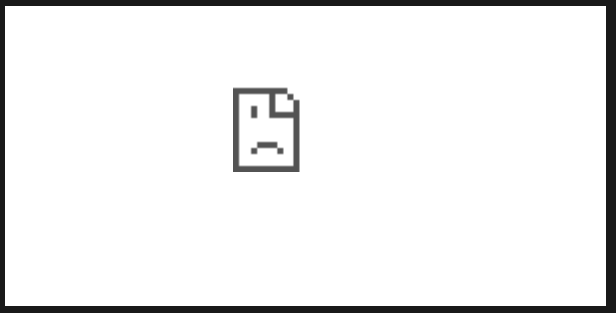
```
type User = {
  name: string,
  id: number,
}

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 co
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

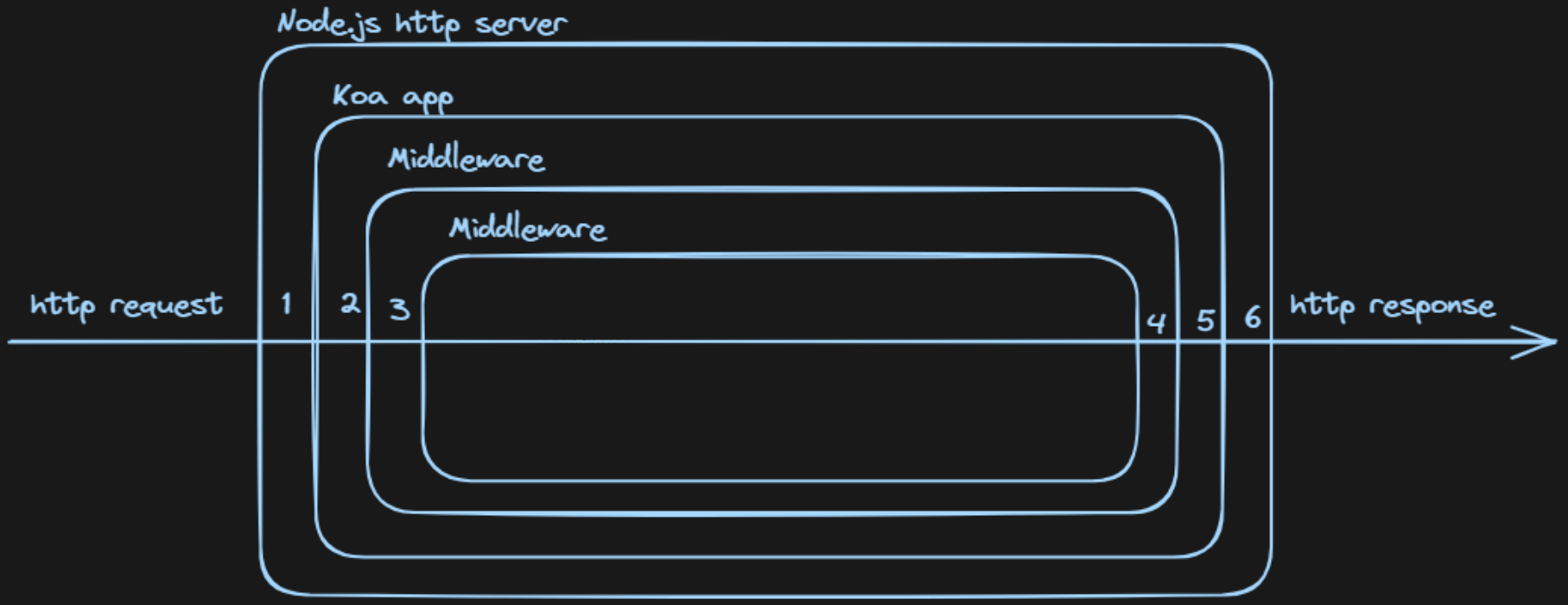
- Express.js
- Koa *(useremo questo)*
- Fastify
- Nestjs
- ...molti altri

FRAMEWORK HTTP

KOA

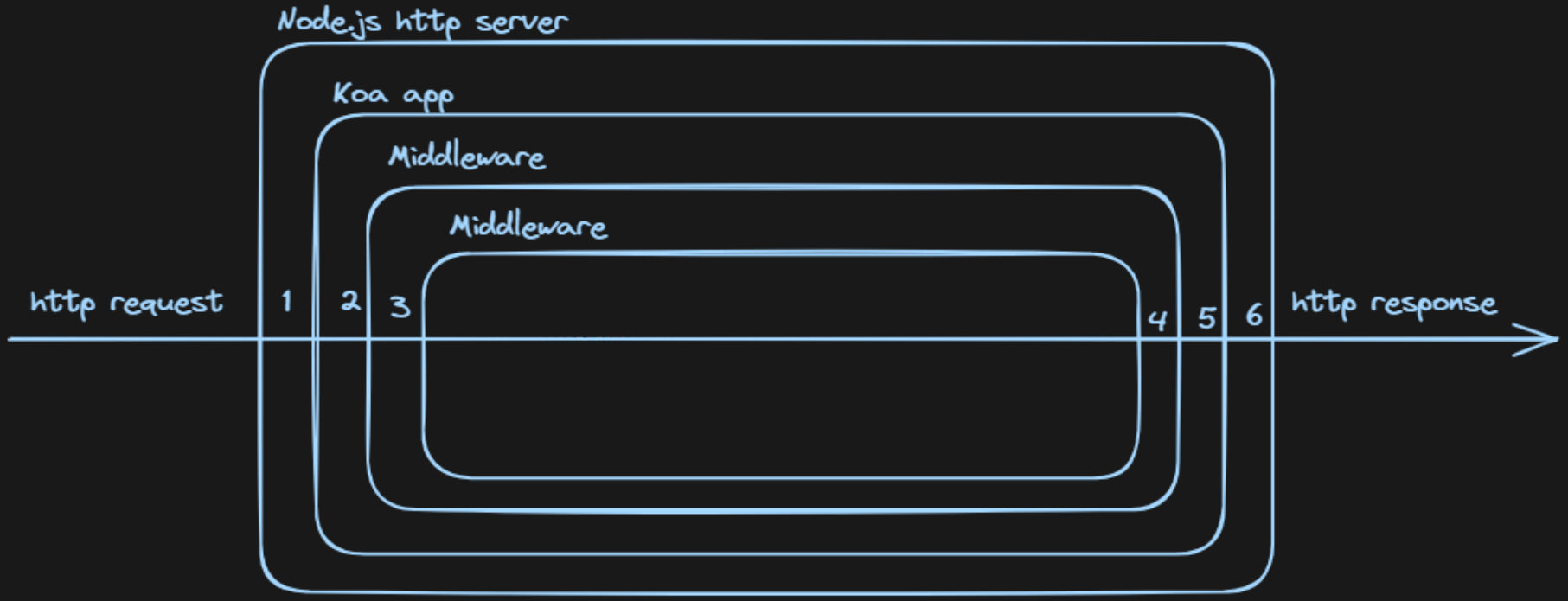
```
import Koa from "koa";

