

The MEAN Stack

Dr. Alejandro Zunino
ISISTAN – UNICEN

1

Why?

- Write one language
- With MongoDB and Mongoose, easy and flexible data management
- With NodeJS, never need threads
- With AngularJS, dynamic client-side templates

3

MongoDB


- **Document database** that provides high performance, high availability, and automatic scaling

```
{
  "_id" : ObjectId("54c955492b7c8eb21818bd09"),
  "address" : {
    "street" : "2 Avenue", "zipcode" : "10075",
    "building" : "1480", "coord" : [ -73.9557413, 40.7720266 ]
  },
  "borough" : "Manhattan", "cuisine" : "Italian",
  "grades" : [
    {
      "date" : ISODate("2014-10-01T00:00:00Z"), "grade" : "A", "score" : 11
    },
    {
      "date" : ISODate("2014-01-16T00:00:00Z"), "grade" : "B", "score" : 17
    }
  ],
  "name" : "Vella", "restaurant_id" : "41704620"
}
```



5

The MEAN Stack

 **ANGULARJS**
by Google
Rich frontend framework

express
Web dev framework for NodeJS


Event-based concurrency framework

 **mongoDB**
{ name: mongo, type: DB }
Key-value database

2

Same Language, Same Objects

 **mongoDB**
{ name: mongo, type: DB }



 **ANGULARJS**
by Google

```
{ "_id" :  
  ObjectId("5161a58b46341f8a46000003"),  
  "username" : "vkarpov" }
```

```
{ "_id" :  
  ObjectId("5161a58b46341f8a46000003"),  
  "username" : "vkarpov" }
```

```
{ "_id" :  
  ObjectId("5161a58b46341f8a46000003"),  
  "username" : "vkarpov" }
```

4

MongoDB

- Unlike a table, however, a collection does not require its documents to have the same schema
- documents stored in a collection must have a unique `_id` field that acts as a primary key.
- Insert, Find, Update, Remove
- Aggregate: similar to Select in SQL
- Indexes to improve performance



6

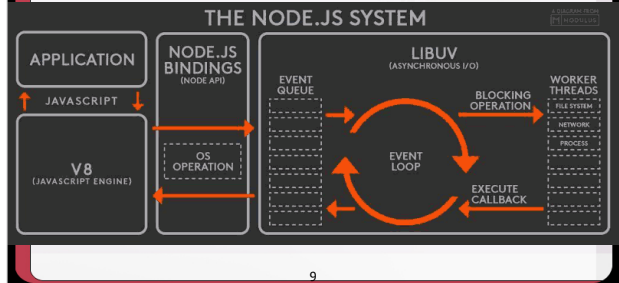
NodeJS: JavaScript Server Framework

- Built on Google Chrome's V8 JavaScript runtime.
- Event based concurrency:
 - no threading issues!
 - but... callbacks (async)
 - event-driven, non-blocking I/O model
 - event-loops via JavaScript's callback functionality to implement the non-blocking I/O.
 - Everything inside Node.js runs in a single-thread.
- Package manager similar to Rails gems or Ubuntu apt-get

7

NodeJS: Event Loops

- NodeJS is event-based and everything is **asynchronous or non-blocking**



9

NodeJS: what for?

- You can create an HTTP server and print 'hello world' on the browser in just 4 lines of JavaScript. (Example included)
- You can create a TCP server similar to HTTP server, in just 4 lines of JavaScript. (Example included)
- You can create a Static File Server.
- You can create a Web Chat Application like WhatsApp in the browser.
- Node.js can also be used for creating online games, collaboration tools or anything which sends updates to the user in real-time.

11

NodeJS: JavaScript Server Framework

- Here is a simple example, which prints 'hello world'

```
var sys = require("util");
setTimeout(function(){
    console.log("world");
}, 3000);
console.log("hello");
//it prints 'hello' first and waits for 3 seconds
and then prints 'world'
```

8

NodeJS: non-blocking I/O

- Traditional I/O


```
var result = db.query("select x from table_Y");
doSomethingWith(result); //wait for result!
doSomethingWithoutResult(); //execution is blocked!
```
- Non-blocking I/O


```
db.query("select x from table_Y", function (result){
    doSomethingWith(result); //wait for result!
});
doSomethingWithoutResult(); //executes without any delay!
```

10

NodeJS: examples

- The following code creates an HTTP Server and prints 'Hello World' on the browser:

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

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(5000,
"127.0.0.1");
```

12

NodeJS: examples

- Simple TCP server which listens on port 6000 and echoes whatever you send it:

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

net.createServer(function (socket) {

  socket.write("Echo server\r\n");

  socket.pipe(socket); }

).listen(6000, "127.0.0.1");
```

