ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Framework Javascript

Seconda parte

Fabio Vitali

Corsi di laurea in Informatica
Alma Mater – Università di Bologna

Qui parleremo di…

Alcuni framework:

- *JQuery*
- Node + Express
- moduli in EcmaScript
- Mongo + Mongoose

- *Il problema dei template*
- *Moustache e Handlebar*
- *Angular*
- *React*
- *Vue*

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Node.js, npm ed Express.js

[XXX] across the stack

- Lo stack è l'insieme degli ambienti di cui un progettista web deve tener conto per realizzare un'applicazione, e dei linguaggi di programmazione da conoscere.
- Java across the stack:
  - *Google Web Toolkit*: ambiente integrato client-server, in cui la parte client viene compilata in Javascript da un sorgente Java
- Javascript across the stack:
  - *ASP* e poi *ASP.NET*: soluzione Microsoft per IIS. ASP era solo Javascript, mentre ASP.NET permette di scegliere tra molti linguaggi, incluso C#, VBA e Javascript.
  - *Google Apps Script*: soluzione Google per fornire comportamenti sofisticati ai siti realizzati con Google Sites.
  - *MongoDB, CouchDB*: database NoSQL forniscono meccanismi di interrogazione e gestione dei dati via Javascript
  - NodeJS: ambiente di esecuzione server-side di applicazioni sofisticate (anche indipendentemente da HTTP).

MEAN stack

- Un termine che adesso è MOLTO di moda è MEAN stack:
  - MongoDB: un database NoSQL che fornisce permanenza ai dati delle applicazioni
  - ExpressJs: un framework per applicazioni web
  - AngularJs: un framework MVC per la creazione di template sofisticati di pagine HTML
  - NodeJs: una piattaforma software per applicazioni server-side
- Lo scopo evidente dello stack MEAN è di ripetere il successo e sostituirsi allo stack LAMP che ha costituito il set fondamentale di tecnologie per lo sviluppo di applicazioni web negli anni 2000.

NodeJS e npm

Creatura di una persona sola, Ryan Dahl (2009), per fornire bi-direzionalità alle applicazioni web (tecnologia *push*).

Un'applicazione nativa server-side, che permette di fare chiamate I/O guidate da eventi, non-bloccanti.

Per uniformare la realizzazione di applicazioni miste server/client, decide di usare un motore Javascript anche server-side

- Non è la prima volta, già Microsoft con ASP usa Javascript server-side
- Utilizza un interprete di Javascript molto potente, V8 di Google

Ad esso viene aggiunto un package manager, ***npm***, che permette di installare velocemente librerie e pacchetti associati.

Single-thread non-blocking execution

Invece di dedicare un thread ad ogni connessione ricevuta dalla rete, l'applicazione nodeJS utilizza un solo thread in cui tutte le richieste ricevono attenzione condivisa.

- In un'applicazione Web tradizionale un thread può avere circa 2Mb di memoria associati, quindi una macchina con 8GB ha un massimo teorico di 4000 richieste concorrenti.
- Mantenendo tutto in un unico stack, NodeJS arriva ad un massimo teorico di 1 milione di richieste concorrenti.

Poiché ogni richiesta può richiedere tempi variabili, è prassi inserire richieste onerose in funzioni asincrone, non-bloccanti, con una funzione di callback al completamento.

- Questo permette al thread principale (main loop) di rimanere molto snello e veloce.
- Per contro, è facilissimo inchiodare NodeJS facendo eseguire parti onerose del codice direttamente dentro al main loop.

NodeJS come motore HTTP

NodeJS può elegantemente sostituire PHP, Python, Perl e ogni altro approccio cgi-bin tradizionale. Di seguito un esempio di un'applicazione che restituisce un frammento HTML:

