



# Node JS

Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifroma@gmail.com

# Agenda

- Node JS
  - Node Package Manager (NPM)
  - Instalação
  - Primeiros passos
  - Acessando módulos externos
- HTTP
  - Express JS
  - Rotas e Middlewares
  - Express Validator
  - Express Generator
  - Jason Web Token
- Banco de dados SQL
  - MySQL e SQLite
  - Sequelize JS
  - Definição e utilizando modelos
  - Consultas
  - Associações
  - Transações

# Node JS

# JavaScript



**JavaScript** is born  
as LiveScript

1997

**ES3** comes out and  
IE5 is all the rage

2000

**ES5** comes out and  
standard JSON

2015

**ES7/ECMAScript2016**  
comes out

2017

1995 **ECMAScript** standard  
is established

1999

XMLHttpRequest,  
a.k.a. AJAX,  
gains popularity

2009

**ES6/ECMAScript2015**  
comes out

2016

ES.Next

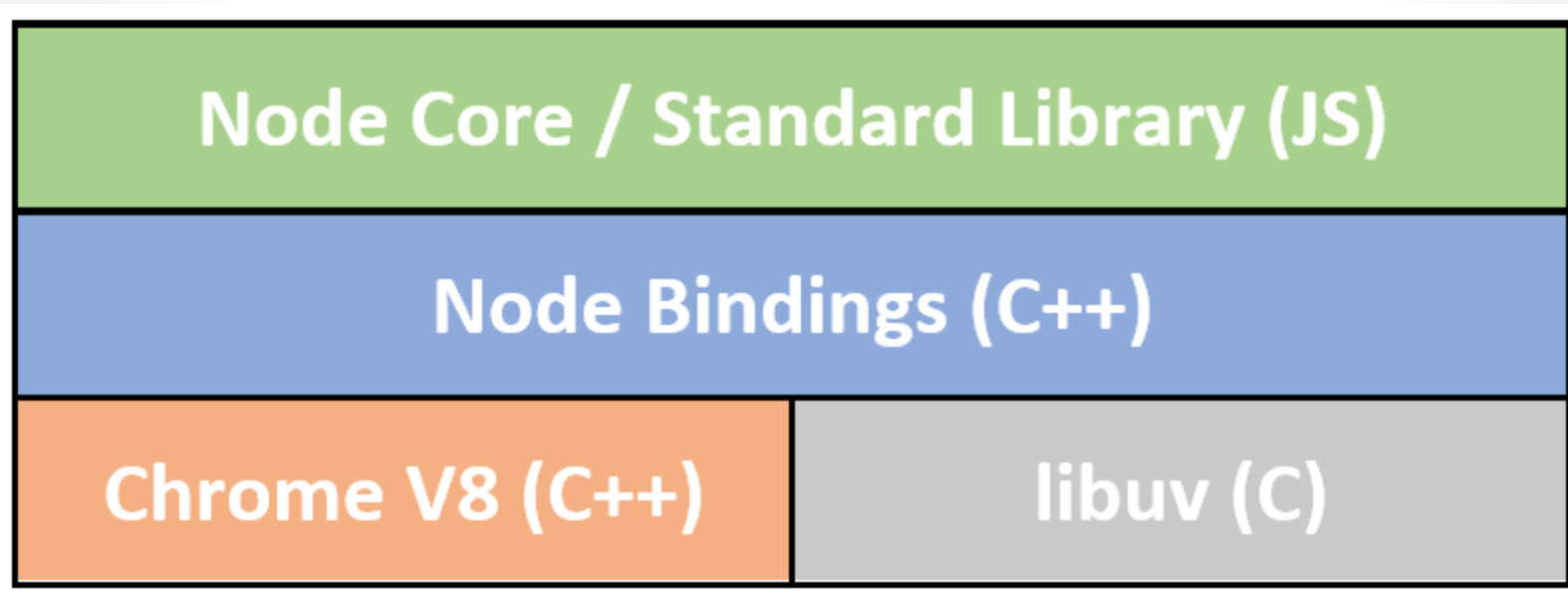
# Node JS

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world.

