Claretiano
CENTRO UNIVERSITÁRIO

DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS COM IONIC

*Fazemos parte do **Claretiano - Rede de Educação***

Claretiano – Centro Universitário
Rua Dom Bosco, 466 - Bairro: Castelo – Batatais SP – CEP 14.300-000
cead@claretiano.edu.br
Fone: (16) 3660-1777 – Fax: (16) 3660-1780 – 0800 941 0006
claretiano.edu.br/batatais

Meu nome é **Marcos Henrique de Paula**. Sou mestre na área de Engenharia de *Software* pela Universidade Federal de São Carlos, departamento de Ciência da Computação – UFSCar. Minha formação é bacharel em Sistemas de Informação, com especialização em Projeto e Desenvolvimento de *Software* com Java e Banco de Dados pelo Claretiano – Centro Universitário. Leciono diversas disciplinas relacionadas com o desenvolvimento de *software* no curso de Sistemas de Informação no Claretiano – Batatais–SP e atuo com análise e desenvolvimento de *software* desde 1997 na região de Batatais e Ribeirão Preto-SP.

Marcos Henrique de Paula

DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS COM IONIC

Batatais
Claretiano
2018

CORPO TÉCNICO EDITORIAL DO MATERIAL DIDÁTICO MEDIACIONAL

Coordenador de Material Didático Mediacional: J. Alves

Preparação: Aline de Fátima Guedes • Camila Maria Nardi Matos • Carolina de Andrade Baviera • Cátia Aparecida Ribeiro • Dandara Louise Vieira Matavelli • Elaine Aparecida de Lima Moraes • Josiane Marchiori Martins • Lidiane Maria Magalini • Luciana A. Mani Adami • Luciana dos Santos Sançana de Melo • Patrícia Alves Veronez Montera • Raquel Baptista Meneses Frata • Simone Rodrigues de Oliveira

Revisão: Eduardo Henrique Marinheiro • Filipi Andrade de Deus Silveira • Rafael Antonio Morotti • Rodrigo Ferreira Daverni • Vanessa Vergani Machado

Projeto gráfico, diagramação e capa: Bruno do Carmo Bulgarelli • Joice Cristina Micai • Lúcia Maria de Sousa Ferrão • Luis Antônio Guimarães Toloi • Raphael Fantacini de Oliveira • Tamires Botta Murakami

Videoaula: André Luís Menari Pereira • Bruna Giovanaz • Marilene Baviera • Renan de Omote Cardoso

Bibliotecária: Ana Carolina Guimarães – CRB7: 64/11

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

006.786 D974s

Paula, Marcos Henrique de
Desenvolvimento para dispositivos móveis com IONIC / Marcos Henrique de Paula – Batatais, SP : Claretiano, 2018.
142 p.

ISBN: 978-85-8377-570-6

1. Ionic. 2. Smartphones. 3. Dispositivos móveis. 4. Cordova. 5. Framework. I. Desenvolvimento para dispositivos móveis com IONIC.

CDD 006.786

INFORMAÇÕES GERAIS

Cursos: Graduação
Título: Desenvolvimento para Dispositivos Móveis com Ionic
Versão: dez./2018
Formato: 20x28 cm
Páginas: 142 páginas

SUMÁRIO

CONTEÚDO BÁSICO DE REFERÊNCIA/ CONTEÚDO DIGITAL INTEGRADOR

Conteúdo

Introdução ao Desenvolvimento para Dispositivos Móveis. Histórico e Aspectos evolutivos. Tecnologias para Desenvolvimento Nativo, Híbrido e Cross-Plataforma. Fundamentos do Framework Ionic. Apache Cordova. Instalação e configuração. Criação de Projetos. Estrutura de um projeto Ionic. Executando Aplicações Ionic. Emulação com o Navegador. Emulação no Dispositivo. Componentes gráficos do framework Ionic. Armazenamento persistente de dados com Ionic. JavaScript Object Notation (JSON). SQLite. Conectividade com Ionic. API REST.

Bibliografia Básica

ARVIND, R. *Learning Ionic*. 2. ed. Birmingham: Packt Publishing, 2017.

GOIS, A. *Ionic framework*. São Paulo: Casa do Código, 2017.

PHAN, C. Ionic 2 cookbook. 2. ed. Birmingham: Packt Publishing, 2016.

Bibliografia Complementar

FRISBIE, M. *Angular 2 cookbook*. Birmingham: Packt Publishing, 2016.

GUEDES, T. *Crie aplicações com Angular*. São Paulo: Casa do Código, 2017.

LEE, V.; SCHNEIDER, H.; SCHELL, R. *Aplicações móveis*: arquitetura, projeto e desenvolvimento. Tradução de: Amaury Bentes e Deborah Rudger, São Paulo: Ed. Pearson Education do Brasil, 2005.

LOPES, C. *AngularJS para desenvolvedores Java*. Rio de Janeiro: Ciência Moderna, 2016.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Addison-Wesley, 2011.

E-referências

IONIC Framework, Components. Site do desenvolvedor, componentes. Disponível em: <https://ionicframework.com/docs/components/>. Acesso em: 03 ago. 2018.

IONIC Framework, API Reference. Site do desenvolvedor, documentação referencial da API. Disponível em: <https://ionicframework.com/docs/api/>. Acesso em: 03 ago. 2018.

É importante saber

Esta obra está dividida, para fins didáticos, em duas partes:

Conteúdo Básico de Referência (CBR): é o referencial teórico e prático que deverá ser assimilado para aquisição das competências, habilidades e atitudes necessárias à prática profissional. Portanto, no CBR, estão condensados os principais conceitos, os princípios, os postulados, as teses, as regras, os procedimentos e o fundamento ontológico (o que é?) e etiológico (qual sua origem?) referentes a um campo de saber.

Conteúdo Digital Integrador (CDI): são conteúdos preexistentes, previamente selecionados nas Bibliotecas Virtuais Universitárias conveniadas ou disponibilizados em *sites* acadêmicos confiáveis. É chamado "Conteúdo Digital Integrador" porque é **imprescindível** para o aprofundamento do Conteúdo Básico de Referência. Juntos, não apenas privilegiam a convergência de mídias (vídeos complementares) e a leitura de "navegação" (hipertexto), como também garantem a abrangência, a densidade e a profundidade dos temas estudados. Portanto, são conteúdos de estudo obrigatórios, para efeito de avaliação.

1. INTRODUÇÃO

No cotidiano da vida moderna existem muitos elementos importantes, mas sem dúvida os *Smartphones* desempenham um papel fundamental em nossas vidas. A conectividade e a mobilidade fornecidas por esses dispositivos têm alterado a forma como realizamos muitas de nossas atividades habituais. Mensagens, chamadas telefônicas, redes sociais, compras, entretenimento, atividades profissionais, quase tudo em nosso dia a dia pode ser realizado em um estilo de vida em movimento.

Estamos vivendo a era da mobilidade e a cada dia temos uma nova variedade de serviços sendo oferecidos por meio de novos aplicativos nos *Smartphones*. Esses mesmos aplicativos também rodam em *Tablets* e outros tipos de dispositivos móveis. As pessoas têm seus "apps" preferidos instalados e prontos para uso. Dessa forma, a demanda por novos tipos de aplicativos continua em alta.

Aparentemente, nos ambientes de negócios e de trabalho, as pessoas usam em média pelo menos dois dispositivos diferentes como parte de suas rotinas diárias, e esse número tende a aumentar. Além disso, as pessoas têm a liberdade de escolher os aplicativos e até mesmo os processos necessários para completar suas atividades.

Gerando, assim, um aumento da pressão sobre os desenvolvedores para criar novas funcionalidades, em curto espaço de tempo, para os aplicativos móveis. A maioria das empresas acha um grande desafio produzir, implantar e distribuir aplicativos móveis no ritmo acelerado que o mercado exige. Encontrar e empregar desenvolvedores com todas as habilidades necessárias nessa área é uma tarefa cada vez mais difícil.

De acordo com o Gartner Group (2017), até o final de 2017, a demanda do mercado para serviços corporativos de desenvolvimento de aplicativos móveis superaria a capacidade das empresas de desenvolvê-los. E até 2019, a venda de telefones celulares atingiria 2,1 bilhões de unidades. Isso significa uma demanda ainda maior para desenvolver aplicações móveis.

Ainda de acordo com o Gartner Group (2017) existem melhores práticas que as empresas de *software* podem considerar para superar com sucesso os desafios e a crescente demanda do desenvolvimento de aplicativos para dispositivos móveis. Entre essas boas práticas está o uso de *frameworks* e APIs, ferramentas de automação e geração de código, cujo desenvolvimento é baseado em modelos e metodologias ágeis para a produção de *software*.

Esse é o cenário que se apresenta diante das empresas e das comunidades de desenvolvedores de aplicativos para dispositivos móveis; o mercado é amplo e as oportunidades se multiplicam na mesma proporção em que os negócios migram para os aplicativos móveis.

Refletindo sobre o exposto, apresentamos neste material uma plataforma de desenvolvimento de aplicações para dispositivos móveis que, diante desse cenário, seja uma boa opção de escolha, tanto para desenvolvedores experientes como para iniciantes, para fazer frente à crescente demanda e ser capaz de atender às expectativas de empresas e desenvolvedores como uma boa solução tecnológica, eficaz e que proporciona os recursos necessários à produção rápida de novos aplicativos.

Apresentamos o desenvolvimento de aplicações móveis com o *framework* Ionic. Na realidade, como veremos mais adiante na primeira unidade, a plataforma Ionic é composta de vários *frameworks*, como o Cordova, o Angular, o próprio Ionic, além de APIs, Servidores, entre outros.

Para construir gradualmente o conhecimento necessário ao aprendizado da plataforma Ionic, organizamos o conteúdo em quatro unidades.

Na Unidade 1, apresentamos uma introdução sobre a infraestrutura e as características particulares nos dispositivos móveis que vão impactar nas decisões que devem ser tomadas logo no início dos projetos de desenvolvimento de aplicativos. Desse modo, delinear sobre as características dos dispositivos móveis considerando uma perspectiva de desenvolvimento de aplicações pode ajudar o desenvolvedor a perceber e adquirir habilidades básicas e a se tornar mais eficiente, pois permitirá tomar decisões mais adequadas logo no início do projeto. Ainda nesta unidade abordamos conceitos de aplicativos nativos e híbridos, suas vantagens e desvantagens, e, por fim, mostramos como preparar e utilizar o seu ambiente de desenvolvimento com o *framework* Ionic.

Na segunda unidade, mostramos como criar e configurar projetos Ionic; fornecemos uma visão completa de como utilizar os componentes do *framework* Ionic; exemplificamos na prática a codificação e a utilização dos principais componentes que introduzirão o desenvolvedor no conhecimento dos conceitos e práticas que envolvem a plataforma; mostramos como executar e testar o aplicativo; e, por fim, exemplificamos como realizar a transição entre os componentes visuais.

Na Unidade 3, abordamos conceitos de armazenamento e gerenciamento de dados nos dispositivos móveis; evidenciamos as práticas de armazenamento interno e externo, suas vantagens e desvantagens; mostramos como criar e configurar projetos com a utilização de *plugins* para armazenamento de dados; criamos um aplicativo para armazenar dados como exemplo; e indicamos as instruções de como preparar o projeto para dar suporte a aplicações nativas, incluindo as instruções de como gerar e construir o aplicativo para ser instalado na plataforma nativa Android.

Na Unidade 4, apresentamos conceitos de armazenamento de dados na nuvem com Web Services; introduzimos conceitos da tecnologia REST para utilização de serviços de dados; e criamos um aplicativo Ionic que se comunica com um Web Service na nuvem para realizar trocas de dados.

No decorrer das unidades de estudo indicamos referências que complementam o conteúdo apresentado com o intuito de apoiar a sua jornada rumo ao aprendizado no desenvolvimento de aplicações para dispositivos móveis. Esperamos que este material seja de grande utilidade para introduzi-lo nos conhecimentos que permeiam essa área e que sirva como fonte inspiradora para que você possa continuar buscando novos conhecimentos.

2. GLOSSÁRIO DE CONCEITOS

O Glossário de Conceitos permite uma consulta rápida e precisa das definições conceituais, possibilitando um bom domínio dos termos técnico-científicos utilizados na área de conhecimento dos temas tratados.

1) **AngularJS**: *framework* para apoio à criação e execução de aplicações web dinâmicas, foi escrito em Javascript e é mantido pelo Google. Permite a criação modular e de componentes reutilizáveis.
2) **API**: (*Application Programming Interface*), Interface de Programação e Aplicação. Biblioteca ou conjunto de funções estabelecidas para a utilização por aplicativos que apenas precisam usar seus serviços.
3) **APK**: *android Package* é uma forma de compactar sua aplicação em um arquivo de instalação para o sistema operacional Android, ou seja, ele é um arquivo compilado, usado para se instalar programas no Android.
4) **APP**: a sigla "app" é uma abreviatura para o termo "aplicação de *software*". Em 2010, tornou popular para designar o termo para "Aplicativos Móveis" Originalmente, as aplicações móveis foram criadas e classificadas como ferramentas de suporte à produtividade e à recuperação de informação como correio eletrônico, calendário, contatos etc.
5) **Bootstrap**: é um *framework opensource* HTML, CSS e JavaScript para desenvolvimento de *sites* web. Possui uma diversidade de componentes que auxiliam o designer a construir rapidamente telas de interface com o usuário. As aplicações desenvolvidas com o *Bootstrap* agregam características de se autoajustar o conteúdo a diversos tamanhos de telas dos dispositivos.
6) **Build**: na plataforma Ionic, o termo "build" se refere à atividade de gerar ou construir o aplicativo executável para a plataforma nativa. O build também ocorre quando o desenvolvedor salva as alterações de código no projeto, pois a aplicação nesse momento é construída novamente para execução e testes no navegador.
7) **CLI**: o termo significa *Command Line Interface* (CLI) e é uma ferramenta para o desenvolvimento de aplicações. O CLI possui uma série de comandos que são digitados no terminal ou *prompt* e tem efeitos de geração de código para criar novos projetos, templates ou componentes para a aplicação.
8) **Cordova**: é um *framework opensource* de desenvolvimento de aplicativos móveis com APIs que permitem que o desenvolvedor acesse funções nativas do dispositivo e a infraestrutura de sensores. O desenvolvimento de aplicações com o apoio do Apache Cordova permite a criação de aplicativo híbrido, em que o código pode ser compilado para diversas plataformas nativas como Android, iOS, Windows Phone.
9) **DAO**: objeto de acesso a dados, acrônimo de *Data Access Object*, é um padrão para persistência de dados que permite separar as regras de acesso a banco de dados de outras operações na aplicação. Em uma aplicação que utilize o padrão arquitetural MVC, todas as funcionalidades de bancos de dados devem ser feitas por classes DAO.
10) **Deploy**: esse termo geralmente se refere a atividade de instalação ou atualização da aplicação no servidor. Na plataforma Ionic é utilizado o termo "build" para se referir a essa mesma atividade. As atualizações podem incluir novas funcionalidades ou até correções de erros.
11) **Framework**: é uma estrutura de *software* reutilizável que fornece funcionalidades específicas para facilitar o desenvolvimento de aplicativos de *software*, produtos e soluções. Os *Frameworks* podem incluir programas de suporte, compiladores,

bibliotecas de códigos, conjuntos de ferramentas e interfaces de programação de aplicativos (APIs) que reúnem todos os diferentes componentes para permitir o desenvolvimento de um projeto ou sistema.

12) **IDE**: *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado), é um aplicativo que fornece apoio para codificar e criar novas aplicações de *software*. Um IDE normalmente consiste em um editor de código-fonte, ferramentas de automação e de compilação, depurador de código e várias ferramentas para simplificar a construção de uma interface gráfica de usuário (GUI).
13) **JSON**: (*JavaScript Object Notation*) é um formato de dados de padrão aberto criado para transmitir objetos de dados na internet.
14) **NodeJS**: é um servidor para implementar aplicações server-side de sistemas distribuídos, rápidas e escaláveis.
15) **NPM**: é um utilitário de linha de comando utilizado para gerenciar a instalação de pacotes de plataformas de desenvolvimento baseadas no NodeJS. Para gerenciar os pacotes, o NPM interage com um repositório online, que ajuda na instalação de pacotes, gerenciamento de versão e gerenciamento de dependências.
16) **MVC**: padrão arquitetural de *software* que determina restrições. O componente da camada de modelagem (Model) gerencia o sistema de objetos que representam os dados. O componente da camada de visão (View) define e gerencia como os dados são apresentados. O componente da camada de controle (Control) gerencia, por meio de eventos, a interação do usuário com as telas do sistema e os componentes de Visão e o Modelo.
17) **REST**: (*Representational State Transfer*), transferência de estado representacional. É um estilo arquitetural de construção de *software* para implementação de serviços web. O REST se tornou um padrão para *softwares* que fornecem interoperabilidade entre sistemas na internet. O estilo descreve qualquer interface web simples que utiliza XML, JSON, ou texto, e utiliza o protocolo HTTP para comunicação de dados.
18) **SASS**: (*Syntactically Awesome Style Sheets*) (SASS, 2017) é uma extensão da linguagem de regras de estilo CSS que permite o uso de novos recursos como, por exemplo, a criação de variáveis, regras aninhadas, importações em linha e outras soluções que tem o objetivo de acelerar o processo de desenvolvimento de código para regras de estilo.
19) **SDK**: é um kit de desenvolvimento de *software*. Geralmente é um conjunto de ferramentas de desenvolvimento de *software* que permite a criação de aplicativos para uma determinada plataforma de *software*, sistema de computador, console de videogames, sistema operacional ou plataforma de *hardware*.
20) **SOA**: *Service-Oriented Architecture* (Arquitetura Orientada a Serviços) é um estilo de arquitetura de *software* que determina que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços por meio de um protocolo de comunicação em uma rede.
21) **SQL**: *Structured Query Language* (Linguagem de Consulta Estruturada) é uma linguagem de domínio específico para bancos de dados relacionais. Utilizada para programação e projetada para gerenciar dados mantidos em um sistema de gerenciamento de banco de dados relacional.

3. ESQUEMA DOS CONCEITOS-CHAVE

O Esquema, a seguir, possibilita uma visão geral dos conceitos mais importantes deste estudo.

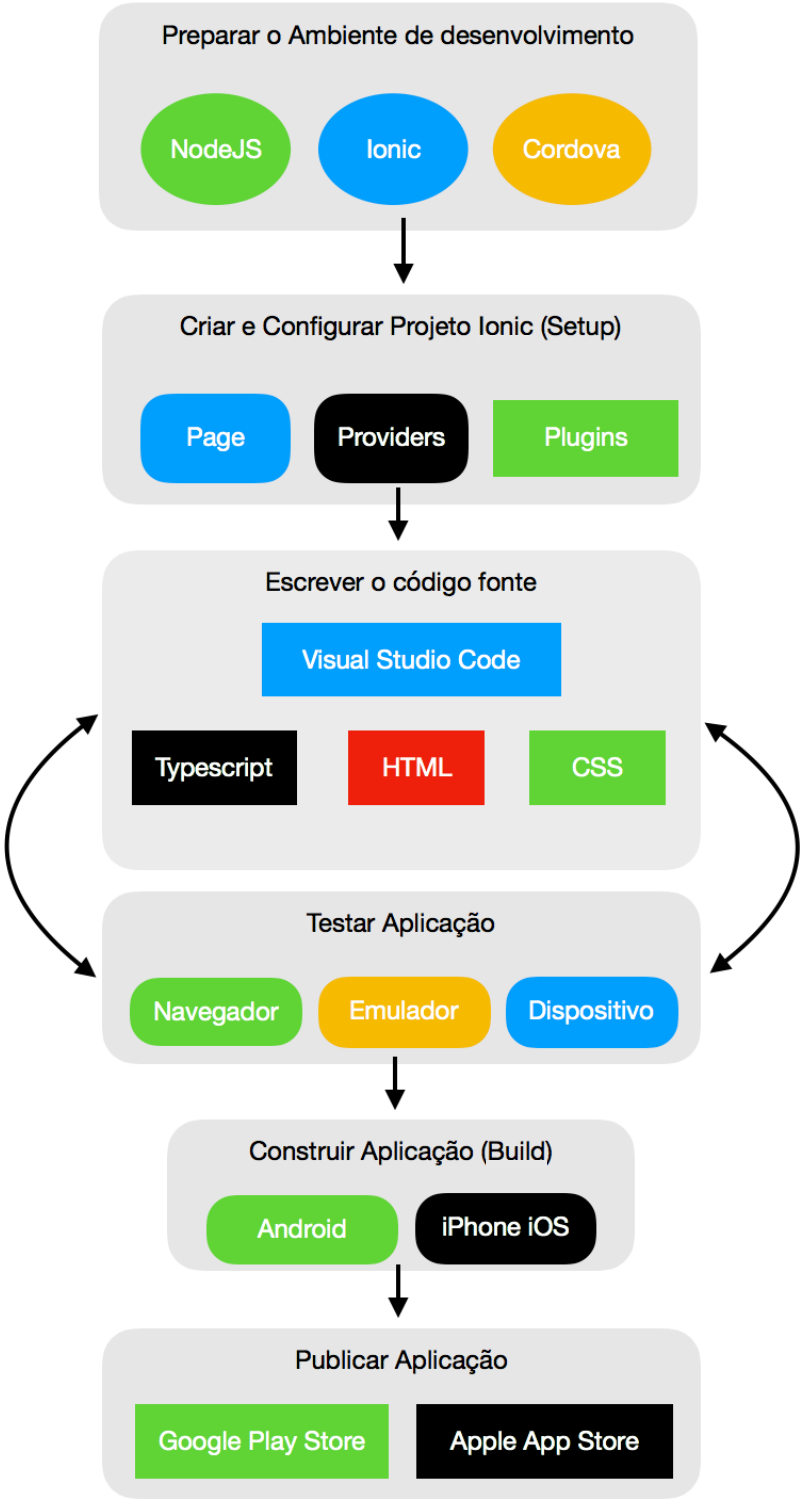


Figura 1 *Esquema dos Conceitos-chave de Desenvolvimento para Dispositivos Móveis com Ionic*.

4. E-REFERÊNCIAS

BOOSTRAP. Site do desenvolvedor, documentação técnica. Disponível em: <https://getbootstrap.com/>. Acesso em: 03 ago. 2018.

IONIC CLI. Site do desenvolvedor, documentação técnica. Disponível em: <https://ionicframework.com/docs/cli/>. Acesso em: 03 ago. 2018.

CORDOVA. Site do desenvolvedor, documentação técnica. Disponível em: <https://cordova.apache.org/>. Acesso em: 03 ago. 2018.

GARTNER GROUP. *Technology Research*. Disponível em: <https://www.gartner.com/newsroom/id/3076817>. Acesso em: 03 ago. 2018.

GARTNER GROUP. *Technology Research*. Disponível em: <https://www.gartner.com/newsroom/id/3725117>. Acesso em: 03 ago. 2018.

DAO Data Access Object. In: *Wikipédia*: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/DAO>. Acesso em: 03 ago. 2018.

Frameworks. In: *Wikipédia*: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Framework>. Acesso em: 03 ago. 2018.

SDK. In: *Wikipédia*: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Kit_de_desenvolvimento_de_software>. Acesso em: 03 ago. 2018.

UNIDADE 1

CARACTERÍSTICAS DOS DISPOSITIVOS MÓVEIS E O DESENVOLVIMENTO DE APLICAÇÕES

Objetivos

- Conhecer os conceitos acerca das características particulares dos dispositivos móveis.
- Compreender a importância do desenvolvimento para dispositivos móveis que utilizam as plataformas nativas e híbridas.
- Preparar o ambiente de desenvolvimento.
- Desenvolver a primeira aplicação utilizando componentes básicos para introduzir a prática de se utilizar a plataforma Ionic.
- Conhecer a plataforma de desenvolvimento Ionic, outros *frameworks* e aplicativos que completam a plataforma.
- Evidenciar aspectos importantes da arquitetura de *software* para dispositivos móveis.

Conteúdos

- Características de *hardware* dos dispositivos móveis.
- Sensores e infraestrutura dos dispositivos móveis.
- Plataforma de desenvolvimento Ionic e suas dependências.
- Instalação da Plataforma Ionic.
- Utilização de componentes do *framework* Ionic.

Orientações para o estudo da unidade

Antes de iniciar o estudo desta unidade, é importante que você leia as orientações a seguir:

1) Não se limite ao conteúdo desta obra, busque outras informações em *sites* confiáveis e/ou nas referências bibliográficas apresentadas ao final de cada unidade. Lembre-se de que, na modalidade EaD, o engajamento pessoal é um fator determinante para o seu crescimento intelectual.

2) O desenvolvimento de *software* para dispositivos móveis deve levar em consideração as questões de usabilidade apropriadas para esses dispositivos, assim como o bom aproveitamento dos recursos de *hardware*; é importante que o desenvolvedor tenha um bom conhecimento sobre as características particulares desses dispositivos.

3) Conhecer as funcionalidades dos componentes de *software* da plataforma de desenvolvimento e a forma como estão estruturados e organizados na arquitetura de *software* ajudará o desenvolvedor a se localizar melhor em seu projeto e também a ter uma visão mais completa para alcançar os seus objetivos.

4) É importante revisar os conhecimentos básicos de desenvolvimento para padrões web, como HTML, Javascript e CSS, que serão claramente reaproveitados no estudo da plataforma de desenvolvimento de aplicativos híbridos.

5) Nas instruções para criar e configurar novos projetos, siga exatamente as orientações, em especial ao que se refere ao nome do projeto e ao nome de componentes, pois esses nomes são utilizados no código-fonte da aplicação.

1. INTRODUÇÃO

O objetivo inicial do estudo desta primeira unidade é ajudar você a compreender características que são particulares nos dispositivos móveis e que os diferenciam dos computadores pessoais, melhorando assim, a perspectiva inicial de abordagem para o desenvolvimento de *software* e, consequentemente, auxiliando o desenvolvedor a obter um aplicativo mais ajustado às características dos dispositivos móveis.