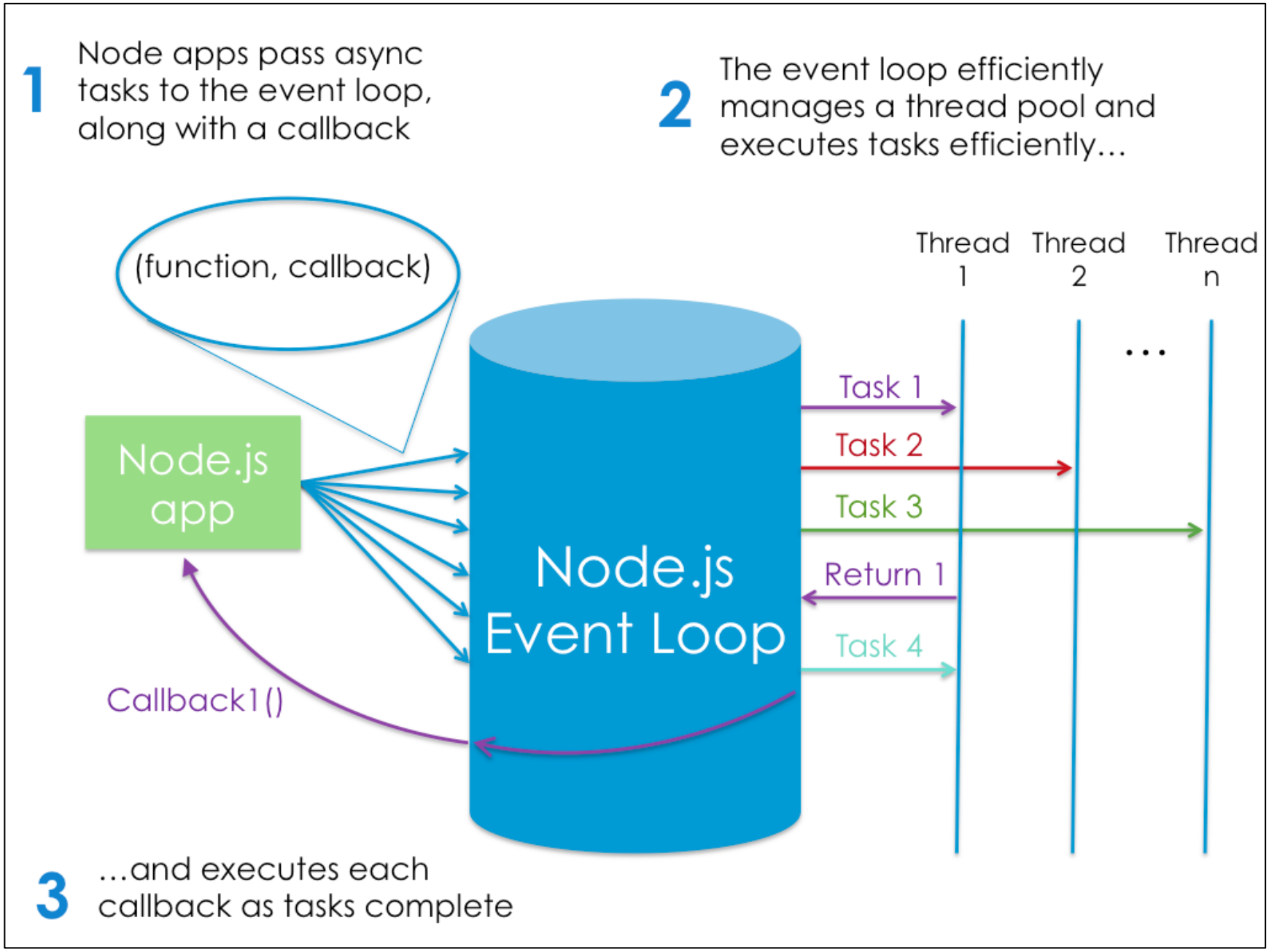
```
var http = require('http');

http.createServer(
   function (request, response) {
      response.writeHead(200, {'Content-Type': 'text/html'});
      response.end('<h1>Hello World</h1>');
   }
).listen(8000);
```

NodeJS con altri protocolli

NodeJS può porsi in ascolto su qualunque porta e implementare anche protocolli diversi da HTTP:

```
var net = require('net');
net.createServer(
  function (stream) {
    stream.write('hello\r\n');
    stream.on('end',
      function () {
        stream.end('goodbye\r\n');
      }
    );
    stream.pipe(stream);
  }
).listen(8000);
```

