**O que é Angular?**

Angular é uma plataforma e framework para construção da interface de aplicações usando HTML, CSS e, principalmente, JavaScript, criada pelos desenvolvedores da Google.

Ele possui alguns elementos básicos que tornam essa construção interessante.

Dentre os principais, podemos destacar os componentes, templates, diretivas, roteamento, módulos, serviços, injeção de dependências e ferramentas de infraestrutura que automatizam tarefas, como a de executar os testes unitários de uma aplicação.

Angular nos ajuda a criar *Single-Page Applications* com uma qualidade e produtividade surpeendente!

Alguns outros pontos dessa plataforma que merecem destaque são o fato de que ela é *open source*, possui uma grande comunidade, existem várias empresas utilizando e tem muito material de estudo para quem deseja se aperfeiçoar.

**Angular vs AngularJS**

É importante deixar claro que nós estamos falando de **Angular**, ou **Angular 2+**. Não estamos falando de AngularJS 1.x.

Caso você seja novo no mundo Angular, deve estar se perguntando: Mas qual é a diferença? É só a versão?

Na verdade, são tecnologias **completamente diferentes**!

AngularJS é um framework JavaScript para desenvolvimento web. Foi o framework queridinho no mercado por alguns anos.

Mas para acompanhar a evolução da tecnologia, os desenvolvedores perceberam que seria melhor criar um novo framework do zero, usando toda a experiência que tiveram com o AngularJS e necessidades dos desenvolvedores.

E então, surgiu o Angular 2, uma verdadeira plataforma para desenvolvimento de aplicações não só web, mas também mobile, com mudanças significativas na sua estrutura.

Obviamente, uma aplicação desenvolvida com AngularJS não é compatível com Angular.

E claro, essa grande mudança deixou alguns desenvolvedores preocupados. Ninguém gosta de saber que a tecnologia que está usando vai ser substituída por uma nova, né?

Embora o projeto do AngularJS continue sendo mantido, essa mudança fez com que ele perdesse força no mercado, dando lugar para o novo Angular.

E não seria nenhuma novidade pra gente se passasse por sua cabeça: “E se eu começar a usar o novo Angular e acontecer isso outra vez, de recriarem o framework?“.

Embora essa possibilidade exista, na nossa opinião ela é muito pequena. E como desenvolvedores de software, nós estamos sempre suscetíveis a isso com **qualquer tecnologia**. E por um lado, achamos isso muito bom!

Se eles precisaram recriar o framework para entregar pra gente uma nova ferramenta muito melhor, porque isso não seria bom?

Uma das únicas certezas que temos nessa área de desenvolvimento de software é que, o que usamos hoje não será mais usado daqui alguns anos, pelo menos não da mesma forma.

Por isso mesmo precisamos estar em constante aprimoramento, lendo livros, artigos e fazendo cursos, como você está fazendo agora.

**O versionamento do Angular**

A partir do Angular 2, a tecnologia começou a mudar suas versões de uma forma diferente.

Foi lançado o Angular 4, Angular 5, Angular 6 e dependendo da época que você está lendo esse artigo, talvez Angular 7 e assim por diante.

Quem não entende o que está acontecendo, pode achar que os desenvolvedores da plataforma ficaram loucos e estão recriando a tecnologia como se não houvesse amanhã. Mas não é nada disso!

A partir do Angular 2, a tecnologia começou a usar Semantic Versioning (SEMVER).

Basicamente, o SEMVER divide o número da versão em 3 partes, sendo Major.Minor.Patch. Por exemplo, 5.2.10, 6.0.1, 6.0.2, etc.

Quando um bug é corrigido e liberado, o número do patch é incrementado. Se uma nova funcionalidade é adicionada, o número do Minor é incrementado, e se uma mudança pode potencialmente quebrar código de quem usa a plataforma, o número do Major é incrementado.

Em outras palavras, isso significa que todas as mudanças que correm o risco de quebrar código de quem usa o framework, ou seja, que potencialmente pode precisar que o desenvolvedor faça algum ajuste para continuar com o projeto funcionando, devem ser lançadas com uma mudança do número Major.

