Sumário

[Introdução à Angular 3](#_Toc436923386)

[- Google Chrome 3](#_Toc436923387)

[- Batarang 3](#_Toc436923388)

[AngularJS 3](#_Toc436923389)

[- ng-app 3](#_Toc436923390)

[- ng-model 3](#_Toc436923391)

[- declarações 3](#_Toc436923392)

[- Hello world simples 4](#_Toc436923393)

[- Data-Binding 4](#_Toc436923394)

[- ng-controllers 4](#_Toc436923395)

[- Lista de tarefas 5](#_Toc436923396)

[HTML 5](#_Toc436923397)

[CSS 5](#_Toc436923398)

[Javascript 6](#_Toc436923399)

[Como utilizar na plataforma 8](#_Toc436923400)

[- HELLO World 8](#_Toc436923401)

[Index.html 8](#_Toc436923402)

[Templates 10](#_Toc436923403)

[Javascript 11](#_Toc436923404)

[- app-page-builder.js 11](#_Toc436923405)

[- UserLog 13](#_Toc436923406)

[- Hello World com Verificação de Log 14](#_Toc436923407)

[- helloworld2.html 14](#_Toc436923408)

[-helloworld3.html 15](#_Toc436923409)

[- userlog.js 15](#_Toc436923410)

[Teste Unitários 21](#_Toc436923411)

[Sobre NodeJS 21](#_Toc436923412)

[Instalação 21](#_Toc436923413)

[Node.JS 21](#_Toc436923414)

[Python 28](#_Toc436923415)

[Visual Studio 34](#_Toc436923416)

[Testes unitários - TDD 37](#_Toc436923417)

[Instalando o *Karma-jasmine* 37](#_Toc436923418)

[Iniciando o projeto 37](#_Toc436923419)

[Separando corretamente as Pastas 44](#_Toc436923420)

[Iniciando o *Karma* para testes 48](#_Toc436923421)

# Introdução à Angular

HTML é ótimo para declarar documentos estáticos , mas limitado quando tentamos usá-lo para declarar visualizações dinâmicas em aplicações web . Angularjs permite estender vocabulário HTML para a sua aplicação . O ambiente resultante é extraordinariamente expressivo , legível e rápido para se desenvolver.

Outros framworkers lidam com as deficiências do HTML abstraindo o HTML , CSS e/ou JavaScript, ou fornecendo uma maneira imperativa para manipular o DOM. Nenhum destes resolver a raiz do problema que o HTML não foi projetado para exibições dinâmicas .

Angularjs é um conjunto de ferramentas para construir um framework mais adequado ao seu desenvolvimento de aplicativos. É totalmente extensível e funciona bem com outras bibliotecas . Cada recurso pode ser modificado ou substituído para atender às suas necessidades de fluxo de trabalho de desenvolvimento e de característica exclusivas .

Mais informações no site <https://angularjs.org/>

### - Google Chrome

Apesar de não haver problemas em usar outros navegadores, o Chrome se destaca por haver melhor suporte ao Angular, facilitando o entendimento das mensagens no console, e trazendo apenas as mensagens relevantes, além também de ter otimas extensões para inspecionar os elementos da aplicação Angular

### - Batarang

Existem outras extensões boas, mas essa se tornou relevante por ser mais simples e direta no seu uso, principalmente para verificar o escopo do elementos

## AngularJS

### - ng-app

È uma propiedade que pode ser inserida a qualquer momento em um documento HTML, ela indica a porção do documento será manipulado pelo AngularJS

### - ng-model

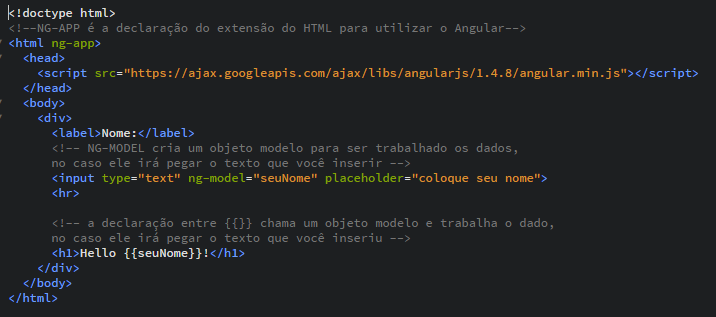
Normalmente utilizado em formulários, essa propiedade cria um modelo e liga o formulário a um objeto especifico. Isto significa que qualquer alteração no controle de atualizar os dados em seu modelo, e quando você mudar o modelo atualiza o controller.

### - declarações

A declaração {{ }} são uma forma declarativa de especificar locais do HTML que terá vinculação de dados. Angularjs irá atualizar automaticamente este campo sempre que o objeto for alterado.

## - Hello world simples

Apenas entender como funciona os itens acima citados, teremos um exemplo “hello world”, nele é possivel enteder a facilidade de manipular dados e DOM na tela, utilizando o conceito de “model”. Veja o código comentado:



O resultado será algo parecido com essa tela



- Data-Binding  
Data-Binding ou Ligação de Dados é uma forma automática de atualizar a exibição sempre que as mudanças do modelo , bem como a atualização do model sempre que a for captado mudanças . Isto é bom porque elimina a manipulação DOM da lista de coisas que você tem que se preocupar.

### - ng-controllers

Controllers são o comportamento por trás dos elementos DOM. Angularjs permite-lhe expressar o comportamento de uma forma legível limpa sem o clichê habitual de atualizar o DOM , registrar retornos de chamada ou assistindo a mudanças do modelo .

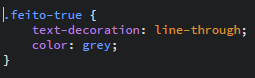
## - Lista de tarefas

O exemplo abaixo, ja é algo um pouco mais complexo que o anterior. Nesse poderemos ver um processo simples de adição e exclusão de elementos na tela, aqui ja utilizamos o “controller” e “data-binding”. Veja os códigos abaixo, ja comentados

HTML



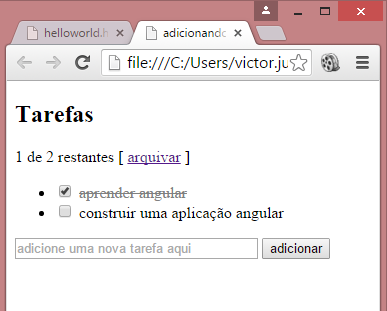
### CSS



Javascript



O resultado será algo parecido com essa tela



# Como utilizar na plataforma

Na plataforma da SPCM teremos diversas funcionalidades a utilizar, porém o intuito desse trecho do documento é apenas de apresentar os passos básicos para compreender o desenvolvimento da camada de apresentação e assim poder realizar as suas manutenções e afins. Para isso será apresentado um “Hello World” que solicitada informação ao banco e a presenta na tela dentro de uma “div“ de mensagem. Esse “Hello World” tem dois momentos, o primeiro é somente a solicitação da mensagem, o segundo ele passa por uma validação de login e verifica a permissão antes de realizar a chamada da mensagem de “hello world”

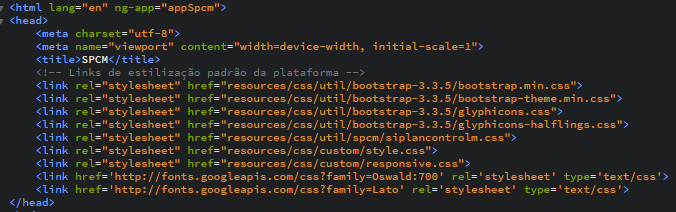
## - HELLO World

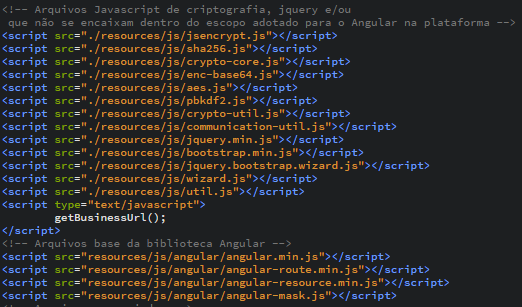
Você precisará estar pré-configurado com as seguintes coisas

- Camada de Business e de Core (Não será abordado)

### Index.html

Começando pela página index, que será a base praticamente de toda a aplicação, será realizada toda importação das bibliotecas de estilos e javascript, você vai notar que são blocos bastante extensos, mas até a versão em que esse documento está sendo redigido, elas que garantem toda a criptografia necessária e acompanham o guia de estilo definido pela empresa e não serão abordadas neste arquivo.

Estilização 

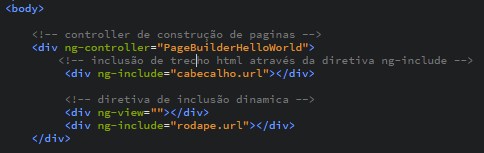
Arquivos Javascript padrões 

As inclusões dos arquivos do Angular já foram realizadas, como você pode perceber, ao final do arquivo. Alguns não foram utilizados anteriormente, mas serão apresentados a seguir:

* **Angular-Route:** Este arquivo permite a utilização do módulo ngRoute que fornece serviços e diretivas de roteamento e deeplinking para aplicativos angulares.
* **Angular-Resource:** Este arquivo permite a utilização do módulo ngResource que fornece suporte à interação com serviços RESTful através do serviço $resource
* **Angular-Mask:** Este arquivo permite a utilização do módulo ngMask que fornece suporte à utilização rápida de mascaras como de Data, telefone, cpf etc, não é relevante para este exemplo mas é interessante já conhecer sua existência pois será bastante utilizado na plataforma

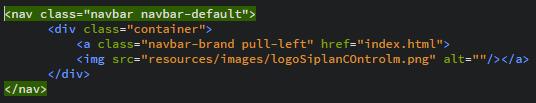
Dentro da tag *<Body>* inclua o código abaixo, nele está sendo associado o nosso controller *PageBuilderHelloWorld,*  dentro dele serão declaradas *divs*  com as propriedades *ng-include* e *ng-view*, o objetivo de ambas é trazer outros documentos html, porém se diferem da seguinte forma:

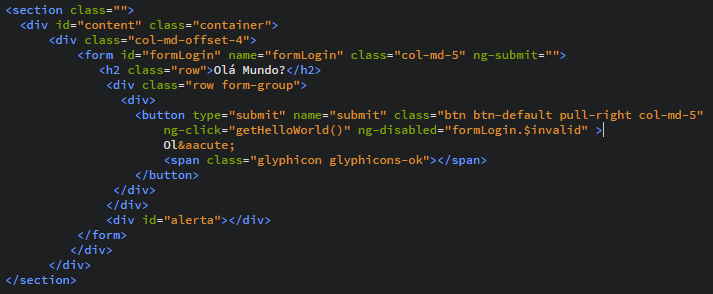
* Ng-include: traz o documento fixamente, você define no controller e ela sempre trará um valor fixo, ou seja, sempre o mesmo HTML
* Ng-VIew: traz o documento dinamicamente, no modulo você irá definir todas as possíveis páginas que ela irá acessar e conforme sua URL se altera, ela altera o valor HTML dentro, trazendo assim a possibilidade de navegação sem precisar recarregar outros elementos da página, apenas o que está dentro dela

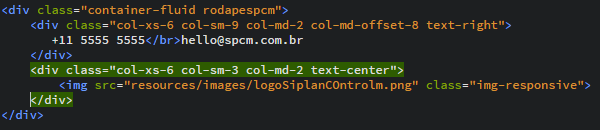
Veja o código abaixo que deverá ser incluso, exemplificando o que foi acima citado:

### Templates

Crie uma pasta dentro do projeto chamada templates, onde serão guardadas os outros documentos html que contém “fatias” da nossa página de Hello World. Crie 3 documentos: “header.html”,”helloworld.html”, “footer.html”.