13

express

- Minimalist web framework for Node.js
- Provides libraries for:
 - Speak HTTP: Accept TCP connections, process HTTP request, send HTTP replies
 - Routing: Map URLs to the web server function for that URL
 - Middleware: Allow request processing layers to be added in
 - HTML templating
 - SSL
 - Authentication
 - Many more

15

express

```
mkdir expresshello
cd expresshello
npm init # answer some questions...
npm install express --save
```

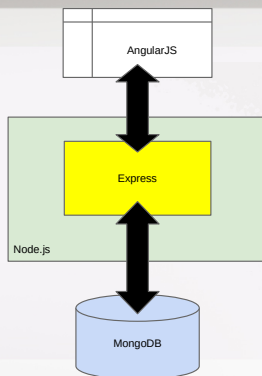
17

NodeJS modules

- Node.js heavily relies on modules, in previous examples require keyword loaded the http & net modules.
- Creating a module is easy, just put your JavaScript code in a separate js file and include it in your code by using keyword require, like:
 - var modulex = require('./modulex');
- Libraries in Node.js are called packages and they can be installed by typing
 - npm install --save "package_name"; *//package should be available in npm registry @ npmjs.org*
- NPM (Node Package Manager) comes bundled with Node.js installation.

14

Typical MEAN architecture



16

express

Create a file app.js and execute (node app.js):

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send("Hello World!");
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

18

• Client side framework for **Single Page Applications** (more on this later...):

- Handles: DOM manipulation, input validation, server communication, URL mangement, etc.
- Uses the MVC pattern
- HTML Templating approach with two-way binding
- Minimal server-side support dictated
- Modules, reusable components, testing, etc.

19

```
<!DOCTYPE html>
<html lang="en-US">
<script
src="http://ajax.goo
n.js"></script>
<body>

<div ng-app="">
  <p>Name : <input type="text" ng-model="name"></p>
  <h1>Hello {{name}}</h1>
</div>

</body>
</html>
```

Scans the ng-app attribute and creates a **scope**

21

Try the Google Chrome Inspector

```
<!DOCTYPE html>
<html lang="en-US">
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.mi
n.js"></script>
<body>

<div ng-app="" class="ng-scope">
  <p>Name : <input type="text" ng-model="name" class="ng-
pristine ng-untouched ng-valid"></p>
  <h1 class="ng-binding">Hello </h1>
</div>

</body>
</html>
```

23

```
<!DOCTYPE html>
<html lang="en-US">
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8
/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name : <input type="text" ng-model="name"></p>
  <h1>Hello {{name}}</h1>
</div>

</body>
</html>
```

Name : "this is my name"

Hello "this is my name"

20

```
<!DOCTYPE html>
<html lang="en-US">
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.mi
n.js"></script>
<body>

<div ng-app="">
  <p>Name : <input type="text" ng-model="name"></p>
  <h1>Hello {{name}}</h1>
</div>

</body>
</html>
```

Compiler - Scans DOM covered by the ng-app looking for templating markup - Fills in with information from scope.

22

Two way binding: type something

```
<!DOCTYPE html>
<html lang="en-US">
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.mi
n.js"></script>
<body>

<div ng-app="" class="ng-scope">
  <p>Name : <input type="text" ng-model="name" class="ng-
valid ng-dirty ng-valid-parse ng-touched"></p>
  <h1 class="ng-binding">Hello you</h1>
</div>

</body>
</html>
```

24

Concept	Description
Template	HTML with additional markup
Directives	extend HTML with custom attributes and elements
Model	the data shown to the user in the view and with which the user interacts
Scope	context where the model is stored so that controllers, directives and expressions can access it
Expressions	access variables and functions from the scope
Compiler	parses the template and instantiates directives and expressions
Filter	formats the value of an expression for display to the user
View	what the user sees (the Document Object Model)
Data Binding	sync data between the model and the view
Controller	the business logic behind views
DI	Creates and wires objects and functions
Injector	dependency injection container
Module	a container for the different parts of an app (cont, serv, filt, conf)
Service	reusable business logic independent of views

Conclusions

- Complete Web stack based on JavaScript
- Mature, popular, enterprise ready, well documented
- There are many MEAN-like solutions:
 - Generators
 - Templates: mean.io, mean.js, ...
- Avoid:
 - When you are doing heavy and CPU intensive calculations on server side:
 - delegate these to some other tech such as NodeJS clusters, ESB