Essas breaking changes não significam uma reescrita do framework. Não significa que o mundo acabou e que você vai precisar reaprender tudo novamente. São apenas mudanças que são inevitáveis e que podem quebrar código, como acontece em 100% das tecnologias que evoluem e não estão paradas no tempo.

E isso nem quer dizer que vai quebrar código com toda certeza. Quer dizer apenas que, potencialmente, pode ter sim quebra de código, e por isso necessita de uma atenção especial por parte do desenvolvedor.

É até muito comum atualizarmos a versão do Angular e tudo continuar funcionando normalmente.

Então, você pode ficar totalmente tranquilo em relação a isso. Angular é uma tecnologia que evolui muito rápido e você, como um bom profissional de tecnologia, deve achar isso excelente!

E para encerrar essa questão sobre as versões, olha só uma curiosidade: você não vai encontrar a versão 3 do Angular.

Isso porque, simplesmente, ela não existiu. Por uma série de conflitos em suas bibliotecas, resolveu-se pular essa versão e ir direto do Angular 2 para o Angular 4.

**A linguagem TypeScript**

Para criarmos aplicações com Angular, podemos usar as linguagens TypeScript, JavaScript e Dart.

O Angular foi desenvolvido em TypeScript, e essa linguagem é a mais usada para aplicações Angular e a que mais encontramos documentações.

Nos cursos de Angular da AlgaWorks, também usamos TypeScript. Por essas e outras razões, nós recomendamos que você use essa linguagem também.

O TypeScript é uma linguagem de programação criada pela Microsoft. Ela é um superset do JavaScript, ou seja, faz tudo o que o JavaScript faz e ainda mais algumas coisas.

Uma das coisas legais ao escolher TypeScript é a possibilidade de utilizar os recursos mais recentes do JavaScript, que ainda não funcionam nos navegadores.

Os navegadores (browsers) não conseguem executar código TypeScript, por isso, o que acontece é que os desenvolvedores escrevem códigos em TypeScript e esses códigos são transpilados para JavaScript.

Transpilação é parecido com a compilação, mas ao invés de gerar um código de mais baixo nível, é gerado o código em outra linguagem, ou na mesma linguagem, só que com o objetivo de deixar compatível com o ambiente que vai ser executado.

No caso de aplicações Angular desenvolvidas com TypeScript, a transpilação é o mesmo que gerar código JavaScript a partir de código TypeScript.

A maioria das pessoas que começam com Angular não conhecem TypeScript ainda.