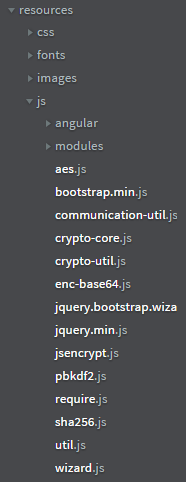
- header.html: 

- helloworld.html: 

- footer.html: 

### Javascript

Crie uma pasta *resources,* e dentro dela uma pasta chamada *js.* Dentro da pasta *js* você colocará todos aqueles arquivos já declarados no *index.html* que são padrão da plataforma e criará uma pasta angular para pôr os arquivos angular base. Após terminado esses passos, cê criará uma pasta chamada *modules* onde criaremos alguns arquivos. Você terá que estar com um diretório parecido com esse



Criando nossos Arquivos Javascript

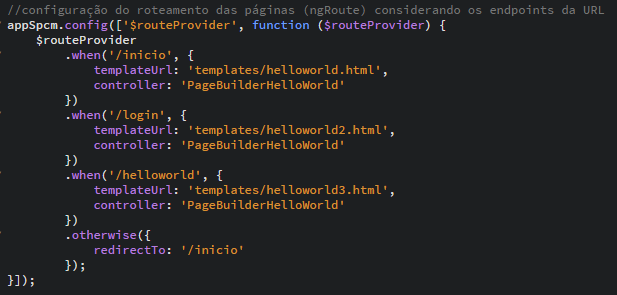
### - app-page-builder.js

O primeiro arquivo que criaremos será o *app-page-builder.js* , nele iremos declarar o modulo principal da aplicação, que foi extendido no começo deste exemplo, dentro dele faremos a inclusão de algumas dependências elas serão:

* ngMask:
* ngResource
* ngRoute
* crud (nós criaremos)
* userLog (nós criaremos)

escreva o código abaixo:

logo em seguida faremos a configuração do *RouteProvider,* que é o responsável pelo roteamento das páginas, ele vem do módulo ngRoute, graças a ele conseguimos trazer o dinamismo no carregamento dos arquivos html contextualizados com nossa url. Ainda no arquivo *app-page-builder, e*screva o seguinte código:



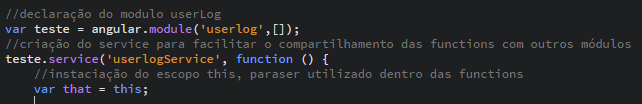
Note que declaramos mais 2 arquivos que ainda não criamos, não tem problema, eles serão criados posteriormente. Vale a pena também notar que caso coloquemos um endpoint que não se encaixe em um dos que foram declarados ele nos redirecionará com a função otherwise para o endpoint inicio, ou seja, ele é nosso default

Continuando, agora traremos inteligência e comportamento para o nosso módulo, iremos declarar o controller, nele declararemos os templates que serão utilizados na index.html e faremos a associação das funções que estão em outros services para podermos utilizarmos no app-page-builder

### - UserLog

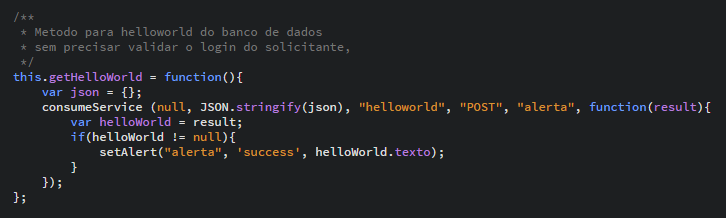
Agora faremos o arquivo base para interação com a camada de negócio que valida as permissões do usuário

Faremos a declaração do modulo chamando-o de userlog, instanciando dentro da variável teste. Criaremos um Service para poder facilitar o compartilhamento das functions com outros módulos como fizemos no documento app-page-builder.

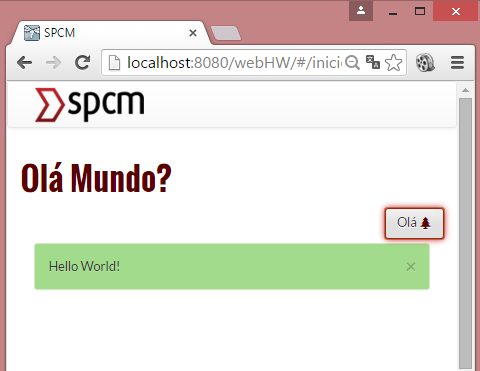


Note que estamos estando instanciando o contexto this dentro de uma variável, isso é feito para facilitar a recuperação de métodos ou variáveis que estejam em outro contexto, por exemplo fora de uma função

Agora iremos criar o helloWorld, esse é mais simples, então não será necessário validar o login do usuário, ele chamará uma função “consumeService” que está dentro do arquivo útil.js e enviará a solicitação ao business e lá será tratada. Como resposta de sucesso ele chama uma função chamada setAlert que irá criar uma div com a mensagem recuperada da camada de negócio. Em caso de insucesso a propia função consumeService retorná uma mensagem de erro, como isso já está sendo tratado lá então não precisa se preocupar. Copie o código abaixo:



Feito isso, salve todos os documentos. Faça o teste, o resultado da tela será algo parecido com isso:

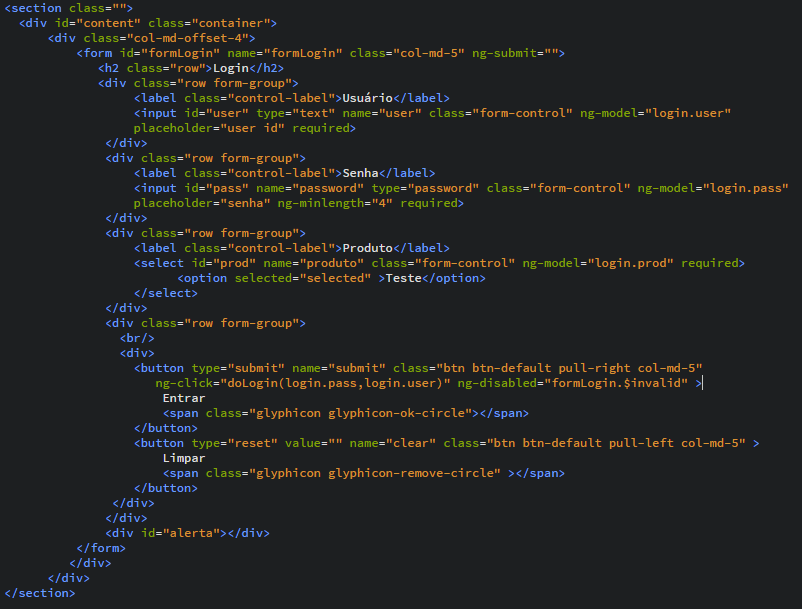


## - Hello World com Verificação de Log

Bom, no exemplo anterior fizemos o hello world dentro da plataforma para exemplificar o caminho de funções que não necessitem de autenticação, agora veremos como será quando precisar de autenticação. Já temos boa parte desse exemplo pronto, agora precisamos adicionar o restante

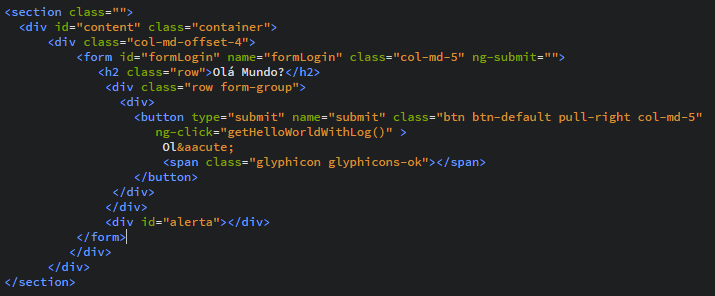
### - helloworld2.html

Esse documento foi declarado no modulo app-pagebuilder porém o tínhamos ainda, não teve problema afinal a pagina era redirecionada para /inicio. Crie um documento helloworld2.html e adicione na pasta templates, copie o seguinte código:



### -helloworld3.html

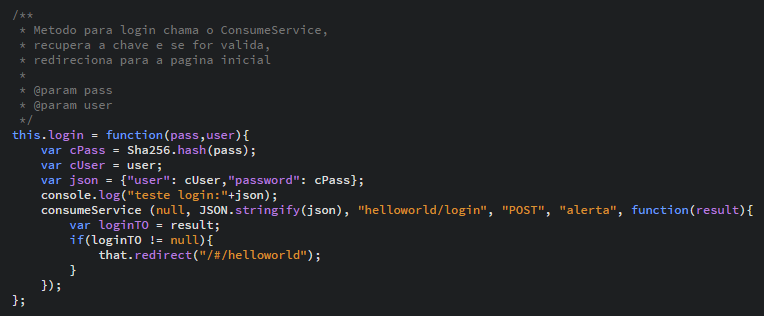
Esse documento foi declarado no modulo app-pagebuilder porém o tínhamos ainda, não teve problema afinal a pagina era redirecionada para /inicio. Crie um documento helloworld3.html e adicione na pasta templates, copie o seguinte código:



### - userlog.js

Neste documento iremos adicionar algumas funções para poder validar o log do usuário

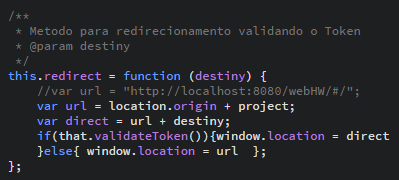
#### Login



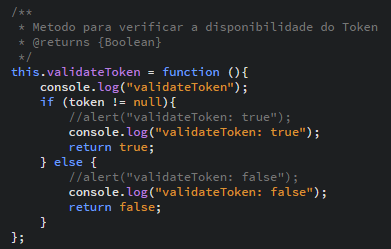
#### Logout



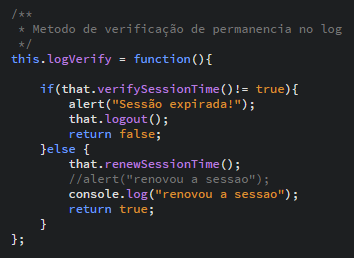
#### Redirect



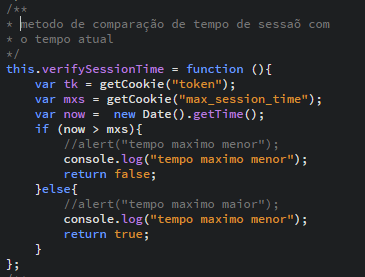
#### validateToken



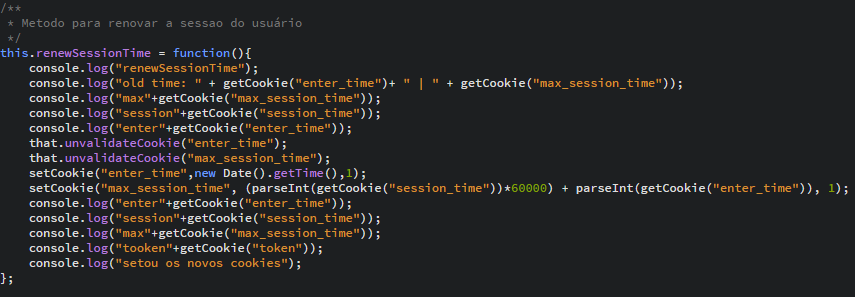
#### logverify



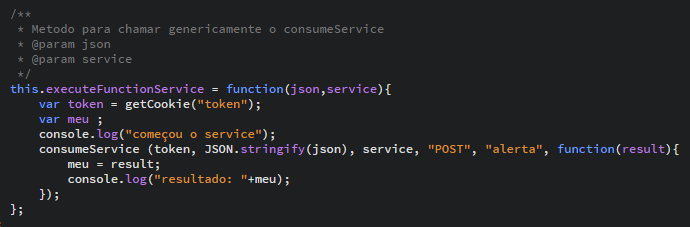
#### verifySessionTime



#### renewSessionTime



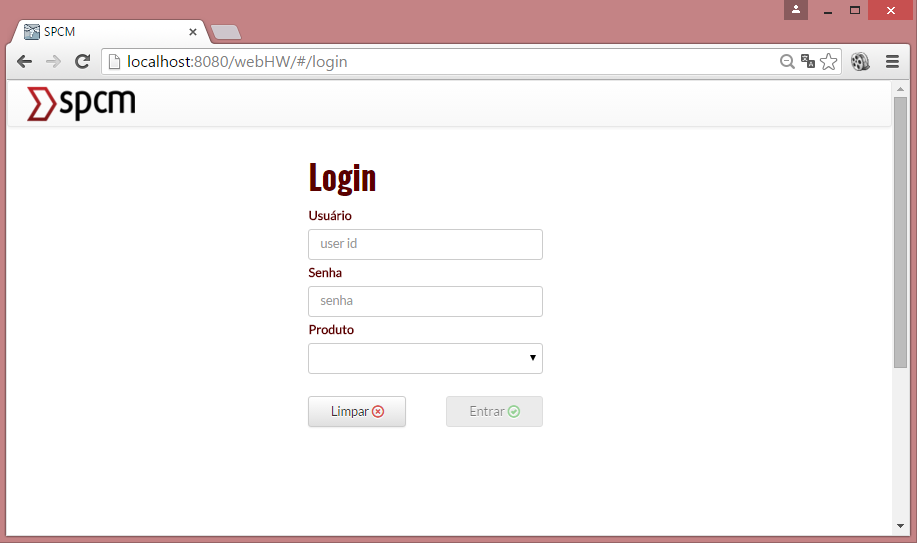
#### executeFunctionService



Como você pode notar, existem muitos comandos de console.log no meio das funções, eles não são usuais na plataforma, foram mantidos aqui para que no próximo passo você possa comparar os resultados e entender melhor o caminho da informação. Volte ao navegador e coloque a seguinte URL

