

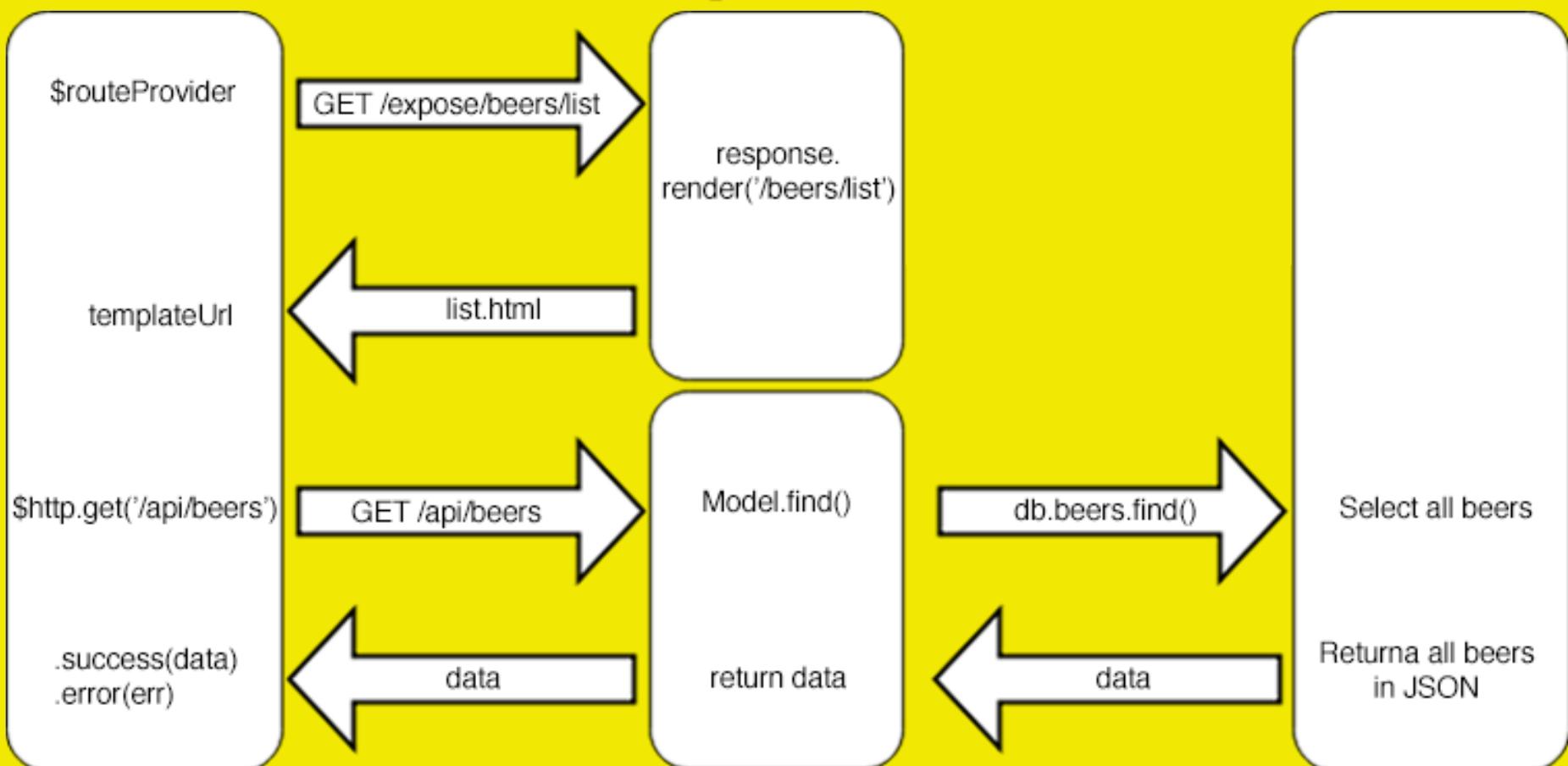


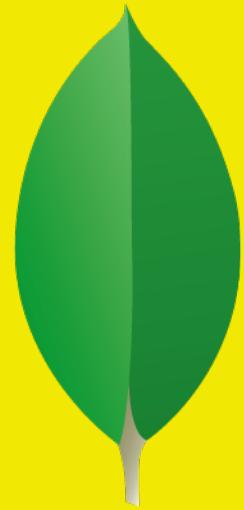
Be MEAN

Objetivo

Criar uma single page application simples, CRUD, integrando frontend, backend e banco de dados totalmente com Javascript.

node express





mongoDB

**NO
SQL**

Pense em cervejas, cada tipo de cerveja é diferente.

Pense no Relacional como as Pilsens que são as mais comuns achadas em qualquer boteco. Agora as cervejas NoSQL são as diferentes, cada uma tem suas características marcantes e podem ser agrupadas pelo tipo. E apenas quem possui gosto apurado usa.





**Com bancos relacionais o que importa são
quais respostas você tem, com os bancos
NoSQL são as perguntas que importam.**



Principais grupos de NoSQL

- chave/valor
- documento
- grafos
- colunas

Chave/valor



tetra



externa



interna

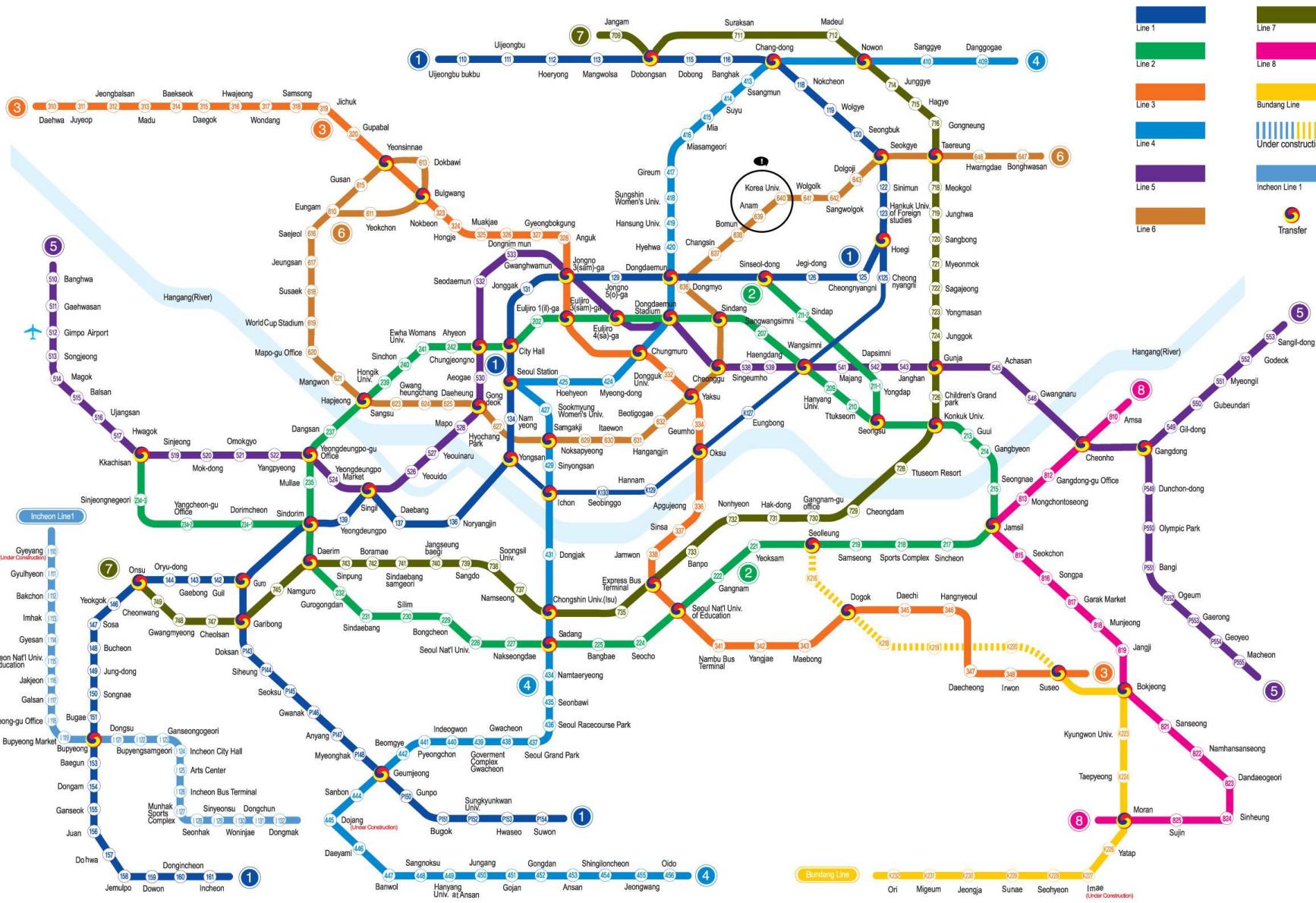


banheiro

Documento



Grafos

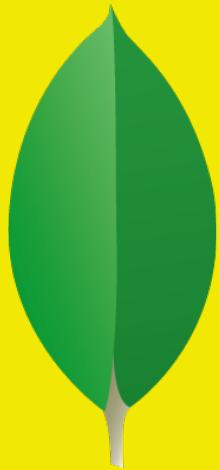


Colunas

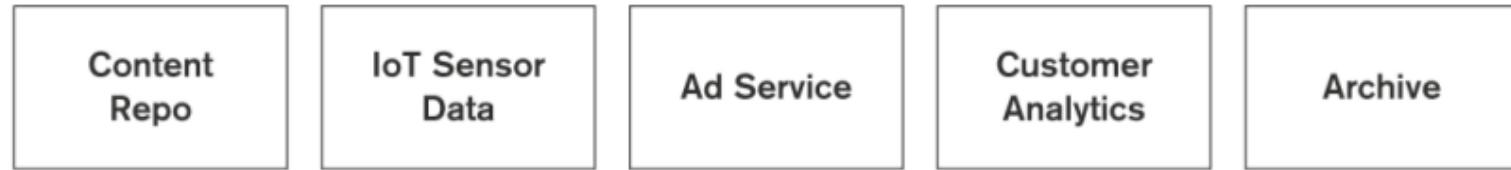


MongoDB

- C++
- Schemaless
- JSON/BSON
- Replica
- Sharding
- GridFS
- Geolocation

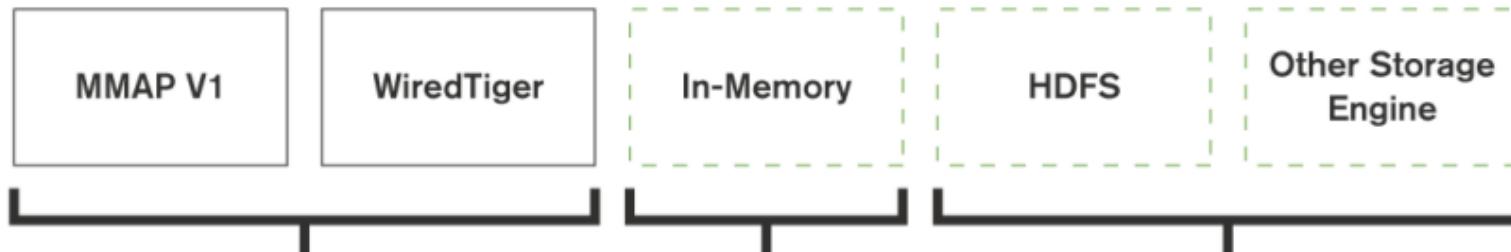


mongoDB³



MongoDB Query Language

MongoDB Data Model



Supported in MongoDB 3.0

Experimental in
MongoDB 3.0

Future Possible Storage Engines

Security

Management

	MongoDB WiredTiger	MongoDB MMAPv1
Write Performance	Excellent Document-Level Concurrency Control	Good Collection-Level Concurrency Control
Read Performance	Excellent	Excellent
Compression Support	Yes	No
Maximum Document Size	16 MB*	16 MB*
MongoDB Query Language Support	Yes	Yes
Secondary Index Support	Yes	Yes
Replica Sets	Yes	Yes
Automatic Sharding	Yes	Yes
Ops Manager & MMS Support	Yes All features including deployment, upgrade backup, restore, and monitoring	Yes All features including deployment, upgrade backup, restore, and monitoring
Security Controls	Yes	Yes
Platform Availability	Linux, Windows, Mac OS X	Linux, Windows, Mac OS X, Solaris (x86)

* GridFS supports larger file sizes

Table 1: Comparing the MongoDB WiredTiger and MMAPv1 storage engines

MongoDB - Terminologia

SQL RDBMS MongoDB

DATABASE DATABASE

TABLE COLLECTION

ROWS DOCUMENT JSON

QUERY QUERY

INDEX INDEX

PARTITION SHARD

MongoDB - Coleção

```
use meu_db
```

```
show dbs
```

```
db.minha_collection.comando()
```

Ex.:

```
db.products.drop() //apaga coleção
```

```
db.products.remove() //limpa coleção
```

MongoDB - Insert

```
> var product = {"name":"Cachaça","description":"Mé  
brasileiro","price":12.00 }  
> db.products.insert(product)
```

```
/**
```

Exercício inserir mais esses produtos:

```
{"name":"Pinga","description":"da braba po tubão","price":  
4.50}  
{ "name": "Uísque", "description": "Pra preiboi toma com  
energético", "price": 80.00 }  
{ "name": "Champagne", "description": "só podia ser  
saopaulino", "price": 130.00 }  
*/
```

MongoDB - Save

```
> var product = db.products.findOne()  
> product.price = 999  
> db.products.save(product)
```

MongoDB - Cursor

Qualquer busca feita com o `find()` irá retornar um cursor e, para retirarmos os valores dele, precisamos iterar nele como no exemplo abaixo:

```
var cur = db.products.find();
while( cur.hasNext() ) { print(tojson(cur.next())) };
```

MongoDB - Find

Consulta

Para fazermos consultas no MongoDB, usamos 2 funções:

`find()`

`findOne()`

Possuímos a seguinte sintaxe para fazermos consultas:

```
db.coleta.find({clausulas}, {campos})
```

E, para usarmos como WHERE, são necessários alguns operadores especiais que listo na próxima página. Caso seja necessária uma consulta por string mais complexa, podemos usar REGEX.

```
db.professores.find( { nome : /^(\J.*?)/i } );
```

```
db.professores.find( { name : { $regex : '/^(\J.*?)/i' } } ) ;
```

MongoDB - Operadores

< ou \$lt

```
db.collection.find({ "campo" : { $lt: value } } );
```

Retorna objetos com valores menores que value.

