BOOTCAMP VNT

APIs REST

COM

NODE.JS

Agenda

- Introdução ao Node.js
- ES6
- NPM
- Construindo a API
- Trabalhando com banco de dados relacional
- Implementando CRUD dos recursos da API
- Autenticando usuários
- Preparando o ambiente de produção
- Documentando a API


Introdução ao
node
JS

O que é?

- Criado por Ryan Dahl em 2009
- Não é uma linguagem
- Runtime de JavaScript que leva o processamento do código JavaScript para o lado do servidor
- Utiliza o motor JavaScript V8 do Google Chrome
- Permite a criação de aplicações web, desktop e ambientes de desenvolvimento front-end
- Single thread
- Orientado a eventos
- Processamento de requisições I/O não-bloqueante

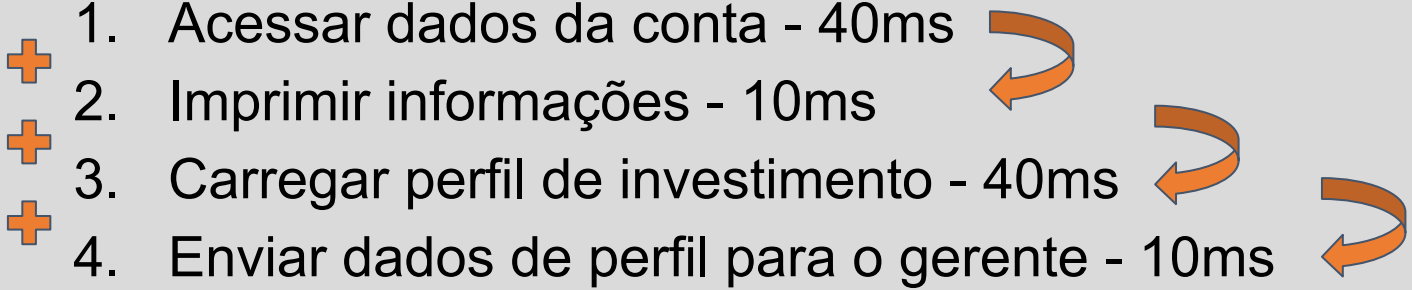
Estudo de caso

Um programa que precise acessar os dados da conta de uma pessoa e imprimir na tela algumas informações. Além disso, ele deve carregar o perfil de investimento do cliente e enviar esses dados para o gerente da conta.

1. Acessar dados da conta - 40ms
2. Imprimir informações - 10ms
3. Carregar perfil de investimento - 40ms
4. Enviar dados de perfil para o gerente - 10ms

Resultado em uma plataforma bloqueante

1. Acessar dados da conta - 40ms
2. Imprimir informações - 10ms
3. Carregar perfil de investimento - 40ms
4. Enviar dados de perfil para o gerente - 10ms