Os objetivos a serem alcançados, nesta unidade, fundamentam-se na compreensão dos conceitos envolvidos na plataforma de desenvolvimento Ionic, nas atividades de instalação e utilização da plataforma. Portanto, é fundamental que você prepare desde já o seu ambiente de desenvolvimento Ionic e comece a utilizá-lo para compreender os conceitos ao longo dos estudos apresentados. Mais adiante, serão explicados todos os procedimentos para a instalação da plataforma Ionic.

Para o bom desenvolvimento deste conteúdo, é fundamentol que você saiba que os dispositivos móveis surgiram por meio da integração dos computadores com os recursos de telecomunicações. Essa união de tecnologias criou um novo panorama digital de computação e mobilidade que possibilita acesso instantâneo a informações em qualquer lugar e a qualquer momento. Assim, a computação móvel, como é chamada hoje, é considerada um novo paradigma computacional.

Desse modo, com a diminuição dos custos dos computadores e os serviços de telecomunicações, como os serviços via satélite, as redes sem fio e a internet, a computação móvel se tornou viável não somente para o governo e o segmento empresarial, mas também para as pessoas de forma geral. De acordo com a enciclopédia digital Wikipédia (2017), a computação móvel pode ser considerada uma revolução, pois é capaz de atingir de forma significativa o cotidiano das pessoas fazendo parte de suas vidas, modificando, assim, suas rotinas e formas de tomar decisões.

No contexto da computação móvel, um dispositivo móvel é um aparelho de computação pequeno o suficiente para segurar e operar na mão, e os elementos desse sistema têm as propriedades de comunicação e mobilidade, pois podem ser operados a distância e sem fio. Mobilidade se refere ao uso de dispositivos móveis portáteis que ofereçam capacidade de deslocamento e podem realizar facilmente um conjunto de funções de aplicação capazes de conectar-se com outros sistemas para obter e fornecer dados (LEE et al. 2005). Um crescimento extraordinário tem ocorrido nesse segmento na última década, e o impacto de sua utilização está redefinindo a forma como é realizada a difusão e o comércio de produtos e serviços, principalmente na área de entretenimento.

Apesar de tudo isso parecer muito óbvio, principalmente para aqueles que já estão acostumados aos Smartphones e Tablets, é importante destacar que, ao desenvolver um *software* para esses dispositivos, o desenvolvedor deve considerar uma abordagem um pouco diferente do *software* tradicional para computadores pessoais. Isso porque os dispositivos móveis possuem características tecnológicas intrínsecas à usabilidade e à mobilidade, como por exemplo os sensores, a tela sensível ao toque e, em muitos casos, a ausência de periféricos como teclado e *mouse*. Portanto, é muito importante que as aplicações sejam planejadas e implementadas desde o início tendo em vista essas diferentes características.

2. CARACTERÍSTICAS DOS DISPOSITIVOS MÓVEIS

Delinear as características dos dispositivos móveis a partir da perspectiva de desenvolvimento de aplicações ajuda o desenvolvedor a perceber e adquirir habilidades básicas do ponto de vista de aproveitamento dos recursos de *hardware*. A compreensão da **infraestrutura móvel** ajuda o desenvolvedor a se tornar mais eficiente, pois lhe permitirá tomar decisões mais adequadas sobre como sua aplicação terá um melhor funcionamento nos dispositivos móveis (LEE et al. 2005).

De acordo com LEE et al. (2005) as principais características de mobilidade nos dispositivos móveis são:

- Portabilidade
- Usabilidade
- Funcionalidade
- Conectividade

Observe, no Quadro 1, a descrição das principais características de mobilidade apresentadas pelos dispositivos móveis.

Quadro 1 – Descrição das principais características de mobilidade dos dispositivos móveis.

CARACTERÍSTICAS DE MOBILIDADE	DESCRIÇÃO
Portabilidade	Capacidade de ser facilmente transportável. Os dispositivos são cada vez menores, mais rápidos e poderosos em suas funcionalidades.
Usabilidade	Utilizados por diferentes tipos de pessoas. A usabilidade de um dispositivo móvel depende de fatores como o ambiente e as características pessoais do usuário.
Funcionalidade	É implementada na forma de uma aplicação móvel e, em geral, estão disponíveis múltiplas aplicações que podem operar em modo independente, isto é, sem qualquer contato com outro usuário ou sistema como, por exemplo: calendário, relógio, jogos, calculadora ou aquelas que operam em modo dependente que precisam conectar-se a outro usuário ou sistema, como exemplo podemos citar: correio eletrônico, agenda, contatos, tarefas, notícias, GPS etc.
Conectividade	Capacidade de utilizar as tecnologias de telecomunicação como redes, internet e serviços de satélite para se conectar com outros sistemas com o objetivo de transmitir e receber dados. Dessa forma, os dispositivos móveis podem fazer uso de sistemas online.

Infraestrutura Móvel

Esse tópico fornece uma visão geral dos componentes de *hardware* e dos principais sensores que estão disponíveis nos *frameworks* das plataformas de desenvolvimento para dispositivos móveis. Ele também fornece uma introdução à infraestrutura de sensores.

Observe que os componentes de *hardware* dos dispositivos móveis lhe conferem características particulares, muitas delas não encontradas nos computadores tradicionais. Por isso esses aparelhos vêm ganhando cada vez mais espaço e atenção das comunidades de desenvolvimento de *software*. De acordo com LEE et al. (2005), os principais componentes dos dispositivos móveis são:

- CPU (Unidade Central de Processamento).
- Sistema Operacional.
- Memória.
- Unidade de Armazenamento permanente.
- Baterias e fonte de alimentação.
- Portas de conexão.
- Tela.
- Teclado.
- Periféricos (câmera, GPS, rede etc.).
- Sensores.

De forma geral, os componentes utilizados nos dispositivos móveis como processador, memória e unidades de armazenamento já são bem conhecidos pelos usuários. Entretanto, é importante conhecer os sensores que são os componentes que adicionam capacidades sensitivas e fornecem a capacidade de "perceber" o ambiente que envolve o dispositivo.

Alguns desses sensores são baseados em dispositivos de *hardware* e outros são baseados em *software*. Os sensores baseados em *hardware* são componentes físicos incorporados em um *Smartphone* ou *Tablet*; eles derivam seus dados medindo diretamente propriedades ambientais específicas, como aceleração, força de campo geomagnético ou mudança angular. Os sensores baseados em *software* não são dispositivos físicos, embora imitem sensores baseados em *hardware*; esses sensores derivam seus dados de um ou mais sensores baseados em *hardware* e, às vezes, são chamados de sensores virtuais ou sensores sintéticos. O sensor de aceleração linear e o sensor de gravidade são exemplos de sensores baseados em *software*.

A maioria dos dispositivos móveis é equipada com esses tipos de sensores integrados e pode medir movimento, orientação e várias condições ambientais. Esses sensores são capazes de coletar e fornecer dados brutos com precisão e são úteis para monitorar o movimento ou posicionamento tridimensional do dispositivo, ou, ainda, monitorar as mudanças ao longo do tempo que ocorrem no ambiente que envolve o dispositivo.

Você sabia que

Um jogo pode registrar leituras do sensor de gravidade de um dispositivo para inferir gestos e movimentos complexos do usuário. Inclinação, agitação, rotação ou balanço do dispositivo podem influenciar no cenário e no comportamento do jogo. Da mesma forma, uma aplicação meteorológica pode usar o sensor de temperatura e o sensor de umidade do dispositivo para calcular e relatar a informação coletada do ambiente. Uma aplicação de turismo pode usar o sensor geomagnético e o acelerômetro para relatar as posições das coordenadas geográficas em um mapa que mostra a localização de determinados locais por onde o usuário se deslocará durante seu passeio.

De acordo com o *site* dos desenvolvedores Android (ANDROID DEVELOPERS, "Sensors", 2017), as plataformas de dispositivos móveis geralmente suportam três grandes categorias de sensores, que são: sensores de movimento, sensores ambientais e sensores de posição. Os quais veremos a seguir:

Sensores de movimento

Os sensores de movimento são capazes de mediar as forças de aceleração e as forças de rotação ao longo de três eixos que determinam a posição do dispositivo. Esta categoria inclui acelerômetros, sensores de gravidade e sensores de rotação.

O **Acelerômetro** é um dispositivo que pode medir a força de aceleração, seja por gravidade ou por movimento. Portanto, um acelerômetro pode medir a velocidade de movimento de um dispositivo ao qual está acoplado. Como um acelerômetro detecta movimento e gravidade, também pode sentir o ângulo em que o dispositivo está sendo mantido (Wikipédia "Accelerometer", 2017). Por isso, o dispositivo móvel pode ser utilizado como um *joystick* em alguns tipos de jogos. O usuário controla o personagem do jogo manipulando o dispositivo móvel como se fosse um *joystick*. O acelerômetro pode sentir a inclinação, o movimento e a velocidade que lhe são aplicados, bem como a direção em relação à tela. Quando utilizado em conjunto com sensor giroscópio, é possível detectar o movimento em seis direções (esquerda, direita, para cima, para baixo, frente e verso). Esse sensor também detecta movimentos baseados em três dimensões, X, Y e Z.

O **Giroscópio** é um dispositivo que pode medir a posição relativa ao ângulo dos eixos X, Y e Z e o sentido da rotação. Esse sensor pode detectar pequenas mudanças de direção em relação aos eixos, tanto em sentido horário como em sentido anti-horário (Wikipédia, "Giroscope", 2017; ANDROID DEVELOPER, 2017). O giroscópio é calibrado para dar uma leitura com o valor zero quando o dispositivo é mantido em uma superfície horizontal plana. Qualquer alteração na orientação do dispositivo é medida pelo sensor giroscópio. Os dispositivos móveis utilizam o giroscópio para detectar sua orientação. Quando o usuário vê uma imagem mantendo seu telefone na horizontal e após girar o telefone para a posição vertical observa-se que a imagem acompanha o movimento, esse efeito é obtido com a ajuda da função do sensor giroscópio.

Sensores ambientais

Os sensores ambientais são capazes de medir vários parâmetros ambientais, tais como temperatura e pressão do ar ambiente, iluminação e umidade. Essa categoria de sensores inclui o barômetro, o fotômetro e o termômetro.

O **Barômetro** é um dispositivo que opera em conjunto com o acelerômetro e pode medir a pressão atmosférica. A tendência de pressão pode prever mudanças de curto prazo no clima. Medições da pressão do ar são utilizadas na análise do tempo (Wikipédia, "Barometer", 2017). Esses sensores são usados para informar a previsão do tempo, ou, por exemplo, para monitorar a saúde do usuário do dispositivo móvel, que com isso consegue saber se está gastando mais ou menos calorias em uma caminhada.

O **sensor de proximidade** é um dispositivo capaz de detectar a presença de objetos próximos ao dispositivo móvel. Esse sensor emite um campo eletromagnético ou um feixe de radiação eletromagnética, como por exemplo o infravermelho, e procura mudanças no campo ou sinal de retorno. A distância máxima que esse sensor pode detectar um objeto é definida como "faixa nominal". Alguns sensores podem ter ajustes da faixa nominal.

Em dispositivos móveis do tipo *smartphone*, os sensores de proximidade são utilizados para detectar se alguém está em alcance nominal. Quando ocorre uma chamada telefônica,

o sensor de proximidade irá medir a distância entre o dispositivo e o rosto e pausar as tarefas que estão sendo executadas, como por exemplo navegação na web, reprodução de música ou vídeo. Após finalizar o telefonema, o sensor vai continuar essas tarefas novamente. Durante uma chamada telefônica, esses sensores desempenham um papel na detecção de toques acidentais na tela *touchscreen* quando segurados próximos a orelha.

O **sensor de luminosidade** é um dispositivo utilizado para fornecer informações sobre os níveis de luz ambiente. Em dispositivos móveis, esses sensores podem medir a intensidade da luz do ambiente em que o usuário está e, com isso, ajustar o brilho da sua tela de acordo com a iluminação externa. Isso é uma forma de economizar energia. Também pode ser utilizado para mudar ou aumentar a intensidade do brilho do *flash* quando o usuário fotografa um objeto.

Sensores de posição

Os sensores de posição medem a posição física de um dispositivo. Essa categoria inclui os sensores de orientação, como o Compass e o magnetômetro.

O **Compass**, mais conhecido como bússola, é um dispositivo utilizado para navegação e orientação que mostra a direção em relação ao norte geográfico. Esse sensor usa componentes eletrônicos internos altamente calibrados para medir a resposta do dispositivo ao campo magnético da Terra.

O **GPS** (*Global Positioning System*), sistema de posicionamento global, é um dispositivo de navegação capaz de receber informações de satélites e depois calcular a posição geográfica do dispositivo. Com o GPS e um *software* apropriado, o dispositivo pode exibir a posição do dispositivo móvel em um mapa e pode oferecer instruções de deslocamento.

De acordo com o site Wikipédia (2017), em áreas urbanas, os sinais de satélite podem ser enfraquecidos devido ao desvio de sinal que ocorre nas estruturas como prédios e túneis ou por condições meteorológicas. Os sistemas de GPS dos smartphones contam com a tecnologia A-GPS, que utiliza a estação base ou as torres de celular para fornecer a capacidade de rastreamento de localização do dispositivo, especialmente quando os sinais GPS são fracos ou indisponíveis. No entanto, a parte da rede móvel da tecnologia A-GPS não estaria disponível quando o *smartphone* estiver fora do alcance das torres de recepção móvel.

O desenvolvedor pode acessar os sensores disponíveis no dispositivo e adquirir os dados coletados usando os componentes da API da plataforma de desenvolvimento. A estrutura de componentes de *software* fornece várias classes e interfaces que o ajudam a realizar uma grande variedade de tarefas relacionadas ao sensoriamento do ambiente.

Com a API de componentes da plataforma, é possível fazer o seguinte:

- Determinar quais sensores estão disponíveis em um dispositivo.
- Determinar quais são os recursos de um sensor individual, como: seu alcance máximo, o fabricante, os requisitos de energia e a resolução de dados.
- Coletar os dados brutos do sensor e definir o intervalo mínimo de tempo no qual a aplicação deve adquirir os dados do sensor.
- Registrar e anotar os métodos de eventos dos objetos que monitoram as mudanças nos dados coletados do sensor.

Tela sensível ao toque (*touchscreen*) – embora este componente já seja utilizado nos computadores tradicionais, isso não é tão comum como nos dispositivos móveis. Das ações que normalmente eram executadas com auxílio do *mouse*, a maioria agora é realizada nas pontas dos dedos. Isso torna as tarefas de digitação não tão agradáveis, os teclados virtuais não são muito amigáveis e essas particularidades influenciarão na usabilidade das aplicações móveis.

Essas são algumas das características mais importantes a serem consideradas no planejamento de aplicações para dispositivos móveis. Agora que você conhece as particularidades funcionais dos dispositivos móveis, é hora de aprender um pouco mais sobre o desenvolvimento de aplicações para esses dispositivos. Vamos começar mostrando uma perspectiva arquitetural, a forma como devem ser estruturados e organizados os componentes da aplicação.

3. DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS

Neste tópico, vamos conhecer um fator importante no desenvolvimento de aplicações para dispositivos móveis: que é entender como estão estruturados e organizados os componentes da aplicação. Isso ajudará o desenvolvedor a encontrar os locais de alteração de código e a localizar mais facilmente os locais onde os erros de código ocorrem. Em geral, a maioria das plataformas de desenvolvimento para dispositivos móveis tem como padrão de desenvolvimento as arquiteturas em três camadas e basicamente o padrão de desenvolvimento Web com a arquitetura Cliente/Servidor.

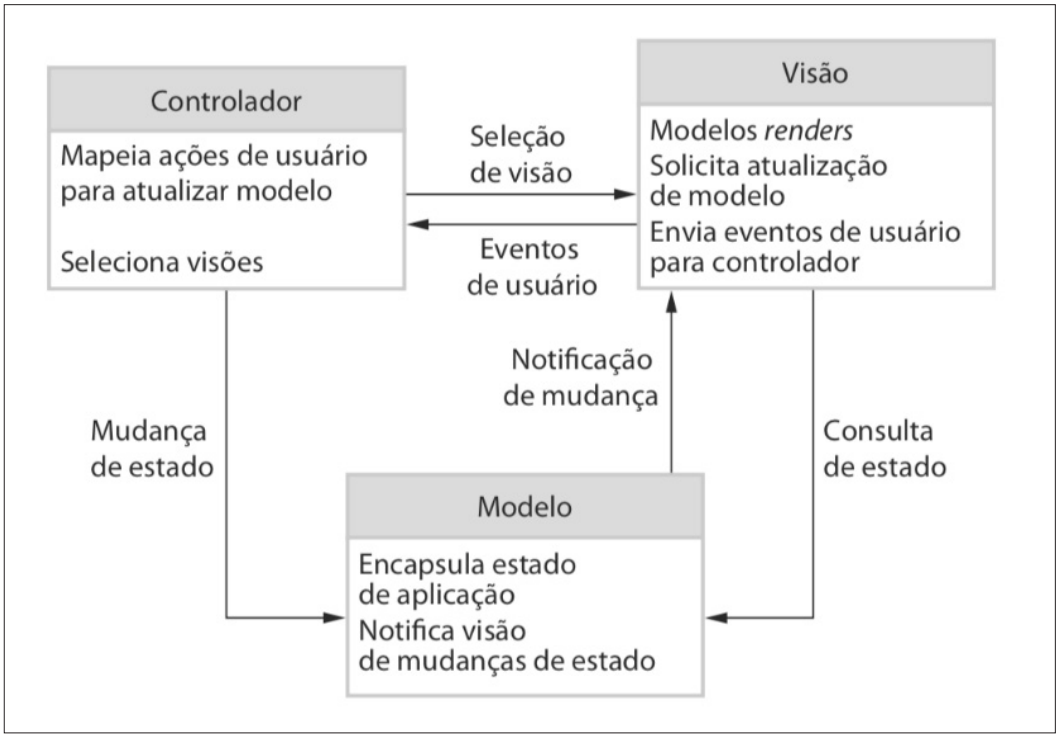
Em uma arquitetura cliente-servidor, a funcionalidade do sistema está organizada em serviços associados e clientes que acessam e usam os serviços. Cada serviço é prestado por um servidor, e os clientes são os usuários, que, por meio de uma rede, podem acessar os servidores e fazer uso dos serviços. Os aplicativos para dispositivos móveis representam o papel de clientes nesse padrão de arquitetura (LEE et al. 2005).

As aplicações para dispositivos móveis, geralmente, utilizam arquitetura que separa os componentes de dados dos componentes de visualização da aplicação. Essa arquitetura é constituída pela Camada de apresentação, Camada de Modelagem e Camada de acesso a dados. Esse modelo estrutural é implementado de acordo com o padrão arquitetural MVC (*Model - View - Controller*) e é a referência para o gerenciamento de interação em muitos sistemas baseados na web.

A arquitetura baseada em camadas organiza o sistema em camadas com um conjunto de funcionalidades relacionadas e associadas a cada camada. Uma camada fornece serviços à camada acima dela; assim, os níveis mais baixos de camadas representam os principais serviços suscetíveis de serem usados em todo o sistema. Em geral, a camada mais baixa inclui *software* de apoio ao sistema, como gerenciadores de banco de dados e sistema operacional (SOMMERVILLE, 2011).

De acordo com Sommerville (2011), esse padrão separa a apresentação e a interação dos dados do sistema. Para tanto, o sistema é estruturado em três componentes lógicos que interagem entre si. Conforme mostra a Figura 1, o componente da camada de modelagem (*model*) gerencia o sistema de objetos que representam os dados e as operações associadas a

esses dados. O componente da camada de visão (*view*) define e gerencia como os dados são apresentados ao usuário. O componente da camada de controle (*control*) gerencia, por meio de eventos, a interação do usuário com as telas do sistema e passa essas interações para os componentes de Visão e Modelo.



Fonte: Sommerville (2011, p. 109).
Figura 1 *O modelo estrutural do padrão MVC*.

É comum associar ao padrão MVC uma quarta camada com o objetivo específico de fornecer o acesso a dados. Geralmente a comunidade de desenvolvedores costuma nomear essa camada como DAO (*Data Accsses Object*). A camada de acesso a dados é composta exclusivamente de classes que acessam os serviços de gerenciadores de bancos de dados (SGDB) e têm o objetivo específico de manipular os dados da aplicação. Todas as funções relacionadas com a manipulação de dados, como inserção, alteração, exclusão e pesquisa, são reservadas a essa camada, inclusive a conexão com o banco de dados.

Os componentes distribuídos por todas essas camadas são implementados como classes no paradigma de programação orientada a objetos; nesse caso, toda a modelagem e relacionamentos de classes podem ser representados com o diagrama de classes UML. A implementação de código geralmente utiliza linguagens como Java, C#, Javascript e Objectve-C; isso vai depender de qual plataforma de desenvolvimento foi escolhida para produzir o aplicativo.

A formação básica da estrutura de um projeto de aplicação para dispositivos móveis é construída de forma automática por meio de geração de código com o auxílio de ferramentas do tipo IDE (*Integrated Development Environment*), que são ambientes de desenvolvimento integrado que, ao gerarem um novo projeto, também criam a estrutura básica de componentes e realizam a configuração padrão inicial. No caso da plataforma Android, a ferramenta IDE

utilizada é o Android Development Kit; no caso do Windows Phone, a ferramenta IDE utilizada é o Microsoft Visual Studio. A instalação dessas ferramentas, incluindo alguns *frameworks* de desenvolvimento, faz parte da atividade de preparação do ambiente de desenvolvimento.

No caso da plataforma Ionic, não são utilizadas ferramentas do tipo IDE para gerar a estrutura do projeto; em vez disso, o projeto é criado por meio de linha de comando executado diretamente no prompt (Windows) ou terminal (MacOS/Linux), cujo processo será demonstrado no decorrer de seus estudos.

A plataforma Ionic, por sua vez, utiliza muitas tecnologias que foram criadas para o desenvolvimento de aplicação para web, incluindo o HTML, CSS, Javascript, Bootstrap, etc que já são mais conhecidas. A seguir, apresentamos as características destas tecnologias.

O **HTML** (W3SCHOOLS, "HTML", 2017) é uma linguagem de marcação padrão para criar conteúdo web; a plataforma Ionic utiliza o HTML ou o HTML5 para criar as páginas visuais de interação com o usuário. O HTML (*Hyper text Markup Language*) descreve a estrutura das páginas da aplicação usando *tags* de marcação. No código HTML, as *tags* são elementos dispostos como blocos de construção e utilizados para estruturar e dar significado ao conteúdo web.

Agora, veremos o **CSS** (*Cascading Style Sheets*) (W3SCHOOLS, "CSS", 2017), que é uma linguagem de regras de estilo que descreve o efeito visual que os elementos devem ter quando apresentados na tela por meio de um documento HTML. O CSS descreve propriedades de cor, forma, tamanho, entre outras, afetando a aparência dos elementos na tela. As definições de estilo normalmente são salvas em arquivos .css externos. Com um arquivo de folha de estilo externo, é possível alterar a aparência de um *site* inteiro sem alterar código da aplicação, mudando apenas no arquivo .css.

O **SASS** (*Syntactically Awesome Style Sheets*) (SASS, 2017) é uma extensão da linguagem de regras de estilo CSS que permite o uso de novos recursos, como, por exemplo, a criação de variáveis, regras aninhadas, importações em linha e outras soluções, e que tem o objetivo de acelerar o processo de desenvolvimento de código para regras de estilo.

O **Javascript** (W3SCHOOLS, "Javascript", 2017) é uma linguagem de programação que permite criar conteúdo que se atualiza dinamicamente nas páginas web. Esse conteúdo dinâmico refere-se à habilidade de atualizar a exibição de uma página web para mostrar os elementos de acordo com novas circunstâncias, gerando novo conteúdo conforme solicitado. É mais utilizada para fornecer comportamento aos elementos do conteúdo web. Inicialmente, foi implementada para rodar apenas no lado do cliente em navegadores da web, porém atualmente os mecanismos da linguagem Javascript estão incorporados em outros aplicativos que permitem rodar do lado do servidor.

Ao carregar uma página web, o código Javascript é executado pelo navegador somente depois que o HTML e CSS forem interpretados e colocados juntos na página. Isso assegura que toda a estrutura e a aparência da página estejam no lugar certo na hora em que o Javascript for executado.

O código HTML e o CSS são implementados na camada *View* da aplicação, já o código Javascript pode ser implementado na camada Cliente ou Servidor. A junção dessas tecnologias

tem o objetivo de definir por meio de programação o conteúdo, a aparência e o comportamento dos elementos. Observe que:

- O código HTML vai definir a estrutura do conteúdo das páginas web.
- O código CSS vai especificar a aparência dos elementos das páginas da web.
- O código Javascript vai programar o comportamento dos elementos e das páginas da web.

Quando o navegador web faz uma requisição para receber uma nova página, todas essas tecnologias (HTML, CSS, Javascript) são carregadas e interpretadas para realizar a apresentação do conteúdo da página web para o usuário. Essa ação realizada pelo navegador web é chamada de renderização.

O **Bootstrap** (BOOTSTRAP, 2017) é um *framework* desenvolvido com tecnologias HTML, CSS e Javascript, que apoia a produção de *sites* e aplicativos para web. Ele contém modelos de *design* para tipografia, formas, botões, navegação e outros componentes de interface. É muito popular entre os desenvolvedores que buscam implementar o *design* web responsivo.

Você sabia que o *design* web responsivo refere-se à criação de *sites* e aplicativos que se ajustam automaticamente para ter boa aparência em todos os tipos de telas dos dispositivos, desde pequenos *smartphones*, *tablets* até grandes monitores de *desktops.*

Como vimos, são várias as tecnologias utilizadas no desenvolvimento de aplicações para dispositivos móveis, por isso é importante conhecê-las bem e saber utilizá-las. Conhecer bem as estruturas nas camadas da arquitetura do aplicativo é fundamental para sua aprendizagem. Outro aspecto importante é você saber que a maioria dessas tecnologias é utilizada para o desenvolvimento de aplicações para web; isso se deve ao fato de que muitas empresas de dispositivos móveis as incorporaram em suas plataformas.

Estudamos o Ionic, que é uma plataforma de desenvolvimento híbrida, e outras plataformas como iOS , Android e Windows Phone que são plataformas de desenvolvimento nativas. A seguir, serão apresentados os conceitos sobre aplicativos de plataformas híbridas e plataformas nativas. Vamos lá!

