Conception d'Applications Interactives : Applications Web

Séance #2 - Côté serveur

NodeJS

Côté navigateur

- Introduction à NodeJS
- Architecture
- L'asynchronisme
- Modules et gestion de dépendances
- Node et le Web (Http, Connect, Express)
- Communication temps réel
- La gestion des streams
- Persistance de données
- Forge
- Node en mode cluster
- Au delà de NodeJS

Introduction à NodeJS

Késaco ?

NodeJS is a JavaScript runtime

http://qlikshare.com/wp-content/uploads/2016/03/jsvsjava.jpg

C'est quoi NodeJS ?

- C'est un outil en ligne de commande
- Qui permet de faire tourner des programmes JavaScript depuis un terminal
  - node myProgramme.js
- Le JS est exécuté par le moteur V8
  - Celui qui fait que Google Chrome soit si rapide

C'est quoi NodeJS ?

- Node embarque des APIs multiples
  - Accès système de fichiers
  - Accès réseau
- Et a un modèle de programmation event-driven
  - avec IO non bloquante
- Ce qui le rend idéal pour faire des serveurs HTTP
  - légers et performants
  - capables de supporter des grandes charges

Mais je peux déjà tout faire en Java !

- Ce n'est pas une solution magique
- Ce n'est pas une solution universelle
- Mais il est pensé pour certaines tâches et il les fait bien

- NodeJS excelle dans les tâches qui demandent de la parallélisation

En NodeJS tout tourne en parallèle...
sauf votre code

Tout tourne en parallèle sauf mon code ?

- Métaphore du roi et ses serviteurs
  - Le roi donne des tâches à faire aux servants
  - Il ne s'en occupe plus, passe à une autre chose
  - Lorsque le servant a fini il revient vers le roi
  - Le roi peut se concentrer sur son travail