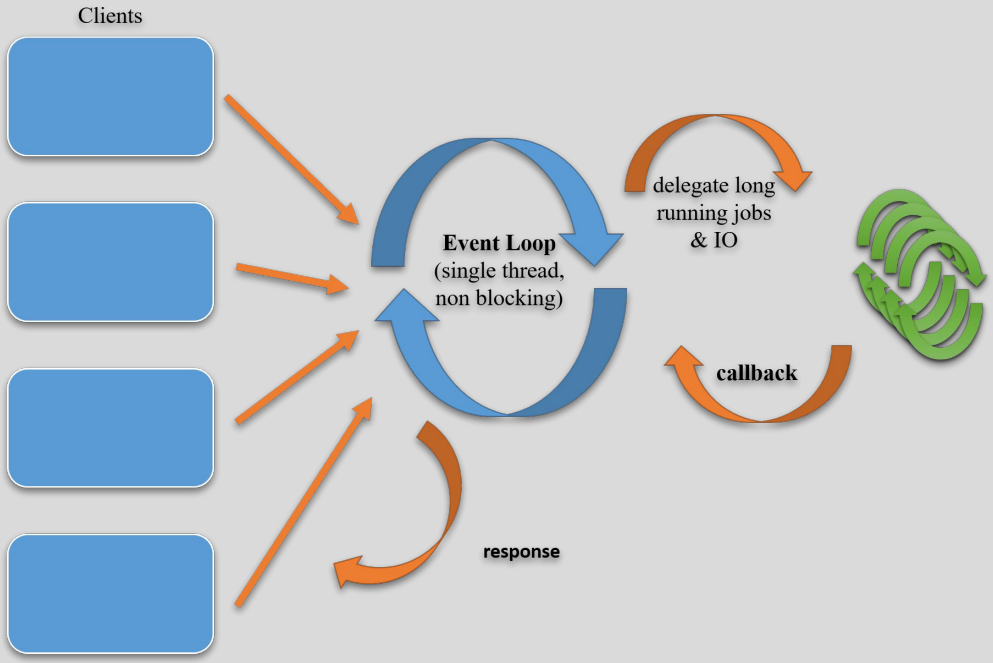
= 100ms

Resultado em Node.js

1. Acessar dados da conta - 40ms
   a. Imprimir informações - 10ms
2. Carregar perfil de investimento - 40ms
   a. Enviar dados de perfil para o gerente - 10ms

= 50ms

Event Loop


Clients
Event Loop
(single thread,
non blocking)
delegate long
running jobs
& IO
callback
response

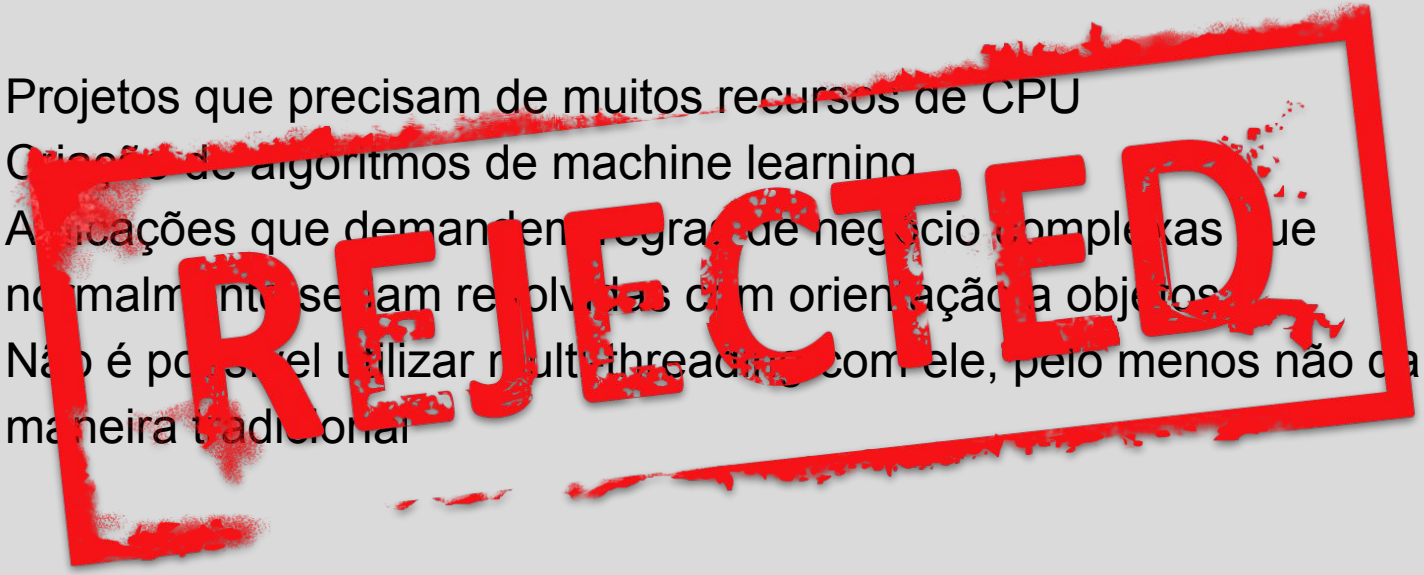

Quando usar
e quando
não usar?