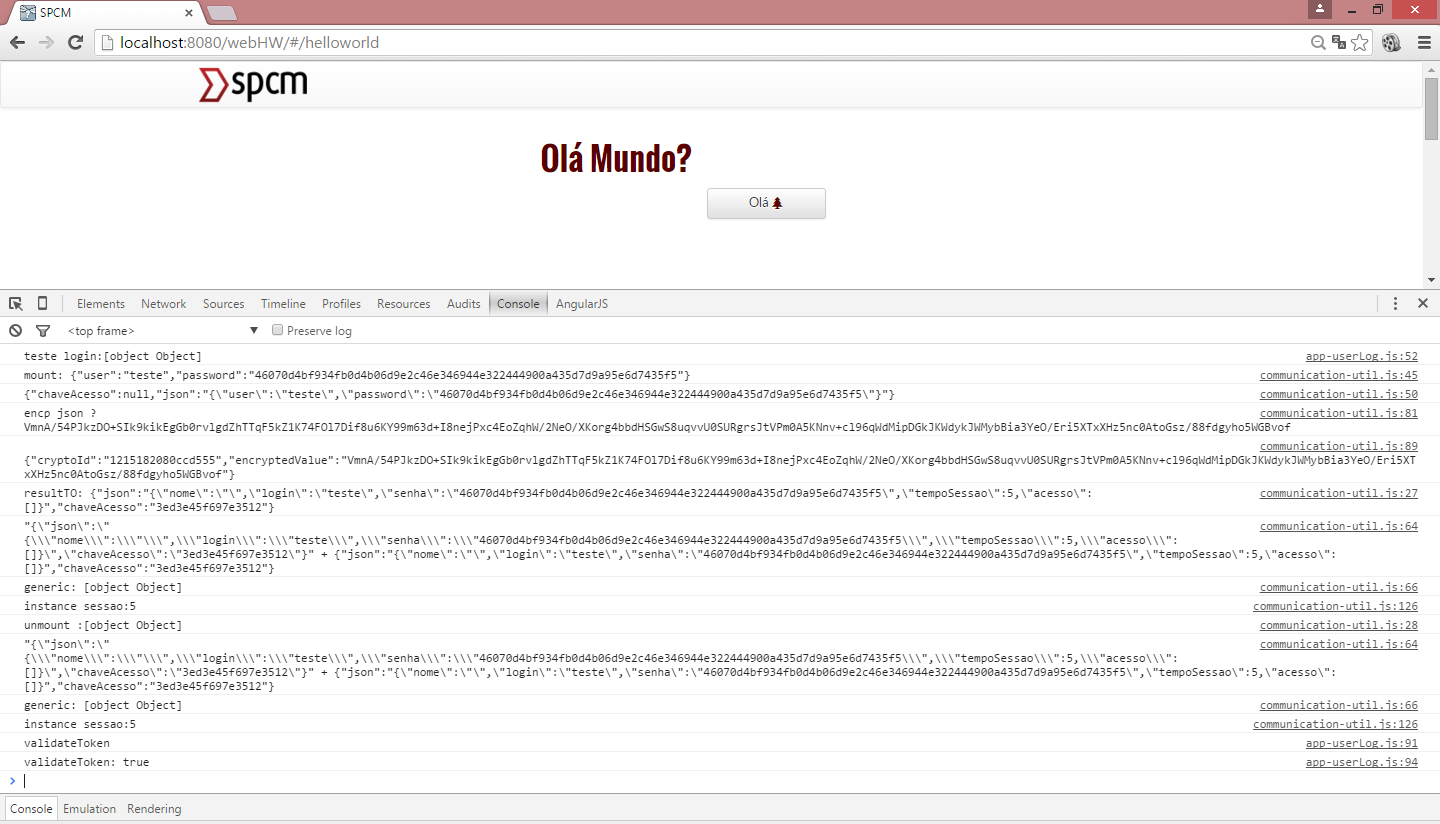


Você irá ser apresentado a uma janela igual a esta:

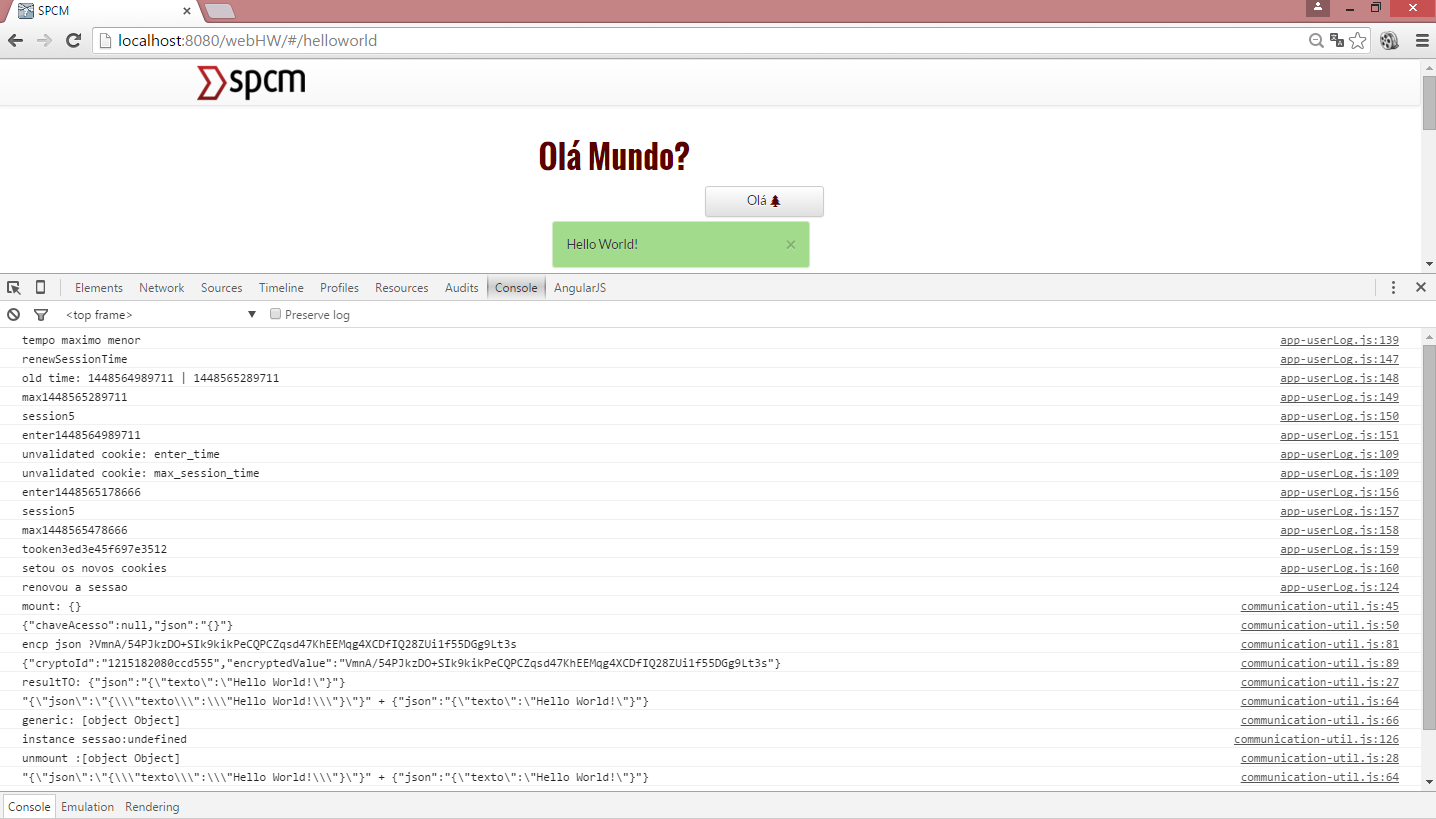


Por padrão o usuário é “teste” e a senha “teste”, ambos minúsculos e idênticos, selecione o produto e clique em entrar, o formulário será validado pelo controller e acionará o botão, é interessante você acompanhar o console do navegador, lá você irá acompanhar os logs, e facilitará o entendimento do caminho.

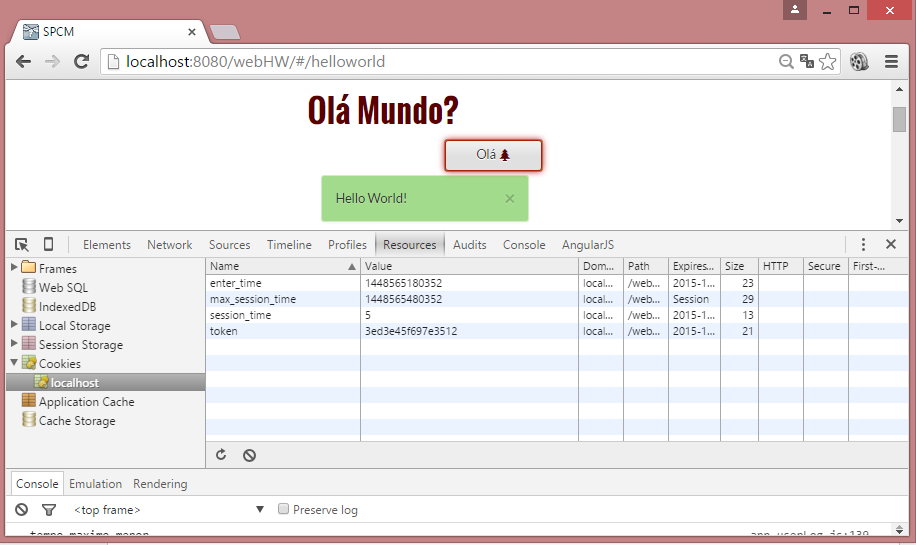
Com o login realizado, você será redirecionado para outra pagina, o endpoint mudará, mas a tela será extremamente parecida porém chamará funções diferentes da primeira vez, as funções serão as declaradas anteriormente, veja abaixo a resposta do servidor no console



Agora ao clicar no botão olá você obterá a resposta, mas o console será muito mais preenchido:



Veja que ele verifica o tempo de sessão antes de solicitar o serviço depois ele revalida a sessão e finalmente dispara a solicitação, todos esses dados são guardados em cookies, e você poderá consulta-los no Resources do navegador



Esse é o caminho mínimo padrão da plataforma, exemplificando o tratamento que o angular dá aos dados e elementos

# Teste Unitários

Para Realizar todos os testes descritos a seguir, é necessário realizar a instalação do *NodeJS*, que fará o papel de interpretar o código simulando um *Server*, segue abaixo a descrição do *NodeJs* e sua instalação. Após instalar o *NodeJs*, será necessário instalar o *Python*, e um compilador C++, no caso foi baixado o *Visual Studio*, pois estava disponível na ocasião:

## Sobre NodeJS

***Node.js*** é um interpretador de código *JavaScript* que funciona do lado do servidor. Seu objetivo é ajudar programadores na criação de aplicações de alta escalabilidade (como um servidor *web*) com códigos capazes de manipular dezenas de milhares de conexões simultâneas, numa única máquina física. O *Node.js* é baseado no interpretador V8. Foi criado por [Ryan Dahl](https://pt.wikipedia.org/w/index.php?title=Ryan_Dahl&action=edit&redlink=1) em 2009, e seu desenvolvimento é mantido pela empresa *Joyent*, onde Dahl trabalha.

