Node Js. Udemy Course

Release : 10.x (25-08-18)

NODE ES JAVASCRIPT DEL LADO del SERVIDOR

#what MAKES node GREAT is THAT BOTH JS running on the BROWSER AND JS RUNNING ON THE SERVER ARE PROCESSED BY THE SAME ENGINE.

V8 Engine, COMPILES TO MACHINE CODE AND it is written on **C++**

TO open the Node CLI → $> node

> console.log("hi");

- **WINDOW** → it is a global object
  - → STORES EVERYTHING we HAVE ACCESS TO.
  - → MAKES REFERENCE TO THE BROWSER
- **global** → STORES MOST THINGS AS WINDOW
  - → EXCLUSIVE OF NODE.
- **document** → RESIDES INSIDE window
  - → WHAT USERS SEES
- **PROCESS** → info, methods, ...
  - → process.exit(0);
  - → INFO ABOUT WHAT NODE PROCESS ARE BEING EXECUTED

BROWSER

Node

TO GENERALIZE

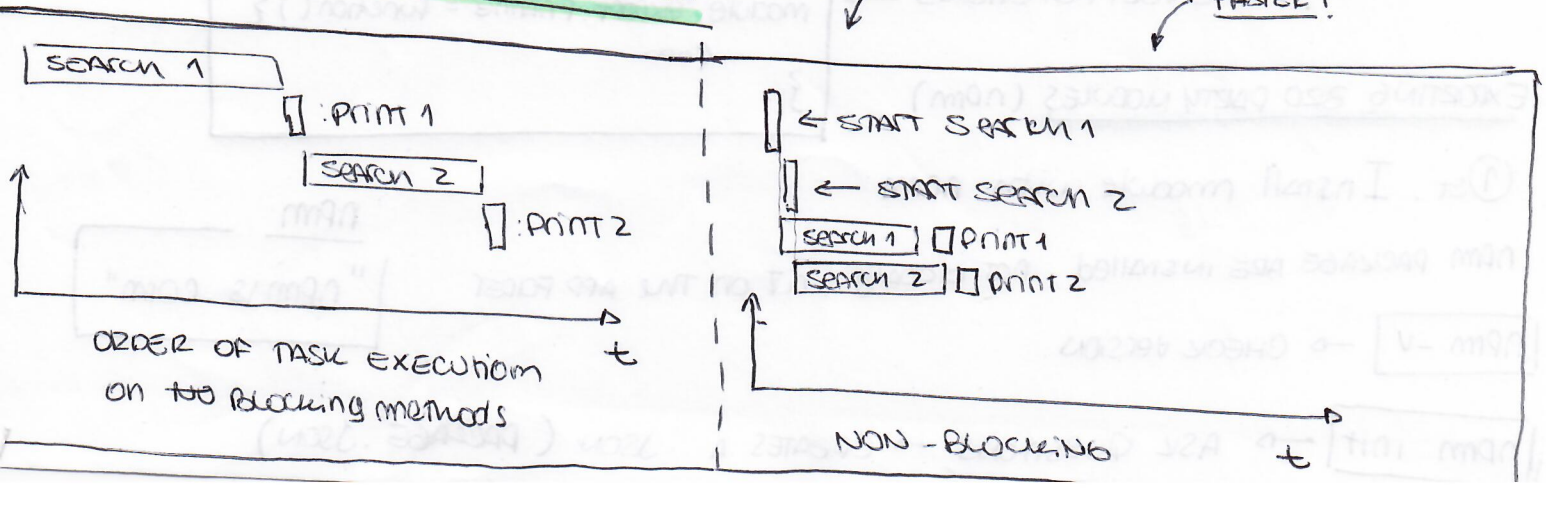
window → global

document → PROCESS

NON-BLOCKING I/O & EVENT DRIVEN

ONE THREADED !

FASTER !