Quando usar

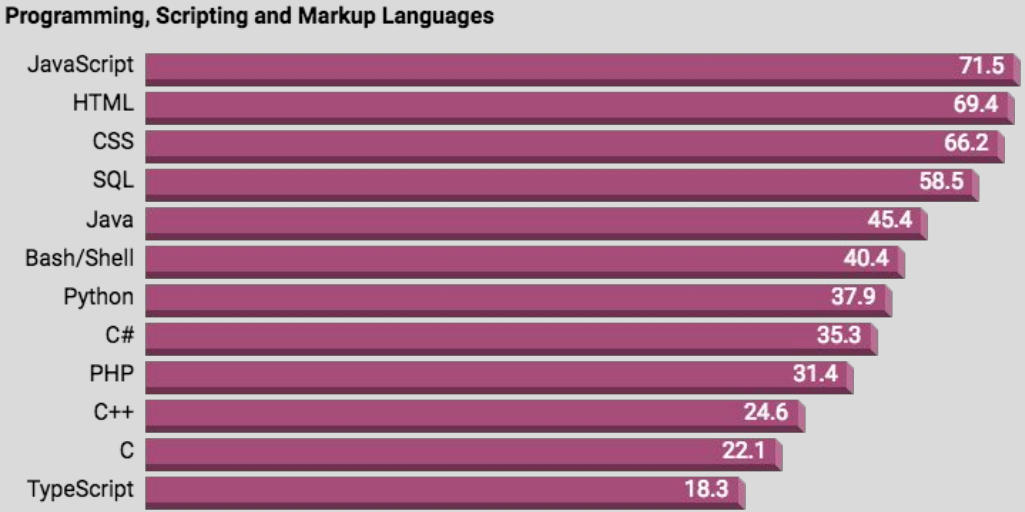
- Criar aplicações de chat e mensag[illegible] tarefas
- Criar APIs qu[illegible] facilmente escalaveis
- D[illegible] em streaming
- Io[illegible]
- O[illegible]ros (Criação de dashboards para monitoramento de aplicações, cri[illegible]ção de sist[illegible]mas para conectores de ações e até mesmo backend pa[illegible]ogos)

Quando NÃO usar

- Projetos que precisam de muitos recursos de CPU
- Criação de algoritmos de machine learning
- Aplicações que demandem regras de negócio complexas que normalmente sejam resolvidas com orientação a objetos
- Não é possível utilizar multithreading com ele, pelo menos não da maneira tradicional

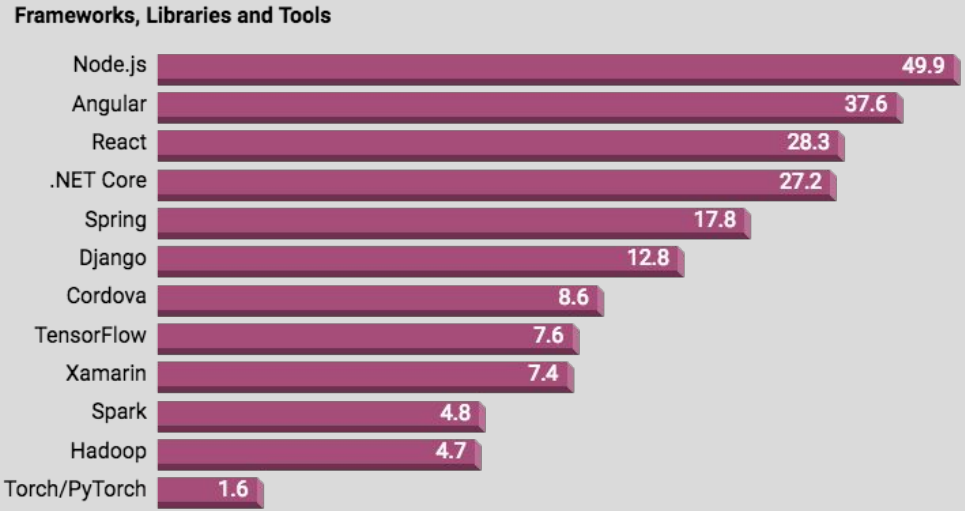


Developer Survey Results 2018



https://insights.stackoverflow.com/survey/2018

Developer Survey Results 2018



https://insights.stackoverflow.com/survey/2018

Big players

Big players


ES6

Declaração de variáveis com const e let

```
var arr = [1, 2, 3];
var obj = {};
var num = 1;
var str = 'Bootcamp';
var bool = false;
```

Declaração de variáveis com const e let

```
var language = 'JavaScript';
language = 'Python';
console.log(language);

Python
```