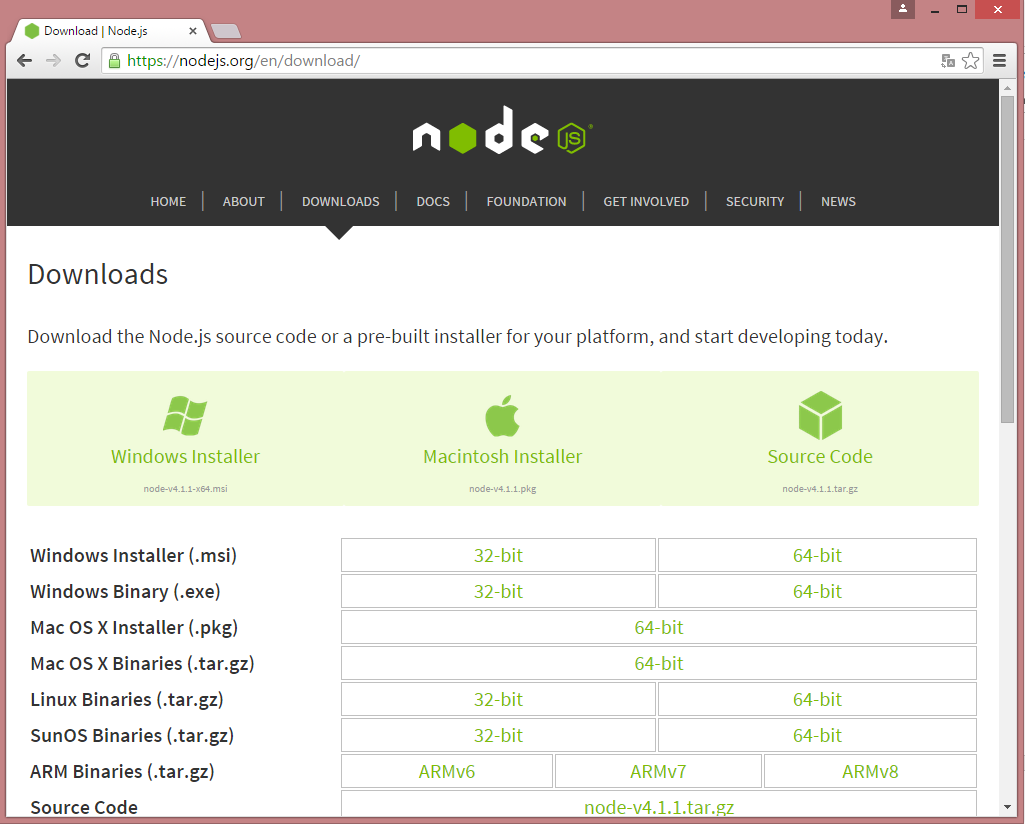
* Um **interpretador *JavaScript*** — também chamado de **motor *JavaScript*** — é um *software* especializado que interpreta e executa *JavaScript* ou *ECMAScript*. Embora existam vários usos para o interpretador *JavaScript*, ele é mais comumente usado em navegadores de *internet*. O interpretador V8 é a versão de interprete do *Google* *Chrome.*

## Instalação

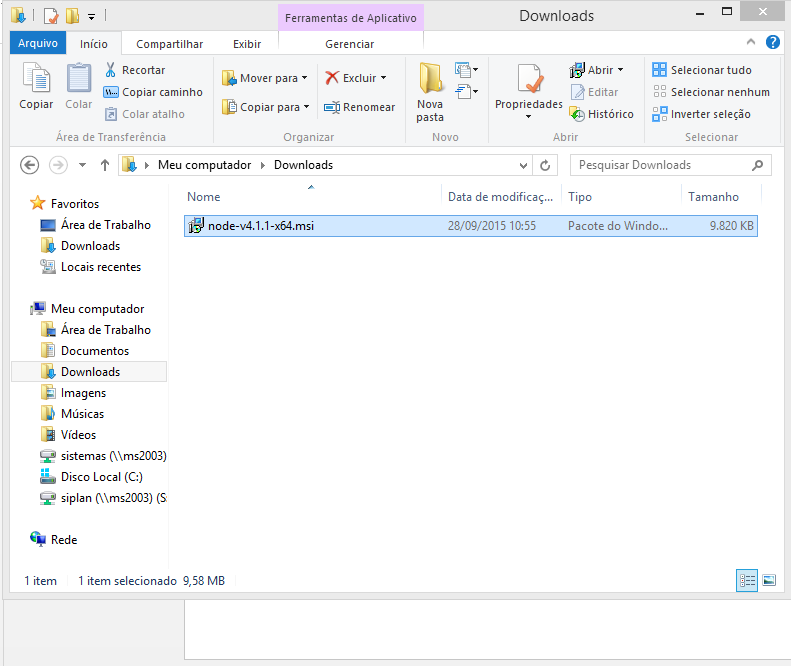
### Node.JS

Abaixo será descrito o passo-a-passo para a instalação do *NodeJS* em *Windows*, da maneira mais simples, sem *plug-ins* ou outros complementos que não façam parte do escopo funcional da ferramenta

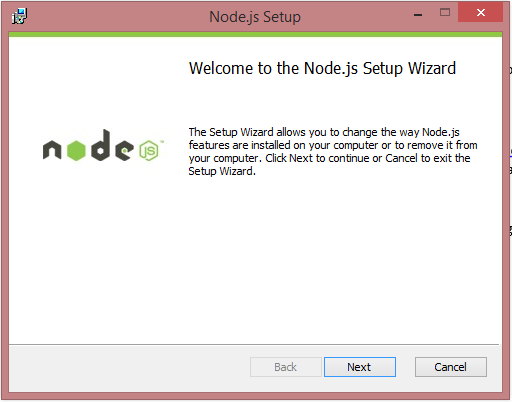
* Vá até o site Oficial do *NodeJs*, na aba de *Downloads* (atualmente encontra-se no link <https://nodejs.org/en/download/> ), escolha a versão que se adapte melhor para sua versão de *Windows.*



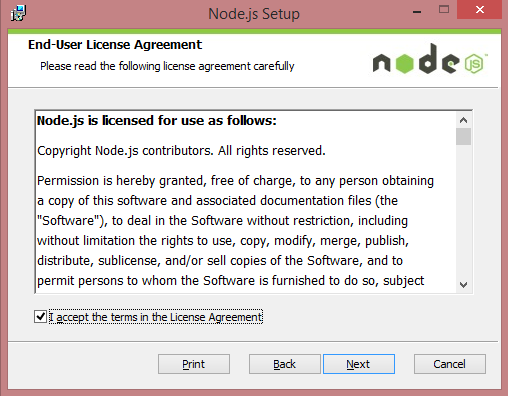
* Após a finalização do *Download*, abra o arquivo “.msi” que foi gerado.



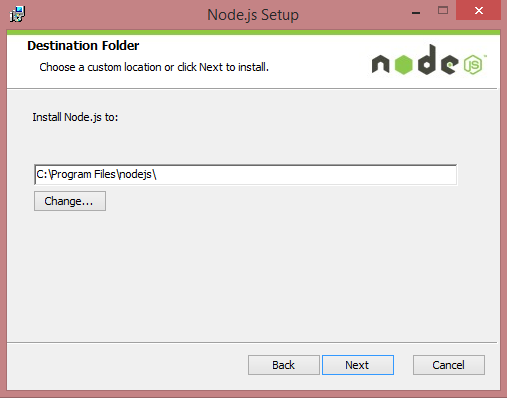
* Será aberto uma janela semelhante a esta que se encontra na imagem abaixo, clique em “Next”.



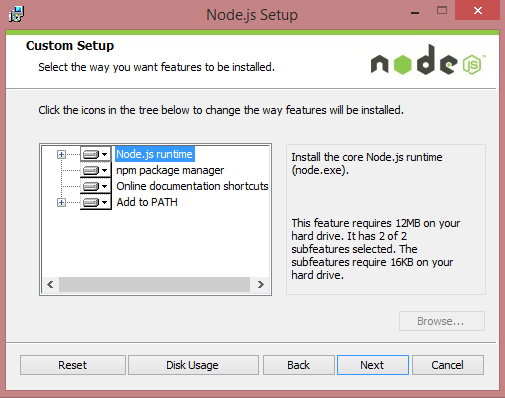
* Será mostrado o contrato de uso, leia, aceite os termos de uso e clique em “Next”.



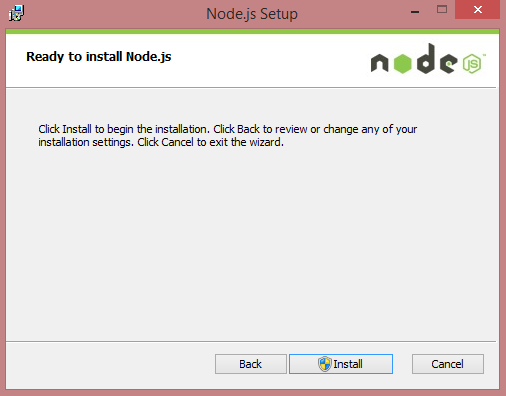
* Logo em seguida, será necessário direcionar o local da instalação, mas por *default* será guardada no caminho “C:\Program Files\nodejs\”.



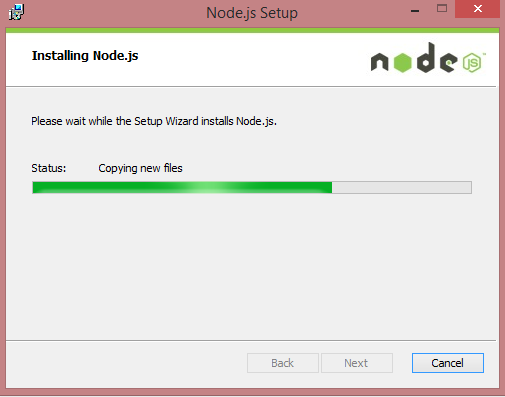
* Após especificar o caminho da instalação, será necessário selecionar o que vai ser instalado, deixe tudo selecionado e clique em “Next”.



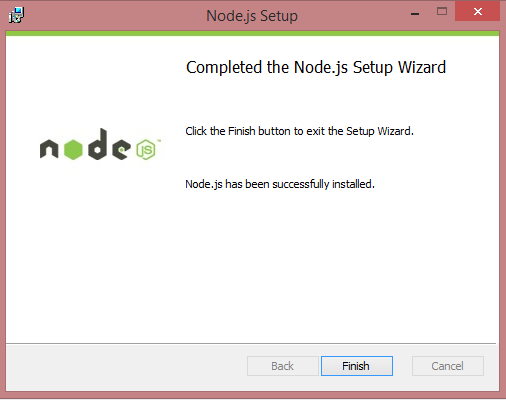
* Haverá uma confirmação de nível administrativo para realizar a instalação clique em “Install”, a tela irá escurecer novamente para perguntar se você realmente deseja autorizar a instalação, confirme novamente.



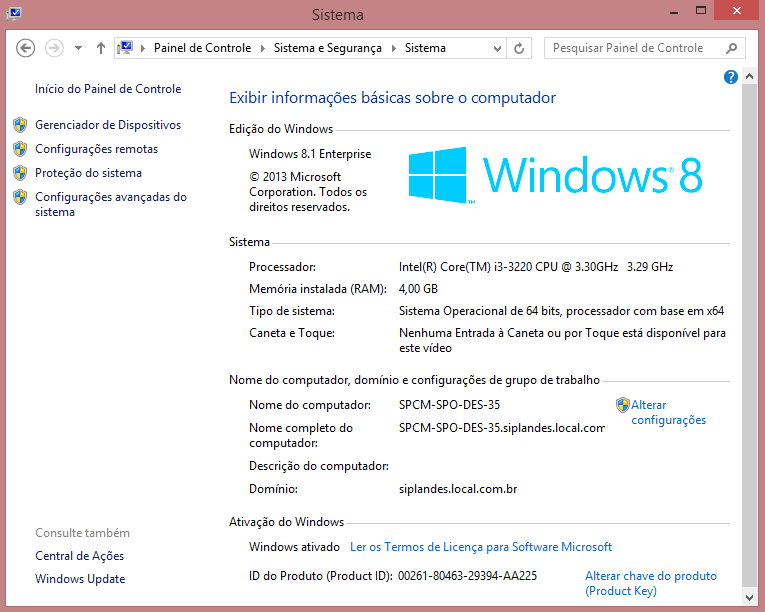
* Começará a instalação.