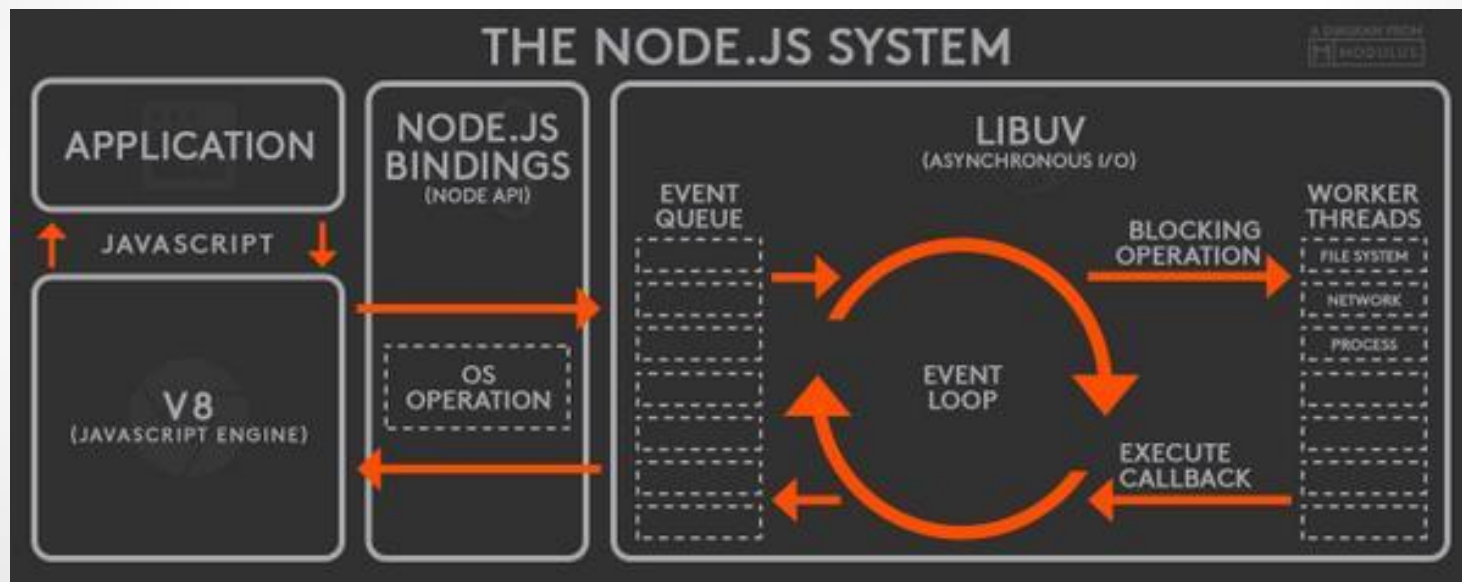
<https://nodejs.org/>

- Runtime JavaScript desvinculado do navegador
- Orientado à eventos
- Gratuito e de código aberto
- Criado por Ryan Dahl

# Arquitetura



# Sistema



# Onde usar?

- APIs
- Backend para Jogos, IoT e Apps
- Aplicações em tempo real
- Automatizar tarefas



