



Projeto de Soluções Móveis Híbridas com Ionic

Raphael Ribeiro Gomide

2018

Projetos de Soluções Móveis Híbridas com Ionic

Raphael Ribeiro Gomide

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

Sumário

Capítulo 1. Introdução	6
<i>Apps</i> híbridos x <i>apps</i> nativos.....	6
Plataforma	6
<i>Apps</i> nativos.....	7
<i>Apps</i> híbridos	8
Node.js	9
Microsoft Visual Studio Code	10
Cordova x Phonegap.....	11
<i>App</i> simples com o Cordova.....	14
Etapa 01 – Convenções	15
Etapa 01 – Instalação do Node.js.....	15
Etapa 02 – Instalação do Cordova	17
Etapa 03 – Criação de um projeto Cordova	18
Etapa 04 – Inclusão de plataformas	19
Etapa 05 – Executando o app	20
Etapa 06 – Alterando o app.....	21
Resumo do capítulo	24
 Capítulo 2. Tecnologias utilizadas pelo Ionic	25
TypeScript	25
Angular	26

Capítulo 3. Ionic Framework	30
Introdução	30
Características	30
Processo de geração de um <i>app</i> (<i>build</i>).....	32
O ionic-cli.....	33
Principais tipos de projetos.....	35
Execução de projetos Ionic	35
Estrutura de pastas de um projeto Ionic.....	37
Conteúdo da pasta <i>src</i> de projetos Ionic	39
Conteúdo da pasta " <i>app</i> ".....	40
Conteúdo da pasta <i>pages</i>	41
Capítulo 4. Componentes visuais do Ionic	42
O projeto ionic-preview-app.....	42
Componentes Visuais - ActionSheet	43
Componentes visuais - Alert.....	44
Componentes visuais – Botões (<i>Buttons</i>).....	45
Componentes visuais – Cards.....	46
Componentes visuais – Modals.....	46
Componentes visuais – SearchBar	47
Capítulo 5. Criação de um app com Ionic	49
JavaScript – trabalhando com iterações – função <i>map</i>	49
JavaScript – trabalhando com iterações – função <i>filter</i>	50
<i>Templates</i> – classes CSS condicionais com a diretiva [<i>ngClass</i>].....	51
JavaScript – trabalhando com <i>Promises</i>	52

Capítulo 6. Comunicação remota e <i>plugins</i> nativos	53
Comunicação remota com Ionic	53
Adicionando as plataformas Android e iOS	54
Cordova plugins e Ionic Native	57
Alguns exemplos de <i>plugins</i>	58
Referências	61

Capítulo 1. Introdução

Apps híbridos x apps nativos

Um dos grandes dilemas atuais, em se tratando da criação de *apps* para dispositivos móveis, é se vale realmente a pena investir em *apps* nativos para cada tipo de dispositivo móvel ou adotar soluções teoricamente mais simples com *apps* híbridos.

Antes de se obter uma opinião que possa servir de base para uma tomada de decisão importante em relação ao tipo de *app* a ser desenvolvido, é interessante conhecer bem esses dois “mundos” através de alguns conceitos fundamentais.

Plataforma

Quanto ao desenvolvimento para dispositivos móveis, o termo *plataforma* se refere ao Sistema Operacional (SO) do dispositivo móvel, que provê o SDK¹ aos desenvolvedores interessados em criar *apps*.

Atualmente é possível afirmar que as duas grandes plataformas para dispositivos móveis são o Android da Google e o iOS da Apple.

Figura 1 – Android e iOS.



Fonte: <http://www.talkofweb.com/pixel-vs-iphone/> Acesso em: 19 out. 2018.

¹ SDK – *Software Development Kit* (Kit para desenvolvimento de software, composto de bibliotecas para acesso a determinada plataforma)

Apps nativos

Apps nativos, metaforicamente falando, se situam o *mais próximo possível* do *hardware* do fabricante. Sendo assim é, em regra, a melhor forma de se explorar todo o potencial que o *hardware* do dispositivo móvel pode oferecer. A abordagem nativa é, portanto, recomendada para jogos, por exemplo.

Em geral, cada plataforma fornece, além do SDK, ferramentas específicas para o desenvolvimento. É possível citar, atualmente, o [XCode](#) da Apple e o [Android Studio](#) do Google.

Apps nativos para a plataforma Android são desenvolvidos atualmente nas linguagens de programação Java e Kotlin, sendo esta última recém suportada.

Apps nativos para a plataforma iOS são desenvolvidos atualmente nas linguagens de programação Objective-C e Swift, sendo esta última a nova *aposta* da Apple.

Com todas essas características e *sopa de letrinhas* de linguagens e tecnologias, é possível apontar algumas desvantagens em se utilizar o desenvolvimento nativo, segundo Griffith (2017):

- O desenvolvedor ou a equipe deve ter proficiência em todas as tecnologias envolvidas.
- Há pouco reaproveitamento de código entre as plataformas.
- Há um esforço maior para se manter a base de código de ambas as plataformas.

Uma alternativa aos problemas acima seria a criação de duas equipes de desenvolvimento, sendo cada uma especializada em uma das plataformas, por exemplo. Entretanto, o problema de manutenibilidade persiste mesmo assim. Além disso, a adição de uma nova equipe torna o custo maior para a empresa. Portanto, pode-se afirmar que nem sempre o desenvolvimento de *apps* nativos é a melhor abordagem, principalmente em ambientes corporativos.

Apps híbridos

Ao contrário de apps nativos, os apps híbridos são construídos com tecnologias web (HTML, CSS e JavaScript). Metaforicamente, é feita uma casca que envolve este container web (também conhecido como WebView) que interage com o dispositivo de forma nativa.

É fato que isso traz algumas desvantagens. Conforme Griffith (2017):

- O *app* fica limitado à performance do navegador do dispositivo.
- Nem sempre se garante a aparência e experiência (*look-and-feel*) inerente ao dispositivo.

A performance otimizada do *app* é a grande questão que faz com que seja feita a opção pelo desenvolvimento nativo. Entretanto, é importante frisar que os dispositivos móveis atuais evoluíram bastante em termos de performance de *hardware* (CPU e memória). Dependendo do que o *app* se propõe a fazer, essa diferença de performance entre *apps* nativos e híbridos pode ser muito pouco perceptível e o grande diferencial pode ser a experiência do usuário (transições fluidas, mensagens amigáveis, *feedback* em operações mais longas, etc.).

Quanto ao *look-and-feel*, sempre haverá clientes que demandarão *apps* com o visual padrão de cada plataforma (onde a solução nativa pode ser mais indicada) e também sempre haverá clientes que demandarão o **visual exatamente idêntico** em ambas as plataformas (onde a solução híbrida pode ser mais indicada).

Há ainda uma outra categoria de *apps* que são desenvolvidos de forma híbrida e que, durante compilação do *app*, o código-fonte é convertido para código nativo. Exemplos de tecnologias nesta categoria são: [React Native](#) do Facebook (JavaScript com *framework* baseado no React.js e CSS com *flexbox*) e [Xamarin](#) da Microsoft (utiliza a linguagem de programação C#).

O **ionic**, tópico principal desta disciplina, é um *framework* executado sobre o **Cordova** para desenvolvimento de *apps* híbridos baseados em **HTML**, **CSS** e **JavaScript** que utiliza **Angular** e **TypeScript**. Além disso, é importante destacar

que os *apps* com Ionic são desenvolvidos na plataforma **Node.js**. Mais detalhes sobre cada uma destas tecnologias serão vistos a seguir.

Node.js

O Node.js é um ambiente de *runtime* JavaScript compatível com os principais sistemas operacionais (Windows / Linux / MacOS), que permite a criação de servidores *web* com JavaScript, por exemplo. O Node.js utiliza internamente a *engine* JavaScript V8, criada pelo Google em linguagem C++ multiplataforma. Esta *engine* é também utilizada no navegador Google Chrome.

Além disso, a sua ferramenta de gerenciamento de pacotes, o **npm** (*Node Package Manager*), é conhecido por ser o maior ecossistema de bibliotecas *open source* mundial.

O Ionic é executado no ambiente do Node.js. A instalação se dá através de pacotes do npm, semelhante ao *apt* do Linux.

A tabela a seguir ilustra alguns comandos importantes do npm:

Tabela 1 – Alguns comandos importantes do npm

Comando	Descrição
node -v	Verifica a versão instalada do Node.js
npm i nome_do_pacote -g	Instalação global do pacote
npm i nome_do_pacote	Instalação local do pacote
npm i nome_do_pacote --save	Instalação local do pacote e registro como dependência do projeto
npm i nome_do_pacote --save-dev	Instalação local do pacote e registro como dependência do desenvolvedor
npm r nome_do_pacote (-g --save --save-dev)	Exclusão do pacote
npm view nome_do_pacote	Visualiza informações importantes sobre o pacote, incluindo as versões
npm i -g npm	Atualiza o próprio npm (que também é um pacote do Node.js)

Fonte: Elaborado pelo próprio professor.

O Node.js será utilizado nesta disciplina basicamente como a base do desenvolvimento de *apps*, incluindo a instalação de ferramentas e execução de comandos de criação de páginas, módulos, classes, etc., que serão vistos posteriormente.

Sendo assim, não há a necessidade de nenhum detalhamento da tecnologia, que será utilizada normalmente durante o desenvolvimento, sem necessidade de nenhuma configuração em específico, por exemplo.

Mais detalhes sobre o Node.js podem ser encontrados em <https://nodejs.org>.

Microsoft Visual Studio Code

Esta ferramenta *open source*, concebida pela Microsoft e conhecida como [VSCode](#), tem sido bastante utilizada recentemente, superando editores concorrentes como o Sublime Text e o Atom. É um editor leve que possui diversas funcionalidades interessantes e extensões, compatível com Windows, MacOS e Linux. Além de JavaScript, suporta diversas outras linguagens de programação e tem integração nativa com o *git*².

Todos os exemplos de código-fonte e aulas gravadas serão feitos no VSCode, que é a ferramenta recomendada pelo professor. Entretanto, nada impede que seja utilizado um outro editor de sua preferência.

Além disso, é recomendada a instalação das seguintes extensões³ do VSCode para facilitar o desenvolvimento de *apps* com o Ionic:

- **Angular Language Service** (implementa o *auto complete* para Angular).
- **Angular v5 Snippets** (cria blocos de código pré-definidos com atalhos).
- **EditorConfig for VSCode** (impõe regras de formatação e indentação).

² O *git* é um *software* livre para controle de versões de código. Os *apps* disponibilizados pelo professor são controlados pelo *git* e hospedados no *gitlab*. É extremamente recomendável que os desenvolvedores estudem e dominem esta tecnologia. Para mais detalhes, acesse <https://git-scm.com/>

³ Para mais detalhes sobre o VSCode, segue um bom tutorial em português feito pela equipe da DevMedia: <https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>

- **Ionic Extension Pack** (pacote com diversas outras extensões interessantes).
- **TSLint** (valida o TypeScript, que será visto posteriormente).

Para melhorar a produtividade durante o desenvolvimento, é sugerida a prática e utilização das seguintes teclas de atalho:

- Alt + Shift + ↓ - duplica linhas
- Ctrl + Shift + D - exclui linhas
- Ctrl + ← ou → - navega mais rapidamente entre os elementos
- Alt + ↓ ou ↑ - move linhas

Para mais dicas e teclas de atalho que ajudam a melhorar a produtividade no Visual Studio Code, acesse: <https://code.visualstudio.com/docs/getstarted/tips-and-tricks> Acesso em: 19 out. 2018.