Event Loop



http://i.stack.imgur.com/zYgcr.png

Asincronia e funzioni callback

- Node.js è quasi esclusivamente basato su funzioni asincrone e callback.
- La convenzione di node.js suggerisce di creare funzioni che accettano una funzione callback asincrona come ultimo parametro
- La funzione callback per convenzione usa la sintassi error-first

```
fs.readFile('test.txt', function(err, data) {
  if(err) {
    console.log('Error: '+error.message);
    return;
  }
  doSomething(data);
});
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

I moduli in Javascript

Modularizzazione del codice

- Separazione del codice di un programma complesso in frammenti indipendenti e intercambiabili.
- Richiedono corretta decomposizione del problema, un'interfaccia alle funzioni chiara e precisa, e un buon incapsulamento delle funzionalità e delle strutture locali al modulo.
- Per lungo tempo Javascript non ha avuto un meccanismo di modularizzazione. Nessuna struttura sopra all'oggetto, nessuna possibilità di avere strutture private (fondamentali per l'incapsulamento).
- Poi, in rapida successione, arrivano prototipi, closure, IIFE, classi, moduli CommonJS e moduli EcmaScript.
- Inizialmente solo server-side, ora anche client-side

Aspetti comuni

- Separation of concern, incampsulamento, intercambiabilità del codice, load on demand.

- Ogni modulo è pensato in un file separato
- Le variabili possono essere:
  - Variabili locali alla funzione (come al solito) e
  - Variabili locali al modulo, globali per il modulo ma invisibili al di fuori del modulo
  - Variabili esportate dal modulo, visibili per chi include il modulo.

- Esistono modi per gestire la namespace pollution quando i moduli inclusi sono tanti

CommonJS e AMD

- Prima proposta di modularizzazione di Javascript (2009), adottata da NodeJS e da altri. NON FUNZIONA SUI BROWSER!
- Esiste una versione parzialmente compatibile, chiamata AMD (asynchronous module definition).
- Ogni modulo usa due costrutti speciali: `module.exports` e `require:`

```
// Saved as file "keys.js"
var keys = {
  8: 'backspace',
  9: 'tab',
  13: 'enter',
  16: 'shift',
  17: 'ctrl',
  18: 'alt'
};
var Utils = {
  name: function(keyCode) {
    return keys[keyCode];
  },
  keyCode: function(name) {
    return keys.indexOf(name);
  }
}
module.exports = Utils;
```

```
var keys = require('./keys.js');

document.addEventListener('keyup',
  function(e) {
    var key = keys.name(e.keyCode);
    console.log("You pressed "+key);
  }
);
```

ES module (1)

- La risposta a CommonJs da parte di WHATWG. A partire dal 2016, qualchw anno dopo anche sui browser.
- Ogni modulo usa due istruzioni speciali: `export` e `import:`

```
// Saved as file "keys.js"
var keys = {
  8: 'backspace',
  9: 'tab',
  13: 'enter',
  16: 'shift',
  17: 'ctrl',
  18: 'alt'
};

export name: function(keyCode) {
    return keys[keyCode];
}
export keyCode: function(name) {
    return keys.indexOf(name);
}
```

```
import {name, keyCode} from ('./keys.js');

document.addEventListener('keyup',
  function(e) {
    var key = name(e.keyCode);
    console.log("You pressed "+key);
  }
);
```

oppure

```
import * as keys from ('./keys.js');

document.addEventListener('keyup',
  function(e) {
    var key = keys.name(e.keyCode);
    console.log("You pressed "+key);
  }
);
```

ES module (2)

- Se voglio esportare un solo nome (un oggetto, un variabile, una funzione, etc.) posso usare default export

```
// Saved as file "keys.js"
var keys = {
  8: 'backspace',
  9: 'tab',
  13: 'enter',
  16: 'shift',
  17: 'ctrl',
  18: 'alt'
};

name = function(keyCode) {
    return keys[keyCode];
}

export default name;
```

```
import newName from ('./keys.js');

