# JavaScript Node.js

## Web Servers



Url: https://github.com/obiSerra/laboratorio-nodejs-2022

# Breve riepilogo

La programmazione asincrona permette di “delegare” alcune operazione a processi esterni senza bloccare il processo principale di Node

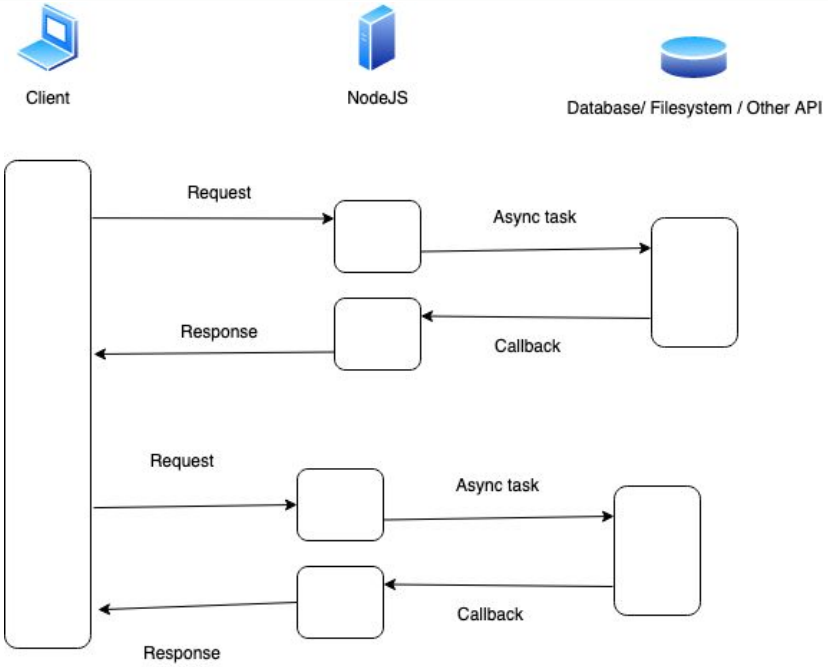
Puo’ essere fatta tramite

- Callbacks
- Eventi
- Promises
- Async/Await

# Webserver con Node

- Esistono diverse librerie e framework per creare un web server con Node, quella piu' utilizzata e' *Express (https://expressjs.com/)*
- Uno dei vantaggi principali di Express e' la sua semplicita' di utilizzo