Cordova x Phonegap

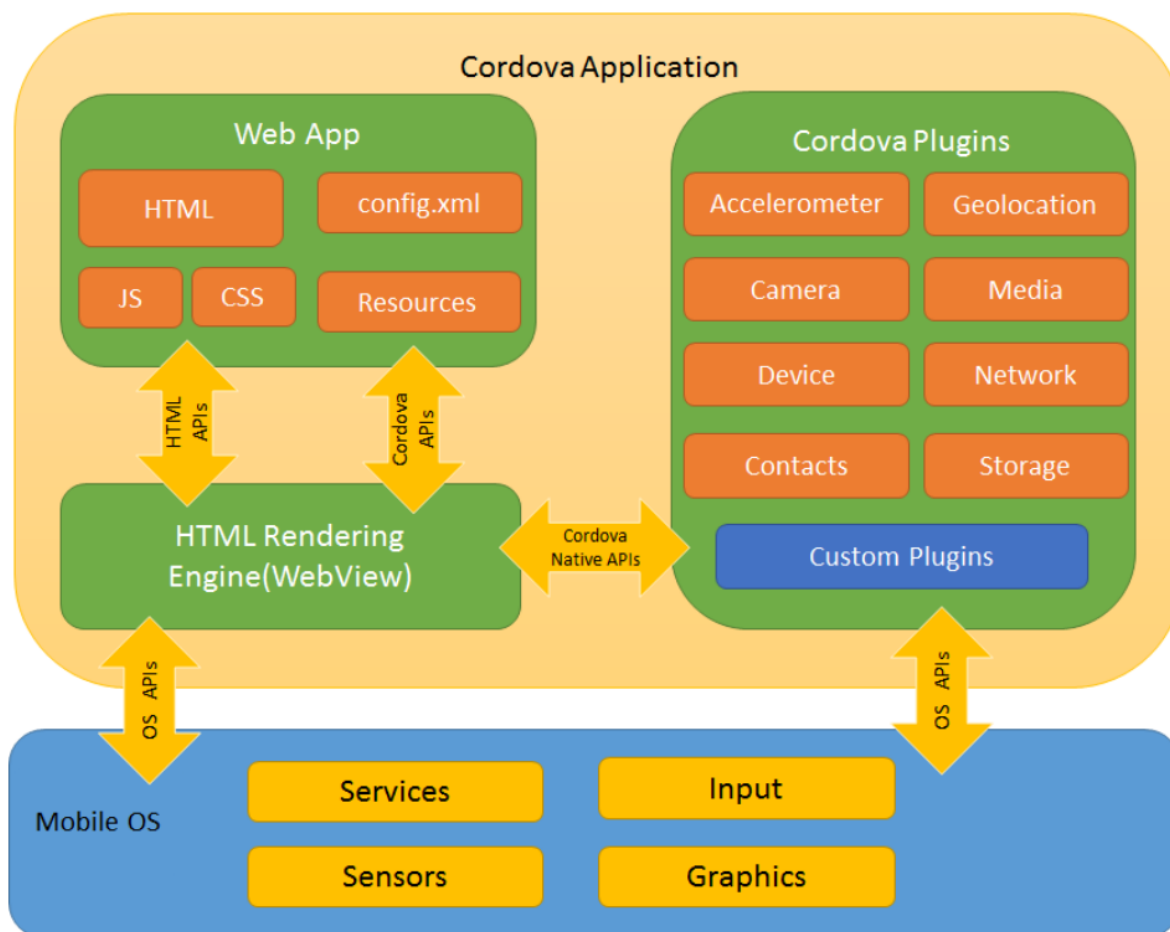
O Ionic é baseado em duas tecnologias: o [Cordova](#) e o Angular; sendo que este último será visto com mais detalhes posteriormente.

O Cordova (Apache Cordova) é uma tecnologia *open source* que permite aos desenvolvedores utilizar HTML, CSS e JavaScript para desenvolver *apps* para diversas plataformas, incluindo dispositivos móveis.

O Cordova acessa a aplicação criada pelo desenvolvedor e a renderiza no *container web* do dispositivo (também conhecido como *WebView*, como visto anteriormente). A *WebView* é um componente nativo do dispositivo responsável por processar conteúdo *web* e utilizado em diversos *apps* nativos. Segundo Griffith (2017), é interessante se pensar na *WebView* como um navegador *www* sem a barra de endereços e botões como *home*, *refresh*, etc.

A Figura 2 ilustra a arquitetura do Cordova.

Figura 2 – Arquitetura do Cordova.



Fonte: <https://cordova.apache.org/docs/en/latest/guide/overview/> Acesso em: 19 out. 2018.

Em relação à figura acima, o desenvolvedor está, na maior parte do tempo de vida de um *app*, *concentrado* apenas na camada *WebView*. O restante é administrado pelo Cordova.

A aplicação web hospedada na *WebView*, graças à constante evolução das tecnologias web, consegue executar a grande maioria das funcionalidades de componentes nativos. Entretanto, por ser uma aplicação híbrida, o acesso aos recursos nativos é limitado.

Para contornar este problema, o Cordova provê esse acesso através de *plugins*⁴ que interagem nativamente com o dispositivo e fornecem uma API JavaScript para o desenvolvedor. Alguns desses *plugins* serão vistos posteriormente.

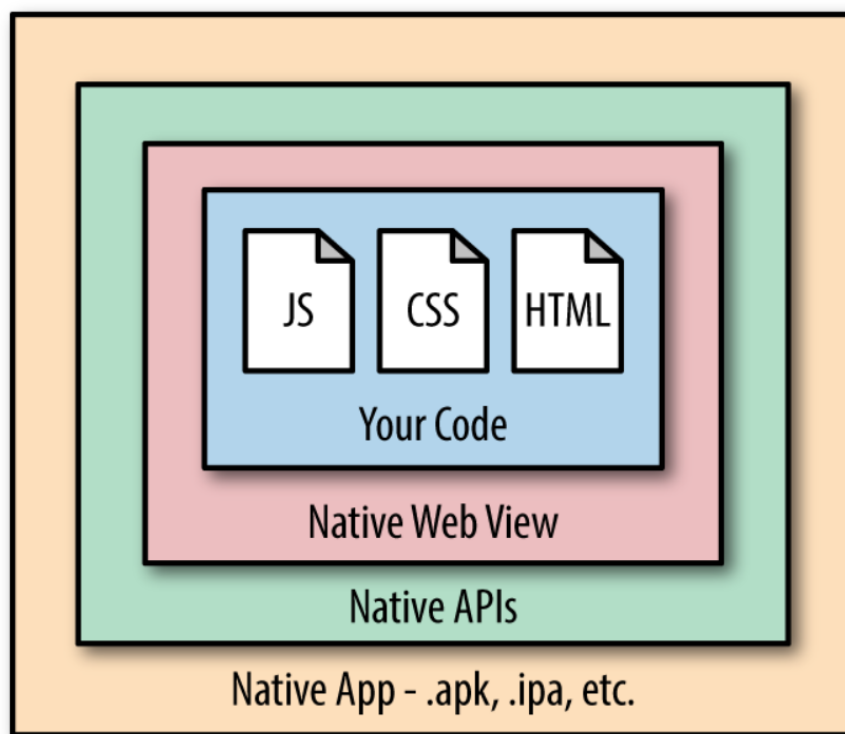
Há uma certa confusão na comunidade entre **Cordova** e [Phonegap](#). Em resumo, o Phonegap era a tecnologia inicial, que um tempo depois foi adquirida pela Adobe. A Adobe doou o código-fonte à Apache, que a partir daí concebeu o Cordova como uma tecnologia *open source*. Ambas as tecnologias existem e são bastante semelhantes atualmente. Um diferencial da Adobe, com o Phonegap, é o fornecimento de serviços de geração de *apps* em nuvem, que é o caso do [Phonegap Build](#).

É importante frisar que o Cordova suporta atualmente diversas plataformas além de Android e iOS, como por exemplo Windows, Blackberry, Linux, MacOS, dentre outras.

A Figura 3 ilustra o funcionamento básico do Cordova. Nela, é possível perceber o código do desenvolvedor (HTML, CSS e JavaScript) hospedado na *WebView*, que se comunica com API's nativas do dispositivo, que por fim forma a estrutura de um *app* completo, inerente a cada plataforma.

⁴ Para mais detalhes sobre *plugins* do Cordova, acesse <https://cordova.apache.org/plugins/>

Figura 3 – Funcionamento do Cordova.



Fonte: Griffith (2017).

App simples com o Cordova

Será criado em seguida um *app* simples para praticar alguns conceitos do Cordova, que também se aplicarão ao Ionic futuramente. Os detalhes de codificação serão vistos nas vídeo aulas. Na apostila, serão documentadas algumas etapas importantes durante a instalação, configuração e desenvolvimento do *app*. Assim, pode-se afirmar que ambos os conteúdos são complementares.

Etapa 01 – Convenções

Considerando que o JavaScript e suas ferramentas evoluem muito rapidamente, ficou definido que será utilizada a seguinte configuração de ambiente (versões estáveis mais atuais na data de elaboração da apostila):

Tabela 2 – Configurações de ambiente desejadas.

Ferramenta	Versão recomendada
Cordova	8.0.0
VScode	Versão mais recente
Ferramenta de linha de comando (cmdr)	Versão mais recente
Ionic-cli	3.20.0
Node.js	Versão LTS (<i>Long Term Support</i>) mais recente
Plataforma móvel	Android 6.0
Sistema Operacional	Windows 10

É desejável que seja utilizada esta configuração para a elaboração e estudo dos *apps*, pois assim fica mais fácil detectar problemas, caso existam. É claro que nada impede que seja utilizado versões anteriores do Windows, Linux ou o MacOS, por exemplo. A utilização do Windows foi definida por ser o Sistema Operacional mais utilizado em geral. Dúvidas e problemas pertinentes ao ambiente desses SO's poderão ser sanadas no fórum.

Etapa 01 – Instalação do Node.js

A primeira etapa é garantir que o Node.js esteja instalado de forma global (padrão). Recomenda-se a instalação da versão LTS (*Long Term Support*). Na data

de elaboração desta apostila, o Node.js se encontrava nas seguintes versões, conforme a Figura 4.

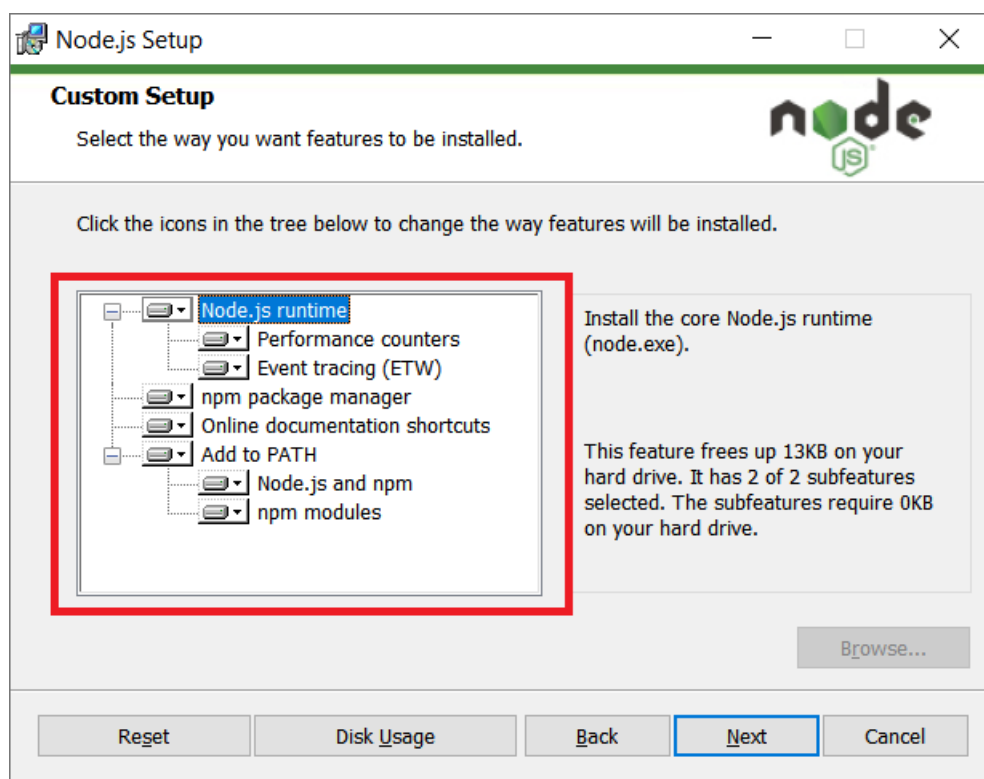
Figura 4 – Versões atuais do Node.js, com destaque para a versão LTS.



Fonte: <https://nodejs.org/en/> Acesso em: 19 out. 2018.

Para a instalação no ambiente Windows, certifique-se de que todas as opções estejam marcadas, conforme imagem abaixo. Isso permite que o Node.js seja executado globalmente, por exemplo (caminho do executável adicionado ao *path* do sistema):

Figura 5 – Configurações de instalação do Node.js em Windows.

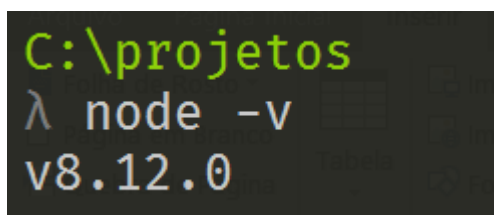


Fonte: Elaborado pelo próprio professor.

Caso encontre alguma dificuldade na instalação do Node.js, será aberto um tópico no fórum para discussões.

