JavaScript Node.js

Tutto quello di cui non abbiamo parlato

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

Riassunto della lezione precedente

- Express.js
- REST API
- Autenticazione con JWT

Database

...vedi slide lezione 3...

MYSQL

- Possiamo installare MYSQL se non lo abbiamo gia' installato
  - Windows https://dev.mysql.com/doc/refman/8.0/en/osx-installation.html
  - macOS https://dev.mysql.com/doc/refman/8.0/en/osx-installation.html
- Per gestire il db possiamo installare MySQL Workbench (https://dev.mysql.com/downloads/workbench/)
- Lato Node dobbiamo installare come pacchetto npm mysql2 (https://www.npmjs.com/package/mysql2) e NON mysql (https://www.npmjs.com/package/mysql) a causa di una incompatibilita' con il "nuovo" metodo autenticazione di MySQL

ref (https://stackoverflow.com/questions/50093144/mysql-8-0-client-does-not-support-authentication-protocol-requested-by-server/56509065#56509065)

Usare MySQL

- Usiamo il file .env per salvare i dati di connessione
- Possiamo creare una helper class per gestire le connessioni con il database
- *.query(...)* per effettuare le query
- *.execute(...)* per utilizzare parametri nella query (consigliato)

- Adesso possiamo provare ad aggiungere un endpoint per cercare i film per titolo

Applicazioni desktop

- Node puo’ essere usato anche per sviluppare applicazioni desktop con Electron (https://www.electronjs.org/)
- Electron e’ basato su Node.js e Chromium ma permette di sviluppare applicazioni con “look & feel” nativo
- Possiamo creare un semplice frontend HTML + CSS + JS oppure usare un framework come React

Unit Test

- Uno unit test e’ un test che verifica la funzionalita’ di una singola unita’ di un software, normalmente una funzione o un metodo
- Aggiungere unit test a un progetto e’ molto importante per diversi motivi tra cui:
  - Assicurano che il codice sia piu’ facile da “rifattorizzare” senza rompersi
  - Permettono di evitare regression
  - Aiutano a mantenere il codice “pulito”
- Un’ottima tecnica per scrivere codice codice facilmente testabile e’ il Test Driven Development, in cui si scrivono prima i test e poi il codice

Unit test in Node.js

- Useremo Mocha (https://mochajs.org/) come framework per i nostri test
- Installiamolo con *$ npm install --save-dev mocha*
- Mocha e' il framework che ci permette di eseguire i test, pero' abbiamo bisogno di altri due elementi:
  - Un'assertion library (https://www.chaijs.com/)
  - Un sistema per creare Mock, Spies e Stubs (https://sinonjs.org/)

Unit Test in pratica

- Aggiungiamo qualche unit test alle applicazioni che abbiamo scritto durante il laboratorio

Progetti piu’ Strutturati

- Typescript e’ ottimo per progetti piu’ strutturati
- Creiamo un progetto express in typescript con *$ npx express-generator-typescript*
- Alcune delle funzionalita’ usato da questo progetto:
  - _modulesAliases
  - Interfaccie
  - Dao
  - type modules

NodeJS in produzione

- A differenza dello stack LAMP, NodeJS normalmente non gira come “service”
- Per un sito reale dobbiamo usare un process manager per gestire eventuali errori o fallimenti
  - pm2 (https://github.com/Unitech/pm2)
  - forever (https://github.com/foreversd/forever)
- *$ pm2 start <boot-file>* per lanciare pm2
- altri comandi utili:
  - $ pm2 list
  - $ pm2 restart <id> | all
  - $ pm2 stop <id> | all
  - $pm2 monit

Ancora un paio di cose

- helmet (https://helmetjs.github.io/) - middleware per header di sicurezza per Express
- socket.io (https://socket.io)