

Rodrigo Cichetto Monteiro

# **Desenvolvimento de aplicações com JavaScript**

Brasil

2018



Rodrigo Cichetto Monteiro

## **Desenvolvimento de aplicações com JavaScript**

Trabalho apresentado a UNIP - UNIVERSIDADE PAULISTA como pré-requisito para obtenção da Certificação de Conclusão do Curso de Bacharelado em Ciência da Computação. Orientador: Prof. Leandro Carlos Fernandez

Universidade Paulista – UNIP  
Faculdade de Ciência da Computação  
Programa de Graduação

Orientador: Leandro Carlos Fernandez

Brasil

2018

---

Rodrigo Cichetto Monteiro

Desenvolvimento de aplicações com JavaScript / Rodrigo Cichetto Monteiro. –  
Brasil, 2018-

47 p. : il. (algumas color.) ; 30 cm.

Orientador: Leandro Carlos Fernandez

Tese (Graduação) – Universidade Paulista – UNIP

Faculdade de Ciência da Computação

Programa de Graduação, 2018.

1. JavaScript. 2. Arquitetura multicamadas. 3. MEAN Stack. I. Orientador  
Leandro Carlos Fernandez. II. Universidade Paulista – UNIP. III. Faculdade de  
Ciência da Computação. IV. Desenvolvimento de aplicações com JavaScript V.  
Rodrigo Cichetto Monteiro

CDU 02:141:005.7

---

Rodrigo Cichetto Monteiro

## **Desenvolvimento de aplicações com JavaScript**

Trabalho apresentado a UNIP - UNIVERSIDADE PAULISTA como pré-requisito para obtenção da Certificação de Conclusão do Curso de Bacharelado em Ciência da Computação. Orientador: Prof. Leandro Carlos Fernandez

Trabalho aprovado. Brasil, 24 de novembro de 2018:

---

**Leandro Carlos Fernandez**  
Orientador

---

**Professor**  
Convidado 1

---

**Professor**  
Convidado 2

Brasil  
2018



*Este trabalho é dedicado aos meus pais,  
pelo apoio em todos os momentos.*





*“Ser o homem mais rico do cemitério não me interessa.  
Ir para a cama à noite dizendo que fizemos algo maravilhoso,  
isso importa para mim.  
(Steve Jobs; The Wall Street Journal, 1993.)*



# Resumo

Já pensou em criar uma rápida prototipação de software ou desenvolver aplicações escaláveis de forma rápida e utilizando apenas uma linguagem? Isso é possível e traz benefícios a empresa, desenvolvedores e até mesmo ao cliente. Nos últimos tempos o JavaScript ganhou muita importância em quaisquer cenários, e vem sendo utilizada em sites, aplicações, *mobile*, servidores, automação de testes, automação de tarefas, internet das coisas, entre outros. Este trabalho tem como principal objetivo apresentar benefícios de utilizar a linguagem JavaScript em todas as camadas do desenvolvimento, destacando ferramentas já existentes aplicadas à arquitetura multicamadas. Mas lembre-se com grandes poderes vem grandes responsabilidades.

**Palavras-chaves:** javascript, arquitetura multicamadas, frameworks



# Abstract

Have you thought about creating a rapid prototyping of software or developing scalable applications quickly and using only one programming language? This is possible and brings benefits to the company, developers and even the customer. In recent times JavaScript has gained a lot of importance in any scenario, and has been used in websites, applications, mobile, servers, automation of tests, automation of tasks, internet of things, among others. This work has as main objective to present benefits of using the JavaScript language in all layers of development, highlighting existing tools applied to multilayer architecture. But remember with great powers comes great responsibilities.

**Key-words:** javascript, multilayer architecture, frameworks



# Lista de ilustrações

Figura 1 – <i>Carrousel</i> - Um componente de apresentação de slides para percorrer imagens ou slides de texto - como um carrossel. – Exemplo de aplicação do JS . . . . .	25
Figura 2 – <i>Validation</i> - Validação de formulários – Exemplo de aplicação do JS . .	25
Figura 3 – <i>Modal</i> - Caixas de diálogo para notificações ao usuário. – Exemplo de aplicação do JS . . . . .	25
Figura 4 – Exemplo de arquitetura multicamadas . . . . .	26
Figura 5 – <i>MEAN Stack</i> - Fluxo das ferramentas que integram o MEAN . . . . .	29
Figura 6 – Exemplo de aplicação cliente criada com Angular para dispositivos móveis	36
Figura 7 – Exemplo de serviço criado com <i>Node.JS</i> , <i>Express</i> consultando <i>MongoDB</i> , expõe a documentação e monitoramento em tempo real dos dados . . .	36
Figura 8 – Aplicação gerada pelo Express Generator . . . . .	44
Figura 9 – Aplicação gerada pelo Angular Cli . . . . .	45
Figura 10 – Exemplo de configurações preenchidas no comando <code>npm init</code> . . . . .	47





# Lista de abreviaturas e siglas

JS	JavaScript
HTML	Abreviação para <i>HyperText Markup Language</i> , que em português significa Linguagem de Marcação de Hipertexto.
DOM	Abreviação para <i>Document Object Model</i> , que em português significa Modelo de Objetos e Documentos.
CSS	Abreviação para <i>Cascading Style Sheets</i> , que em português significa Folhas de Estilo em Cascata.
ECMA	Abreviação para <i>European Computer Manufacturers Association</i> , que em português significa Associação Européia de Fabricantes de Computadores.
API	Abreviação para <i>Application Programming Interface</i> , que em português significa Interface de programação de aplicações.
REST	Abreviação para <i>Representational State Transfer</i> , que em português significa Transferência de Estado Representacional.
HTTP	Abreviação para <i>Hypertext Transfer Protocol</i> , que em português significa Protocolo de Transferência de Hipertexto.
URI	Abreviação para <i>Uniform Resource Identifier</i> , que em português significa Identificador Uniforme de Recurso.
SQL	Abreviação para <i>Structured Query Language</i> , que em português significa Linguagem de Consulta Estruturada.



# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>19</b>
<b>1.1</b>	<b>Objetivos</b>	<b>19</b>
<b>2</b>	<b>REVISÃO BIBLIOGRÁFICA</b>	<b>21</b>
<b>2.1</b>	<b>Uma linguagem para todo o desenvolvimento</b>	<b>21</b>
<b>2.2</b>	<b>JavaScript</b>	<b>21</b>
2.2.1	A linguagem nos dias atuais	23
2.2.1.1	JavaScript além dos navegadores	23
2.2.1.2	ECMA	24
<b>2.3</b>	<b>Arquitetura</b>	<b>24</b>
2.3.1	Cliente	26
2.3.2	Serviço	27
2.3.3	Servidor	27
2.3.4	Banco de dados	28
<b>2.4</b>	<b>A pilha MEAN</b>	<b>28</b>
2.4.1	MongoDB	28
2.4.1.1	Mongoose	29
2.4.2	Express	30
2.4.3	Angular	30
2.4.4	Node.js	31
2.4.4.1	NPM	31
<b>3</b>	<b>PROJETO</b>	<b>33</b>
<b>3.1</b>	<b>Aplicação mobile</b>	<b>33</b>
<b>3.2</b>	<b>Servidor</b>	<b>33</b>
<b>3.3</b>	<b>Banco de Dados</b>	<b>33</b>
<b>4</b>	<b>CONCLUSÃO</b>	<b>35</b>
	<b>REFERÊNCIAS</b>	<b>37</b>
	<b>ANEXOS</b>	<b>39</b>
	<b>ANEXO A – PRIMEIROS PASSOS MONGO</b>	<b>41</b>
	<b>ANEXO B – PRIMEIROS PASSOS EXPRESS</b>	<b>43</b>