Para testar se o Node.js foi instalado corretamente, execute a instrução "node -v" no *prompt* de comando e confira o resultado. Para que o Node.js funcione sem problemas, o acesso ao comando node deve ser global, ou seja, a pasta de instalação do Node deve estar contida no *path* do Sistema Operacional sendo, portanto, acessível de qualquer pasta. Esta é a configuração padrão quando se instala o Node.js.

Figura 6 – Teste com o Node.js.



```
C:\projetos
λ node -v
v8.12.0
```

Fonte: Imagem elaborada pelo próprio professor.

Etapa 02 – Instalação do Cordova

A partir de agora, todas as ferramentas serão instaladas através do próprio Node.js, que possui o gerenciador de pacotes **npm**.

Para a instalação do Cordova, execute o seguinte comando⁵ no terminal, conforme figura abaixo: "npm install -g cordova@8.0.0"

⁵ A utilização de @X.X.X ao final do pacote força a instalação de uma versão específica do mesmo. Sem essa notação, o npm instala a versão mais recente.

Figura 7 – Instalação do Cordova.

```
C:\projetos
λ npm install -g cordova@8.0.0
```

Fonte: Imagem elaborada pelo próprio professor.

O comando indica ao *npm* para **instalar** o Cordova na versão **8.0.0** de forma **global** (parâmetro **-g**).

Para testar se o Cordova foi instalado corretamente, execute "cordova -v".

Figura 8 – Teste de funcionamento do Cordova.

```
C:\projetos
λ cordova -v
8.0.0
```

Fonte: Imagem elaborada pelo próprio professor.

Etapa 03 – Criação de um projeto Cordova

Escolha uma pasta de sua preferência (sugestão: c:\projetos\cordova) e execute o seguinte comando: "cordova create appteste br.com.rrgomide.appteste AppTeste".

Figura 9 – Criação de um projeto Cordova.

```
C:\projetos\cordova
λ cordova create appteste br.com.rrgomide.appteste AppTeste
Creating a new cordova project.
```

Fonte: Imagem elaborada pelo próprio professor.

Este comando realiza a criação⁶ de um projeto Cordova, informando o **caminho** do *app* (pasta 'appteste'), o **identificador único** do *app* (br.com.rrgomide.appteste) e finalmente o **nome** do *app* (AppTeste). Para mais detalhes sobre a criação de apps com o Cordova, digite "cordova help create".

Em seguida acesse a pasta recém-criada com o comando "cd appteste" e digite um dos comandos a seguir para listar os arquivos ("ls" / "ls -la" / "dir"):

Figura 10 – Estrutura de arquivos de um projeto Cordova.

```
C:\projetos\cordova
λ cd appteste

C:\projetos\cordova\appteste (br.com.rrgomide.appteste@1.0.0)
λ ls -la
total 10
drwxr-xr-x 1 rusha 197609  0 Dec 24 10:02 ./
drwxr-xr-x 1 rusha 197609  0 Dec 24 10:02 ../
-rw-r--r-- 1 rusha 197609 17 Dec 24 10:02 .npmignore
-rw-r--r-- 1 rusha 197609 989 Dec 24 10:02 config.xml
drwxr-xr-x 1 rusha 197609  0 Dec 24 10:02 hooks/
-rw-r--r-- 1 rusha 197609 371 Dec 24 10:02 package.json
drwxr-xr-x 1 rusha 197609  0 Dec 24 10:02 platforms/
drwxr-xr-x 1 rusha 197609  0 Dec 24 10:02 plugins/
drwxr-xr-x 1 rusha 197609  0 Dec 24 10:02 res/
drwxr-xr-x 1 rusha 197609  0 Dec 24 10:02 www/
```

Fonte: Imagem elaborada pelo próprio professor.

Mais detalhes sobre a estrutura de pastas e arquivos de projetos Cordova/Ionic serão vistos no capítulo 3, que fala especificamente sobre o Ionic.

Etapa 04 – Inclusão de plataformas

Conforme visto anteriormente, o Cordova trabalha com o conceito de plataformas. Como o nosso foco será o Ionic, será exemplificado aqui somente uma das plataformas do Cordova – *browser* – que é a mais simples e pode ser utilizada na *web*.

⁶ O processo de criação de uma estrutura de arquivos inicial para projetos é conhecido como *scaffolding*.

Para adicionar a plataforma, execute o seguinte comando: "cordova platform add browser":

Figura 11 – Inclusão de plataformas no cordova.

```
C:\projetos\cordova\appteste (br.com.rrgomide.appteste@1.0.0)
λ cordova platform add browser
Using cordova-fetch for cordova-browser@~5.0.0
Adding browser project...
Creating Cordova project for cordova-browser:
  Path: C:\projetos\cordova\appteste\platforms\browser
  Name: AppTeste
Discovered plugin "cordova-plugin-whitelist" in config.xml. Adding it to the project
Installing "cordova-plugin-whitelist" for browser
Adding cordova-plugin-whitelist to package.json
Saved plugin info for "cordova-plugin-whitelist" to config.xml
--save flag or autosave detected
Saving browser@~5.0.2 into config.xml file ...
```

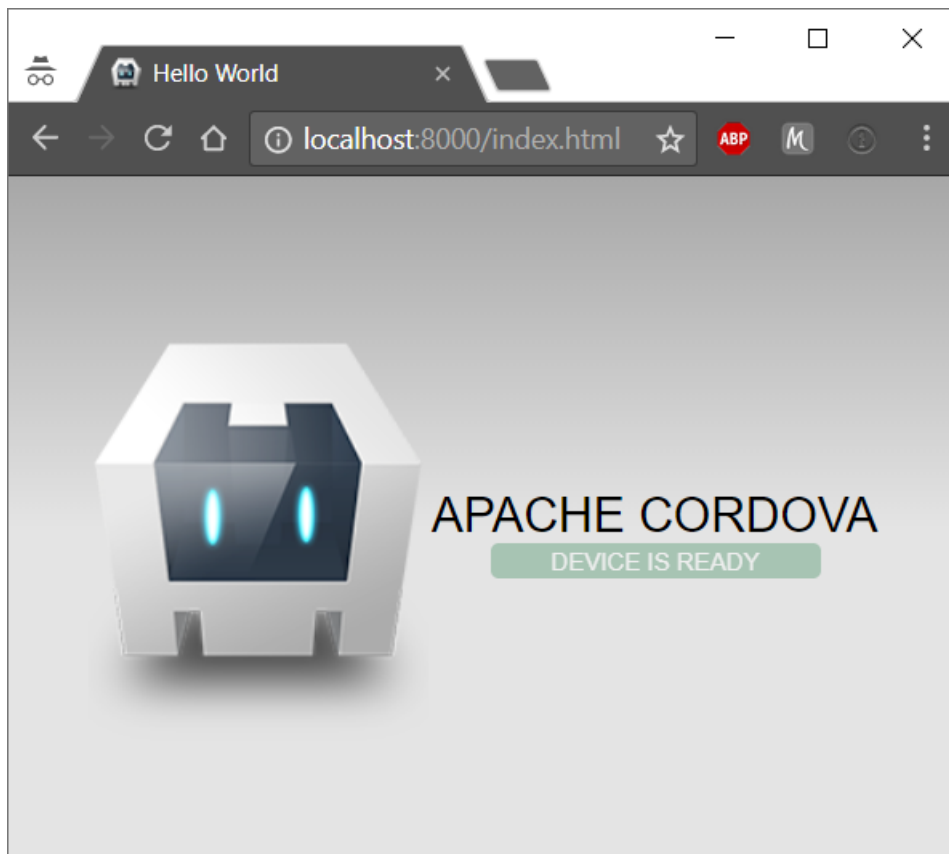
Fonte: Imagem elaborada pelo próprio professor.

Etapa 05 – Executando o app

Uma das maneiras de se executar o *app* é a seguinte: "cordova run browser". Este comando não se encerra sozinho – ele cria um servidor *web* local, que abre o navegador padrão automaticamente criando uma nova aba que geralmente *aponta* para o seguinte endereço: <http://localhost:8000/index.html>.

Figuras 12 e 13 – Executando um projeto Cordova.

```
C:\projetos\cordova\appteste (br.com.rrgomide.appteste@1.0.0)
λ cordova run browser
startPage = index.html
Static file server running @ http://localhost:8000/index.html
CTRL + C to shut down
200 /index.html (gzip)
200 /css/index.css (gzip)
200 /js/index.js (gzip)
200 /cordova.js (gzip)
200 /img/logo.png
200 /cordova_plugins.js
200 /favicon.ico (gzip)
```



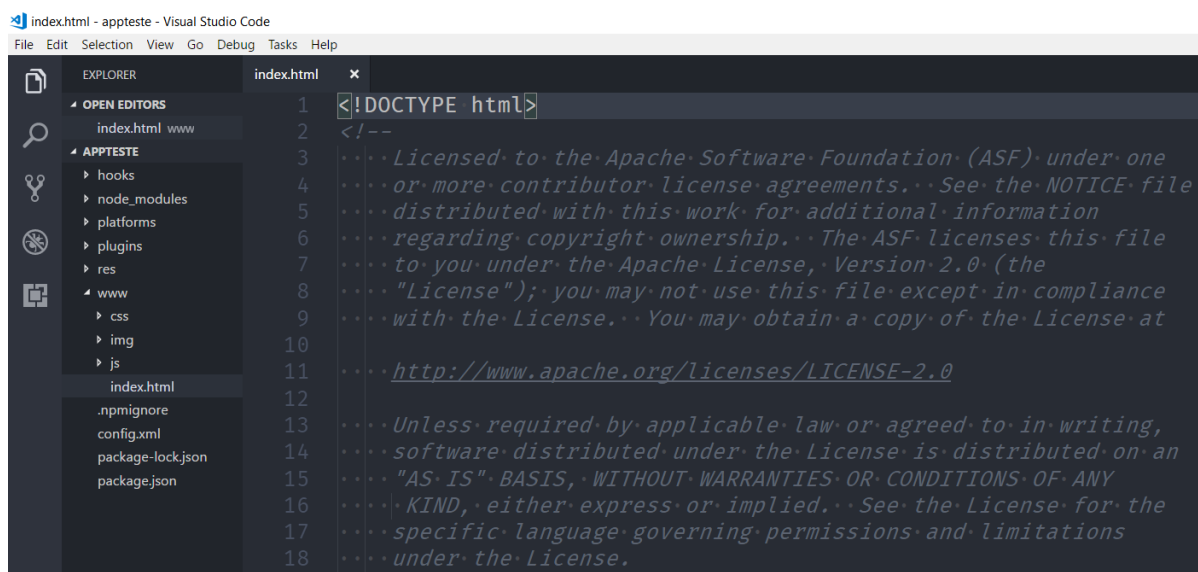
Fonte: Imagem elaborada pelo próprio professor.

Etapa 06 – Alterando o app

Mantenha o Cordova *rodando* no terminal. Para acessar os arquivos do seu projeto via terminal, abra uma nova aba ou um novo terminal. Acesse a pasta do *app* com o VSCode⁷ e abra o seguinte arquivo: "www\index.html"

⁷ Dica: se o VSCode foi instalado corretamente, ele pode ser acionado no terminal com o comando *code*. Assim, através do terminal, com a pasta do seu projeto aberto, digite "code ." que o VSCode abrirá com a estrutura de pastas do seu projeto.

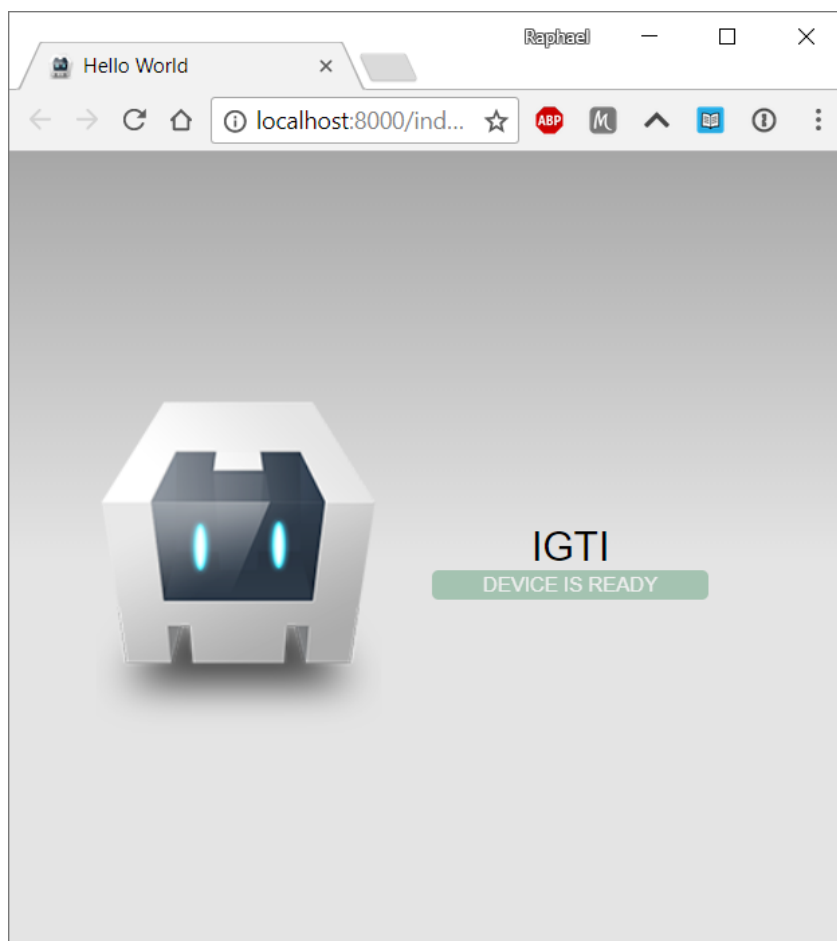
Figura 14 – Abrindo o projeto Cordova com o VSCode.