TO RUN files → $ node filename.js

NPM → BIG ECOSISTEM (BIG COMMUNITY)

THIRD PARTY SOFTWARE, SOLVE COMMON PROBLEMS

Require → Loads in Built-in Modules

Module → Functions

(fs = FileSystem) (.AppendFile)

List of Built-in Modules → "https://nodejs.org/api/"

Modules Docs

Modules Functions

```
const fs = require('fs');
```

JUST 1

one string only!

```
fs.AppendFile('hi.txt', 'hello');
```

CREATES FILE!

writes on it.

GRAB USER WITH OS MODULE →

```
const os = require('os');
```

```
console.log(user.name)
```

```
VAR user = os.userInfo();
```

TEMPLATE STRING (ES6) →

```
console.log(`hola ${user.name}`);
```

=

5|

5|

4|

Requiring our own files

```
VAR my_file = require("RELATIVE_PATH");
```

But b-4 → TO USE FUNCTIONS INSIDE MY files I MUST EXPORTE em.

EXPORTS

- export ATTRIBUTES →

```
module.export.AGE = 25;
```

- EXPORT FUNCTIONS →

```
module.export.FName = function(){
    //ooo
}
```

EXPORTING 3RD PARTY MODULES (npm)

1st. Install module with npm

npm

"npmjs.com"

NPM PACKAGE ARE INSTALLED, NOT GLOBALLY, but on the APP FOLDER

npm -v → CHECK VERSION

npm init → ASK QUESTIONS → CREATES A .JSON (PACKAGE.JSON)

SOME NPM PACKAGES

- lodash → $> npm install lodash --save
  - Adds lodash to my folder.

PROJECT-Folder

- /node-modules ← Do not git this folder.
- YOUR APP
- ...

```
const _ = require('lodash');
```
→ `_.isString('string');` → T / F

`_.uniq([10,10,2,'was','was'])` → Removes duplicates

WITH THE CORRECT `PACKAGE.JSON` we can restore the PACKAGES with the `npm install` command, this will install all modules.

Nodemon

- node mon is like A continious RUNNING SERVER
- RESTART THE APP WITH EVERY CHANGE

`$> npm install nodemon -g` → Install nodemon globally
- is not added to the /node_modules

User inputs

`process.argv` → Array de comandos

["node", "APP.js", "add", ...]
0 1 2 3...

for boys: console.log(process.argv)

PROCESS.ARGV[2]

YARGS. LIBRARY → PARSES ARGUMENTS THE CORRECT WAY.

good for things like

```
>$ node APP.js add --title "holo"
```

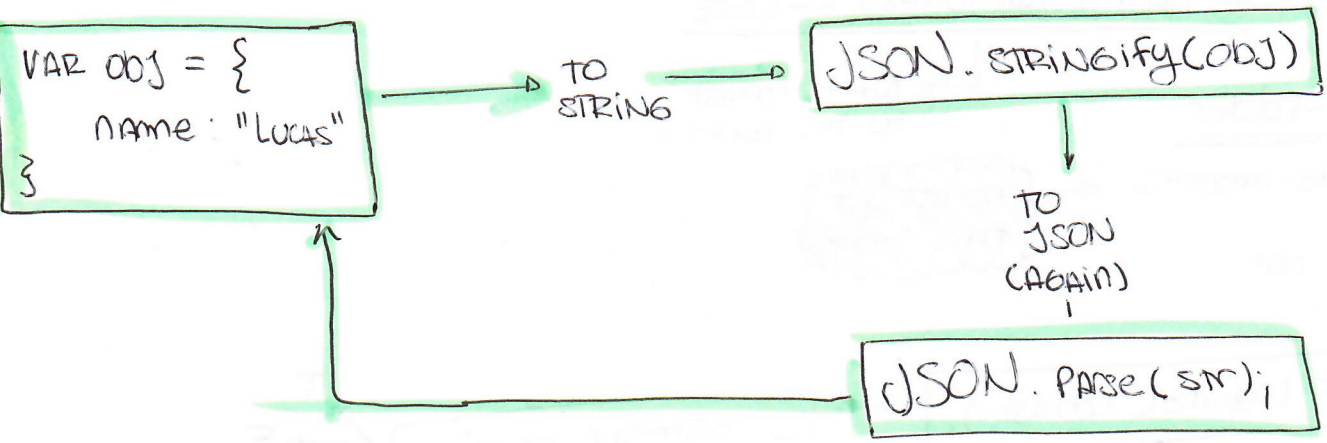
TO ACCESS ARG

```
const ARGUMENTOS = YARGS.ARGV;
```
- ARGUMENTOS.TITLE
- ARGUMENTOS.BODY