<b>B.1</b>	<b>Express Generator . . . . .</b>	<b>43</b>
	<b>ANEXO C – PRIMEIROS PASSOS ANGULAR CLI . . . . .</b>	<b>45</b>
	<b>ANEXO D – PRIMEIROS PASSOS NODE.JS . . . . .</b>	<b>47</b>

# 1 Introdução

Inicialmente implementada com o objetivo no desenvolvimento web para o lado do cliente, a linguagem criada por Brendan Eich enquanto trabalhou na Netscape se tornou uma das linguagens mais populares da atualidade, sendo a terceira camada do bolo quando se fala de tecnologias web, das quais HTML e CSS também fazem parte.

Nos últimos anos a linguagem JavaScript ganhou maior importância, com o surgimento de bibliotecas e *frameworks* que possibilitaram o desenvolvimento de não somente web sites mas também aplicativos, *single page applications*, programas *desktop*, *progressive web apps* e muito mais.

Talvez nenhuma outra linguagem tenha conseguido ganhar tanta atenção dos desenvolvedores como o JavaScript. Em busca de sua identidade a linguagem foi a única que conseguiu se enraizar nos navegadores, e atualmente também passou a se empoderar dos servidores de alta performance através do Node.js.

O surgimento do Node.js trouxe novas possibilidades para os desenvolvedores, levando a linguagem para um novo patamar. Criado com um modelo não bloqueante na entrada e saída de dados, possibilitou que a linguagem fosse levada agora também para aplicações *back-end*, ou seja, para o *server-side* (lado do servidor).

Trabalhar com JavaScript em diversas plataformas como *client-side* e *server-side* a tempos atras poderia ser considerado uma utopia. Graças ao crescimento da linguagem agora é possível a execução da linguagem nos mais diversos ambientes.

## 1.1 Objetivos

O trabalho tem como objetivo relatar a evolução da linguagem JavaScript e sua ampla utilização, para os mais diferentes tipos de sistemas, no contexto atual.

Não há soluções tecnológicas a serem resolvidas neste projeto, nele é apresentado informações e utilizado tecnologias já existentes, que se encaixem com a arquitetura apresentada e que compartilhem da mesma linguagem.



## 2 Revisão bibliográfica

### 2.1 Uma linguagem para todo o desenvolvimento

Por que não em vez de utilizar uma linguagem para o *client-side* e outra para o *server-side*, utilizar apenas uma linguagem para ambos? Ao utilizar a mesma linguagem em todas as camadas de uma aplicação, você traz vantagens para o projeto, equipe e até mesmo para o cliente final. Todo mundo ganha!

A vantagem mais clara que temos é a padronização, o ser humano tem uma enorme facilidade no reconhecimento de padrões, por tudo ser escrito na mesma linguagem a uniformidade é maior, tornando assim o trabalho mais compreensivo para os programadores envolvidos no projeto, seja ele *front-end*, *back-end* ou *full-stack*.

Se um programador compreende bem a linguagem ele será capaz também de desenvolver sobre todas as camadas, reduzindo o tempo ocioso de equipes que dependem do trabalho de outras equipes. Isso faz com que times *front-end* e *back-end* se tornem times *full-stack* sendo assim mais versáteis, ágeis e que compreendem melhor o produto como um todo, trazendo assim benefícios também a empresa em suas contratações.

Outra vantagem a se considerar é a possibilidade de reaproveitar código. Reutilizar trechos de código é uma forma de reduzir a carga de trabalho dos desenvolvedores e acelerar o desenvolvimento. Além de objetos serem bem parecidos, se não idênticos, facilitando muito o debug e a comunicação entre aplicações e serviços.

Também há benefícios para o desenvolvedor, uma vez que conhecer uma linguagem que pode ser um elemento comum a vários *frameworks* e de base para o desenvolvimento em qualquer ambiente.

Com todas essas vantagens atingimos também diretamente o cliente, que precisa esperar menos para seu projeto ficar pronto e consequentemente reduz o custo final do desenvolvimento.

A abordagem de se utilizar uma linguagem para todo o desenvolvimento, vem sendo muito utilizada em hackathons e competições curtas de programação, onde deve-se perder o menor tempo possível e com isso criar protótipos de forma rápida.

### 2.2 JavaScript

As muitas vantagens de se utilizar o JavaScript fizeram com que ela se tornasse a linguagem de programação mais popular do mundo, desbancando até mesmo linguagens

de maior expressão como Java, C#, PHP, Python e outras.

JavaScript é uma linguagem de programação dinâmica interpretada, inicialmente utilizada pelos navegadores para execução de *scripts* no lado do cliente, ou seja, no seu *browser*. Os *scripts* são incorporados a páginas HTML tendo como função adicionar interatividade para o usuário.

Atualmente, é impossível imaginar a internet sem a existência do JavaScript, certamente você já se deparou com alguns dos exemplos como os famosos carrosséis (Figura 1, página 25), as tão importantes validações de formulários (Figura 2, página 25), ou até mesmo com um *modal* (Figura 3, página 25) que mostram algumas das aplicações da linguagem em *web sites*.

O JavaScript apresenta uma sintaxe simples que facilita o aprendizado, mas não confunda pois a primeira vista muitos desenvolvedores podem acreditar que a linguagem é defeituosa ou esquisita, pois não compreendem o real poder que se esconde por trás desta simplicidade.

Inicialmente classificada como linguagem do tipo *client side*, é por si só uma linguagem compacta, mas muito flexível, com comportamentos diferenciados das demais ela permite, por exemplo, que um objeto tenha seus atributos adicionados ou removidos em tempo de execução, o que não é muito comum para desenvolvedores de outras linguagens.

É uma linguagem interpretada, pois seus comandos são executados sem que haja necessidade de compilação, tendo como interpretador de *script* o *browser* do usuário. Sendo assim independente de plataformas, como os comandos são interpretados pelo navegador do usuário, é irrelevante se o usuário está utilizando Windows, Linux ou Mac OS.

Sua tipagem é dinâmica, ou seja, tipos são associados com valores. Por exemplo, uma variável pode ser associada a um número e posteriormente associada a um texto.

Com JavaScript também é possível a detecção de eventos, sempre que algo de importante acontece é disparado um evento, o clique de um botão, o preenchimento de um campo de formulário, a movimentação do mouse, são alguns exemplos dos eventos que são disparados. Isso nos permite reagir a estes eventos deixando assim que nossa aplicação deixe de ser estática.

Podemos executar códigos JS de várias formas na web, sendo uma delas pelo próprio console do navegador pressionando as teclas F12, importando um script em uma página HTML, ou até mesmo envolvendo o trecho de código na tag `<script>`.