<= ou \$lte

```
db.collection.find({ "campo" : { $lte: value } } );
```

Retorna objetos com valores menores ou igual que value.

> ou \$gt

```
db.collection.find({ "campo" : { $gt: value } } );
```

Retorna objetos com valores maiores que value.

>= ou \$gte

```
db.collection.find({ "campo" : { $gte: value } } );
```

Retorna objetos com valores maiores ou igual que value.

MongoDB - Operadores

\$or

```
db.collection.find( { $or : [ { a : 1 } , { b : 2 } ] } )  
db.foo.find( { name : "bob" , $or : [ { a : 1 } , { b : 2 } ] } )
```

Retorna objetos caso a cláusula OU for verdadeira.

\$nor

```
db.collection.find( { $nor : [ { a : 1 } , { b : 2 } ] } )
```

Retorna objetos caso a cláusula negação do OU for verdadeira, ou seja, não pode encontrar nenhuma cláusula verdadeira.

\$and

```
db.foo.insert( { a: [ 1, 10 ] } )  
db.foo.find( { $and: [ { a: 1 } , { a: { $gt: 5 } } ] } )
```

Retorna objetos caso a cláusula E for verdadeira.

MongoDB - Operadores

\$ne

```
db.collection.find( { x : { $ne : 3 } } );
```

Retorna objetos se o valor não for igual.

\$in

```
db.collection.find( { campo : { $in : array } } );
```

```
db.professores.find( { cursos : { $in : [Node.js, MongoDB] } } );
```

Retorna objetos se o valor foi encontrado.

\$nin

```
db.collection.find({j:{$nin: [2,4,6]}},
```

Retorna objetos se nenhum dos valores foi encontrado.

MongoDB - Operadores

\$all

```
{db.collection.find( { a: { $all: [ 2, 3 ] } } );}
```

O operador \$all é similar ao \$in, a diferença é que em vez de acertar apenas um valor da lista, tem que acertar todos os valores da lista.

\$exists

```
db.collection.find( { campo : { $exists : true } } );
```

Retorna o objeto caso o campo exista.

\$not

```
db.collection.find( { campo : { $not : { $gt: 666 } } } );
```

Retorna o objeto que não satisfaz a condição do campo, isso inclui documentos que não possuem o campo.

MongoDB - Alteração

A sintaxe para a função update() é a seguinte:

```
db.collection.update( query, mod, upsert, multi )
```

Exemplo:

```
product = db.products.findOne( { name : "Pinga" } );
product.price = 7.25;
db.products.save(product);
```

```
/**
```

lembando que o findOne retorna apenas um registro,
ja o find mesmo retornando um registro retornará dentro de um cursor

<http://www.mongodb.org/display/DOCS/Queries+and+Cursors>

```
*/
```

MongoDB - Operadores Modificação

Os operadores de modificação servem para que você possa alterar valores nos registros diretamente na query do MongoDB.

```
db.nome_colection.update( {query_da_busca}, {operador_mod}, upsert, multi )
```

MongoDB - Operadores Modificação

\$set

```
{ $set : { campo : valor } }
```

```
db.professores.update( { nome: 'Suissa' }, { $set: { idade: 28 } } );
```

Seta o valor do campo.

\$unset

```
{ $unset : { campo : 1} }
```

```
db.professores.update( { nome: 'Suissa' }, { $unset: { idade: 1 } } );
```

Deleta o campo.

MongoDB - Operadores Modificação

\$inc

```
{ $inc : { campo : valor } }  
db.products.update( { name: 'Pinga' }, { $inc: { views: 1 } } );
```

Incrementa um valor no campo; caso o campo não exista, ele irá criar o campo e setar o valor. Para decrementar, basta passar um valor negativo.

MongoDB - Operadores Modificação

\$push

```
{ $push : { campo : valor } }  
db.products.update( { name: 'Pinga' }, { $push: { tags:  
'marvada' } } );
```

Adiciona um valor ao campo se o campo for um array existente. Caso contrário, transforma o campo em um array com o valor como índice. Porém, se o campo existe e não for um array, irá retornar um erro.

Arrays múltiplos podem ser alterados em uma única operação separando os pares de campo : valor por vírgula.

```
{ $push : { campo1 : valor, campo2 : valor2 } }
```

MongoDB - Operadores Modificação

\$pushAll

```
{ $pushAll : { campo : valor_array } }
```

```
db.professores.update( { nome: 'Suissa' }, { $pushAll: {  
cursos: [ 'MongoDb', 'AngularJs', 'Node.js' ] } } );
```

Adiciona cada valor dentro de `valor_array` para o campo se o campo for um array existente. Caso contrário, seta o campo com o `valor_array`. Se o campo estiver presente mas não for um array, irá retornar um erro.

MongoDB - Operadores Modificação

\$pull

```
{ $pull : { campo : valor } }  
db.products.update( { name: 'Pinga' }, { $pull: { tags: 'marvada' } } );
```

Remove um valor ao campo se o campo for um array existente. Caso contrário, transforma o campo em um array com o valor como índice. Porém, se o campo existe e não for um array, irá retornar um erro.

Arrays múltiplos podem ser alterados em uma única operação separando os pares de campo : valor por vírgula.

```
{ $pull : { campo1 : valor, campo2 : valor2 } }
```

MongoDB - Operadores Modificação

\$pullAll

```
{ $pullAll : { campo : valor_array } }
```

```
db.professores.update( { nome: 'Suissa' }, { $pullAll: {  
cursos: [ 'MongoDb', 'AngularJs', 'Node.js' ] } } );
```

Adiciona cada valor dentro de `valor_array` para o campo se o campo for um array existente. Caso contrário, seta o campo com o `valor_array`. Se o campo estiver presente mas não for um array, irá retornar um erro.

MongoDB - Count

```
select count(*) from products  
db.products.count()
```

```
select count(*) from products where price > 2  
db.products.count({price:{$gt: 2}})
```

MongoDB - Find

Agora veremos um paralelo entre algumas funções de seleção da SQL e sua alternativa para o Mongo:

```
select count(*) from professores  
db.professores.count()
```

```
select count(*) from professores where idade > 30  
db.professores.count({idade:{$gt: 30}})
```

```
select nome from professores where idade < 18  
db.professores.find({idade: {$lt: 18}}))
```

```
select nome, idade from professores  
db.professores.find(null, {nome:1, idade:1} )
```

```
select nome, idade from professores where idade >= 18  
db.professores.find({idade:{$gte: 18}}, {nome:1, idade:1} )
```

MongoDB - Ordenação

Para ordenarmos uma consulta no MongoDB, precisamos apenas utilizar a função `sort()`, como no exemplo a seguir:

```
db.products.find( {} ) .sort( {price: 1} );
```

Utilizamos o valor 1 para ordenação ASCENDENTE e -1 para ordenação DESCENDENTE.

MongoDB - Ordenação

```
select * from products order by name ASC  
db.products.find().sort({name:1})
```

```
select * from products order by idade DESC  
db.products.find().sort({price:-1})
```

MongoDB - Limit

```
select * from products order by name ASC limit 0,2  
db.products.find().sort({name:1}).limit(2)
```

```
select * from products limit 2 offset 10  
db.products.find().limit(2).skip(10)
```

```
//10 produtos por pagina na página 2  
//limit = 10  
skip = numero da pagina * limit  
//pag 0  
db.products.find().limit(10).skip(0)  
//pag 1  
db.products.find().limit(10).skip(10)
```

MongoDB - Find

```
select * from professores order by nome ASC  
db.professores.find().sort({nome:1})
```

```
select * from professores order by idade DESC  
db.professores.find().sort({idade:-1})
```

```
select * from professores order by nome ASC limit 0,2  
db.professores.find().sort({nome:1}).limit(2)
```

```
select * from professores limit 2 offset 10  
db.professores.find().limit(2).skip(10)
```

MongoDB - Exclusão

```
db.products.remove({name: "Champagne"});  
// exclui todos os registro com name = Champagne
```

```
db.products.drop();  
// exclui tudo
```

MongoDB - Índices

Para criarmos um índice usamos o `ensureIndex()`.

```
db.products.ensureIndex({ name:1 }) ; db.products.  
ensureIndex({ name:1, price:-1 }) ;
```

Para pegarmos todos os índices da nossa collection `professores` usamos `getIndexes()`.

```
db.products.getIndexes()
```

Para pegar todos os índices da database usamos o `find()` em uma collection do sistema.

```
db.system.indexes.find()
```

Para deletar um índice usamos o seguinte comando

```
db.products.dropIndex("name_1")
```

MongoDB - Explain

Um ótimo comando para verificarmos como nossas queries estão sendo executadas também podemos usar o conhecido explain e ele será de grande utilidade quando precisarmos otimizar nossas queries.

```
db.products.find( {} ) .explain () ;  
{  
    "cursor" : "BasicCursor",  
    "indexBounds" : [ ],  
    "nscanned" : 57594,  
    "nscannedObjects" : 57594,  
    "nYields" : 2 ,  
    "n" : 3 ,  
    "millis" : 108,  
    "indexOnly" : false,  
    "isMultiKey" : false,  
    "nChunkSkips" : 0  
}
```

MongoDB - Explain

Vamos entender um pouco melhor esses valores:

- cursor: tipo de índice usado
- BasicCursor: sem index, fez um tablescan
- BtreeCursor: uso de algum índice
- indexBounds: faixa de busca usada
- nscanned: número de itens(documentos e índices) scaneados
- nscannedObject: número de documentos scaneados
- nYields: número de vezes que esperou um lock de leitura
- n: número de documentos encontrados para a busca
- millis: tempo de execução da query em milisegundos
- indexOnly: verdadeiro se a busca pode ser feita apenas com índice
- isMultiKey: verdadeiro de um índice múltiplo foi usado
- nChunkSkips: número de documentos ignorados por causa da migração de chunks de sharding



Relacionamentos

No MongoDB podemos utilizar 2 formas de relacionar os documentos.

Podemos usar seu ObjectId ligando diretamente pelo seu `_id` e fazendo uma segunda busca para retornar o documento relacionado ou podemos usar o DBRef que faz essa ligação automática.

MongoDB - DBRef

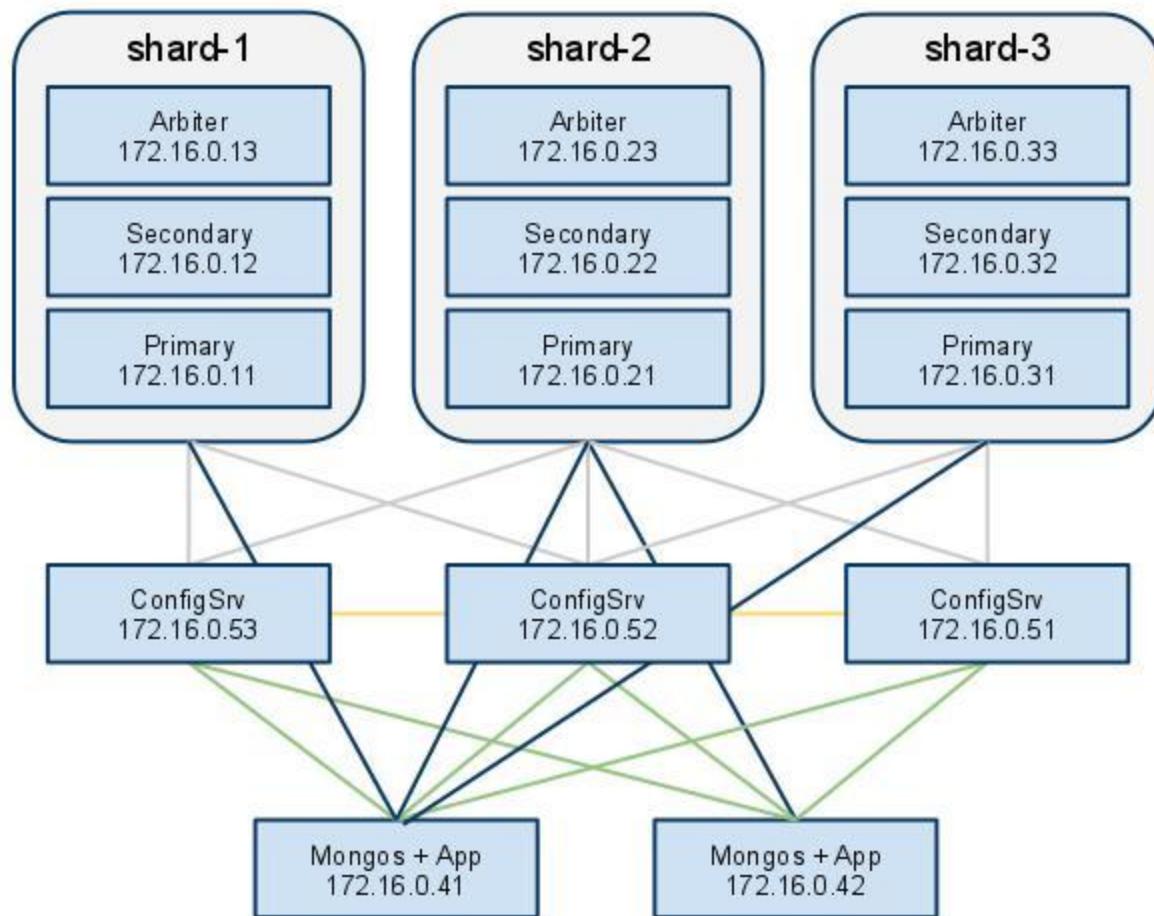
DBRef é o esquema nativo de referencia interna entre os documentos.

```
> db.professores.insert({login: "suissa"})
> db.professores.find()
{ "_id" : ObjectId("51ccffa2c49bec6fd946c929") , "login" :
"suissa" }
> db.workshop.insert({nome: "Be MEAN", professor: {"$ref": "professores", "$id": "51ccffa2c49bec6fd946c929"} })
> var obj = db.workshop.findOne()
> obj
{
  "_id" : ObjectId("51ccffe9c49bec6fd946c92a"),
  "nome" : "Be MEAN",
  "professor" : DBRef("users",
"51ccffa2c49bec6fd946c929")
}
```

MongoDB - Sharding

Sharding é a divisão dos dados em partes e divididas entre os nós.