Le roi est votre code,

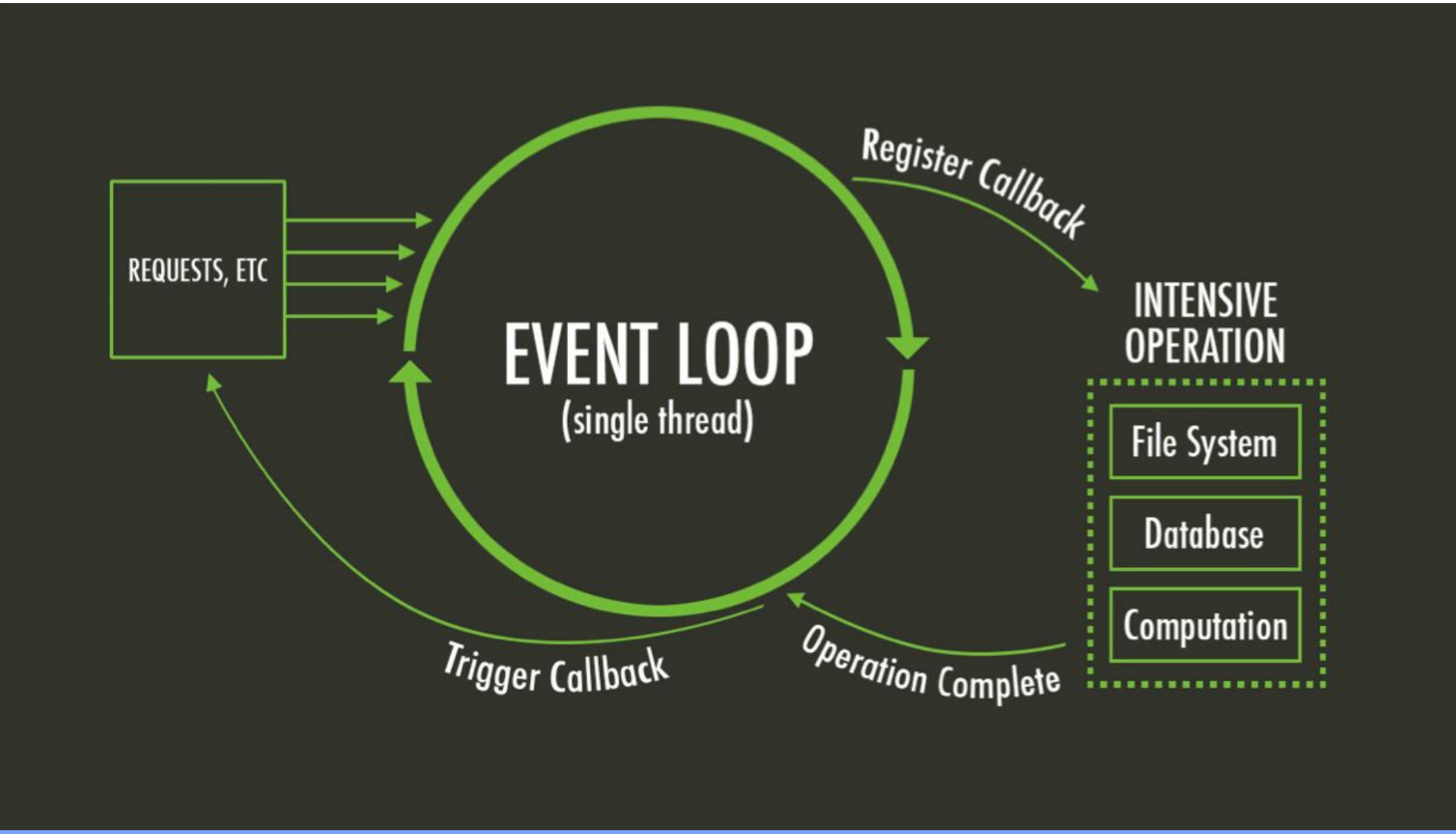
les serviteurs sont les APIs NodeJS

J'aime pas les métaphores...

- Votre code tourne sur un seul thread
- Lorsque vous avez des choses à faire, vous appelez des APIs NodeJS
- En leur donnant un callback, une fonction à appeler lorsqu'ils auront fini leur tâche
- Et votre code passe à une autre chose
  - Pas de problème de concurrence
  - Pas de goulot d'étranglement

That's the entire beauty of JavaScripts single-threaded / event loop design!