# Quem utiliza?



GROUPON



DOW JONES



NETFLIX



PayPal



UBER



LinkedIn

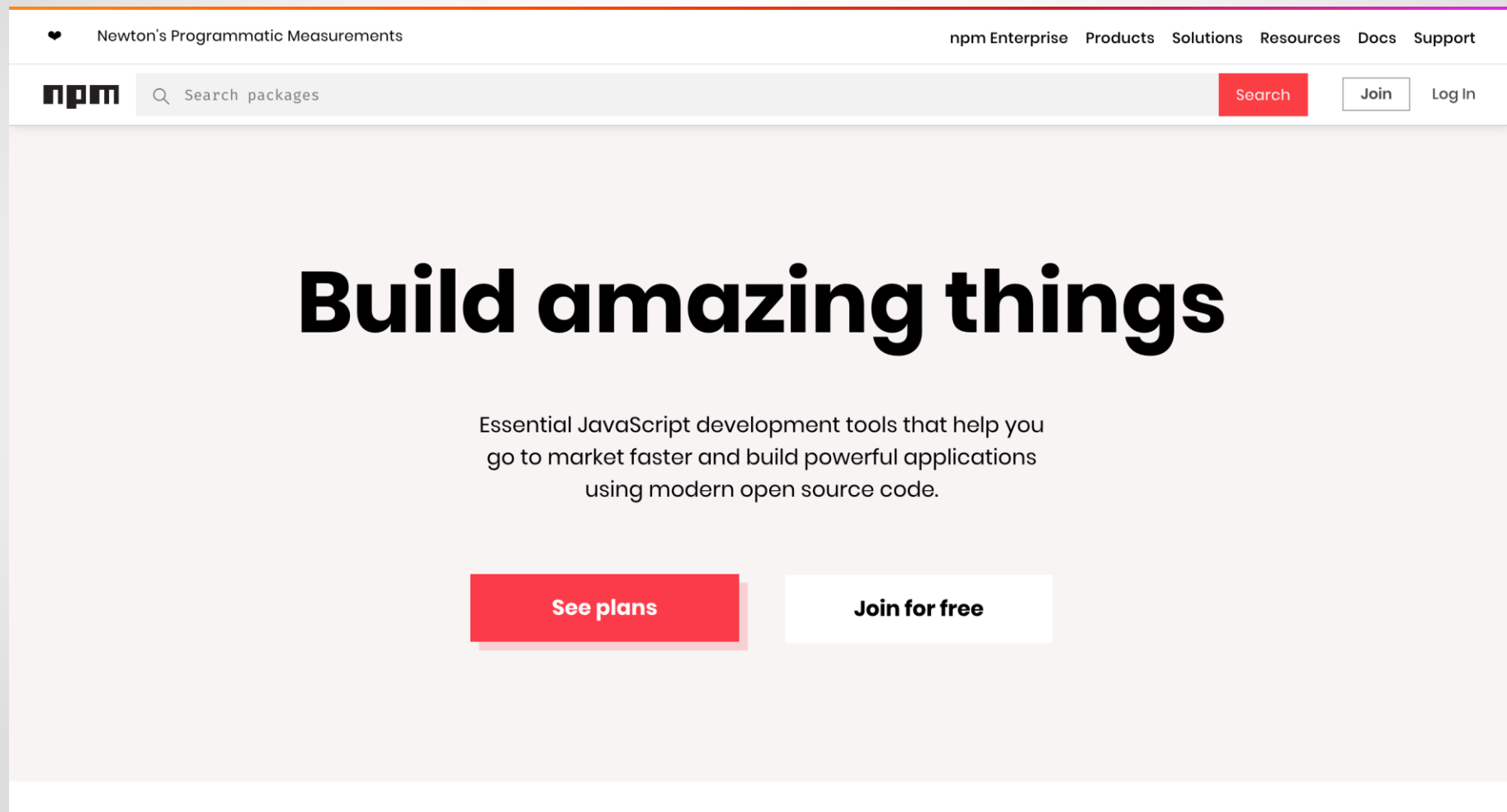


Walmart



Go Daddy®

# Node Package Manager (npm)



The screenshot shows the npm website interface. At the top, there is a navigation bar with a heart icon and the text "Newton's Programmatic Measurements" on the left, and links for "npm Enterprise", "Products", "Solutions", "Resources", "Docs", and "Support" on the right. Below this is a search bar with the "npm" logo, a search icon, and the placeholder text "Search packages". To the right of the search bar are buttons for "Search", "Join", and "Log In". The main content area features a large heading "Build amazing things" in a bold, black font. Below this heading is a paragraph of text: "Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code." At the bottom of this section are two buttons: a red button labeled "See plans" and a white button labeled "Join for free".

Newton's Programmatic Measurements

npm Enterprise Products Solutions Resources Docs Support

npm Search packages Search Join Log In

## Build amazing things

Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

See plans Join for free

<http://npmjs.com>

# Instalando no Windows

1. Instale o Git com GitBash (Recomendado)

<https://git-scm.com/download/win>

2. Faça o download do NodeJS de acordo com sua arquitetura

<https://nodejs.org/en/download/>

3. Execute o instalador

Next, Next, Next and Finish.

# Instalando em Debian like

## 1. Adicione o repositório de pacotes

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
```

## 2. Execute a instalação

```
$ sudo apt-get install -y nodejs
```

# Instalando em Red Hat like

## 1. Adicione o repositório de pacotes

```
$ curl --silent --location https://rpm.nodesource.com/setup_8.x | sudo bash -
```

## 2. Execute a instalação

```
$ sudo yum -y install nodejs
```

# Instalando no Mac OSX

## 1. Instale o Homebrew (caso ainda não tenha)

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

## 2. Execute a instalação

```
$ brew install node@8
```

# Conferindo a Instalação

- Verificando a versão do Node

```
$ node --version
```

- Verificando a versão do NPM

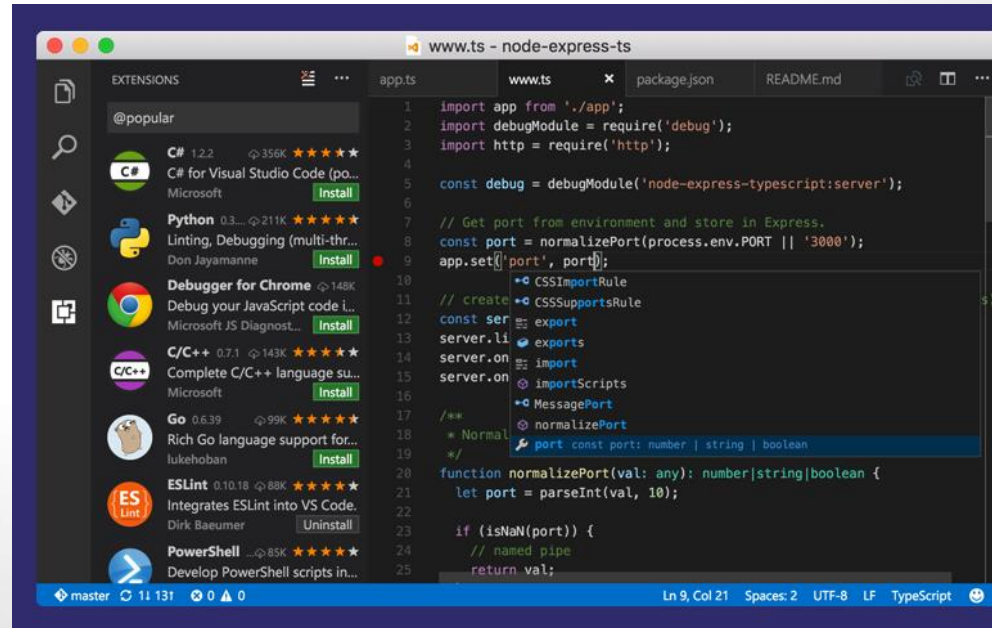
```
$ npm --version
```

- Atualizando o NPM