O MondoDB ira dividir nossas informações através dos servidores de sharding setados em sua configuração, levando em conta a shard key.



MongoDB - Sharding Configurando

Primeiramente precisamos criar os serviços de sharding. Esse exemplo é apenas didático e será rodado em apenas um servidor.

```
mkdir /data/db/a /data/db/b
```

```
mongod --shardsvr --dbpath /data/db/a --port 10000 >
/tmp/sharda.log &
cat /tmp/sharda.log
```

```
mongod --shardsvr --dbpath /data/db/b --port 10001 >
/tmp/shardb.log &
cat /tmp/shardb.log
```

Obs.: O cat serve apenas para verificarmos se o serviço subiu realmente e o & serve para rodar em background.

MongoDB - Sharding Configurando

Vamos configurar os servidores.

```
mkdir /data/db/config
```

```
mongod --configsvr --dbpath /data/db/config --port 20000 >
/tmp/configdb.log &
cat /tmp/configdb.log
```

```
mongos --configdb localhost:20000 > /tmp/mongos.log &
cat /tmp/mongos.log
```

MongoDB - Sharding Configurando

Precisamos dizer qual database, collection e nossa shard key.

```
db.runCommand( { enablesharding : "database" } )
{ "ok" : 1 }
```

```
db.runCommand( { shardcollection : "database.colecao", key
: { name : 1} } )
{ "collectionsharded" : "test.people", "ok" : 1 }
```

MongoDb GridFs

MongoDB - GridFs

GridFS é o sistema de arquivos do MongoDB e deve ser usado quando precisamos armazenar arquivos maiores que 1Mb.

```
mongofiles -d myfiles put my_music.mp3
connected to: 127.0.0.1

added file: {
  _id: ObjectId('4ce9ddcb45d74ecaa7f5a029'),
  filename: "my_music.mp3",
  chunkSize: 262144,
  uploadDate: new Date(1290395084166),
  md5: "7872291d4e67ae8b8bf7aea489ab52c1",
  length: 1419631 }
```

MongoDB - Admin Uis

Sistemas para administração visual são uma mão na roda para qualquer Banco de dados, como o largamente usado phpmyadmin, para o Mysql. Não seria diferente para o MongoDB, existem diversos sistemas desses nas mais variadas linguagens, uma boa listagem se encontra em

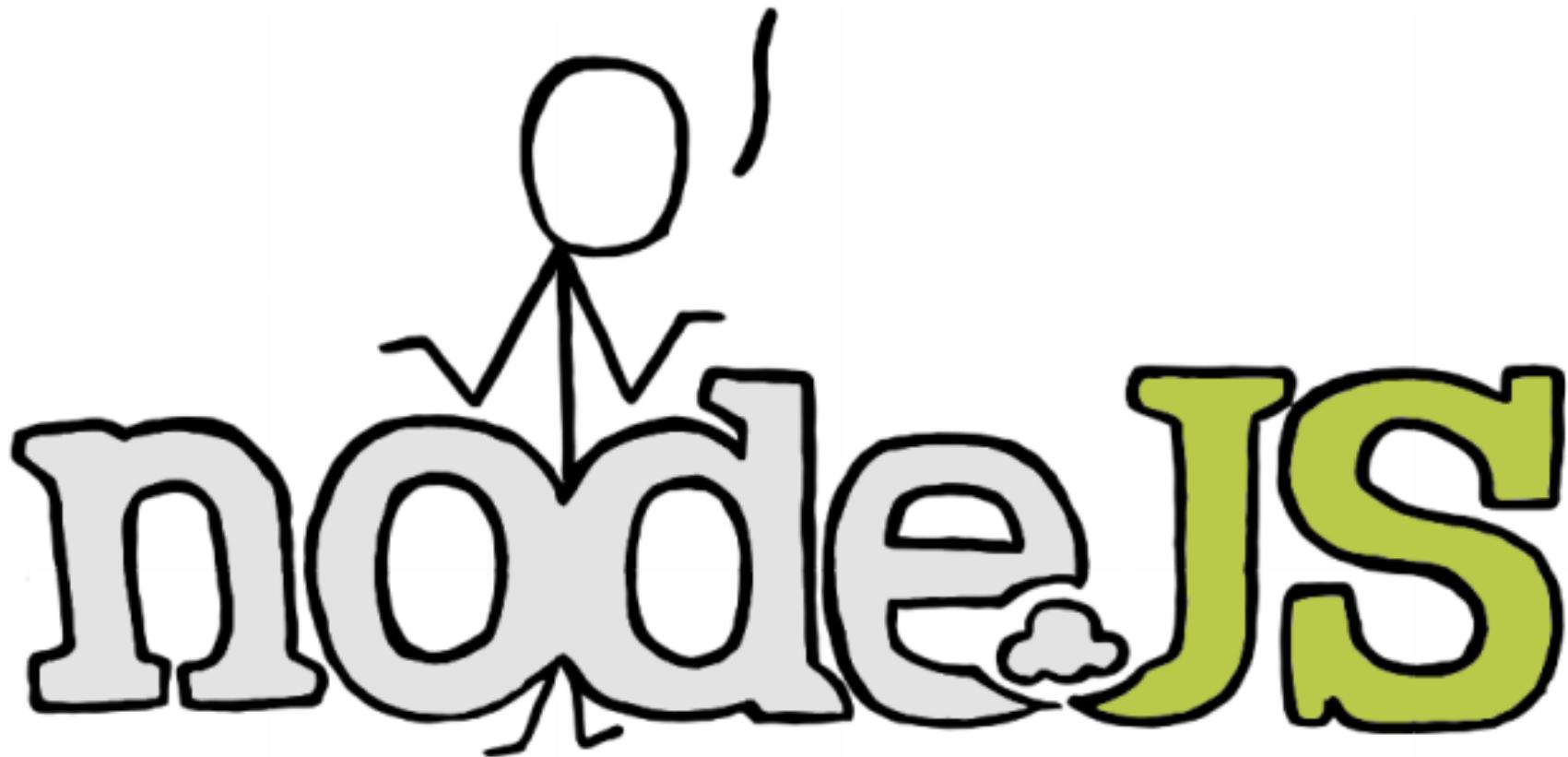
<http://www.mongodb.org/display/DOCS/Admin+UIs>

AdminUI para Node.js:

Smog: <https://github.com/wearefractal/smog> (web)

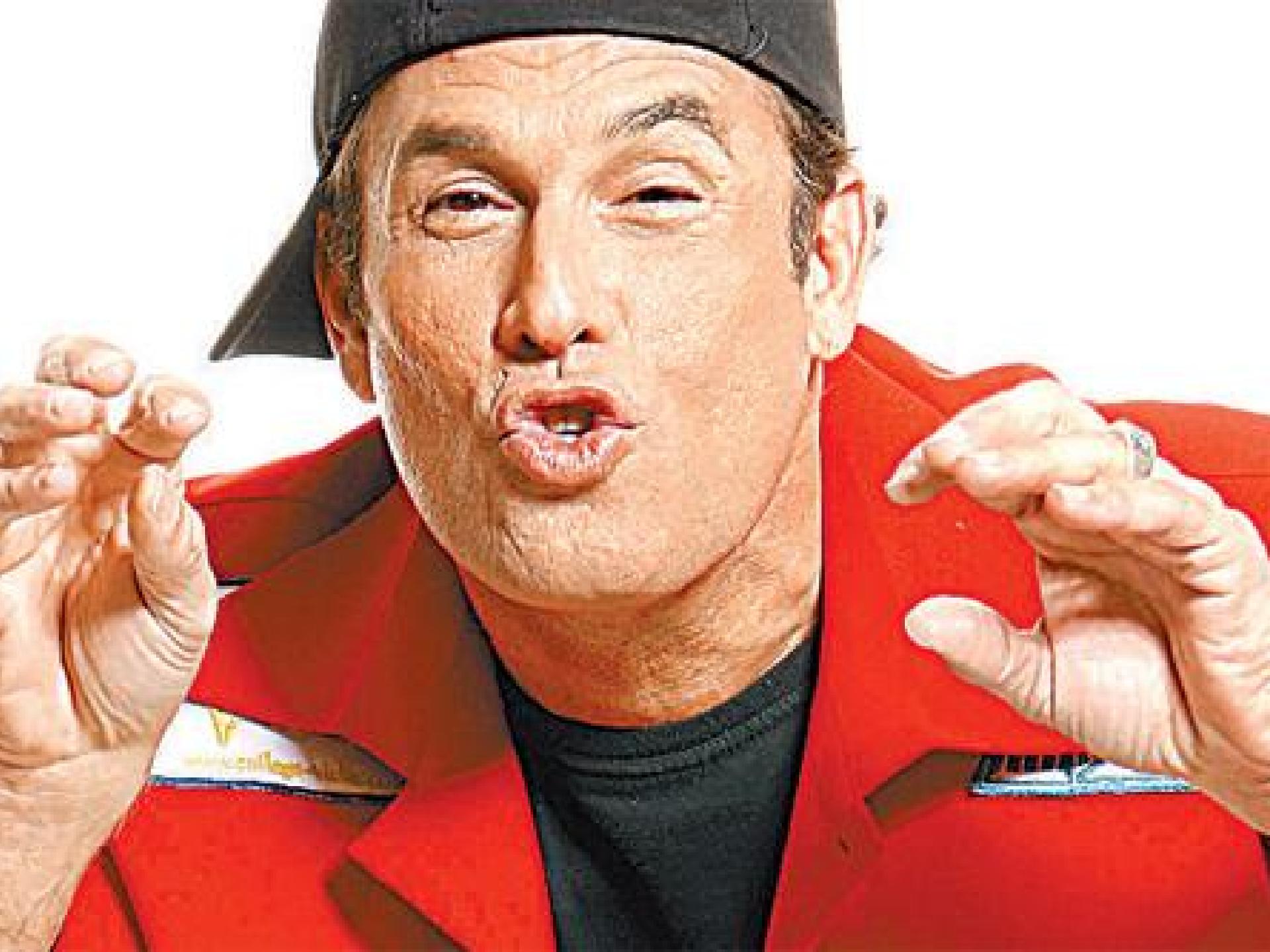
Robomongo: <http://www.robomongo.org/> (desktop)

NODE? JS? WTF?!