Declaração de variáveis com const e let

```
const language = 'JavaScript';
language = 'Python';
console.log(language);

TypeError: Assignment to constant variable.
```

Declaração de variáveis com const e let

```
const person = {
  name: 'Homer'
};
person.age = 39;
console.log(person);

{ name: 'Homer', age: 39 }
```

Declaração de variáveis com const e let

```
let result = 0;
result = 5;
console.log(result);

5
```

Declaração de variáveis com const e let

```
var id = 1;
var id = 2;
console.log(id);

2
```

Declaração de variáveis com const e let

```
let id = 1;
let id = 2;
console.log(id);

SyntaxError: Identifier 'id' has already been declared
```

Declaração de variáveis com const e let

```
var message = 'Hello!';
{
  var message = Bye!';
}
console.log(message);


Bye!
```

Declaração de variáveis com const e let

```
let message = 'Hello!';
{
  let message = Bye!';
}
console.log(message);


Hello!
```

Manipulação de textos com template strings

```
const name = 'Bart';
console.log(Welcome, ' + name + '!');

Welcome, Bart!
```

Manipulação de textos com template strings

```
const name = 'Bart';
console.log(`Welcome, ${name}!`);

Welcome, Bart!
```

Manipulação de textos com template strings

```
showing
one
word
per
line

console.log('showing\none\nword\nper\nline');
```

Manipulação de textos com template strings

```
showing
one
word
per
line

console.log(`
showing
one
word
per
line
`);
```

Arrow functions

```
function sayHello(name) {
  return `Hello, ${name}!`;
}
```

```
const sayHello = function(name) {
  return `Hello, ${name}!`;
}
```

```
const sayHello = (name) => {
  return `Hello, ${name}!`;
}
```

```
const sayHello = name => {
  return `Hello, ${name}!`;
}
```

```
const sayHello = name => `Hello, ${name}!`;
```

Métodos auxiliares para Array (forEach)

```
const names = ['Marge', 'Lisa', 'Maggie'];
for (let i = 0; i < names.length; i++) {
  console.log(names[i]);
}

Marge
Lisa
Maggie
```

Métodos auxiliares para Array (forEach)

```
const names = ['Marge', 'Lisa', 'Maggie'];
names.forEach(name => {
  console.log(name);
});

Marge
Lisa
Maggie
```

Métodos auxiliares para Array (map)

```
const numbers = [1, 2, 3];
const double = [];

for (let i = 0; i < numbers.length; i++) {
  double.push(numbers[i] * 2);
}

console.log(numbers);
console.log(double);

[ 1, 2, 3 ]
[ 2, 4, 6 ]
```

Métodos auxiliares para Array (map)

```
const numbers = [1, 2, 3];

const double = numbers.map(number => {
  return number * 2;
});

console.log(numbers);
console.log(double);

[ 1, 2, 3 ]
[ 2, 4, 6 ]
```

Métodos auxiliares para Array (filter)

```
const students = [
  { name: 'Milhouse', age: 10 },
  { name: 'Nelson', age: 11 },
  { name: 'Martin', age: 8 }
];
const studentsOlderThan9 = [];
for (let i = 0; i < students.length; i++) {
  if (students[i].age > 9) {
    studentsOlderThan9.push(students[i]);
  }
}
console.log(studentsOlderThan9);

[ { name: 'Milhouse', age: 10 }, { name: 'Nelson', age: 11 } ]
```

Métodos auxiliares para Array (filter)

```
const students = [
  { name: 'Milhouse', age: 10 },
  { name: 'Nelson', age: 11 },
  { name: 'Martin', age: 8 }
];

const studentsOlderThan9 = students.filter(student => {
  return student.age > 9;
});

console.log(studentsOlderThan9);



[ { name: 'Milhouse', age: 10 }, { name: 'Nelson', age: 11 } ]
```

Métodos auxiliares para Array (find)

```
const students = [
  { name: 'Milhouse' },
  { name: 'Nelson' },
  { name: 'Martin' }
];
let student;
for (let i = 0; i < students.length; i++) {
  if (students[i].name === 'Martin') {
    student = students[i];
    break;
  }
}
console.log(student);
{ name: 'Martin' }
```

