Corso Node.js

Corso Node.js - Lezione 1

Node.js è un ambiente (environment) dove poter "runnare" Javascript, aldi fuori del browser

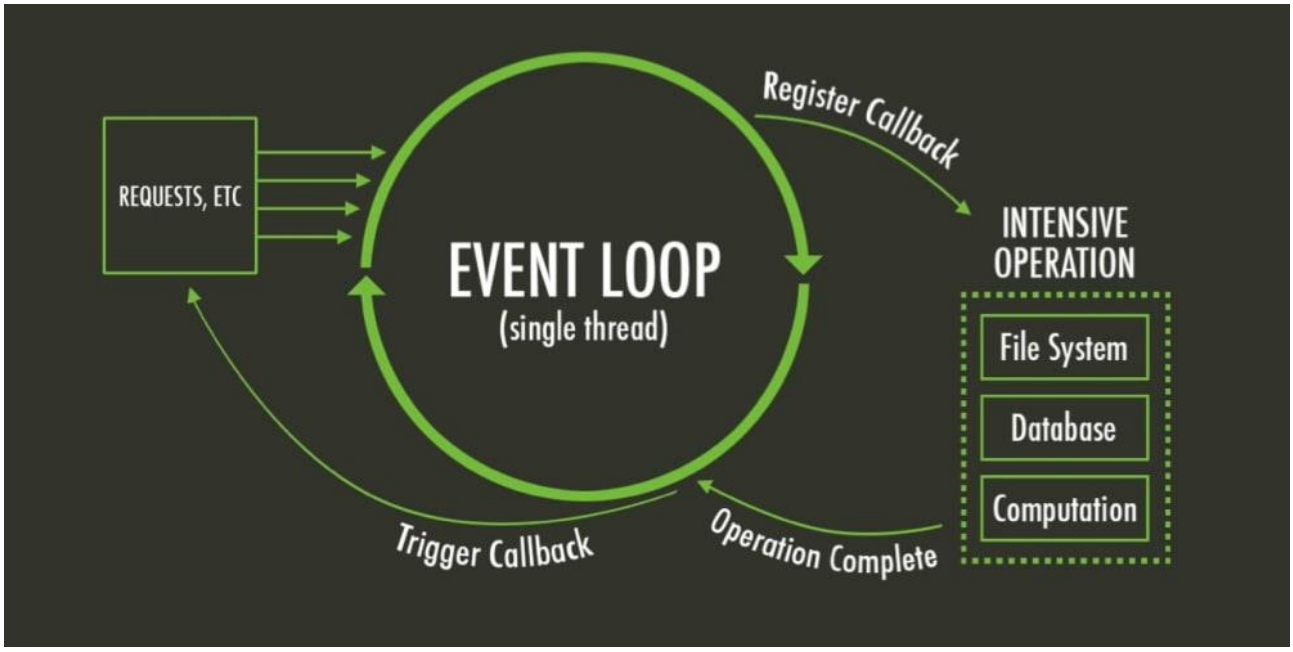
Back-End: Applicazione → C++ → V8 → JS → funzionalità

Front-End: Browser → chrome → V8 → JS → Funzionalità

Single thread: Una coda unica, tutto impilato

Multi thread: Più code, tante file paralleli

--- Node.js ha un single thread con Event Loop---



- E' scalabile

- Permette di scrivere tutto con un linguaggio, ovvero JavaScript

Parole chiave: Event Loop ("eventi"), non blocking code, asincrono, promise, callback.

Corso Node.js - Lezione 2

Installazione

Controllo installazione

Editor di testo (VSCode)

Estensioni utili VSCode: JS Code Snippet, Node Snippet, NPM.

Corso Node.js - Lezione 3

Scrivere del codice di qualsiasi tipo.

Eseguirlo su **terminale** (cmd o terminale di vsCode) con

"**node nomeFile**" (es. node index.js).