La boucle d'événements



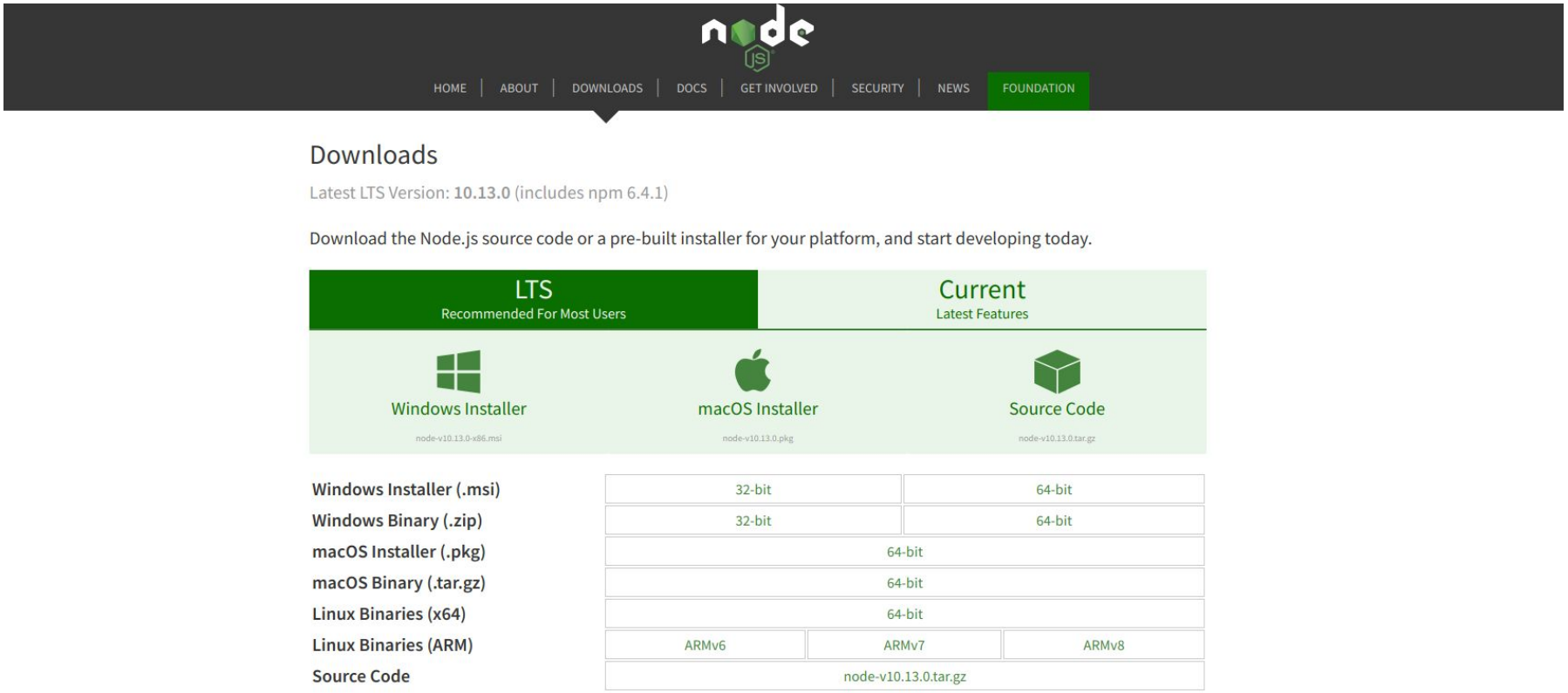
Des avantages

- Parallélisation
  - des accès aux sources de données
  - des tâches chronophages
- Asynchronisme
  - Tenu de charge
  - Pas de threads bloqués
- Vitesse
  - Moteur V8, très rapide
- JavaScript
  - Tout le monde connaît un peu de JS, non ?
  - Même langage dans le frontend que dans le backend

Hello Node

Installation et premier script

NodeJS, the ENIB's way



Install local de node

Pas besoin de droits d'admin

NodeJS, the ENIB's way

- Décompresser le binaire dans votre home
- Ajouter le répertoire ~/node-vx.y.z-xxxx/bin au PATH
  - export PATH=~/node-vx.y.z-xxxx/bin:$PATH
- Ouvrir un nouveau terminal et tester node --version :

```
$ node --version
v8.9.4
```

Hello Node!

- Fichier helloworld.js
  - console.log("Hello Node!");
- Lancer le programme avec Node :

```
$ node helloworld.js
Hello Node!
```

- Le rendre auto exécutable
  - Ajouter au fichier :
    - #!/usr/bin/env node
  - Lancer les commandes :

```
$ chmod +x helloworld.js
$ ./helloworld.js
Hello World
```

Et si on fait un serveur HTTP ?

- Fichier helloWorld.js :

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

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8888);
```

- Lancé en ligne de commande :

```
$ node helloWorld.js
```

Analysons le HelloWorld

- require("http") fait appel à un module NodeJS
  - Module pour gérer le HTTP
  - Des dizaines de modules disponibles

- http.createServer(function(request, response) { [...] }
  - On appelle à la fonction createServeur
  - En lui donnant comme argument une autre fonction
    - Un callback, qui sera appelé quand une requête sera reçue

Analysons le HelloWorld

- Lorsqu'on reçoit la requête, on exécutera

```
response.writeHead(200, {"Content-Type": "text/plain"});
response.write("Hello World");
response.end();
```

- Et on fait écouter sur le port 8888

```
listen(8888);
```

Event-driven asynchronous callbacks

- Concept à garder en tête avec NodeJS
  - Modèle applicatif à respecter
  - Le thread de votre code est unique
    - Il faut pas le bloquer
- Mauvaise pratique : synchronisme

```
var result = database.query("SELECT * FROM hugetable");
console.log("Hello World");
```

- Bonne pratique : asynchronisme