Fonte: Imagem elaborada pelo próprio professor.

Localize a tag `<h1>Apache Cordova</h1>` (linha 40) e altere o texto para `<h1>IGTI</h1>`. Salve o arquivo e recarregue a página no navegador. Perceba que a alteração não foi refletida. Isso acontece porque é necessário reiniciar o servidor web. Vá até a aba do terminal onde o Cordova está sendo executado e cancele a execução (Ctrl + C). Em seguida, execute o comando "cordova run browser" novamente e perceba que a alteração foi finalmente refletida.

Figura 15 – Alteração refletida no navegador.



Fonte: Imagem elaborada pelo próprio professor.

Nas vídeo-aulas deste capítulo serão vistos mais detalhes sobre o desenvolvimento para o Cordova.

É possível perceber que esse ciclo (iniciar o servidor, realizar alterações, reiniciar o servidor para refletir as alterações) é entediante e demorado. Isso pode ser resolvido com a inclusão de *plugins*.

Será visto posteriormente que o Ionic já inclui várias facilidades e funcionalidades para agilizar o processo de desenvolvimento de *apps*.

Resumo do capítulo

Neste capítulo, foram vistos os seguintes tópicos:

- Introdução ao desenvolvimento de *apps*.
- Diferenças entre *apps* nativos x híbridos.
- Critérios importantes para a escolha de se desenvolver um *app* nativo ou híbrido.
- Introdução ao Node.js.
- Introdução ao Cordova.
- Instalação e criação de um *app* simples com o Cordova.

No próximo capítulo serão introduzidas algumas tecnologias utilizadas pelo Ionic.

Capítulo 2. Tecnologias utilizadas pelo Ionic

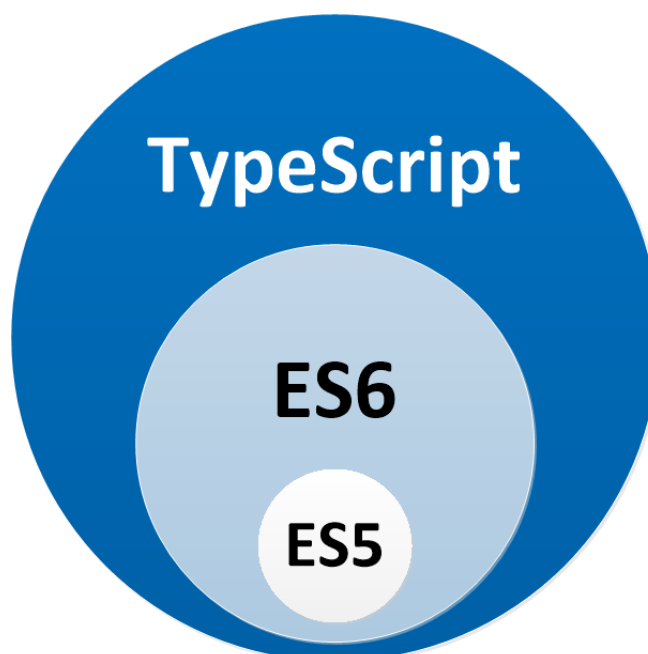
TypeScript

O *TypeScript* é uma linguagem de programação criada pela Microsoft. É *open source* e pode ser encontrada em <https://github.com/Microsoft/TypeScript>.

O TypeScript é considerado um *superset* do JavaScript e seu principal objetivo é servir de *açúcar sintático* (*syntactic sugar*) ao JavaScript através do provimento de novas funcionalidades, em especial a tipagem de dados (característica encontrada em linguagens de *backend* como o Java e o C#).

Além disso, o TypeScript provê todas as funcionalidades já existentes em novas versões do JavaScript ainda não homologadas oficialmente (ES6+). O *site* oficial do TypeScript é o <https://www.typescriptlang.org/>. A imagem abaixo ilustra melhor o TypeScript:

Figura 16 – TypeScript.



Fonte: <https://medium.com/front-end-hacking/adding-typescript-support-to-react-es6-components-f555ada645ee> Acesso em: 19 out. 2018.

O Angular adotou o TypeScript a partir de sua versão 2. Isso fez com que o Ionic também o suportasse a partir da versão 2.

A seguir, são elencadas algumas características importantes do TypeScript:

- Instalação básica do TypeScript via Node.js: “npm i -g typescript”.
- Possui verificação de tipos em tempo de compilação, reduzindo erros em produção.
- Isso pode fazer com que o custo de manutenção de um projeto diminua.
- Possui suporte a *arrays* (vetores) tipados e *decorators* (muito utilizados pelo Angular e Ionic).
- Possui um tipo especial para se referir a qualquer valor: *any*
- O *açúcar sintático* provê um melhor suporte à OO (Orientação a Objetos) com atributos e métodos públicos e privados, por exemplo.
- O funcionamento do TypeScript se dá através da compilação (alguns consideram como transpilação⁸) do seu código para JavaScript, de forma que o resultado fique compatível com a maioria dos navegadores atuais.

Mais características sobre o TypeScript, incluindo um exemplo prático podem ser vistas nas vídeo aulas do capítulo 2.

Angular

O Angular é um *framework* criado pelo Google em 2009 para a criação de SPA's (*Single Page Applications*). Atualmente, a equipe o considera não só um

⁸ O termo **transpilação** se refere a uma conversão de determinada linguagem para uma variação da mesma linguagem. Isso é muito comum em JavaScript atualmente.

framework mas sim uma plataforma de desenvolvimento. O Angular também é *open source*.

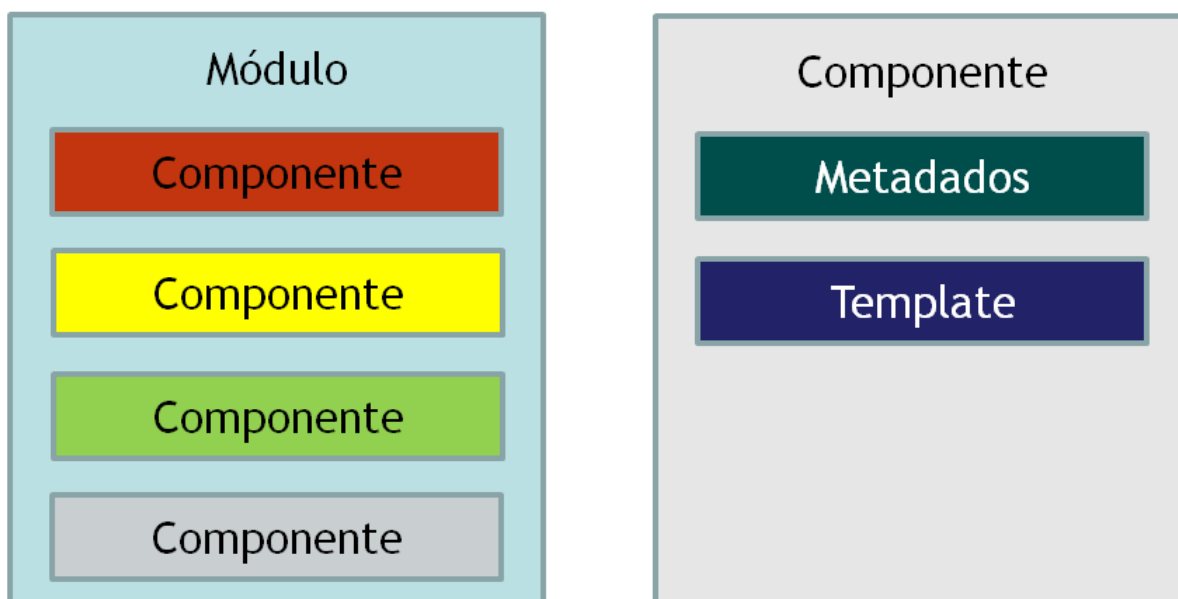
- Site oficial: <https://angular.io>
- Repositório github: <https://github.com/angular/angular>.

Atualmente, o Angular se encontra na versão 6.x (setembro/2018) e foi totalmente reescrito a partir da versão 2. Assim, ficou convencionado que a versão 1.x seria denominada AngularJS ou Angular.js para não haver confusão na busca de documentação, por exemplo.

Algumas características importantes sobre o Angular a partir da versão 2:

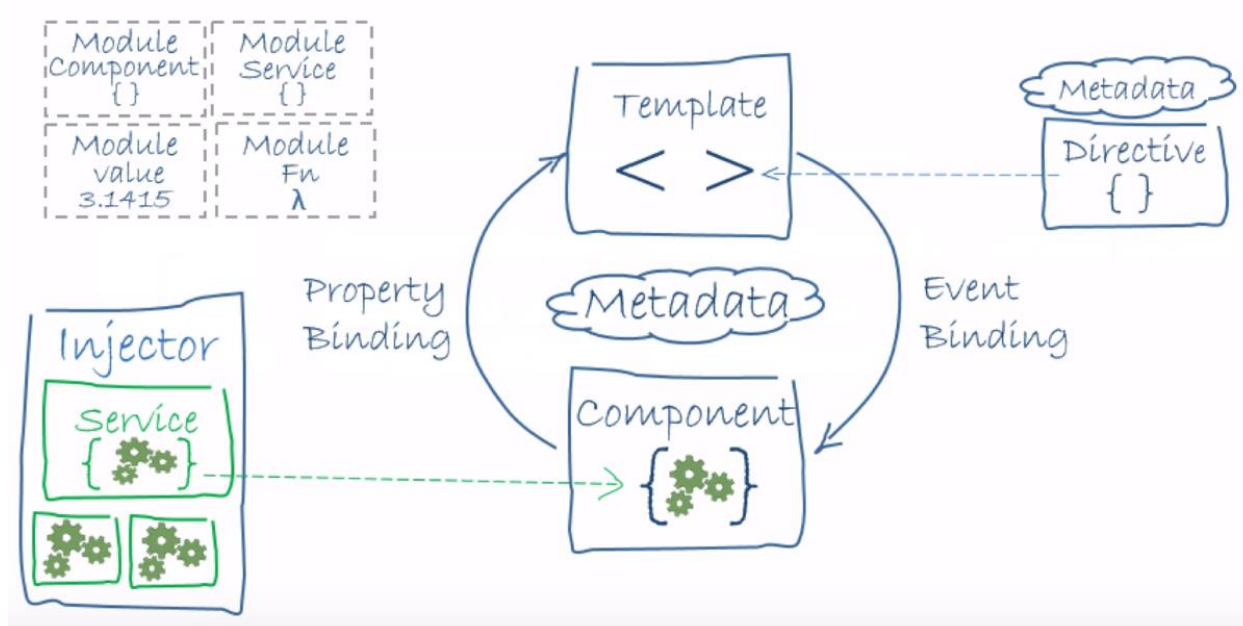
- Adesão ao *semver* (*Semantic Versioning*), que controla melhor as versões, evitando *breaking changes*.
- Utilização de TypeScript como linguagem de programação padrão.
- Para auxiliar na organização e reutilização do código, além da separação de responsabilidades, o Angular possui as seguintes entidades:
 - Módulos.
 - Componentes.
 - Metadados.
 - Diretivas.
 - *Pipes*.
 - *Templates*.
 - *Services*.
- As figuras a seguir ilustram melhor esses conceitos. Mais detalhes podem ser vistos no capítulo 3 e nas vídeo aulas dos capítulos 2 e 3.

Figura 17 – Módulos e componentes do Angular.



Fonte: Elaborado pelo próprio professor.

