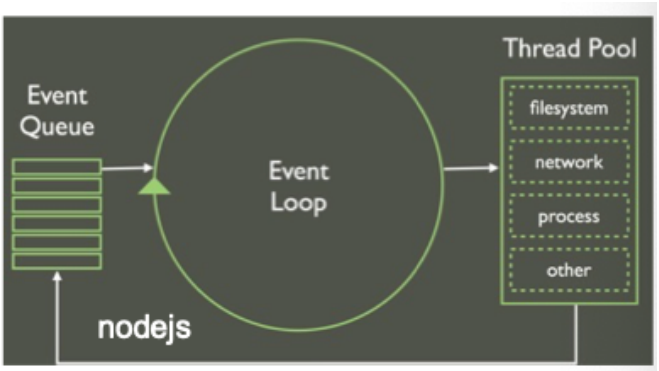
Nojo.js

Monday, 15 May 2023 08:40

- E' javascript lato server che ha bisogno di un engine -> V8
  - Efficente, open source, poche risorse, I/O non bloccanti, modulare
  - Npm = permette di configurare/installare moduli (package))
  - Estremamente asincrono (con eventi callback)



    Ed esso avviene con un loop logico, aka quando succede una richiesta viene messa in coda
    E quando arriva il turno di servire quella richiesta, viena assegnato un thread che lo esegue
    - Stateless
    - Scalabilità
  - Quindi concettualmente è 1 singolo thread basato sugli eventi
    Se però 1 evento è lungo, freeze dell'UI, ed ajax risposta potrebbe lavorare su un contesto diverso
- Librerie:
  - Possiamo implementare delle librerie con "require"
    Const fs = require('fs') -> permette lavorare con file system, include file system
    Fs.readfile('file.txt', 'utf-8', function(err, nome){});
  - Per creare un webserver "http"
    Const http = require('http')
    // Setto la risposta delle richieste
    Const server = http.createServer((req, res) => {res.statusCode=200; res.setHeader('...'); res.write(content); res.end('end')});
    // Si può fare anche res.write(data)
    // Listen
    Server.listen(port, hostnamem () => {});
    // è possibile fare http.createServer().listen()

- Per gestire gli url che riceviamo: "url"
  Const url = require('url')
  Var q = url.parse(adress, true) -> e qui fa un parser
- JSON è un oggetto predefinito che non bisogna importare
  Per prendere oggetto e trasformare in json si fa JSON.stringify

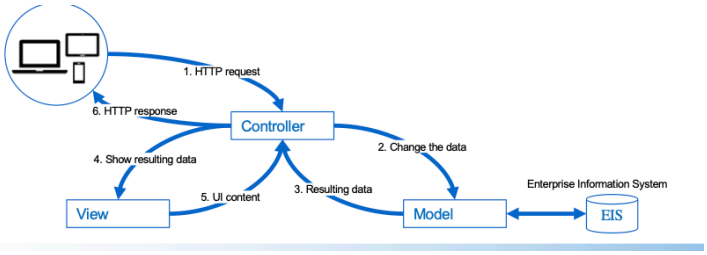
- Express.js fonde tante librerie di node.js che sono necessarie per tutte le applicazioni server
  - Npm install express
    Const express = require('express')
    Var app = express()
    // Cambiamo la root directory dentro la cartella public
    // Tutte le cartelle prima di public non saranno accessibili all'utente
    app.use(express.static('./public'));
    // Possiamo avere get/post/delete
    // Stiamo dicendo che abbiamo una variabile chiamata d
    App.get('path/:d', fun(req, res){});
    Var serve = app.listen(3000, 'localhost', function() {});
- Costruire rest
  Dobbiamo implementare seguente tabella:

URI	Metodo	Operazioni	Return code
/students	Get	Restituisce elenco studenti	200 ok
	Post	Aggiunge studente	201 created
	Put	-	-
	Delete	Cancella tutti gli studenti	200 OK

Ed ogni get/post/ecc lo dobbiamo implementare con app.get/post/put/ecc
Per potere creare queste rest api si può utilizzare express

- Associare dei percorsi ai metodi HTTP
- Associare metodi HTTP delle funzioni

Segue la struttura MCV (piccolo ricordo di MCV)



E la nostra folder è strutturata:

- Bin, sono la configurazione principale della nostra applicazione
- Public, ciò che le persone possono vedere
- Routes, contengono metodi che servono per creare delle navigazioni tra le pagine

tra le pagine
- Views, lo scheletro che l'utente vede
- App.js, il file che viene chiamato
- Package.json, contiene le librerie