N.B. Naturalmente non esistono tutte quelle web API che si usano nel front-End, come Window, tipico oggetto del FE.

Corso Node.js - Lezione 4

Variabili Globali (https://nodejs.org/api/globals.html)

Oggetti globali molto importanti:

- **__dirname:** prende la cartella corrente
- **__filename:** nome file corrente
- **require:** funzione per importare moduli
- **module:** informazioni sui moduli correnti
- **process:** informazioni relative all'ambiente di esecuzione

```
.log("__dirname:", __dirname); // __dirname: C:\Users\Dedalo\desktop

console.log("__filename:", __filename); // __filename: C:\Users\Dedalo\desktop\index.js


console.log("require:", require);  // const module = require('modulo che ci interessa')


// console.log("module:", module);  // un mega oggetto con dentro di tutto


// console.log("process:", process);
```

// ci serve per mettere delle costanti che ci permette di capire se stiamo facendo partire node in locale o su una macchina esterna, così capiamo da dove stiamo prendendo i dati (vedi che chiami da locale e non dal server per la produzione che serve ai clienti?)

Corso Node.js - Lezione 5

Creare i moduli

I moduli è codice incapsulato

Possono essere:

- interni (creati da noi)
- Built-in (nativi)
- Esterni (scaricati da fuori)

Per esempi vedi codice in cartella Lezione 5

Corso Node.js - Lezione 6

Moduli Built- in e OS Path

- **Importare modulo OS**
- **Importare modulo Path**

Modulo OS (per lavorare con il Sistema operativo)

Modulo Path (modulo per lavorare con i percorsi)

Per importare i moduli built-in bisogna inserire un percorso

//Es. const moduloBuiltin = require('modulobuiltin');

Metodi importanti del modulo OS:

Modulo OS

```
const os = require('os');
console.log(os.userInfo()) // Utente del sitema operativo
console.log(os.uptime()) // Tempo di accensione del pc
console.log(os.version()) // Versione OS
console.log(os.arch()) // Architettura CPU
```

```
const prova = {
    nome: os.type(),
    release: os.release(),
    memoria: os.totalmem(),
    disponibile: os.freemem(),
}

console.log(prova);
```

Modulo Path

```
const path = require('path');

console.log(path.sep) // Da il separatore usato per definire i path
const percorsoFile = path.join('./cartella', 'sottocartella', 'file.txt') // Unisce
più percorsi file
console.log(percorsoFile);

console.log(path.basename(percorsoFile)) // Da solo il nome del file in fondo al
percorso

const percorsoAssoluto = path.resolve(__dirname, 'cartella', 'sottocartella',
'file.txt');
console.log(percorsoAssoluto);
```

Corso Node.js - Lezione 7

Modulo fs e Sincrono e Asincrono

const {readFileSync, writeFileSync} = require('fs');

Importiamo solo le funzioni built-in del modulo che ci interessa;

Sincrone:

readFileSync: funzione che legge file

writeFileSync: funzione che scrive file

Asincrone:

readFile: funzione che legge file

writeFile: funzione che scrive file

Esempio di chiamata asincrona

readFile(percorso, codifica, callback) // Esempio

```
readFile(percorso, codifica, function callback(error, result) {
        if(error) {
        stampa il mio errore
        } else {
            readFile(percorso, codifica, function callback(error, result) {
            if(error) {
                stampa il mio errore
                } else {
                    Fai un'altra chiamata, ecc…
                }
                });
        }
});
```

Corso Node.js - Lezione 8

Crea una base per creare un **http web server:**

Gestire la **richiesta**, gestire la **risposta** alla richiesta, gestire il **routing**, ovvero l'istradamento

Accedo alla risorsa (localhost:3000) risposta e gestione di essa

N.B. Vedi Lezione 8

Corso Node.js - Lezione 9

Gestione dei pacchetti NPM

Pacchetto di codice in javascript

Package (pacchetto) = dependency (dipendenza) = module (modulo)

Comandi utili NPM:

npm | Info su tutto npm

npm --version opp. npm -v | Controllo Versione

npm install nome_pacchetto opp. npm i nome_pacchetto | Installazione in Locale

npm install -g nome_pacchetto | Installazione in Globale

npm unistall nome_pacchetto | Disinstallazione

N.B. Per quando riguarda la creazione di scripts dentro il package.json, bisogno scrivere, per avviarli:

Npm run nome_script

Per alcuni, con parole chiave tipo "**start**", si può omettere.