JavaScript™



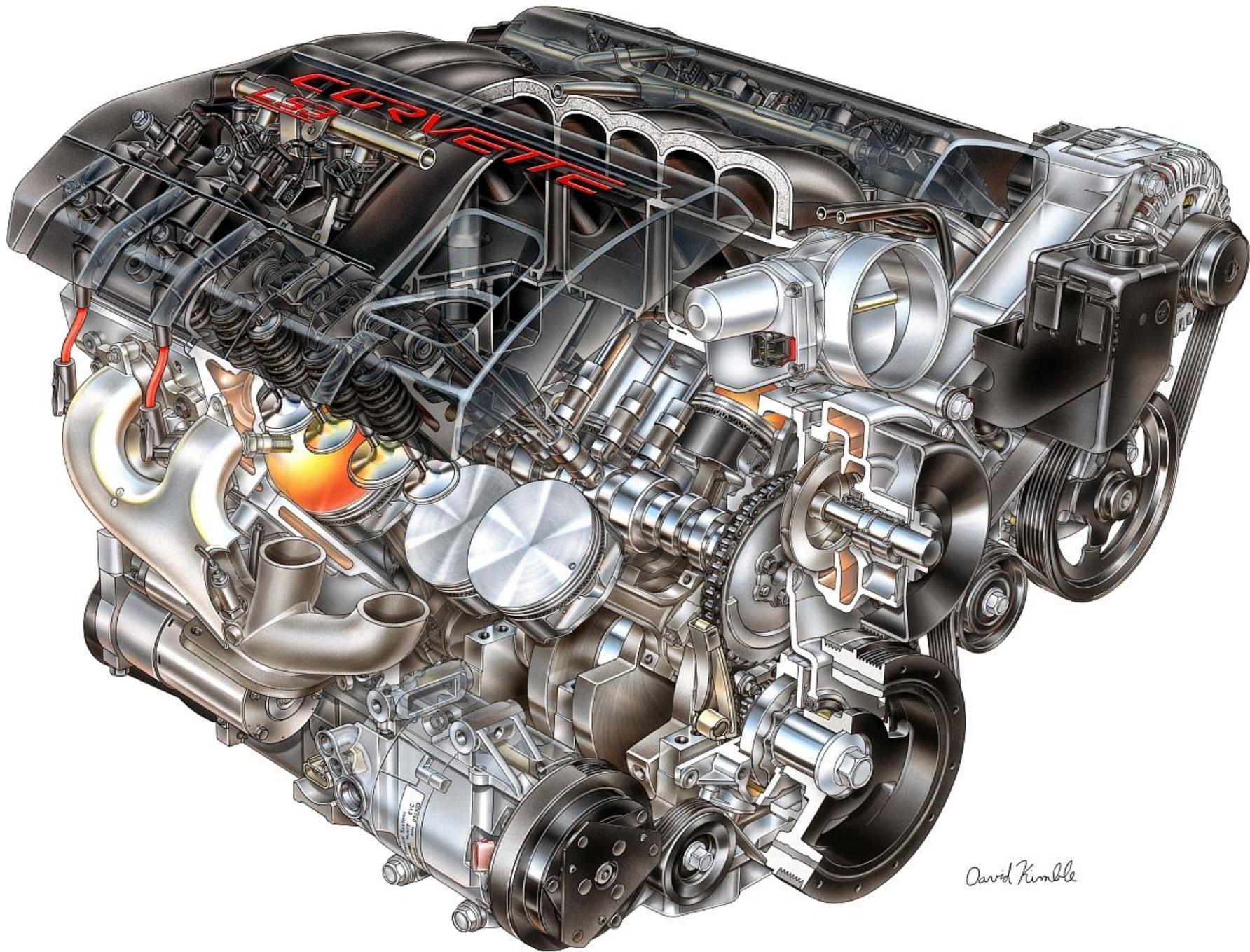
JS



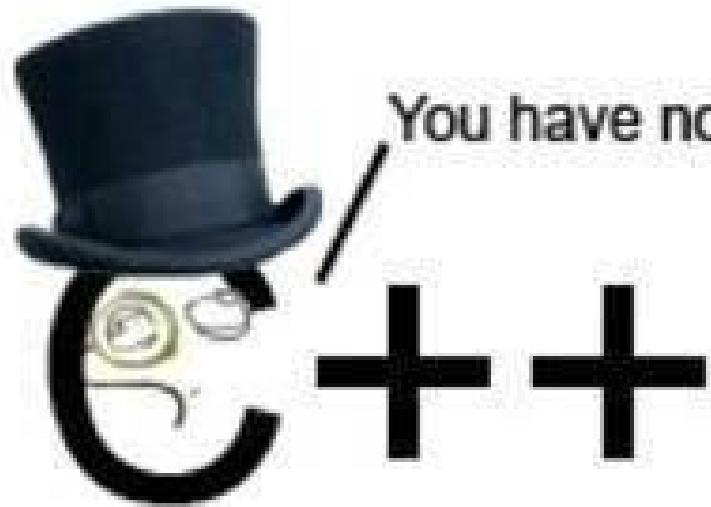


$$\begin{array}{r} \text{fish} \\ \times 2 \\ \hline \end{array}$$
$$\begin{array}{r} \text{clouds} \\ \times 5 \\ \hline \end{array}$$

**V8
C++
I/O Async
Event Drive**

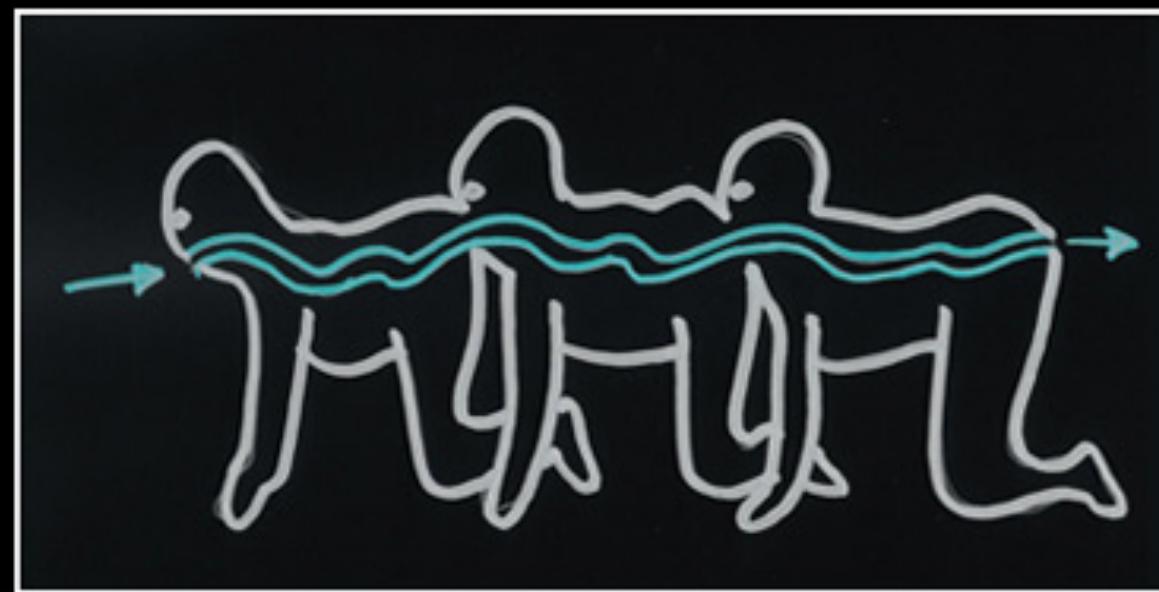
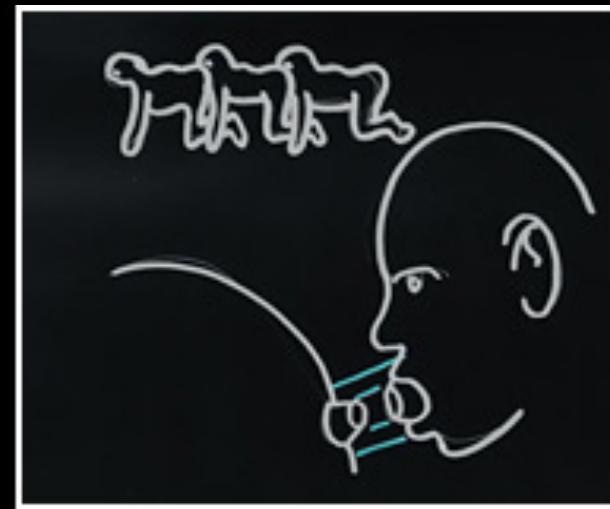
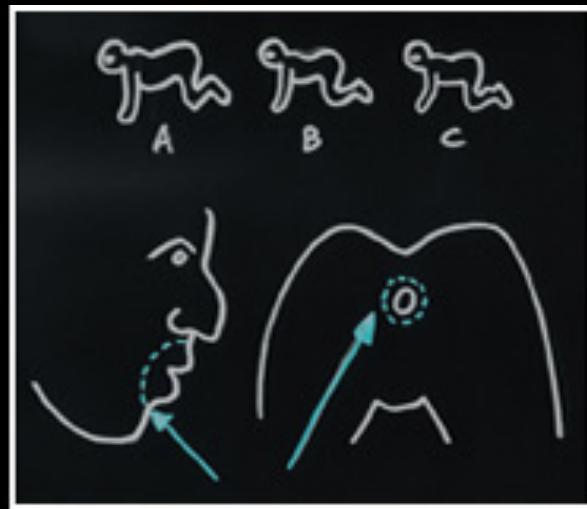
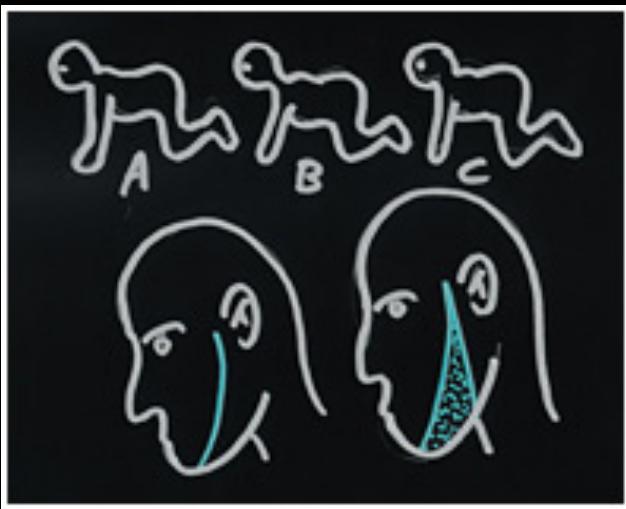


David Kimble



You have no class.





ASYNCHRONOUS JAVASCRIPT IS

ASYNCHRONOUS

The cost of I/O

L1-cache	3 cycles
L2-cache	14 cycles
RAM	250 cycles
Disk	41 000 000 cycles
Network	240 000 000 cycles

EVENTOS



PARA TODOS



eBay recently launched ql.io, a gateway for HTTP APIs, using Node.js as the runtime stack.

They were able to tune a regular developer-quality Ubuntu workstation to handle more than



120,000
active connections
per Node.js process

2,000

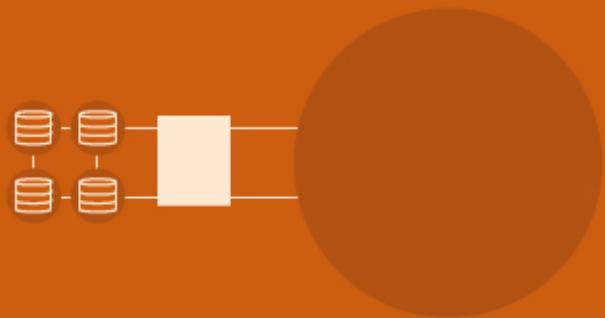
with each connection consuming
about 2k memory.



On the server side, LinkedIn's entire mobile software stack is completely built in Node.js.



They went from running 15 servers with
15 instances on each physical machine

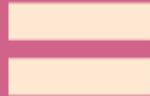


to just 4 instances that can handle 2x
the traffic.



Walmart re-engineered their mobile app to be powered by Node.js.

JAVASCRIPT



A RICH AND DYNAMIC
EXPERIENCE FOR
CUSTOMERS

They pushed all their JavaScript processing
to the server.

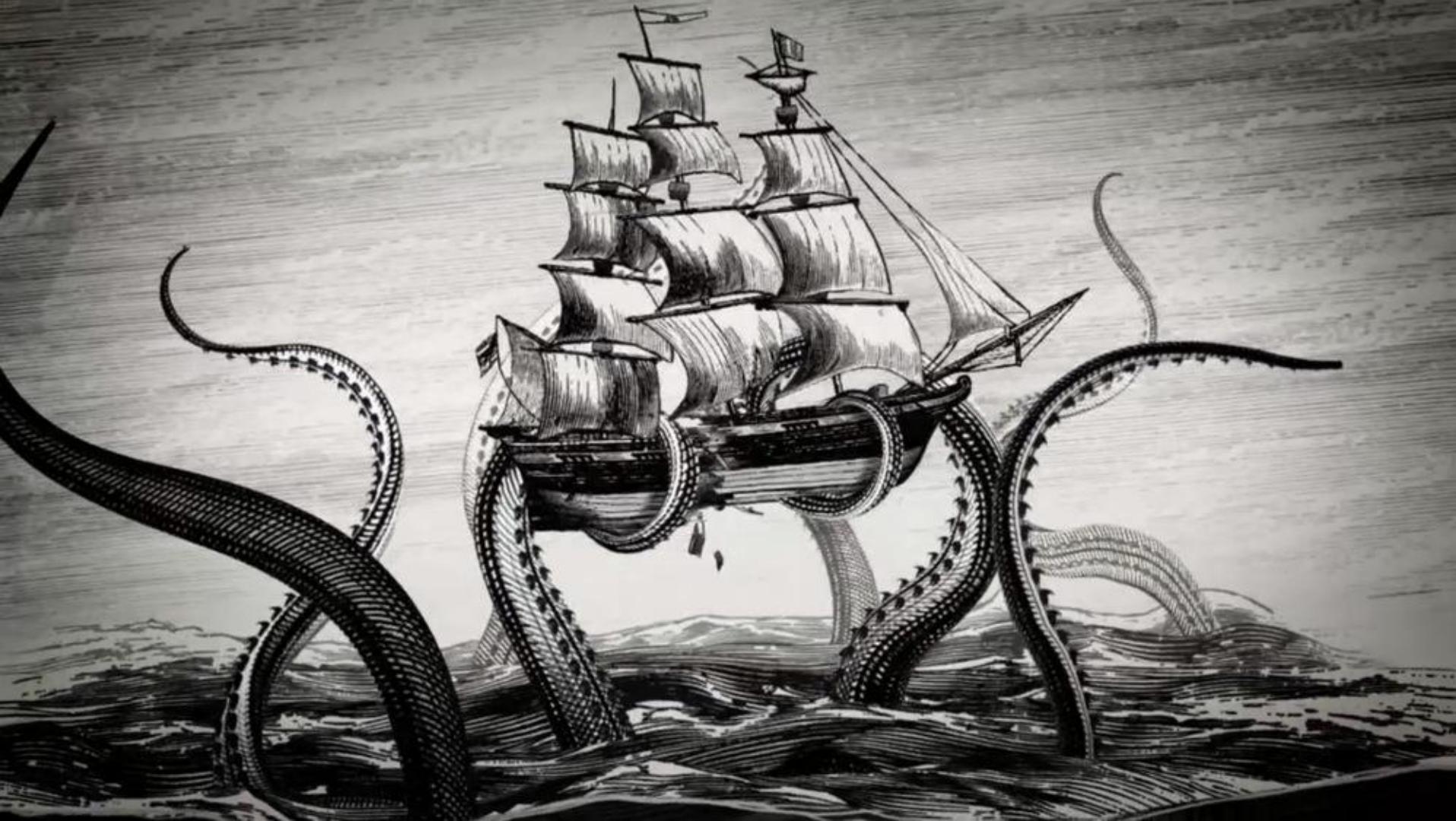
BIG BRANDS USE NODE

Big brands already use Node.js to power their business.