```
database.query("SELECT * FROM hugetable",
        (rows) => dealWithRows(rows));
console.log("Hello World");
```

Event-driven asynchronous callbacks

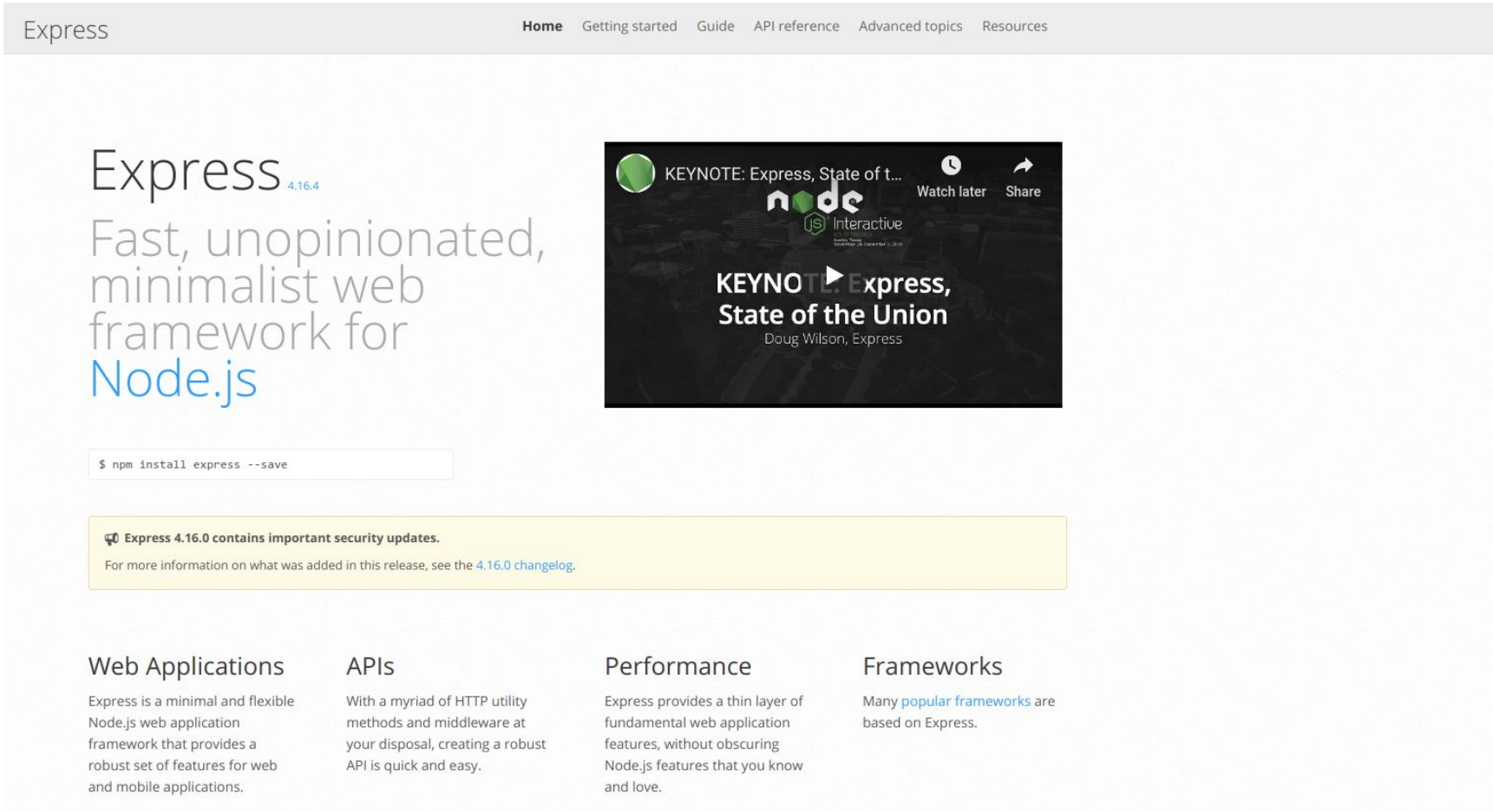
- Vérifions le comportement sur le HelloWorld