4. APLICAÇÃO NATIVA OU HÍBRIDA?

Antes de aprofundarmos nossos estudos sobre as plataformas nativas e as plataformas híbridas, é importante que você entenda que na plataforma nativa o desenvolvedor fica atrelado ao sistema operacional e às ferramentas de desenvolvimento fornecidas pelo fabricante, enquanto que nas plataformas híbridas as tecnologias e ferramentas são as mesmas utilizadas para o desenvolvimento web, acrescidas com o complemento de *frameworks* e APIs que, na maioria dos casos, são de código livre. O desenvolvedor implementa a aplicação base e ao final gera uma versão de seu aplicativo para cada plataforma nativa. Os detalhes, as vantagens e as desvantagens entre os dois tipos de plataforma serão abordadas a seguir.

Um aplicativo móvel nativo é uma aplicação para dispositivos móveis codificada em uma linguagem de programação específica da plataforma de desenvolvimento a qual pertence o dispositivo. Geralmente, a empresa que detém a propriedade do dispositivo é quem define qual será o sistema operacional e a linguagem de programação oficial a ser utilizada.

Como os aplicativos nativos são escritos para uma plataforma específica, eles podem interagir e aproveitar os recursos do sistema operacional e outros *softwares* normalmente instalados nessa plataforma. Assim, ele possui a capacidade natural de usar *hardware* e *software* específico do dispositivo, o que significa que aplicativos nativos podem aproveitar a tecnologia mais recente disponível em dispositivos móveis. Isso pode ser interpretado como uma vantagem para os aplicativos.

Além disso, os aplicativos móveis nativos oferecem um desempenho rápido e um alto grau de confiabilidade. Eles também têm uma boa integração com a infraestrutura de *hardware* e fornecem acesso a todos os recursos, inclusive aos sensores do dispositivo. No entanto, na aplicação produzida para a plataforma nativa, o desenvolvedor fica vinculado ao sistema operacional e à plataforma de desenvolvimento. As plataformas nativas mais utilizadas são:

- iOS desenvolvido pela empresa Apple.
- Android desenvolvido pela empresa Google.
- Windows Phone desenvolvido pela empresa Microsoft

Todas essas empresas oferecem um SDK (*software development kit*) para instalação das ferramentas de desenvolvimento. A Apple utiliza o sistema operacional iOS e as linguagens *Objective-C* e *Swift*; o Google utiliza o sistema operacional Android e a linguagem Java; e a Microsoft utiliza o sistema operacional Windows Phone e linguagens como C# e C++.

Como já foi mencionado, os aplicativos móveis híbridos são criados com uma combinação de tecnologias da web como HTML, CSS e JavaScript. Desenvolvedores que já possuem experiência com essas tecnologias podem reutilizar esse conhecimento para desenvolver nas plataformas híbridas. Uma vez que o aplicativo móvel está pronto, ele pode ser executado como qualquer outro aplicativo. Uma grande vantagem é que a partir da aplicação criada na plataforma híbrida podem ser gerados aplicativos para outras plataformas nativas, ou seja, o aplicativo é implementado uma única vez e pode ser reutilizado para outras plataformas.

As plataformas de desenvolvimento híbrido mais conhecidas são Ionic, PhoneGap e Xamarin, além destas plataformas existem outras, pois esta é uma área que está em expansão.

Em ambas as plataformas, a distribuição dos aplicativos ao usuário final é realizada por meio de *sites* de lojas de aplicativos conhecidas como App Store.

Por exemplo, uma loja de aplicativos é um portal online de distribuição digital onde os programas de *software* estão disponíveis para aquisição e *download*. Todos os principais fornecedores de sistemas operacionais móveis, incluindo Apple, Google e Microsoft, disponibilizam suas próprias lojas de aplicativos, o que lhes garante controle sobre o *software* disponível em suas respectivas plataformas, além disso, pode existir lojas de aplicativos de terceiros em operação.

As lojas de aplicativos de terceiros geralmente assumem a forma de uma loja online, em que os usuários podem navegar por meio dessas diferentes categorias de aplicativos, visualizar

informações sobre cada aplicativo com indicadores de avaliações ou classificações feitas pelos usuários e adquirir o aplicativo por meio de compra, se necessário. Muitos aplicativos são oferecidos sem nenhum custo. O aplicativo selecionado é disponibilizado pela loja para *download* automático, em seguida, o aplicativo é instalado no dispositivo do usuário.

Alguns proprietários das plataformas móveis exigem que os aplicativos passem por um processo de aprovação antes de disponibilizá-los para *download* ao usuário final. Esses aplicativos são inspecionados quanto ao cumprimento de determinadas diretrizes impostas, como o controle de qualidade, por exemplo, incluindo o requisito de que uma comissão seja cobrada sobre a venda de cada cópia do aplicativo.

Os estudos realizados neste tópico apresentam uma visão geral das questões que o desenvolvedor deve responder quando estiver pensando sobre qual tipo de plataforma escolher ao iniciar projetos para aplicativos móveis.

No próximo tópico, estudaremos como os dados da aplicação podem ser armazenados e recuperados; esse é outro ponto importante que afeta a forma como uma aplicação deve ser planejada e produzida e deve ser considerado desde o início do projeto.

5. INSTALANDO A PLATAFORMA IONIC

Neste tópico, vamos descrever o processo de instalação do ambiente de desenvolvimento da plataforma Ionic *Framework* e todas as dependências necessárias para permitir a criação de novos projetos. É importante que você saiba que as dependências são todos os aplicativos, *frameworks* e APIs necessários para o funcionamento do Ionic. Vamos lá!

O Ionic *framework* permite desenvolver aplicativos para qualquer sistema operacional que o usuário preferir. Desde o início, o Ionic foi desenvolvido para ser multiplataforma, no que se refere a sistemas operacionais. O desenvolvedor pode optar por MacOS, Linux e Windows. Para todos esses sistemas, é comum utilizar o prompt de linha de comando para a maioria dos procedimentos realizados, tanto na instalação quanto na criação e no desenvolvimento de novos projetos.

Inicialmente, apresentaremos cada uma das dependências necessárias para compor o ambiente e, em seguida, daremos sequência aos procedimentos para a instalação.

NodeJS

O NodeJS é a base que sustenta a execução de um aplicativo Ionic na fase de desenvolvimento. Inicialmente se parece com um servidor de aplicações, mas, de acordo com o fabricante o NodeJS, não é um servidor de aplicações.

Construído sobre a máquina virtual Javascript V8 do Google e aproveitando a sua simplicidade para escrever aplicações assíncronas, o NodeJS é uma plataforma para implementar aplicações *server-side* de sistemas distribuídos, rápidas e escaláveis. Utiliza um modelo de Input/Output direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.

De acordo com o site Nodejs.org (2017), em linguagens que utilizam servidores de aplicação, cada conexão cria uma nova *thread* que potencialmente utilizará 2 MB de memória RAM. Em um sistema que tenha 8 GB de RAM, isso leva, teoricamente, ao número máximo de conexões concorrentes de cerca de 4.000 usuários. Caso o número de usuários precise aumentar, será necessário adicionar mais e mais servidores de aplicação para fazer o balanceamento de carga, o que pode aumentar a ocorrência de possíveis problemas técnicos.

O NodeJS resolve este problema utilizando o método *single threaded* para tratar o gerenciamento de conexões no servidor. Ao invés de criar uma nova *thread* a cada conexão, um evento é executado para cada conexão dentro do controle de processos e as chamadas de input/output são controladas para evitar possíveis *deadlocks*, assim um servidor rodando NodeJS pode suportar dezenas de milhares de conexões simultâneas.

Entre as dependências do Ionic *framework*, o NodeJS é o primeiro *software* a ser instalado no ambiente de desenvolvimento. O Ionic utiliza o NodeJS para executar os aplicativos.

NPM – *Node Package Manager*

O NPM (*Node Package Manager*) é um utilitário de linha de comando utilizado para gerenciar a instalação de pacotes de plataformas de desenvolvimento baseadas no NodeJS. Para gerenciar os pacotes, o NPM interage com um repositório online, que ajuda na instalação de pacotes, gerenciamento de versão e gerenciamento de dependências. Os pacotes podem ser encontrados por meio do portal de busca da NPM. Uma vez encontrado o pacote, ele pode ser instalado com uma única linha de comando. O NPM é instalado junto com o NodeJS.

Outro fator importante no uso do NPM é a facilidade no gerenciamento de dependências de um projeto. Quando um projeto NodeJS possui um arquivo package.json em sua estrutura de arquivos, basta rodar o comando *npm install* na pasta raiz do projeto e o NPM vai instalar todas as dependências declaradas no arquivo package.json (NODEBR, 2017).

O Ionic utiliza o NPM para instalação ou atualização das dependências de sua plataforma.

Apache Cordova

Apache Cordova é um *framework* de desenvolvimento para dispositivos móveis. Sua licença de uso é de código aberto. Ele fornece acesso a toda infraestrutura de *hardware* do dispositivo móvel, permite o uso de tecnologias web padrão, HTML5, CSS3 e Javascript e o desenvolvimento de aplicativos para várias plataformas nativas (APACHE CORDOVA, 2017).

Os aplicativos são executados dentro de pacotes, chamados de *Engine WebView*, direcionados a cada plataforma e dependem de APIs compatíveis com padrões para acessar as capacidades, como sensores, dados, status da rede etc., de cada dispositivo (APACHE CORDOVA, 2017). A Figura 2 ilustra a arquitetura de uma aplicação construída com o apoio do *framework* Cordova.

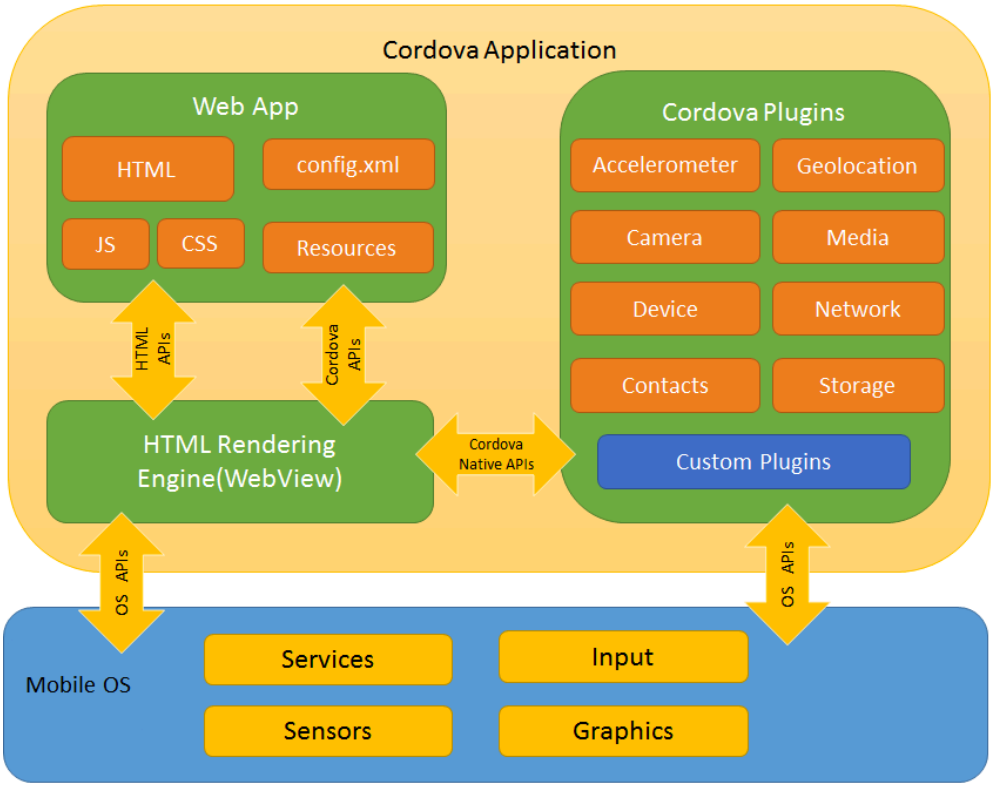


Figura 2 *Arquitetura utilizando o framework Cordova*.

O componente WebView envolve toda a aplicação em um pacote construído para executar na plataforma móvel, fornecendo toda a parte visual de interface com o usuário, ou seja, as páginas ou telas de interação com o usuário do aplicativo. O WebView também inclui o Cordova *plugin*, que são os componentes que fornecem acesso aos recursos nativos; isso inclui toda a infraestrutura de *hardware* do dispositivo. Essa é a forma como um aplicativo híbrido baseado no *framework* Cordova é construído para funcionar no dispositivo móvel.

Além dos *plugins* do Cordova, existem também vários outros *plugins* de terceiros que fornecem ligações adicionais para recursos que não estão necessariamente disponíveis em todas as plataformas móveis.

Mobile OS representa o sistema operacional e os componentes de *software* que interagem com o dispositivo móvel.

O Ionic utiliza o Apache Cordova para obter acesso à infraestrutura de *hardware* do dispositivo móvel e para empacotar o projeto em um aplicativo executável no sistema operacional do dispositivo.

Ionic Framework

Atualmente, existe um amplo número de aplicativos web sendo utilizados por usuários da internet, assim como existem milhares de desenvolvedores que dominam essas tecnologias que envolvem o HTML. Com os avanços da tecnologia móvel, *smartphones* e *tablets* agora também são capazes de executar aplicações web com o apoio da plataforma Ionic.

O Ionic é um *framework* de desenvolvimento para a construção de aplicativos móveis que utiliza as tecnologias Web, HTML5, CSS e Javascript. O Ionic, em conjunto com suas dependências, estabelece uma base para um desenvolvimento mais avançado de aplicativos híbridos para dispositivos móveis. Se você já utiliza as tecnologias Web para desenvolvimento de *software*, o Ionic será uma boa escolha, pois todo o conhecimento poderá ser reaproveitado no aprendizado dessa nova plataforma. Basicamente, o Ionic fornece uma ampla coleção de componentes de interface com o usuário que proporcionam uma atraente experiência de uso com belas animações gráficas. Os componentes se ajustam perfeitamente à tela dos dispositivos móveis.

Como o Ionic é um *framework* Web, ele precisa ser empacotado em um aplicativo nativo para poder ser executado no dispositivo, cuja função é atribuída ao *framework* Apache Cordova.

Outro detalhe importante é que o Ionic utiliza o AngularJS Extensions, o qual segue o padrão View Controller, popularizado em *frameworks* que implementam a arquitetura MVC (*Model, View, Controler*). No padrão do controlador de visualização, são tratadas as diferentes seções de interface com o usuário. Os controladores de visualização utilizam o Javascript para ativar as visualizações e fornecer funcionalidades de interação com o usuário.

Durante as etapas de desenvolvimento, as aplicações construídas com o *framework* Ionic podem ser executadas e testadas diretamente no navegador Web de sua preferência, Firefox, Chrome, Safari etc. Dessa forma, você poderá obter uma visão de seu aplicativo funcionando durante o desenvolvimento. A função de executar o aplicativo no navegador é obtida com o apoio do NodeJS, que desempenha o papel de servidor.

Muitas atividades de desenvolvimento são realizadas digitando comandos na linha de prompt. O CLI, ou interface de linha de comando, é uma ferramenta que fornece uma série de comandos úteis para desenvolvedores da plataforma Ionic. Além de instalar e atualizar o Ionic, o CLI é como um servidor de desenvolvimento embutido; ferramenta de compilação e depuração são outras funções fornecidas por ele. Se você estiver usando o Ionic online, a CLI pode ser usada para exportar código e até mesmo acessar e interagir com sua conta do *site* Ionic.

Na aplicação Ionic, a navegação funciona como uma estrutura de dados em pilha, ao contrário de muitas aplicações em que as páginas vão sendo substituídas umas pelas outras para a visualização do usuário. No Ionic, a página que está sendo mostrada é a que está no topo da pilha; isso é feito com o comando *push* (empurre uma página para a pilha para navegar até ela) e o comando *pop*, que é acionado para voltar.

A aparência visual da aplicação foi projetada com o conceito de configuração por temas, ou seja, as características visuais de cor e forma são definidas pelo tema e por padrão; o aplicativo vai se adequar ao visual das aplicações nativas, de acordo com o sistema operacional, seja iOS, Android ou Windows Phone.

Assim, como é prática em outras tecnologias, o *framework* Ionic dispõe de uma enorme comunidade de desenvolvedores espalhados pelo mundo todo e que trabalham em conjunto para o seu crescimento e desenvolvimento. Várias empresas já estão utilizando o Ionic para produzir novos aplicativos móveis. Sua distribuição é feita na forma de *software* com licença de código aberto. Isso significa que você pode usar o Ionic em seus próprios projetos, pessoais ou comerciais gratuitamente.

Typescript e Javascript

A partir da versão Ionic 2.0, o Typescript foi incorporado à plataforma de desenvolvimento.

O Typescript, opcionalmente, pode ser uma linguagem estaticamente tipada, isto é, uma linguagem de programação que usa variáveis com tipos específicos. O Typescript não obriga os desenvolvedores a adicionar tipos, mas, uma vez declarados, eles se tornam estáticos. Os tipos podem ser adicionados a variáveis, funções, propriedades etc. Isso ajudará o compilador a mostrar avisos sobre possíveis erros no código antes que um aplicativo seja executado.

Nas linguagens estaticamente tipadas, uma vez que a variável foi declarada com um tipo, por exemplo, tipo inteiro, ela será assim até o seu fim, não podendo mais ser alterada ao longo do programa. Normalmente, possuem declaração explícita de tipo da variável, e isso deve ser especificado logo no momento de sua declaração. Operações entre tipos diferentes muitas vezes ocasionam erro de código que são logo identificados e apontados pelos compiladores.

Encontrar erros de código é uma tarefa difícil, à medida que o código de um aplicativo fica maior e mais complexo; nesse caso, as linguagens com tipos estáticos podem oferecer algumas ótimas vantagens.

O Javascript, por sua vez, é uma linguagem fracamente tipada; no Ionic 1.0, os desenvolvedores codificavam diretamente com Javascript, e isso era um grande problema para identificar erros de código. Na maioria das vezes, o programa simplesmente não executava, e a tarefa de encontrar a causa do erro era extremamente dificultada para programadores não experientes.

A partir do Ionic 2.0, o desenvolvedor codifica a aplicação utilizando o Typescript, e logo após, antes de executar, o código é transcrito para a linguagem Javascript em um movimento denominado "*transpile*".

Para obter mais informações e ampliar seus conhecimentos referentes à linguagem Typescript, consulte a documentação no site typescriptlang disponível no *link* a seguir:

- TYPESCRIPT. Disponível em: <http://www.typescriptlang.org/index.html>. Acesso em: 10 ago. 2018.

Visual Studio Code

O Visual Studio Code é um editor de código-fonte leve e poderoso e está disponível para sistemas operacionais Windows, iOS e Linux. Ele vem com suporte incorporado para as linguagens Javascript, Typescript e Node.js. O editor possui também um rico ecossistema de extensões para outras linguagens, como C++, C #, Java, Python, PHP, Go etc.

Muitos editores de código podem ser utilizados para trabalhar com a plataforma Ionic. Adotamos o Visual Studio Code por ser leve e totalmente compatível com Typescript e com a plataforma Ionic. Outras alternativas para editores de código-fonte podem ser o Atom, Sublime Text, Notepad++ etc.

Sugerimos que você acesse a documentação referente ao editor Visual Studio Code no *site* code.visualstudio.com. Nele, você encontrará informações relevantes para sua atuação profissional. Disponível em: <https://code.visualstudio.com/docs>. Acesso em: 10 ago. 2018.

Preparando o Ambiente de Trabalho

O ambiente de trabalho é a própria plataforma de desenvolvimento e precisa ser bem preparado e configurado para evitar problemas de funcionamento.

Para tanto, é fundamental que você instale o NodeJS, e o primeiro passo é acessar o *site* oficial disponível em <https://nodejs.org>, acesso em: 10 ago. 2018. Clique em *Download* para baixar os instaladores encontrados nessa página. Na Figura 3, é mostrado o *site* do NodeJS com a versão utilizada neste material.

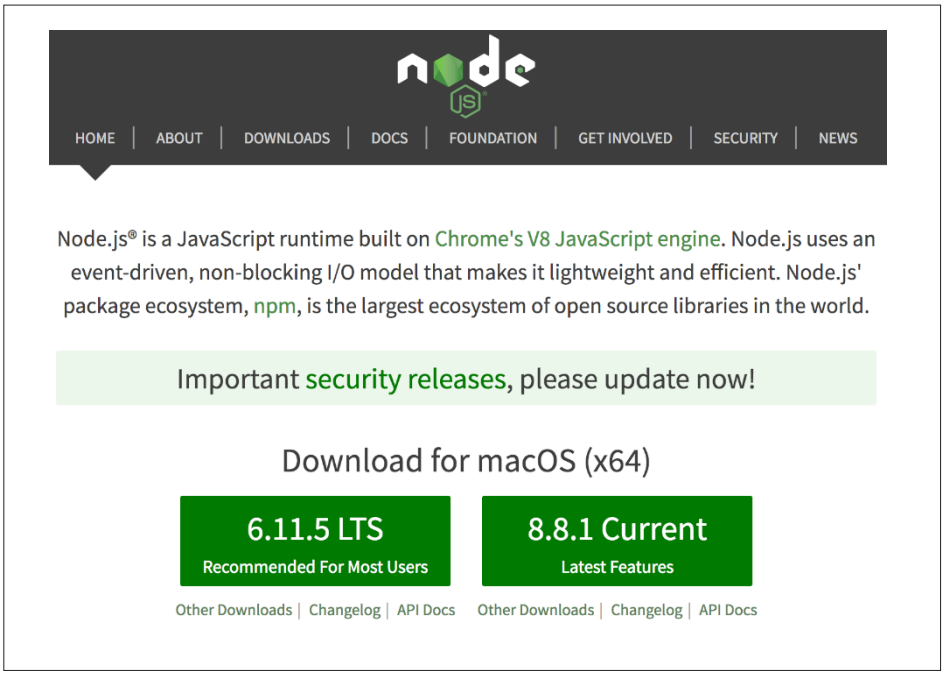


Figura 3 *Realizando download do NodeJS*.

Execute o programa baixado para instalar normalmente. Ao terminar esses procedimentos, o NodeJS e o NPM estarão instalados.

Acesse o prompt de comandos do Windows ou o Terminal do iOS ou Linux e digite o seguinte comando para verificar o funcionamento:

```
node -v
```

Se a instalação ocorreu de forma correta, o sistema mostrará a versão: v6.11.5.

O próximo passo, agora que o NodeJS e o NPM estão instalados, é instalar o Cordova e o Ionic. Isso é feito com apenas um comando na linha de prompt e você deverá digitar:

```
npm install -g cordova ionic
```

Observe a necessidade de usar o comando “sudo” para habilitar a execução de comandos de instalação, caso você utilize o sistema operacional iOS ou Linux.

Após a instalação, para verificar se o funcionamento está correto, digite o comando `cordova –v`, e depois `ionic –v`. Se a instalação ocorreu de forma correta, o sistema mostrará a versão. Em seguida, instale o Typescript com o comando:

```
npm install -g typescript
```

Para finalizar, instale o Visual Studio Code. O primeiro passo é acessar o *site* oficial do editor, disponível em: <https://code.visualstudio.com/Download>, acesso em: 10 ago. 2018. Escolha o *download*, de acordo com o sistema operacional que você utiliza, para baixar o instalador encontrado nessa página.

Após essa sequência de procedimentos, o seu ambiente de desenvolvimento estará pronto para ser utilizado.

Para saber mais sobre como começar a trabalhar com a plataforma Ionic, acesse o *site* disponível em: <https://ionicframework.com/getting-started>. Acesso em: 10 ago. 2018.

6. UTILIZANDO A PLATAFORMA IONIC

Após a instalação do *framework* Ionic e de todas as suas dependências, você já está com o ambiente de desenvolvimento pronto para usar, então é hora de começar com um novo projeto. Vamos lá!

Abra o Terminal ou o prompt de comando e crie uma nova pasta para organizar projetos Ionic. Os comandos são:

```
mkdir dev
```

Em seguida, acesse a pasta dev com o comando:

```
cd dev
```

Agora que você está dentro da pasta dev, vamos utilizar o Ionic CLI para criar um novo projeto; o CLI é instalado junto com o *framework* Ionic e são comandos utilizados na linha de prompt. Para esse procedimento, seu computador deverá estar conectado à internet.

Para criar um novo projeto, digite o comando:

```
ionic start meuprojeto blank
```

Aguarde alguns minutos, pois a criação do projeto depende do *download* de vários arquivos, e o Ionic faz tudo de forma automática. O comando `ionic start` é usado para criar o projeto. Neste exemplo, nomeado como `meuprojeto` , você poderá atribuir o nome que desejar ao seu novo projeto; evite usar espaço, acentuação ou cedilha no nome do seu projeto, e `blank` é o modelo (*template*) de projeto que deve ser criado.

Existem mais três modelos disponíveis, cada modelo possui alguns recursos prontos, o modelo `blank` é o modelo mais básico. Esse assunto será comentado com mais detalhes no decorrer de seus estudos.

Ainda dentro da pasta dev, acesse a pasta do novo projeto digitando o comando:

```
cd meuprojeto
```

Hora de executar seu projeto pela primeira vez. É importante saber que o próximo comando só funciona se você estiver dentro da pasta do projeto Ionic. O comando para executar a aplicação vem do Ionic CLI, digite da seguinte forma:

```
ionic serve
```

Aguarde alguns minutos, pois o Ionic realizará o *build* da aplicação, ou seja, o aplicativo será gerado por meio do projeto. O navegador padrão de seu sistema operacional será carregado e mostrará a tela inicial da aplicação de acordo com o *template blank*. A Figura 4 mostra a tela da aplicação.