# NodeJs as Web Server



# Middlewares

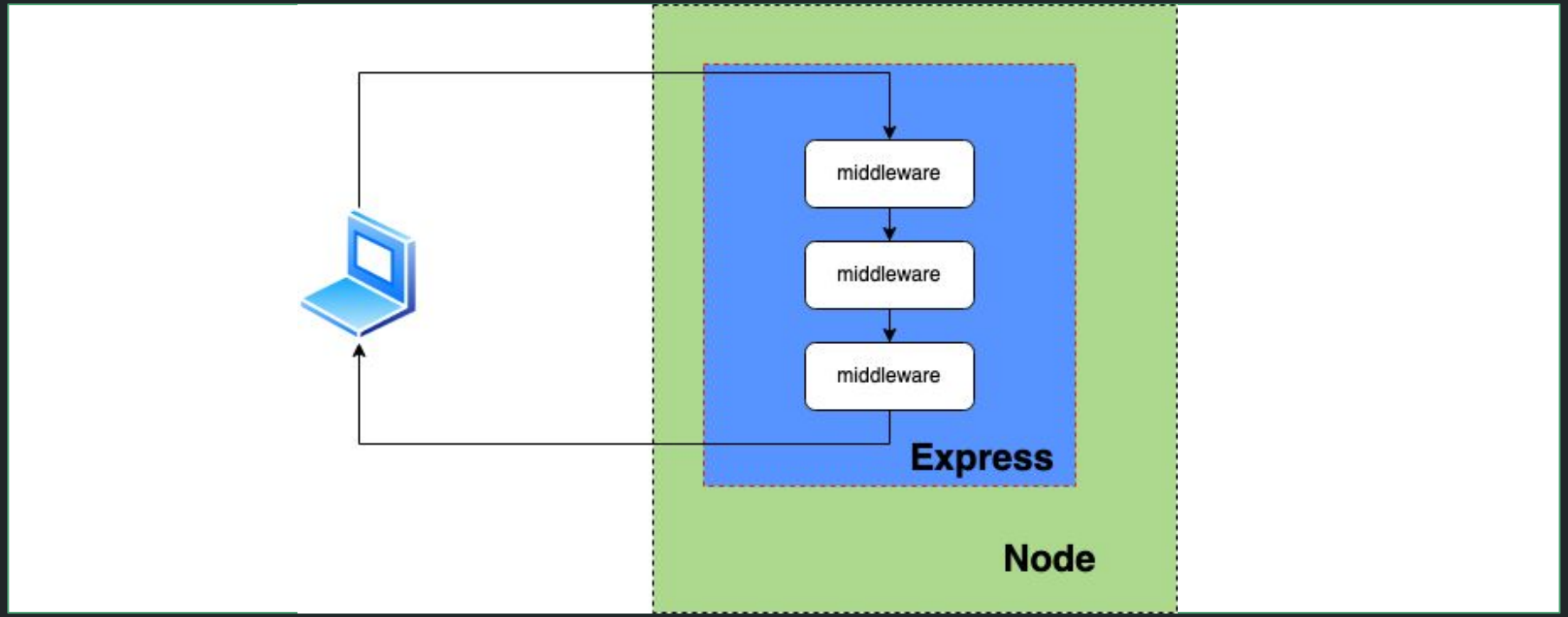
L’architettura di Express si basa su una serie di *middleware* che si “passano” la *request* fino a quando uno di questi non invia la risposta.

Un middleware e’ semplicemente una funzione che elabora dei dati (la risposta o la richiesta) e la passa al layer successivo.

Un middleware prende come parametri *req, res, next*, dove *next* e’ la funzione usata per “passare” al middleware successivo.

L’ultimo middleware restituisce la risposta al client (ad esempio al browser).