```
$ npm install -g npm
```

# Bônus

- **Visual Studio Code (Recomendado)**
  - <https://code.visualstudio.com/>
  - Debug integrado
  - Auto-completar
  - Identificação automática
  - Terminal integrado
  - Variedade de extensões





# Primeiros passos com Node JS

- Crie uma pasta para o projeto. Ex:  
`MeuPrimeiroProjeto`
- Crie um arquivo `index.js` que imprima uma mensagem no console.

```
console.log('Olá Node JS!');
```

- Execute o arquivo com o Node JS.  
`$ node index.js`

# Primeiros passos com NPM

- Dentro do diretório `MeuPrimeiroProjeto` inicie o arquivo `package.json`.

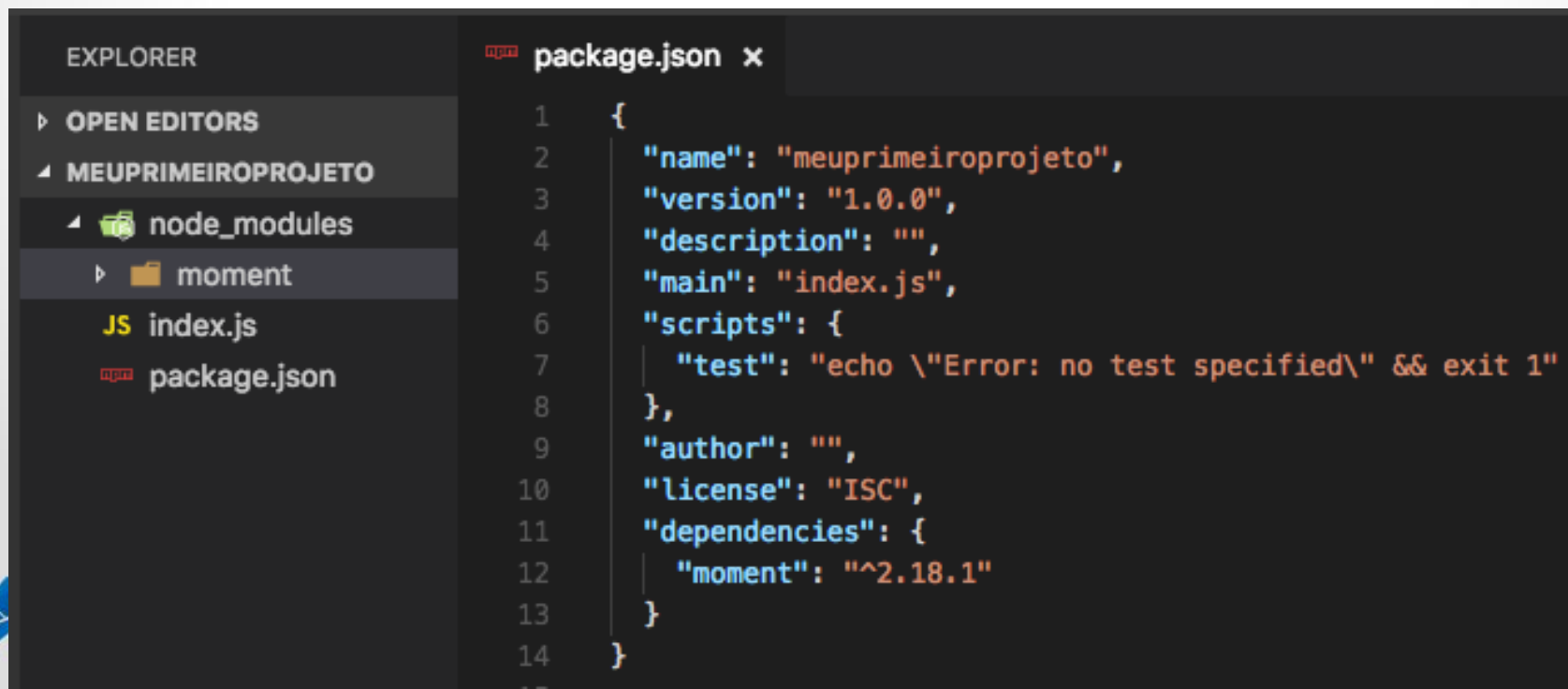
```
$ npm init
```

```
{
  "name": "meuprimeiroprojeto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

# Primeiros passos com NPM

- Instale a dependência `moment` para manipulação de datas.

```
$ npm install --save moment
```