const app = new Koa();

app.listen(3000); // fa partire il server
```

MIDDLEWARE



Organizzazione a buccia di cipolla

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

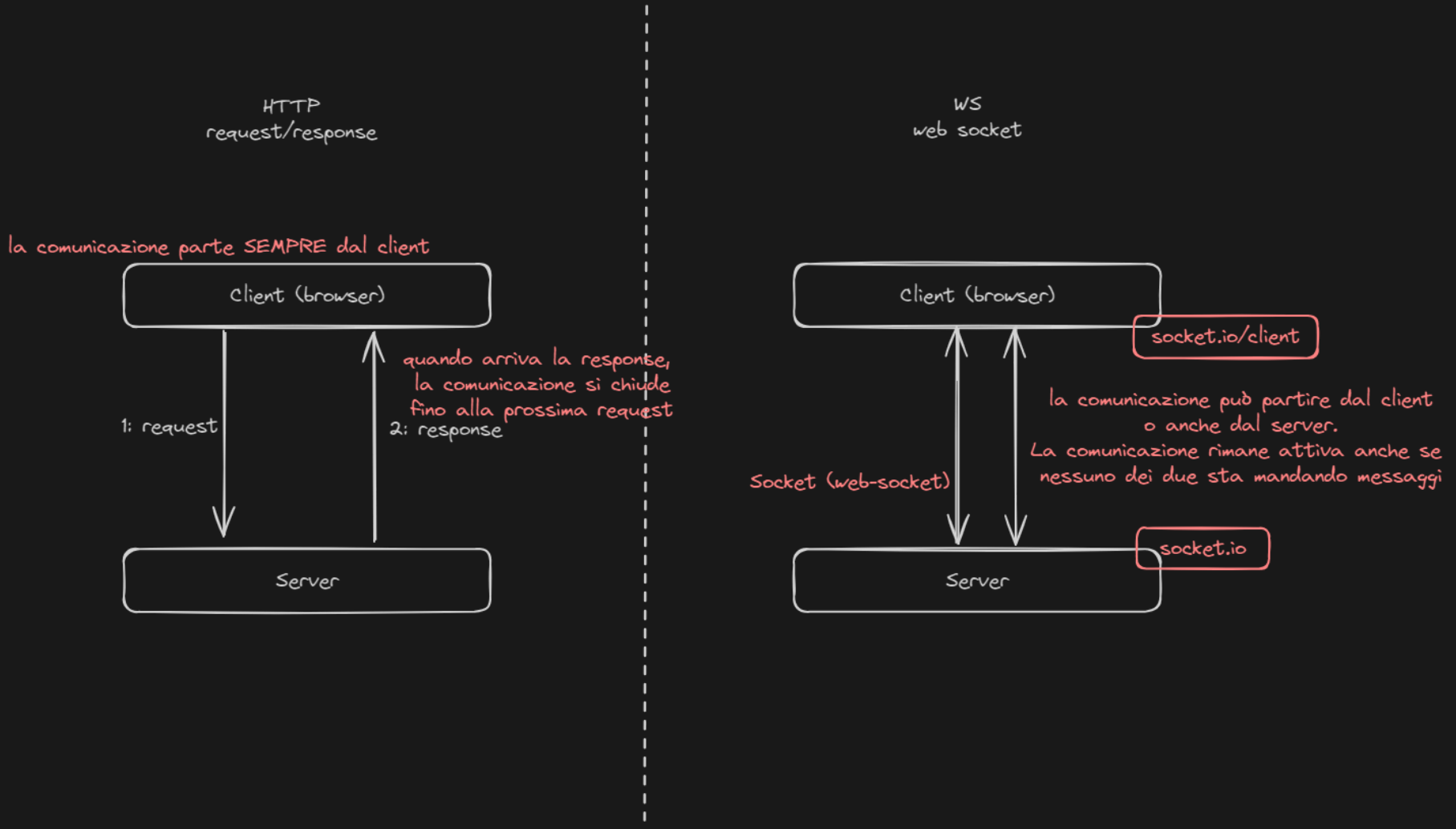
```
import Koa from "koa";
import serveStatic from "koa-static";
import cors from "@koa/cors";