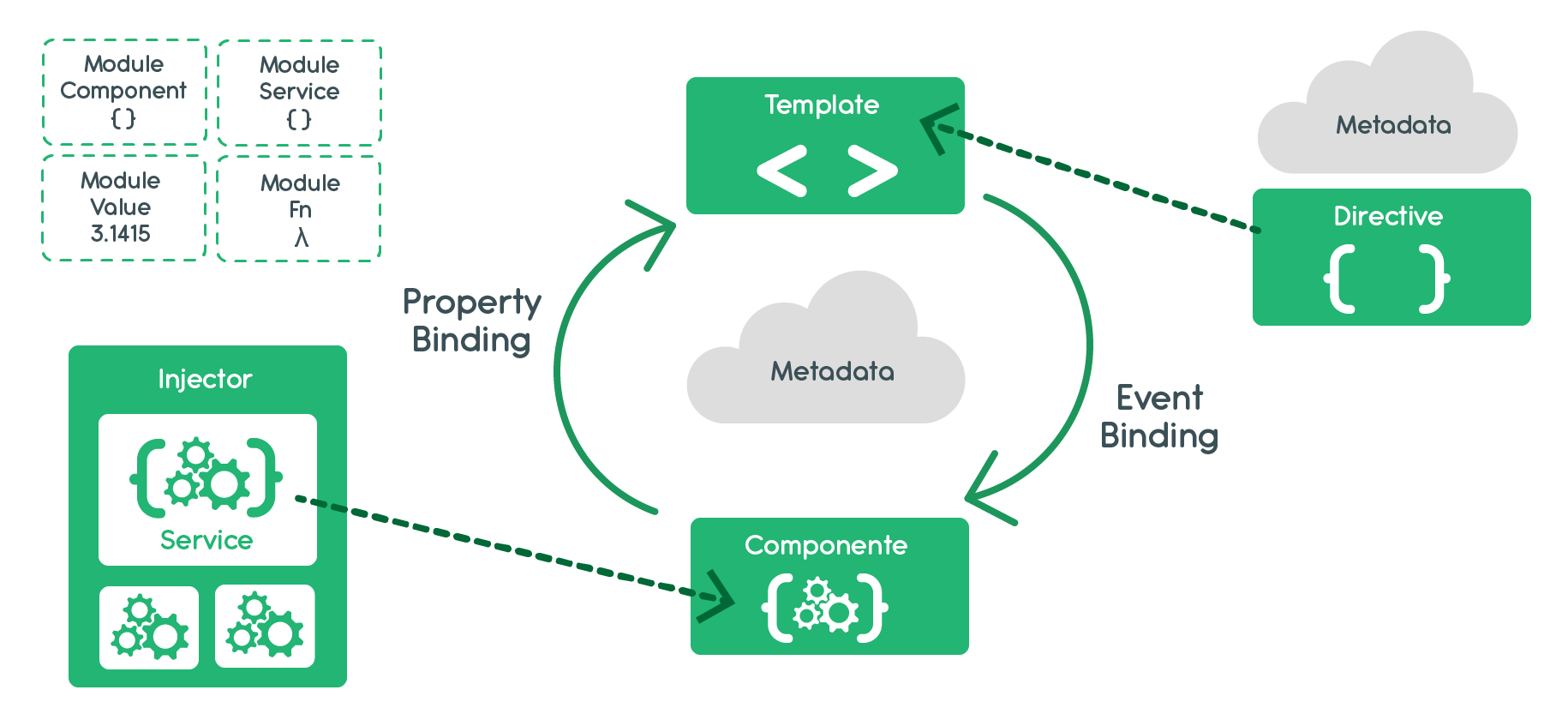
A boa notícia é que o TypeScript é uma linguagem com uma documentação excelente e bem fácil de aprender.

Melhor ainda para quem vem do Java, pois as duas linguagens são bem parecidas. Se você já é um programador Java, vai se sentir em casa.

**Elementos de uma aplicação Angular**

Os elementos básicos (building blocks) de uma aplicação Angular são:

* Módulos
* Componentes
* Templates
* Metadata
* Data binding
* Diretivas
* Serviços
* Injeção de dependências



**Componentes**

Uma aplicação Angular é baseada em **componentes**. Com eles, nós podemos encapsular comportamentos e regras da interface, o que torna a criação de aplicações algo mais simples. Inclusive, um componente pode encapsular outros componentes.

Isso é um ponto muito positivo, porque o componente pode ser reaproveitado em vários lugares da aplicação.

Um componente é composto de 3 itens:

* Template HTML
* CSS
* Uma classe que gerencia as propriedades e comportamentos

Poderíamos, por exemplo, criar uma barra de ferramentas (com botões “Salvar”, “Limpar”, “Excluir”, etc) e reutilizar ela em várias telas de cadastro.

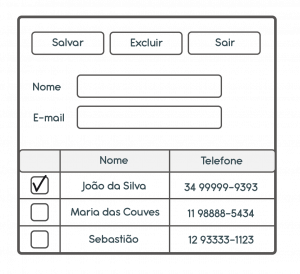
Seria possível também criar um componente de listagem de dados, com paginação para exibir resultados de uma pesquisa qualquer, e reaproveitá-lo em todas as telas de pesquisa de um sistema.

Quem sabe, criar também uma entrada de texto que já venha com a possibilidade de incluir uma máscara (de telefone ou CPF, por exemplo).