document.addEventListener('keyup',
  function(e) {
    var key = newName(e.keyCode);
    console.log("You pressed "+key);
  }
);
```

ES module (3)

- Se voglio importare un modulo sul browser debbo esplicitamente indicare al browser che non sto caricando lo script totalmente (otterrei un errore), ma solo i nomi esplicitamente esportati dal modulo

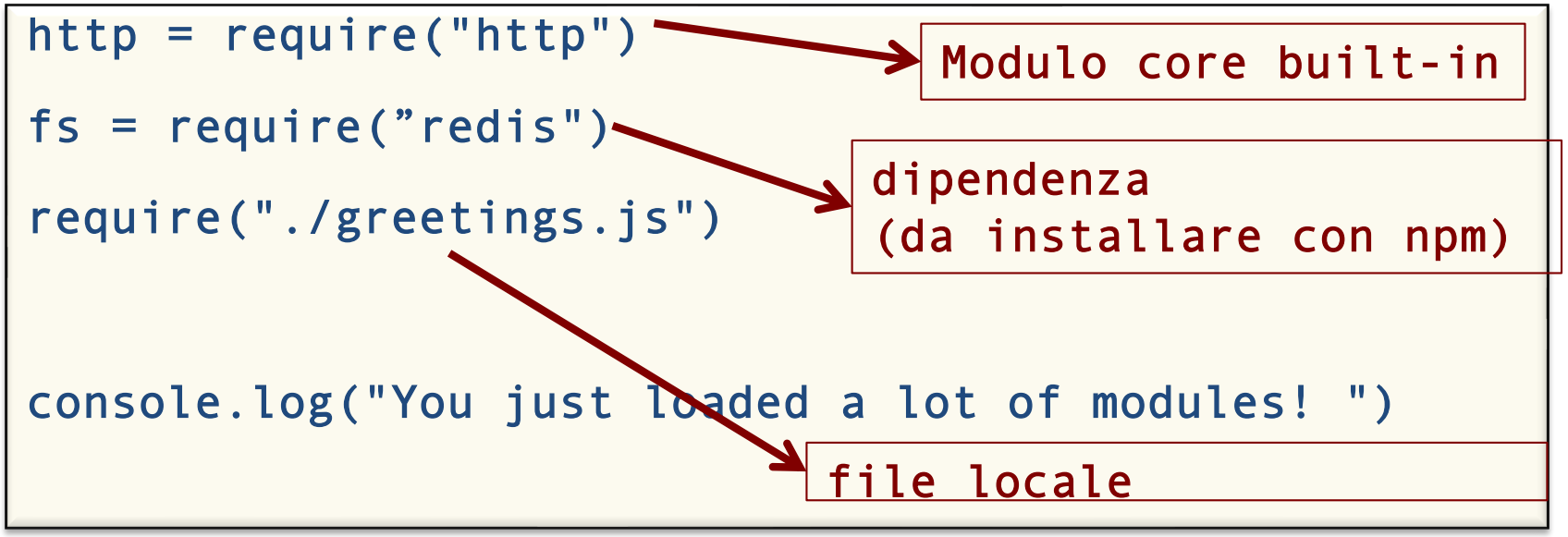
```
// Saved as file "keys.js"
var keys = {
  8: 'backspace',
  9: 'tab',
  13: 'enter',
  16: 'shift',
  17: 'ctrl',
  18: 'alt'
};

name = function(keyCode) {
    return keys[keyCode];
}

export default name;
```

Nell'HTML:

```
<html>
  <head>
    <script type="module" src="keys.js"></script>
    <script>
      document.addEventListener('keyup',
        function(e) {
          var key = name(e.keyCode);
          console.log("You pressed "+key);
        }
      );
    <script>
  </head>
  ...
</html>
```

I moduli di node.js

- Node.js è estremamente modulare. E' possibile mettere codice indipendente in file javascript ed eseguirlo secondo necessità.
- Il meccanismo di caricamento di moduli di Node.js è semplice:
  - un modulo è un file Javascript.
  - Quando si richiede un modulo, esso viene cercato localmente o globalmente secondo un modello preciso.
- I moduli vengono caricati sia con `require()` sia con `import`.



Creare un modulo

```
greetings = require("./greetings.js")
greetings.hello()
greetings.ciao()
```

greetings.js

```
hello = function() {
        console.log("\n Hello! \n")
}
ciao = function() {
        console.log("\n Ciao! \n")
}
module.exports.hello = hello;
module.exports.ciao = ciao;
```

Creare un modulo

```
import greetings from "./greetings.js")
greetings.hello()
greetings.ciao()
```

greetings.js

```
hello = function() {
  return("\n Hello! \n")
}
ciao = function() {
  return("\n Ciao! \n")
}
greetings = {hello, ciao}
export default greetings;
```

npm

- I moduli di node.js vengono distribuiti ed installati con npm (*node package manager*)
- npm viene eseguito via command-line e interagisce con il registro npm.
  - meccanismo robusto per gestire dipendenze e versioni dei pacchetti (moduli)
  - semplice processo di pubblicazione di pacchetti e condividerli con altri utenti.
- In più: una piattaforma per repository pubbblici e privati, servizi enterprise (integrati con firewall aziendali), integrazione con altre piattaforme, ecc.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Express.js

Express.js

- Express è un modello di server web per node.js
  - Open-source, licenza MIT
  - molto usato e molto supportato dalla comunità
  - molto semplice e molto espandibile con plugin
- Express si dedica al routing, alla gestione delle sessioni, all'autenticazione degli utenti, e molti altri principi fondanti delle connessioni HTTP.
- Nato nel 2010, acquistato nel 2015 da IBM, poi regalato da IBM alla Node.js Foundation.

Basic server

```
express = require("express")
app = express()

docs_handler = function(request, response){
        var docs = {server : "express”}
        response.json(docs);
}

app.get("/docs", docs_handler);

app.listen(8099, function(){
     console.log("\nExpress is running!!! \n")
})
```

Routing

Express fornisce una semplice interfaccia per fare routing:

```
app.method(path, function(request, response) { ... })
```

dove

- method è uno dei metodi HTTP (get, post, put, delete, ecc.)
- path è il local path dell'URI richiesto al server
- function(req, res) è un handler da eseguire quando viene richiesto il path. I due parametri contengono gli object della richiesta HTTP (uri, intestazioni, parametri, dati, ecc.) e della risposta HTTP in via di restituzione (intestazioni, dati, ecc.)

Ogni route gestisce una o più handler anche sugli stessi path.