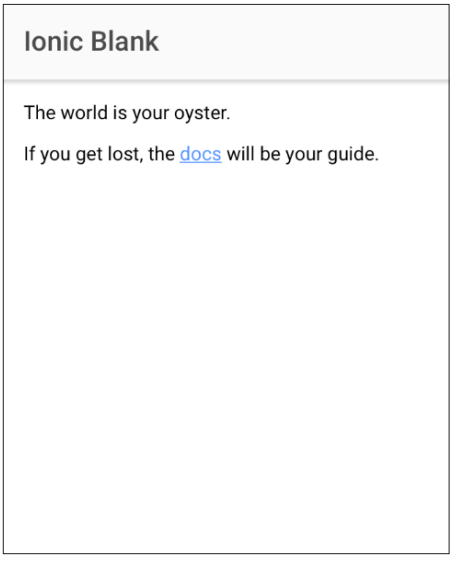


Figura 4 *Tela da aplicação meuprojeto*.

Vamos conhecer um pouco mais sobre a estrutura do projeto Ionic e realizar algumas alterações no código-fonte da aplicação. Nesse momento, utilizaremos alguns comandos e componentes do *framework*, mas não se preocupe, basta seguir as instruções. Explicações mais detalhadas sobre os componentes do *framework* Ionic serão abordadas mais adiante no decorrer de seus estudos.

Abra o editor de código Visual Studio Code. Será apresentada a tela conforme a Figura 5.

Figura 5 *Tela do editor Visual Studio Code*.

Observe a Figura 5, clique no primeiro ícone do lado esquerdo, representado pela figura de uma página, em seguida clique em *open folder*, conforme é mostrado na Figura 6, e direcione para a pasta do seu projeto.

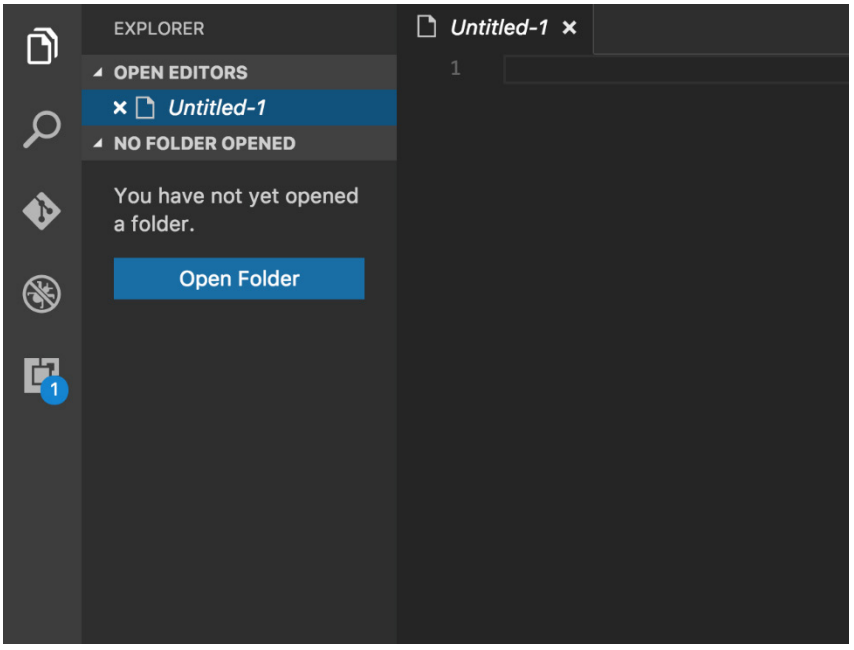


Figura 6 *Abrindo a pasta do projeto*.

Após abrir a pasta do seu projeto Ionic no editor do *Visual Studio Code*, será apresentada a tela como é mostrado na imagem 7.

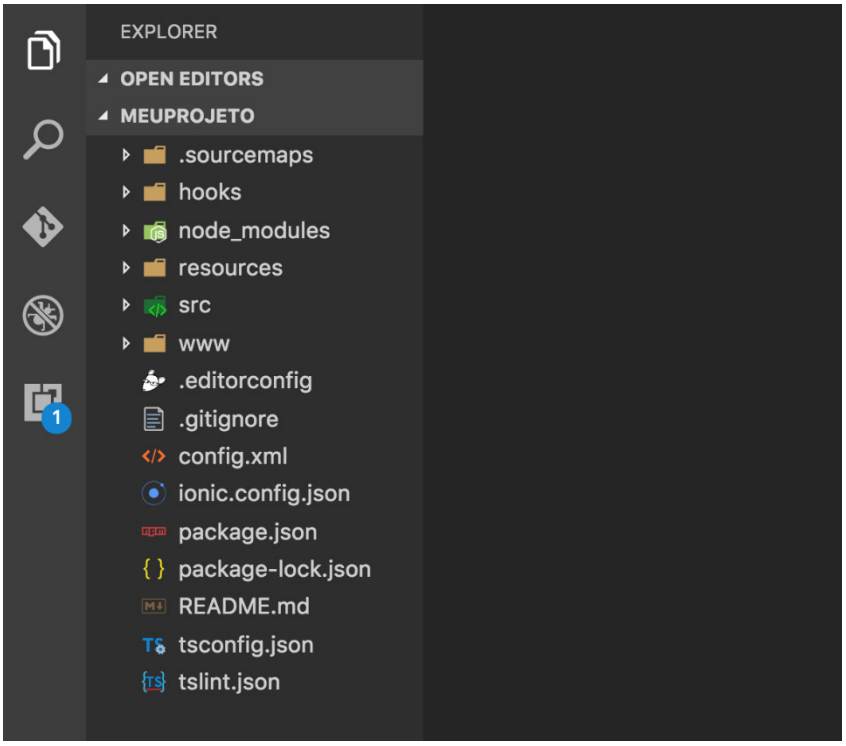


Figura 7 *Estrutura de diretórios do projeto*.

Abra a pasta src e, em seguida, abra a pasta home. Dentro da pasta home, abra o arquivo home.html. Será apresentada a tela, conforme é mostrado na Figura 8.

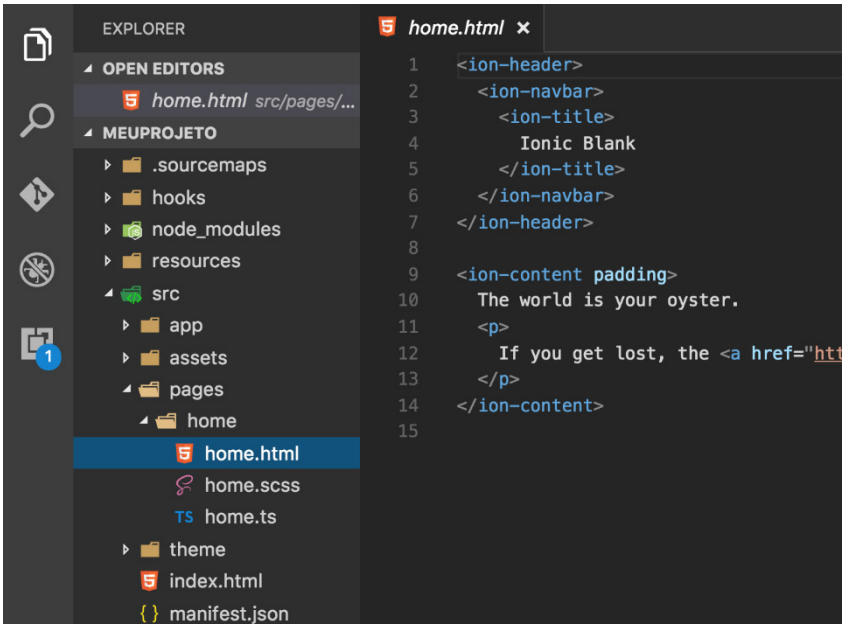


Figura 8 *Abrindo o arquivo home.html*.

Vamos agora alterar a aparência do cabeçalho e incluir um rodapé na aplicação. Observe o Código 1:

Código 1 Página home.html

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Ionic Blank
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  The world is your oyster.
  <p>
    If you get lost, the <a href="http://ionicframework.com/docs/
v2">docs</a> will be your guide.
  </p>
</ion-content>
```

Fim código 1

Faça as alterações para que o código da página home.html fique como o Código 2, mostrado a seguir.

Código 2 Novo código da página home.html

```
<ion-header>
  <ion-navbar color="primary">
    <ion-title>
      Meu Projeto
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  Página Home do projeto.
```

```
    <p>
      Esse é um parágrafo usando uma tag HTML.
    </p>
  </ion-content>

  <ion-footer>
    <ion-toolbar color="secondary">
      <ion-title>Rodapé</ion-title>
    </ion-toolbar>
  </ion-footer>
```

Fim código 2

Após as modificações no código, salve o arquivo. O Ionic realizará o processo de *build* da aplicação e a tela da sua página home.html deverá ser apresentada conforme a Figura 9. Observe que o endereço da URL no navegador é http://localhost:8100; esse endereço aponta para a pasta do seu projeto, é um endereço local em seu computador. O NodeJS é o servidor que simula um ambiente web em seu computador e faz a sua aplicação executar como uma página da Web. Nesse cenário, a função do NodeJS é fornecer o ambiente e o endereço local para aplicação.

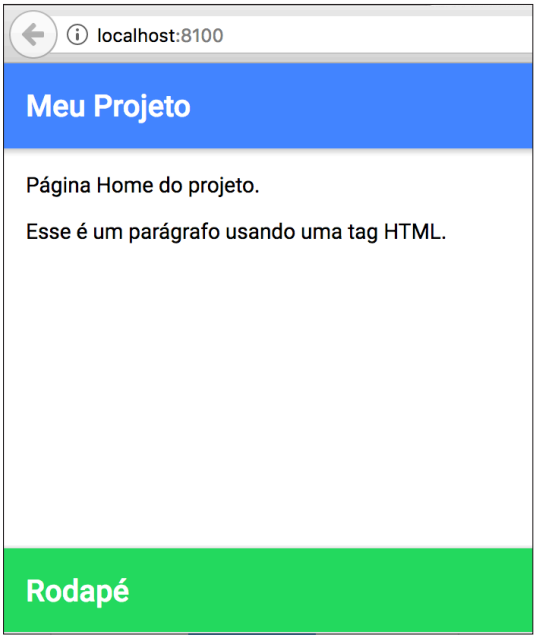


Figura 9 *Tela home.html alterada*.

Observe as alterações que foram efetuadas no código. O primeiro ponto a destacar é que o código da página home.html é dividido em três partes. A parte superior, delimitada

entre as *tags* <ion-header> e </ion-header>. O conteúdo de código entre essas duas *tags* é mostrado na parte superior da página, que recebe o nome de cabeçalho ou também pode ser chamado de barra de navegação.

A parte central, delimitada entre as tags <ion-content> e </ion-content>. Essa é a parte de conteúdo da página, ou seja, os elementos e texto que estarão em evidência, bem no meio da página. Por fim, a parte delimitada entre as *tags* <ion-footer> e </ion-footer>, os elementos colocados entre essas duas *tags* serão apresentados na parte inferior da página.

A maioria dos aplicativos móveis possui alguns elementos fundamentais. Normalmente, esses elementos incluem um cabeçalho e um rodapé que irão reservar a parte superior e inferior da tela. Todos os outros elementos serão colocados entre esses dois. Elementos de conteúdo como esses servem como um recipiente que envolvem todos os outros elementos da tela.

Vamos continuar modificando a aparência da aplicação. Faça novas alterações no código da página home.html para que fique conforme é mostrado no Código 3.

Código 3 Inclusão de botões na página home.html

```
<ion-header>
  <ion-navbar color=”primary”>
    <ion-title>
      Meu Projeto
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  Página Home do projeto.
  <p>
    Esse é um parágrafo usando uma tag HTML.
  </p>

  <button ion-button color=”light”>Light</button>
  <button ion-button>Default</button>
   <br/>
  <button ion-button color=”secondary”>Secondary</button>
  <button ion-button color=”danger”>Danger</button>
```

```
    <br/>
    <button ion-button color="dark">Dark</button>

</ion-content>

<ion-footer>
    <ion-toolbar color="secondary">
      <ion-title>Rodapé</ion-title>
    </ion-toolbar>
</ion-footer>
```

Fim código 3

Após as modificações no código, salve o arquivo. O Ionic realizará novamente o processo de *build* da aplicação. A tela da sua página home.html deverá ser apresentada conforme é mostrado na Figura 10.

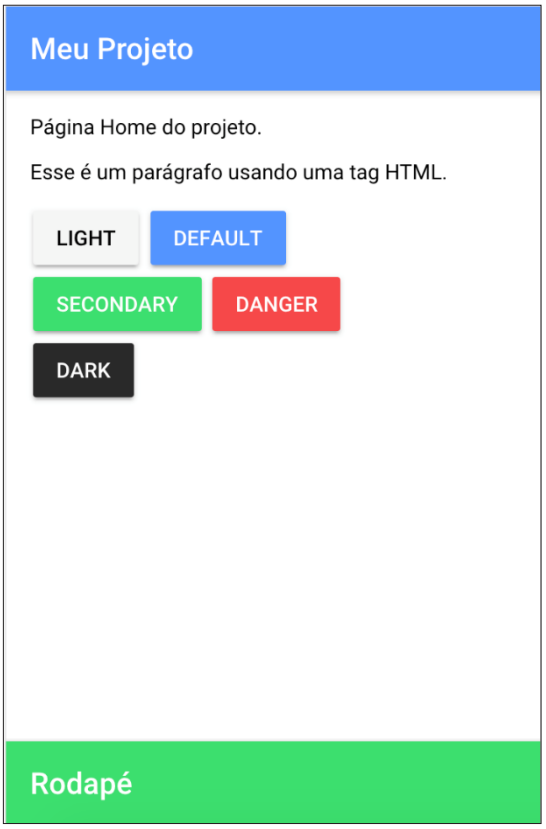


Figura 10 *Inclusão de botões na página home.*

Veja, esse novo código possibilitou incluir na página componentes que representam botões de ação. O componente botão é incluído por meio da *tag* <button>, que é uma *tag* HTML comum, mas o que torna o botão como um componente Ionic é a diretiva complementar

“ion-button”. A formatação que atribui a cor ao botão é a diretiva “color=”. Por exemplo, para formatar o botão com a cor verde, use a diretiva color=”secondary”. A palavra secondary é uma formatação SASS que pertence aos componentes Ionic.

As alterações de código realizadas até o momento afetaram apenas a parte visual da aplicação, ou seja, se pensarmos no padrão MVC (*Model, View, Controller*), o código do arquivo home.html pertence à parte *View* do padrão. Para criar comportamentos que representem ações e funcionalidades na aplicação, é necessário trabalhar em conjunto entre *Controller* e *View*. Isso quer dizer que os métodos que realizam as funções devem ser declarados e implementados no controlador, e na aplicação é representado pelo arquivo home.js.

Resumindo, podemos entender que, quando o usuário interage com a aplicação clicando em um *link*, botão ou outro componente da página home.html, a ação ou comportamento que ocorrerá em seguida estará implementada no código no controlador (home.js). Vamos entender isso utilizando outro componente Ionic, o Action Sheets (folha de ação), e realizando a próxima modificação de código.

Faça novas alterações no código da página home.html para que fique conforme é mostrado no Código 4.

Código 4 Componente botão para o Action Sheets.

```
<ion-header>
  <ion-navbar color=”primary”>
    <ion-title>
      Meu Projeto
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  Página Home do projeto.
  <p>
    Esse é um parágrafo usando uma tag HTML.
  </p>

  <button ion-button block (click)=”mostrarComponente()”>
    Mostrar a folha de ação
  </button>
```

```
</ion-content>

<ion-footer>
  <ion-toolbar color="secondary">
    <ion-title>Rodapé</ion-title>
  </ion-toolbar>
</ion-footer>
```

Fim código 4

Após as modificações no código, salve o arquivo. O Ionic realizará novamente o processo de *build* da aplicação, e a tela da sua página home.html deverá ser apresentada conforme é mostrado na Figura 11.

Observe que na página home.html foi incluído um botão com um evento "click", implementado com o comando `(click)="mostrarComponente()"`. Isso significa que, quando o usuário clicar no botão, o evento irá acionar o método `mostrarComponente()`.

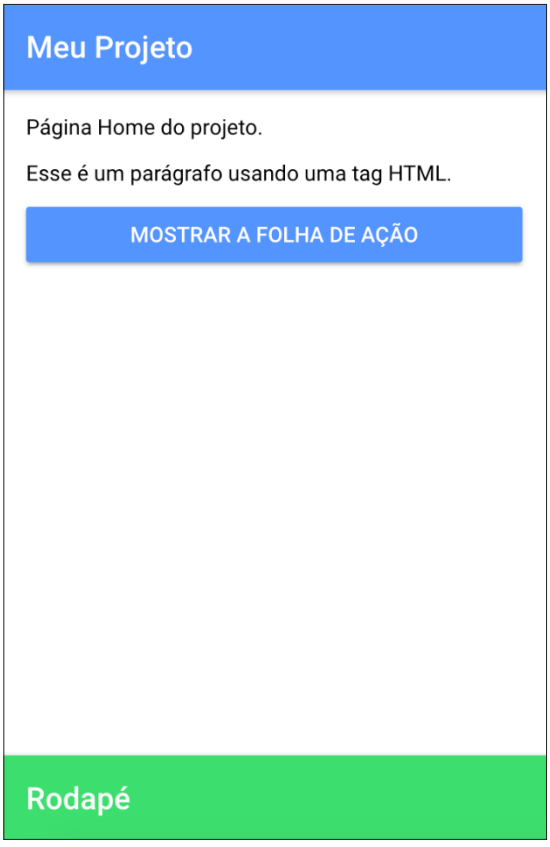


Figura 11 *Página home.html alterada.*

Agora vamos alterar o código do controlador, o arquivo home.js. É importante verificar as alterações que foram feitas, por isso compare o código atual com o código 5 e veja o que

foi modificado, em seguida, faça as alterações para que o código do controlador fique como o Código 5, mostrado a seguir.

Código 5 Arquivo home.js, controlador da página home.html

```
import { Component } from '@angular/core';
import { NavController, ActionSheetController } from 'ionic-angular';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {
  constructor(public navCtrl: NavController,
              public actionsheetCtrl: ActionSheetController) {

  }

  mostrarComponente(){

    let actionSheet = this.actionsheetCtrl.create({

      title: 'Escolha uma ação para a música',
      cssClass: 'action-sheets-basic-page',
      buttons: [
        {
          text: 'Apagar',
          role: 'destructive',
          icon: 'trash',
          handler: () => {
            console.log('Clicou em Apagar');
          }
        },
```

```
        {
          text: 'Compartilhar',
          icon: 'share',
          handler: () => {
            console.log('Clicou em Compartilhar');
          }
        },
        {
          text: 'Tocar',
          icon: 'arrow-dropright-circle',
          handler: () => {
            console.log('Clicou em Tocar');
          }
        },
        {
          text: 'Favoritar',
          icon: 'heart-outline',
          handler: () => {
            console.log('Clicou em Favoritar');
          }
        },
        {
          text: 'Cancelar',
          role: 'cancel',
          icon: 'close',
          handler: () => {
            console.log('Clicou em Cancelar');
          }
```

```
            }
          ]
        });

        actionSheet.present();
      }

}
```

Fim código 5

Quais foram as modificações de implementação realizadas no arquivo home.js?

1) O componente Action Sheets do Ionic foi adicionado na linha de import:

```
import { NavController, ActionSheetController } from 'ionic-angular';
```

2) O componente Action Sheets foi declarado no construtor das classes:

```
constructor(public navCtrl: NavController,
 public actionsheetCtrl: ActionSheetController) {

}
```

3) A implementação de código do método `mostrarComponente()` , que inclui o uso do componente Action Sheets, sua formatação e acionamento.

Observe o comando `actionSheet.present()`, logo no final do método `mostrarComponente()`, essa é a linha de comando que realmente aciona e mostra o componente. O restante do código no método serve para configurar a forma visual de apresentação do componente Action Sheets.

Após realizar as alterações, salve todos os arquivos alterados da aplicação. Faça um teste, clicando no botão “Mostrar a folha de ação”. O resultado deve ser como mostrado na Figura 12. Escolha uma das opções e clique na folha de ação.

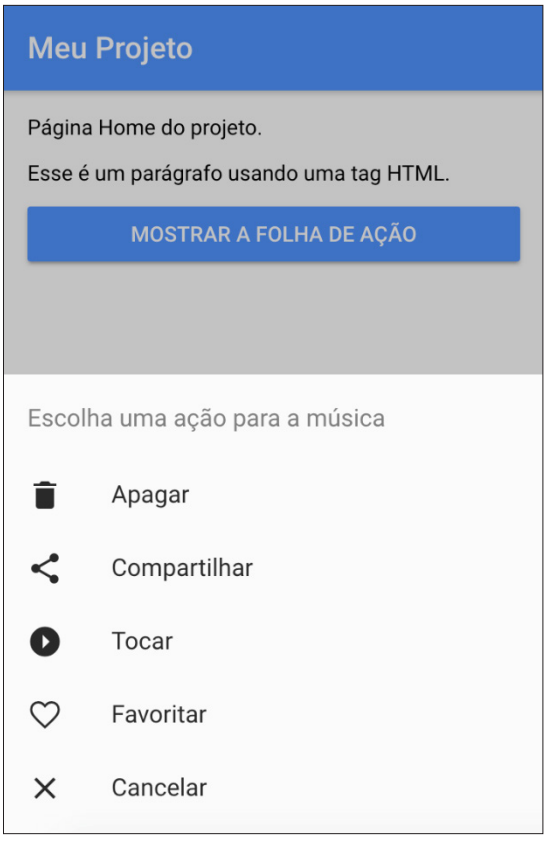


Figura 12 *Página home.html alterada*.

Os Action Sheets deslizam para cima a partir da borda inferior da tela do dispositivo e exibem um conjunto de opções com a capacidade de confirmar ou cancelar uma ação. As folhas de ação às vezes podem ser usadas como uma alternativa aos menus, no entanto, elas não devem ser usadas para navegação.

Um Action Sheets sempre aparece acima de qualquer outro componente na página e deve ser descartado para interagir com o conteúdo subjacente. Quando é acionado, o resto da página escurece para dar mais foco às opções da Folha de Ação.

Essas foram algumas alterações para iniciar e ajudar a entender a utilização da plataforma de desenvolvimento do *framework* Ionic; outros componentes serão mostrados mais adiante.

Vale a pena lembrar que nosso primeiro projeto utilizou o template “blank”, porém outros modelos de templates de aplicação Ionic estão disponíveis para a criação de projetos. Esse assunto será abordado mais adiante, em outra unidade.

Na próxima unidade, estudaremos a estrutura de diretórios do projeto Ionic e vários outros componentes de interface com o usuário. Como já foi mencionado, o *framework* Ionic possui um grande número de componentes para melhorar a experiência do usuário na interação com o aplicativo. Uma maneira de começar com o desenvolvimento na plataforma é conhecer bem esses componentes do *framework*.

7. QUESTÕES AUTOAVALIATIVAS

A autoavaliação pode ser uma ferramenta importante para você testar o seu desempenho. Se encontrar dificuldades em responder as questões a seguir, você deverá revisar os conteúdos estudados para sanar as suas dúvidas.

1) Entre os sensores disponíveis na infraestrutura de um dispositivo móvel, qual é capaz de medir a velocidade de movimento de um dispositivo ao qual está acoplado?
   a) Sensor de posicionamento global (GPS)
   b) Sensor de distância
   c) Acelerômetro
   d) Giroscópio
   e) Compass

2) Entre as dependências do ambiente do framework Ionic, qual é responsável por fornecer uma plataforma para implementar aplicações server-side de sistemas distribuídos?
   a) NPM
   b) Apache Cordova
   c) WebView
   d) NodeJS
   e) ECMA Script

Gabarito

Confira, a seguir, as respostas corretas para as questões autoavaliativas propostas:

1) Alternativa C.

2) Alternativa D.

8. CONSIDERAÇÕES

Com o estudo desta unidade, você teve a oportunidade de compreender as características de *hardware* particulares dos dispositivos móveis, os conceitos de instalação e a utilização da plataforma de desenvolvimento Ionic.

Também teve a oportunidade de conhecer as principais tecnologias utilizadas para o desenvolvimento de aplicações híbridas em dispositivos móveis, que são NodeJS, Apache Cordova, Typescript e outras já conhecidas como HTML, Javascript e CSS.

No decorrer desta unidade, utilizamos a plataforma Ionic com a implementação de um exemplo de aplicação. O objetivo desse primeiro exemplo foi introduzi-lo nas práticas de desenvolvimento e nos primeiros conceitos que envolvem a utilização de componentes da plataforma.

Assim, é fundamental que você compreenda que no desenvolvedor de *software* para dispositivos móveis, além de conhecer bem as linguagens de programação, você deve conhecer bem sua plataforma de trabalho, assim poderá analisar as melhores estratégias de desenvolvimento, desenvolver estudos e desenhar a solução mais adequada ao domínio do problema.

Esperamos que você aprofunde seus estudos com as leituras indicadas sobre esses assuntos. Conhecer as fontes do conhecimento relacionadas ao desenvolvimento de *software* para dispositivos móveis pode ajudá-lo futuramente no desempenho de suas atividades.

9. E-REFERÊNCIAS

Lista de figuras

Figura 2 *Arquitetura utilizando o framework Cordova*. Disponível em: <https://cordova.apache.org/static/img/guide/cordovaapparchitecture.png>. Acesso em: 07 ago. 2018.

Sites pesquisados

Accelerometer. In: *Wikipédia*: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Accelerometer>. Acesso em: 10 ago. 2018.

ANDROID Developers "Data Storage". Site do desenvolvedor Android. Disponível em: <https://developer.android.com/guide/topics/data/data-storage.html>. Acesso em: 10 ago. 2018.

ANDROID DEVELOPER. Site do desenvolvedor da plataforma Android. Disponível em: <https://developer.android.com/guide/topics/sensors/sensors_overview.html>. Acesso em: 10 ago. 2018.

APACHE CORDOVA, site do desenvolvedor. Disponível em: <http://cordova.apache.org>. Acesso em: 10 ago. 2018.

APACHE CORDOVA, site do desenvolvedor. Disponível em: <https://cordova.apache.org/docs/en/latest/guide/overview/>. Acesso em: 10 ago. 2018.

BAROMETER. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Barometer>. Acesso em: 20 nov. 2017.

BOOTSTRAP. Site do framework para conteúdo web. Disponível em: <http://getbootstrap.com/>. Acesso em: 20 nov. 2017.

GIROSCOPE. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Gyroscope>. Acesso em: 20 nov. 2017.

IONIC Framework, Getting Started. Disponível em: <https://ionicframework.com/docs/components/>. Acesso em: 10 ago. 2018.

NODEJS, JavaScript Runtime Site. Disponível em: <https://nodejs.org/>. Acesso em: 10 ago. 2018.

SASS. Site da linguagem de extensão CSS. Disponível em: <http://sass-lang.com/>. Acesso em: 10 ago. 2018.

TECNOLOGIA MÓVEL. In: Wikipédia: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Tecnologia_m%C3%B3vel>. Acesso em: 20 nov. 2017.

TYPESCRIPTLANG. Site oficial da linguagem typescript. Disponível em: < http://www.typescriptlang.org/index.html>. Acesso em: 10 ago. 2018.

VISUAL STUDIO CODE. Site do editor. Disponível em: <https://code.visualstudio.com/>. Acesso em: 10 ago. 2018.

W3SCHOOLS. Tutoriais para estudo e treinamento CSS. Disponível em: <https://www.w3schools.com/css/default.asp>. Acesso em: 10 ago. 2018.

W3SCHOOLS. Tutoriais para estudo e treinamento HTML. Disponível em: <https://www.w3schools.com/html/default.asp>. Acesso em: 10 ago. 2018.

W3SCHOOLS. Tutoriais para estudo e treinamento HTML5 Webstorage. Disponível em: <https://www.w3schools.com/html/html5_webstorage.asp>. Acesso em: 10 ago. 2018.

W3SCHOOLS. Tutoriais para estudo e treinamento Javascript. Disponível em: <https://www.w3schools.com/js/default.asp>. Acesso em: 10 ago. 2018.

10. REFERÊNCIAS BIBLIOGRÁFICAS

LEE, V.; SCHNEIDER, H.; SCHELL, R. *Aplicações móveis*: arquitetura, projeto e desenvolvimento. Tradução Amaury Bentes e Deborah Rudger, São Paulo: Ed. Pearson Education do Brasil, 2005.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Addison-Wesley, 2011.

UNIDADE 2

COMPONENTES DO FRAMEWORK IONIC

Objetivos

- Criar e configurar projeto de aplicação com o *framework* Ionic.
- Desenvolver um projeto utilizando componentes da plataforma Ionic.
- Executar e testar o funcionamento do aplicativo desenvolvido.
- Conhecer a forma correta de se utilizar componentes da plataforma Ionic.
- Compreender os conceitos que estão incluídos nas práticas de utilização dos componentes da plataforma Ionic.

Conteúdos

- Estrutura de um projeto Ionic.
- Componentes da plataforma Ionic.
- Comandos utilizados no desenvolvimento de aplicativos Ionic.
- Práticas iniciais para o desenvolvimento na plataforma Ionic.
- Conceitos de navegação entre os componentes da plataforma Ionic.

Orientações para o estudo da unidade

Antes de iniciar o estudo desta unidade, é importante que você leia as orientações a seguir:

1) Certifique-se de que você instalou todos os componentes da plataforma Ionic conforme orientações da unidade anterior.

2) Acesse os links das referências sobre os componentes da plataforma Ionic e pesquise as instruções de utilização dos componentes que serão abordados nesta unidade.

3) Aprofunde seus conhecimentos com as leituras indicadas referentes à forma correta de implementação dos componentes da plataforma Ionic.

4) Nas instruções para criar e configurar novos projetos, siga exatamente as orientações, em especial a que se refere ao nome do projeto e de componentes, pois esses nomes são utilizados no código-fonte da aplicação.

1. INTRODUÇÃO

O objetivo do estudo desta segunda unidade é ajudar você a compreender os métodos e os conceitos que estão incluídos nas práticas de utilização dos componentes da plataforma Ionic. Conhecer a forma correta de utilização dos componentes, bem como sua distribuição de código nos componentes do padrão MVC, ajuda o profissional a se familiarizar com o desenvolvimento de aplicativos desde o início, melhorando, assim, a perspectiva inicial de produção de *software* e, consequentemente, antecipa o conhecimento que irá auxiliar em sua jornada de aprendizado.

Os objetivos a serem alcançados, nesta unidade, fundamentam-se na compreensão da função de cada componente da plataforma e sua utilidade no contexto de interação com o usuário da aplicação. É fundamental que você conheça a utilidade particular de cada componente.

2. COMPONENTES VISUAIS

Os aplicativos da plataforma Ionic são feitos com blocos de construção de alto nível chamados componentes. Esses componentes permitem que você construa rapidamente uma interface para interação com o usuário. Existem vários componentes disponíveis, dentre eles modais, *popups* e cartões.

Confira os exemplos a seguir para ver como cada componente se parece e para aprender a usar cada um deles. Quando você estiver familiarizado com o básico, dirija-se aos documentos da API para obter ideias de como personalizar cada componente.

Você deve estar lembrado de que na Unidade 1 criamos um novo projeto e abordamos a utilização dos componentes Botão de ação e *Action Sheets* (folhas de ação). O botão foi o componente usado para mostrar o *Action Sheets*. Como foi visto, por meio do evento (*click*), o botão aciona a apresentação das opções para o usuário escolher. O *Action Sheets* é útil quando queremos interagir com o usuário para que ele escolha uma opção de ação entre várias disponíveis em uma determinada situação.

No próximo exemplo de aplicação, criaremos um novo projeto para implementar e conhecer vários componentes da API Ionic, começando por um menu de navegação que aciona a transição entre as páginas da aplicação.

Vamos criar o novo projeto baseado no *template blank*. O primeiro passo é gerar a pasta com a estrutura do projeto. Para isso, vamos utilizar novamente o *prompt* de comando.

Abra o Terminal ou o *prompt* de comando e acesse a pasta de desenvolvimento dos projetos Ionic. O comando para isso é:

```
cd dev
```

Agora que você está dentro da pasta dev, vamos utilizar o Ionic CLI para criar um novo projeto. Para esse procedimento, seu computador deverá estar conectado à internet.

Para criar um novo projeto, digite o comando:

```
ionic start appmenu blank
```

Aguarde alguns minutos, a criação do projeto depende do *download* de vários arquivos, o Ionic faz tudo de forma automática. O comando `ionic start` é usado para criar o projeto, `appmenu` é o nome dado ao projeto e `blank` é o modelo (*template*) de projeto que deve ser criado.

Ainda dentro da pasta dev, acesse a pasta do novo projeto digitando o comando:

```
cd appmenu
```

Execute o seu projeto e aguarde alguns minutos, sua aplicação aparecerá na janela do navegador web. É importante saber que o próximo comando só funciona se você estiver dentro da pasta do projeto Ionic. O comando para executar a aplicação vem do Ionic CLI, digite da seguinte forma:

```
ionic serve
```

Aguarde alguns minutos, o Ionic realizará o *build* da aplicação, ou seja, o aplicativo será gerado por meio do projeto. O navegador padrão de seu sistema operacional será carregado e mostrará a tela inicial da aplicação de acordo com o *template blank*. A Figura 1 mostra a tela da aplicação.

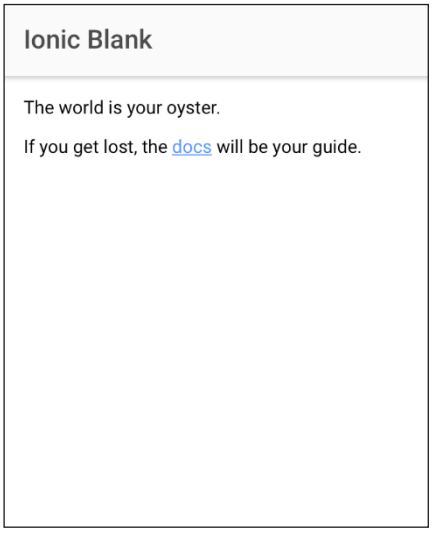


Figura 1 *Tela da aplicação appmenu*.

Agora, vamos gerar novas páginas, serão nove páginas ao todo. Para isso vamos utilizar novamente o *prompt* de comando. Verifique se está dentro da pasta do projeto e digite o comando a seguir para gerar a página “alerts” (use apenas letras minúsculas para o nome das páginas):

```
ionic g page alerts
```

Observe que dentro da pasta src/pages/ foi gerada uma nova pasta com o nome alerts, dentro desta página foram gerados alguns arquivos, entre eles o alerts.html (*view*), e o alerts.ts (controlador).

Repita o procedimento, digitando os comandos para criar as outras páginas.

```
ionic g page inputs
```

```
ionic g page lists

ionic g page checkbox

ionic g page cards

ionic g page range

ionic g page toast

ionic g page badges

ionic g page toggles
```

Agora que as páginas estão criadas, podemos iniciar os procedimentos de implementação dos componentes. Vamos começar pelo componente Menu.

Antes de iniciarmos a criação de páginas, para saber mais sobre os componentes acesse a documentação disponível em: <https://ionicframework.com/docs/components/>. Acesso em: 10 ago. 2018.

Menu

O componente menu é uma "gaveta" com opções de navegação que desliza a partir da lateral da tela. Por padrão, ele desliza para a esquerda, mas o lado pode ser configurado e deslizar para a direita. Com a aplicação rodando no navegador web, o menu pode ser acionado com a seta do *mouse* no canto esquerdo da tela, clicando e arrastando. No dispositivo móvel, o menu pode ser acionado arrastando o dedo no canto esquerdo da tela para "puxar" o menu.

O menu é uma forma de apresentar várias opções de escolha para o usuário realizar transições entre várias telas do aplicativo; ele oferece uma forma de "Navegar pela aplicação". Pode ser apresentado também na forma de um menu lateral e pode ser arrastado para abrir ou alternado para se mostrar ao usuário. O conteúdo de um menu será escondido quando o menu estiver fechado e o seu formato visual se adapta de acordo com o estilo base da plataforma nativa em uso.

Para implementar um menu, é necessário alterar o código dos arquivos app.module.ts, app.html e app.component.ts, esses arquivos estão localizados na pasta src/app/.

Abra o arquivo app.module.ts e declare a importação dos componentes de todas as páginas que foram criadas. Para esse procedimento, faça as alterações para que o código do arquivo fique como o Código 1, mostrado a seguir.

Código 1 Arquivo app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';

import { ErrorHandler, NgModule } from '@angular/core';

import { IonicApp, IonicErrorHandler, IonicModule } from 'ionic-
angular';
```

```
import { SplashScreen } from '@ionic-native/splash-screen';
import { StatusBar } from '@ionic-native/status-bar';

import { MyApp } from './app.component';
import { HomePage } from '../pages/home/home';
import { AlertsPage } from '../pages/alerts/alerts';
import { InputsPage } from '../pages/inputs/inputs';
import { ListsPage } from '../pages/lists/lists';
import { CheckboxPage } from '../pages/checkbox/checkbox';
import { CardsPage } from '../pages/cards/cards';
import { RangePage } from '../pages/range/range';
import { ToastPage } from '../pages/toast/toast';
import { BadgesPage } from '../pages/badges/badges';
import { TogglesPage } from '../pages/toggles/toggles';

@NgModule({
  declarations: [
    MyApp,
    HomePage,
    AlertsPage,
    InputsPage,
    ListsPage,
    CheckboxPage,
    CardsPage,
    RangePage,
    ToastPage,
    BadgesPage,
    TogglesPage
  ],
```

```
    imports: [
      BrowserModule,
      IonicModule.forRoot(MyApp)
    ],
    bootstrap: [IonicApp],
    entryComponents: [
      MyApp,
      HomePage,
      AlertsPage,
      InputsPage,
      ListsPage,
      CheckboxPage,
      CardsPage,
      RangePage,
      ToastPage,
      BadgesPage,
      TogglesPage
    ],
    providers: [
      StatusBar,
      SplashScreen,
      {provide: ErrorHandler, useClass: IonicErrorHandler}
    ]
  })
  export class AppModule {}
```

Fim código 1

Essa alteração ainda não é suficiente para que o menu funcione. Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Observe que as modificações de código, além da declaração de importação das páginas criadas, incluem também a inclusão delas nos arrays `declarations` e no array `entryComponents` do arquivo.

É importante compreender que uma nova página é considerada como um novo componente na aplicação e deve ser incluída no arquivo app.module.ts, na forma como foi mostrado no Código 1, para que a página possa ser utilizada.

A implementação deve ser feita no arquivo app.component.ts. Declare a importação dos componentes de todas as páginas que foram criadas. Declare a variável de array com o nome "pages" e dentro do construtor da classe preencha o array com os componentes das páginas. Implemente o código do método `openPage()`. Para esses procedimentos, faça as alterações para que o código do arquivo fique como o Código 2, mostrado a seguir.

Código 2 Arquivo app.component.ts

```
import { Component, ViewChild } from '@angular/core';

import { Platform, Nav } from 'ionic-angular';

import { StatusBar } from '@ionic-native/status-bar';

import { SplashScreen } from '@ionic-native/splash-screen';


import { HomePage } from '../pages/home/home';

import { AlertsPage } from '../pages/alerts/alerts';

import { InputsPage } from '../pages/inputs/inputs';

import { ListsPage } from '../pages/lists/lists';

import { CheckboxPage } from '../pages/checkbox/checkbox';

import { CardsPage } from '../pages/cards/cards';

import { RangePage } from '../pages/range/range';

import { ToastPage } from '../pages/toast/toast';

import { BadgesPage } from '../pages/badges/badges';

import { TogglesPage } from '../pages/toggles/toggles';


@Component({

  templateUrl: 'app.html'
```

```
})
export class MyApp {

  @ViewChild(Nav) nav: Nav;

  rootPage:any = HomePage;

  pages: Array<{title: string, component: any}>;

  constructor(platform: Platform, statusBar: StatusBar, splashScreen:
SplashScreen) {
    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might
need.
      statusBar.styleDefault();
      splashScreen.hide();
    });

    this.pages = [
      { title: 'Alerts', component: AlertsPage },
      { title: 'Inputs', component: InputsPage },
      { title: 'Lists', component: ListsPage },
      { title: 'CheckBox', component: CheckboxPage },
      { title: 'Cards', component: CardsPage },
      { title: 'Range', component: RangePage },
      { title: 'Toast', component: ToastPage },
      { title: 'Badges', component: BadgesPage },
      { title: 'Toggles', component: TogglesPage }
    ];

  }
```

```
    openPage(page: any): void{
      this.nav.push(page.component);
    }

}
```

Fim código 2

Essa alteração ainda não é suficiente para que o menu funcione, mas salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Observe, logo nas primeiras duas linhas do código 2, que também foram incluídas as importações dos componentes `ViewChild` e `Nav` da API Ionic. Esses componentes são utilizados para realizar a transição entre páginas no método `openPage()`.

As alterações de código nos componentes estão prontas, mas para o menu funcionar efetivamente é necessário fazer as alterações na página app.html. Modifique o arquivo para que fique como o Código 3, mostrado a seguir.

Código 3 Página app.html

```
<ion-menu [content]=”content”>
  <ion-header>
    <ion-toolbar>
      <ion-title>Menu</ion-title>
    </ion-toolbar>
  </ion-header>

  <ion-content>
    <ion-list>
      <button ion-item *ngFor=”let page of pages”
(click)=”openPage(page)” menuClose>
        {{page.title}}
      </button>
    </ion-list>
  </ion-content>
```

```
</ion-menu>

<ion-nav id="nav" #content [root]="rootPage"></ion-nav>
```

Fim código 3

Observe no código 3, que o menu começa na tag <ion-menu> e as opções do menu estão em uma lista começando pela tag <ion-list>. A lista é construída dinamicamente a partir do array pages que é percorrido iterativamente por meio da diretiva `*ngFor`. Dessa forma, cada linha do array pages se transforma em um item da lista. Cada item possui um título que será exibido nas opções do menu e um evento "click" que aciona o método openPage passando o nome da página que será aberta.

Abra o arquivo home.html e modifique-o, incluindo apenas o código realçado em azul da parte superior em destaque, conforme é mostrado no trecho de código a seguir.

```
<ion-header>
  <ion-navbar color="primary">
 <button ion-button menuToggle> <ion-icon name="menu"></ion-icon>
 </button>
 <ion-title>
       Exemplos Componentes
 </ion-title>
</ion-navbar>
</ion-header>
...
...
```

Após as modificações no código, salve o arquivo. Neste momento, é fundamental conferir se todos os arquivos alterados foram salvos. O Ionic realizará novamente o processo de *build* da aplicação, e a tela da sua página home.html deverá ser apresentada com o botão de menu logo acima no canto direito, conforme é mostrado na Figura 2. A funcionalidade de menu está concluída e podemos testá-la.

Figura 2 *Código da página home.html*.

Realize alguns testes de navegação clicando no botão menu e escolhendo algumas opções. O menu é apresentado conforme é mostrado na Figura 3.

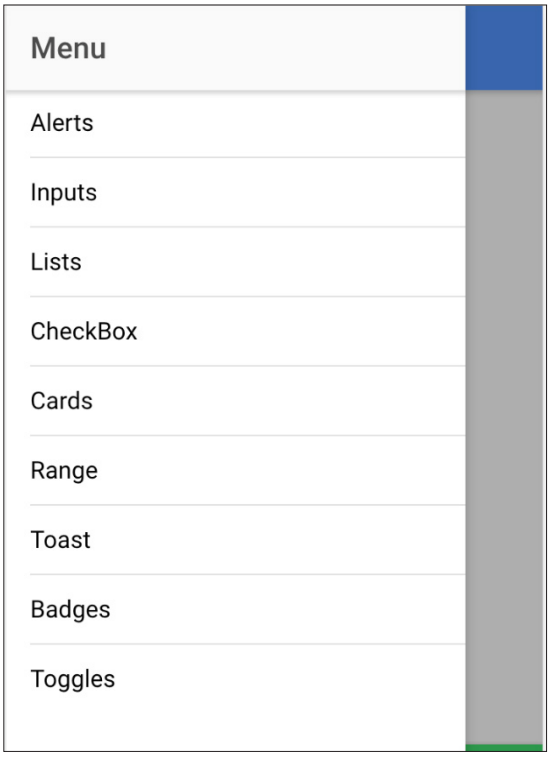


Figura 3 *Menu de opções da aplicação*.

Após a implementação e teste do componente menu, o próximo exemplo será o componente Alert.

Antes de dar continuidade aos seus estudos, para saber mais sobre o componente Menu, sugerimos que acesse a documentação do componente no *site* do Ionic Framework disponível em: <https://ionicframework.com/docs/components/#menus>. Acesso em: 21 ago. 2018.

Alerts

Alerts são uma ótima forma de oferecer ao usuário a capacidade de escolher uma ação específica. Podem ser utilizados para fornecer informações importantes ao usuário ou exigir que eles tomem uma decisão. Em uma perspectiva de interação com o usuário, os Alerts podem ser pensados como uma apresentação visual de um "quadro flutuante" que cobre apenas uma parte da tela. Isso significa que os Alerts só devem ser usados para ações rápidas, como verificação de senha, notificações de pequenas aplicações ou opções rápidas.

Feche todas as abas de arquivos abertos que estão no editor Visual Studio Code. Abra o arquivo alert.ts que está na pasta pages/alerts/. Faça as alterações para que fique como é mostrado no Código 4.

Código 4 Arquivo alert.ts**.**

```
import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams, AlertController } from
'ionic-angular';


@IonicPage()

@Component({

  selector: 'page-alerts',

  templateUrl: 'alerts.html',

})

export class AlertsPage {


  user: string = '';


  constructor(public navCtrl: NavController,

              public navParams: NavParams,

              public alertCtrl: AlertController) {
```

```
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad AlertsPage');
  }

  mostrarAlert() {
    let prompt = this.alertCtrl.create({
      title: 'Prompt',
      message: "Digite seu nome",
      inputs: [
        {
          name: 'nome',
          placeholder: 'Nome'
        },
      ],
      buttons: [
        {
          text: 'Cancelar',
          handler: data => {
            console.log('Clicou no Cancelar');
          }
        },
        {
          text: 'Entrar',
          handler: data => {
            this.user = 'Bem vindo ' + data.nome + '!';
            console.log('Clicou no Entrar');
          }
```

```
            }
          ]
        });
        prompt.present();
      }
  }
```

Fim código 4

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Após as alterações realizadas, você deve verificar se:

- Foi incluído, logo na segunda linha, a importação para o componente `AlertController`.
- O componente `AlertController` foi declarado no construtor da classe.
- Foi declarada a variável do tipo string, com o nome "user".
- Foi implementado o método `mostrarAlert()`.
- O componente Alert possui um campo de entrada e dois botões (Ok e Cancelar), o conteúdo do campo é transferido para a variável user, quando o usuário clicar no botão Ok. A transferência do valor é realizada pela linha: `this.user = 'Bem vindo ' + data.nome + '!';`.

Abra o arquivo alert.html que está na pasta pages/alerts/. Faça as alterações de maneira que fique como é mostrado no Código 5.

Código 5 Arquivo alert.html.

```
<ion-header>

  <ion-navbar>
    <ion-title>Componente Alert</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>

  <button ion-button color="dark" (click)="mostrarAlert()"
round>Mostrar Alert</button>
```

```
    <p> {{user}} </p>

</ion-content>
```

Fim código 5

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Após as alterações realizadas, observe que:

1) Foi incluído um botão com o evento click que aciona o método `mostrarAlert()`.
2) Foi incluída a varável `{{user}}` para mostrar dinamicamente o valor dessa varíavel na página alert.html.

O componente Alert é apresentado conforme é mostrado na Figura 4. Realize alguns testes clicando no botão “Mostrar Alert”. Digite seu nome no campo e clique no botão OK.

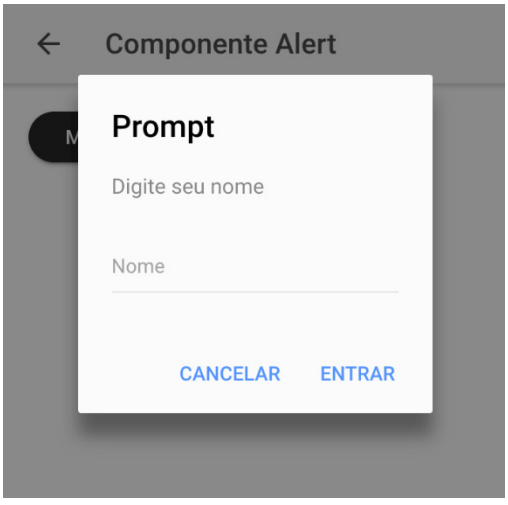


Figura 4 *O componente Alert*.

Para saber mais sobre o componente alert, acesse a documentação do componente no site do Ionic Framework. Disponível em: <https://ionicframework.com/docs/components/#alert>. Acesso em: 21 ago. 2018.

Na sequência, estudaremos o componente Input.

Inputs

Os inputs (Campo de entrada de dados) são essenciais para coletar e manipular a entrada de dados de forma segura. Eles devem ser apresentados de maneira intuitiva para os usuários interagirem. Cada campo de entrada em um formulário possui um controle, uma função que se liga ao valor no campo e executa a validação. Um grupo de campos é uma coleção de controles que manipulam o envio de formulários e fornecem uma ligação de alto nível que pode ser usada para determinar se o formulário inteiro é válido.

Vamos à implementação do componente Input. Feche todas as abas de arquivos abertos que estão no editor Visual Studio Code. Abra o arquivo inputs.ts que está na pasta pages/inputs/. Faça as alterações para que a implementação fique como é mostrado no Código 6.

Código 6 Arquivo inputs.ts.

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-angular';


@IonicPage()
@Component({
  selector: 'page-inputs',
  templateUrl: 'inputs.html',
})
export class InputsPage {

  login: string = '';
  senha: string = '';
  mensagem: string ='';

  constructor(public navCtrl: NavController, public navParams:
NavParams) {
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad InputsPage');
  }

  enviar(campoLogin, campoSenha){
    this.login = campoLogin;
    this.senha = campoSenha;
    this.mensagem = 'Seus dados foram enviados!';
  }

}
```

Fim código 6

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Observe as alterações realizadas:

1) Declaração das variáveis login, senha e mensagem.
2) Implementação do método enviar.

Abra o arquivo inputs.html e faça as alterações conforme o Código 7.

Código 7 Arquivo inputs.html.

```
    <ion-header>

      <ion-navbar>

        <ion-title>inputs</ion-title>

      </ion-navbar>

    </ion-header>



    <ion-content padding>

      <ion-list>

      <ion-item>

        <ion-input type=”text” placeholder=”Usuário”
[(ngModel)]=”campoLogin”></ion-input>

      </ion-item>

      <ion-item>

        <ion-input type=”password” placeholder=”Senha”
[(ngModel)]=”campoSenha”></ion-input>

      </ion-item>

      </ion-list>

      <button ion-button (click)=”enviar(campoLogin, campoSenha)”
outline>Enviar</button>

      <p>{{campoLogin}}</p>
```

```
    <p>{{mensagem}}</p>

</ion-content>
```

Fim código 7

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Verifique as alterações realizadas:

1) A tag <ion-input> implementa o campo de entrada de dados.
2) Foi utilizada a diretiva ngModel para fazer a transferência de valores das variáveis entre a View e o Controlador.
3) Para mostrar o conteúdo das variáveis na página foi usada a forma {{}}.

O componente input é apresentado conforme demonstra a Figura 5. Realize alguns testes, digite os dados nos campos e clique no botão “Enviar”.

Figura 5 *O componente Input*.

Antes de dar continuidade aos seus estudos, certifique-se de que entendeu como funciona o compomente input e obtenha mais informações sobre esse componente, acessando o site do Ionic Framework disponível em: <https://ionicframework.com/docs/components/#inputs>. Acesso em: 21 ago. 2018.

Lists

As lists são usadas para exibir linhas de informações, como lista de contatos, lista de reprodução ou menu de opções. As listas são os elementos mais populares de qualquer aplicação web ou móvel. Geralmente, são usadas para exibir várias informações separadas em linhas e podem ser combinadas com outros elementos HTML, como imagens, ícones etc., para criar diferentes apresentações. O framework Ionic oferece diferentes tipos de listas para facilitar seu uso.

Vamos à implementação do componente list. Para isso, feche todas as abas de arquivos abertos que estão no editor Visual Studio Code. Abra o arquivo lists.ts que está na pasta pages/lists/. Faça as alterações conforme é mostrado no Código 8.

Código 8 Arquivo lists.ts.