Figura 18 – Arquitetura do Angular.



Fonte: <https://angular.io/guide/architecture> (Acesso em 27/09/2018)

Sobre a figura acima:

- **Component:** classe que agrupa uma lógica bem definida. Possui **metadados** que identificam a classe como um componente para o Angular. Esses metadados conectam a classe a um *template*.
- **Template:** contém o código HTML vinculado ao componente. A comunicação entre componente e *template* se dá nos dois sentidos.
- Quando o sentido da comunicação é do componente ao *template*, há o **Property Binding** (vínculo de propriedades), onde um atributo do componente pode ser exibido no *template* – notação de *double curly braces* - {{ }}.
- Quando o sentido da comunicação é do *template* ao componente, geralmente ocorre o **Event Binding**, que se refere aos eventos (cliques do mouse, edição de campos via teclado, etc.) – notação de parênteses: (click)="funcao(parametro1, parametro2, ... parâmetro_n)".
- Os *templates* podem conter **diretivas** que auxiliam o desenvolvedor na criação do *layout* e disposição dos atributos do componente. As principais diretivas são o *ngIf (estrutura de decisão) e *ngFor (estrutura de repetição). Mais detalhes serão vistos nas videoaulas.
- **Serviços** são classes especiais que modelam regras de negócio e/ou comunicação remota, auxiliando na organização do código e separação de responsabilidades. Em geral, os serviços são instanciados nos componentes através de **injeção de dependência**, onde o próprio Angular *sabe* como instanciá-los, retirando uma responsabilidade a mais do o desenvolvedor. Mais detalhes serão vistos nas videoaulas.
- Todos os elementos acima podem ser agrupados em **módulos**, que podem conter um ou mais **componentes**. Esse conjunto de módulos interligados de forma organizada pode constituir um sistema Angular complexo e ao mesmo tempo mais fácil de ser mantido e evoluído.

Capítulo 3. Ionic Framework

Introdução

O Ionic *Framework* surgiu como uma camada de abstração ao Cordova para o desenvolvimento de *apps* híbridos.

Para isso, o Ionic incorporou tecnologias como o Angular e o TypeScript, vistos anteriormente.

Sua primeira versão (1.x) utilizava o Angular de mesma versão (Angular.js [1.x]) como *framework* JavaScript. A partir da versão 2 o Ionic passou a adotar versões superiores do Angular e de certa forma ambas as tecnologias evoluem semelhantemente. Atualmente a versão **mais estável** do Ionic se encontra na versão 3.20.0 (Setembro/2018)⁹ e possui suporte ao Angular (5.x).

Em relação ao Cordova, o Ionic depende do mesmo para a criação de *apps* e provê, juntamente com o Angular e TypeScript, diversas funcionalidades para auxiliar o desenvolvedor a desenvolver *apps* de forma mais eficiente e com mais qualidade.

Características

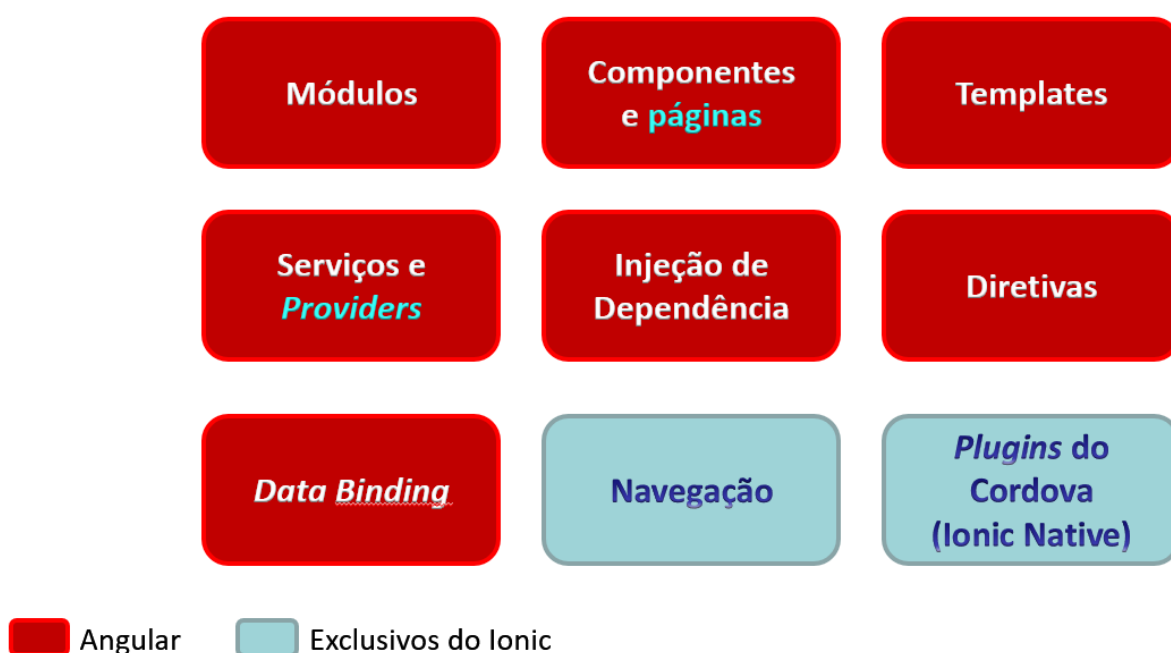
De forma semelhante ao Angular, o Ionic fornece, juntamente com seu pacote Node.js, uma ferramenta CLI (*Command Line Interface*) que realiza diversas automações, tais como:

- Criação de projetos em diversos modelos.
- Criação de *building blocks* tais como:
 - Páginas (abstração de telas).

⁹ Na verdade, a versão mais atual oficialmente é a 4.1.2 (Setembro/2018). Entretanto encontrei diversos *bugs* em algumas ferramentas que serão utilizadas ao longo desta disciplina. Assim, ficou definido que será utilizada a versão mais estável (3.20.0)

- Diretivas (apoio aos *templates*).
- *Pipes* (funções transformadoras de dados).
- Componentes (bloco de código relacionado e reaproveitável).
- *Providers* (provedores de serviços com injeção de dependência).

Figura 19 – Components do Ionic.



Fonte: Elaborado pelo próprio professor.

Analisando a figura acima é possível perceber a semelhança entre Angular e Ionic. A principal diferença é que o Ionic inclui a comunicação com o Cordova através do *Ionic Native* e possui o conceito de **páginas (*pages*)**, que são componentes especiais que representam telas do *app*, possuem funcionalidades de navegação e podem conter diversos outros componentes mais simples.

Mais detalhes sobre o ionic-cli podem ser vistos nas vídeo aulas dos capítulos 3, 4 e 5.

O Ionic é um *framework* bastante compatível às plataformas Android e iOS. Quanto à interface gráfica, seus componentes modelam as *guidelines* de cada dispositivo de forma bastante fiel.

Outra característica importante do Ionic é a excelente documentação (em inglês) e uma comunidade bastante ativa. Isso ajuda muito os iniciantes nesta tecnologia. O *site* oficial do Ionic é o <https://ionicframework.com/> e o repositório de código-fonte do GitHub é o <https://github.com/ionic-team/ionic>.

O lema do Ionic é: "*Write once, run everywhere*" (escreva uma vez e rode em todo lugar). Ou seja: com apenas uma base de código é possível gerar *apps* tanto em plataforma Android quanto iOS.

Por fim, uma grande característica do Ionic é o suporte a PWA's (*Progressive Web Apps*) que muitos defendem que será o futuro dos *apps* – fim da dependência de lojas como Google Play e App Store e destruição dos *apps* através dos respectivos *sites*. O assunto de PWA's fogue do escopo desta disciplina, mas é aconselhado que seja estudado pois ainda é, de certa forma, novidade no mercado.

Processo de geração de um *app* (*build*)

Como todo *framework* baseado em JavaScript, toda essa abstração de módulos e componentes é feita apenas durante a etapa de desenvolvimento. Quando chega a hora de gerar o *site/app* em produção, é necessário empacotar todo o código-fonte e gerar *bundles* (agrupamento de código). Quanto menor o *bundle*, menos tempo de *download* e mais probabilidade de eficiência do *app*. Lembre-se de que, no fim das contas, *sites/apps* híbridos são puramente HTML, CSS e JavaScript que são executados em um *browser/WebView* respectivamente.

Para a geração do *bundle*, o Ionic utiliza atualmente o Webpack (<https://webpack.js.org>) para *orquestrar* essa geração de *apps*.

Em resumo, o Ionic utiliza um pacote à parte (*ionic-app-scripts*) para configurar e orquestrar o *build*. A configuração padrão já faz inúmeras transformações importantes, mas pode ser customizada pelo desenvolvedor. O *webpack* por fim executa as seguintes tarefas (fonte: GRIFFITH (2017)):

1. O código TypeScript é *transpilado* para ES5 (mais compatível).
2. Compilações diversas para otimização.
3. Os módulos são unidos (*bundle*).
4. É feito o *tree shaking* (remoção de módulos e código desnecessário).
5. É feito o *bundle* do CSS a partir dos arquivos Sass (scss).
6. É feito o *minify* do código JavaScript.
7. Compressão do CSS.
8. Arquivos copiados para *www*.
9. Ativação do *live reloading*.

O ionic-cli

O Ionic possui um CLI (*Command Line Interface*) muito poderoso. O CLI do Ionic é integrado ao CLI do Cordova e com ele(s), é possível realizar diversas automações, tais como:

- Criação de novos projetos.
- Criação de novos elementos em projetos, tais como:
 - Páginas.
 - *Providers*.
 - Diretivas.
 - *Pipes*.
 - Etc.
- Inclusão/remoção de plataformas (Android/iOS)
- Etc.

Para efetuar a instalação do Ionic execute, no terminal, o seguinte comando:

npm install -g ionic@3.20.0

Lembre-se de forçar a versão 3.20.0 com a notação de @, senão será instalada a versão 4.1.2 que se encontra atualmente instável.

É importante salientar que esse processo de geração de código-fonte estrutural de projetos, independente da tecnologia aplicada, é conhecido na comunidade como *scaffolding* do projeto. Isso auxilia na eficiência e produtividade do processo de desenvolvimento de *apps/software*, já que muitos erros manuais são evitados.

O Ionic não nos obriga necessariamente a decorar comandos. O comando *ionic start* traz um assistente (*wizard*) que guia o desenvolvedor a criar o projeto da maneira mais adequada possível.

Considerando a versão 3.20.0 do Ionic, o comando "*ionic start*" realiza as seguintes etapas durante o assistente:

1. Verifica atualizações no ionic-cli e pergunta se desejamos instalá-las.
2. Solicita o nome do projeto.
3. Solicita o tipo do projeto (detalhes a seguir).
4. Pergunta se desejamos adicionar as plataformas Android/iOS.
 - a. Em caso negativo, podemos adicioná-las posteriormente.
5. Realiza o *download* das dependências do projeto (*npm install*).
 - a. É a etapa mais demora e necessita de conexão com a Internet.
6. Pergunta se desejamos adicionar o projeto ao Ionic Pro.
 - a. O Ionic Pro é um serviço de *cloud* do Ionic.
 - b. Por padrão, a opção é *Yes*.
7. Finalização da criação do projeto.

- a. Criada pasta do projeto com todo o *scaffolding* necessário.

Principais tipos de projetos

A figura a seguir mostra os principais tipos de projetos que podem ser criados pelo ionic-cli:

Figura 20 – Principais tipos de projeto criados pelo ionic-cli.

Projeto	Descrição
<i>tabs</i>	Abas.
<i>blank</i>	Em branco.
<i>sidemenu</i>	Menu de navegação lateral.
<i>super</i>	Abas e vários exemplos de telas.

Fonte: Elaborado pelo próprio professor.

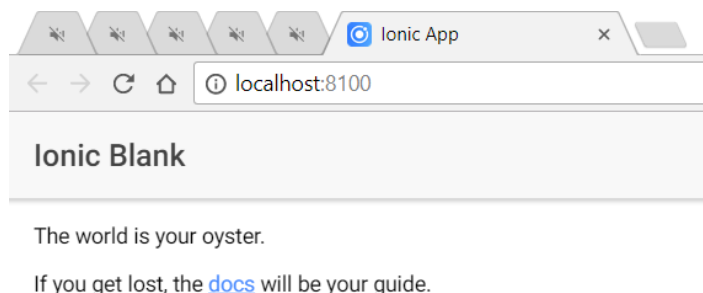
Execução de projetos Ionic

Existem basicamente duas formas de execução de projetos. Para ambas, é necessário que o desenvolvedor esteja situado na pasta raiz do projeto:

1. Comando *ionic serve*.

- a. O ionic-cli cria um servidor *web*.
- b. O navegador padrão é aberto.
- c. É criada uma nova aba *apontando* para o servidor.
- d. Por padrão, o servidor *aponta* para a porta 8100.
- e. Por padrão, é ativado o *live reloading*.
- f. Toda modificação no código é refletida automaticamente.