Corso Node.js - Lezione 10

File Package.json

È un file **manifest** che da informazioni utili sull' applicazione (autore, dipendenze, ecc..)

npm init // crea package.json

npm init -y // Flagga tutto in automatico

npm install // Vede le dependencies del package.json e scarica tutte queste (vedi es. **node_modules**).

File package-lock.json

In **package-lock.json** c'è la versione specifica della dipendenza che era stata installata, mentre in **package.json** da dalla versione in su (^).

Versioni: A.B.C // "5.0.1"

A: Major changes, B: Minor changes, C: patch

Corso Node.js - Lezione 11

Nodemon: Abbiamo un pacchetto che sta sempre in ascolto che automaticamente aggiorna tutto,

definendo prima lo **script** dal **package.json**.

dependencies: Servono a far funzionare l'applicazione (si installano con il classico npm install nome_pacchetto).

devDependencies: Servono solo allo sviluppatore, non servono per far funzionare l'applicazione (Si installano con **–save-dev**).

Corso Node.js - Lezione 12

Event Loop

Ciclo di eventi che caratterizza Node.js

- Event Loop / single thread / async / non-blocking code
- Event queue
- Call Stack

Tutti gli eventi vanno nella **event queue** (la coda), qui l'event loop fa una chaiamta per vedere se ci sono altri eventi che devono runnare (**call stack**);

Event Loop → Evento1

(**call stack**) Evento2

Evento3

Evento4

Link alla **spiegazione** dell'**Event Loop** (https://nodejs.dev/learn/the-nodejs-event-loop)

Corso Node.js - Lezione 13

Event Emitters

Programmazione orientate agli eventi (es. In una chat)

Importare

Metodo on() //Registra un evento e manda una callback

Metodo emit() // Emette un evento

Corso Node.js - Lezione 14

Stream: Flusso costante di dati (es. nelle LiveStream)

- Usato per creare file pesanti inviato a chunk
- Leggere file senza stream e con stream

Per esempi andare a vedere la cartella **Lezione14**.

Corso Node.js - Lezione 15

client (utente, browser) → richiesta http → server(no/express)

server (node/express) → risposta http → client (browser/utente)

Metodi http

GET: leggere dati

POST: inviare dati

PUT: modificare dati

DELETE: cancellare dati

GET - sito.it/api/ordini (vedere tutti gli ordini)

- sito.it/api/ordini/:id (vedere un ordine specifico)

POST - sito.it/api/ordini (eseguire un ordine)

PUT - sito.it/api/ordini/:id (modificare un ordine)

DELETE - sito.it/api/ordini/:id (modificare un ordine)

Corso Node.js - Lezione 16

Express

App.get(path, callback(request, response))

App.all() // Tutti i metodi insieme

```
// Importiamo express
const express = require('express');

// Definiamo app, elemento base per racchiudere tutto (variabili, oggetti, metodi, ecc...) di express
const app = express();
```

Corso Node.js - Lezione 17

Chiamate Express a files static

I file sono "**statici**" perché express non ha nulla a che fare con la loro modifica, li richiama e li mostra e basta.