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure: 'MEUPRIMEIROPROJETO' containing 'node\_modules' (with a subfolder 'moment') and 'index.js'. The main editor area shows the 'package.json' file with the following content:

```

1  {
2    "name": "meuprimeiroprojeto",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "moment": "^2.18.1"
13   }
14 }
```

# Acessando módulos externos

- Importe o módulo `moment` e imprima a data atual, formatada, no console.

```
const moment = require('moment');  
  
const dataAtual = new Date();  
  
const dataFormatada =  
moment(dataAtual).format('DD/MM/YYYY HH:mm:ss');  
  
console.log(dataFormatada);
```

- Execute o arquivo com o Node JS  
\$ node index.js

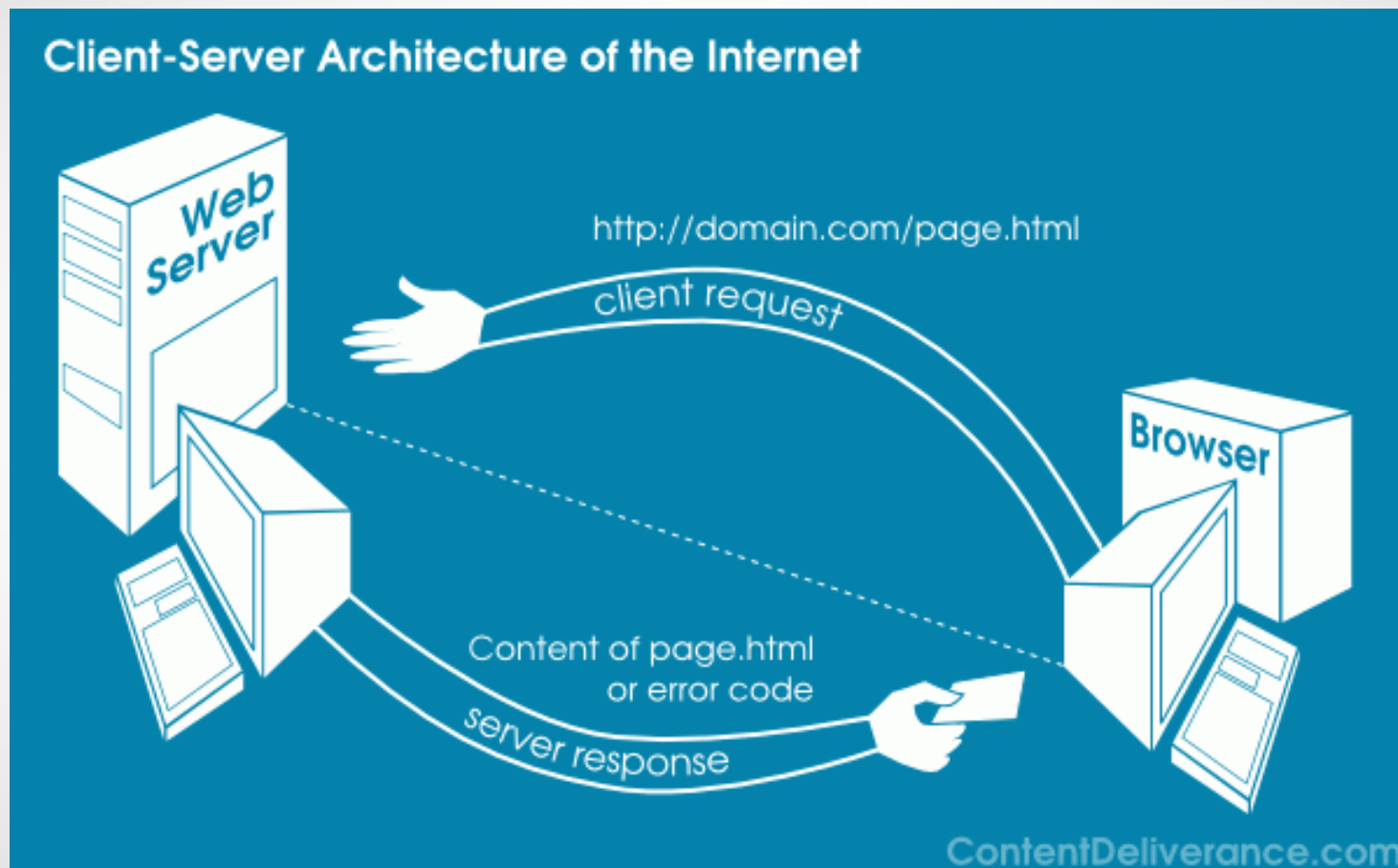
# HTTP com Node JS

# Protocolo HTTP



<http://arquivo.devmedia.com.br/REVISTAS/java/imagens/117/3/1.png>

# Protocolo http



<http://contentdeliverance.com/cms-school/wp-content/uploads/2011/05/client-server-diagram-internet.png>

# HTTP com Node JS

- Importe o módulo `http` e inicie o serviço na porta desejada.

```
const http = require('http');
const porta = 3000;

http.createServer((request, response) => {

    response.setHeader('content-type', 'text/html; charset=utf-8');
    response.writeHead(200);
    response.write('Olá Node JS!');
    response.end();

}).listen(porta, () => {
    console.log('Servidor iniciado na porta:', porta);
});
```

- Abra no navegador `http://localhost:3000`



# Express JS

- O Express é um framework Node JS para aplicativos Web pequeno e flexível que fornece um conjunto robusto de recursos.