ARGV

YARGS.ARGV._[0] = PROCESS.ARGV[2]

Working with JSONS



TO CHECK TYPE

`console.log(typeof object);` → RETURNS TYPE

Write Files with the "File system" Module.

- `fs.readFileSync(file);`
- `fs.writeFileSync(file, string);`

TO APPEND TO AN ARRAY

`arr-name.push(str);`

Try and Catch

```
try {
  //
} catch {
  //
}
```

The filter(); function

Creates and array with all the elements that have passed the test.

Syntax.

`arr-name.filter(function);` → RUNS FOR EACH element.

Debugging

Node.js debugging works on node V8 > (not the engine)

`node inspect APP.JS` → Run on debug mode.

$debug> list(n) → list n lines
n → next block of code
repl → > examine VARS

Add a `debugger;` to jump to that line of code when debugging (3)

`nodemon inspect` → live debugging

debug

Debugging via Chrome tools

`$> node --inspect-brk APP.JS` on (CLI)

(nodemon)

+

`chrome://inspect` on (Chrome URI)

Arrow Functions

(ES6 Feature)

ESS vs ES6
f(x) f(x)

```
VAR square = (x) => {
    VAR R = x * x;
    return R;
}
```

with 1 argument you can omit the parentesis.

with 0 Arg or >1 you must use () or (x,y)

```
VAR square = (x) => x * x;
```

← simpler

Arrow functions on object (can't always be used

DO NOT use "this" on objects as you would use with normal funetions

```
VAR user = {
  name: 'LUCAS',
  sayHi : function(){
  }} console.log(`Hi ${this.name}`);
```

```
VAR user = {
    name : 'LUCAS',
    sayHi : () => {
        console.log(`hi ${this.name}`);
    }
    sayHiAlt () {
        console.log(`Hi ${this.name}`);
    }
}
```

THIS WONT WORK

HAS NOT

THIS will WORK

HAS

the object 'Arguments' → object array with All the Arguments of the method.

Section 4 (Weather App)

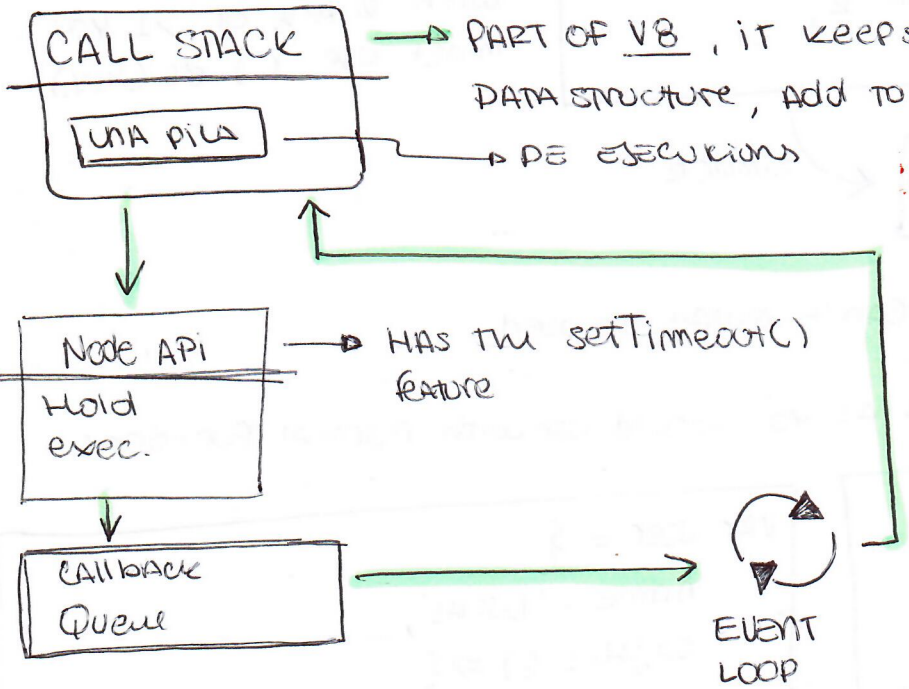
Async Basics

```
setTimeout(function, 2000);
```