Figura 21 – Resultado do comando *ionic serve*.

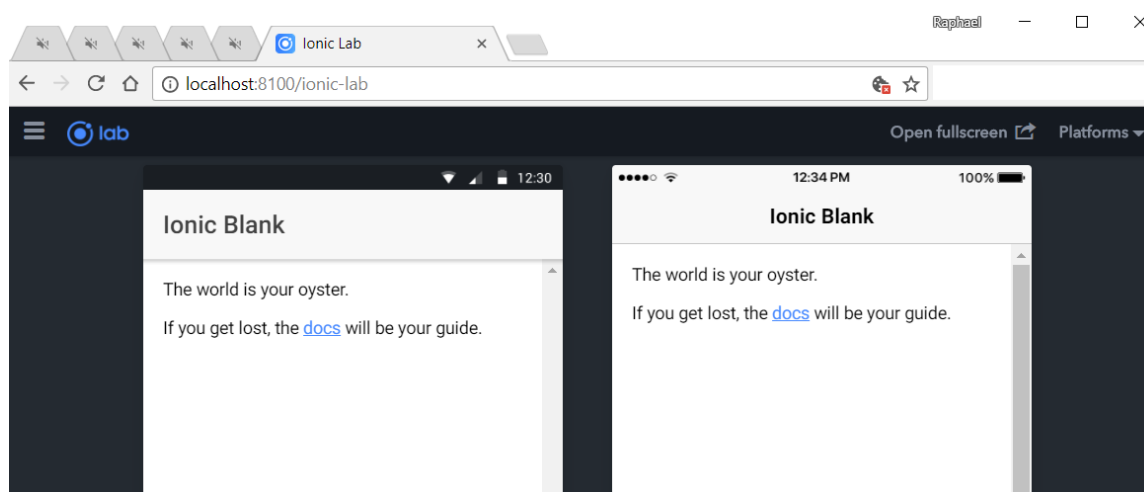


Fonte: Elaborado pelo próprio professor.

2. Comando *ionic serve –lab*.

- a. Etapas semelhantes ao comando anterior.
- b. A diferença se situa na interface do servidor *web*.
- c. São simuladas até três plataformas:
 1. Android.
 2. iOS.
 3. Windows Phone.
- d. Melhor visualização da interface.
- e. Consome mais recursos do computador.
- f. Na versão atual do Ionic (3.20.0), é feita a instalação do respectivo pacote separadamente durante a primeira execução.

Figura 22 – Resultado do comando *ionic serve –lab*.



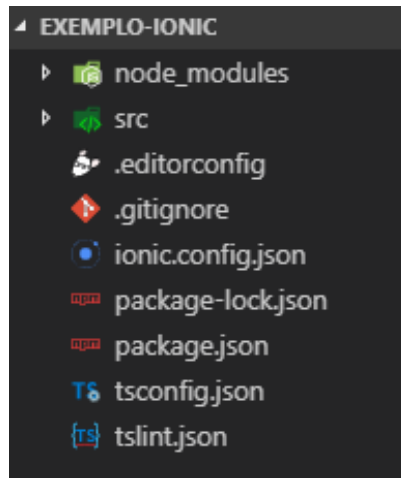
Fonte: Elaborado pelo próprio professor.

Mais detalhes sobre a execução de projetos podem ser vistos nas vídeo aulas deste capítulo.

Estrutura de pastas de um projeto ionic

Analisando um projeto Ionic, é possível perceber a quantidade de pastas e arquivos criados pelo *scaffolding*. A figura a seguir mostra a pasta *raiz* do projeto, considerando a versão 3.20.0:

Figura 24 – Pasta raiz de um projeto Ionic.



Fonte: Elaborado pelo próprio professor.

A pasta **node_modules** está presente em qualquer projeto Node.js. Ela contém o código-fonte e binários/executáveis de todas as dependências do projeto. Costuma ser, portanto, a maior pasta do projeto, com muitas subpastas e milhares de arquivos. Um detalhe importante é que, em regra, não é necessário fazer *backup* desta pasta. As dependências ficam registradas no arquivo **package.json**, que podem ser *baixadas* novamente caso necessário (troca de computador, por exemplo).

A pasta **src** é a principal pasta para o desenvolvedor, pois contém o código-fonte do projeto. Portanto, é o local onde o desenvolvedor vai atuar por mais tempo no projeto. Mais detalhes sobre o conteúdo desta pasta serão vistos a seguir.

Por fim, a pasta **www** é preenchida e controlada automaticamente pelo Ionic. Ela contém o *bundle* (todos os recursos necessários ao app compilados/transpilados). Em geral, não é manipulada pelo desenvolvedor, pois qualquer alteração direta nesta pasta é perdida durante o processo de compilação.

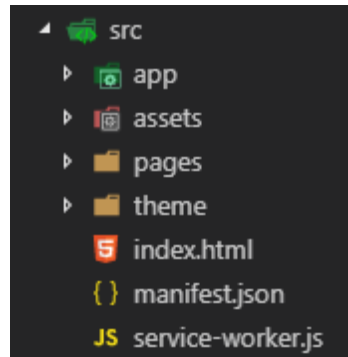
Arquivos de configuração

- **.editorconfig** – contém configurações que o editor pode utilizar para formatar o código-fonte, como por exemplo a quantidade de espaços de indentação de código. Muito útil para padronização de codificação em equipes/times.
- **.gitignore** – arquivos a serem ignorados pelo git.
- **package*.json** – lista de dependências do *app*, controlado pelo Node.js.
- **tsconfig.json** e **tslint.json** – arquivos de configuração e validação do código TypeScript do projeto.

Conteúdo da pasta *src* de projetos Ionic

A figura a seguir mostra o conteúdo da pasta **src** de projetos Ionic:

Figura 25 – Conteúdo da pasta "src".



Fonte: Elaborado pelo próprio professor.

As pastas **app** e **pages** são as mais importantes no projeto e serão detalhadas separadamente a seguir.

A pasta **assets** guarda recursos estáticos do *app*, tais como fontes e imagens.

A pasta **theme** contém o arquivo **variables.scss**, que é baseado na tecnologia **Sass** para customização de estilos do **app**. Mais detalhes sobre este arquivos podem ser vistos nas videoaulas dos capítulos 4 e 5.

O arquivo **index.html** é, assim como em sistemas *web*, o *local inicial* do **app**. Contém algumas configurações de plataformas e também agrupa os módulos do Cordova e os *bundles* gerados pelo Webpack.

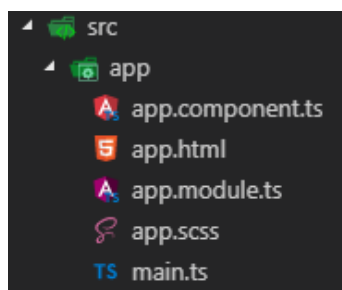
O arquivo **manifest.json** contém algumas configurações do **app**, tais como nome do **app**, localização dos ícones, etc.

O arquivo **service-worker.js** é utilizado para PWA's (*Progressive Web Apps*).

Conteúdo da pasta "app"

A imagem a seguir mostra o conteúdo da pasta **app**. Esta pasta representa o módulo principal do **app** que é carregado durante a inicialização:

Figura 26 – Conteúdo da pasta "app".



Fonte: Elaborado pelo próprio professor.

O arquivo **app.component.ts** representa o componente base do **app** que define, por exemplo, a página a ser carregada na inicialização (*RootPage*).

O arquivo **app.html** contém o *template* inicial do **app** que contém basicamente o componente Ionic de navegação `<ion-nav>` indicando a página inicial (*rootPage*).

O arquivo **app.module.ts** contém a definição de módulos, provedores e componentes a serem carregados na inicialização do **app**.

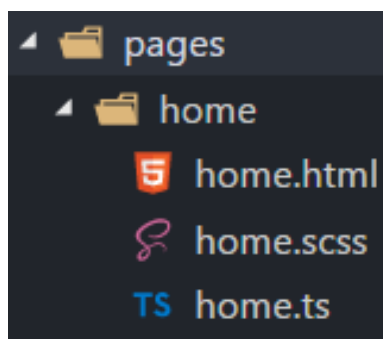
O arquivo **app.scss** pode ser utilizado para a aplicação de estilos globais.

Por fim, o arquivo **main.ts** invoca a “execução” do *app*.

Conteúdo da pasta *pages*

Esta pasta é utilizada para armazenar as telas do *app*, que no ambiente Ionic são conhecidas como páginas (*pages*). Em geral, há uma pasta por página. A imagem a seguir mostra o conteúdo básico de uma pasta do tipo *page*:

Figura 27 – Conteúdo da pasta "pages".



Fonte: Elaborado pelo próprio professor.

O arquivo **nome_da_pagina.html** representa o *template* da página e contém, em geral, código HTML com *tags* do Ionic, além de *property bindings*, eventos, etc.

O arquivo **nome_da_pagina.scss** pode conter um estilo customizado para a página, caso necessário.

Por fim, o arquivo **nome_da_pagina.ts** representa a **classe** da página e contém código TypeScript.

Mais detalhes sobre a estrutura de pastas e arquivos de projetos Ionic podem ser vistos nas vídeo aulas do capítulo 3.

Capítulo 4. Componentes visuais do Ionic

Neste capítulo serão demonstrados alguns dos componentes visuais que o Ionic já fornece por padrão ao desenvolvedor. Para mais detalhes, acesse a documentação completa do Ionic em <https://ionicframework.com/docs/components/> Acesso em: 19 out. 2018.

O projeto ionic-preview-app

Este projeto foi concebido pela própria equipe do Ionic e é utilizado na documentação oficial. O projeto pode ser localizado em <https://github.com/ionic-team/ionic-preview-app> Acesso em: 19 out. 2018. Para realizar a instalação em seu computador, siga os seguintes passos:

1. Escolha uma pasta para servir de base para o projeto e acesse-a via linha de comando.
2. Considerando que o git esteja instalado em seu computador:
 - a. Na sua ferramenta de linha de comando, digite:

```
git clone https://github.com/ionic-team/ionic-preview-app
```
3. Considerando que o git não esteja instalado no computador:
 - a. É possível realizar o *download* do projeto no formato .zip através do *link* <https://github.com/ionic-team/ionic-preview-app/archive/master.zip> Acesso em: 19 out. 2018.
4. Acesse a pasta “ionic-preview-app”
5. Digite: *npm install*
 - a. Este comando realiza o *download* de todas as dependências necessárias ao projeto.

- b. É necessário que o Node.js esteja instalado e devidamente configurado

Componentes Visuais - ActionSheet

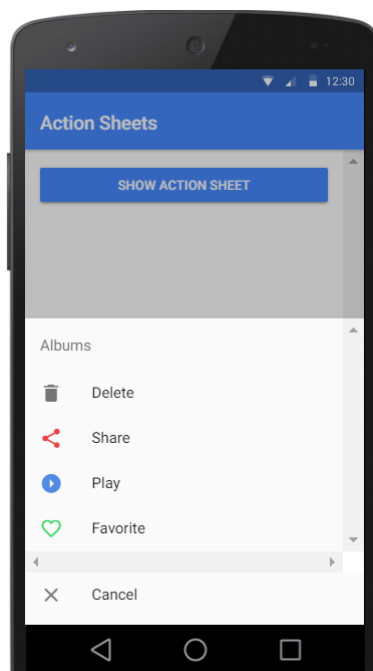
Este componente exibe uma tela *modal* de opções ao usuário, de baixo para cima. É normalmente utilizada quando há poucos itens.

Sua criação é somente via código, ou seja, não existem *tags* do Ionic para serem utilizadas em *templates*.

Há a opção de inclusão de ícones para melhor descrever o texto de cada ação. Entretanto, essa abordagem não é recomendada para a plataforma iOS pois foge das suas *guidelines* de interface. Portanto, utilize com cautela.

Segue um *link* para uma documentação completa sobre este componente: <https://ionicframework.com/docs/api/components/action-sheet/ActionSheetController>
Acesso em: 19 out. 2018.

Figura 28 – ActionSheet.



Fonte: Elaborado pelo próprio professor.