No início da internet as páginas não eram nada interativas, documentos apresentavam seu conteúdo exatamente como foram criados para serem exibidos no navegador e só. O JavaScript revolucionou o que podemos fazer, hoje não só na web, mas praticamente em todas as áreas que se possa programar.



## 2.2.1 A linguagem nos dias atuais

Se existe alguma linguagem que evoluiu nos últimos tempos, essa linguagem é o JavaScript. Conforme a linguagem evoluiu se tornou mais poderosa e independente do navegador. Isso possibilitou que a linguagem fosse utilizada não somente para a web como *client side* mas agora em vários lugares.

### 2.2.1.1 JavaScript além dos navegadores

Inicialmente tratada como um extra para os navegadores, podemos dizer que a linguagem caminhou com o avanço da tecnologia, isso possibilitou o uso da linguagem em diversas áreas sendo algumas delas:

- a) Aplicações web
- b) Aplicativos mobile
- c) Automação de testes e de tarefas
- d) Controle de hardware
- e) Desenvolvimento de jogos
- f) Internet das coisas
- g) Realidade virtual e aumentada
- h) Softwares desktop
- i) Servidores

Praticamente tudo que envolve programação o JavaScript está presente, para criar um software desktop existe por exemplo o *framework* Electron <sup>1</sup>, desenvolvido pelo GitHub ele possibilita criar aplicativos desktop multiplataforma através do JavaScript. No desenvolvimento de jogos a *engine* Unity <sup>2</sup> é uma das plataformas que oferecem suporte a linguagem.

O ecossistema JavaScript é gigante e tem atraído empresas de todos os portes. Hoje grandes empresas como Google, Microsoft, Netflix, Uber e LinkedIn usam JavaScript no *back-end*.

Isso acabou impactando o mundo dos desenvolvedores, fazendo com que seja obrigatório todo programador saber pelo menos o básico da linguagem, mesmo atuando em outras áreas, como *back-end* e até mesmo teste de *softwares*.

<sup>1</sup> <<https://electronjs.org>><https://electronjs.org>

<sup>2</sup> <<https://unity3d.com/pt>><https://unity3d.com/pt>

### 2.2.1.2 ECMA

Pela linguagem rodar em ambientes que podem variar, algo importante a considerar é a compatibilidade entre os navegadores. Para isso é necessário um padrão, tal criado e mantido até hoje pelo ECMA Internacional.

Já em 1996 a Netscape, detentora do JavaScript, anunciou que submetia a linguagem para o ECMA Internacional como candidata a padrão industrial, resultando então no ECMAScript.

Padronização que define a estrutura da linguagem, seus comportamentos e comandos, dando assim um padrão aos interpretadores da linguagem.

Como vimos, as vantagens do JavaScript vão muito além de ser a linguagem mais popular da web, ou de até mesmo possuir a maior comunidade de programadores. Depois de 2015, a partir do ES6 (ECMAScript 6) fez com que a linguagem passa-se por uma série de mudanças essenciais que fizeram com que a linguagem se consolidasse e desde então não parou de evoluir.

Atualmente a linguagem se encontra na 9ª edição chamada de ECMAScript 2018, finalizada em Junho de 2018.

Com participação colaborativa de empresas que implementam o *run-time* da linguagem, como Mozilla, Google, Microsoft e Apple, além da participação de desenvolvedores da comunidade, o ECMA coordena e faz o trabalho de desenvolvimento contínuo e descentralizado do JS.

No site oficial <sup>3</sup> a ECMA disponibiliza informações como versão atual e a documentação da linguagem, porém aconselho a buscar documentações na internet, uma boa referência a seguir é o site da Mozilla <sup>4</sup>, ou também o site da W3Schools <sup>5</sup>.

A 10ª edição já está em desenvolvimento, devendo chegar até o final de Junho no ano de 2019, sendo chamada de ECMAScript 2019.

## 2.3 Arquitetura

A arquitetura de um software consiste na definição de seus componentes, relacionamentos com softwares externos e suas propriedades. Na arquitetura multicamadas temos o famoso Cliente-Servidor onde as camadas de apresentação, processamento de aplicativos e gerenciamento de dados são separados.

Com este modelo de arquitetura isolamos funções da aplicação de forma modular, facilitando sua manutenção e depuração, além de criar uma hierarquia de níveis de acesso,

<sup>3</sup> <<http://www.ecma-international.org/publications/standards/Ecma-262.htm>>

<sup>4</sup> <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>

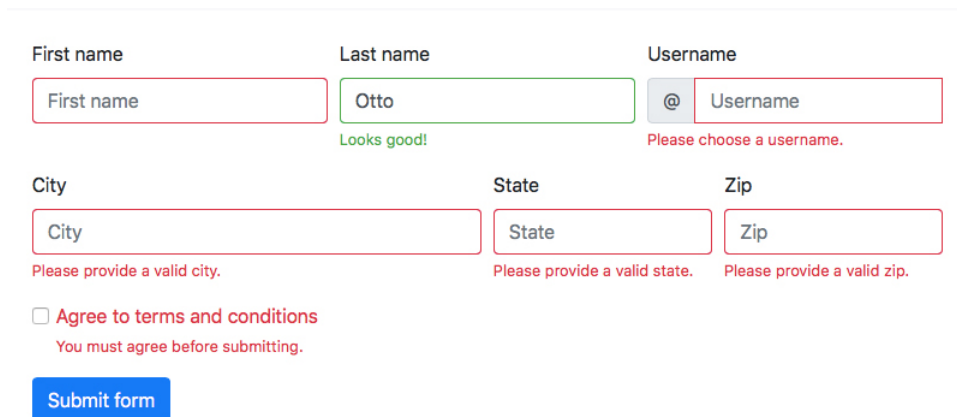
<sup>5</sup> <<https://www.w3schools.com/jsref/default.asp>>

Figura 1 – *Carousel* - Um componente de apresentação de slides para percorrer imagens ou slides de texto - como um carrossel. – Exemplo de aplicação do JS



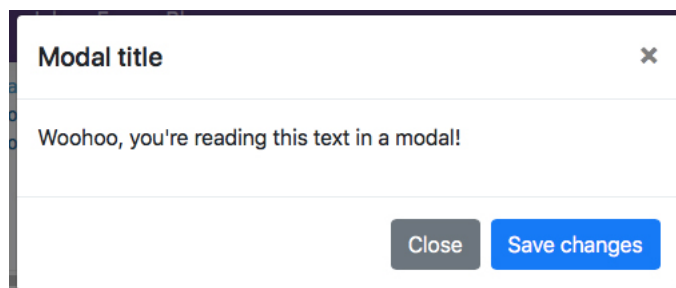
Fonte: Bootstrap <sup>6</sup>

Figura 2 – *Validation* - Validação de formulários – Exemplo de aplicação do JS



Fonte: Bootstrap <sup>7</sup>

Figura 3 – *Modal* - Caixas de diálogo para notificações ao usuário. – Exemplo de aplicação do JS