<https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/>



The kraken suite

Kraken is the main pillar of the framework, but the following modules can also be used independently.



Lusca

[Application security](#)



Makara

[Internationalization](#)



Adaro

[Dust.js Support](#)

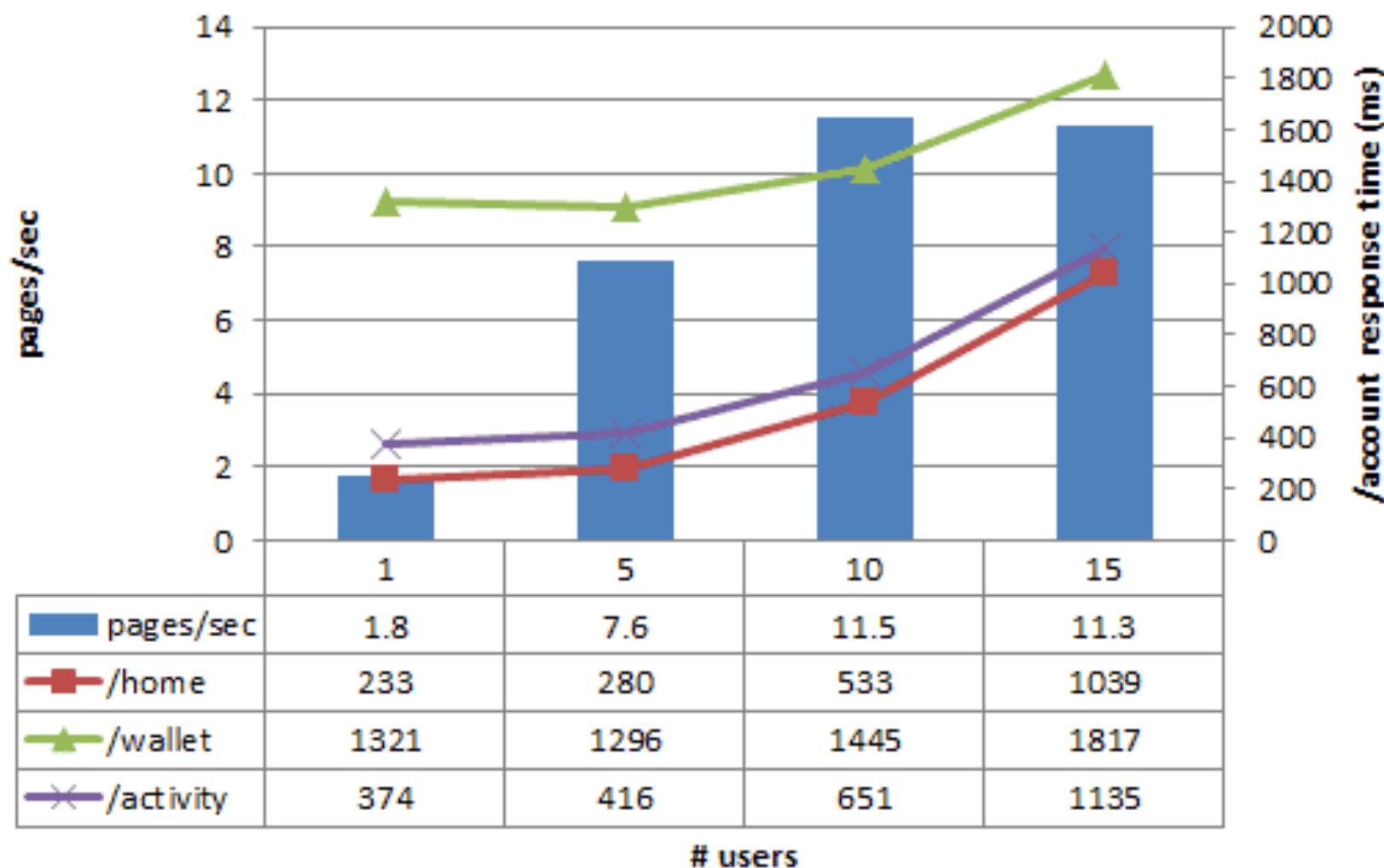


Kappa

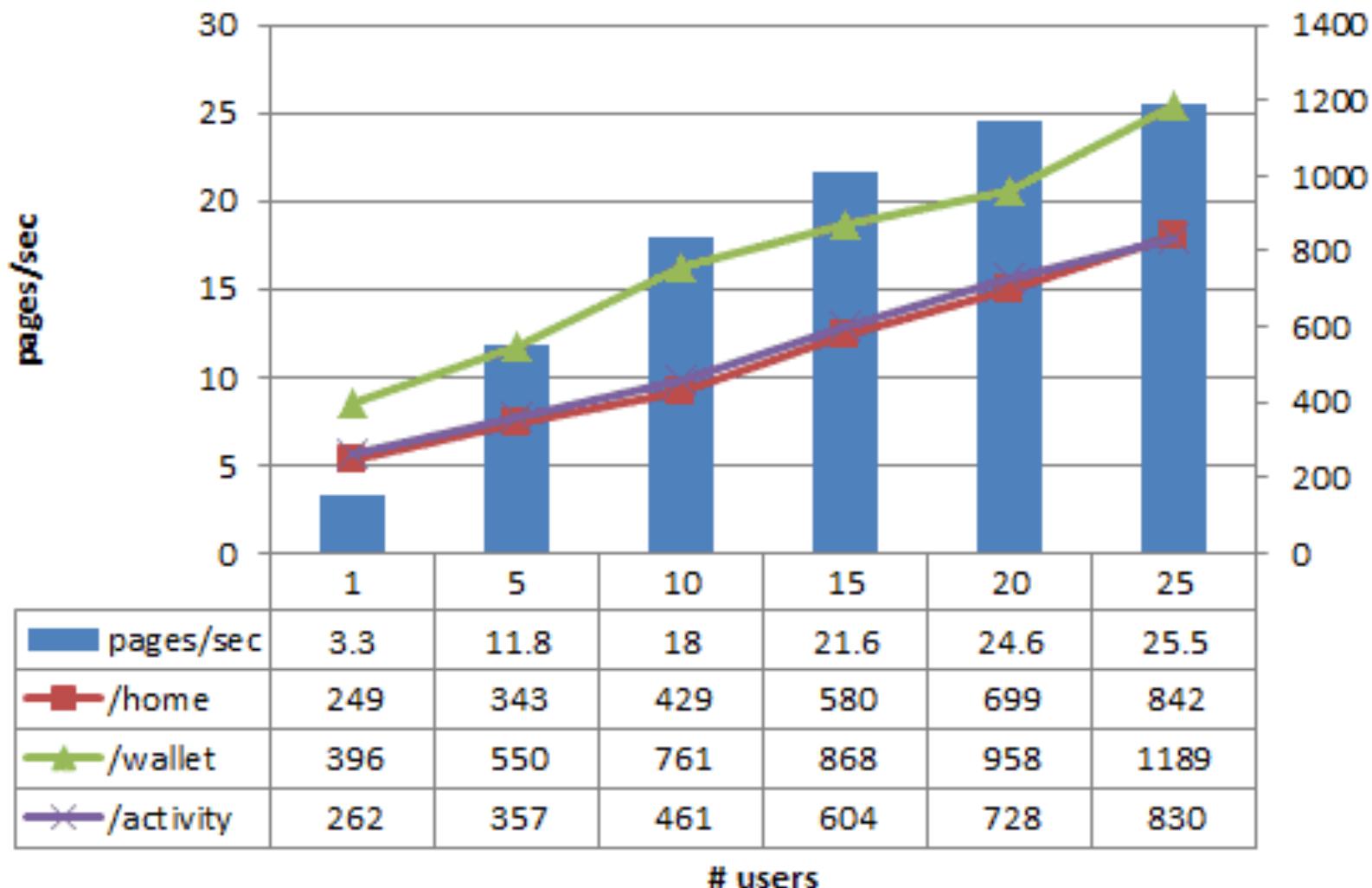
[NPM Proxy](#)

- Built almost twice as fast with fewer people
- Written in 33% fewer lines of code
- Constructed with 40% fewer files

Java application



Node.js application



- **Double the requests per second vs. the Java application.** This is even more interesting because our initial performance results were using a single core for the node.js application compared to five cores in Java. We expect to increase this divide further.
- **35% decrease in the average response time** for the same page. This resulted in the pages being served **200ms faster**— something users will definitely notice.

HELLO
World



hello-world.js

```
var http = require("http") ;  
  
http.createServer(function(request, response) {  
    response.writeHead(200,  
    {"Content-Type": "text/plain"}) ;  
    response.write("Hello World") ;  
    response.end() ;  
}).listen(3000) ;  
console.log('Entre em http://localhost:3000/') ;
```

hello-html.js

```
var http = require('http');
var fs = require('fs');
var index = fs.readFileSync('index.html');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end(index);
}).listen(3000);
```

Sync World

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

console.log("Vou ler", Date.now());
console.time("leitura");

var file = fs.readFileSync("file.zip");
console.log(file);

console.timeEnd("leitura");
console.log("Ja li", Date.now());
```

Async World

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

console.log("Vou ler", Date.now());
console.time("leitura");
// var file = fs.readFileSync("file.zip");

fs.readFile('drive.mp4', function(err, data) {
    console.log(data);
}) ;

console.timeEnd("leitura");
console.log("Ja li", Date.now());
```

npm

Gerenciador de pacotes para Node.js

Iniciar um package.json

```
npm init
```

Instalar um pacote

```
npm install
```

```
npm install --global //--g
```

```
npm install --save
```

```
npm install --save-dev
```

```
npm install --save-optional
```

<https://npmjs.org/>

Mongoose

<https://github.com/LearnBoost/mongoose>

Mongoose

```
var mongoose = require('mongoose');  
mongoose.connect('mongodb://localhost/_database');
```

Mongoose

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');

var db = mongoose.connection;
db.on('error', function(err) {
  console.log('Erro de conexão.', err);
});

db.on('open', function () {
  console.log('Conexão aberta.');
});

db.on('connected', function(err) {
  console.log('Conectado');
});

db.on('disconnected', function(err) {
  console.log('Desconectado');
});
```

Mongoose

Exemplo de melhores práticas: <https://gist.github.com/suisse/1058f338ee628d7c60e2>

Mongoose

```
var Cat = mongoose.model('Cat', { name: String });

var kitty = new Cat({ name: 'Osvaldinho' });
kitty.save(function (err, data) {
  if (err) {
    console.log('Erro: ', err);
  }
  console.log('meow', data);
});
```

Mongoose

```
var Schema = mongoose.Schema
, _schema = { name: { type: String, default: '' }
, description: { type: String, default: '' }
, alcohol: { type: Number, min: 0, default: '' }
, price: { type: Number, min: 0, default: '' }
, category: { type: String, default: '' }
, created_at: { type: Date, default: Date.now() }
}
, BeerSchema = new Schema(_schema)
, Beer = mongoose.model('Beer', BeerSchema)
;
```

Mongoose - create

```
var dados = {  
    name: 'Heineken',  
    description: 'Até q eh boazinha',  
    alcohol: 5.5,  
    price: 3.5,  
    category: 'lager'  
}  
  
var model = new Beer(dados);  
model.save(function (err, data) {  
    if (err) {  
        console.log('Erro: ', err);  
    }  
    else{  
        console.log('Cerveja Inserida', data);  
    }  
}) ;
```

Mongoose - retrieve

```
var query = {};  
  
Beer.find(query, function (err, data) {  
  if(err) {  
    console.log(err);  
  } else {  
    console.log(data)  
  }  
})
```

Mongoose - update

```
var query = {name: /heineken/i}  
, mod = {  
    name: 'Brahma'  
, alcohol: 4  
, price: 6  
}  
, optional = {  
    upsert: false  
, multi: true  
}  
;  
;
```

Mongoose - update

```
Beer.update(query, mod, function (err, data) {  
  if (err) {  
    console.log('Erro: ', err);  
  }  
  else{  
    console.log('Cervejas atualizadas com sucesso: ', data);  
  }  
  process.exit(0);  
}) ;
```

Mongoose - delete

```
var query = {name: /brahma/i};

Beer.remove(query, function (err, data) {
  if (err) {
    console.log('Erro: ', err);
  }
  else{
    console.log('Cerveja deletada com sucesso', data.result);
  }
});
```

Rotas

Rotas manual

```
var url = request.url;
switch(url) {
  case '/create':
    Beer.create(request, response);
    break;
  case '/retrieve':
    Beer.retrieve(request, response);
    break;
  case '/update':
    Beer.update(request, response);
    break;
  case '/delete':
    Beer.delete(request, response);
    break;
}
```

Express

<http://expressjs.com/>
npm install -g express-generator

Express

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

app.get('/hello', function(req, res) {
    var body = 'Hello World';
    res.setHeader('Content-Type', 'text/plain');
    res.setHeader('Content-Length', body.length);
    res.end(body);
});

app.listen(3000);
```



ANGULARJS

by Google[®]