```
var http = require("http");
function onRequest(request, response) {
  console.log("Request received.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}

http.createServer(onRequest).listen(8888);

console.log("Server has started.");
```

Express JS

Des APIs rapides et souples avec Node JS

Express



Infrastructure Web minimaliste, souple et rapide pour Node JS

Du l'outillage

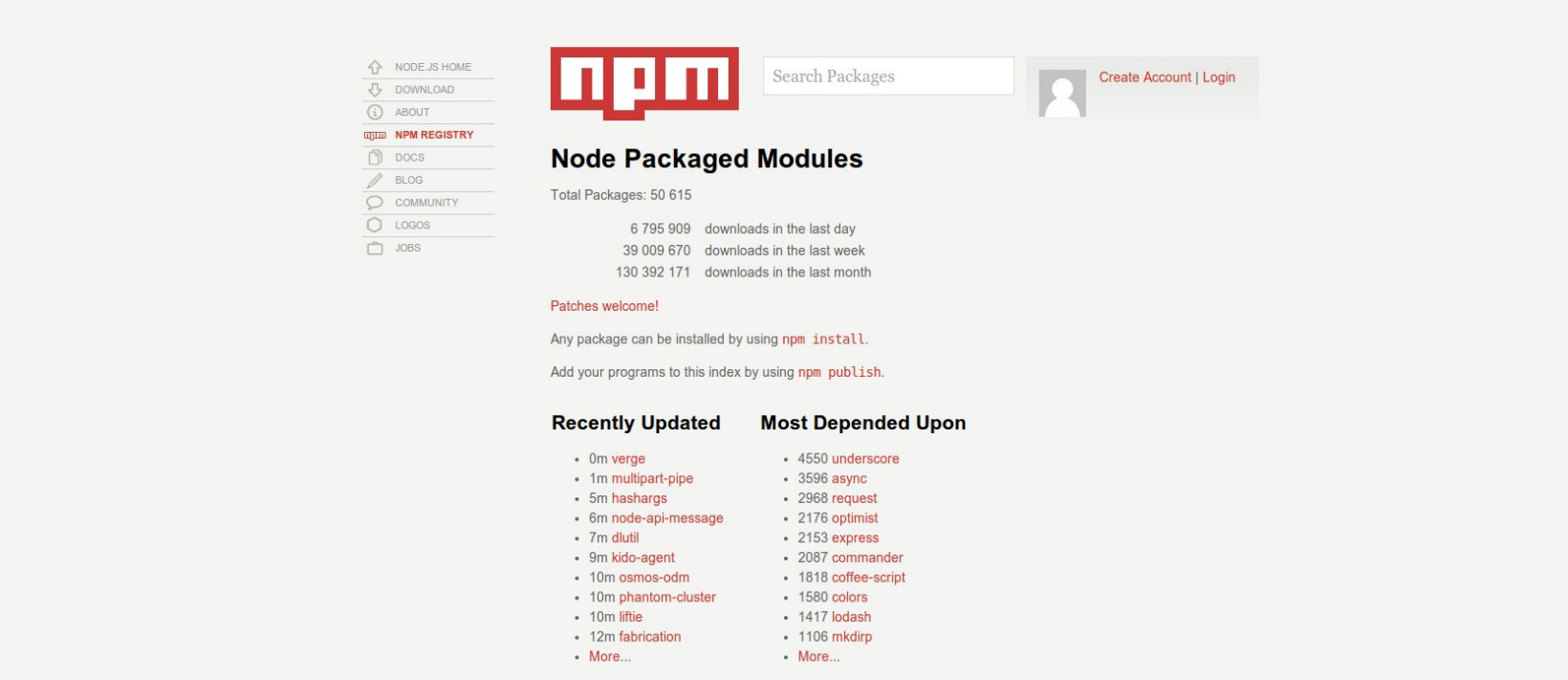
npm, package.json...

What is npm?

- NodeJS built modularly
  - Each functionality in a package

- npm is the official package manager for Node.js
  - runs through the command line
  - manages dependencies for an application
  - install applications available on the npm registry

What is npm registry?



NodeJS equivalent to Maven Central

http://npmjs.org

npm behind a corporate proxy

- Proxy must be defined as environment variable

```
export http_proxy=user:password@proxy.example.com:3128
export https_proxy=user:password@proxy.example.com:3128
```

  - Potential problem with `proxy-pac`…

- Using of `npm config` could be needed