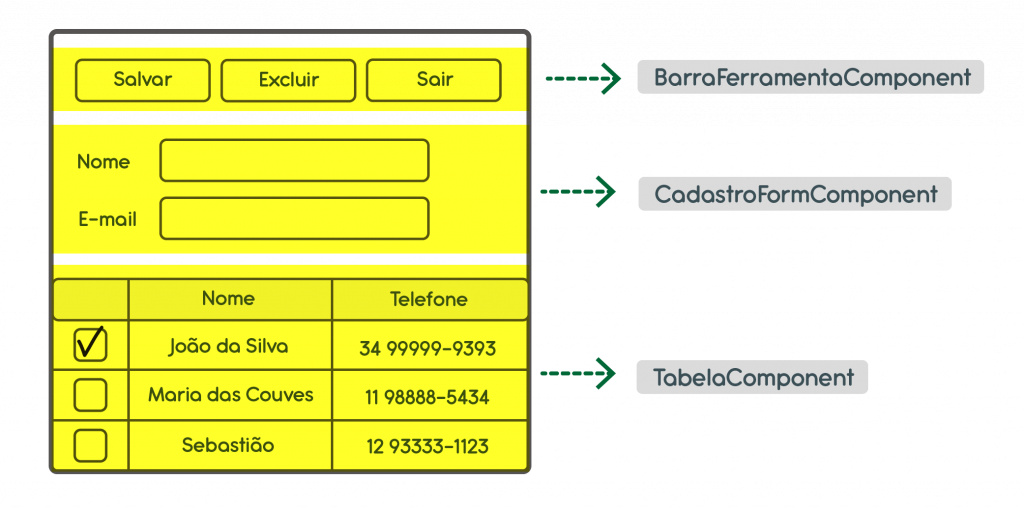
Com esses exemplos, acho que agora você já entendeu que pode criar qualquer coisa que precisar reutilizar, né?

É importante notar que mesmo uma tela completa ou parte de uma tela do sistema que não seja possível ou necessário reaproveitar também será um componente.

Ao invés de criar apenas uma página, como talvez você já esteja acostumado, você vai criar um componente que representa aquela página. Você pode criar um componente chamado CadastroUsuarioComponent e já colocar tudo o que precisa:



Você ainda poderia incluir um ou mais componentes dentro do componente que representa a sua tela, como uma barra de ferramentas, um componente de formulário e uma tabela para exibir a listagem dos registros.



Essa escolha sobre o que vai virar um componente ou não será sua e dependerá da necessidade da sua aplicação.

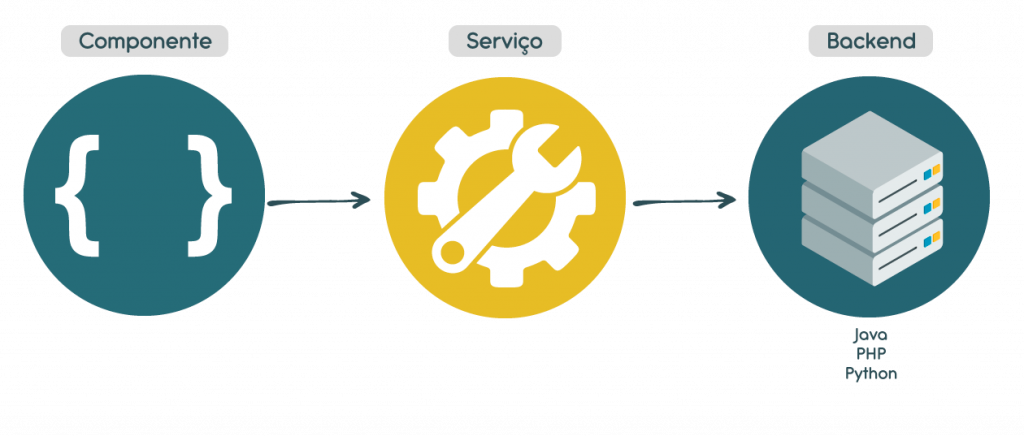
Caso esteja na dúvida sobre criar certos componentes, então não crie. Espere e, se identificar futuramente que será vantajosa a separação de alguma parte da tela em um componente isolado e reutilizável, então faça isso.

Com o tempo você vai se tornando mais ágil e conseguirá antecipar trechos reaproveitáveis na sua aplicação e que merecerão se tornar componentes.

**Serviços**

Os **serviços** são criados para concentrar todas as regras de negócio da aplicação. Cada serviço pode ser usado por mais de um componente.

Eles não são restritos a isso, mas o que normalmente acontece é que usamos serviços para encapsular o código responsável por regras com um propósito específico, como por exemplo para recuperar ou submeter dados em uma API RESTful (*back-end*).