```
import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';


@IonicPage()

@Component({

  selector: 'page-lists',

  templateUrl: 'lists.html',

})

export class ListsPage {


  constructor(public navCtrl: NavController, public navParams:
NavParams) {

  }


  ionViewDidLoad() {

    console.log('ionViewDidLoad ListsPage');

  }


  enviar(item: string){

    console.log("Item Selecionado: ", item);

  }


}
```

Fim código 8

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação. Abra o arquivo lists.html e faça as alterações conforme demonstra o Código 9.

Código 9 Arquivo lists.html

```
<ion-header>
  <ion-navbar>
    <ion-title>lists</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>

  <ion-list>
    <ion-item (click)=”enviar(‘Saturno Planet’)”>
      <ion-icon name=’planet’></ion-icon>
      Saturno Planet
    </ion-item>

    <ion-item (click)=”enviar(‘Alarme’)”>
      <ion-icon name=’alarm’></ion-icon>
      Alarme
    </ion-item>

    <ion-item (click)=”enviar(‘Android’)”>
      <ion-icon name=’logo-android’></ion-icon>
      Android
    </ion-item>

    <ion-item (click)=”enviar(‘Apple’)”>
      <ion-icon name=’logo-apple’></ion-icon>
      Apple
```

```
      </ion-item>

      <ion-item (click)=”enviar(‘Contatos’)”>
        <ion-icon name=’contacts’></ion-icon>
        Contatos
      </ion-item>

      <ion-item (click)=”enviar(‘Bicicleta’)”>
        <ion-icon name=’bicycle’></ion-icon>
        Bicicleta
      </ion-item>
    </ion-list>

  </ion-content>
```

Fim código 9

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação. Neste exemplo, observe que as tags <ion-list> e <ion-item> são utilizadas para implementar o componente list. Repare no código que utilizamos um novo elemento visual, os ícones, que são apresentados pela tag <ion-icon>.

O componente list é apresentado conforme demonstrado na Figura 6. Realize alguns testes clicando nos itens da lista.

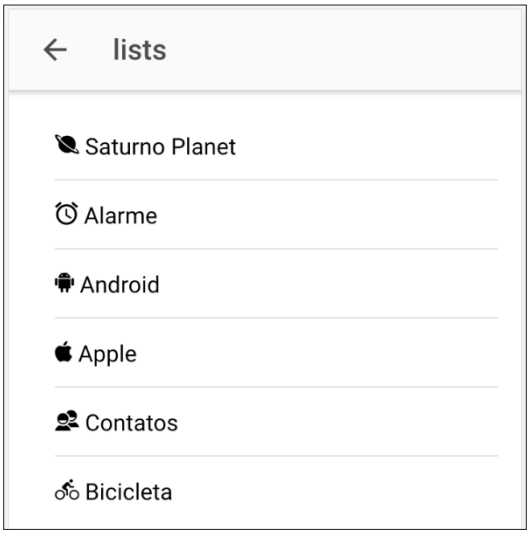


Figura 6 *O componente List*.

Para saber mais sobre o componente list, acesse a documentação do componente no site do Ionic Framework disponível em: <https://ionicframework.com/docs/components/#lists>. Acesso em: 21 ago. 2018.

Checkbox

Um checkbox (caixa de seleção) é um componente de entrada que contém um valor booleano. As caixas de seleção são iguais aos componentes de caixa de seleção da linguagem HTML. No entanto, a plataforma Ionic oferece várias formas de configurações para caixas de seleção diferentes. Um atributo pode ser utilizado para definir o valor padrão do checkbox e o atributo poderá ser ativado ou desativado pela ação do usuário quando interage com o checkbox.

Vamos à implementação do componente checkbox. Para tanto, feche todas as abas de arquivos abertos que estão no editor Visual Studio Code. Abra o arquivo checkbox.ts que está na pasta pages/checkbox/. Faça as alterações para que a implementação fique como é mostrado no Código 10.

Código 10 Arquivo checkbox.ts

```
import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';


@IonicPage()

@Component({

  selector: 'page-checkbox',

  templateUrl: 'checkbox.html',

})

export class CheckboxPage {


  azeitona: boolean;

  martini: boolean;

  manhattan: boolean;


  constructor(public navCtrl: NavController, public navParams:
NavParams) {

  }
```

```
  ionViewDidLoad() {
    console.log('ionViewDidLoad CheckboxPage');
  }

  updateAzeitona() {
    console.log('O item Azeitona foi atualizado:' + this.azeitona);
  }

  updateMartini() {
    console.log('O item Martini foi atualizado:' + this.martini);
  }

  updateManhattan() {
    console.log('O item Manhattan foi atualizado:' + this.manhattan);
  }

}
```

Fim código 10

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Em seguida, abra o arquivo checkbox.html e faça as alterações conforme o Código 11.

Código 11 Arquivo checkbox.html

```
<ion-header>

  <ion-navbar>
    <ion-title>checkbox</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>

  <ion-list>
```

```
        <ion-item>

          <ion-label>Azeitona</ion-label>

          <ion-checkbox [(ngModel)]=”azeitona” (ionChange)=”updateAzeito
na()”></ion-checkbox>

        </ion-item>


        <ion-item>

          <ion-label>Martini</ion-label>

          <ion-checkbox [(ngModel)]=”martini”
(ionChange)=”updateMartini()”></ion-checkbox>

        </ion-item>


        <ion-item>

          <ion-label>Manhattan</ion-label>

          <ion-checkbox [(ngModel)]=”manhattan” (ionChange)=”updateManhatt
an()”></ion-checkbox>

        </ion-item>


      </ion-list>


    </ion-content>
```

Fim código 11

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Observe as alterações realizadas:

1) A tag <ion-checkbox> é utilizada em conjunto com as tags <ion-list> e <ion-item> para implementar o componente checkbox.
2) Foi utilizada a diretiva ngModel para fazer a transferência de valores das variáveis entre a View e o Controlador.

O componente checkbox é apresentado conforme é mostrado na Figura 7. Realize alguns testes clicando nas caixas de checagem.

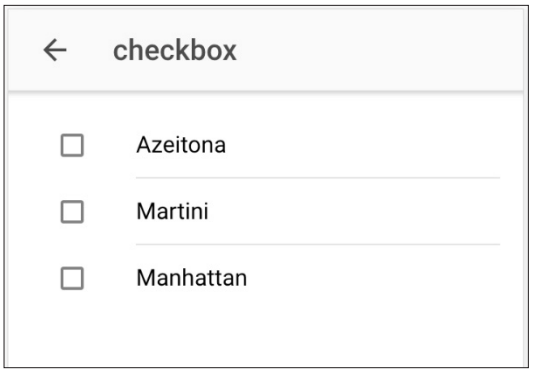


Figura 7 *O componente Checkbox*.

Caso você queira obter mais informações sobre o componente checkbox, acesse a documentação do componente no *site* do Ionic Framework. Disponível em: <https://ionicframework.com/docs/components/#checkbox>. Acesso em: 22 ago. 2018.

Cards

Os cards (cartões) são uma ótima forma de exibir partes importantes de conteúdo. Eles são relativamente mais novos que os outros componentes, mas estão emergindo rapidamente como um padrão de design para aplicativos. Além disso, eles possibilitam conter e organizar informações para destacar de forma visual e melhorar as expectativas do usuário.

Os cartões facilitam a exibição visual da mesma informação em vários tamanhos de tela diferentes quando utilizados em aplicativos móveis. Eles também permitem mais controle, são flexíveis e podem até ser animados. Os cartões, geralmente, são colocados uns sobre os outros, mas eles também podem ser usados como uma "página" e deslocados entre esquerda e direita na tela.

Para implementar o componente cards, você deverá fechar todas as abas de arquivos abertos no editor, abrir o arquivo cards.html que está na pasta pages/cards/ e fazer as alterações para que a implementação fique como é mostrado no Código 12.

Código 12 Arquivo cards.html

```
<ion-header>

  <ion-navbar>

    <ion-title>cards</ion-title>

  </ion-navbar>
```

```
    </ion-header>


    <ion-content padding>

      <ion-card>

      <ion-card-header>

        Pink Floyd

      </ion-card-header>

      <ion-card-content>

        Pink Floyd performed at the London Live 8 concert with Roger
Waters rejoining David Gilmour,

        Nick Mason and Richard Wright. It was the quartet's first
performance together in over 24 years

        — the band's last show with Waters was at Earls Court in London on
17 June 1981.

      </ion-card-content>

    </ion-card>


    </ion-content>
```

Fim código 12

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

As tags <ion-cards...> são utilizadas em conjunto para implementar o componente cards.

O componente cards é apresentado conforme demostra a Figura 8. Vale destacar que vários cards podem ser colocados na mesma página.

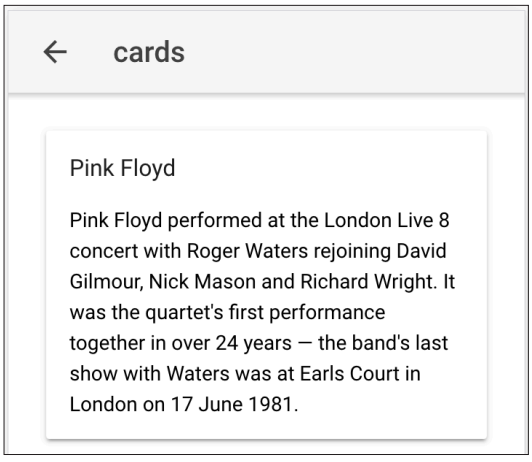


Figura 8 *O componente Cards.*

Antes de prosseguir seus estudos sobre o componente range, sugerimos que aprofunde seus conhecimentos referentes ao componente cards, acessando a documentação do componente no *site* do Ionic Framework. Disponível em: <https://ionicframework.com/docs/components/#cards>. Acesso em: 22 ago. 2018.

Range

O range (alcance) é um componente que permite aos usuários selecionar diversos valores movendo um botão deslizante ao longo da barra ou faixa. Além disso, ele permite o uso de etiquetas de escala colocadas em ambos os lados do intervalo que adicionam propriedades de início e fim ao intervalo. Valores mínimo e máximo também podem ser passados para o intervalo por meio das propriedades min e max, respectivamente. Por padrão, o intervalo define o valor mínimo para 0 e o máximo para 100.

Para implementar o componente range, você deverá fechar todas as abas de arquivos abertos, abrir o arquivo range.html e fazer as alterações para que a implementação fique como é mostrado no Código 13.

Código 13 Arquivo range.html

```
<ion-header>

  <ion-navbar>

    <ion-title>range</ion-title>

  </ion-navbar>

</ion-header>
```

```
    <ion-content padding>

      <ion-list>
        <ion-list-header>
          Ajuste os controles
        </ion-list-header>
        <ion-item>
          <ion-range [(ngModel)]=”brightness”>
            <ion-icon range-left small name=”sunny”></ion-icon>
            <ion-icon range-right name=”sunny”></ion-icon>
          </ion-range>
        </ion-item>
        <ion-item>
          <ion-range min=”-200” max=”200” pin=”true” [(ngModel)]=”contrast”
color=”secondary”>
            <ion-icon range-left small name=”contrast”></ion-icon>
            <ion-icon range-right name=”contrast”></ion-icon>
          </ion-range>
        </ion-item>
        <ion-item>
          <ion-range dualKnobs=”false” pin=”true” [(ngModel)]=”structure”
color=”dark”>
            <ion-icon range-left small name=”brush”></ion-icon>
            <ion-icon range-right name=”brush”></ion-icon>
          </ion-range>
        </ion-item>
        <ion-item>
          <ion-range min=”1000” max=”2000” step=”100” snaps=”true”
[(ngModel)]=”warmth” color=”danger”>
            <ion-icon range-left small color=”danger” name=”thermometer”></
ion-icon>
```

```
            <ion-icon range-right color="danger" name="thermometer"></ion-
icon>

          </ion-range>

        </ion-item>

    </ion-list>


    </ion-content>
```

Fim código 13

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Verifique as alterações realizadas:

1) A tag <ion-range> é utilizada em conjunto com as tags <ion-list> e <ion-item> para implementar o componente range.
2) Foi utilizada a diretiva ngModel para fazer a transferência de valores das variáveis entre a View e o Controlador.

A Figura 9 apresenta o componente range. Realize alguns testes deslizando os botões pelas barras.

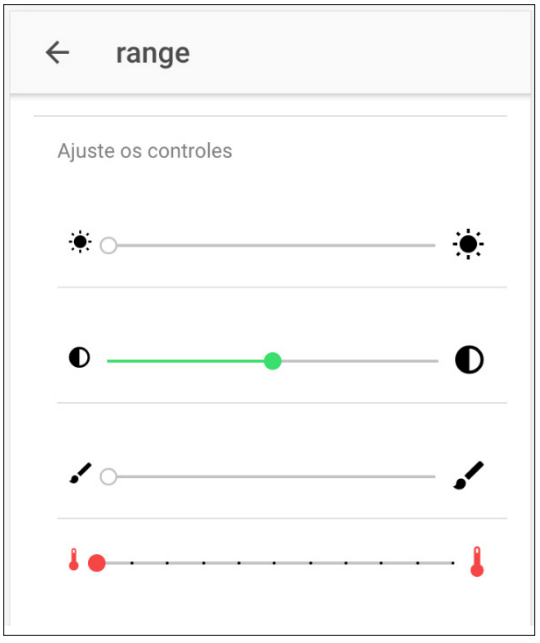


Figura 9 *O componente Range.*

Para saber mais sobre o componente range, acesse a documentação do componente no *site* do Ionic Framework. Disponível em: <https://ionicframework.com/docs/components/#range>. Acesso em: 22 ago. 2018.

Toast

Um toast (torrada) é uma notificação sutil comumente usada como mensagem de interação com o usuário. Pode ser usado para fornecer comentários sobre uma operação ou para exibir uma mensagem do sistema. A mensagem aparece sobre o conteúdo do aplicativo e pode ser fechada pelo aplicativo para retomar a interação do usuário com a tela. A mensagem a ser exibida deve ser passada na propriedade message. A opção showCloseButton pode ser definida como verdadeira para exibir um botão de fechamento no Toast.

Vamos à implementação do componente toast. Feche todas as abas de arquivos abertos no editor. Abra o arquivo toast.ts e faça as alterações para que a implementação fique como é mostrado no Código 14.

Código 14 Código do arquivo toast.ts

```
    import { Component } from '@angular/core';

    import { IonicPage, NavController, NavParams, ToastController } from
'ionic-angular';


    @IonicPage()

    @Component({

      selector: 'page-toast',

      templateUrl: 'toast.html',

    })

    export class ToastPage {


      constructor(public navCtrl: NavController,

                  public navParams: NavParams,

                  public toastCtrl: ToastController) {

      }


      ionViewDidLoad() {

        console.log('ionViewDidLoad ToastPage');

      }


      showToast(position: string) {

        let toast = this.toastCtrl.create({
```

```
            message: 'Mmmm, buttered toast',
            duration: 2000,
            position: position
          });

          toast.present(toast);
        }

        showToastWithCloseButton() {
          const toast = this.toastCtrl.create({
            message: 'Your files were successfully saved',
            showCloseButton: true,
            closeButtonText: 'Ok'
          });
          toast.present();
        }

        showLongToast() {
          let toast = this.toastCtrl.create({
            message: 'Lorem ipsum dolor sit amet, consectetur adipisicing
elit. Ea voluptatibus quibusdam eum nihil optio, ullam accusamus magni,
nobis suscipit reprehenderit, sequi quam amet impedit. Accusamus dolorem
voluptates laborum dolor obcaecati.',
            duration: 2000,
          });
          toast.present();
      }

      }
```

Fim código 14

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Abra o arquivo toast.html e faça as alterações conforme o Código 15.

Código 15 Código do arquivo toast.html

```
<ion-header>

  <ion-navbar>

    <ion-title>toast</ion-title>

  </ion-navbar>

</ion-header>

<ion-content padding>

  <button ion-button block (click)=”showToast(‘top’)”>Show Toast Top
Position</button>

  <button ion-button block (click)=”showToast(‘middle’)”>Show Toast
Middle Position</button>

  <button ion-button block (click)=”showToast(‘bottom’)”>Show Toast
Bottom Position</button>

  <button ion-button block (click)=”showLongToast()”>Show Long Toast</
button>

  <button ion-button block (click)=”showToastWithCloseButton()”>Show
Toast W/ Close Button</button>

</ion-content>
```

Fim código 15

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Observe as alterações realizadas:

1) A importação do componente ToastController, incluindo sua declaração no construtor da classe.
2) A implementação dos métodos showToast, por exemplo o `showToast(position: string)`.
3) O componente Button aciona a apresentação do Toast por meio do evento “click”.

Verifique, na Figura 10, como o componente toast é apresentado. Realize alguns testes clicando nos botões.

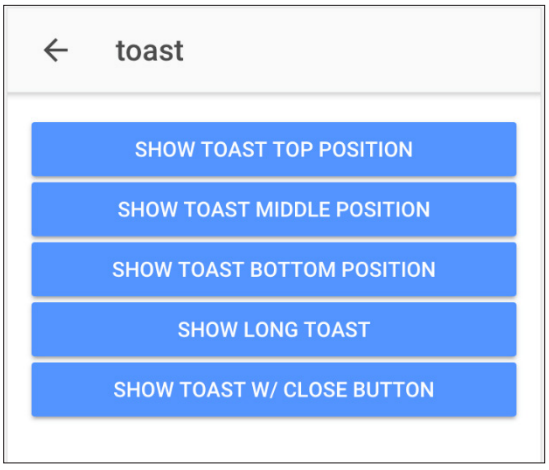


Figura 10 *O componente Toast*.

Antes de continuar seus estudos, sugerimos que aprofunde seus conhecimentos sobre o componente toast, acessando a documentação do componente no *site* do Ionic Framework. Disponível em: <https://ionicframework.com/docs/components/#toast>. Acesso em: 22 ago. 2018.

Badge

O badge (distintivo) é um componente simples que apresenta números, texto ou ícones em um formato visual distintivo. São componentes pequenos que se destacam na tela do aplicativo e tipicamente comunicam um valor visual ao usuário. Um badge geralmente é usado para exibir uma combinação de imagem, textos e números associados a um item que está em destaque na tela.

Vamos à implementação do componente badge. Para tanto, feche todas as abas de arquivos abertos no editor. Abra o arquivo badges.html e faça as alterações para que a implementação fique como é mostrado no Código 16.

Código 16 Arquivo badge.html

```
<ion-header>

  <ion-navbar>

    <ion-title>badges</ion-title>

  </ion-navbar>

</ion-header>


<ion-content padding>

  <ion-card>
```

```
        <ion-card-content>
          <ion-card-title>
            Claude Debussy
          </ion-card-title>
          <p>
            From the 1890s Debussy began to develop his own musical
language,
            which was largely independent of Wagner’s style, coloured in
part from the dreamy,
            sometimes morbid romanticism of the Symbolist movement.
          </p>
        </ion-card-content>

        <ion-item>
          <ion-icon name=’musical-notes’ item-start style=”color:
#d03e84”></ion-icon>
          Albums
          <ion-badge item-end>9</ion-badge>
        </ion-item>

        <ion-item>
          <ion-icon name=’logo-twitter’ item-start style=”color:
#55acee”></ion-icon>
          Followers
          <ion-badge item-end>260k</ion-badge>
        </ion-item>

    </ion-card>

    </ion-content>
```

Fim código 16

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

Observe as alterações realizadas:

- A tag <ion-badge> é utilizada em conjunto com as tags <ion-card> e <ion-item> para implementar o componente badge.

A Figura 11 demonstra como o componente badge é apresentado. Você também deverá saber que é possível incluir vários badges na mesma página.

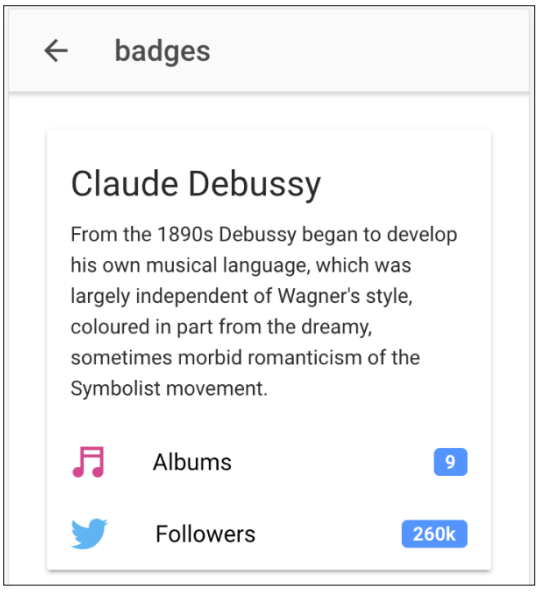


Figura 11 *O componente Badge*.

Acesse a documentação do componente badge no *site* do Ionic Framework e obtenha mais informações sobre esse componente. Disponível em: <https://ionicframework.com/docs/components/#badges>. Acesso em: 22 ago. 2018.

Toggle

O toggle (alternar) é um componente de entrada a ser utilizado com a função booleana de liga ou desliga. Como o componente de caixa de seleção, as ações de alternância entre valores são frequentemente usadas para permitir ao usuário ativar ou desativar uma configuração. Atributos com as propriedades de desabilitado ou verificado também podem ser aplicados ao toggle para controlar seu comportamento.

Para implementar o componente toggle, você deverá fechar todas as abas de arquivos abertos no editor. Abrir o arquivo toggles.html e fazer as alterações para que a implementação fique como é mostrado no Código 17.

Código 17 Código do arquivo toggles.html

```
<ion-header>

  <ion-navbar>
    <ion-title>toggles</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>

  <ion-list>
    <ion-list-header>
      Configuração
    </ion-list-header>

    <ion-item>
      <ion-label>Rodas de liga leve</ion-label>
      <ion-toggle value=”foo” checked=”true”></ion-toggle>
    </ion-item>

    <ion-item>
      <ion-label>Teto Solar</ion-label>
      <ion-toggle color=”energized”></ion-toggle>
    </ion-item>

    <ion-item>
      <ion-label>Freios ABS</ion-label>
      <ion-toggle color=”danger” checked=”true”></ion-toggle>
    </ion-item>

    <ion-item>
```

```
        <ion-label>Airbag Cortina</ion-label>
        <ion-toggle color="royal" checked="true"></ion-toggle>
      </ion-item>

      <ion-item>
        <ion-label>Frigobar</ion-label>
        <ion-toggle color="danger"></ion-toggle>
      </ion-item>

  </ion-list>

  </ion-content>
```

Fim código 17

Salve o arquivo, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação.

A tag <ion-toggle> é utilizada em conjunto com as tags <ion-list>, <ion-label> e <ion-item> para implementar o componente toggle.

A Figura 12 apresenta o componente toggle. Oserve a figura e realize alguns testes clicando nos botões.

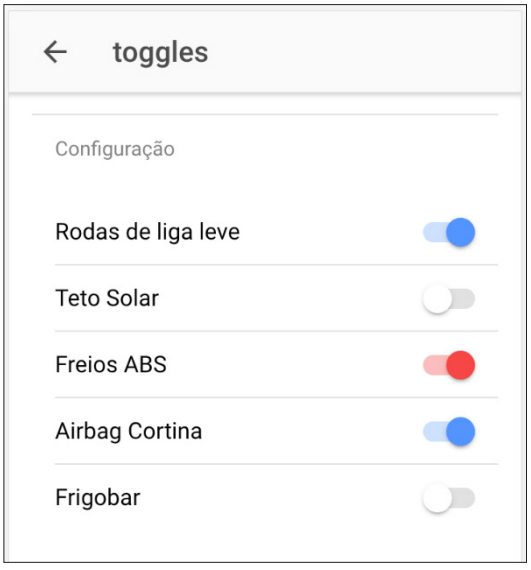


Figura 12 *O componente Toggle*.

Acesse a documentação do componente no *site* do Ionic Framework para saber mais sobre:

- o componente Toggle. Disponível em: <https://ionicframework.com/docs/components/#toggle>. Acesso em: 22 ago. 2018.a API de componentes. Disponível em: <https://ionicframework.com/docs/api/>. Acesso em: 22 ago. 2018.

Navigation

Navigation não é um componente propriamente dito, mas é a forma como os usuários se movem entre diferentes páginas em um aplicativo.

A navegação na plataforma Ionic pode ser realizada de várias formas e com um conjunto de componentes. Para habilitar a navegação, a API Ionic fornece a classe NavController para ser instanciada e servir como componente base para controlar a navegação no aplicativo.

O NavController tem a funcionalidade de controlador de navegação. Basicamente, um controlador de navegação é um vetor de páginas que representam um histórico de navegação em particular. Esse vetor pode ser manipulado para navegar em todo o aplicativo, pressionando e exibindo páginas ou inserindo e removendo-as no vetor do histórico. A página atual, ou seja, aquela que está sendo exibida no momento, é a última que está no topo do vetor. Podemos comparar o vetor a uma estrutura de dados do tipo pilha, a página a ser exibida é colocada no topo da pilha; para voltar à página anterior, basta remover a página atual que está no topo.

Já vimos anteriormente como navegar pela aplicação utilizando um menu de opções que realiza a transição para várias páginas no aplicativo. Agora, vamos implementar um exemplo de navegação utilizando a classe NavController e também as diretivas navPush e navPop.

Conforme exemplos anteriores, no prompt de comando, crie uma nova página com o nome “principal”. Abra o arquivo app.module.ts, faça a importação da página principal e declare a página nos arrays `declarations` e `entryComponents`. Em seguida, abra o arquivo home.ts e faça as modificações de acordo com o Código 18.

Código 18 Arquivo home.ts

```
import { Component } from '@angular/core';

import { NavController, ActionSheetController } from 'ionic-angular';

import { PrincipalPage } from '../principal/principal';


@Component({

  selector: 'page-home',

  templateUrl: 'home.html'

})

export class HomePage {
```

```
  params: Object;
  pushPage: any;

  constructor(public navCtrl: NavController,
              public actionsheetCtrl: ActionSheetController) {

    this.pushPage = PrincipalPage;

  }

  novaPagina(){
    this.navCtrl.push(PrincipalPage);
  }

}
```

Fim código 18

Abra o arquivo home.html e faça as alterações conforme o Código 19.

Código 19 Código do arquivo home.html