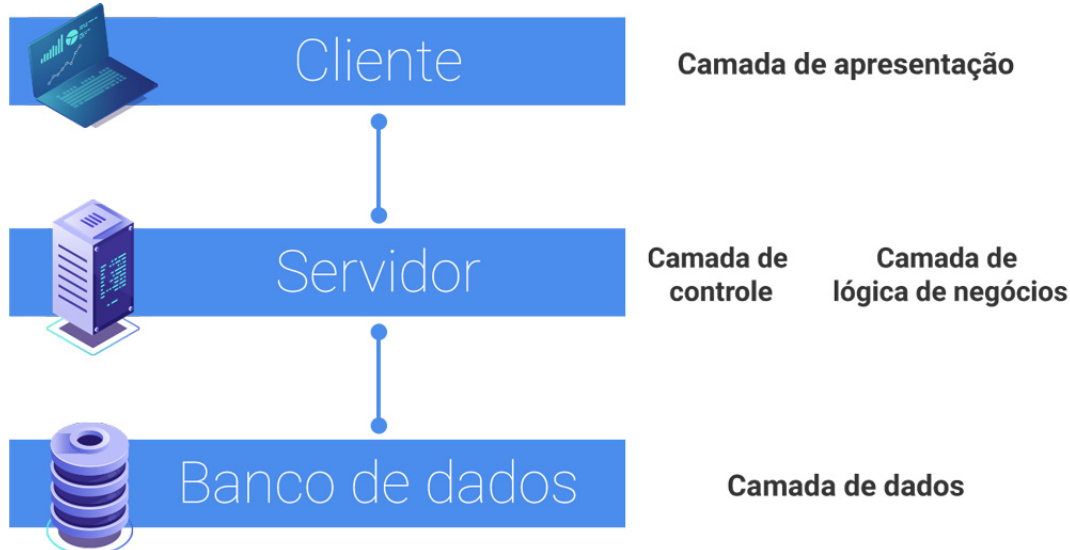
Fonte: Bootstrap <sup>8</sup>

protegendo as camadas mais internas.

Também conhecidas como stacks, as camadas de uma aplicação são basicamente o *front-end* (cliente) e *back-end* (servidor). Camadas são uma maneira de separar responsabilidades, uma camada superior pode usar serviços de uma camada inferior, mas não o oposto.

A modularização refere-se a separar a lógica de negócios e regras de acesso ao banco de dados da camada de apresentação.

Figura 4 – Exemplo de arquitetura multicamadas



Na parte inferior da figura 4, está localizado o servidor de banco de dados, o qual se comunica com os servidores de aplicação através de protocolos de rede. Já na parte superior estão os clientes, que fazem a comunicação com a camada intermediária através da utilização de interfaces. Este é basicamente o esquema de comunicação desta arquitetura.

A arquitetura apresentada foi escolhida por criar um cenário de aplicação que exemplifica bem as separações das responsabilidades, podendo assim mostrar a aplicação de apenas uma linguagem em todos eles.

### 2.3.1 Cliente

O **cliente** representa a **camada de apresentação**, ela é responsável pela interação com o usuário além de se comunicar com a camada de controle e ter o dever de garantir a consistência entre o dado e sua apresentação.

Para o cliente pouco importa as regras de negócio, uma vez que seu papel é enviar informações e espera receber os dados processados, além de ser responsável por executar lógica no lado do cliente. Aqui encontramos a parte da aplicação *client-side*, em ambientes web também conhecida como *front-end*.

Um benefício da modularização da arquitetura multicamadas nos possibilita é a criação de várias aplicações clientes que podem compartilhar as mesmas regras, pois ficam encapsuladas em uma camada de acesso comum. Por exemplo, posso ter um aplicativo e um web-site consumindo um mesmo serviço, compartilhando assim das mesmas regras.

Eliminando problemas que podem ocorrer no controle de versão em outros tipos de

arquitetura, pois em uma arquitetura diferente da modular, se determinado usuário possui uma versão mais antiga do que outro, pode ocorrer erros de dados lógicos no processamento das regras de negócios, uma vez que não compartilham da mesma regra.

### 2.3.2 Serviço

O **serviço** serve como **camada de controle**, onde recebe as requisições e comanda o fluxo servindo como uma camada intermediária entre a camada de apresentação e a lógica. De forma que o banco de dados e o servidor podem estar fisicamente distantes da aplicação cliente.

Normalmente a camada de controle costuma trabalhar junto da camada de lógica de negócios, ou seja o serviço trabalha junto com o servidor, resumidos a apenas uma camada.

Uma das formas de fazer essa comunicação é através de protocolos, como o *HTTP* que através de um modelo, como por exemplo *REST* nos permite realizar essa comunicação entre camadas.

*REST* é um modelo descrito por Roy Fielding, um dos principais criadores do protocolo *HTTP*, em sua tese de doutorado e que foi adotado como o modelo a ser utilizado na evolução da arquitetura do protocolo *HTTP*, porém muitos desenvolvedores perceberam de que ele poderia também ser utilizado na implementação de *Web Services*.

O modelo *REST* utiliza como gerenciamento de informações os chamados recursos, que nada mais são do que uma abstração sobre um determinado tipo de informação que uma aplicação gerencia, sendo que um dos princípios é de que todo recurso deve possuir uma identificação única, também conhecidos como *URI*.

### 2.3.3 Servidor

Podemos considerar o **servidor** como a **camada de lógicas de negócios**. Nela recebemos requisições devolvendo informações que podem ser processadas por esta própria camada ou retornar informações através da camada de acesso a dados. Aqui está a parte da aplicação *server-side*, em ambientes web também conhecida como *back-end*.

Modularizar a lógica de negócios também traz benefícios, por os clientes acessarem uma mesma camada em comum (servidor), qualquer alteração realizada nas regras de negócios serão vistas por todas as aplicações clientes.

Também nos possibilita escalabilidade, com a utilização de outras arquiteturas, é comum que ocorra uma queda de desempenho quando um grande número de máquinas clientes simultâneas se conectam ao servidor. Sendo isso evitado com a arquitetura modularizada, uma vez que é possível ter a mesma regra de negócio dividida entre

vários servidores através do balanceamento de carga, ou seja, quando algum deles ficar sobrecarregado o outro entra em ação para ajudá-lo.

### 2.3.4 Banco de dados

O **banco de dados** representa a **camada de dados**, pois nele estão as técnicas de persistência de dados que são expostos para camadas superiores, como por exemplo a camada de lógica de negócios. São de vital importância para empresas e hoje é considerado a principal peça dos sistemas da informação.

Seu principal objetivo é armazenar dados que quando relacionados criando algum sentido são chamados de informações. Operado por um Sistema Gerenciador de Banco de Dados (SGBD), executa comandos em linguagens, como por exemplo o SQL, com o objetivo de retirar da aplicação cliente a responsabilidade de gerenciar o acesso, a persistência, a manipulação e a organização dos dados.

## 2.4 A pilha MEAN

Como visto anteriormente no capítulo 2.1 é bem atraente a possibilidade de usar apenas uma linguagem em todo o desenvolvimento, ganhando não só reaproveitamento de recursos humanos como de código, sendo assim possível criar códigos que podem ser executados em qualquer plataforma que interprete JavaScript, isso é o que acontece com o MEAN.