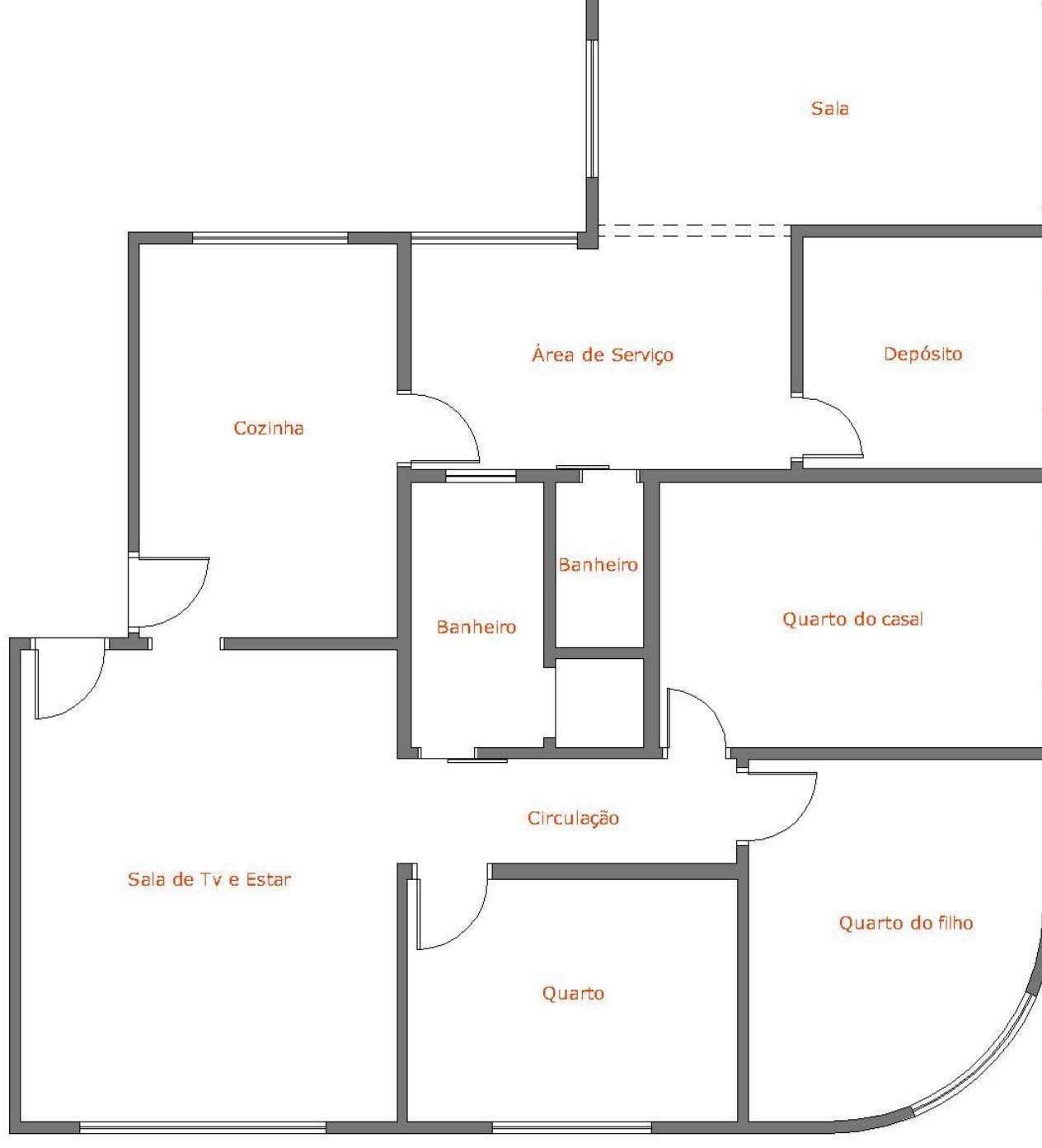
Ementa

1. Introdução
2. Templates
3. Data Bindings
4. Ajax
5. Injeção de Dependência
6. Rotas
7. Diretivas
8. Serviços
9. Projeto final

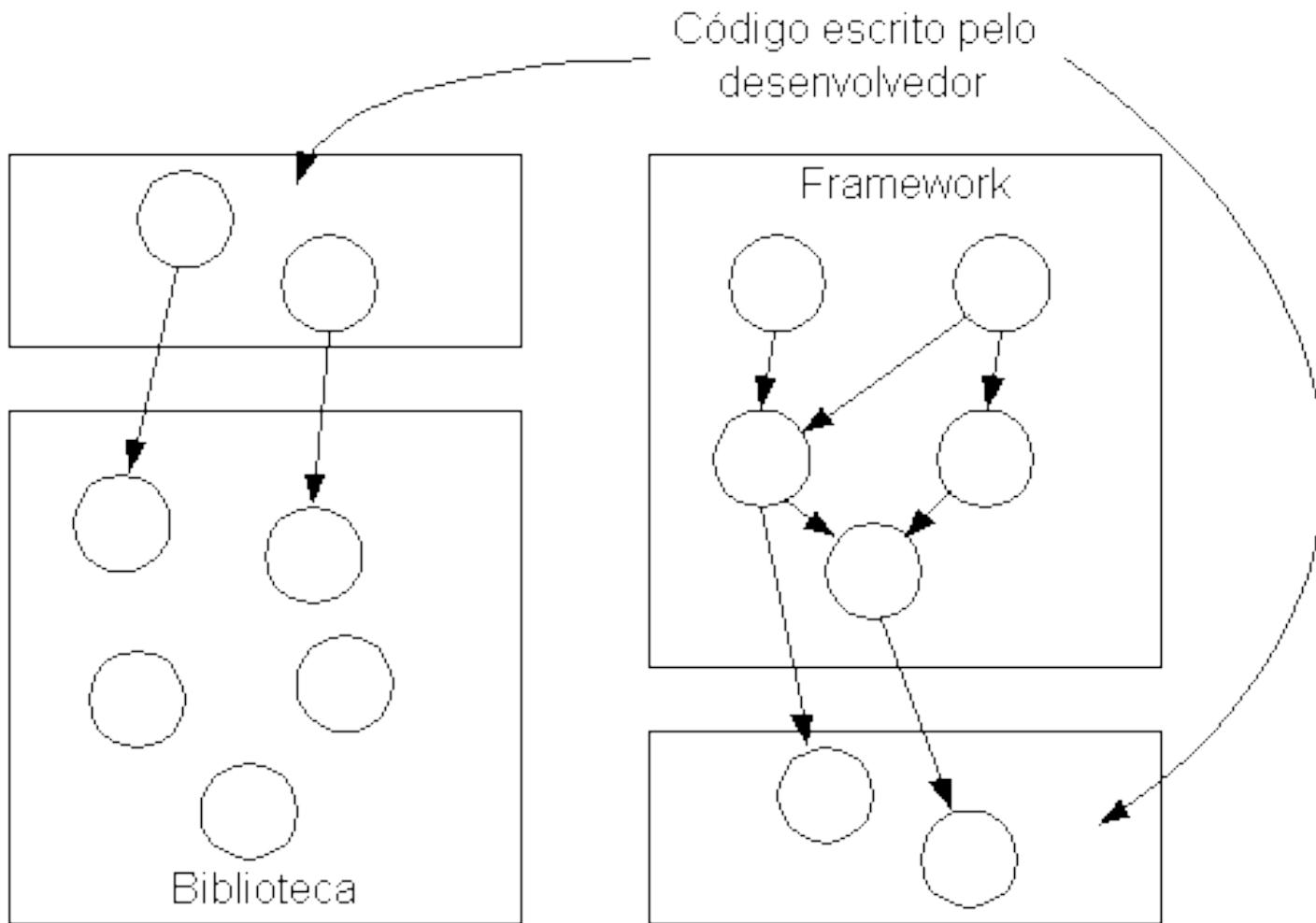
Introdução

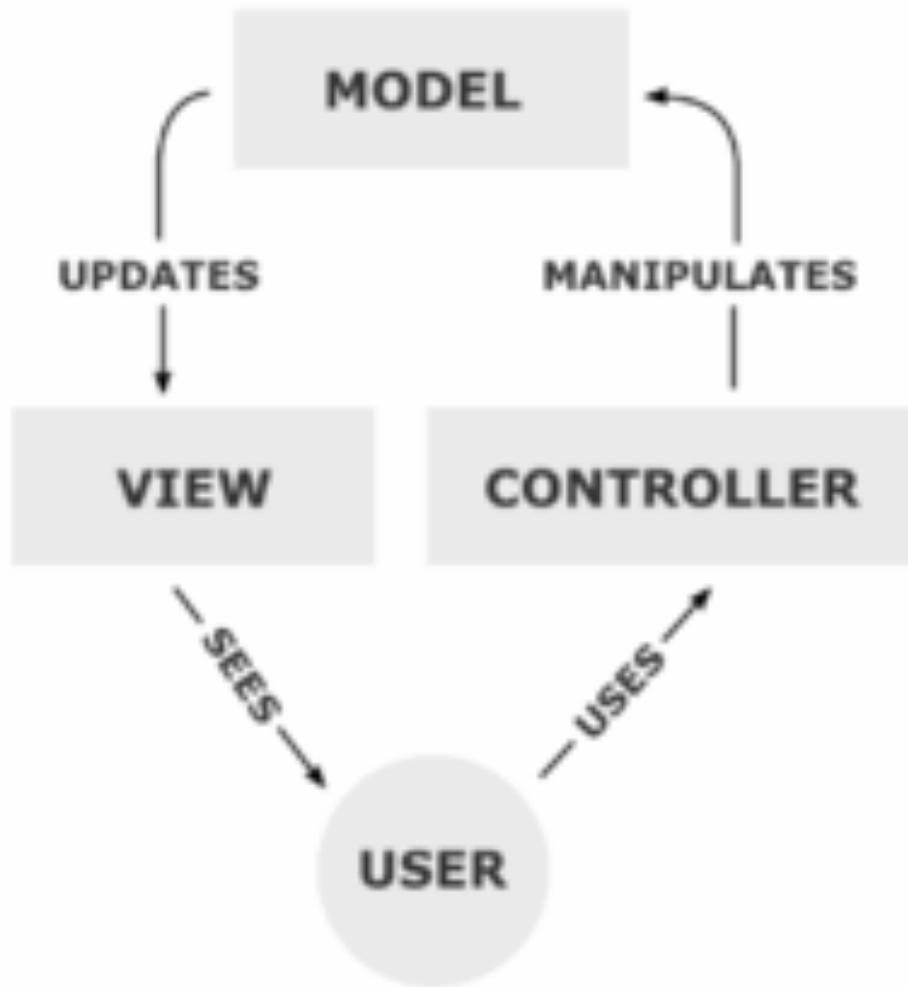
AngularJs é um framework MVC de Javascript mantido pelo Google.

Ele aumenta o poder do seu HTML, criando um mecanismo rápido e fácil para criação de aplicativos.



PLANTA BAIXA





Startup

Possuímos 3 modos diferentes de iniciar uma aplicação com Angularjs:

- Automaticamente via ng-app
- Passando o nome do módulo
- Via angular.bootstrap

Startup

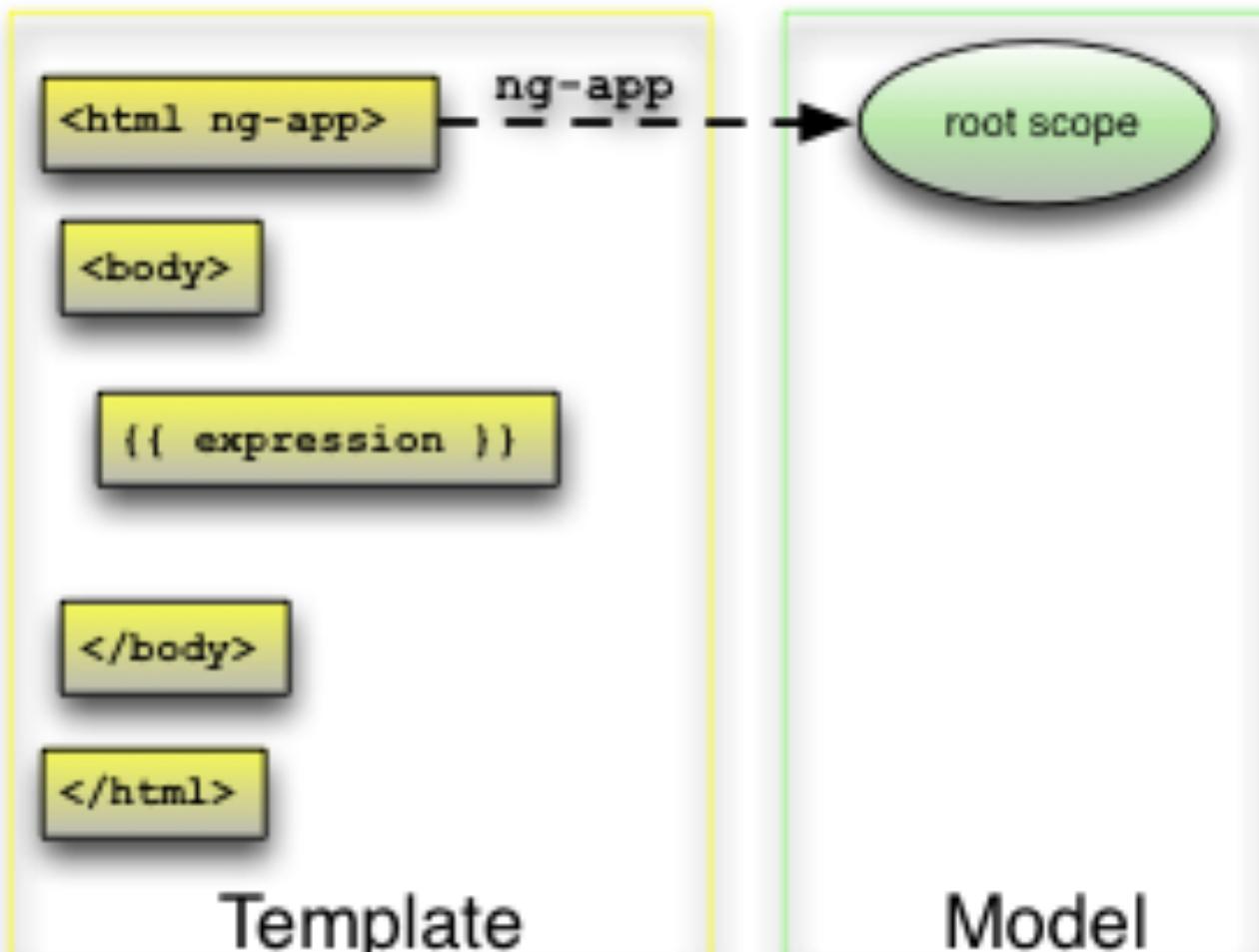
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Workshop Be MEAN</title>
</head>
<!-- Iniciando minha aplicação com ng-app -->
<body data-ng-app>
  {{ 2 + 2}}
  <script src="angular.min.js"></script>
</body>
</html>
```

Startup

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Workshop Be MEAN</title>
</head>
<body data-ng-app="workshopBeMEAN">
  2 + 2 = {{ 2 + 2}}
  <script src="angular.min.js"></script>
  <script>
    angular.module('workshopBeMEAN', []);
  </script>
</body>
</html>
```

Startup

```
<!doctype html>
<html>
<body>
{{ 2 + 2 }}
<script src="http://code.angularjs.org/angular.js"></script>
<script>
angular.element(document).ready(function() {
  angular.bootstrap(document);
});
</script>
</body>
</html>
```



Implicit Scope Declaration



Templates

São os modelos do nosso documento, onde estará estruturada da nossa informação e é onde o Model irá atualizar as informações.