```
<ion-header>
  <ion-navbar color="primary">

    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>

    <ion-title>
      Componentes
    </ion-title>
  </ion-navbar>
</ion-header>
```

```
    <ion-content padding>
      Página Home do projeto.
      <p>Teste de Navegação</p>
      <br>
      <button ion-button [navPush]=”pushPage”
[navParams]=”params”>Principal com navPush</button>
      <br>
      <button ion-button color=”dark” (click)=”novaPagina()”>Principal com
NavController</button>
    </ion-content>

    <ion-footer>
      <ion-toolbar color=”secondary”>
        <ion-title>Rodapé</ion-title>
      </ion-toolbar>
    </ion-footer>
```

Fim código 19

Abra o arquivo principal.ts e faça as alterações conforme o Código 20.

Código 20 Código do arquivo principal.ts

```
    import { Component } from ‘@angular/core’;
    import { IonicPage, NavController, NavParams } from ‘ionic-angular’;

    @IonicPage()
    @Component({
      selector: ‘page-principal’,
      templateUrl: ‘principal.html’,
    })
    export class PrincipalPage {

      constructor(public navCtrl: NavController, public navParams:
NavParams) {
```

```
    }

    ionViewDidLoad() {
      console.log('ionViewDidLoad PrincipalPage');
    }

    voltar(){
      this.navCtrl.pop();
    }

}
```

Fim código 20

Abra o arquivo principal.html e faça as alterações conforme o Código 21.

Código 21 Código do arquivo principal.html

```
<ion-header>

  <ion-navbar>
    <ion-title>principal</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>

  <button ion-button color="secondary" round navPop>Voltar com
navPop</button>
  <br>
  <button ion-button color="danger" (click)="voltar()">Voltar com
NavController</button>

</ion-content>
```

Fim código 21

Salve os arquivos, aguarde o *rebuild* da aplicação e veja no navegador se a aplicação está funcionando sem erros de compilação. Após as alterações no código, a página home.html é apresentada conforme demonstrado na Figura 13.

Realize alguns testes clicando nos botões de navegação, verifique se as regras de navegação estão funcionando e estude as alterações que foram realizadas no código.

Figura 13 *Página home.html com os botões de navegação.*

- Sugerimos que acesse o *link* a seguir para aprofundar seus estudos referentes ao Navigation. Disponível em: <https://ionicframework.com/docs/components/#navigation>. Acesso em: 22 ago. 2018.

Para finalizar seus estudos referentes aos componentes, sugerimos que faça novos estudos de acordo com as instruções da documentação.

3. QUESTÕES AUTOAVALIATIVAS

Sugerimos que você faça as questões autoavaliativas propostas a seguir para verificar o seu aprendizado.

1) Entre os componentes do framework Ionic, qual é útil quando queremos interagir com o usuário para que ele escolha uma opção de ação entre várias disponíveis em uma determinada situação.
   a) Listas
   b) Badges
   c) Toasts
   d) Action Sheets
   e) Alerts

2) Qual das alternativas apresenta o comando CLI que tem como efeito executar o projeto Ionic.
   a) Ionic start meuprojeto
   b) Ionic serve
   c) Ionic generate meuprojeto
   d) Ionic -g meuprojeto
   e) Ionic md meuprojeto

Gabarito

Confira, a seguir, as respostas corretas para as questões autoavaliativas propostas:

1) Alternativa D.

2) Alternativa b.

4. CONSIDERAÇÕES

Com o estudo desta unidade você teve a oportunidade de conhecer e implementar vários componentes de API do framework Ionic, além disso pôde utilizar e testar as funcionalidades de cada um deles.

Também teve a oportunidade de conhecer a estrutura do projeto Ionic, alguns comandos da linguagem Typescript e outras já conhecidas como HTML, Javascript.

Esperamos que você aprofunde seus conhecimentos com as leituras indicadas sobre esses assuntos. Conhecer bem os componentes do framework Ionic e saber utilizá-los no desenvolvimento de *software* para dispositivos móveis pode ajudá-lo futuramente no desempenho de suas atividades.

5. E-REFERÊNCIAS

IONIC Framework, Componentes. Site do desenvolvedor, documentão técnica. Disponível em: <https://ionicframework.com/docs/components/>. Acesso em: 22 ago. 2018.

IONIC Framework, API. Site do desenvolvedor, documentão técnica. Disponível em: < https://ionicframework.com/docs/api/>. Acesso em: 22 ago. 2018.

UNIDADE 3

DESENVOLVIMENTO DE APLICAÇÃO COM GERENCIAMENTO DE DADOS

Objetivos

- Criar e configurar projeto com a utilização de plugin para armazenamento de dados.
- Preparar o projeto para dar suporte a aplicações nativas.
- Compreender conceitos de armazenamento de dados em dispositivos móveis.
- Executar e testar o aplicativo.
- Gerar e construir o aplicativo para ser instalado na plataforma nativa Android.

Conteúdos

- Preparação de projetos para trabalhar com banco de dados.
- Armazenamento e gerenciamento de dados.
- Banco de dados SQLite.
- Componentes de entrada de dados.

Orientações para o estudo da unidade

Antes de iniciar o estudo desta unidade, é importante que você leia as orientações a seguir:

1) É importante revisar os conceitos de bancos de dados SQL, assim como os comandos da linguagem SQL utilizados para essa finalidade; acesse os *sites* indicados no tópico e-referências.

2) Nas instruções para criar e configurar novos projetos, siga exatamente as orientações, em especial a que se refere ao nome do projeto e de componentes, pois esses nomes são utilizados no código-fonte da aplicação.

3) Nas atividades de codificação, altere um arquivo de cada vez e acompanhe sempre no *browser* o funcionamento correto de sua aplicação.

4) Procure corrigir erros de código logo no início, evite acúmulo de linhas com erros.

5) Ao concluir a codificação da aplicação, instale um dos SDKs, Android ou iOS para poder realizar a geração do aplicativo e testar diretamente no dispositivo.

1. INTRODUÇÃO

O objetivo do estudo desta terceira unidade é estabelecer os conceitos e as práticas relacionadas ao armazenamento e ao gerenciamento de dados em aplicações para dispositivos móveis, além disso, no decorrer deste estudo, você terá a oportunidade de compreender as práticas e os métodos utilizados quando os requisitos desejados pelo usuário se referem a aplicativos que utilizam armazenamento e recuperação de dados. E terá, também, a oportunidade de conhecer quais são os componentes do framework Ionic utilizados para o gerenciamento de dados, incluindo sua forma correta de utilização.

Os objetivos a serem alcançados, nesta unidade, fundamentam-se na compreensão das técnicas de armazenamento interno e externo ao dispositivo móvel.

2. O ARMAZENAMENTO DE DADOS EM APLICAÇÕES MÓVEIS

Ao construir aplicativos móveis, muitas vezes você precisará considerar a funcionalidade de armazenar os dados utilizados na aplicação de forma persistente, isso significa que de alguma forma você precisará de recursos de bancos de dados. Para isso, existem duas formas, uma que é armazenar no próprio dispositivo e outra que é armazenar online utilizando serviços na “nuvem” por meio de um servidor. Qualquer que seja a forma escolhida, ela terá de ser bem planejada, pois a escolha irá afetar diretamente a forma como sua aplicação vai funcionar para atender aos requisitos desejados. Em muitos casos, pode ser necessário ter os dois tipos de armazenamento, interno do lado do cliente e o armazenamento externo do lado do servidor, e seu aplicativo deverá gerenciar o fluxo de dados entre as duas fontes.

A solução escolhida depende das necessidades específicas das funcionalidades da aplicação desejada pelo usuário, incluindo a preocupação se os dados devem ser privados para o aplicativo ou acessíveis para outros aplicativos, e quanto espaço é necessário para o armazenamento dos dados.

Armazenamento Interno

O armazenamento interno refere-se a armazenar dados no próprio dispositivo, ou seja, utilizar a infraestrutura interna do dispositivo e os recursos de armazenamento instalados no sistema operacional. Esse tipo de armazenamento permite que seus aplicativos funcionem efetivamente quando não há conexão, além de oferecer funcionalidades mais avançadas e, muitas vezes, mais rápidas do que o armazenamento no lado do servidor.

O armazenamento interno pode ser realizado também por meio de arquivos em cache temporário. Um cache é um armazenamento temporário que pode ser mantido para armazenar dados e transações recentemente acessados pela aplicação evitando que sejam perdidos devido a falha de conexão.

Por padrão, os arquivos salvos no armazenamento interno de cache são privados do aplicativo e não podem ser acessados por outros aplicativos (nem pelo usuário do dispositivo). Quando o usuário desinstalar um aplicativo, esses arquivos serão removidos. Quando houver pouco espaço de armazenamento interno no dispositivo, o sistema operacional poderá excluir esses arquivos de cache para liberar espaço. No entanto, não dependa do sistema

para limpar esses arquivos. Mantenha sempre você mesmo os arquivos de cache e respeite um limite razoável de espaço consumido (ANDROID Developers “Data-Storage”, 2017). Para preservar o espaço de arquivos e manter o desempenho do aplicativo, é importante gerenciar cuidadosamente os arquivos de cache durante todo o ciclo de vida do aplicativo, para isso é recomendado ir removendo os que não forem mais necessários.

O armazenamento interno pode ainda ser realizado utilizando armazenamento local (*local storage*). O armazenamento local é um padrão HTML5 que oferece armazenamento de dados offline persistente. Ele é seguro por origem (domínio e protocolo), os dados nunca são transferidos para o servidor e grandes quantidades de dados podem ser armazenadas localmente (5MB), sem afetar o desempenho da aplicação (W3SCHOOLS “HTML5 Webstorage”, 2017). É muito fácil de usar e permite que qualquer tipo de dados, incluindo objetos e *arrays*, seja rapidamente armazenado no dispositivo móvel.

Imagine uma situação em que seja necessário implementar uma aplicação simples como um cadastro com nomes de contatos de clientes. Nesse caso, pode-se usar uma abordagem simples de armazenamento local. Uma boa prática muito comum para esse tipo de abordagem é carregar todos os seus dados de contatos em uma matriz javascript e mantê-la ali mesmo. Você pode adicionar, atualizar ou excluir contatos realizando operações de dados em sua matriz. Em pontos estratégicos, ao longo de determinados procedimentos da aplicação, você pode optar por gravar seus dados no armazenamento local utilizando uma operação simples de código.

Essa não é a abordagem mais sofisticada, mas para muitas aplicações funciona perfeitamente, é rápida o suficiente, mantém o código de forma realmente simples e, a menos que seja necessário lidar com grandes quantidades de dados, é a ferramenta certa para a situação. Se suas necessidades de armazenamento de dados forem mais complexas, com muitas tabelas de dados e relacionamentos envolvendo regras de negócios com restrições, pode ser o caso de se examinar a utilização de um banco de dados SQL tradicional.

O armazenamento interno pode ser realizado também utilizando banco de dados. Muitas aplicações exigem a capacidade de acessar e baixar informação de um repositório de dados e operar sobre essas informações, estando conectado ou mesmo quando está desconectado. Dependendo da quantidade de espaço para armazenar dados, esse repositório poderá ser um banco de dados, que pode estar disponível através da rede ou da internet, ou poderá ser um aplicativo de banco de dados executado no próprio dispositivo móvel.

Você sabe em quais situações podemos optar por um aplicativo de banco de dados executando no próprio dispositivo móvel?

São nas seguintes situações:

- Quando o aplicativo que utiliza os dados armazenados depende apenas de dados offline.
- Quando a largura de banda de rede não é ideal e o aplicativo depende de dados armazenados.
- Quando o desempenho da aplicação é estável e previsível independentemente da disponibilidade da conexão com a rede.

Existem vários aplicativos de banco de dados específicos para dispositivos móveis. O mais popular é o SQLite, outra opção é o WebSQL. Geralmente, os bancos de dados fornecem meios para criar tabelas e relacionamentos entre elas no banco de dados. Os métodos de conexão retornam um objeto que representa o banco de dados, assim é possível executar comandos SQL para gerenciamentos dos dados no banco como consultas, projeção, seleção, colunas e agrupamento, entre outros.

Uma consulta SQL no banco de dados retorna um 'cursor', que aponta para as linhas encontradas pela consulta. O 'cursor' é sempre o mecanismo usado para navegar pelos resultados de uma consulta de banco de dados e ler linhas e colunas.

Armazenamento Externo

O Armazenamento externo pode ser implementado por meio de mídia de dados, cartão SD, pendrive, HD-externo etc. Qualquer função da aplicação ligada ao armazenamento externo deve verificar se a mídia está disponível, antes de executar procedimentos de leitura ou gravação de dados. A mídia pode estar montada no dispositivo, ausente, somente para leitura ou em algum outro estado. Por exemplo, a mídia pode ter sido removida ou pode estar sendo compartilhada com outro aplicativo e estar em uso no momento.

Este tipo de armazenamento pode ser por meio da rede ou internet. Você pode usar a rede ou a internet para armazenar e recuperar dados em seus próprios serviços baseados em aplicações na web. Esse é um outro cenário de armazenamento de dados bem típico, um aplicativo móvel usa um banco de dados hospedado na nuvem e se conecta remotamente a ele para acessar e armazenar seus dados. Isso, é claro, implica que, para ter um bom desempenho, um aplicativo móvel precisa de uma conexão de rede ativa e bastante rápida. A parte do sistema que fornece um serviço de armazenamento para ser acessado remotamente por outros aplicativos recebe o nome de back-end.

Geralmente esse tipo de aplicação utiliza uma arquitetura de *software* orientada a serviços. Nesse cenário temos a aplicação dividida em duas partes distintas, que são o aplicativo, que é executado no dispositivo móvel, representa o front-end e está no lado Cliente da arquitetura, e o serviço web, que representa o back-end e é executado do lado do servidor na arquitetura.

Os serviços web são implementados em linguagens como Java, C#, PHP e utilizam SOA (*Service Oriented Architecture* – Arquitetura Orientada a Serviços); é um estilo de arquitetura de *software* que utiliza um protocolo de troca de mensagens, cujo princípio fundamental determina que as funcionalidades implementadas pelas aplicações devam ser disponibilizadas na forma de serviços.

Os serviços também podem ser implementados utilizando REST (*Representational State Transfer* – Transferência de Estado Representacional), estilo arquitetural de construção de *software* para implementação de serviços web. O REST se tornou um padrão para *softwares* que fornecem interoperabilidade entre sistemas na internet. O estilo descreve qualquer interface web simples que utiliza XML, JSON, ou texto, e utiliza o protocolo HTTP para comunicação de dados.

As arquiteturas orientadas a serviço permitem a criação de serviços de negócio interoperáveis que podem facilmente ser reutilizados e compartilhados entre aplicações. É importante observar que, nesse sentido, esse recurso é muito importante para aplicações para dispositivos móveis.

Em alguns tipos de aplicações pode ser necessária a combinação das abordagens de armazenamento remoto e local, ou seja, um cenário onde os requisitos da aplicação determinam uma solução para manter um banco de dados online, mas, ao mesmo tempo, manter uma réplica local atualizada do banco de dados que permita que seu aplicativo seja totalmente funcional, mesmo quando nenhuma conexão de rede estiver disponível para sincronizar com o banco de dados remoto periodicamente ou quando restabelecer a conexão.

Para saber mais sobre estratégias de armazenamento de dados acesse o guia do desenvolvedor Android. Disponível em: <https://developer.android.com/guide/topics/data/data-storage.html?hl=pt-br>. Acesso em: 23 ago. 2018.

Agora que você já conhece alguns conceitos de armazenamentos de dados em dispositivos móveis, vamos desenvolver um projeto para colocar esse conhecimento em prática.

3. APLICAÇÃO COM GERENCIAMENTO DE DADOS

Neste tópico, vamos desenvolver um exemplo de aplicação com gerenciamento de dados e vamos criar um novo projeto para implementar e conhecer os componentes de gerenciamento de dados da API Ionic; para isso será necessário criar um novo projeto baseado no template blank.

O primeiro passo é gerar a pasta com a estrutura do projeto. Vamos utilizar novamente o prompt de comando; para isso, abra o Terminal ou o prompt de comando e acesse a pasta "dev" de desenvolvimento e crie o novo projeto. O nome do projeto pode ser "appDados".

```
ionic start appDados blank
```

O próximo passo é instalar alguns complementos. Para trabalhar com gerenciamento de dados no framework Ionic, é necessário estabelecer uma ligação com a parte nativa do dispositivo móvel, como já foi mencionado anteriormente. O Cordova é responsável por estabelecer essa ligação, e isso é feito com a instalação de alguns complementos "plugins" no ambiente de desenvolvimento e no próprio projeto.

Para maiores informações sobre o plugin SQLite, acesse o *site* do desenvolvedor Ionic e leia as instruções disponíveis em: <https://ionicframework.com/docs/native/sqlite/>. Acesso em: 23 ago. 2018.

Para a instalação dos plugins, acesse a pasta appDados e digite os comandos:

```
ionic cordova plugin add cordova-sqlite-storage

npm install --save @ionic-native/sqlite
```

Essa aplicação deverá ser instalada no dispositivo móvel para realizar os testes quando estiver concluída. Para habilitar a instalação do seu aplicativo diretamente no dispositivo Android ou iOS, é necessário instalar o suporte à plataforma nativa. No caso da plataforma Android, isso é feito com o seguinte comando:

```
ionic cordova platform add android
```

No caso do iOS, isso é feito com o seguinte comando:

```
ionic cordova platform add iOS
```

Outras instruções sobre como instalar o aplicativo no dispositivo móvel serão abordadas no decorrer dos seus estudos.

Antes de começar com a implementação de código, vamos aprender um pouco mais sobre o framework Ionic conhecendo o componente *provider*.

Os *providers* (provedores) permitem que você crie um componente independente para fornecer algum tipo de funcionalidade específica ao seu aplicativo. O papel de um provedor pode incluir funções, como buscar dados de um servidor, realizar operações em dados, compartilhar dados, fornecer um conjunto específico de operações etc.

Em geral, usamos provedores para separar funcionalidades reutilizáveis em nossas aplicações. Ao implementar um provedor, abstraímos o componente que fornece uma funcionalidade específica; isso torna o código mais sustentável e nos permite reutilizar sua funcionalidade em vários locais diferentes na aplicação.

Continuando com o projeto, ainda dentro da pasta appDados, crie três *providers* com os comandos:

```
ionic g provider basedados

ionic g provider produtos

ionic g provider categorias
```

Ainda no *prompt* de comando, crie uma nova página com o nome editproduto:

```
ionic g page editproduto
```

Agora que o projeto está configurado com os *plugins*, os componentes *providers* e as páginas, vamos iniciar a implementação de código. Para isso, abra o arquivo app.module.ts e faça as alterações para que a implementação fique como é mostrado no Código 1.

Código 1 Código do arquivo app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';

import { ErrorHandler, NgModule, LOCALE_ID } from '@angular/core';
```

```
    import { IonicApp, IonicErrorHandler, IonicModule } from 'ionic-
angular';
    import { SplashScreen } from '@ionic-native/splash-screen';
    import { StatusBar } from '@ionic-native/status-bar';

    import { MyApp } from './app.component';
    import { HomePage } from '../pages/home/home';
    import { EditprodutosPage } from '../pages/editprodutos/editprodutos';

    import { SQLite } from '@ionic-native/sqlite'
    import { BasedadosProvider } from '../providers/database/database';
    import { ProdutosProvider } from '../providers/product/product';
    import { CategoriasProvider } from '../providers/category/category';

    @NgModule({
      declarations: [
        MyApp,
        HomePage,
        EditprodutosPage
      ],
      imports: [
        BrowserModule,
        IonicModule.forRoot(MyApp)
      ],
      bootstrap: [IonicApp],
      entryComponents: [
        MyApp,
        HomePage,
        EditprodutosPage
      ],
```

```
  providers: [
    StatusBar,
    SplashScreen,
    // Configuracao para formatar numeros e datas no formato brasileiro
    {provide: LOCALE_ID, useValue: 'pt-BR'},
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    SQLite,
    BasedadosProvider,
    ProdutosProvider,
    CategoriasProvider
  ]
})
export class AppModule {}
```

Fim código 1

Salve o arquivo e verifique se não ocorreram erros de código.

Conforme já vimos em exemplos anteriores, os componentes que foram gerados (páginas e *providers*) devem ser declarados como `import` no arquivo `app.modules.ts`, e, nesse mesmo arquivo, as páginas devem ser incuídas nos `arrays declarations` e `entryComponents`, e os providers devem ser declarados no `array providers`. O plugin SQLite também é declarado com `import`; observe essas alterações realizadas no código 2.

Abra o arquivo basedados.ts e faça as alterações para que a implementação fique como é mostrado no Código 2.

Código 2 Código do arquivo basedados.ts

```
import { Injectable } from '@angular/core';
import { SQLite, SQLiteObject } from '@ionic-native/sqlite';
import 'rxjs/add/operator/map';

@Injectable()
export class BasedadosProvider {

  constructor(private sqlite: SQLite) { }
```

```
      // Cria um banco caso não exista ou utiliza um banco existente com o
nome no parametro
      public getDB() {
        return this.sqlite.create({
          name: 'products.db',
          location: 'default'
        });
      }

      //Cria a estrutura do banco de dados
      public createDatabase() {
        return this.getDB()
          .then((db: SQLiteObject) => {

            // Criando as tabelas
            this.createTables(db);

            // Inserindo dados de categorias
            this.insertDefaultItems(db);

          })
          .catch(e => console.log(e));
      }


      // Cria as tabelas no banco de dados
      private createTables(db: SQLiteObject) {
        db.sqlBatch([
          ['CREATE TABLE IF NOT EXISTS categories (id integer primary key
AUTOINCREMENT NOT NULL, name TEXT)'],
          ['CREATE TABLE IF NOT EXISTS products (id integer primary
key AUTOINCREMENT NOT NULL, name TEXT, price REAL, duedate DATE, active
integer, category_id integer, FOREIGN KEY(category_id) REFERENCES
categories(id))']
```

```
          ])
            .then(() => console.log('Tabelas criadas'))
            .catch(e => console.error('Erro ao criar as tabelas', e));
        }

        //Incluindo os dados iniciais
        private insertDefaultItems(db: SQLiteObject) {
          db.executeSql('select COUNT(id) as qtd from categories', {})
          .then((data: any) => {
            //Se não existe nenhum registro
            if (data.rows.item(0).qtd == 0) {
              //Registra categorias iniciais
              db.sqlBatch([
                ['insert into categories (name) values (?)',
['Computadores']],
                ['insert into categories (name) values (?)', ['Acessórios']],
                ['insert into categories (name) values (?)', ['Impressoras']]
              ])
                .then(() => console.log('Dados de categorias incluídos'))
                .catch(e => console.error('Erro ao incluir dados de
categorias', e));

            }
          })
          .catch(e => console.error('Erro ao consultar a tabela categorias',
e));
        }

      }
```

Fim código 2

Salve o arquivo e verifique se não ocorreram erros de código.

O provider basedados.ts tem a função de criar o banco e as tabelas de dados caso não existam no dispositivo. O método `getDB()` tem a função de prover acesso ao banco de dados. O método `createDatabase()` tem a função de criar o banco de dados e o método `createTables()` cria e configura as tabelas de dados. Para que a aplicação fique funcional, algumas categorias de produtos são previamente cadastradas; isso é realizado pelo método `insertDefaultItems()`. Observe no código 3 que todos esses procedimentos são realizados com o código SQL.

Abra o arquivo app.component.ts e faça as alterações para que a implementação fique como é mostrado no Código 3.

Para conhecer mais ou relembrar sobre a programação com SQL, acesse o *site* da W3Schools. Disponível em: <https://www.w3schools.com/sql/default.asp>. Acesso em: 23 ago. 2018.

Código 3 Código do arquivo app.component.ts

```
import { Component } from '@angular/core';

import { Platform } from 'ionic-angular';

import { StatusBar } from '@ionic-native/status-bar';

import { SplashScreen } from '@ionic-native/splash-screen';

import { HomePage } from '../pages/home/home';

import { BasedadosProvider } from '../providers/basedados/basedados'


@Component({

  templateUrl: 'app.html'

})

export class MyApp {

  rootPage: any = null;


  constructor(platform: Platform, statusBar: StatusBar,

                splashScreen: SplashScreen, dbProvider:
BasedadosProvider) {


    platform.ready().then(() => {

      // Plataforma pronta.

      // Plugins carregados.
```

```
      statusBar.styleDefault();

      //Criando o banco de dados
      dbProvider.createDatabase()
        .then(() => {
          // fechando a SplashScreen somente quando o banco for criado
          this.openHomePage(splashScreen);
        })
        .catch(() => {
          // Caso ocorrer erro na criação do banco
          this.openHomePage(splashScreen);
        });
    });
  }

  private openHomePage(splashScreen: SplashScreen) {
    splashScreen.hide();
    this.rootPage = HomePage;
  }
}
```

Fim código 3

Salve o arquivo e verifique se não ocorreram erros de código.

O arquivo app.component.ts tem a função de acionar o procedimento para criar e configurar o banco de dados diretamente no dispositivo móvel para que a aplicação possa ser utilizada com o armazenamento de dados. Esse procedimento é realizado no método `platform.ready()` declarado no construtor. Observe a implementação no Código 4. O objeto provider `BasedadosProvider` é declarado no construtor com o nome de `dbProvider` e a criação e configuração é realizada com a chamada ao método `dbProvider.createDatabase()`.

Abra o arquivo do *provider* produtos.ts e faça as alterações para que a implementação fique como é mostrado no Código 4.