```
app.post('/', function (req, res) {
  res.send('<p>Richiesta POST');
});
```

Routing parametrico

E' possibile modularizzare il routing soprattutto per siti molto complessi o pieni di URI diversi. Tre casi:

- ***URI regex.*** Questa route accetterà URI come abcd, abbcd, abbbcd, ecc.

```
app.get('/ab?cd', (req, res) => {
  res.send('ab?cd')
})
```

- **URI parametrici**: Questa route accetta URI di API REST tipiche:

```
app.get('/users/:userId/sales/:saleId', (req, res) => {
  res.send(req.params)
})
```

Routing parametrico

Handlers di secondo livello.
Usando un oggetto Router posso delegare parte della responsabilità dei route a moduli indipendenti.

```
// save as informatica.js
const express = require('express')
const router = express.Router()

router.get('/', (req, res) => {
  res.send('Home page dipartimento Informatica')
})
router.get('/about', (req, res) => {
  res.send('A proposito del dipartimento')
})
module.exports = router
```

questo modulo risponde a URI del tipo /cs/ e /cs/about

```
const info = require('./informatica.js')

// ...
app.use('/cs/', info)})
```

Accedere ai dati di un POST

- La libreria bodyparser (integrata con express) permette di accedere al corpo dei dati spediti dal POST nel body della richiesta.
- Questi middleware si occupano di recuperare e elaborare i dati sottomessi da un form via post e li convertono in oggetti accedibili dall'applicazione.
- I dati del POST si accedono come:

```
<request>.body.<post_variable>
```

POST data

```
var express = require("express")
var bodyParser = require('body-parser');

app = express()

app.use(bodyParser());

app.post('/login',function(request,response){
  console.log("Username:" + request.body.user)
  console.log("Password:" + request.body.pwd)

 // Continue and do authentication….
});
```

‘user’ / ‘pwd’:
nomi degli input usati nel form

Express middleware stack

- Express ha pochissime funzionalità proprie (routing), ma utilizza un grande numero di librerie middleware grandemente personalizzabili in uno stack di servizi progressivamente più complessi.
- Per aggiungere un middleware allo stack:

  `app.use(<middleware>)`
- Un'applicazione Express è allora essenzialmente una sequenza di chiamate funzioni di middleware tra la richiesta e la risposta.
- Il middleware può
  - accedere agli oggetti di richiesta e risposta
  - cambiarli ed eseguire del codice di modifica
  - chiamare la prossima funzione del middleware (`next()`)
  - uscire dal ciclo e mandare la risposta.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Il sito di default su Gocker

Usare node sulle macchine del DISI

- Il progetto di TW richiede l'uso di node come motore dell'applicazione server-side.
- Inoltre le regole del corso impongono l'uso di una macchina DISI per ogni servizio server-side del progetto.
  - Ogni struttura dati e ogni servizio sw creato da studenti UniBo deve essere fornito da una macchina *.cs.unibo.it
- Il DISI mette a disposizione DUE modi per attivare un servizio node:
  - attraverso *gocker*, un servizio docker di virtualizzazione di una macchina online (*container*)
  - Via linea di comando su una porta alta (>1024)
- Entrambi richiedono qualche competenza di linea di comando (poca roba)
- Entrambi funzionano sullo spazio dati e con i permessi dello studente, e non dell'amministratore o di root.

Node da linea di comando

```
var http = require('http');

http.createServer(
   function (request, response) { ... }
).listen(8000);
```

- Il modo più semplice per attivare node è da linea di comando:
  - `fabio@eva:~$ cd /home/web/siteXXXX/html/`
  - `fabio@eva:/home/web/siteXXXX/html/$ node index.js`
- A questo punto node risponde all'indirizzo:
  - `http://eva.cs.unibo.it:8000/`

PATH e shell

- Può succedere che otteniate errori tipo