Métodos auxiliares para Array (find)

```
const students = [
  { name: 'Milhouse' },
  { name: 'Nelson' },
  { name: 'Martin' }
];

const student = students.find((aluno) => {
  return student.name === 'Martin';
});

console.log(student);

{ name: 'Martin' }
```

Métodos auxiliares para Array (every)

```
const people = [
  { name: 'Moe', age: 38 },
  { name: 'Skinner', age: 36 },
  { name: 'Apu', age: 34 }
];
let allOlderThan30 = true;
for (let i = 0; i < people.length; i++) {
  if (people[i].age <= 30) {
    allOlderThan30 = false;
    break;
  }
}
console.log(allOlderThan30);
true
```

Métodos auxiliares para Array (every)

```
const people = [
  { name: 'Moe', age: 38 },
  { name: 'Skinner', age: 36 },
  { name: 'Apu', age: 34 }
];

const allOlderThan30 = people.every(person => {
  return person.age > 30;
});

console.log(allOlderThan30);
```

```
true
```

Métodos auxiliares para Array (some)

```
const baggageWeights = [12, 32, 21, 29];
let hasWeightExceeded = false;

for (let i = 0; i < baggageWeights.length; i++) {
  if (baggageWeights[i] > 23) {
    hasWeightExceeded = true;
  }
}

console.log(hasWeightExceeded);
```

```
true
```

Métodos auxiliares para Array (some)

```
const baggageWeights = [12, 32, 21, 29];

const hasWeightExceeded = baggageWeights.some(baggageWeight => {
  return baggageWeight > 23;
});

console.log(hasWeightExceeded);
```

```
true
```

NPM (Node Package Manager)

O que é?

- Gerenciador de pacotes
- Integrado ao instalador principal do Node.js
- Possui mais de 627.000 módulos criados por terceiros e comunidades
- 120 milhões de downloads diariamente
- 2,9 bilhões de downloads mensalmente

https://www.npmjs.com/

Principais comandos

- `npm init`: exibe um questionário para auxiliar na criação do package.json do projeto
- `npm install nome_do_modulo`: instala um módulo no projeto
- `npm install -g nome_do_modulo`: instala um módulo global
- `npm install nome_do_modulo --save`: instala o módulo e adiciona-o ao arquivo package.json, dentro do atributo "dependencies"
- `npm install nome_do_modulo --save-dev`: instala o módulo e adiciona-o no arquivo package.json, dentro do atributo "devDependencies"
- `npm list`: lista todos os módulos que foram instalados no projeto
- `npm list -g`: lista todos os módulos globais que foram instalados

Principais comandos

- `npm remove nome_do_modulo`: desinstala um módulo do projeto
- `npm remove -g nome_do_modulo`: desinstala um módulo global
- `npm remove nome_do_modulo --save`: desinstala um módulo do projeto, removendo também do atributo "dependencies" do package.json
- `npm remove nome_do_modulo --save-dev`: desinstala um módulo do projeto, removendo também do atributo "devDependencies" do package.json
- `npm update nome_do_modulo`: atualiza a versão de um módulo do projeto

Principais comandos

- `npm update -g nome_do_modulo`: atualiza a versão de um módulo global
- `npm -v`: exibe a versão atual do NPM
- `npm help`: exibe em detalhes todos os comandos

```
{
  "name": "meu-primero-node-app",
  "description": "Meu primeiro app Node.js",
  "author": "User",
  "version": "1.2.3",
  "private": true,
  "scripts": {
    "start": "node app.js",
    "clean": "rm -rf node_modules",
    "test": "node test.js",
  "dependencies": {
    "modulo-1": "1.0.0",
    "modulo-2": "~1.0.0",
    "modulo-3": "^1.0.0",
    "modulo-4": ">=1.0.0"
  },
  "devDependencies": {
    "modulo-5": "*"
  }
}
```


Criando nossos
primeiros
módulos

Construindo a API

Introdução ao Express

- Framework para a construção de APIs e também aplicações web
- Minimalista
- Foco em manipular views, routes e controllers
- Trabalha com o conceito de middlewares
- Possui uma grande lista de middlewares 3rd-party

https://www.expressjs.com/

```
npm install --save express
```

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

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

app.get('/projects', (req, res) => {
  res.send('Got a GET request');
});

app.post('/projects', (req, res) => {
  res.send('Got a POST request');
});
```

Outros tipos de response