Questi vanno nella cartella **public** (o static);

```
Struttura della chiamata in express:
app.metodoDiChiamata('directory', callback(richiesta, risposta) => {
    risposta.inviaFileStatico('nomeFile.estensione', {oggetto dove si trova il file})
})
```

Corso Node.js - Lezione 18

API vs SSR

- **API**: Interfaccia per comunicare con il database (res, res.json)
  Invia dati in json che poi vengono infilati nel front end.
- **SSR**: Server side rendering, ovvero la pagina (template, ejs, res.render)
  Invia richiesta, la gestisce, crea un template con i dati già dentro e poi invia la pagina bella e fatta Al client.

Es. :

```
app.get('/api/hotels', (request, response) => {
        Response.sendFile('Hotels.html', {root: __dirname, '/public'})
});
```

Corso Node.js - Lezione 19

Richiamare JSON

Creare un json, esportarlo e infilarlo nella risposta tramite **response.json()**

N.B. Vedi cartella **lezione19** per ulteriori info.

Corso Node.js - Lezione 20

Route parameters

Inserire parametri nell' **url/ endpoint** così da richiedere elementi specifici

Mapping per inviare dati in modo selettivo

/

/:id

/:id/ ecc…

Request.params

N.B. Vedi cartella **lezione20** per ulteriori info.

Corso Node.js - Lezione 21

Query String Params

Parametri URL

request.query → Oggetto che contiene le query

/search?nome=Luca&cognome=Rossi

Ricorda il case sensitive nella query!

?query=A != **?query=a**

&limit=2 Definisce il limite della lista

Corso Node.js - Lezione 22

Middleware

Richiesta → Middleware (next/res) → Risposta

Es. Di middleware giustapposto ad un solo get

```
const middlewareProva = (request, response, next) => {
   const {method, url} = request;
   const time = new Date().getMinutes();
   console.log(method, url, time);
   next(); // O si aggiunge un next per passare dal midleware al response della get...
   // response.send('Ok') // ...o si risponde con una response finale
};
```

Es. Di middleware giustapposto a più get

- Aggiunge il middleware a tutti gli endpoints

```
app.use(middlewareProva);
```

- Aggiunge il midleware al ramo del percorso specificato

```
app.use('/persone', middlewareProva);
```

- Aggiunge più middleware tramite un array

```
App.use('/persone', [middleware1, middleware2]);
```

N.B. L'uso del middleware va sopra le get di riferimento

Da rivedere la parte relativa l'interruzione di un middleware e l'autorizzazione

Corso Node.js - Lezione 23

Postman (da vedere dopo)

GET: leggere dati

POST: inviare dati

PUT: modificare dati

DELETE: cancellare dati

GET - sito.it/api/ordini (vedere tutti gli ordini)

- sito.it/api/ordini/:id (vedere un ordine specifico)

POST - sito.it/api/ordini (eseguire un ordine)

PUT - sito.it/api/ordini/:id (modificare un ordine)

DELETE - sito.it/api/ordini/:id (modificare un ordine)

Corso Node.js - Lezione 24

GET: Ottenere dati

N.B. Vedi lezione su codice

Corso Node.js - Lezione 25

POST: Inviare dati

N.B. Vedi lezione su codice

Copia **temporanea** in ram, ma non è salvato in un database

Fare esercizio su post

Esempio oggetto inviato da Postman:

```
{
    "id": "5",
    "nome": "Giulio",
    "età": 100,
    "indirizzo": {
        "via": "Roma",
        "civico": 7,
        "cap": 42
    },
    "interessi": [
        "Galli",
        "Politica",
        "Conquistare la Gallia"
    ]
}
```

“raw e JSON”

Guarda il test della dashboard dentro cui infilare una persona alla lista

Corso Node.js - Lezione 26

PUT: Modificare dati

N.B. Vedi lezione su codice

Corso Node.js - Lezione 27

DELETE: Cancellare dati

Da rivedere tutto

Corso Node.js - Lezione 28

Routing

Per **route "persone":**

"/api/persone" → "/"

"/api/persone/:id" → "/:id"