```
fabio.vitali@eva:~$ node index.js
-bash: node: command not found
fabio.vitali@eva:~$ npm install express
-bash: npm: command not found
```

- Questo significa che non avete la directory di node e npm nel PATH (che sono le directory cercate automaticamente dalla shell).
  - E' necessario o specificare il path assoluto di node e npm...
    - `/usr/local/node/bin/node index.js`
    - `/usr/local/node/bin/npm install express`
  - oppure inserire la directory di node nel PATH:
    - `cd $home`
    - `nano .bash_profile`
    - Aggiungete la seguente riga: export PATH=/usr/local/node/bin/:$PATH
    - Riavviate la shell, lanciando il comando: exit

Limiti e problemi di node da linea di comando

- Funziona solo finché la shell è aperta.
  - Quindi non possiamo scollegarci finché vogliamo che funzioni
  - Non è vero, c'è un modo, ma non è il momento per parlarne.
- Funziona solo dalla macchina da cui abbiamo lanciato l'eseguibile.
  - Quindi l'URI deve specificare esattamente la macchina scelta.
- Funziona solo su porte alte > 1024, e la porta deve essere libera per quella macchina.
  - quindi l'URL da usare DEVE specificare la porta
  - Suggerimento: poiché Gocker funziona necessariamente sulla porta 8000, usate la porta 8000.
- Ci può essere solo UN servizio attivo su una data porta e un dato server
  - Quindi se usate tutti la porta 8000 dovete tutti usare server diversi.
- Utile nelle fasi di programmazione e debug.
- Una volta che il server funzioni dovrete cercare qualcosa di più stabile.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Moduli NPM preinstallati

npm

- I moduli di node vengono distribuiti ed installati con npm (*node package manager*)
- npm viene eseguito via command-line e interagisce con il registro npm.
  - meccanismo robusto per gestire dipendenze e versioni dei pacchetti (moduli)
  - semplice processo di pubblicazione di pacchetti e condividerli con altri utenti.
- In più: una piattaforma per repository pubbblici e privati, servizi enterprise (integrati con firewall aziendali), integrazione con altre piattaforme, ecc.

Usare npm

Un pacchetto npm è semplicemente un insieme di file, il più importante dei quali è *package.json*, che contiene metadati sul pacchetto (contenuto, dipendenze, ecc.)

Comandi utili di npm:

- npm init
  - Genera un file package.json globale nella directory in cui si trova. Tipicamente è la directory in cui eseguire node.
  - Viene anche creata la directory node_modules in cui verranno posizionati i package installati.
- npm install [package]
  - installa il pacchetto specificato, comprese tutte le dipendenze specificate, nella directory node_modules
- npm uninstall [package]
  - rimuove il pacchetto specificato, comprese tutte le dipendenze specificate e non usate da altri pacchetti installati
- npm update [package]
  - aggiorna esplicitamente il pacchetto specificato, comprese tutte le dipendenze specificate.

Express.js

- Express è un modello di server web per node.js
  - Open-source, licenza MIT
  - molto usato e molto supportato dalla comunità
  - molto semplice e molto espandibile con plugin
- Express si dedica al routing, alla gestione delle sessioni, all'autenticazione degli utenti, e molti altri principi fondanti delle connessioni HTTP.
- Nato nel 2010, acquistato nel 2015 da IBM, poi regalato da IBM alla Node.js Foundation.

Express/cors

Permette di definire server Express che realizzano completamente le funzionalità CORS (*Cross Origin Resource Sharing*).

- Per ragioni di sicurezza, un documento HTML può ricevere risorse (immagini, dati Ajax, script, ecc.) solo da un server posto nello stesso dominio del documento HTML di origine (*Same Origin Policy*).
- E' a discrezione del server permettere ad un'applicazione di un dominio diverso di richiedere le proprie risorse (*Cross Origin Resource Sharing*).
- Il browser, subito prima di eseguire un collegamento Ajax ad una risorsa di un dominio diverso, esegue un OPTION al server.
- Affinché sia poi possibile eseguire la richiesta, il server deve rispondere con alcune intestazioni HTTP specifiche, altrimenti il browser non effettuerà la richiesta vera e propria.
- Tipicamente: `Access-Control-Allow-Origin: *`

Il package express/cors inserisce automaticamente la risposta cors più appropriata e gestisce la creazione di servizi server-side aperti a tutti i richiedenti.

Handlebar.js

- Handlebar.js è una semplice estensione di Mustache.js
- Un linguaggio di template logicless per tenere viste e codice separati