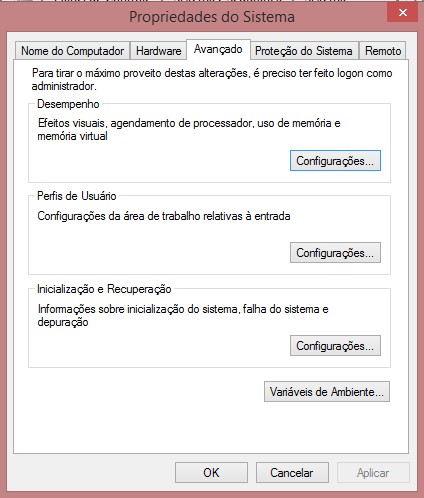
* Ao final, o instalador confirmará a conclusão, basta agora clicar em “Finish”.



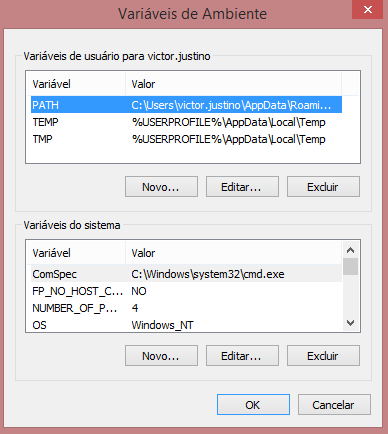
* Após instalado, é necessário registrá-lo no Path do Windows, para isso vá nas Propriedades do Sistema e clique em Configurações avançadas do Sistema.



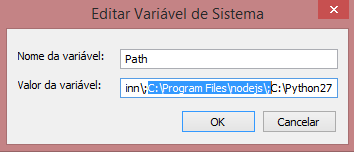
* Abrirá uma janela chamada Propriedades do Sistema, clique em Variáveis de ambiente.



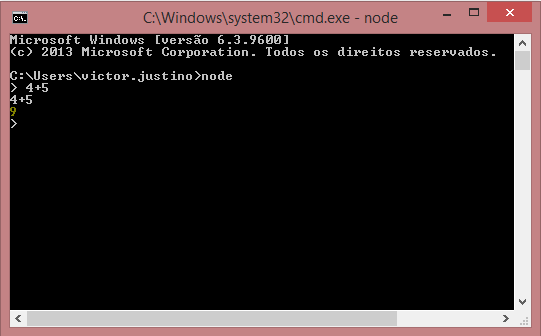
* Será aberta uma nova janela, vá na caixa Variáveis do sistema, encontre o item Path e clique em editar.



* Na caixa de texto Valor da variável, coloque o seguinte texto: “C:\Program Files\nodejs\;” e salve tudo.



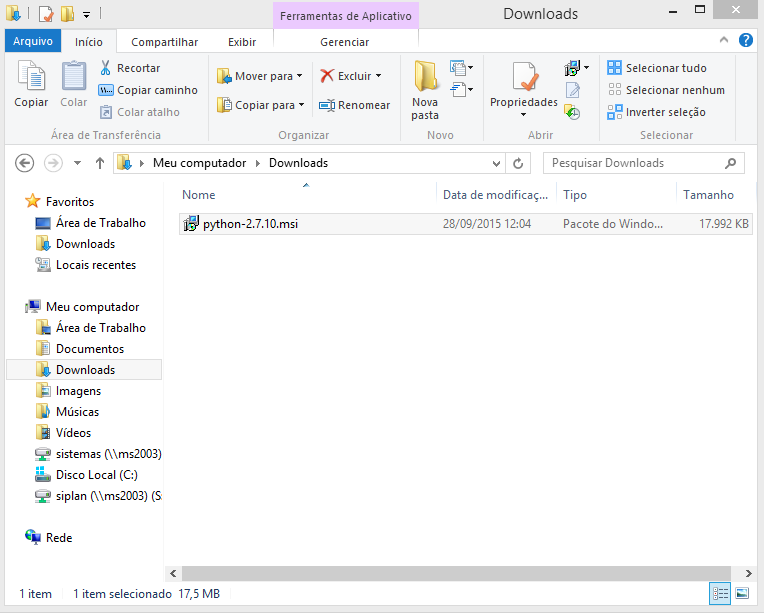
* Para testar se a instalação está correta abra o *Prompt* de comando escreva o comando “node”, aperte “*Enter*”. Isso deve te mandar para a linha abaixo, realize uma conta simples como “4+5” isso deve te retornar o resultado em verde, como na imagem abaixo’



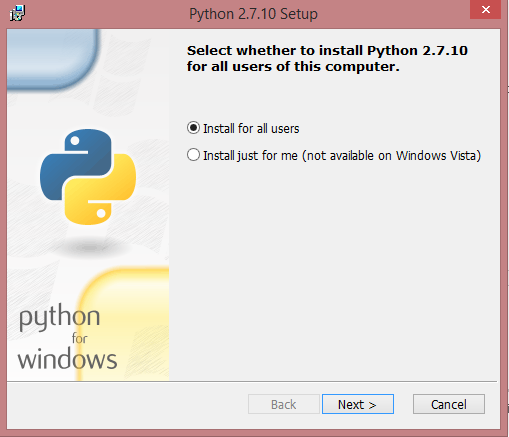
### Python

Abaixo será descrito o passo-a-passo para a instalação do Python em *Windows*, da maneira mais simples, sem *plug-ins* ou outros complementos que não façam parte do escopo funcional da ferramenta

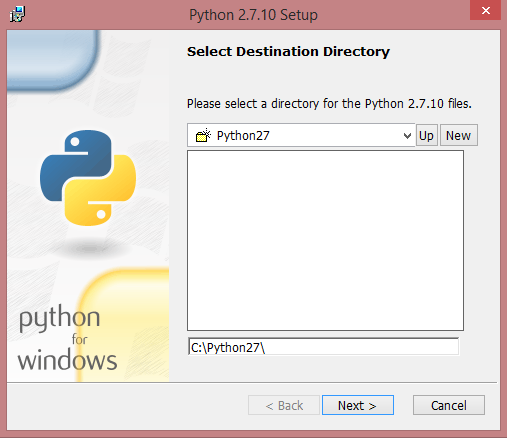
* Vá até o site Oficial do *Python*, na aba de *Downloads* (atualmente encontra-se no link https://www.python.org/downloads/ ), na ocasião foi escolhida a versão 2.7.10 por ser a mais estável.
* Após a finalização do *Download*, abra o arquivo “.msi” que foi gerado.



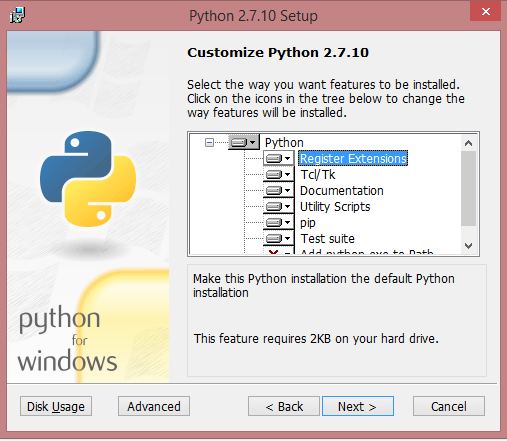
* Será aberto uma janela semelhante a esta que se encontra na imagem abaixo, perguntando se deseja instalar para todos usuários, clique em “Next”.



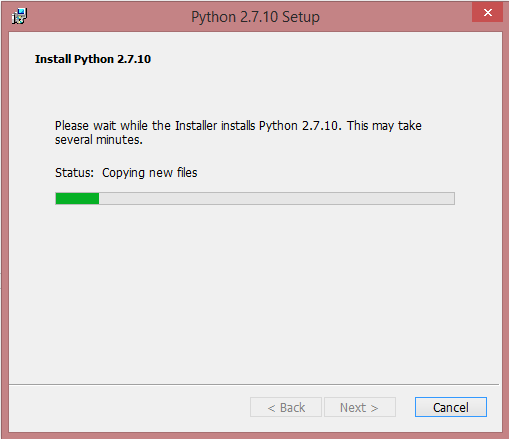
* Logo em seguida, será necessário direcionar o local da instalação, mas por *default* será guardada no caminho “C:\Python27\”.



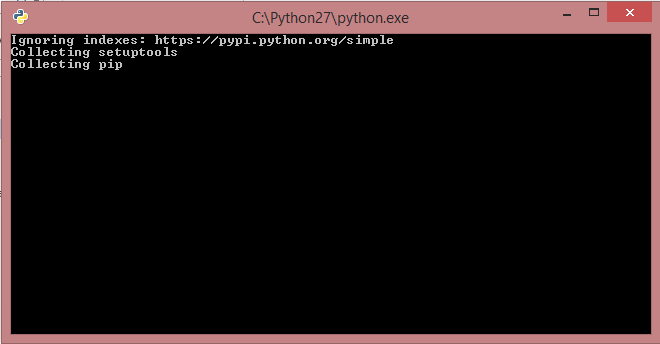
* Após especificar o caminho da instalação, será necessário selecionar o que vai ser instalado, deixe tudo selecionado, exceto “*Add python .exe to Path*”, e clique em “Next”.



* Começará a instalação.



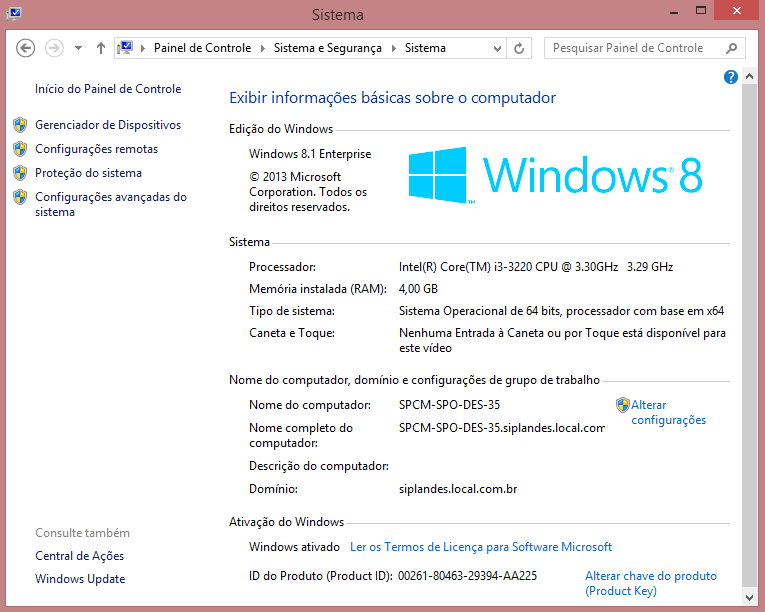
* Ele abrirá o *Prompt* de Comando para realizar algumas tarefas de instalação



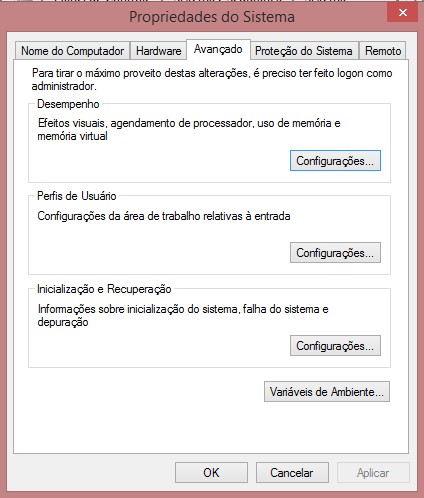
* Ao final, o instalador confirmará a conclusão, basta agora clicar em “*Finish*”.



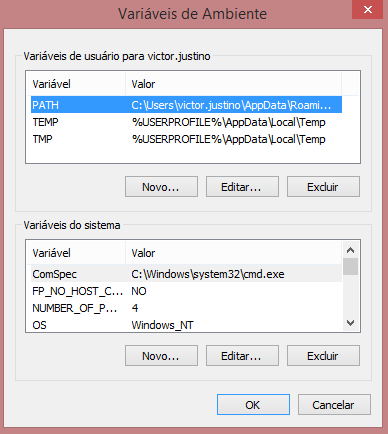
* Após instalado, é necessário registrá-lo no Path do Windows, para isso vá nas Propriedades do Sistema e clique em Configurações avançadas do Sistema.