<http://expressjs.com/>

- Com o Express ganhamos produtividade e uma camada maior de recursos para construir aplicações Web e Mobile com Node JS.

# Instalando o Express JS

- O primeiro passo para trabalhar com o Express JS é realizar a instalação do módulo, utilizando o NPM:

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

# Rotas

- No Express, rotas são as URLs que a aplicação será capaz de responder.
- Por exemplo, para definir uma rota para listagem de usuários, poderíamos escrever:

```
const express = require('express');  
const server = express();  
  
server.get('/usuarios/:usuarioId/', (request, response) => {  
  const usuarioId = request.params.usuarioId;  
  response.status(200).send("Listando usuários: " + usuarioId);  
})
```

# Criando rotas

```
const express = require('express');
const server = express();
const porta = 3000;

server.get('/usuarios/:usuarioId/', (request, response) => {
  const usuarioId = request.params.usuarioId;
  response.status(200).send("Listando usuários: " + usuarioId);
})

server.post('/usuarios/', (request, response) => {
  let usuario = "";
  request.on('data', (chunk) => { usuario += chunk });
  request.on('end', () => {
    console.log('Usuários recebido:', usuario);
    response.status(201).send();
  });
})

server.listen(porta, () => {
  console.log('Servidor iniciado na porta:', porta);
})
```

# Middlewares

- Middlewares atuam como mediadores que interceptam as requisições para realizar algum tipo de tarefa.
  - Validação, autenticação, body-parser e etc.
- Registrando o middleware “body-parser”:

```
server.use(express.json());  
server.use(express.urlencoded({ extended: false }));
```

- Registrando um middleware próprio:

```
server.use((request, response, next) => {  
  if (usuarioAutenticado()) {  
    next();  
  } else {  
    response.status(403).send();  
  }  
});
```

# Express Validator

- Express Validator fornece **middleware** que facilita a validação do corpo da requisição e parâmetros da URL.

```
server.post('/usuarios/', [  
  check('email').isEmail().trim(),  
  check("nome").isString().isLength({ min: 2 })  
], (request, response) => {  
  const errors = validationResult(request);  
  if (!errors.isEmpty()) {  
    return response.status(422).json({ errors: errors.array() });  
  }  
  
  let usuario = request.body;  
  console.log('Usuários recebido:', usuario);  
  response.status(201).send();  
});
```

# Express Generator

- Facilita a criação e organização inicial do projeto.  
Instale a dependência globalmente:

```
$ npm install express-generator -g
```

- Para criar um novo projeto:

```
$ express meu-projeto
```

- Instale as dependências do novo projeto:

```
$ cd meu-projeto
```

```
$ npm install
```

# Jason Web Token (JWT)

- JSON Web Token (JWT) é um padrão industrial aberto RFC 7519 para representar reivindicações de forma segura entre duas partes.
- O JWT é formado de três partes: **Header**, **Payload** e **Signature**.
- Exemplo:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjE0NjQ5ODUwMC4iLCJ0eXAiOiJKV1BB95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ



# Depurando o JWT

- É possível testar e validar o seu JWT diretamente no site <https://jwt.io>

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiOTkxMTQyNDAwInQ=

Decoded

EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYLOAD: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

VERIFY SIGNATURE

HMACSHA256(  
 base64UrlEncode(header) + "." +  
 base64UrlEncode(payload),  
 secret  
) ☐secret base64 encoded

Signature Verified

# JWT com Node JS

- Para trabalhar com JWT no Node, basta instalar a dependência:

```
$ npm install --save jsonwebtoken
```

- Para codificar um JWT:

```
const jwt = require("jsonwebtoken");  
  
const secret = "minha-senha-super-secreta";  
const token = jwt.sign(payload, secret);
```

- Para decodificar:

```
const jwt = require("jsonwebtoken");  
  
const secret = "minha-senha-super-secreta";  
const payload = jwt.verify(token, secret);
```

# Banco de Dados SQL

# Banco de Dados Relacional

- Assim como em linguagens como Java e C#, no NodeJS é preciso instalar um módulo (driver) para conexão com o banco de dados desejado.
- Para consultar a disponibilidade do banco de dados desejado, consulte o nome do banco no repositório <http://npmjs.com>
- Por exemplo, para MySQL podemos utilizar o módulo `mysql2` e para SQLite o módulo `sqlite3`.

```
$ npm install --save sqlite3
```

```
$ npm install --save mysql2
```