- Permette di accedere agli oggetti annidati dentro alle variabili da interpolare e di ottenere l'effetto di cicli e istruzioni condizionali usando bocchi contestuali.
- https://handlebarsjs.com/

Passport.js (non pre-installato)

- L'autenticazione può venire realizzata da package aggiuntivo di express.
  - Ce ne sono dozzine
  - Alcune autenticazioni base sono incluse direttamente in Express
  - Ad esempio *HTTP basic digest authentication*
- Passport.js è uno dei package più adottati
  - Flessibile e modulare
  - Supporta numerosissimi modelli di autenticazione: HTTP digest, Facebook, Google, Twitter, LinkedIn, ecc., chiamati *strategie*.
  - Permette anche di aggiungere nuove strategie ad-hoc.

nodemon

- Una semplicissima, utilissima utility per node.
- Node.js fa prefetch e cache degli script da eseguire. Quindi ogni modifica sugli script non viene percepita ed usata da node.js fino al riavvio, che va fatto a mano ogni volta.
- In fase di debugging, questo significa che node va riavviato a mano ogni pochi minuti, che può essere sgradevole e pesante.
- Nodemon tiene sotto controllo tutti i file di una directory, e ogni volta che percepisce il salvataggio di una modifica riavvia node.js automaticamente.
- Attenzione!
  - Voi scrivete il file su una delle macchine del laboratorio;
  - Il file system di gocker viene avvisato della modifica dopo 2-3 secondi;
  - Nodemon si accorge della modifica dopo 1 secondo e riavvia node.js;
  - Node.js richiede 3-4 secondi per riavviarsi.
- Nel frattempo il vostro sito darà errore 500 Internal Server Error
- Ci vogliono mai meno di 5 secondi, e a volte anche 10 secondi per rivedere il vostro sito online!!!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Mongo e Mongoose

MongoDB

- In generale, un'applicazione web, anche data-oriented, non ha bisogno di un DB né SQL né di altro tipo, poiché tutti i linguaggi server-side (da Perl a PHP a node.js) forniscono metodi per utilizzare nativamente il file system e la applicazione può scrivere i suoi dati persistenti su un file qualunque.
- Si usa un DB:
  - quando il file diventa troppo grande per stare tutto in memoria,
  - quando ho bisogno di metodi veloci per cercare un sottoinsieme di dati all'interno del file,
  - quando ho bisogno di gestire accessi concorrenti in scrittura.
- Usare un DB per leggere, ma non modificare, un file intero di dimensioni non enormi è come comprare una Ferrari per andare a far la spesa. Funziona, ma è complicato, ed è uno spreco di tempo e risorse.

MongoDB - Introduzione

- MongoDB è un database NoSQL orientato ai documenti.
- Esso memorizza i documenti in JSON, formato basato su JavaScript e più semplice di XML, ma comunque dotato di una buona espressività.
- I documenti sono raggruppati in collezioni che possono essere anche eterogenee. Ciò significa che non c'è uno schema fisso per i documenti. Tra le collezioni non ci sono relazioni o legami garantiti da MongoDB.
- MongoDB si adatta a molti contesti, in generale quando si manipolano grandi quantità di dati eterogenei e senza uno schema.
- MongoDB fornisce un driver per JavaScript **lato server** ossia per **Node.js**. Sulle macchine di cs è preinstallato, sulle vostre dovrete installarlo di persona.

Concetti

- **Le collezioni** in Mongo sono la struttura fondamentale di Mongo, un po' tabella relazionale, un po' directory di file.
- **I documenti** sono strutture di contenimento di dati omogenei, come se fossero righe di una tabella.
- **I campi**, o proprietà o attributi, sono le celle della riga. Contengono valori semplici di riferimento al documento.
- **Schemi:** Mongo non ha schemi, ma attraverso Mongoose è possibile definire uno schema e forzare omogeneità tra i vari documenti di una collezione
- **Schema type**: Oltre a definire la sequenza e i nomi dei vari campi, Mongoose permette di definire anche i tipi di dati di ciascun campo. (String, Number, Boolean, ecc).
- I **Modelli** sono costruttori di alto livello che generano istanze di documenti a partire da uno schema. Garantiscono l'omogeneità strutturale tra documenti.

MongoDB - CRUD

- Un database è un insieme di collezioni che corrisponde ad un insieme di file nel disco fisso.
- Un'istanza di MongoDB può gestire più database. Di seguito mostreremo brevemente come effettuare le operazioni CRUD (Create, Read, Update, Delete).
- Su MongoDB, non esiste alcun comando esplicito per creare un database.
- Anche le collezioni vengono create implicitamente al primo utilizzo.

MongoDB - CRUD

- Per inserire dati:

```
try {
   let dbname ="mySite" ;
   let coll = "myCollection" ;
   let data = {
     name: "Fabio Vitali",
     age: 28,
     url: "http://www.fabiovitali.it",
     tags: ["Development", "Design"]
   } ;

   let result = await db[dbname]
     .collection[coll]
     .insert(data);
} catch (e) {
   alert("something went wrong")
}
```

```
try {
   let dbname ="mySite" ;
   let coll = "myCollection" ;
   let data = [{
     name: "Fabio Vitali",
     age: 28,
     url: "http://www.fabiovitali.it",
     tags: ["Development", "Design"]
}, {
     name: "Angelo Di Iorio",
     age: 26,
     url: "http://www.angelodiiorio.it"
     tags: ["Programming", "Systems"]
   }] ;

   let result = await db[dbname]
     .collection[coll]
     .insertMany(data);
} catch (e) {
   alert("something went wrong")
```

- La risposta corretta del server sarà tipo:

```
{
    "acknowledged": true,
    "insertedCount": 1,
    "insertedIds": {
        "0":"62694126e084590335f12f44"
    }
}
```

che dice che è stato inserito un record nella collezione.

MongoDB - CRUD

Per interrogare tutti i documenti presenti nella collezione:

```
db[dbname].collection[coll].find(query)
```

dove query è un oggetto complesso composto da valori, nomi di campo, e operatori (introdotti da $) e il nesting si ottiene attraverso contenimento.

SQL query	Mongo query
`SELECT * FROM dbname WHERE name = "Fabio Vitali"`	`db[dbname].collection[coll].find({ name: "Fabio Vitali" })`
`SELECT * FROM dbname WHERE age > 18`	`db[dbname].collection[coll].find({ age: { $lt: 18 } })`
`SELECT * FROM dbname WHERE name LIKE "fabio%" AND age > 18`	`db[dbname].collection[coll].find({ name: /^fabio/, age: { $lt: 18 } })`
`SELECT * FROM dbname WHERE name LIKE "fabio%" OR age > 18`	`db[dbname].collection[coll].find({ $or: { name: /^fabio/, age: { $lt: 18 } } })`

MongoDB - CRUD

L'aggiornamento dei documenti avviene tramite i metodi **updateOne, updateMany*(filter,update,options), e replaceOne***:

Mongo update

`db[dbname].collection[coll].updateOne(`
`  {  name: "Fabio Vitali" },`
`  {`
`  $set: { age: 35 }`
`  })`

`db[dbname].collection[coll].updateMany(`
`  {`
`  age: { $lt: 40 }`
`  },`
`  {`
`  $set: {`
`  age: 40,`
`  status: "presumed"`
`  }`
`  }`
`})`

MongoDB - CRUD

L'eliminazione di un documento dalla collezione viene effettuata per mezzo dei metodi **deleteOne** o **deleteMany**.

Mongo delete

`db[dbname].collection[coll].deleteOne(`
`  { name: "Fabio Vitali" }`
`)`

`db[dbname].collection[coll].updateMany(`
`  { age: { $lt: 40 } }`
`})`

MongoDB - Server

Ad esempio, per accedere ad una collection su un database di MongoDB la sintassi è la seguente:

```
var client = require('mongodb').MongoClient;
client.connect("mongodb://site2223XX:[PWD]@mongosite2223XX?writeCon
cern=majority",
    async function(error, db) {
        if(!error) {
            var people = db.collection("people");
            await people.insert(
                { nome: "Fabio", cognome: "Vitali" });
            db.close();
        }
    }
);
```

Mongoose

- Mongoose permette di imporre schemi sui documenti di una collezione Mongo, e creare omogeneità dove opportuno.

```
// save as people.js
let mongoose = require('mongoose');
let people = new mongoose.Schema({
  name: String,
  surname: String,
  email: String,
  age: Number
});

module.exports = mongoose.model('customer', people);
```

```
let Person = require('./people.js');

let fv = new Persom({
  name: "Fabio",
  surname: "Vitali",
  email: "fabio.vitali@unibo.it",
  age: 28
});
```

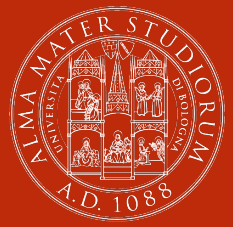
```
fv.save()
  .then((doc) => {
    console.log(doc);
  })
  .catch((err) => {
    console.error(err);
  });
```

MongoDB e il progetto

- Bisogna lanciare MongoDB su un docker separato.
- Usate gocker per lanciare MongoDB.

  create mongoDB site2223XX
- Il server risponde:

  Mongodb username: site2323XX - Mongodb password: YYYYYYY
  You can connect your mongodb from your site web using hostname mongo_site2223XX
- Questi tre dati vanno conservati ed utilizzati dallo script su node.js per collegarsi con successo.
- Mongo non è accessibile all'esterno, ma solo alla vostra installazione gocker.



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fabio Vitali

Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna

Fabio.vitali@unibo.it

www.unibo.it