Componentes visuais - Alert

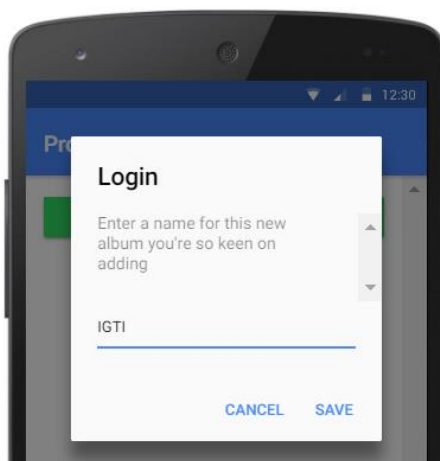
Os Alerts são utilizados para exibição de pequenas telas modais de informações ao usuário e, assim como os ActionSheets, são criados somente via código, sem *templates*.

A seguir são mostrados os principais tipos de alertas:

- **Basic:** apenas informação textual.
- **Confirm:** o usuário deve escolher Sim/Não.
- **Prompt:** o usuário deve informar dados.
- **Radio:** várias opções para o usuário escolher somente uma delas.
- **Checkbox:** várias opções para o usuário escolher uma ou mais.

A documentação da API pode ser acessada através do seguinte *link*: <https://ionicframework.com/docs/api/components/alert/AlertController/> Acesso em: 19 out. 2018.

Figura 29 – Prompt Alert.



Fonte: Elaborado pelo próprio professor.

Componentes visuais – Botões (*Buttons*)

No Ionic, existem vários tipos de renderização de botões, tais como:

- Block (ocupa toda a largura da tela).
- Clear (cores somente no texto).
- Outline (cores no texto e contorno).
- Round (contornos arredondados).

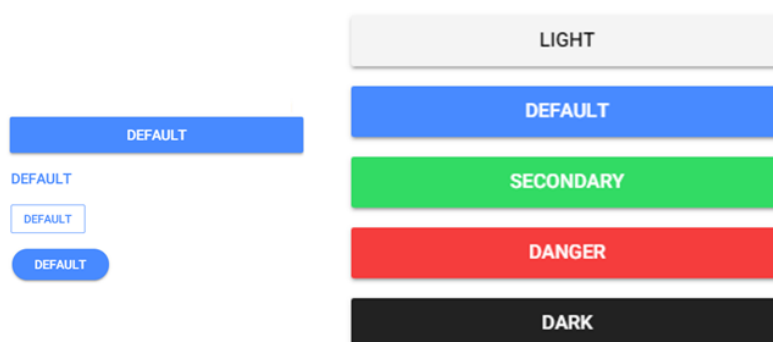
Os botões são renderizados conforme a plataforma e utilizam as cores padrão do Ionic, que podem ser customizadas através da edição do arquivo `.theme\variables.scss`.

Os botões são utilizados nos *templates* da seguinte forma, em geral: `<button ion-button>Texto do botão</button>`

O principal evento utilizado pelo botão é o (click).

A documentação da API pode ser acessada através do seguinte *link*: <https://ionicframework.com/docs/api/components/button/Button/> Acesso em: 19 out. 2018.

Figura 30 – Botões e cores padrão do Ionic.



Fonte: Elaborado pelo próprio professor.

Componentes visuais – Cards

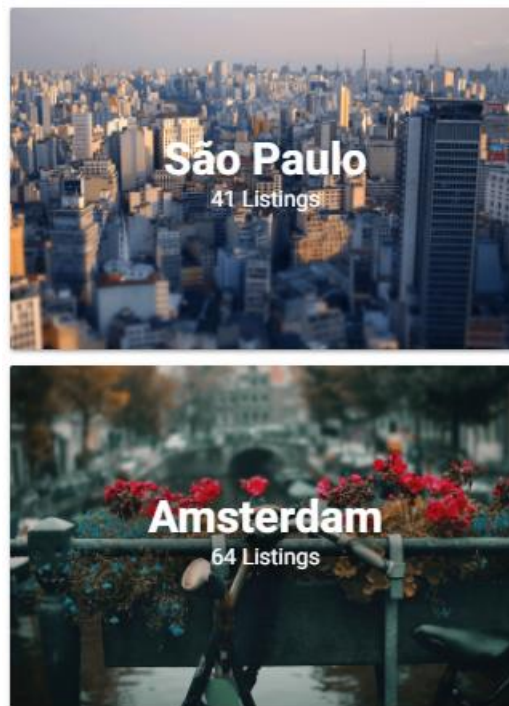
Cards são muito utilizados para dar informações ao usuário de forma elegante. Em geral, os *cards* são apresentados em formato de lista.

Exemplos clássicos de *apps* que utilizam bastante o recurso de *cards* são: Instagram e Facebook.

Os *cards* são componentes mais complexos que permitem, por exemplo, subdivisões. A principal *tag* utilizada é o `<ion-card>`.

Link da documentação: <https://ionicframework.com/docs/components/#cards>
Acesso em: 19 out. 2018.

Figura 31 – Exemplos de *cards*.



Fonte: Elaborado pelo próprio professor.

Componentes visuais – Modals

A utilização de *modals* é uma maneira alternativa de utilização de caixa de diálogo, principalmente quando a informação a ser exibida é mais complexa e necessita de mais interação.

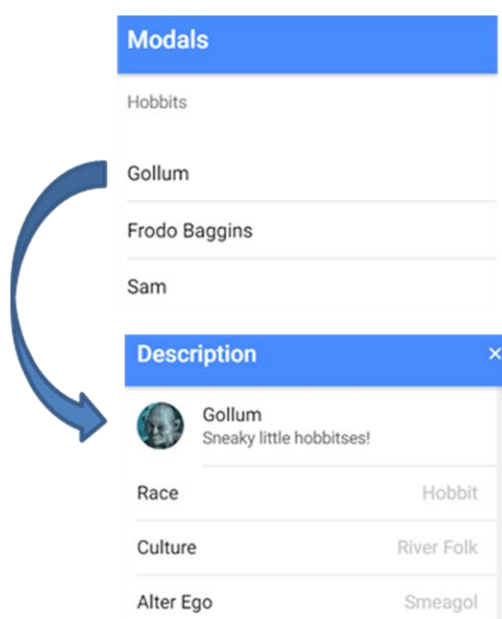
Uma tela modal substitui totalmente a tela atual, ou seja, o usuário perde o acesso à tela anterior. O retorno só é possível caso o modal seja fechado.

Modais também podem ser utilizados para substituir os *detalhes* de telas mestre/detalhe.

No Ionic, grande parte da implementação de modais é feita via código TypeScript.

Link para documentação mais completa (classe ModalController): <https://ionicframework.com/docs/api/components/modal/ModalController/> Acesso em: 19 out. 2018.

Figura 32 – Transição de uma tela *mestre* para uma *modal*.



Fonte: Elaborado pelo próprio professor.

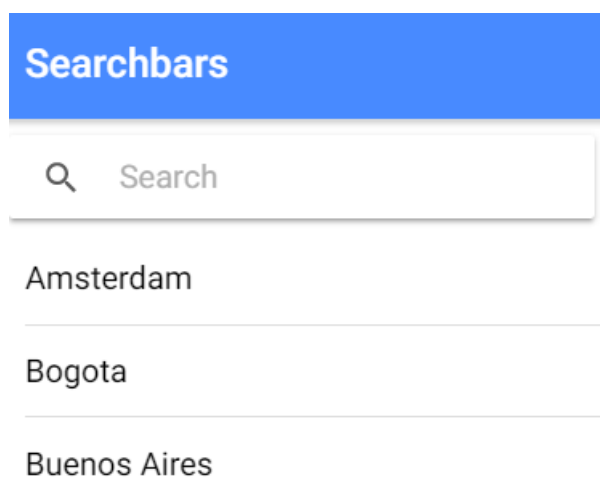
Componentes visuais – SearchBar

SearchBars são compostas de uma coleção de dados e um *input* para a realização de buscas nesses dados. Está disponível na grande maioria dos *apps*, especialmente *apps* corporativos.

A interação nas buscas é, sem sombra de dúvida, um dos grandes gargalos em qualquer app. É preciso muita cautela para que seja feita uma implementação performática. Atrasos (*lags*) nesse tipo de tela prejudicam muito a experiência do usuário.

A documentação sobre este componente pode ser encontrada em: <https://ionicframework.com/docs/api/components/searchbar/Searchbar/> Acesso em: 19 out. 2018.

Figura 33 – Transição de uma tela *mestre* para uma *modal*.



Fonte: Elaborado pelo próprio professor.

Para mais detalhes sobre cada componente, incluindo implementações, acompanhe as vídeo aulas deste capítulo.

Capítulo 5. Criação de um app com Ionic

Neste capítulo, será demonstrado um *app* de tarefas. Portanto, grande parte do conteúdo é prático e poderá ser visto nas vídeo aulas do capítulo. Na apostila, serão abordados tópicos importantes sobre a implementação do *app*.

JavaScript – trabalhando com iterações – função *map*

A função *map* em JavaScript faz uma transformação de uma coleção qualquer, percorrendo-a item a item de forma que seja feita alguma lógica. Ao final, uma nova coleção é gerada (o que garante a imutabilidade). Além da flexibilidade, o desenvolvedor não precisa mais se preocupar com índices de vetores, por exemplo

Figura 34 – Aplicação da função *map*

```
> const vetor1 = [{nome: 'Raphael', idade: 35}, {nome: 'João', idade: 33}, {nome: 'Maria',
idade: 25}];
< undefined
> const vetor2 = vetor1.map(item => item.nome);
< undefined
> vetor2
< ▶ (3) ["Raphael", "João", "Maria"]
```

Fonte: Elaborado pelo próprio professor.

No exemplo acima, a coleção percorrida é criado um vetor (vetor1) de objetos com nomes e idades. Em seguida, é feita uma transformação do vetor, obtendo apenas os nomes. Essa transformação é atribuída a um outro vetor (vetor2)

Para mais detalhes sobre a função *map*, acesse o site da MDN - https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Acesso em: 19 out. 2018.

JavaScript – trabalhando com iterações – função *filter*

A função *filter* em JavaScript elimina itens de uma coleção com base em um critério (que deve retornar *true* ou *false*), gerando uma nova coleção (garantindo imutabilidade). Esta função foi utilizada pelo *app* para excluir tarefas, conforme imagem abaixo:

Figura 35 – Função *filter*.

```
public excluir( tarefa: Tarefa ) {
    const newTarefas = this.tarefas.filter(
        ( tarefaDaVez ) => {
            return ( tarefa.nome !== tarefaDaVez.nome )
        }
    );

    this.tarefas = newTarefas;
    this.persistirStorage();
}
```

Fonte: Elaborado pelo próprio professor.

No exemplo acima, a coleção percorrida é *this.tarefas*. **Para cada item da coleção** (apelidado de *tarefaDaVez*), é feita uma **comparação** entre o nome da *tarefa da vez* com a tarefa a ser comparada. Caso a comparação, que na verdade é uma **diferenciação**, retorne *true*, o valor da coleção é mantido. Assim, a tarefa que tiver o mesmo nome será descartada. É importante frisar que a função *filter* retorna

uma nova coleção (*newTarefas*), que posteriormente é atribuída novamente a *this.tarefas*.

Para mais detalhes sobre a função *filter*, acesse o site da MDN - https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/filtro

Acesso em: 19 out. 2018.

Templates – classes CSS condicionais com a diretiva [ngClass]

A diretiva [ngClass] foi utilizada para determinar se a tarefa estava ou não concluída (cumprida), aplicando a classe *tarefaConcluida* (definida em *tarefas.scss*) em caso afirmativo. A implementação foi feita conforme imagem abaixo:

Figura 36 – Utilização da diretiva [ngClass]

```
<ion-item
  · [ngClass]="{tarefaConcluida: tarefa.estado === 'cumprida'}"
  · (click)='detalheTarefa(tarefa)'
>
  · {{ tarefa.nome }}
</ion-item>
```

Fonte: Elaborado pelo próprio professor.

No exemplo acima, o valor de [ngClass] foi um objeto JavaScript (delimitado por {}), que pode ser lido da seguinte forma: "A classe *tarefaConcluida* será atribuída ao elemento se e somente se a expressão (*tarefa.estado* === '*cumprida*') retornar **true**

Para mais detalhes sobre a diretiva [ngClass], acesse o site da documentação do Angular em <https://angular.io/api/common/NgClass> Acesso em: 19 out. 2018.).