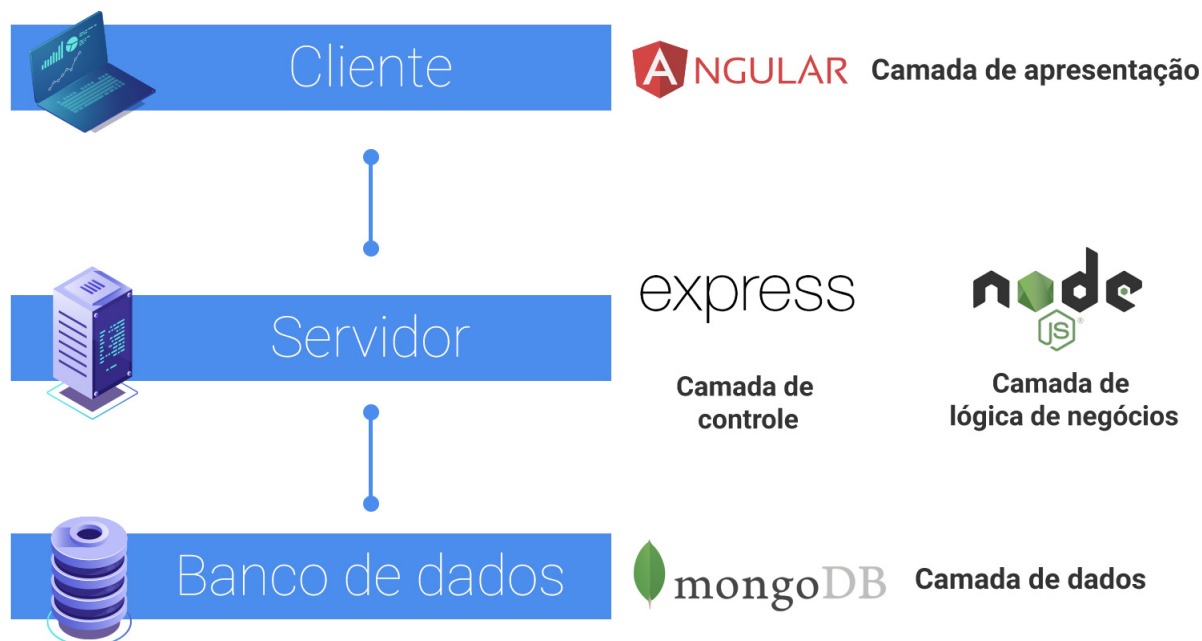
Das iniciais das ferramentas MongoDB, Express, Angular e Node.js, MEAN é o nome dado quando integramos todas essas ferramentas para desenvolver uma aplicação. Sendo que todas utilizam como linguagem o JavaScript, possibilitando que um programador da linguagem tenha maior facilidade para trabalhar em todas as partes da aplicação, seja *front-end*, *back-end* ou banco de dados.

A figura 5 ilustra bem como a MEAN Stack segue o modelo de arquitetura em multicamadas apresentado no capítulo 2.3, apenas aplicando as tecnologias nas camadas exibidas na figura 4.

Por utilizar quatro tecnologias distintas, podemos pensar que dará mais trabalho no desenvolvimento, mas ao contrário disso a MEAN Stack é utilizada até mesmo em curtas competições de programação conhecidas como *Hackathon*, conseguindo protótipos de forma rápida e provando assim sua produtividade.

### 2.4.1 MongoDB

Um banco de dados flexível, poderoso, escalonável e de alta performance orientado a documentos. Lançado em 2009, escrito em C++, o MongoDB é gratuito e de código

Figura 5 – *MEAN Stack* - Fluxo das ferramentas que integram o MEAN

aberto.

Por ser orientado à documentos JSON, ou seja, retém os dados usando pares de chave/valor, podemos modelar dados de forma mais natural, utilizando a forma como os dados realmente serão utilizados em nossa aplicação, ao invés de criar várias ligações entre tabelas, o que dá a característica ao MongoDB de banco não-relacional.

Para a execução de comandos no Mongo, existe um console que executa códigos JavaScript. Por esse motivo, desenvolvedores da linguagem terão facilidade em manter um banco MongoDB.

Com ele é possível receber a entrada no navegador com JavaScript, transportar a demanda para o *back-end* ainda com a linguagem e, por fim, salvar um objeto no MongoDB ainda com o JavaScript. Naturalmente, o fluxo reverso também será tão prático quanto.

Em uma aplicação desenvolvida em cima do MEAN Stack o Mongo tem a responsabilidade de persistir os dados, ou seja, permite armazenar e recuperar dados. Ele representa na arquitetura multicamadas a camada de dados, como visto anteriormente no capítulo 2.3.4.

#### 2.4.1.1 Mongoose

Com o Mongo é possível inserir qualquer formato JSON nas coleções, por esse motivo não há nenhum controle sobre os dados inseridos. Isso pode causar inconsistência nos dados, inserindo informações inválidas podendo quebrar a aplicação, deixando a responsabilidade para o dev garantir a exatidão das informações.

Para resolver esse problema existe o Mongoose que nos ajuda a modelar objetos de

forma elegante para o MongoDB, ele fornece uma solução direta e baseada em esquemas para modelar os dados da nossa aplicação, incluindo conversão de tipo incorporada, validação, criação de consulta, ganchos de lógica de negócios e muito mais.

### 2.4.2 Express

Express é um *framework* web rápido, flexível e minimalista para Node.js inspirado no Sinatra, um *framework* para Ruby. Ele facilita o desenvolvimento de aplicações web e *APIs*, tanto pequenas quanto mais robustas, tornando fácil escalar aplicações criadas com ele.

Com um conjunto de métodos utilitários *HTTP* e *middlewares* a seu dispor, criar uma *API* robusta utilizando Express é rápido e fácil.

Na MEAN Stack, o Express tem a responsabilidade de disponibilizar endpoints *REST*, que serão consumidos pela aplicação cliente. Ele representa na arquitetura multicamadas a camada de controle, e também como camada de lógica de negócios como visto anteriormente nos capítulos 2.3.2 e 2.3.3.

Através da figura 5 percebemos que o Express e Node.js fazem parte da mesma camada, o servidor.

Com o Express podemos criar nossa *REST API*, configurando funções a serem executadas após o recebimento de chamadas em rotas, também conhecidas como URIs, pelos métodos *HTTP GET, POST, PUT, PATH* e *DELETE*.

### 2.4.3 Angular

Angular é um *framework* JavaScript de código aberto, mantido pela Google. E tem como princípio "*One framework. Mobile and desktop*", que basicamente significa utilizarmos o Angular para criar nossa aplicação web e também aplicações para dispositivos móveis.

O *framework* utiliza TypeScript como linguagem, isso pode nos ajudar a manter melhor um projeto grande, já que a tipagem de variáveis nos força a sempre utilizar uma variável ou objeto do mesmo tipo, dando assim de certa forma uma garantia de que não teremos problemas ao acessar algum dado. Mas no final todo o código é compilado e transformado em JavaScript.

Assim como o React o novo Angular trabalha com componentes, que são trechos de códigos que definem elementos com aparência e comportamentos da interface. A ideia é que componentes sejam genéricos para que sejam utilizados e principalmente reutilizados em qualquer lugar da aplicação.