```
res.download()
res.end()
res.redirect()
res.render()
res.sendStatus()
```


Criando o
projeto
piloto

Requisitos

- ❏ Listagem de tarefas do usuário
- ❏ Consulta, cadastro, exclusão e alteração de uma tarefa do usuário
- ❏ Cadastro de um usuário
- ❏ Autenticação de usuário
- ❏ Página de documentação da API

nodemon

- Monitora todas as alterações nos arquivos da sua aplicação e a reinicia quando for necessário
- Permite listar arquivos que não precisam ser monitorados

https://nodemon.io/

```
npm install --save-dev nodemon

npx nodemon
```

consign

- Permite carregar e injetar dependências de forma bem simples

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

consign()
  .include('models')
  .then('routes')
  .into(app);

// app.models.tasks
// app.models.users
// app.routes.tasks
// app.routes.users
```

```
npm install --save consign
```

Trabalhando com banco de dados relacional

Introdução ao Sequelize

- Framework Node.js do tipo ORM (Object Relational Mapper)
- Suporte a MySQL, PostgreSQL, SQL Server e SQLite
- Utiliza o padrão Promises
- Modelagem e relacionamento de tabelas
- Tratamento de transações
- Replicação de bancos para modo de leitura
- Migrações

http://docs.sequelizejs.com/

```
npm install --save sequelize sqlite3
```

Configurando o Sequelize

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: 'mysql'|'sqlite'|'postgres'|'mssql',

  logging: false,

  pool: {
    max: 5,
    min: 0
  },

  // SQLite only
  storage: 'path/to/database.sqlite'
});
```

Modelando a aplicação com Sequelize

```
const Foo = sequelize.define('Foo', {
  id: {
    type: DataType.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  title: {
    type: DataType.STRING,
    allowNull: false,
    unique: true
  },
  flag: {
    type: DataType.BOOLEAN,
    allowNull: false,
    defaultValue: true
  }
});
```

Mais tipos de dados

```
STRING                  // VARCHAR(255)
STRING(1234)            // VARCHAR(1234)
TEXT                    // TEXT
TEXT('tiny')            // TINYTEXT

INTEGER                 // INTEGER
BIGINT                  // BIGINT
BIGINT(11)              // BIGINT(11)

FLOAT                   // FLOAT
FLOAT(11)               // FLOAT(11)
FLOAT(11, 12)           // FLOAT(11,12)

DOUBLE                  // DOUBLE
DOUBLE(11)              // DOUBLE(11)
DOUBLE(11, 12)          // DOUBLE(11,12)
```

Mais tipos de dados

```
DECIMAL                  // DECIMAL
DECIMAL(10, 2)           // DECIMAL(10,2)

DATE                     // DATETIME for mysql / sqlite, TIMESTAMP WITH TIME ZONE for postgres
DATE(6)                  // DATETIME(6) for mysql 5.6.4+. Fractional seconds support with up to 6 digits of
precision
DATEONLY                 // DATE without time.

BOOLEAN                  // TINYINT(1)

ENUM('value 1', 'value 2')  // An ENUM with allowed values 'value 1' and 'value 2'

JSON                     // JSON column. PostgreSQL, SQLite and MySQL only.
JSONB                    // JSONB column. PostgreSQL only.

BLOB                     // BLOB (bytea for PostgreSQL)
BLOB('tiny')             // TINYBLOB (bytea for PostgreSQL. Other options are medium and long)
```

Validando o modelo

```
const Foo = sequelize.define('Foo', {
  title: {
    type: DataType.STRING,
    validate: {
      isAlphanumeric: true,     // will only allow alphanumeric characters, so "_abc" will fail
      isUppercase: true,        // checks for uppercase
      contains: 'foo',          // force specific substrings
      len: [2,10]               // only allow values with length between 2 and 10
    }
  },
  amount: {
    type: DataType.INTEGER,
    validate: {
      isInt: true,              // checks for valid integers
      max: 30,                  // only allow values <= 23
      min: 20                   // only allow values >= 23
    }
});
```

Mais validadores