32

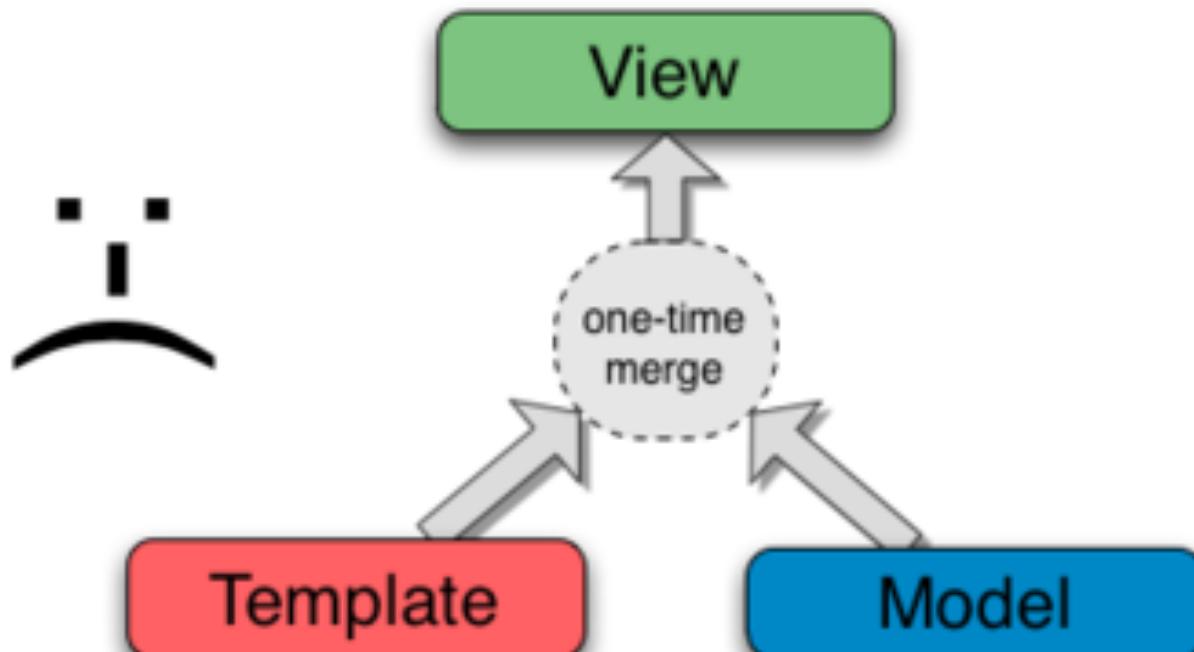
Data bindings

Data bindings servem para simplificar a manipulação do DOM, retirando esse fardo do programador. São ligados os elementos da interface gráfica com o model da aplicação. A maioria dos frameworks usam o padrão Observer.

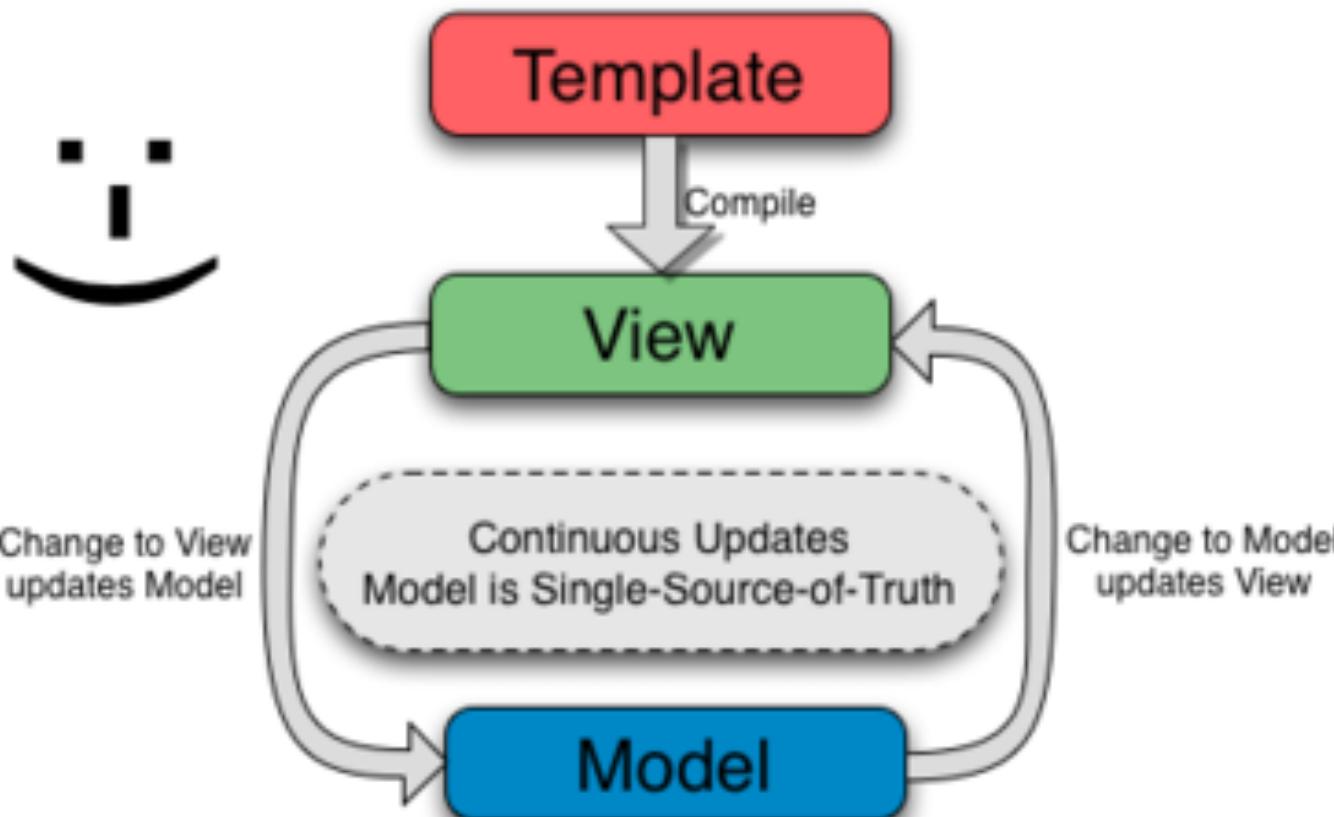
Data bindings

<https://docs.google.com/forms/d/1sUk83QBFm0jX1flF1lOsmlsSLBTriGg8glk2b2fHbE8/viewform>

One-Way Data Binding



Two-Way Data Binding





Filters

Os filtros são usados para a formatação de dados exibidos para o usuário.

Sintaxe geral:

```
{{ expression [| filter_name[:parameter_value] ... ] }}
```

Para criarmos um filtro precisamos de 3 coisas:

- Criar um módulo
- Criar um filtro e sua função
- Registrar o módulo na aplicação

Filters - criando módulo

```
angular.module('yourModuleName', [])
```

Filters - criando filtro e sua função

```
angular.module('yourModuleName', [])
.filter('yourFilterName', function () {
    return function (text) {
        return;
    };
});
```

Filters - registrar o módulo

```
angular.module('yourAppName', ['yourModuleName']);
```

Filters - exemplo

```
.filter('reverseName', function () {  
    return function (text) {  
        if(text)  
            return text.split("").reverse().join("");  
    };  
});
```

Filters - exemplo

```
.filter('truncate', function () {
  return function (text, length, end) {
    if(text){
      if (isNaN(length))
        length = 10;
      if (end === undefined)
        end = "...";
      if (text.length <= length || text.length - end.length <= length) {
        return text;
      }
      else {
        return String(text).substring(0, length) + end;
      }
    }
  };
});
```

Filters - exemplo

```
<div ng-app="myApp">
  <h2>Truncate Filter Example</h2>
  <div>
    <input type="text" ng-model="myText" placeholder="add your text here" />
  </div>
  <div>
    <h3>Output:</h3>
    <p>{{myText|truncate}}</p>
    <p>{{myText|truncate:5}}</p>
    <p>{{myText|truncate:25:" ->"}}</p>
  </div>
</div>
```

Controllers



Rotas

Rotas

```
angular.module('agenda', []).config( ['$routeProvider',
  function($routeProvider) {
    $routeProvider
      .when('/contatos',
        {templateUrl: 'partials/contact-list.
          html',
         controller: PhoneListCtrl})
      .when('/contatos/:contactId',
        {templateUrl:
          'partials/contact-detail.html',
         controller: PhoneDetailCtrl})
      .otherwise({redirectTo: '/contatos'});
  }
]); //fim routerProvider
```

Rotas - when

Adiciona uma nova definição de rota para o serviço \$route

template: passa uma string html para ser usada na view

templateUrl: passa o endereço de um template HTML

Rotas - otherwise

Definição da rota que será usada quando a rota requisitada não existir.

UPP
TILL 800kr Värdeklarning på
vara en överraskning hänta SPA!

Se baksidan för detaljer

NY FRISK DOFT

AJAX

Color

Water Lily



\$http

Nós vamos usar o serviço \$http em nosso controller para fazer uma solicitação HTTP para o servidor buscando os dados.

\$http

```
function ProductsListCtrl($scope, $http) {  
  $http.get('products/products.json') .  
success(function(data) {  
  $scope.products = data;  
} );  
  
$scope.orderProp = 'created_at';  
}  
}
```

\$http

```
function ProductsListCtrl($scope, $http) {  
  $http.get('products/products.json') .  
success(function(data) {  
  $scope.products = data;  
} );  
  
$scope.orderProp = 'created_at';  
}  
}
```

Injeção de Dependência

É um padrão de desenvolvimento de software onde as dependências entre os módulos são injetadas e não programadas, mantendo o baixo nível de acoplamento.

Injeção de Dependência

```
function PhoneListCtrl($scope, $http) { . . . }
```

Minificado

```
PhoneListCtrl.$inject = ['$scope', '$http'];
```

```
var PhoneListCtrl = ['$scope', '$http',
function($scope, $http) {
/* constructor body */
}];
```

HOSPITAL DAS PANELAS

CONsertos e vendas de panelas de pressão

3552-2817

Na compra de uma panela de pressão,
não importa qual o seu

SEXO
ganhe
GRÁTIS

uma maravilhosa caneca Mimo

AQUI
no Hospital das Panelas!

TATA
5 / 9658-2400



Serviços

Para manipularmos as informações internamente no AngularJs precisamos utilizar/criar serviços.

Os serviços são gerenciados pelo subsistema de Injeção de Dependência.

<http://docs.angularjs.org/guide/di>

service

```
app.service('nameReverseService',  
  function()  {  
    this.reverse = function(name)  
    {  
      return name.split("")  
        .reverse()  
        .join("");  
    } ;  
} ) ;
```

factory

```
app.factory('nameReverseFactory',  
  function()  {  
    return {  
      reverse : function(name)  {  
        return name.split("")  
          .reverse()  
          .join("");  
      }  
    }  
  }) ; //fim factory
```

Injectando no controller

```
app.controller( 'AppCtrl',
  function AppCtrl($scope,
                  nameReverseService,
                  nameReverseFactory)
  {
    $scope.name = 'Suissa';
    $scope.reverseNameByService = function() {
      $scope.name = nameReverseService.reverse($scope.
name);  };
    $scope.reverseNameByFactory = function() {
      $scope.name = nameReverseFactory.reverse($scope.
name);
    };
  });
});
```

Chamada no template

```
<button ng-click="reverseNameByService()">  
    Reverse Name via Service  
</button>
```

```
<button ng-click="reverseNameByFactory()">  
    Reverse Name via Factory  
</button>
```

Exemplo

<http://jsfiddle.net/suisse/P3M9d/1/>

Diferenças entre Service, Factory e Provider

Service

```
var myApp = angular.module('myApp', []);
//service style, probably the simplest one
myApp.service('helloWorldFromService', function() {
  this.sayHello = function() {
    return "Hello, World!"
  };
}) ;
```

Factory

```
//factory style, more involved but more sophisticated
myApp.factory('helloWorldFromFactory', function() {
  return {
    sayHello: function() {
      return "Hello, World!"
    }
  };
}) ;
```

Factory

```
//provider style, full blown, configurable version
myApp.provider('helloWorld', function() {
    // In the provider function, you cannot inject any
    // service or factory. This can only be done at the
    // "$get" method.

    this.name = 'Default';

    this.$get = function() {
        var name = this.name;
        return {
            sayHello: function() {
                return "Hello, " + name + "!"
            }
        }
    };
}

this.setName = function(name) {
    this.name = name;
};

}));
```