Na MEAN Stack, o Angular é responsável pela aplicação do usuário, ele cria interfaces dinâmicas sem manipulação direta no DOM, permite execução de lógica no *client-*



*side* além de facilitar a troca de dados *REST*. Ele representa na arquitetura multicamadas a camada de apresentação, como visto anteriormente no capítulo 2.3.1.

Perceba que o Angular é apenas uma tecnologia que faz o papel de cliente na arquitetura, aqui poderíamos adotar qualquer outra ferramenta ou tecnologia, ou até mesmo diferentes tecnologias que consumissem o mesmo serviço, possibilitando assim criar infinitas aplicações clientes, como por exemplo um aplicativo para *mobile* e outra aplicação mas voltada para web.

#### 2.4.4 Node.js

O Node.js é uma plataforma de desenvolvimento construída sobre a linguagem JavaScript, por sua natureza assíncrona, executada pelo interpretador V8 criado pela Google e utilizado no Google Chrome, focado em migrar a linguagem JS para servidores. Foi criado pensando em um modelo não bloqueante para as operações de entrada e saída de dados (I/O - *Input and Output*).

Neste modelo, as operações de I/O não bloqueiam o atendimento aos outros clientes, ou seja, quando são feitas operações como uma leitura no disco ou consulta de banco de dados, as requisições de outros clientes vão sendo enfileiradas. Após o processamento ser finalizado e respondido ao primeiro cliente, o próximo cliente é atendido.

Na MEAN Stack, o Node é o core pois ele é nosso ambiente de execução, que além de manter tudo disponível, contém as regras de negócio da nossa aplicação *back-end*, também conhecida como *server-side*. Ele representa na arquitetura multicamadas a camada de lógica de negócios, que trabalhando junto do Express também serve como camada de controle como visto anteriormente nos capítulos 2.3.3 e 2.3.2.

##### 2.4.4.1 NPM

O *Node Package Manager*, mais conhecido como npm é o gerenciador de pacotes do Node.js. Com ele podemos fazer download de códigos, bibliotecas e *frameworks* que nossos projetos podem usar, ou até mesmo ferramentas que exerçam alguma função em nosso sistema operacional.

O arquivo `package.json` é responsável por gerenciar os pacotes cujo nosso projeto depende. Ele basicamente serve como uma lista para as dependências que nosso projeto necessita. Executando o comando do node `npm install` automaticamente todas as dependências listadas serão baixadas.

Isso também nos ajuda a padronizar as versões de nossas dependências, facilitando informar os requisitos mínimos e atualizações, garantindo assim que todos os desenvolvedores terão a mesma versão em suas máquinas. Outra vantagem é para que outro desenvolvedor tenha acesso a nosso código, basta enviar o código em si, sem se preocupar

com as dependências, pois basta ele executar o comando do node `npm install` para fazer o download.

## 3 Projeto

O projeto desenvolvido tem como objetivo abordar o assunto sobre informações ambientais e o desenvolvimento de uma aplicação que facilite o dia a dia de uma pessoa com o gerenciamento de irrigações de uma pequena plantação, ou apenas a grama de um jardim de forma sustentável.

A solução dos problemas ambientais tem sido considerada cada vez mais urgente para garantir o futuro da humanidade, e como hoje existem meios que facilitem o cuidado com o ambiente e a diminuição de desperdício, uma das formas de unirmos o cuidado com a diminuição de desperdício é desenvolvendo uma aplicação que tenha essa funcionalidade.

Para o desenvolvimento do projeto foi utilizado a linguagem JavaScript em conjunto dos *frameworks* que compõem a MEAN Stack, seguindo a arquitetura apresentada no Capítulo 2.3.

Foi utilizado contêineres da tecnologia Docker para iniciar e gerenciar as camadas da aplicação. Um contêiner para cada responsabilidade, um para a Aplicação mobile, um para o Servidor e outro para o Banco de Dados, totalizando 3 contêineres. Sendo possível um maior controle por funcionalidade, podendo gerir e escalar em unidades, de forma que não influencie nos outros contêineres.

### 3.1 Aplicação mobile

### 3.2 Servidor

### 3.3 Banco de Dados



## 4 Conclusão

Foi apresentado os benefícios na abordagem de utilizar apenas uma linguagem, no caso o JavaScript, em todas as camadas do desenvolvimento. Através da arquitetura multicamadas aplicamos a MEAN Stack, relacionando uma ferramenta para cada camada, mostrando ser capaz utilizar a linguagem nos mais diversos ambientes.

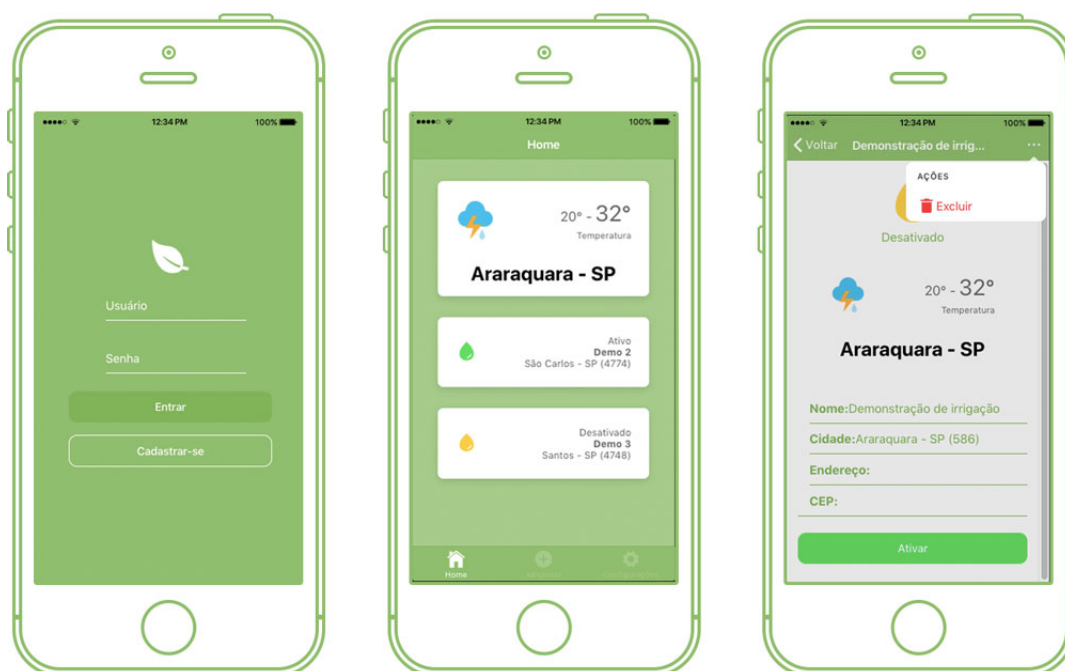
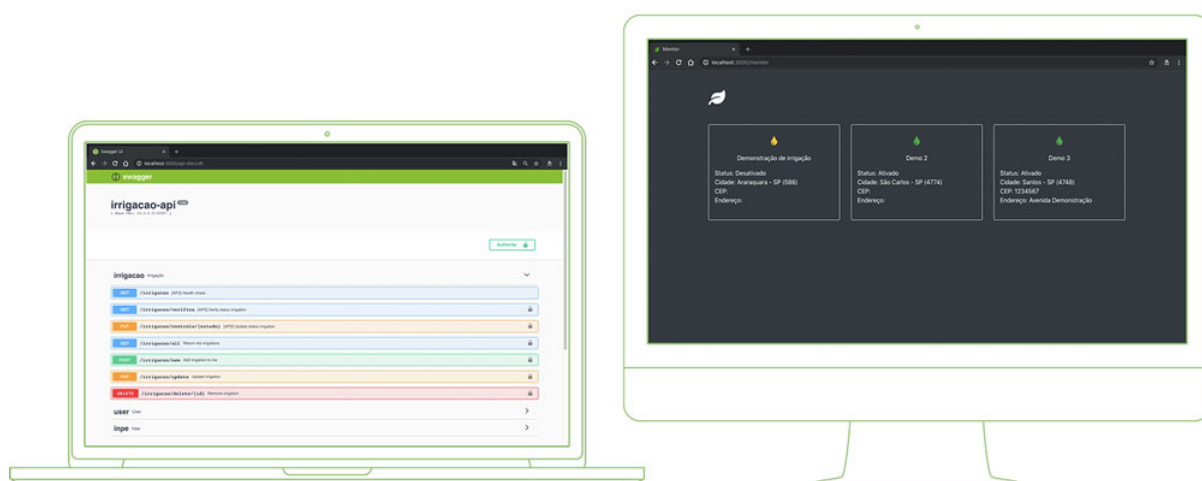
O JavaScript evolui a cada dia que passa, se eu pudesse apostar em uma linguagem pro futuro certamente escolheria o JS. Ao mesmo tempo que é uma linguagem de fácil aprendizado é uma linguagem forte e consolidada. Assim foi relatado a evolução da linguagem e sua ampla utilização, para os mais diferentes tipos de sistemas, no contexto atual.