```
is: ["^[a-z]+$",'i']     // will only allow letters
is: /^[a-z]+$/i          // same as the previous example using real RegExp
not: ["[a-z]",'i']       // will not allow letters
isEmail: true            // checks for email format (foo@bar.com)
isUrl: true              // checks for url format (http://foo.com)
isIP: true               // checks for IPv4 (129.89.23.1) or IPv6 format
isIPv4: true             // checks for IPv4 (129.89.23.1)
isIPv6: true             // checks for IPv6 format
isAlpha: true            // will only allow letters
isNumeric: true          // will only allow numbers
isFloat: true            // checks for valid floating point numbers
isDecimal: true          // checks for any numbers
isLowercase: true        // checks for lowercase
notIn: [['foo', 'bar']] // check the value is not one of these
isIn: [['foo', 'bar']]   // check the value is one of these
notContains: 'bar'       // don't allow specific substrings
isDate: true             // only allow date strings
isCreditCard: true       // check for valid credit card numbers
```

Validadores customizados

```
const Foo = sequelize.define('Foo', {
  amount: {
    type: DataType.INTEGER,
    validate: {
      isInt: true,              // checks for valid integers
      max: 30,                  // only allow values <= 30
      min: 20,                  // only allow values >= 20
      isEven(value) {
        if (parseInt(value) % 2 != 0) {
          throw new Error('Only even values are allowed!')
        }
      }
    }
  }
});
```

Queries (find)

```
Projects.findById(123).then(project => {

});

Projects.findOne({
  where: { title: 'aProject' },
  attributes: ['id', 'title']
}).then(project => {

});

Projects.findAll().then(projects => {

});
```

Mais opções para queries

```
findOrCreate
findAndCountAll
count
max
min
sum
```

Operadores

```
Projects.findAll({
  where: {
    id: {
      [Op.gt]: 6,                 // id > 6
      [Op.lt]: 10                 // id < 10
    }
  }
});
```

Mais alguns operadores

```
[Op.and]: {a: 5}              // AND (a = 5)
[Op.or]: [{a: 5}, {a: 6}]     // (a = 5 OR a = 6)
[Op.gte]: 6                   // id >= 6
[Op.lte]: 10                  // id <= 10
[Op.ne]: 20                   // id != 20
[Op.between]: [6, 10]         // BETWEEN 6 AND 10
[Op.notBetween]: [11, 15]     // NOT BETWEEN 11 AND 15
[Op.in]: [1, 2]               // IN [1, 2]
[Op.notIn]: [1, 2]            // NOT IN [1, 2]
[Op.like]: '%hat'             // LIKE '%hat'
[Op.notLike]: '%hat'          // NOT LIKE '%hat'
```

Criar novos registros

```
User.create({
  username: 'barfooz',
  isAdmin: true
}).then(user => {

});

User.bulkCreate([
  { username: 'barfooz', isAdmin: true },
  { username: 'foo', isAdmin: true },
  { username: 'bar', isAdmin: false }
]).then(() => {

});
```

Atualizar registros

```
task.update({
  title: 'a very different title now'
}).then(() => {

});

Task.update(
  { status: 'inactive' },
  { where: { subject: 'programming' }
}).spread((affectedCount, affectedRows) => {

});
```

Excluir registros

```
task.destroy();

Task.destroy({
  where: {
    subject: 'programming'
  }
}).then(affectedCount => {

});
```

Hooks

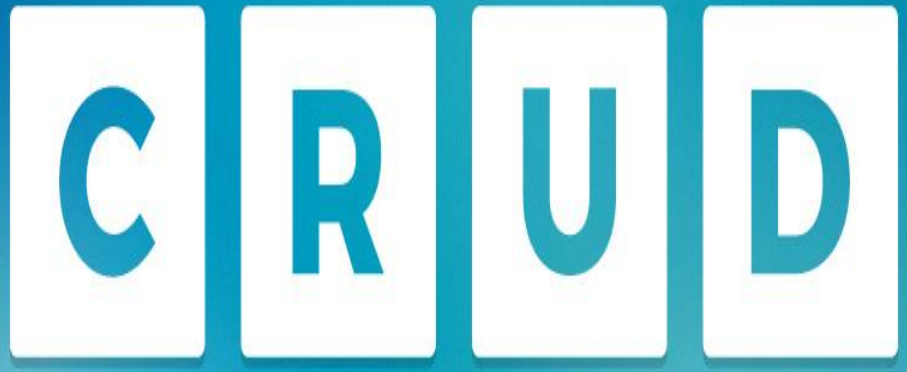
```
const User = sequelize.define('User', {
  name: DataType.STRING
});