↳ waits 2000ms until it runs function

```
console.log('Hello');
setTimeout(()=>{console.log('How are you');}, 2000);
console.log('Goodbye');
```

How it works (The behind the scenes)



#YOU DON'T DIRECTLY INTERACT WITH THIS CONCEPTS. BUT IS IMPORTANT TO UNDERSTAND EM.

CallBack Functions

F(x) That are called inside another functions arguments.

```
VAR getUsue = (id, callback) => {
    VAr user = {
      id: id,
      name: 'Luis'};
    callback(user)
}
```

```
getUser(3, (userObject) => {
    console.log(userObject);
});
```

ES6 Promises


```
VAR somePromise = new Promise( () => {
  // ...
});
```

← SYNTAX

"Promise fullfilled" ⇒ it has done what expected.

"Promis rejected" ⇒ it hasn't done what it has promised

```
VAR newPromise = new Promise ( (resolve, reject) => {
  resolve("it worked"); });
```

#1 Succes

```
VAR newPromise = new Promise( (resolve, reject) => {
  reject("it dion't work"); });
```

#2 Erros

Functions f(x)

How to handle.

```
newPromise.then( f(x) , g(x) );
```

Binary STATE: fullfilled, rejected } settled; Pending

```
newPromise.then( (messege) => {
  console.log(messege);
}, (errorMessey) => {
  console.log(errormessage);
});
```

Just one of both is executed ever, newn both And only once

Usefull Example (Returning A Promise)

```
VAR asyncAdd = (a, b) => {
  return new Promise( (resolve, reject) => {
    if (type of a === 'number' && type of b === 'number'){
      resolve(a + b);
    } else {
      reject('Arguments must be numbers');
    }
  });
};
```

Get the json from google API

https://maps.googleapis.com/maps/api/geocode/json?address=MARTINEZ,ARGENTINA

Request Package

NPM Request → see docs

callback

HTML body

```
request('url', (error, response, body) => { });
```

it executes once the requires function finishes

THIS CAN also be a JSON that tells the server

how it would be nice to get that response

```
request({url:'___', json:true}, (erro...
```

response
- statusCode: 200 / 400 / 500
- header — server / resiver
- body
- ...

error — code — error on the request process
- domain not found
- no conection

body → has a body.status that helps handle errors

How to encode an URL (decode)

```
encodeURIComponent("2060 MARTINEZ, BUENOS AIRES");   → encode
decodeURIComponent("      ");                        → decode
```

ERROR

CATCH THEM ALL!

```
if(error){
 //
}
```

ES6 Promises

(comparation with callback)

Chaining Promises

```
AsyncAdd(4,5).then((res) => {
    console.log(res);
    return AsyncAdd(res,3);
}).then((res) => {
    console.log(res);
}).catch((ErrorMessage) => {
}); console.log(ErrorMessage);
```