# Exemplo de uso do SQLite

- Para conectar diretamente com o banco de dados SQLite, basta seguir o exemplo:

```
const sqlite3 = require('sqlite3');
const db = new sqlite3.Database('database.sqlite');

db.serialize(function () {
  db.run("CREATE TABLE IF NOT EXISTS user (name TEXT)");
  const stmt = db.prepare("INSERT INTO user VALUES (?)");
  for (let i = 0; i < 10; i++) {
    stmt.run("Person " + i);
  }
  stmt.finalize();
  db.each("SELECT rowid AS id, name FROM user", function (err, row) {
    console.log(row.id + ": " + row.name);
  });
});
```

# Exemplo de uso do MySQL

- Para conectar diretamente com o banco de dados MySQL, basta seguir o exemplo:

```
const mysql = require('mysql2');

const connection = mysql.createConnection({
  host: 'localhost', user: 'root',
  password: '1234', database: 'test'
});

connection.query(
  'SELECT * FROM `user` WHERE `name` = "Paul" AND `age` > 45',
  function (err, results, fields) {
    console.log(results); // resultado contendo as linhas da consulta
    console.log(fields); // campos contendo metadados sobre os resultados
  }
);
```

# Exemplo de uso do MySQL

- Utilizando Prepared Statements

```
connection.execute(  
  'SELECT * FROM `user` WHERE `name` = ? AND `age` > ?',  
  ['Paul', 53],  
  function (err, results, fields) {  
    console.log(results); // resultado contendo as linhas da consulta  
    console.log(fields); // campos contendo meta dados sobre o resultados  
  }  
);
```

# Sequelize JS



Sequelize

- Sequelize é um framework para Mapeamento de Objetos Relacionais (ORM) para Node JS.
- Possui suporte à PostgreSQL, MySQL, SQLite e MSSQL.
- Possui funcionalidades sólidas como relacionamento entre entidades e transações.



# Introdução ao Sequelize JS

- Para iniciar com o Sequelize basta instalar o módulo:

```
$ npm install --save sequelize
```

- Em seguida, instalar o driver para o banco de dados desejado:

```
$ npm install --save pg pg-hstore
```

```
$ npm install --save mysql2
```

```
$ npm install --save mariadb
```

```
$ npm install --save sqlite3
```

```
$ npm install --save tedious // MSSQL
```

<http://docs.sequelizejs.com/manual/getting-started.html>

# Introdução ao Sequelize JS

- Conectando ao banco de dados SQLite.

```
const Sequelize = require('sequelize');  
  
const sequelize = new Sequelize(null, null, null, {  
  dialect: 'sqlite',  
  storage: './database.sqlite',  
});
```

- Conectando ao banco de dados MySQL.

```
const sequelize = new Sequelize('database', 'user', 'password', {  
  dialect: 'mysql',  
  host: '172.0.0.1',  
  port: '3306'  
});
```

# Testando a Conexão

```
sequelize.authenticate()  
  .then(() => {  
    console.log('Banco de dados conectado com sucesso.');  }).catch((ex) => {  
    console.error("Não foi possível se conectar ao banco de dados.", ex);  
  });
```

# Sincronizando o banco de dados

- O Sequelize também é capaz de criar as tabelas no banco de dados baseado na definição dos modelos.

```
sequelize.sync()  
.then(() => {  
  console.log('Banco de dados criado com sucesso.');
```

```
}).catch((ex) => {  
  console.error("Não foi possível se criar ao banco de dados.", ex);  
});
```

- Para forçar a destruição e recriação das tabelas forneça o parâmetro “force”.

```
sequelize.sync({ force: true });
```

# Definição dos Modelos

Vamos definir a entidade que representará a tabela “usuario” no banco de dados.

```
const Usuario = sequelize.define('usuario', {
  id: {
    primaryKey: true,
    type: Sequelize.BIGINT,
    autoIncrement: true,
  },
  nome: {
    type: Sequelize.STRING(200),
    allowNull: false,
  },
  nascimento: Sequelize.DATEONLY,
  email: Sequelize.STRING(150),
});
```

# Utilizando os Modelos

- Inserindo um elemento no banco de dados.

```
Usuario.create({  
  nome: 'Douglas Junior', email: 'nassifrroma@gmail.com'  
}).then(usuario => {  
  // você pode acessar agora o usuário criado  
  // através da variável "usuario"  
  console.log("Usuário inserido:", JSON.stringify(usuario));  
})
```

<http://docs.sequelizejs.com/manual/instances.html>

# Utilizando os Modelos

- Buscando por um elemento específico

```
Usuario.findById(123).then(usuario => {  
  // Retorna o usuário correspondente ao ID especificado,  
  // ou Null caso não seja encontrado.  
  console.log("Usuário selecionado:", JSON.stringify(usuario))  
})
```

```
Usuario.findOne({  
  where: {  
    nome: 'Douglas Junior'  
  }  
}).then(usuario => {  
  // Retorna o primeiro usuário com a condição especificada,  
  // ou Null caso não seja encontrado.  
  console.log("Usuário selecionado:", JSON.stringify(usuario))  
})
```