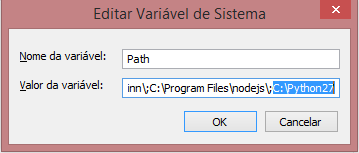
* Abrirá uma janela chamada Propriedades do Sistema, clique em Variáveis de ambiente.



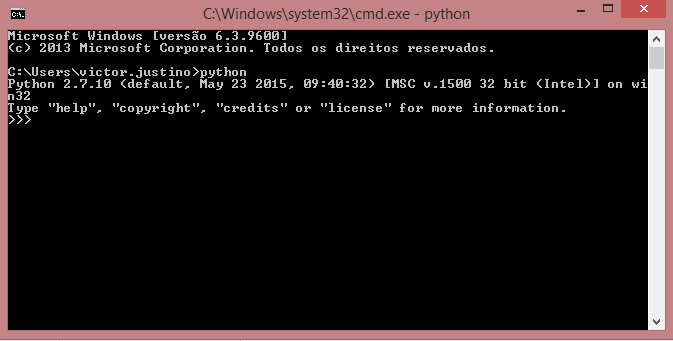
* Será aberta uma nova janela, vá na caixa Variáveis do sistema, encontre o item Path e clique em editar.



* Na caixa de texto Valor da variável, coloque o seguinte texto: “C:\Python27;”



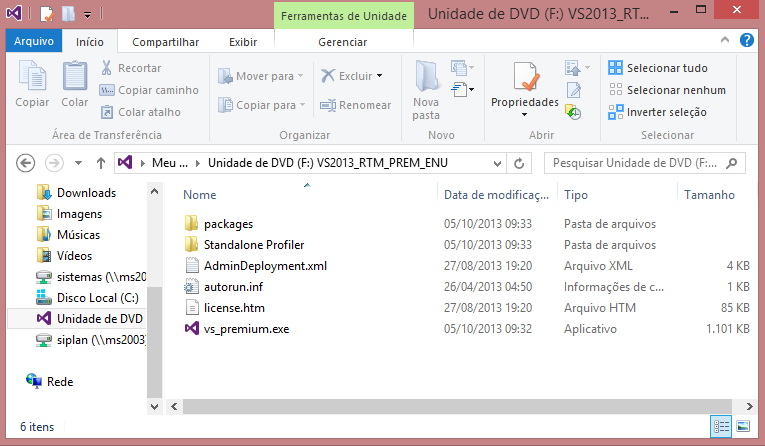
* Salve tudo, abra o *Prompt* de Comando, digite “*python*” e aperte “*enter*”, se tudo der certo ele retornará uma mensagem com a versão do Python utilizado



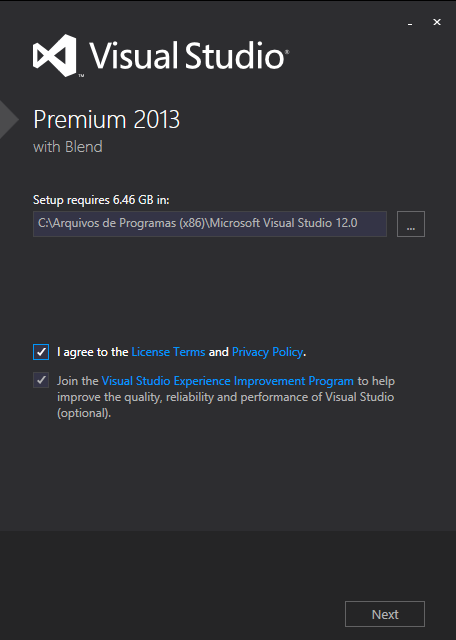
### Visual Studio

A instalação do Visual Studio se torna necessária, o *NodeJs* precisa de um compilador C++, foram realizadas algumas pesquisas para ver a necessidade real de instalar o Visual Studio, e ela realmente existe, mas a versão pode ser a partir da 2005 e ainda pode ser a versão mais enxuta, mais simples e menor. O processo de instalação do Visual Studio é bastante simples:

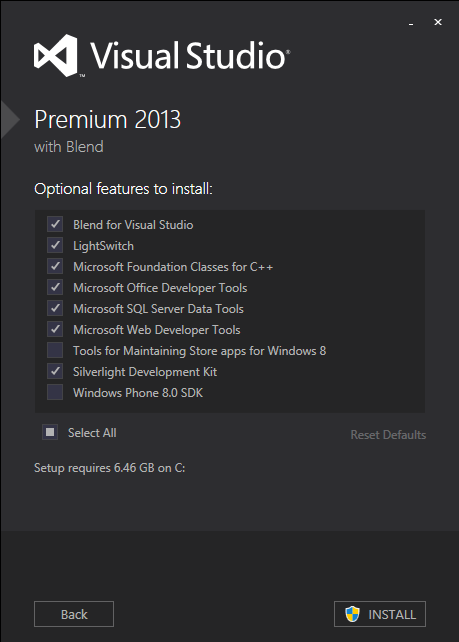
* Abra o CD ou a pasta com os arquivos e selecione o “*.exe*”



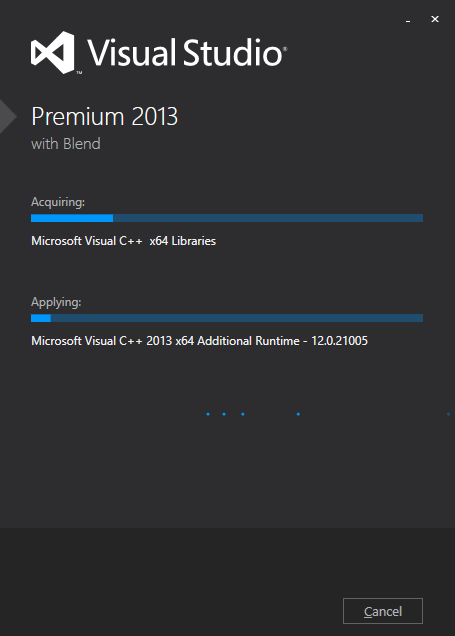
* Será aberta uma janela mostrando o local de instalação do programa, caso seja necessário pode alterar, mas por default será mantido o caminho “*C:\Arquivos de Programas (x86)\Microsoft Visual Studio 12.0*”, aceite os termos de uso e clique em “Next”



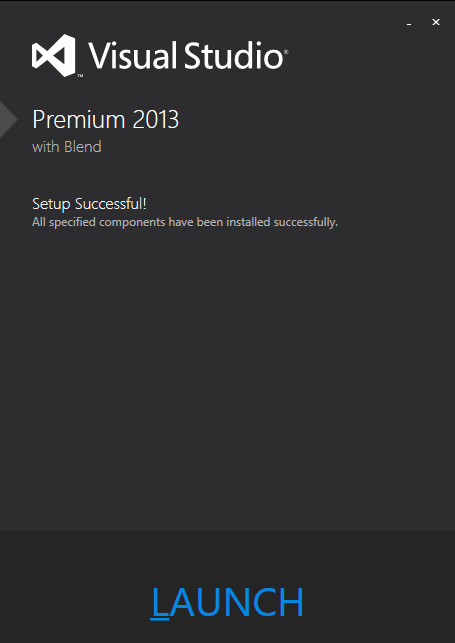
* Após especificar o caminho da instalação, será necessário selecionar o que vai ser instalado, pode deixar apenas o que já está, não há necessidade de escolher mais coisas. Clique no botão “*Install*” que fará uma verificação administrativa se você realmente deseja instalar o produto



* Ele começará a instalação, pode demorar um pouco



* Quando terminar será mostrado uma mensagem de sucesso, não precisa iniciar o programa, pode fechá-lo



# Testes unitários - TDD

Para realizar testes unitários em Angular é necessário realizar uma preparação de ambiente

## Instalando o *Karma-jasmine*

• Primeiramente abra o *Prompt* de Comando do *Windows*

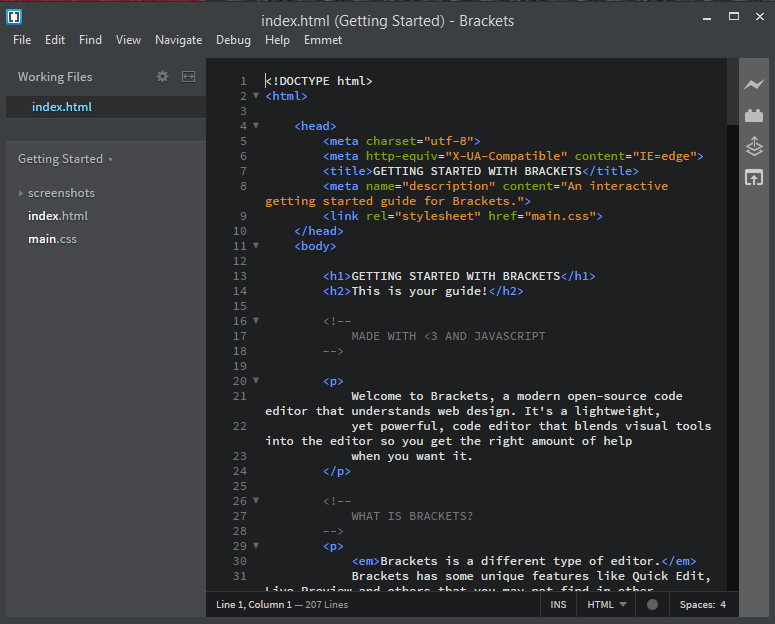
• Realize a instalação do *Karma Runner*, digite “*npm install karma --save-dev*”

• após a instalação do *Karma*, realize a instalação dos plug-ins necessários, sendo eles *Jasmine* e *Chrome launcher*, digite “*npm install karma-jasmine karma-chrome-launcher --save-dev*”

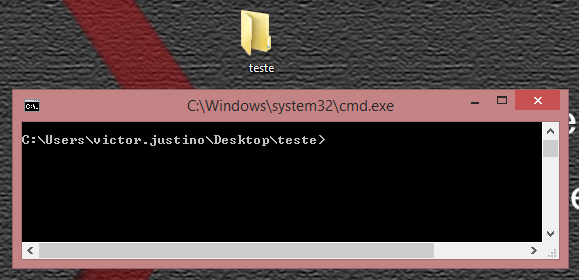
• isso será necessário ser realizado tanto dentro do diretório do projeto, como globalmente, ou seja dentro das configurações do Usuário, que abrem por default no *Prompt*

## Iniciando o projeto

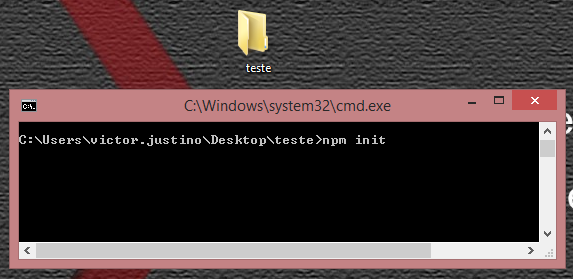
• Abra um editor de texto, no caso foi utilizado o *Brackets*, mas pode ser o Sublime e até mesmo o *Notepad*, por serem mais simples para realizar as tarefas



• Crie uma pasta para o projeto e abra o *Prompt* de Comando, ela não pode ter nome separado por espaço, para facilitar os passos seguintes

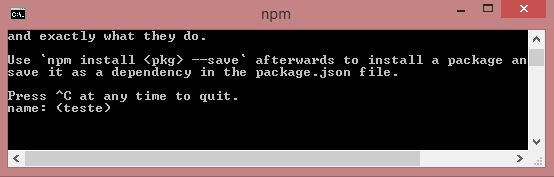


• No *Prompt*, direcione-se até a localização da pasta do projeto, e digite “*npm init*”

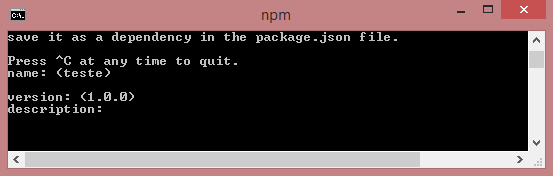


• será apresentada alguns campos para preencher

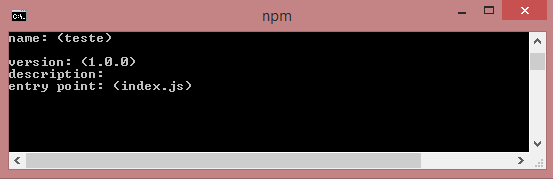
1. Nome do Projeto: ele virá já preenchido e virá entre parentes, aperte “*Enter*” para confirmar