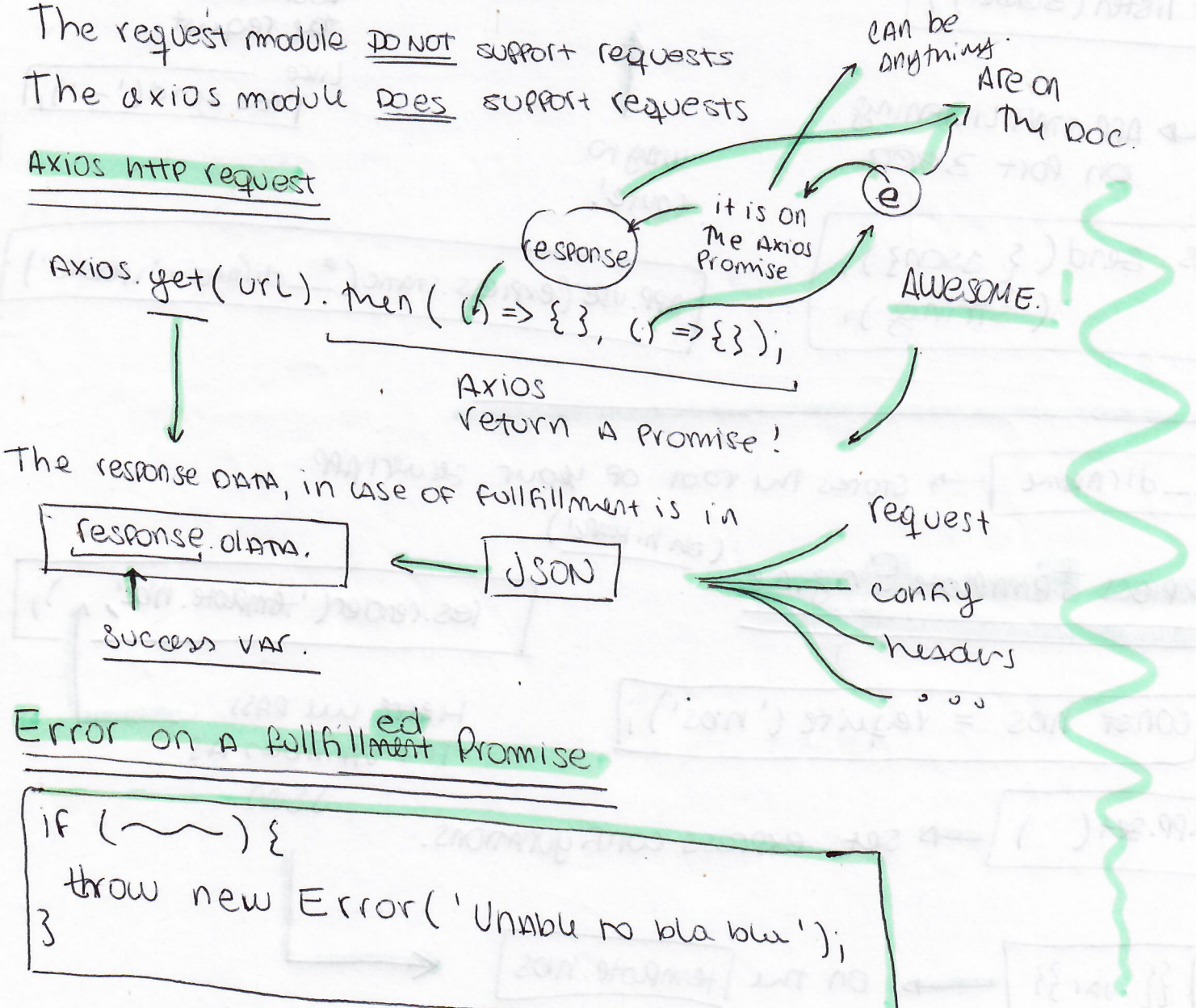
→ Chaining Promises.

→ In case any of the Promises fails.

No second arguments for .then()

The request module DO NOT support requests
The axios module DOES support requests

Axios http request



Error on a fullfilled Promise

```
if (~~~) {
  throw new Error('Unable to bla bla');
}
```

Hello Express

```
$> npm install express
```

Node JS Web Server

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

```
var app = express();
```
← express APP

Http route handler

Register a handler →

```
app.get('/', (req, res) => {
});
```

Info about the req

callback functions

methods use to respond to the request

Like:
```
res.send('~');
```

```
app.listen(3000);
```
→ APP START LISTENING on PORT 3000

way to route!

```
res.send({json});
         (string);
```

```
APP.use(express.static(__dirname + '\public'));
```

```
__dirname
```
→ stores the root of your server APP