Com isso, podemos deixar os componentes focados em gerenciar partes da tela que o usuário visualizará e os serviços focados em código com um outro propósito específico.

**Injeção de dependências**

**Injeção de dependências** é um padrão de projeto que pode ser aplicado independentemente da linguagem.

Esse padrão permite deixar as classes de componentes mais limpas e eficientes, delegando tarefas complexas e regras de negócios para os serviços.

Indo mais a fundo, injeção de dependências é o processo de prover as instâncias necessárias que uma determinada classe precisa para ser instanciada e utilizada.

A grande vantagem é que temos um menor acoplamento entre nossas classes.

O Angular implementa um mecanismo de injeção de dependências e executa ele na inicialização da aplicação.

**Módulos**

Outro elemento importante são os **módulos**. Eles ajudam a tornar nossa aplicação mais organizada.

É possível criar módulos que agrupem componentes, serviços e outros elementos por alguma semelhança.

**Bibliotecas de componentes**

Apesar de Angular ser um framework relativamente novo, já temos várias bibliotecas de componentes excelentes e gratuitas a nossa disposição.

A própria filosofia do framework, de ser baseada em componentes, cria um ambiente propício a isso.

Atualmente, as principais bibliotecas de componentes para Angular são:

* PrimeNG;
* Angular Material;
* ng-bootstrap;

E nós sabemos que nesse momento você está perguntando: **Qual é a melhor biblioteca?**

A melhor biblioteca é a que resolve o seu problema. Não existe uma resposta 100% correta para isso.

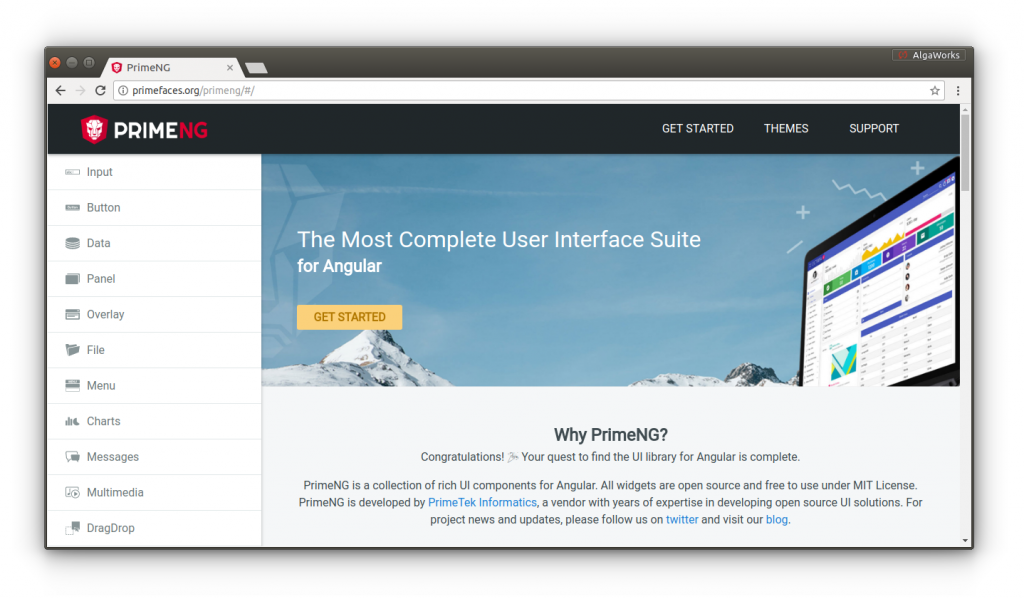
O recomendado é que você entre nos sites de cada biblioteca, conheça os componentes disponíveis, a facilidade de uso, comunidade, documentação, etc, e analise se o que existe disponível atende os seus requisitos.

Isso será importante para você tomar uma decisão na hora de escolher qual delas irá usar em seus projetos profissionais.

**PrimeNG**

O [**PrimeNG**](http://primefaces.org/primeng) é a biblioteca com o maior número de componentes e a que evolui com maior velocidade.

Ela tem uma qualidade incrível e é excelente para desenvolver projetos comerciais e corporativos.