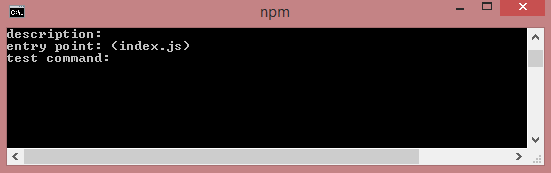
1. Versão: é opcional preencher, provavelmente virá como 1.0.0, aperte “*Enter*”
2. Descrição: é opcional preencher, aperte “*Enter*”



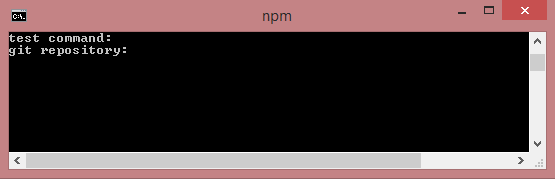
1. Ponto de entrada: é o nome do arquivo principal, é opcional mudar, ele virá como “index.js”, a princípio não tem problema nenhum manter, aperte “*Enter*”



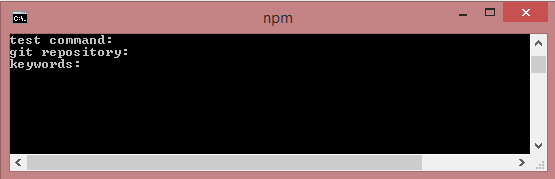
1. Comando de Teste: é opcional, aperte “*Enter*”



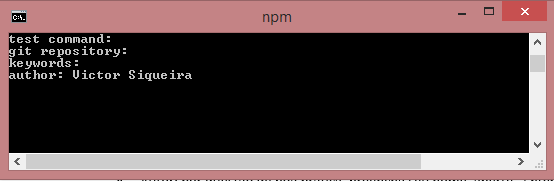
1. Repositório git: é opcional indicar o repositório, aperte “*Enter*”



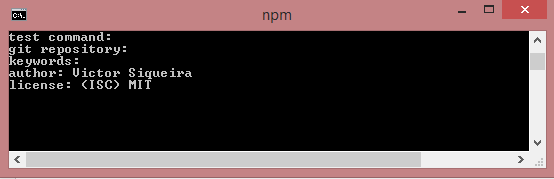
1. Palavras chave: é opcional também, novamente aperte “*Enter*”



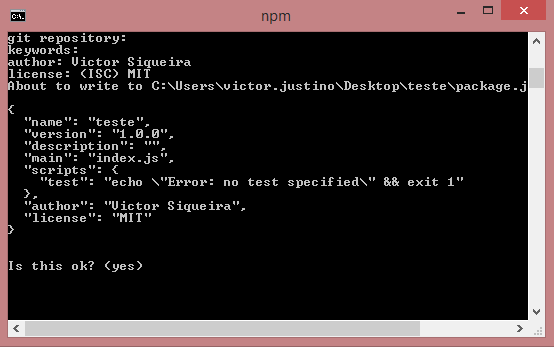
1. Autor: por questão de boa prática, preencha seu nome, aperte “*Enter*”



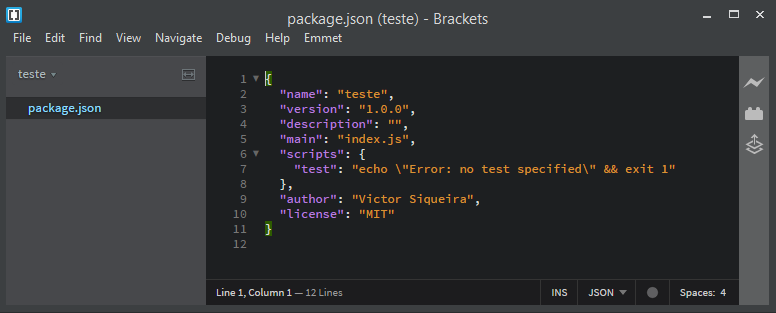
1. Licença: virá definida como ISC, escreva MIT e dê “*Enter*”



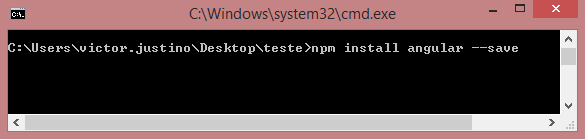
1. Será gerado um arquivo JSON, e o mesmo será apresentado na tela, será solicitado a confirmação, aperte “*Enter*”



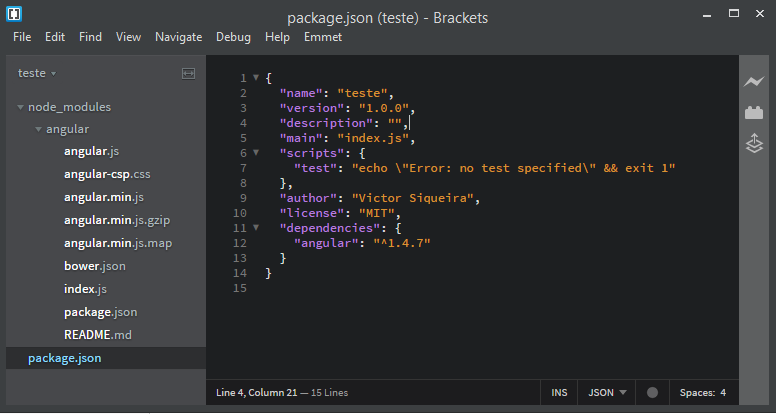
• Confira na pasta do projeto se o arquivo JSON realmente foi gerado, por default ele se chamará package.json



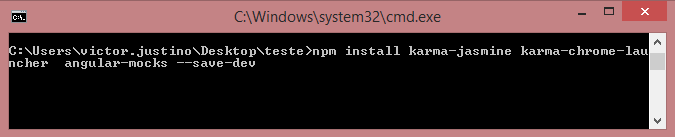
• Instale as dependências do projeto digite “npm install angular --save”



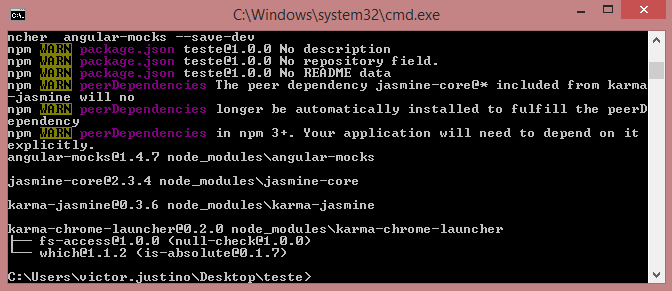
1. Confira na pasta do projeto se foi criada uma subpasta “node\_modules”, dentro desta sub pasta haverá outra chamada “angular”, dentro dela vários arquivos com final .js e a maioria com o nome começando por angular



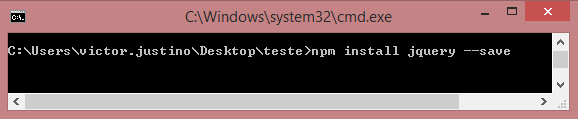
* Volte ao Prompt, ainda dentro da pasta do projeto, digite “npm install karma-jasmine karma-chrome-launcher angular-mocks --save-dev”



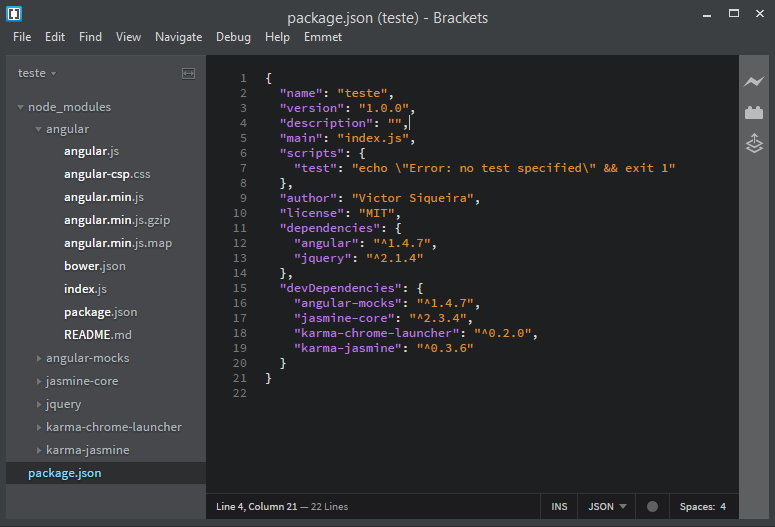
* 1. Provavelmente aparecerá algum erro relacionando ao karma-jasmine, mas não se preocupe, mais para frente isso será resolvido



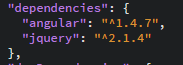
• Agora traga o jquery para o projeto, digite “npm install jquery --save”



• Para finalizar abra a pasta do node\_modules, verifique se todas as pastas foram baixadas, caso o *Jasmine* não tenha sido baixado não tem problema, na versão atual está vindo dentro da pasta do próprio *Karma*. Se as pastas estiverem disponíveis dentro da *node\_modules* vá até o arquivo *package.json* verifique o campo de dependências



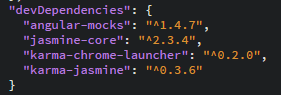
1. Provavelmente ele estará descrevendo sobre o angular, o *jquery* e suas respectivas versões, eles devem estar alocados no campo de dependências comum



1. Já nas “*devDependencies*” (dependências de desenvolvimentos), estará descrito *angular-mocks, karma, karma-chrome-laucher* e suas respectivas versões



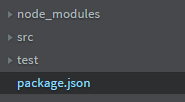
1. Se durante a instalação ocorreu um erro no momento de instalar o *Jasmine*, muito provavelmente ele não estará descrito na *devDependencies*
2. Para solucionar isso vá até um buscador e digite as palavras chave “*karma-jasmine* *last* *version*”, realize a busca e encontre o número da versão atual, no caso deste descritivo, a versão se encontrava na “ ~0.3.6 ”
3. Adicione dentro de *devDependecies* a linha “*karma-jasmine*”:”{número da versão}”, como mostra o exemplo abaixo



## Separando corretamente as Pastas

É necessário separar corretamente cada arquivo pois assim o *Karma* não se confunde em reconhecer os arquivos testes dos arquivos que estão vão para produção. Para isso siga os passos abaixo:

• Crie duas pastas, uma chama “*src*” e outra chamada “*test*”

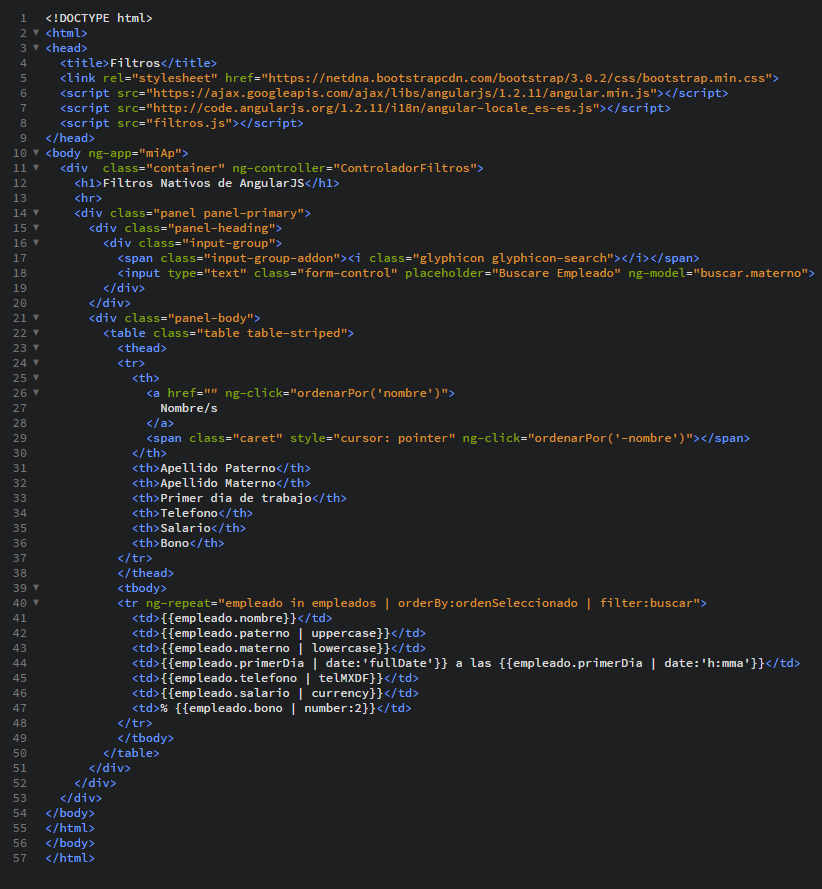


• na pasta “*src*”