Express Template Engine (oh hi MARK!)

```
res.render('template.nbs', );
```

```
const nbs = require('nbs');
```

Here we PASS the VARIABLES AS JSON.

```
app.set( )
```
→ set express configurations.

```
{{ var }}
```
→ on the `template.NBS`

Partials (hbs)

once hbs is required

```
hbs.registerPartials(__dirname + '/views/partials');
```

```
{{> footer }}
```

Helpers (hbs)

to load content to various templates ~~AT~~ A time

```
hbs.registerHelper('helpername', () => { });
```

we can also pass a var. as an ~~arg~~ and use it like this

needs to return something

```
{{helpername page-num}}
```

Express Middleware

Register a middleware

express.js.com
API — req, res

```
app.use( (req, res, next) => {
});
next();
```

→ to continue...

Testing Applications

`describe()` organizes the test

I have skipped all of the videos but here are some things

`$> npm test`

`'__'.test.js` files are in charge of the test

"mocha" for testing dependencies for dev.

`--save-dev` ← or saves as

Section 7

Mongo DB → it is a non-relational db, so data can be repeated.

Download and install de Community Edition on Linux.

with the `$> tar -xzfv '____'` once installed, just run the files on `'____'/bin`

`mongod --dbPath '/mongo-data'` → Here is how to start the mongo db

↳ This is how to specify where data should be stored

MySQL Workbench
DBeaver
HeidiSQL

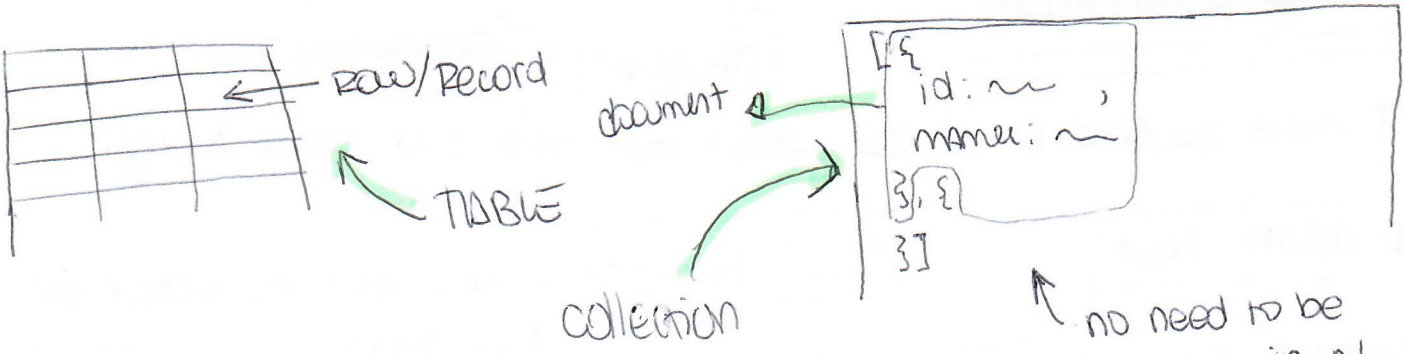
Default port: 27017

~~Robomongo~~ → ROBO 3T
(mongoname) (robo3t)
to

NO SQL VS SQL

SQL → TABLE AS STRUCTURE

NO SQL → ARRAY AS (collection) STRUCTURE



Connect to Mongo DB

we would use a node-module developed by Mongo team

6it hub | mongodb/~~mongo~~-node-mongodb-native

```
$> npm i mongodb --save
```

```
const MongoClient = require('mongodb').MongoClient
MongoClient.connect(url, callback);
Handle errors and has the client.
(err, client) => {
  if(err) {
    // ~
  }
const db = client.db("dbname");
```

→ select DB, if it does not I it creates it!!