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>";
});
```

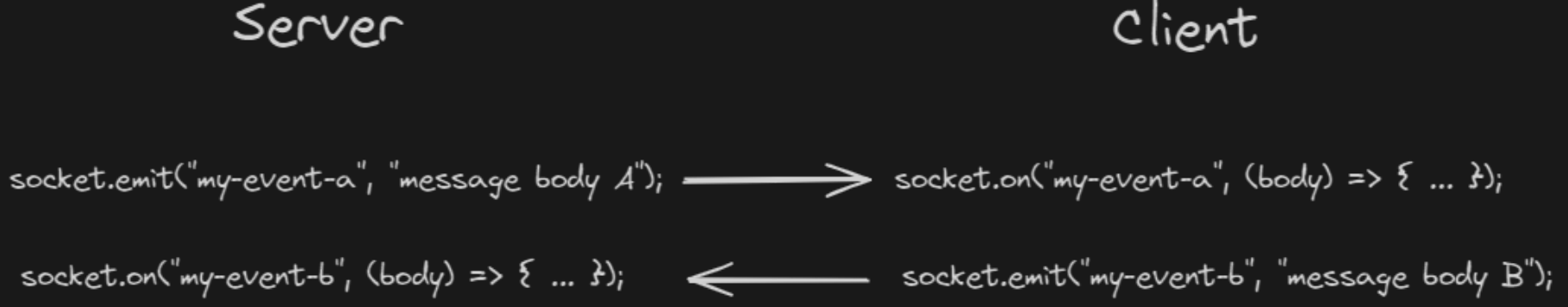
WEB SOCKET


HTTP
request/response
la comunicazione parte SEMPRE dal client
Client (browser)
1: request
2: response
quando arriva la response,
la comunicazione si chiude
fino alla prossima request
Server
WS
web socket
Client (browser)
socket.io/client
Socket (web-socket)
la comunicazione può partire dal client
o anche dal server.
La comunicazione rimane attiva anche se
nessuno dei due sta mandando messaggi
socket.io
Server

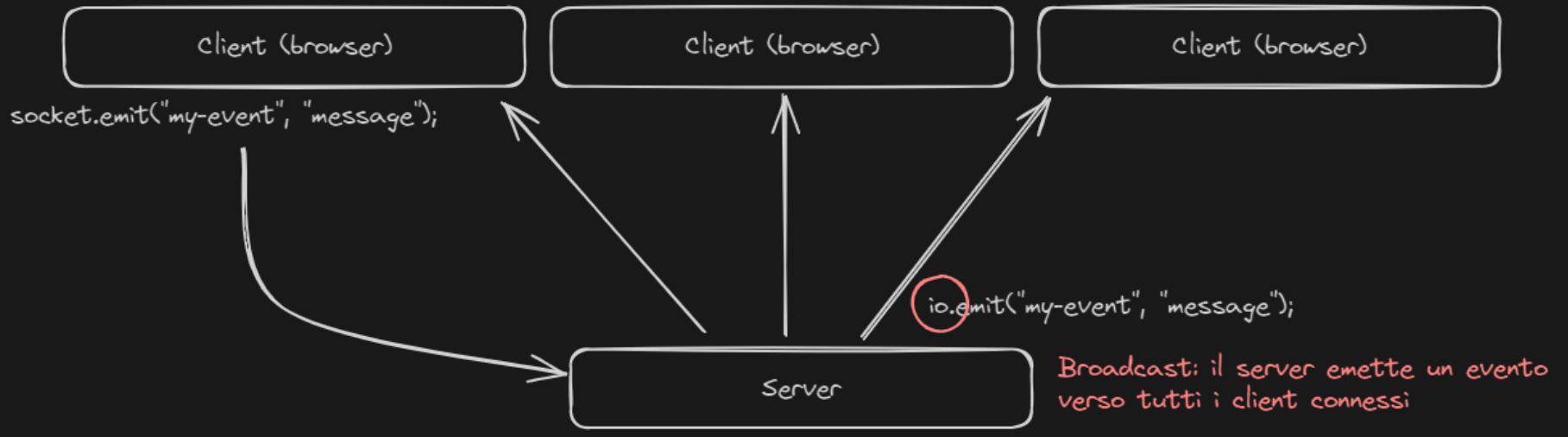
CONNESSIONE WEB SOCKET

- il client si connette al server in HTTP
- i due si scambiano un messaggio di “handshake”
- viene stabilita una connessione stabile utilizzando il protocollo `ws`
- tramite `ws`, sia il server che il cilent inviano e ricevono messaggi

SERVER E CLIENT POSSONO INVIARSI MESSAGGI



BROADCAST


client (browser)
client (browser)
client (browser)
socket.emit("my-event", "message");
io.emit("my-event", "message");
Server
Broadcast: il server emette un evento verso tutti i client connessi

LIBRERIE

Socket.io

SERVER

```
1  import { createServer } from "http";
2  import { Server } from "socket.io";
3  const httpServer = createServer();
4
5  const io = new Server(httpServer);
6  io.on("connection", (socket) => {
7    socket.emit("connected", "Hello");
8    socket.on("some-event", (data) => { /*...*/ })
9  })
10
11 httpServer.listen(3000);
```

SERVER BROADCAST

```
1  import { createServer } from "http";
2  import { Server } from "socket.io";
3  const httpServer = createServer();
4
5  const io = new Server(httpServer);
6  io.on("connection", (socket) => {
7    io.emit("broadcasted-message", "Sent to all clients");
8  })
9
10 httpServer.listen(3000);
```

CLIENT

```
const socket = io("http://localhost:3000");

socket.on("connected", (data) => { /*...*/ });

socket.emit("some-event", "Message body");

socket.on("broadcasted-message", (message) => { /*...*/ })
```