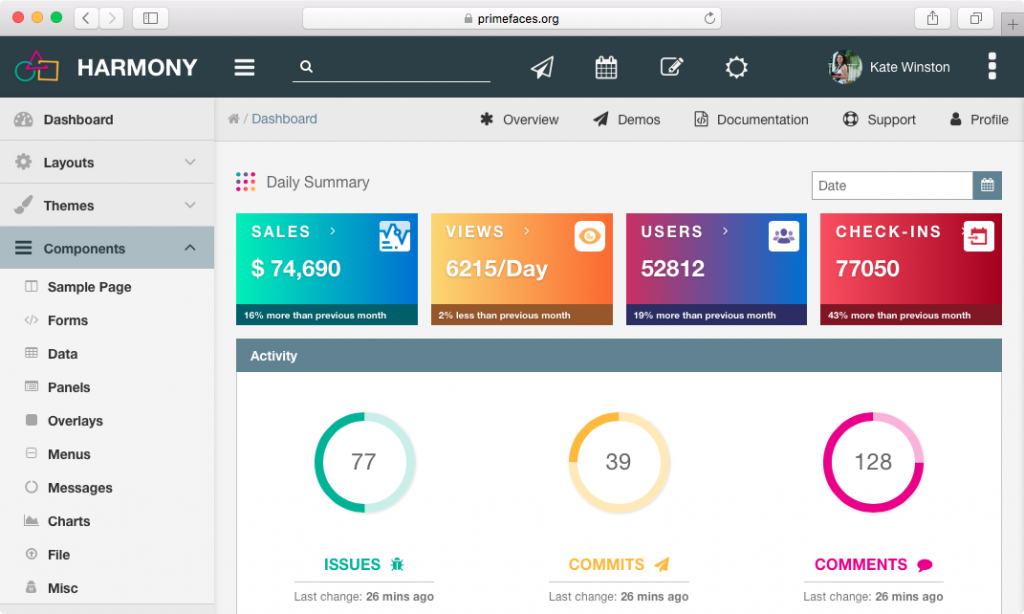
A biblioteca é desenvolvida e mantida pela **PrimeTek**, uma empresa com sede na Turquia.

A PrimeTek tem bastante experiência com desenvolvimento de bibliotecas de componentes. É a mesma empresa por trás do **PrimeFaces** (JSF), **PrimeReact** e **PrimeUI**.

As bibliotecas de componentes desenvolvidas pela PrimeTek são gratuitas e *open source*, mas você tem opção de adquirir serviços de suporte profissionais, caso ache interessante.

Além das bibliotecas, a PrimeTek também fornece temas gratuitos e comerciais (pagos), além de layouts comerciais prontos.

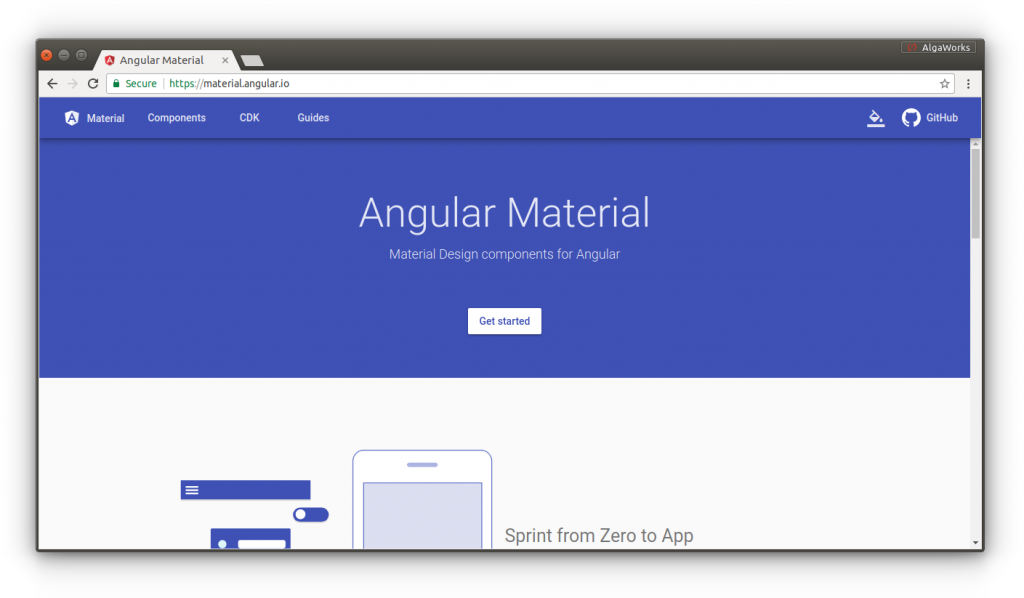
Ou seja, você pode mudar a “pele” da sua aplicação com uma infinidade de opções, e ainda pode adquirir layouts completamente prontos, com exemplos de *dashboards*, páginas de cadastro, página de login, etc.