1. Você pode montar toda a estrutura do projeto normalmente, não há problemas quanto a isso dentro desta pasta
2. Para exemplificação, crie um arquivo .js com o nome de “filtros” e copie o código abaixo

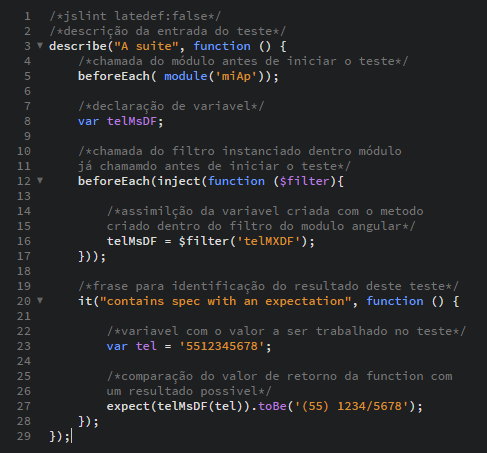


1. Crie um arquivo .html com o nome de “index” e copie o código abaixo



• na pasta “*test*”

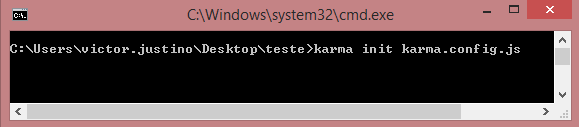
1. Crie um arquivo “.js” com o nome de “filtros.spec” e copie o código abaixo



1. A linha 3 é o começo do código lá você irá colocar uma *string* que identifique o teste em si e chamará uma *function*
2. Na linha 5 está “dizendo” que antes de cada teste, deve ser trazido o modulo “x”
3. Na linha 8, está sendo criada uma variável apenas
4. Na linha 12 está “dizendo” que antes do cada teste, deve ser inserido uma function dentro do processo e essa *function* recebe, neste caso, um “*filter*”, que é justamente o que foi declarado no nosso modulo angular, e será exatamente sua function aninhada que será testada
5. Na linha 16 está sendo assimilada a variável anteriormente declarada à *function* do *filter* trazido, declarando uma *string* equivalente ao nome do método
6. Na linha 20 está sendo criada o método de resposta do teste, que onde é realmente onde as coisas acontecem, é necessário colocar uma *string* para poder identificar a resposta no *Prompt*
7. Na linha 23 foi criada uma variável com um valor para ser testado pelo método assimilado anteriormente
8. Na linha 27 está sendo realizada a comparação de resposta entre o resultado que virá do método assimilado com o possível resultado predefinido pelo programador
9. Observação: note o comentário no começo do código, ele está ali desabilitando o *JSLint*, que é um validador de sintaxe *javascript*, é interessante colocá-lo apenas ao final para poder verificar se tudo está correto, quando ele não estiver escrito no topo da página, o primeiro erro que ele vai pegar é em relação ao “module”, ele dirá que o “module” não foi descrito antes de ser instanciado

## Iniciando o *Karma* para testes

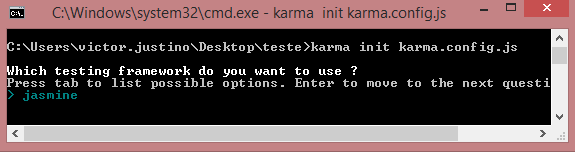
• Abra o *Prompt* de comando, digite “*karma* *init karma.config.js*”, você estará criando o arquivo base para o *Karma* realizar suas tarefas



• Serão abertas algumas perguntas

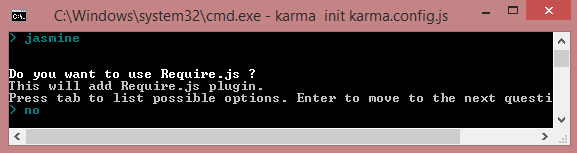
1. “*Which testing framework do you want to use?*”

Aqui está perguntando qual framework de teste você quer usar, por default virá “*jasmine*” como resposta, pode apertar “*Enter*”



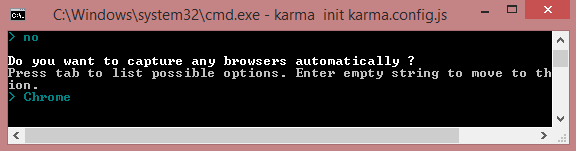
1. “*Do you want to use Require.js?*”

Aqui está perguntado se você quer usar o framework Require.JS, aqui não será necessário, digite “no” e aperte “Enter”



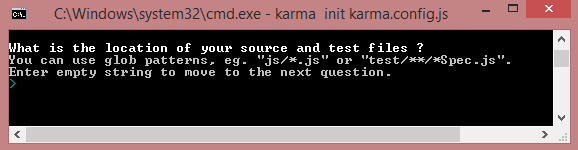
1. “*Do you want to capture a browser automatically*?”

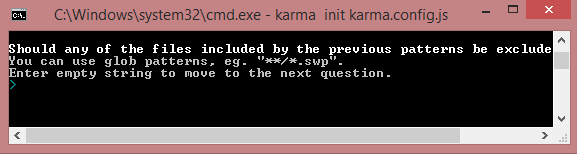
Aqui ele está perguntando se você deseja realizar teste com algum navegador, por default ele escolhe o *Chrome*, mas é importante destacar que você pode acrescentar outros, ao final da escolha dos navegadores, aperte “Enter” 2 vezes



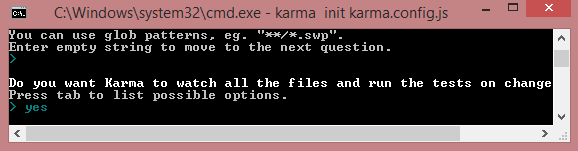
1. “*What is the location of your source code and test files*?”

Aqui ele está perguntando onde está seu código para realizar os testes, pode deixar em branco, pois será colocado posteriormente direto no arquivo, aperte “*Enter*”

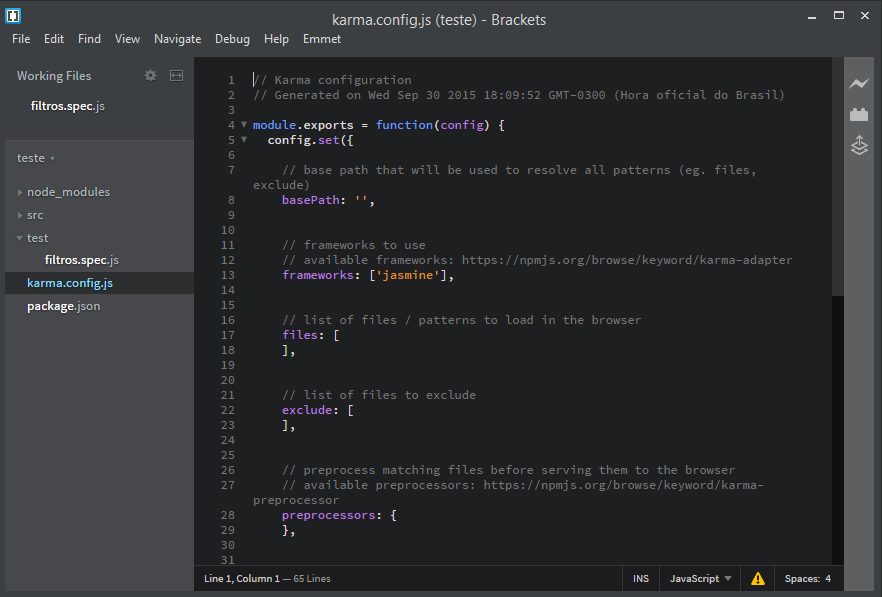




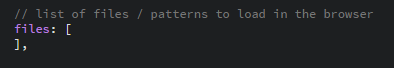
1. “*Do you want karma to watch all the files and run test on change*?”  
   Aqui está perguntando se deseja habilitar automaticamente o *Karma* quando você alterar os arquivos teste, escreva “*yes*”, é interessante deixar valida essa pergunta pois é uma da formas mais eficientes de ver se tudo foi configurado corretamente, pelo fato de revalidar automaticamente o código de teste toda vez que ele for salvo. Aperte “*Enter*”



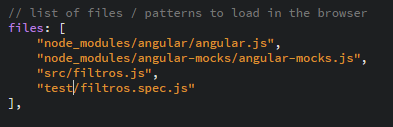
• Será retornado que foi gerado o arquivo, verifique na pasta do projeto se foi realmente gerado o arquivo “*karma.config.js*”. Abra ele



• Verifique o campo de “*files*”, ele deve estar vazio, pois não foi declarado nenhum conteúdo anteriormente no *Prompt*, hora de alterá-lo

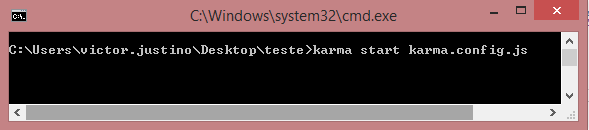


1. Preencha entre aspas duplas a localização especifica de cada arquivo a ser utilizado para o processamento do teste, incluindo os que serão os da aplicação, no caso deste exemplo, esse campo fica preenchido como na imagem abaixo

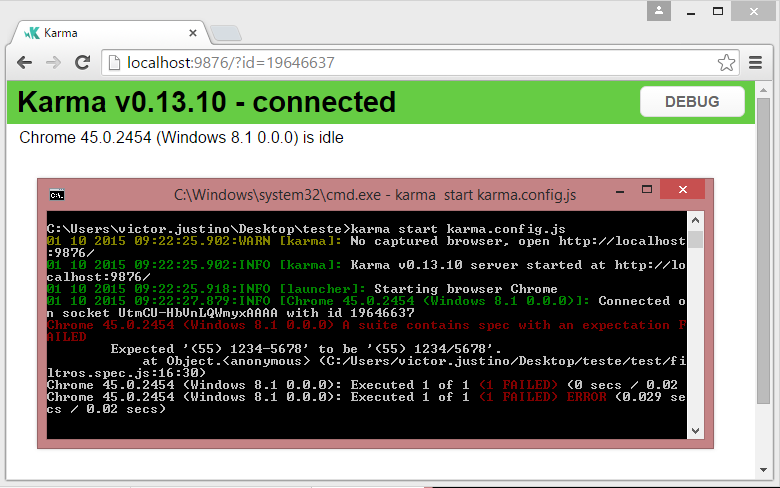
r

1. Observação: foi solicitado colocar o nome especifico do caminho de cada arquivo, pois o *Karma* as vezes enfrenta dificuldade para encontra-los. Fazendo dessa maneira é mais tranquilo as coisas serem processadas corretamente

• Abra o *Prompt* novamente e digite “*karma start karma.config.js*”, ele irá começar o processamento de teste, abrirá uma janela do *Chrome* (não adianta fechá-la, enquanto estiver rodando o teste ela será reaberta)



• Se os passos anteriores foram seguidos corretamente, o resultado do teste será um erro como está sendo mostrado na imagem abaixo. Isso acontece porque o resultado obtido não foi equivalente ao resultado esperado. Se a mensagem de erro que aparecer for diferente, algum passo anterior foi realizado erroneamente



• Para obter um resultado positivo, vá no arquivo “filtros.spec.js”, na linha 13, no método “*toBe*”. Dentro deste método tem uma *string* que vai ser comparada ao resultado do teste o valor dela está ‘(55) 1234/5678’, altere a barra(“/”) por um hífen (“-”) e salve. Se tudo foi montado corretamente desde as configurações iniciais, o próprio *karma* irá revalidar o teste e irá retornar uma mensagem de sucesso parecido com a da imagem abaixo