Configurando e Usando

```
/hey, we can configure a provider!
myApp.config(function(helloWorldProvider) {
  helloWorldProvider.setName('World');
}) ;

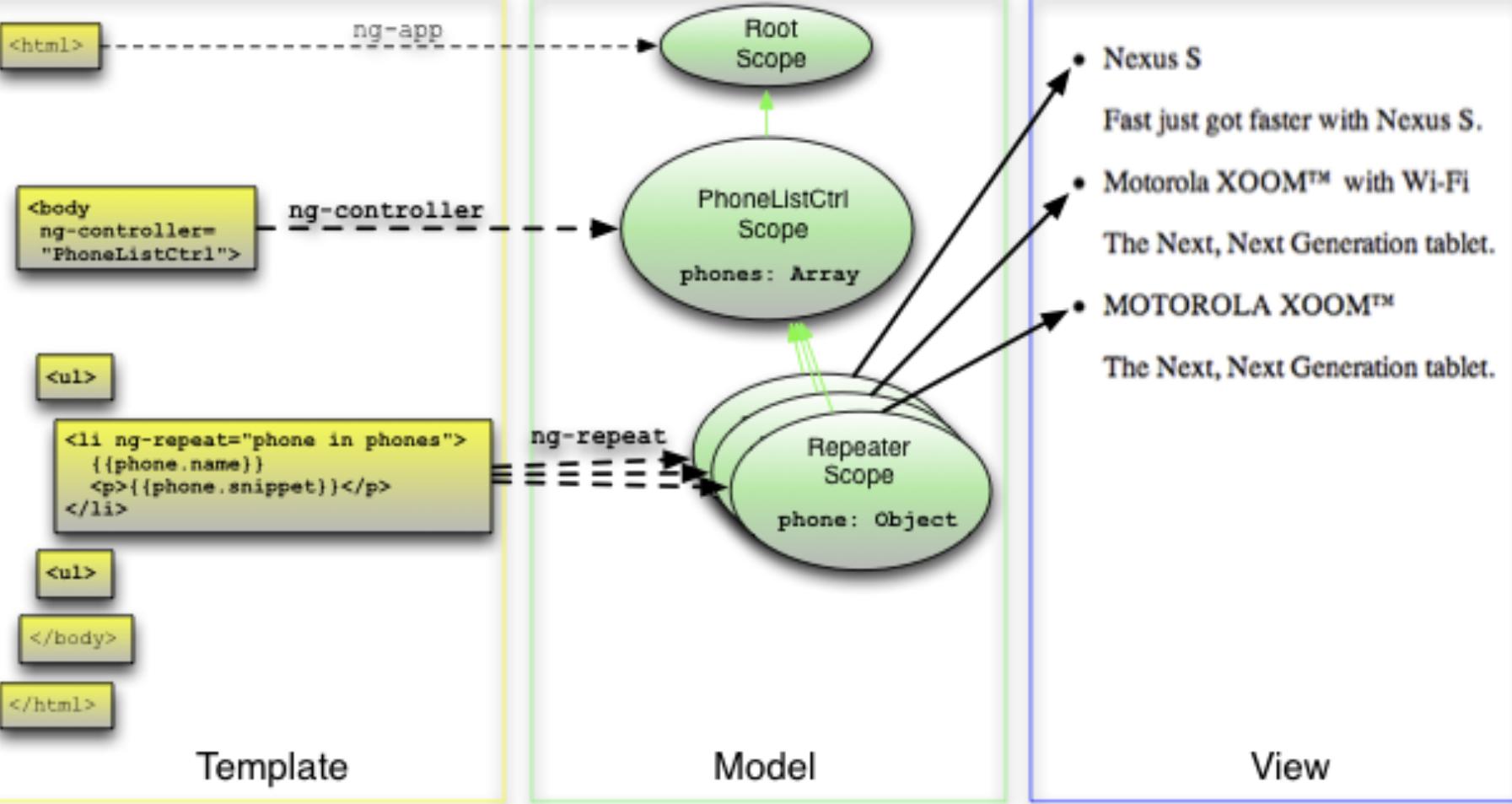
function MyCtrl($scope, helloWorld, helloWorldFromFactory, helloWorldFromService) {
  $scope.hellos = [
    helloWorld.sayHello(),
    helloWorldFromFactory.sayHello(),
    helloWorldFromService.sayHello()];
}
```

Directives

As directives são uma forma para criar elementos HTML personalizados nos seus templates, podendo reutilizá-los em diversos templates. Porém antes precisamos conhecer os seus tipos de restrição

Usando repetição - ng-repeat

```
<body ng-controller="ProductsListCtrl">  
  <ul>  
    <li ng-repeat="product in products">  
      {{product.name}}  
      <p>{{product.description}}</p>  
    </li>  
  </ul>  
</body>
```



-----> Implicit Scope Declaration

→ Scope Inheritance

↔ Model / View Data-binding

Directives - restricts

restrict: 'E' – um elemento DOM com um certo nome, ex.:

```
<my-directive></my-directive>
```

restrict: 'A' - um elemento DOM contendo um certo atributo, ex.:

```
<div my-directive="exp"></div>
```

restrict: 'C' - chamada através de uma classe, ex.

```
<div class="my-directive: exp"></div>
```

restrict: 'M' - chamada através de um comentário: <!-- directive: my-directive exp -->

Directives – Data Bind

“@” - Local scope property

Utilizando o “@” passa o atributo como string e fique disponível dentro da directiva podendo ser manipulada caso necessário dentro do escopo.

“=” - Bi-directional binding

Utilizando o “=” apenas faz referência ao ngModel dentro da directiva.

“&” - Parent execution binding

Utilizando o “&” pode passar uma função como referência para ser executada dentro da directiva.

Directives – Data Bind Exemplo

```
<input type="text" ng-model="email" />

<!-- Nossa Directive -->
<div scope-example ng-model="email" enviar="enviarMsg(email)" para="contato@gmail.com" />

scope : {
    ngModel: '=' // Fazendo referência ao atributo dentro do escopo da directive.
    enviar: '&' // Passando a função como referência para ser executada.
    para: '@' // Apenas passando como string para o escopo da directive
}
```

Exemplo: <http://jsfiddle.net/hWC46/>

Directives

```
var app = angular.module('myApp', []);

app.controller('directiveController', ['$scope', function($scope) {
    $scope.enviarMsg = function(mail) {
        alert(mail);
    };
}]);
```

Directives

```
app.directive('directivebinds', function() {  
  return {  
    restrict: 'A',  
    scope: {  
      ngModel: '=',  
      enviar: '&',  
      para: '@'  
    },  
    template: '<div>Seu e-mail é: {{ngModel}}</div><br><button ng-click="enviar(ngModel);">Enviar</button><br><br>Email padrão é {{para}} que tem  
{{para.length}} caracteres.'  
  }  
});
```

Directives

```
angular.module('ng').directive('testElem', function () {
  return {
    restrict: 'A',
    template: '<div class="mydirectiveclass"><h1>hello...</h1><p ng-repeat="obj in arr">{{obj}}</p></div>',
    //templateUrl: '/partials/template.html',
    link: function (scope, iterStartElement, attr) {
      $(".mydirectiveclass").css({'background-color' : 'yellow'});
      scope.arr = ["angularjs", "eh", "mto", "bao"];
    }
  };
});
```

Directives

Primeira linha declara o nome da diretriz "testElem". Veja como E é capital. Isso significa que o elemento ou atributo nome será "test-elem".

restrict: você pode combiná-los também, por exemplo, "EA".

template: uma string HTML que substituirá a diretriz.

TemplateUrl: opcionalmente, você pode ter o template HTML dentro de outro arquivo, principalmente se for longo.

link: a função de ligação. Depois que o modelo foi carregado, esta função é chamada para estabelecer o escopo e efeitos de última hora, como a animação jQuery ou outra lógica.

Directives - Transclude

Isso não é uma palavra que você vai encontrar em um dicionário :p
Tranclude faz com que o conteúdo do elemento da directiva seja empurrado para o div que tem ng-transclude. É apenas uma função de movimento.

```
angular.module('ng').directive('testElemTransclude', function () {  
  return {  
    restrict: 'EA',  
    transclude: true,  
    scope: 'isolate',  
    template: '<h3>heading 3</h3><p>preface... blah blah</p><div ng-  
transclude></div>',  
  };  
});
```

mais infos: <http://blog.omkarpatil.com/2012/11/transclude-in-angularjs.html>

Material interessante

<http://trochette.github.io/Angular-Design-Patterns-Best-Practices/>

<http://www.yearofmoo.com/2012/11/angularjs-and-seo.html>

<http://www.yearofmoo.com/2012/10/more-angularjs-magic-to-supercharge-your-webapp.html>

<http://www.yearofmoo.com/2013/04/animation-in-angularjs.html>

<http://www.yearofmoo.com/2013/01/full-spectrum-testing-with-angularjs-and-testacular.html>

Grupos Facebook

[AngularJs Brasil](#)

[NodeJs Brasil](#)

[MongoDb Brasil](#)

[NoSQL Brasil](#)

[Javascript Brasil](#)

Projeto Final

Projeto Final

Criar um CRUD para cervejarias re-utilizando nosso código em aula, porém refatorando todo o código para seguir o styleguide do John Papa, menos a parte de `Controller As`.

Utilizar o POPULATE do Mongoose para ligar a Cervejaria com suas Cervejas.

Criar uma diretiva para mostrar a cerveja com seus dados e uma imagem fictícia.

Utilizar o RESOLVE no list para mostrar as Cervejas e Cervejarias.

Refatorar a parte do Angular para seguir a modularização mostrada na primeira aula com o angular-seed.

Criar sistema de mensagem de erros.

Model da cervejaria:

- nome
- cidade
- estado
- pais
- cervejas (ARRAY)

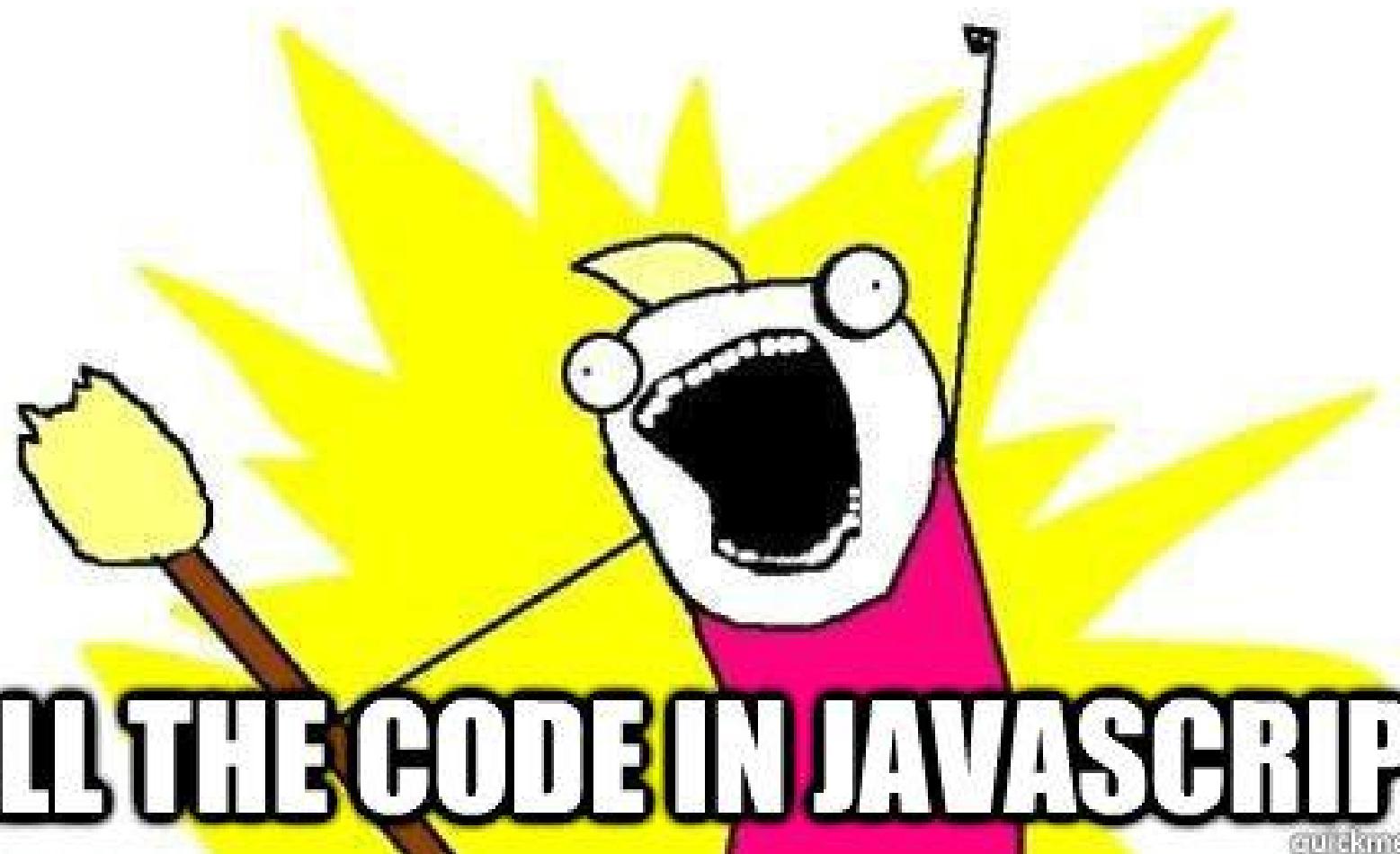
###PLUS:

Utilizar EVENTOS no Node ou Angular para comunicação.

Utilizar Socket para integrar Front e Back.

Criar uma interface elegante utilizando CSS, pode usar um framework para isso como o Foundation.

WRITE



ALL THE CODE IN JAVASCRIPT