# Utilizando os Modelos

- Atualizando um elemento no banco de dados.

```
// Maneira 1
usuario.nome = 'Douglas Nassif'
usuario.save().then(() => { });
```

```
// Maneira 2
usuario.update({
  nome: 'Douglas Nassif'
}).then(() => { });
```

```
// Maneira 3
Usuario.update({
  nome: 'Douglas Nassif'
},{
  where: {
    id: 123
  }
}).then(() => { });
```



# Utilizando os Modelos

- Excluindo um elemento no banco de dados.

```
// Maneira 1  
usuario.destroy().then(() => { });
```

```
// Maneira 2  
Usuario.destroy({  
  where: {  
    id: 1  
  }  
}).then(() => { });
```

# Consultas

- Definindo quais atributos devem ser retornados na consulta.

```
Usuario.findAll({  
  attributes: ['nome', 'email']  
}).then(usuarios => {  
  console.log('Usuários selecionados:', JSON.stringify(usuarios));  
})
```

# Consultas

- Executando funções do banco de dados, como COUNT, MAX, MIN, etc.

```
Usuario.findAll({
  attributes: [[sequelize.fn('COUNT', sequelize.col('id')), 'qtd_usuarios']]
}).then(resultado => {
  console.log('Quantidade de usuários:', JSON.stringify(resultado));
})
```

# Consultas

- Filtrando consultas com “where”.

```
Usuario.findAll({  
  where: {  
    nome: {  
      [sequelize.Op]: '%douglas%'  
    }  
  }  
}).then(usuarioes => {  
  console.log('Usuários selecionados:', JSON.stringify(usuarioes));  
})
```

# Consultas

- Filtrando e contando o total de registros, útil para uso em paginação.

```
Usuario.findAndCountAll({  
  where: { },  
  limit: 10,  
  offset: 0,  
}).then(usuarioes => {  
  console.log('Quantidade de usuários:', JSON.stringify(usuarioes.count));  
  console.log('Usuários selecionados:', JSON.stringify(usuarioes.rows));  
})
```

# Associações

- Associar a entidade “usuario” à “tarefa”, onde a tarefa recebe a chave estrangeira do usuário.

```
// O usuário tem muitas tarefas
Usuario.hasMany(Tarefa, {
  onDelete: 'NO ACTION',
  onUpdate: 'NO ACTION'
})

// A tarefa tem a chave estrangeira do usuário
Tarefa.belongsTo(Usuario, {
  onDelete: 'NO ACTION',
  onUpdate: 'NO ACTION'
});
```

# Associações

- Consultando com JOINS.

```
Usuario.findAll({  
  where: { },  
  include: [{  
    model: Tarefa,  
    required: true, // true para inner join, false para left join  
  }],  
}).then(usuarios => {  
  console.log('Usuários com tarefas:', JSON.stringify(usuarios));  
})
```

# Transações

- Transações são utilizadas para garantir a integração entre diversas ações no banco de dados.

```
sequelize.transaction((transaction) => {  
  
    // as operações transacionadas são executadas aqui  
  
}).then(() => {  
    console.log('transação comitada');  
}).catch(ex => {  
    console.error('transação revertida (rollback):', ex);  
})
```



# Transações

- Exemplo:

```
sequelize.transaction((transaction) => {
  return Usuario.create(
    {
      nome: 'Douglas Junior', email: 'nassifrroma@gmail.com'
    }, {
      transaction
    }
  ).then(usuario => {
    return Tarefa.create(
      {
        titulo: 'Minha tarefa', usuarioId: usuario.id
      }, {
        transaction
      }
    )
  })
})
.then(() => { })
.catch(ex => { })
```

# Referências

- Node JS - <https://nodejs.org/>
- Lib UV - <http://libuv.org/>
- Node Package Manager (NPM) - <https://www.npmjs.com/>
- Git e GitBash - <https://git-scm.com/downloads>
- Visual Studio Code - <https://code.visualstudio.com/>
- Moment JS - <https://momentjs.com/>
- DevMedia: Como funcionam as aplicações Web - <http://www.devmedia.com.br/como-funcionam-as-aplicacoes-web/25888>
- Node JS HTTP - [https://nodejs.org/api/http.html#http\\_class\\_http\\_server](https://nodejs.org/api/http.html#http_class_http_server)
- Express JS - <http://expressjs.com/pt-br/>
- Express Validator - <https://github.com/ctavan/express-validator>
- Express Generator - <http://expressjs.com/pt-br/starter/generator.html>
- Json Web Token - <https://jwt.io/>
- Node JsonWebToken - <https://github.com/auth0/node-jsonwebtoken>
- Node MySQL2 - <https://github.com/sidorares/node-mysql2>
- Node SQLite3 - <https://github.com/mapbox/node-sqlite3>
- Sequelize JS - <http://docs.sequelizejs.com>
- Promises - [https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise)

# Obrigado!

Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifrroma@gmail.com