Insert into a Collection

```
db.collection('coleccion_name')
.insertOne({JSONTOINSERT}, (err, res) => {})
```

DATA (document)

res.ops → ARRAY of the document inserted

The ObjectID (are uniques)

```
_id : "~~~~~~~~~~~~"
```

→ Random id generated., Pensado para ser escalables, No hay necesidad de fijarse el último id. genera otro random

Por ejemplo

```
_id : "57ac8d4787a244e5dc21bc8"
```

↳ 12 bytes value
- (1st) 4 bytes is a timestamp.
- Next 3 bytes machine identifies.
  - ↳ next 2 bytes process id
  - ↳ 3 byte counter.

You can specify the _id → Replace the auto generated.

Pull timestamp from id → result.ops[0]._id.getTimeStamp()

ES6 feature

Object destructuring

```
VAR USER = { NAME: 'Andrew', AGE: 25};
VAR {NAME} = USER;
console.log(NAME); ⟶ Andrew
```

bread → delete
→ edit
→ read
~~broadcast~~ Add
Browse.

Destructuring for calling mongo node module

```
const {MongoClient} = require("mongodb");
```

ES6

```
const {MongoClient, ObjectID} = require("mongodb");
```

for using or creating ObjectID.

```
VAR obj = new ObjectID();
```
⟶
```
VAR idObj = new ObjectID(str);
```

Fetch DATA on MongoDB

```
db.collection('colectname').find()
```
query.

```
.toArray().then()
```

RETURNS A Promsie
so
Returns an Object Cursor.

SET the key VALUE PAIR

```
ooo.find({ #A key: value});
```

```
.count()
```

.find()

↳ Returns An Object with multiple methods that ARE cool to google because they do a lot. thing great!

↳ Returns A Promise And counts the value if fullfilled

Delete Documents for mongo

Delete many

```
db.collection('col_name').deletMany({key: value}).then(//)
```

Delete One

```
.deleteOne({key: value})
```
→ just delete the first one (wtf firstone)

Delete one and Find it (?

- ~~delete~~
- findOneAndDelete({//}) ~~then()~~ .then()

→ Returns the deleted document - within ~~that~~ the promise.

Update one for Documents in Mongo

```
db.collection('_').findOneAndUpdate(filter, update, options, callb)
```

If no callback passed. it Returns a (Promise) (!)

Mongo DB update operators

$inc → Increments value of the field by the spec. Amount
$mul → multiplies " " " "
$rename → Renames field
$set → Sets the value of a field if an update results in an insert of a document. Has no effect on update operators that modify existing documents

USE OPERATORS

```
db.collection('collection_name').findAndUpdate({_id: new ObjectID('~')}, {
  $set: {
  }, {
    completed: true
  }, {
    returnOriginal: false
  }).then((result) => {console.log(result)});
```

The Mongoose ORM (sequelize?)

[ORM] stands for { Object, Recational, Mapping }

To connect:

```
mongoose.connect('~~localhost~~:mongodb://localhost:27017/DBNAME');
```

↑ mongoose MAINTAIN our connection over time

Does not returns a Promise, because → it waits until it connects. to exec any request.

Mongoose compels you TO CREATE A Model for everything you wanna store

Mongoose Model

Model :)

```
VAR Todo = mongoose.model('Name', {
    text: {
        type: String
    }, {
    completed: {
        type: Boolean
    },
    completedAt {
        type: Number
    }
});
```

On fullfilled the doc. is PASSED.

```
VAR newTodo = new Todo({
    text: "cook dinner"
});
```

~~newTodo().save()~~

```
newTodo.save().then( | ),
(returns A Promise)
```

Validators, Types And Defaults

Google → Mongoose Validators { required, min/max for numbers, minlength/max length 4 str

Google → Mongoose Schemas

```
VAR myMongooseModel = new mongoose.Model('name', );
```

#Mongoose parse ~~N~~umber and boolean to str if type is str on Schema.

"true" ← true , "34" ← 34

Express Routes

CRUD → CREATE, READE, UPDATE, DELETE

```
VAR APP = express();
APP.POST('/todos', (req, res) => {
});
~~APP.ROOT~~ console.log(req.body);
```

```
APP.use(bodyParser.json());
```
(express middleware) ← npm module.

Please, separate Routes From Controllers (I like it THAT WAY) ♡

LARAVEL ♡