É muito legal ter essas opções, para caso você não seja muito bom em deixar o front-end bonito e com uma boa experiência para o usuário, mas é algo totalmente opcional.

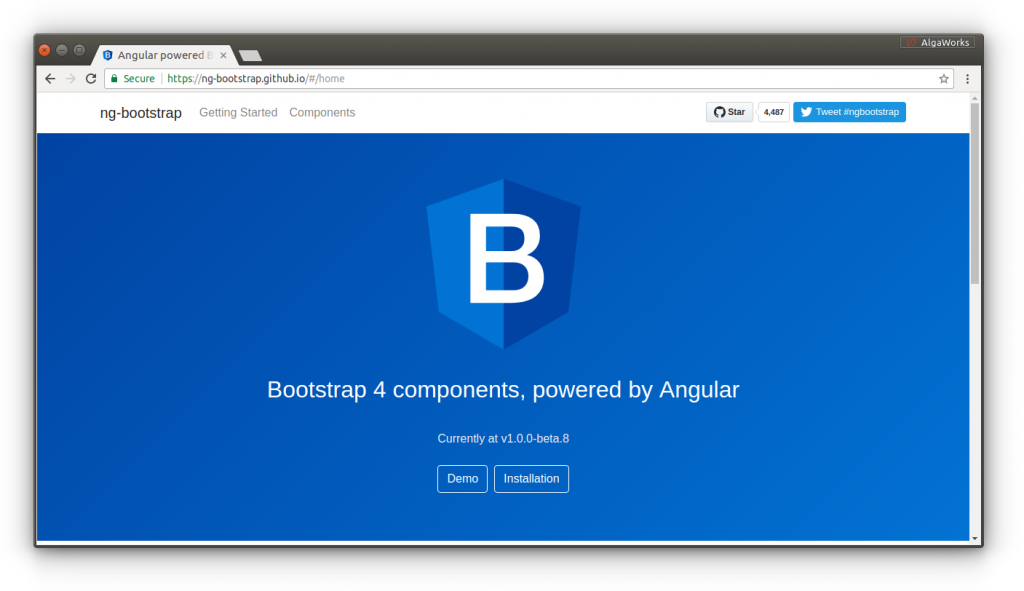
**Angular Material**

O [**Angular Material**](https://material.angular.io/) é uma biblioteca baseada no **Material Design**, criada pelo mesmo time de desenvolvedores do Angular. Já tem vários componentes bacanas e vem evoluindo muito.



**ng-bootstrap**

O [**ng-bootstrap**](https://ng-bootstrap.github.io/) é uma biblioteca desenvolvida baseada no **Bootstrap 4**, que é um framework CSS bem conhecido para criação de interfaces web.



**Vantagens**

Algumas vantagens da ferramenta que fazem as empresas utilizarem o Angular podem ser descritas como:

* **Produtividade:** desenvolver uma aplicação com ele requer bem menos código, sendo inclusive intuitivo e dando suporte para a modularização;
* **Fácil manuseamento:** muito fácil entender o funcionamento das aplicações lendo apenas o HTML;
* **Criação de frameworks:** é possível criar nosso próprio framework a partir dele;
* **Teste unitário:** é simples, pois toda a estrutura é desacoplada e as dependências são injetadas, o que facilita a criação de fakes, stubs, spies e mocks, melhorando  –  e muito – todo o processo de teste de controladores, serviços e diretivas, todos conceitos bastante utilizados pelo framework.