Também foi criado uma aplicação (Figuras 6 e 7) para mostrar na prática os conceitos aqui apresentados, e todo o código foi disponibilizado via GitHub, com todas as informações de licença e de como executar, através do link <https://github.com/rodrigocichetto/tcc>.

A aplicação desenvolvida teve também como objetivo abordar o assunto ambiental, visando facilitar o dia a dia de uma pessoa com os cuidados de uma plantação, ou apenas a grama de um jardim, de uma forma mais sustentável.

A solução dos problemas ambientais tem sido considerada cada vez mais urgente para garantir o futuro da humanidade, e como hoje temos meios que facilitem o cuidado com o meio ambiente e a diminuição de desperdício, uma das formas de unirmos o cuidado com a diminuição de desperdício é desenvolvendo uma aplicação que tenha tais funcionalidades.

Figura 6 – Exemplo de aplicação cliente criada com Angular para dispositivos móveis

Figura 7 – Exemplo de serviço criado com *Node.JS*, *Express* consultando *MongoDB*, expõe a documentação e monitoramento em tempo real dos dados

## Referências





## Anexos



## ANEXO A – Primeiros passos Mongo

Por padrão o terminal do Mongo inicia conectado ao banco de dados `test`. Para mudarmos para outro banco de dados utilizamos o comando `use nome_do_banco`. Caso o banco indicado não exista, o Mongo criará um novo assim que dados forem incluídos nele.

*Insert*, através da função `insert()` insere novos dados e pode receber como parâmetro um objeto JSON ou um array de objetos.

```
1 // Exemplo insert
2 db.nome_do_banco.insert()
```

*Find*, através da função `find()`, busca por objetos inseridos em uma *collection*, podendo receber como parâmetro um objeto com os critérios de filtragem, ou nenhum para retornar todos os objetos.

```
1 // Exemplo find
2 db.nome_do_banco.find()
```

*Update*, através da função `update()` atualiza os dados, recebendo dois parâmetros, sendo o primeiro a condição para achar o documento, e o segundo o novo documento.

```
1 // Exemplo update
2 db.nome_do_banco.update()
```

*Remove*, através da função `remove()` remove os dados, podendo passar um parâmetro para informar qual objeto remover ou nenhum para remover todos, caso esse seja seu objetivo, também é possível utilizar o comando `drop()`.

```
1 // Exemplo remove
2 db.nome_do_banco.remove()
3 // Exemplo drop
4 db.nome_do_banco.drop()
```



## ANEXO B – Primeiros passos Express

Para criarmos um servidor com o Express, precisamos primeiro do Node.js e o NPM instalados, caso não saiba como instalá-los veja a explicação no capítulo ??.

No terminal inicie um projeto node a partir do comando `npm init`, preencha as informações e então execute o comando `npm install express` dentro do diretório do seu projeto para instalar o express. Desta maneira conseguimos importar o express para nosso código. Finalmente é só criar um arquivo `app.js` com o código abaixo.

```
1 const express = require('express');
2 const app = express();
3 const PORT = 3000;
4
5 app.get('/', (req, res) => {
6   res.send('Hello World');
7 });
8
9 app.listen(PORT, () => {
10   console.log('Servidor disponivel na porta ' + PORT);
11 });
```

Para iniciar o servidor basta executar o comando `node app.js`, enfim abra seu navegador no endereço <http://localhost:3000> ele deve retornar nosso *Hello World*.

### B.1 Express Generator

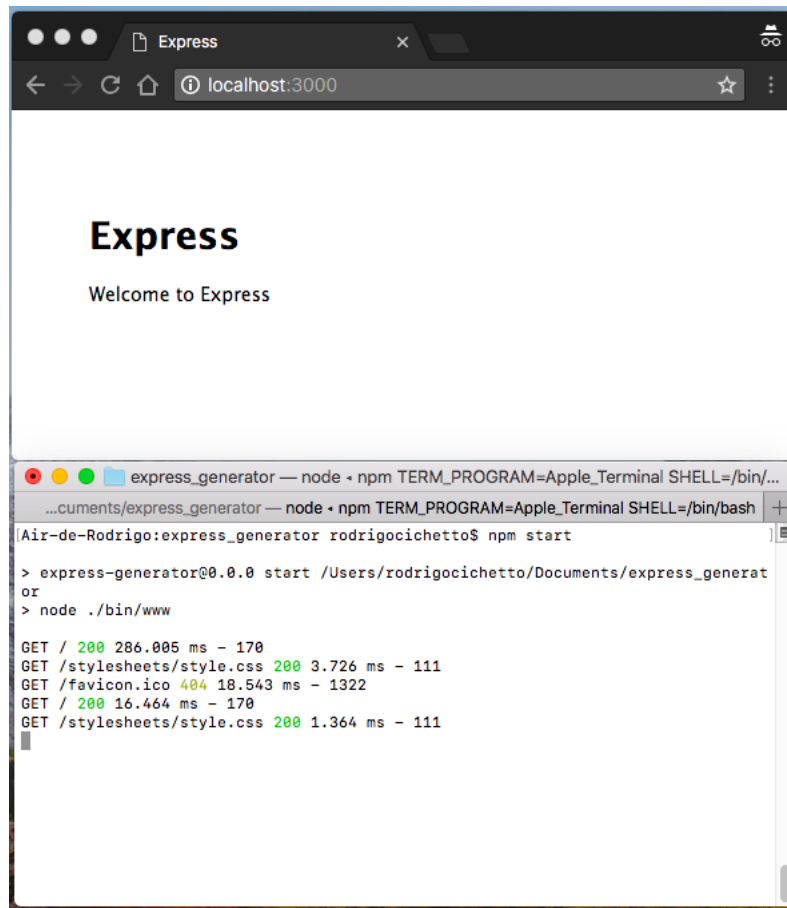
Com apenas o comando `express nome_da_app` do Express Generator ele já estrutura uma aplicação Express simples. Após a execução do comando ele irá criar um novo diretório com o nome da aplicação informado anteriormente, contendo arquivos e dependências necessárias.