# Express



# Hello World Express

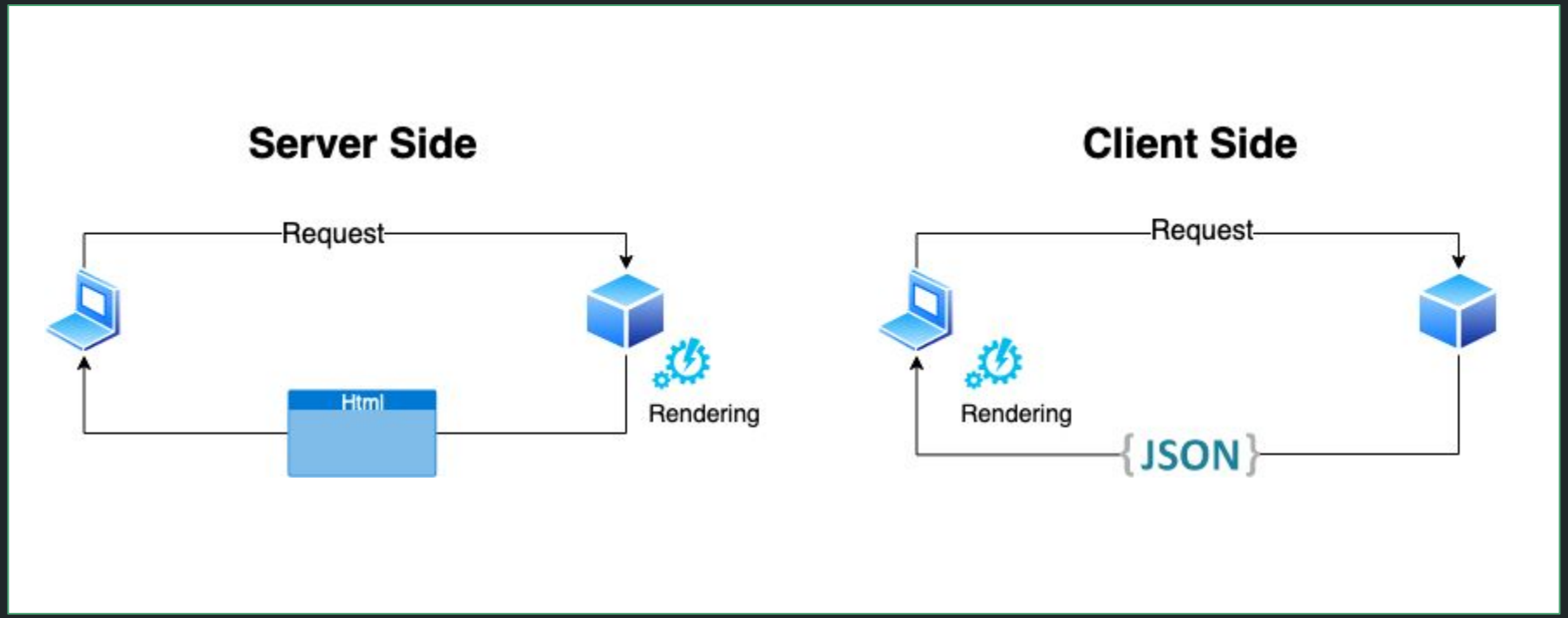
- Creiamo un semplice progetto con Express e un solo file index.js
- Tutti quelli dichiarati con *app.use*, *app.get* ... sono middleware
- *req* contiene la request, *res* contiene la risposta da inviare; *next* serve per passare la chiamata al *middleware* successivo

Una volta creato il server possiamo provare ad aggiungere un semplice middleware per loggare le richieste.

# Server Side Rendering vs API

- Un server puo’ con una pagina Html o con una struttura dati (generalmente un JSON o un XML)
- Se il server “costruisce” e restituisce una pagina, parliamo di Server Side Rendering
- Se il server restituisce solamente dei dati, parliamo di API
- Esistono diversi tipi di API: REST e’ il piu’ comune, ma ci anche SOAP e GraphQL

# Server Side vs Client Side Rendering



# Middlewares 2

Un middleware puo’ anche effettuare delle richieste asincrone.

Proviamo a creare un nuovo middleware che scarica alcuni dati da questo sito https://jsonplaceholder.typicode.com/ prima di ritornare la risposta.

# Il nostro primo (vero) web server

- Creiamo un nuovo progetto
- ePer fare un semplice server con Express bastano poche righe di JS
- Se invece vogliamo fare qualcosa di piu' complicato possiamo usare un *generator*
- Lanciamo *$ npx express-generator* *(https://expressjs.com/en/starter/generator.html)*
- Possiamo andare a vedere i file che sono stati generati
- Lanciando *$ npm install* e *$ npm start* possiamo far partire il server

# Npx

(Installation-less) Command Execution

Esegue dei comandi, usando i pacchetti installati in node_modules oppure cercandoli su npm

E’ molto usato per generator (come quello di express)

Provate a lanciare

npx cowsay "Hello"

# Routing in Express

- Routing delle risorse statiche con *app.use(express.static(...))*
- Gli endpoint vengono dichiarati con *router.get router.post*
- E’ possibile spezzare le route in sotto-route
- I path parameters vanno dichiarati nella route
- I query parameters invece non fanno parte della route e possono essere direttamente letti

# Template Engine

- Express supporta diversi template engine
- Un template engine trasforma un DSL (Domain Specific Language) in Html
- Jade e’ il template engine di default (https://jade-lang.com/)
- E’ basato su un struttura gerarchica ti template
- Handlebars e’ un altro template engine molto popolare e piu’ semplice da usare (https://handlebarsjs.com/)