```
npm config set proxy http://proxy.example.com:3128
npm config set https-proxy http://proxy.example.com:3128
```

- If necessary use credentials:

```
npm config set proxy http://user:password@proxy.example.com:3128
npm config set proxy https://user:password@proxy.example.com:3128
```

Exercise: our first NodeJS app

package.json

Either written by hand or using `npm init`

```
{
 "name": "awesome-test",
 "main": "server.js"
}
```

server.js

Main file

```
console.log('Hello World');
```

Run the project using node server.js

Restarting a Node Application on File Changes

- NodeJS won't restart when file changes are made
  - We need a 3rd party package for that: nodemon

```
npm install -g nodemon
```

  - Then use nodemon instead of node command

```
nodemon server.js
```

Installing packages

To install a package for our app we add it to packages.json

- By manually writing the dependency

```
{
  "name": "awesome-test",
  "main": "server.js",
   "dependencies": {
      "express": "~4.8.6"
   }
}
```

- By using the command line

npm install express --save

An HTTP server in pure NodeJS

package.json

Either written by hand or using npm init

```
{
 "name": "http-server",
 "main": "server.js"
}
```

index.html

Static index file

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Super Cool Site</title>
</head>
<body>
        <h1>Hello Universe!</h1>
</body>
</html>
```

An HTTP server in pure NodeJS

```
// get the http and filesystem modules
var http = require('http')
var fs = require('fs');

// create our server using the http module
http.createServer(function(req, res) {
    // write to our server. set configuration for the response
    res.writeHead(200, {
     'Content-Type': 'text/html',
     'Access-Control-Allow-Origin' : '*'
    });
    var readStream = fs.createReadStream(__dirname + '/index.html');
    // send a message
    readStream.pipe(res);
}).listen(1337);
// tell ourselves what's happening
console.log('Visit me at http://localhost:1337');
```

So what's ExpressJS?

ExpressJS

Lightweight platform for building web apps using NodeJS

Lots of useful features:

- Router
- Handling Requests
- Application Settings
- Middleware

An HTTP server with ExpressJS

package.json

Either written by hand or using npm init

```
{
 "name": "http-server",
 "main": "server.js",
  "dependencies": {
    "express": "~4.8.6"
  }
}
```

index.html

Static index file

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Super Cool Site</title>
</head>
<body>
        <h1>Hello Universe!</h1>
</body>
</html>
```

An HTTP server with ExpressJS

```
// load the express package and create our app
var express = require('express');
var app     = express();

// send our index.html file to the user for the home page
app.get('/', function(req, res) {
        res.sendFile(__dirname + '/index.html');
});

// start the server
app.listen(1337);
console.log('1337 is the magic port!');
```

Express Router

Full-fledged routing engine:

- Basic Routes
- Site Section Routes (Admin section with sub routes)
- Route Middleware to log requests to the console
- Route with Parameters
- Route Middleware for Parameters to validate specific parameters
- Login routes doing a GET and POST on /login
- Validate a parameter passed to a certain route

Basic routes

```
// load the express package and create our app
var express = require('express');
var app     = express();
// set the port based on environment
// (more on environments later)
var port    = 1337;

// send our index.html file to the user for the home page
app.get('/', function(req, res) {
        res.sendFile(__dirname + '/index.html');
});

// get an instance of the router
var adminRouter = express.Router();

// create routes for the admin section
adminRouter.get('/', function(req, res) {
        res.send('I am the dashboard!');
});
```

```
// users page
adminRouter.get('/users', function(req, res) {
        res.send('I show all the users!');
});

// users page
adminRouter.get('/users/:aUser', function(req, res) {
        res.send('I show the user '+req.params.aUser);
});
// posts page
adminRouter.get('/posts', function(req, res) {
        res.send('I show all the posts!');
});

// apply the routes to our application
app.use('/admin', adminRouter);
```

Route Middleware

Actions to do before a request is processed

```
// route middleware that will happen on every request
adminRouter.use(function(req, res, next) {

    // log each request to the console
    console.log(req.method, req.url);

    // continue doing what we were doing and go to the route
    next();
});
```