Antes de iniciarmos o servidor Express devemos instalar as dependências através do comando `npm install` que irá baixar as dependências de acordo com o arquivo `package.json` gerado anteriormente. Enfim é hora de iniciar a aplicação executando o comando `npm start` e acessá-la pelo endereço <http://localhost:3000>.

Essa página só é exibida porque no arquivo `routes/index.js` está configurado a rota padrão para que renderize o arquivo `views/index.jade` passando o título com o valor Express, sendo interpretado pela *template engine* JADE, configurada por padrão pelo Express Generator.

Para saber mais opções de geração de estruturas através do Express Generator,

Figura 8 – Aplicação gerada pelo Express Generator



execute o comando `express -h`.

## ANEXO C – Primeiros passos Angular CLI

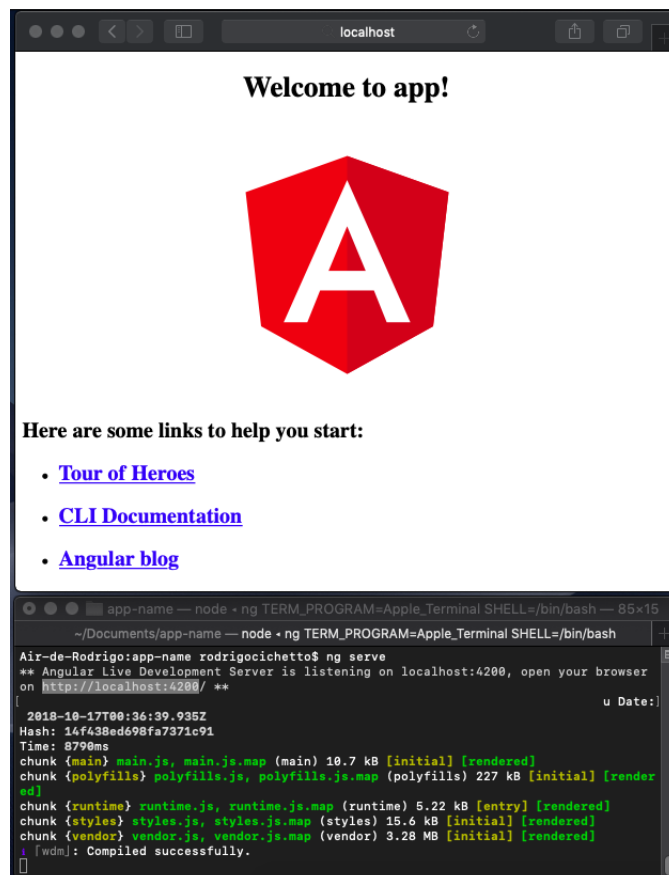
Instalado a partir do comando do node `npm install -g @angular/cli`, o Angular CLI é uma interface de linha de comando para o Angular, que nos permite trabalhar de forma mais amistosa com a tecnologia.

Com o Angular CLI é possível criar projetos e estruturá-los a partir do comando `ng`. Por exemplo, para criar um novo projeto, basta executar o comando `ng new app-name`, assim não precisamos nos preocupar com a estrutura inicial do projeto e suas dependências cruciais para o funcionamento do Angular.

Para iniciar o servidor com o Angular, basta executar o comando `ng serve` e abrir seu navegador no endereço <http://localhost:4200>.

Ainda é possível criar componentes, pipes, módulos, diretivas e serviços de forma rápida com o comando `ng generate`.

Figura 9 – Aplicação gerada pelo Angular Cli







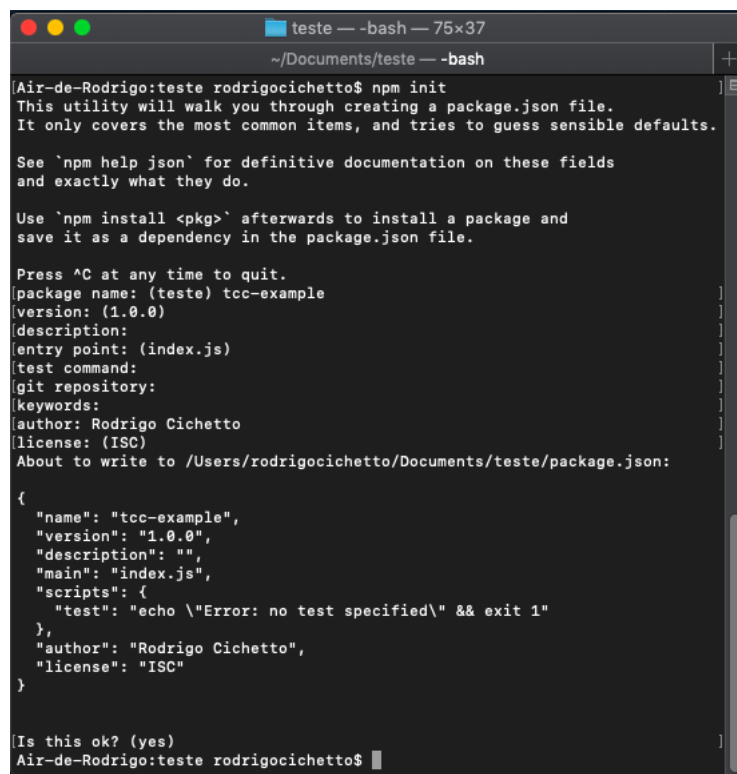
## ANEXO D – Primeiros passos Node.js

Instalar o Node.js é muito simples, basta acessar o link <https://nodejs.org/en/download/> e baixar o instalador ou seguir as instruções de instalação de acordo com o seu sistema operacional.

Para iniciar um projeto em Node.js devemos primeiro criar um diretório, em seguida acessá-lo via terminal, enfim digitamos o comando `npm init` e preenchemos as informações de acordo com nosso projeto. Perceba que ele gerou o arquivo `package.json` com as informações preenchidas no comando via terminal.

Agora é só começar a desenvolver e instalar suas dependências, que devem ficar salvas no arquivo gerado anteriormente.

Figura 10 – Exemplo de configurações preenchidas no comando `npm init`



```
teste — -bash — 75x37
~/Documents/teste — -bash
Air-de-Rodrigo:teste rodrigocichetto$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (teste) tcc-example
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: Rodrigo Cichetto
license: (ISC)
About to write to /Users/rodrigocichetto/Documents/teste/package.json:
{
  "name": "tcc-example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Rodrigo Cichetto",
  "license": "ISC"
}

Is this ok? (yes)
Air-de-Rodrigo:teste rodrigocichetto$
```