User.hook('beforeCreate', (user, options) => {
  user.name = `Mr. {user.name}`;
});
```

Outros hooks

```
beforeValidate
afterValidate
beforeDestroy
beforeUpdate
beforeSave
beforeUpsert
afterCreate
afterDestroy
afterUpdate
afterSave
```

Implementando CRUD dos recursos da API



body-parser

- É um middleware para "parsear" o payload das requisições antes de chegar às rotas

```
npm install --save body-parser
```

https://www.npmjs.com/package/body-parser

Validações usando express-validator

- É um middleware para validação de dados de rotas do Express

```
npm install --save express-validator
```

https://www.npmjs.com/package/express-validator

```
const { check, validationResult } = require('express-validator/check');
const { matchedData } = require('express-validator/filter');

app.post('/users', [
  check('name', 'Required field').exists()
  check('name', 'Invalid length').isLength({ min: 1, max: 100 })
], (req, res) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const user = matchedData(req);

  createUser(user).then(user => res.json(user));
});
```

Autenticando usuários na API

Introdução ao Passport e JWT

- Framework flexível e extremamente modular
- Suporte às principais estratégias de autenticação: Basic, Digest, Bearer Token, OAuth, OAuth 2.0 e JWT
- Permite trabalhar com autenticação via serviços externos como Facebook, Google+, Twitter e muito mais
- Mais de 300 estratégias de autenticação

```
npm install --save passport passport-jwt jwt-simple
```

http://www.passportjs.org/

https://www.npmjs.com/package/passport-jwt

bcrypt-nodejs

- Criptografa dados baseado em uma chave
- Compara se um dado valor é igual a um valor criptografado

```
const bcrypt = require('bcrypt-nodejs');

const hash = bcrypt.hashSync('bacon');

bcrypt.compareSync('bacon', hash); // true
bcrypt.compareSync('veggies', hash); // false

npm install --save bcrypt-nodejs
```

https://www.npmjs.com/package/bcrypt-nodejs

Preparando o ambiente de produção

O que precisamos levar em consideração?

- CORS
- Processamento paralelo com o módulo cluster
- Compactação de requisições com GZIP
- HTTPS

CORS (Cross-Origin Resouce Sharing)

- Responsável por permitir ou barrar requisições realizadas por outros domínios

```
const express = require('express');
const cors = require('cors');
const app = express();
app.use(cors({
  origin: ['http://localhost:3001'],
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization']
}));

npm install --save cors
```

https://www.npmjs.com/package/cors

Processamento paralelo com o módulo cluster

- Instancia novos processos de uma aplicação, trabalhando de forma distribuída
- Compartilha a mesma porta da rede entre os clusters ativos
- Os clusters são independentes uns dos outros

https://nodejs.org/docs/latest-v8.x/api/cluster.html

Compactação de requisições com compression

- Middleware responsável por compactar as respostas JSON e também arquivos estáticos para o formato GZIP

```
const express = require('express');
const compression = require('compression');

const app = express();

app.use(compression());
```

```
npm install --save compression
```

https://www.npmjs.com/package/compression

HTTPS

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

const credentials = {
  key: fs.readFileSync('bootcamp.key', 'utf8'),
  cert: fs.readFileSync('bootcamp.cert', 'utf8')
};

https.createServer(credentials, (req, res) => {
  console.log('Bootcamp API');
}).listen(3000);
```

http://www.selfsignedcertificate.com/

Documentando a API

Introdução à ferramenta apiDoc

- É um CLI (Command Line Interface)
- Gera a documentação da API através da leitura de comentários padronizados

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

http://apidocjs.com/


BLACK OPS II
MISSION
COMPLETE

Daniel
daniel.moraes.xx@venturus.org.br

Danilo
danilo.souza@venturus.org.br

Rodrigo
rodrigo.mendonca@venturus.org.br