JavaScript – trabalhando com *Promises*

A API JavaScript do IndexedDB utiliza o conceito de *promises* para ler e gravar dados. A utilização de *promises* permite a execução de instruções assíncronas de forma controlada, conforme imagem abaixo:

Figura 37 – Utilização de *promise*.

```
private carregarTarefas() {
    this._storage.get('tarefas')
        .then((tarefas) => {
            if (tarefas && tarefas.length > 0) {
                console.log('Obtendo tarefas de indexeddb');
                this.mapearTarefas(tarefas);
            }
            else {
                console.log('indexeddb vazio. Obtendo tarefas padrão');
                this.mapearTarefas(tarefasPadrao);
            }
        })
}
```

Fonte: Elaborado pelo próprio professor.

No exemplo acima, o método *get()* de *_storage* retorna uma *promise*, que é assíncrona. Caso não a capturemos com *then*, a execução continua e não é possível obter o resultado da requisição, pois não é possível prever *quando* a mesma será finalizada. O comando *then* captura o retorno da *promise* quando ela é **resolvida** (*resolve*). Dentro de *then*, podemos manipular os dados que foram retornados. Caso ocorra algum problema durante a execução da *promise* ela é, em geral, **rejeitada** (*reject*). Para capturarmos as rejeições, podemos utilizar o comando *.catch()*.

Promises são muito utilizadas para obtenção de recursos que geralmente levam certo tempo, como por exemplo requisições *http* e leitura em disco.

Para mais detalhes sobre a API de *Promises*, acesse https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise

Acesso em: 19 out. 2018.

Capítulo 6. Comunicação remota e *plugins* nativos

Grande parte deste capítulo, assim como o capítulo 5, está demonstrado de forma prática nas respectivas vídeo aulas. Serão vistos a seguir alguns tópicos sobre as tecnologias utilizadas durante a implementação.

Comunicação remota com Ionic

Conforme visto na vídeo aula do capítulo 6, o Ionic utiliza o Angular internamente para comunicação remota, como por exemplo em requisições *http* para url's que retornam dados no formato JSON.

Também na vídeo aula foi exibido um pequeno *app* que lista países e algumas informações como bandeira e região; cuja fonte de dados é a url: <https://restcountries.eu/rest/v2/all> Acesso em: 19 out. 2018.

As principais tecnologias envolvidas nesse processo de comunicação remota são, portanto:

- **Objeto *http* do tipo *Http*:** a classe *Http* é fornecida com o Angular através do módulo `@angular/http`. Originalmente, requisições com esse objeto retornam dados do tipo *Observable*. No exemplo da aula, foi feita a conversão de um *Observable* para *Promise* através do método *toPromise*
- **Funções *map* e *toPromise* da biblioteca *rxjs*:** essas funções devem ser explicitamente importadas e são utilizadas para, respectivamente: transformar dados vindos da requisição e converter *Observables* para *Promises*. Elas foram utilizadas no exemplo da vídeo aula.

Figura 38 – Exemplo de comunicação remota utilizando *promise*.

```
obterPaíses() {
  this.http
    .get(this.apiUrl)
    .map(resposta => resposta.json())
    .toPromise()
    .then(dados => this.países = dados);
}
```

Fonte: Elaborado pelo próprio professor.

Adicionando as plataformas Android e iOS

Até o momento, todos os projetos vistos não precisam necessariamente dos dispositivos para testes. Tudo pôde ser feito somente no ambiente web com as ferramentas fornecidas pelo Ionic. Isso é bom pois garante uma alta produtividade no início do projeto. Além disso, essa técnica é bastante utilizada para a criação de protótipos dos *apps* para validação pelos clientes, por exemplo.

Entretanto, sempre chega o momento onde é realmente necessária a utilização dos dispositivos, como por exemplo:

- Testes mais realísticos, com um ou vários dispositivos e/ou emuladores.
- Testes de performance em cada dispositivo.
- Necessidade de utilização de recursos nativos nos dispositivos, tais como:
 - GPS.
 - Vibração.
 - Bateria.
 - Lanterna.

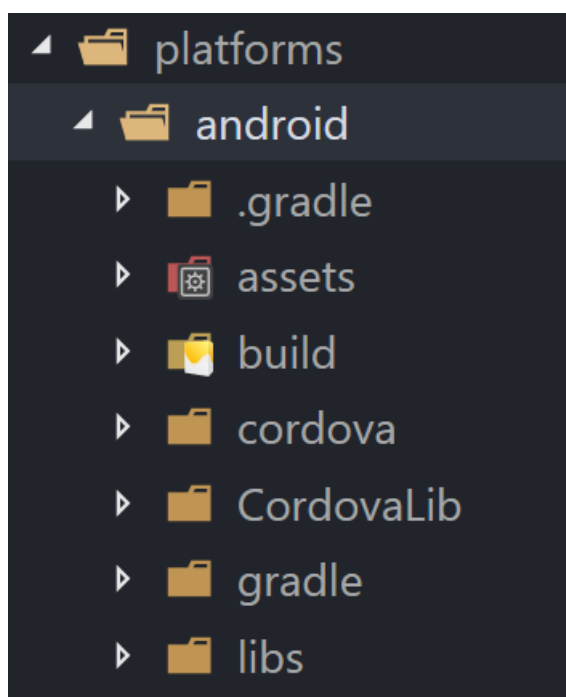
- Câmera.
- Etc.,

Em relação a projetos Ionic, é necessária a inclusão da plataforma à qual se deseja trabalhar. Isso pode ser feito através dos seguintes comandos:

- Android
 - **Inclusão da plataforma:** ionic cordova platform add android.
 - **Execução do *app* no dispositivo (conectado) ou emulador:**
ionic cordova run android.
- iOS
 - **Inclusão da plataforma:** ionic cordova platform add ios
 - **Execução do *app* no dispositivo (conectado) ou emulador:**
ionic cordova run ios
- Em ambos os casos:
 - São criados os arquivos de projeto nativo em `./platforms/“plataforma”`. No caso do Android, é criado um projeto compatível com o Android Studio. No caso do iOS, é criado um projeto compatível com o XCode.
- Dica:
 - O comando “ionic cordova requirements” verifica se determinada plataforma está apta a ser executada, validando o ambiente em seu computador.

- Para mais detalhes sobre a instalação e configuração das plataformas com Ionic e/ou Cordova, acesse:
 - Android:
<https://cordova.apache.org/docs/en/latest/guide/platforms/android/>
Acesso em: 19 out. 2018.
 - iOS:
<https://cordova.apache.org/docs/en/latest/guide/platforms/ios/>
Acesso em: 19 out. 2018.

Figura 39 – Algumas pastas que são criadas com a inclusão da plataforma Android



Fonte: Elaborado pelo próprio professor.

Cordova plugins e Ionic Native

Como foi visto anteriormente, a comunicação nativa em *apps* híbridos feitos com o Cordova é feita através de *plugins*. Esses *plugins* são construídos com as linguagens de programação nativas de cada plataforma. Já a utilização é feita com JavaScript.

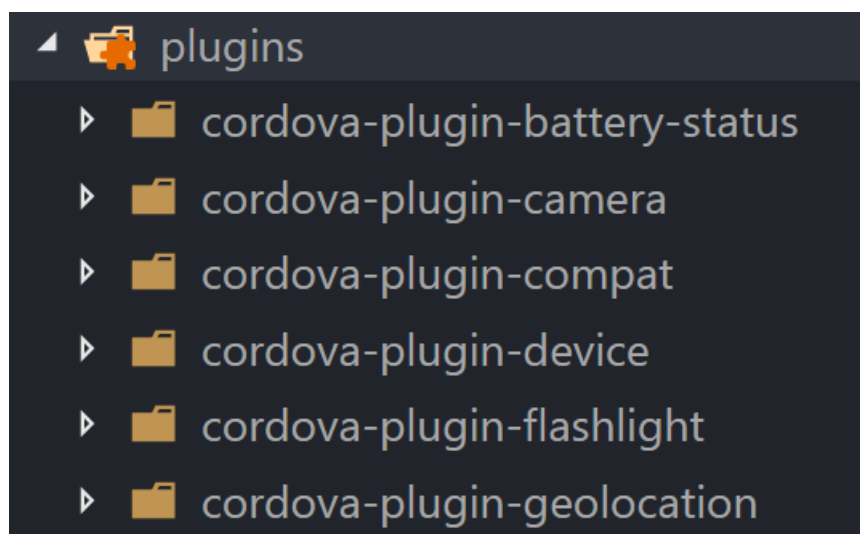
O Cordova possui atualmente quase 4.000 *plugins*. Para mais detalhes, acesse: <https://cordova.apache.org/plugins/> Acesso em: 19 out. 2018.

Já o Ionic Native funciona como uma camada de abstração aos *plugins* do Cordova permitindo, por exemplo, a utilização de TypeScript ao invés de JavaScript para a utilização dos *plugins*. Para mais informações sobre o Ionic Native e sua documentação, acesse o endereço <https://ionicframework.com/docs/native/> Acesso em: 19 out. 2018.

Para a instalação dos *plugins*, é necessária a execução de alguns comandos no terminal. A imagem abaixo ilustra a instalação do *plugin* de vibração. Para descobrir os nomes oficiais do *plugin* desejado, basta pesquisar na documentação oficial. Uma vez que o *plugin* é instalado, é criada a pasta "plugins" no projeto.

```
$ ionic cordova plugin add cordova-plugin-vibration  
$ npm install --save @ionic-native/vibration
```

Figura 40 – Exemplo de instalação do *plugin* de vibração e das pastas criadas no projeto.



Fonte: Elaborado pelo próprio professor.

Alguns exemplos de *plugins*

A seguir, será exibida uma documentação básica sobre cada *plugin*. Mais detalhes podem ser vistos nas respectivas vídeo aulas.

Vibração

O *plugin* de vibração pode ser utilizado para efetuar a vibração do dispositivo. Na plataforma Android, é possível especificar o tempo de vibração em milisegundos.

Documentação do *plugin*: <https://ionicframework.com/docs/native/vibration/>
Acesso em: 19 out. 2018.

Bateria

O *plugin* de bateria pode ser utilizado para monitorar o estado da bateria no dispositivo.

Documentação do *plugin*: <https://ionicframework.com/docs/native/battery-status/> Acesso em: 19 out. 2018.

No *app* de exemplo, foi demonstrada a utilização com *Observables*, conforme imagem abaixo:

Figura 40 – Exemplo de utilização de *Observables*.

```
public verificarBateria() {
    console.log('Verificando bateria...');
    this.platform.ready().then(() => {
        console.log('platform ready');
        this.subscriptionChange =
            this.bateria
                .onChange()
                .subscribe((batteryData) => {
                    console.log('dados atualizados!');
                    this.zone.run(() => { //Sincroniza as ações
                        this.statusBateria = batteryData;
                        this.validarBateria();
                    });
                });
    });
}
```

Fonte: Elaborado pelo próprio professor.

De forma sucinta, ao utilizar *Observables*, é feita uma *inscrição* em um serviço qualquer (*subscribe*). Uma vez que esse serviço é atualizado, todos os objetos que se inscreveram são notificados e podem executar uma ação com os dados que *chegaram*. Neste exemplo relativo à bateria do dispositivo móvel, sempre que houve uma alteração (no percentual da carga, por exemplo) o objeto que se *inscreveu* é notificado e são feitas atualizações dos dados da bateria na tela (*template*).

Para mais detalhes técnicos sobre *Observables*, acesse o seguinte link: <https://tableless.com.br/entendendo-rxjs-observable-com-angular/> Acesso em: 19 out. 2018.

Lanterna

O *plugin* de lanterna (*flashlight*) pode ser utilizado para, por exemplo:

- Ligar/desligar a lanterna
- Verificar se a lanterna está ligada

Documentação do *plugin*: <https://ionicframework.com/docs/native/flashlight/>
Acesso em: 19 out. 2018.

Geolocalização

O *plugin* de geolocalização pode ser utilizado para, por exemplo, a obtenção de coordenadas (latitude/longitude).

No exemplo do *app* foi feita de forma complementar a geolocalização reversa através da utilização de um serviço com comunicação remota.

Documentação do *plugin*:
<https://ionicframework.com/docs/native/geolocation/> Acesso em: 19 out. 2018.

Referências

GRIFFITH, C. Mobile App Development with Ionic: cross-platform apps with Ionic, Angular and Cordova. United States: O'Reilly, 2017. 506p.

CORDOVA. Cordova Documentation. Disponível em: <https://cordova.apache.org/docs/en/latest/>. Acesso em: 19 out. 2018.

ANGULAR. Angular Docs. Disponível em: <https://angular.io/docs>. Acesso em: 27 de Set. 2018.

TYPESCRIPT. TypeScript Documentation. Disponível em: <https://www.typescriptlang.org/docs/home.html>. Acesso em: 19 out. 2018.

IONIC. Ionic Documentation. Disponível em: <https://ionicframework.com/docs/>. Acesso em: 19 out. 2018.