Código 4 Código do arquivo produto.ts

```
import { Injectable } from '@angular/core';

import { SQLiteObject } from '@ionic-native/sqlite';

import { BasedadosProvider } from '../../providers/basedados/
basedados';

import 'rxjs/add/operator/map';


@Injectable()

export class ProdutosProvider {


  constructor(private dbProvider: BasedadosProvider) { }


  public insert(product: Product) {

    return this.dbProvider.getDB()

      .then((db: SQLiteObject) => {

        let sql = 'insert into products (name, price, duedate, active,
category_id) values (?, ?, ?, ?, ?)';

        let data = [product.name, product.price, product.duedate,
product.active ? 1 : 0, product.category_id];


        return db.executeSql(sql, data)

          .catch((e) => console.error(e));

      })

      .catch((e) => console.error(e));

  }


  public update(product: Product) {

    return this.dbProvider.getDB()

      .then((db: SQLiteObject) => {

        let sql = 'update products set name = ?, price = ?, duedate =
?, active = ?, category_id = ? where id = ?';

        let data = [product.name, product.price, product.duedate,
product.active ? 1 : 0, product.category_id, product.id];
```

```
      return db.executeSql(sql, data)
        .catch((e) => console.error(e));
    })
    .catch((e) => console.error(e));
}

public remove(id: number) {
  return this.dbProvider.getDB()
    .then((db: SQLiteObject) => {
      let sql = 'delete from products where id = ?';
      let data = [id];

      return db.executeSql(sql, data)
        .catch((e) => console.error(e));
    })
    .catch((e) => console.error(e));
}

public get(id: number) {
  return this.dbProvider.getDB()
    .then((db: SQLiteObject) => {
      let sql = 'select * from products where id = ?';
      let data = [id];

      return db.executeSql(sql, data)
        .then((data: any) => {
          if (data.rows.length > 0) {
            let item = data.rows.item(0);
            let product = new Product();
            product.id = item.id;
```

```
                product.name = item.name;
                product.price = item.price;
                product.duedate = item.duedate;
                product.active = item.active;
                product.category_id = item.category_id;

                return product;
              }

              return null;
            })
            .catch((e) => console.error(e));
        })
        .catch((e) => console.error(e));
    }

    public getAll(active: boolean, name: string = null) {
      return this.dbProvider.getDB()
        .then((db: SQLiteObject) => {
          let sql = 'SELECT p.*, c.name as category_name FROM products p
inner join categories c on p.category_id = c.id where p.active = ?';
          var data: any[] = [active ? 1 : 0];

          // filtrando pelo nome
          if (name) {
            sql += ' and p.name like ?'
            data.push('%' + name + '%');
          }

          return db.executeSql(sql, data)
            .then((data: any) => {
              if (data.rows.length > 0) {
```

```
                let products: any[] = [];
                for (var i = 0; i < data.rows.length; i++) {
                  var product = data.rows.item(i);
                  products.push(product);
                }
                return products;
              } else {
                return [];
              }
            })
            .catch((e) => console.error(e));
        })
        .catch((e) => console.error(e));
    }
  }

export class Product {
  id: number;
  name: string;
  price: number;
  duedate: Date;
  active: boolean;
  category_id: number;
}
```

Fim código 4

Salve o arquivo e verifique se não ocorreram erros de código.

O *provider* produtos.ts tem a função de realizar as operações básicas de gerenciamento de dados da tabela de produtos. Isso inclui métodos para inserção, alteração, pesquisa e exclusão de dados. Observe os métodos `insert()`, `update()`, `remove()`, `get()` e

`getAll()`, implementados no Código 4, veja que todos os métodos utilizam código SQL, observe também, no final do arquivo, a classe de modelagem de dados Products.

Abra o arquivo do *provider* categorias.ts e faça as alterações para que a implementação fique como é mostrado no Código 5.

Código 5 Código do arquivo categorias.ts

```
import { Injectable } from '@angular/core';

import { SQLiteObject } from '@ionic-native/sqlite';

import { BasedadosProvider } from '../../providers/basedados/
basedados';

import 'rxjs/add/operator/map';


@Injectable()

export class CategoriasProvider {


  constructor(private dbProvider: BasedadosProvider) { }


  public getAll() {

    return this.dbProvider.getDB()

    .then((db: SQLiteObject) => {


      return db.executeSql('select * from categories', [])

        .then((data: any) => {

          if (data.rows.length > 0) {

            let categories: any[] = [];

            for (var i = 0; i < data.rows.length; i++) {

              var category = data.rows.item(i);

              categories.push(category);

            }

            return categories;

          } else {

            return [];

          }
```

```
            })
            .catch((e) => console.error(e));
        })
        .catch((e) => console.error(e));
    }

}
```

Fim código 5

Salve o arquivo e verifique se não ocorreram erros de código.

Observe, no Código 5, que o *provider* categorias.ts tem apenas a função de pesquisa de dados da tabela de categorias no método `getAll()`. Isso significa que os dados dessa tabela são somente para leitura.

Abra o arquivo do controlador da página editproduto.ts e faça as alterações para que a implementação fique como é mostrado no Código 6.

Código 6 Código do arquivo editproduto.ts

```
import { Component } from '@angular/core';

import { IonicPage, NavController, ToastController, NavParams } from
'ionic-angular';

import { ProdutosProvider, Product } from '../../providers/produtos/
produtos';

import { CategoriasProvider } from '../../providers/categorias/
categorias';


@IonicPage()

@Component({

  selector: 'page-editprodutos',

  templateUrl: 'editprodutos.html',

})

export class EditprodutosPage {


  model: Product;

  categories: any[];
```

```
    constructor(
      public navCtrl: NavController, public navParams: NavParams,
      private toast: ToastController, private productProvider:
ProdutosProvider,
      private categoryProvider: CategoriasProvider) {

      this.model = new Product();

      if (this.navParams.data.id) {
        this.productProvider.get(this.navParams.data.id)
          .then((result: any) => {
            this.model = result;
          })
      }
    }

    ionViewDidLoad() {
      this.categoryProvider.getAll()
        .then((result: any[]) => {
          this.categories = result;
        })
        .catch(() => {
          this.toast.create({ message: 'Erro ao carregar as
categorias.', duration: 3000, position: 'botton' }).present();
        });
    }

    save() {
      this.saveProduct()
        .then(() => {
          this.toast.create({ message: 'Produto salvo.', duration: 3000,
position: 'botton' }).present();
```

```
            this.navCtrl.pop();
          })
          .catch(() => {
            this.toast.create({ message: 'Erro ao salvar o produto.',
duration: 3000, position: 'botton' }).present();
          });
      }

      private saveProduct() {
        if (this.model.id) {
          return this.productProvider.update(this.model);
        } else {
          return this.productProvider.insert(this.model);
        }
      }

    }
```

Fim código 6

Salve o arquivo e verifique se não ocorreram erros de código.

O controlador da página editproduto utiliza as operações disponíveis nos *providers* `CategoriasProvider` e `ProdutosProvider` para realizar as operações de edição de dados. Observe no Código 6 que a operação `save()` utiliza um método de inclusão ou alteração de dados por meio de um teste com o id (código de identificação) do produto. Se existe um id, o método realiza a operação de alteração; caso não exista o id, o método realiza a operação de inclusão na base de dados.

Abra o arquivo da página editproduto.html e faça as alterações para que a implementação fique como é mostrado no Código 7.

Código 7 Código do arquivo editproduto.html

```
    <ion-header>

      <ion-navbar>
        <ion-title>Cadastro Produtos</ion-title>
      </ion-navbar>
```

```
    </ion-header>


    <ion-content padding>

        <ion-list>


          <ion-item>

            <ion-label stacked>Descrição</ion-label>

            <ion-input type=”text” name=”name” [(ngModel)]=”model.name”></
ion-input>

          </ion-item>


          <ion-item>

            <ion-label stacked>Preço</ion-label>

            <ion-input type=”number” name=”price” [(ngModel)]=”model.
price”></ion-input>

          </ion-item>


          <ion-item>

            <ion-label stacked>Inclusão</ion-label>

            <ion-datetime displayFormat=”DD/MM/YYYY” name=”duedate”
min=”2017” max=”2020-12-31” [(ngModel)]=”model.duedate”></ion-datetime>

          </ion-item>


          <ion-item>

            <ion-label stacked>Categoria</ion-label>

            <ion-select name=”category_id” [(ngModel)]=”model.category_
id”>

              <ion-option *ngFor=”let category of categories” value=”{{
category.id }}”>{{ category.name}}</ion-option>

            </ion-select>

          </ion-item>


          <ion-item>
```

```
            <ion-label>Ativo</ion-label>

            <ion-checkbox name="active" [(ngModel)]="model.active"></ion-
checkbox>

          </ion-item>

        </ion-list>

        <button ion-button block (click)="save()">Salvar</button>

    </ion-content>
```

Fim código 7

Salve o arquivo e verifique se não ocorreram erros de código.

Observe no Código 7 que a página editprodutos.html utiliza um conjunto de componentes Ionic para apresentar a funcionalidade de edição de produtos ao usuário. As tags <ion-list> e <ion-item> são utilizadas para implementar a lista de produtos. A diretiva ngModel é utilizada para fazer a transferência de valores das variáveis entre a View e o Controlador. A tag <button> aciona o método `save()` no controlador.

Abra o arquivo do controlador da página home.ts e faça as alterações para que a implementação fique como é mostrado no Código 8.

Código 8 Código do arquivo home.ts

```
    import { Component } from '@angular/core';

    import { IonicPage, NavController, ToastController } from 'ionic-
angular';

    import { ProdutosProvider, Product } from '../../providers/produtos/
produtos'

    import { EditprodutosPage } from '../../pages/editprodutos/
editprodutos';


    @IonicPage()

    @Component({

      selector: 'page-home',

      templateUrl: 'home.html'

    })

    export class HomePage {
```

```
products: any[] = [];
onlyInactives: boolean = false;
searchText: string = null;

constructor(public navCtrl: NavController,
            private toast: ToastController,
            private productProvider: ProdutosProvider) { }

ionViewDidEnter() {
  this.getAllProducts();
}

getAllProducts() {
  this.productProvider.getAll(!this.onlyInactives, this.searchText)
    .then((result: any[]) => {
      this.products = result;
    });
}

addProduct() {
  this.navCtrl.push(EditprodutosPage);
}

editProduct(id: number) {
  this.navCtrl.push(EditprodutosPage, { id: id });
}

removeProduct(product: Product) {
  this.productProvider.remove(product.id)
    .then(() => {
      // Removendo o produto do array de produtos
      var index = this.products.indexOf(product);
```

```
            this.products.splice(index, 1);

            this.toast.create({ message: 'Produto removido.', duration:
3000, position: 'botton' }).present();

          })

      }


      filterProducts(ev: any) {

        this.getAllProducts();

      }


  }
```

Fim código 8

Salve o arquivo e verifique se não ocorreram erros de código.

O controlador da página home utiliza as operações disponíveis no provider `ProdutosProvider` para realizar as operações de gerenciamento de dados. Observe os métodos implementados no Código 8, `getAllProducts()`, `addProduct()`, `removeProduct()` e `filterProducts()`. Todos esses métodos fazem chamadas aos métodos do provider `ProdutosProvider`.

Abra o arquivo da página home.html e faça as alterações para que a implementação fique como é mostrado no Código 9.

Código 9 Código do arquivo home.html

```
  <ion-header>


    <ion-navbar>

      <ion-title>Exemplo SQLite</ion-title>

    </ion-navbar>


  </ion-header>



  <ion-content padding>

    <ion-searchbar (ionInput)="filterProducts($event)"
[(ngModel)]="searchText"></ion-searchbar>

    <ion-list>
```

```
        <ion-item-sliding *ngFor="let product of products">

          <button ion-item (click)="editProduct(product.id)">
            <h1>{{ product.name }}</h1>
            <h2>{{ product.category_name }}</h2>
            <h2>{{ product.duedate | date:'dd/MM/yyyy' }}</h2>
            <h2 class="price">{{ product.price | currency:'BRL':true }}</
h2>
          </button>

          <ion-item-options side="left">
            <button ion-button color="danger" (click)="removeProduct(produ
ct)">
              <ion-icon name="trash"></ion-icon>
              Excluir
            </button>
          </ion-item-options>

        </ion-item-sliding>
      </ion-list>

      <ion-fab right bottom>
        <button ion-fab color="light" (click)="addProduct()"><ion-icon
name="add"></ion-icon></button>
      </ion-fab>

    </ion-content>

    <ion-footer>
      <ion-toolbar>

        <ion-list>
          <ion-item no-lines>
            <ion-label>Listar inativos</ion-label>
```

```
            <ion-toggle name="listarInativos" [(ngModel)]="onlyInactives"
(ionChange)="getAllProducts()"></ion-toggle>

          </ion-item>

        </ion-list>

      </ion-toolbar>

    </ion-footer>
```

Fim código 9

Verifique se todos os arquivos foram salvos e não restaram erros de código.

Conforme já vimos anteriormente, observe no Código 9 que a página home.html utiliza um conjunto de componentes Ionic para apresentar a funcionalidade de edição de produtos ao usuário. As tags <ion-list> e <ion-item> são utilizadas para implementar a lista de produtos. A diretiva ngModel é utilizada para fazer a transferência de valores das variáveis entre a View e o Controlador. No rodapé da página, a tag <ion-toggle> aciona o método `getAllProducts()` passando um valor booleano para determinar o filtro de produtos no controlador.

Após todas as alterações de código, a página home.html é apresentada conforme é mostrado na Figura 1.

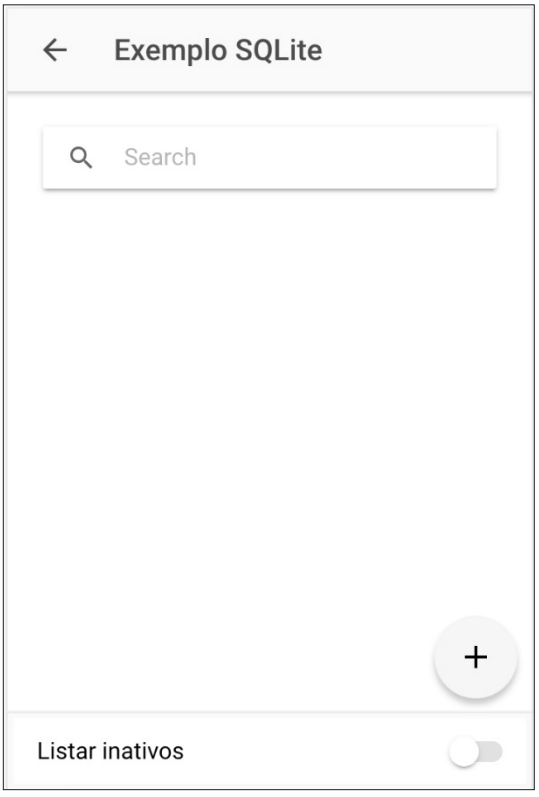


Figura 1 *Tela inicial do aplicativo appDados.*

Clique no botão (+) para incluir novos registros no cadastro de produtos.

Para excluir um registro, arraste da direita para a esquerda, conforme é mostrado na Figura 2.

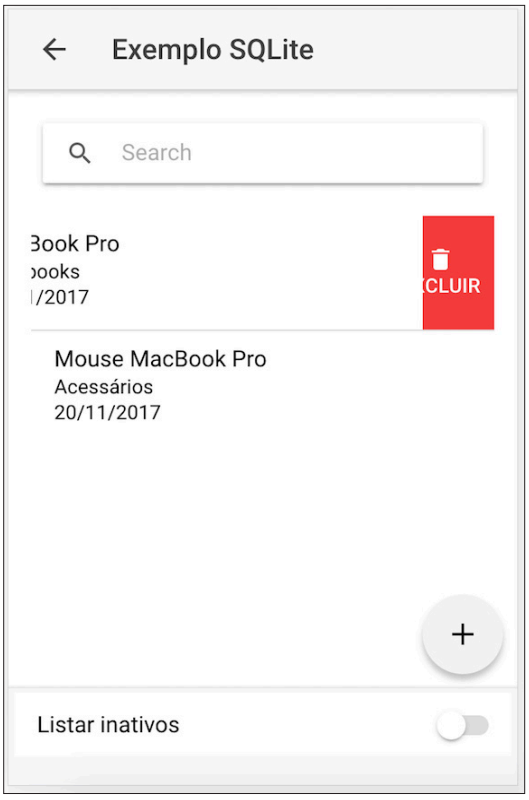


Figura 2 *Exclusão de registros no cadastro de produtos.*

Com a codificação do projeto concluída, podemos passar à instalação do aplicativo no dispositivo móvel.

4. EXECUTANDO O APLICATIVO NO DISPOSITIVO MÓVEL

Com a aplicação pronta, é hora de gerar o arquivo de instalação (apk). Antes de começar, verifique a instalação das dependências necessárias, verifique também os requisitos de *hardware* e *software* necessários. Caso já tenha instalado, desconsidere.

Dependências:

- Java JDK
- Android Studio

Caso não tenha as instalações necessárias, solicitamos que faça as instalações. A seguir, indicamos os *sites* que você deverá acessar e fazer as instalações.

- Para instalar o Java SDK, consulte o site da Oracle disponível em: <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>. Acesso em: 23 ago. 2018.
- Para instalar o Android Studio, consulte o site do desenvolvedor Android disponível em: <https://developer.android.com/studio/index.html>. Acesso em: 23 ago. 2018.

As instruções para a instalação das dependências deverão ser de acordo com a versão de seu sistema operacional.

Após a instalação, entre na pasta do projeto appDados e digite o comando:

```
ionic cordova build android
```

Esse processo vai demorar alguns minutos e será necessário estar conectado à internet. Ao final do processo, o arquivo APK do seu aplicativo será gerado nesta pasta:

```
Dev/appDados/platforms/android/build/outputs/apk/android-debug.apk
```

O arquivo gerado `android-debug.apk` pode ser transferido e executado no seu dispositivo móvel android, porém você deve desbloquear as configurações que permitem a instalação de aplicativos não assinados. Caso opte por fazer isso, desbloqueie as configurações, instale o aplicativo e depois bloqueie novamente.

Para que o seu aplicativo seja assinado, você deverá criar uma conta no Google Play Store, fazer um *upload* do arquivo apk e realizar a instalação em seu dispositivo móvel a partir da loja Google Play Store.

Para gerar um arquivo de instalação para o sistema iOS, veja o procedimento no *site* do Ionic framework e leia as instruções do tópico iOs Devices disponíveis no *link*: <http://ionicframework.com/docs/intro/deploying/>. Acesso em: 20 set. 2018.

Assim, concluímos o estudo desta unidade e o exemplo com gerenciamento de dados local com SQLite. Esperamos que você aprofunde seus conhecimentos com as leituras indicadas sobre esses assuntos. Na próxima unidade, veremos a utilização de dados na nuvem com *Webservice Restful.*

5. QUESTÕES AUTOAVALIATIVAS

Agora é o momento ideal para que você faça uma revisão dos estudos realizados nesta unidade. Lembre-se de que, na Educação a Distância, a construção do conhecimento ocorre de forma cooperativa e colaborativa; compartilhe, portanto, as suas descobertas com seus colegas.

Confira, a seguir, as questões propostas para verificar o seu desempenho no estudo desta unidade.

1) Qual diretiva Ionic (Angular) é utilizada para fazer a transferência de valores das variáveis entre a View e o Controlador?
   a) {{ion-item}}
   b) *ngFor
   c) ngApp
   d) {{ngShow}}
   e) ngModel

2) Componentes do framework Ionic como Page e Provider, após gerados, devem ser incluídos no arquivo app.modules.ts. Quais os nomes dos arrays que devem receber a inclusão desses componentes?
   a) Imports, ngModels e providers

b) EntryComponents, Declarations e Providers
c) ngModuls, Declarations e EntryComponents
d) Imports, providers e EntryComponents
e) AppModules, ngModels e Declarations

Gabarito

Confira, a seguir, as respostas corretas para as questões autoavaliativas propostas:

1) Alternativa E.

2) Alternativa B.

6. CONSIDERAÇÕES

Nesta unidade, você teve a oportunidade de aprofundar seus conhecimentos relacionados aos conceitos e práticas de armazenamento de dados em aplicações para dispositivos móveis. Você pôde perceber que existem boas opções como estratégias de armazenamento e gerenciamento de dados, seja internamente ou externamente ao dispositivo móvel, mas tudo dependerá da conexão com a internet, da quantidade de dados e do desempenho da aplicação.

Você pôde acompanhar e implementar um exemplo prático de aplicativo que utiliza o banco de dados SQLite para armazenar e gerenciar os dados. Com esse exemplo de aplicação, adotamos a estratégia de armazenamento local, assim você pôde conferir na prática que este tipo de armazenamento permite que seus aplicativos funcionem efetivamente quando não há conexão, além de oferecer funcionalidades mais avançadas e, muitas vezes, mais rápidas do que o armazenamento no lado do servidor.

Na próxima unidade, veremos um exemplo de aplicação que utiliza um serviço na nuvem para armazenar e gerenciar dados. O objetivo é que você complemente seus estudos de maneira que compreenda os conceitos necessários ao estudo de interoperabilidade de sistemas utilizando as tecnologias de Web Services.

7. E-REFERÊNCIAS

ANDROID Developers. Data-Storage. Disponível em: <https://developer.android.com/guide/topics/data/index.html>. Acesso em: 23 ago. 2018.

ANDROID STUDIO. Site do desenvolvedor Android. Disponível em: <https://developer.android.com/studio/index.html>. Acesso em: 23 ago. 2018.

IONIC Framework, Deployng. Disponível em: <http://ionicframework.com/docs/intro/deploying/>. Acesso em: 23 ago. 2018.

IONIC Framework, Storage. Disponível em: <https://ionicframework.com/docs/storage/>. Acesso em: 23 ago. 2018.

IONIC, SQLite plugin. Site do desenvolvedor, documentão técnica. Disponível em: <https://ionicframework.com/docs/native/sqlite/>. Acesso em: 23 ago. 2018.

Java SDK Oracle. Site do desenvolvedor, documentão técnica. Disponível em: <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>. Acesso em: 23 ago. 2018.

SQLITE Database. Site do desenvolvedor, documentão técnica. Disponível em: <https://www.sqlite.org/>. Acesso em: Acesso em: 23 ago. 2018.

W3SCHOOLS. HTML5 Webstorage. Site do desenvolvedor, documentão técnica. Disponível em: <https://www.w3schools.com/html/html5_webstorage.asp>. Acesso em: 23 ago. 2018.

W3Schools, SQL. Site do desenvolvedor, documentão técnica. Disponível em: <https://www.w3schools.com/sql/default.asp>. Acesso em: 23 ago. 2018.

UNIDADE 4

CONSUMINDO WEB SERVICE RESTFUL COM APLICATIVO IONIC

Objetivos

- Compreender conceitos de armazenamentos de dados na Web.
- Compreender conceitos de aplicação orientada a serviços.
- Conhecer conceitos da tecnologia Restful para utilização de Web Services.
- Desenvolver um projeto utilizando a tecnologia REST para utilizar serviço de gerenciamento de dados na web.
- Entender a importância da utilização de *Web Services* em aplicações para dispositivos móveis.

Conteúdos

- Armazenamentos de dados na nuvem.
- Restful Web Services.

Orientações para o estudo da unidade

Antes de iniciar o estudo desta unidade, é importante que você leia as orientações a seguir:

1) Nas instruções para criar e configurar novos projetos, siga exatamente as orientações, em especial a que se refere ao nome do projeto e de componentes, pois esses nomes são utilizados no código-fonte da aplicação.

2) Pesquise mais sobre as tecnologias REST e SOA, procure entender bem suas características, vantagens e desvantagens.

3) Não se limite a implementar apenas as funcionalidades abordadas no exemplo desta unidade, procure melhorar a aplicação observando a necessidade de novas funções e a utilização de outros componentes da plataforma Ionic.

1. INTRODUÇÃO

Na unidade anterior, você teve a oportunidade de compreender o armazenamento de dados local utilizando o banco de dados SQLite.

Nesta unidade, você terá a possibilidade de aprender a utilizar a tecnologia Restful para realizar integração de dados com serviços web na nuvem. O objetivo desta unidade é estabelecer os conceitos necessários para o estudo de interoperabilidade de sistemas utilizando as tecnologias de Web Services. Para isso, inicialmente abordamos os termos de interoperabilidade e Web Service, em seguida explicamos os fundamentos básicos da tecnologia Restful, e, ao final, implementamos um exemplo de aplicação com Ionic que se interliga com um serviço Restful para troca de dados.

Caso você já tenha estudado a disciplina *Engenharia de Software*, deve estar lembrado que o termo interoperabilidade é usado para descrever a capacidade de diferentes sistemas de *software* em realizar trocas de dados por meio de protocolos. Na verdade, a interoperabilidade permite a integração de dados entre sistemas diferentes. De acordo com a normativa ISO/IEC 2382-01, o termo interoperabilidade é definido como: "A capacidade de se comunicar, executar programas ou transferir dados entre várias unidades funcionais de uma maneira que exige que o usuário tenha pouco ou nenhum conhecimento das características únicas dessas unidades".

Como vimos anteriormente, no caso de aplicativos para dispositivos móveis, o uso de Web Service é considerado como armazenamento externo.

Neste caso, os serviços baseados em aplicações web disponibilizados na internet para armazenar e recuperar dados é um cenário de gerenciamento de dados bem típico nos aplicativos atuais. Um aplicativo móvel, por sua vez, se conecta remotamente a uma aplicação (serviço) hospedada na nuvem e utiliza seus recursos para acessar e armazenar dados. Isso implica que, para ter um bom desempenho, o aplicativo móvel precisa de uma conexão de rede relativamente rápida. A parte do sistema que fornece um serviço de web para ser acessado remotamente por outros aplicativos pode utilizar tecnologias como SOA ou Restful e recebe o nome de back-end.

Quando dois sistemas interagem por meio de uma camada de serviços, dizemos que utilizam arquitetura de *software* orientada a serviços. Nesse cenário, temos a aplicação dividida em duas partes distintas, aquela em que o aplicativo executado no dispositivo móvel representa o front-end e está no lado Cliente da arquitetura e aquela em que o serviço web representa o back-end e é executado do lado do servidor na arquitetura.

Para darmos continuidade aos nossos estudo, nessa unidade implementaremos um projeto Ionic que se comunica com um serviço web por meio da tecnologia REST (*Representational State Transfer* – transferência de estado representacional).

A tecnologia REST se refere a um conjunto coordenado de restrições arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuído, descreve qualquer interface web simples que utiliza XML, JSON, ou texto e utiliza o protocolo HTTP para comunicação de dados.

Além disso, as arquiteturas orientadas a serviço permitem a criação de aplicações que se comunicam independentemente da linguagem em que foram construídas, o que é fundamental quando de pensa em manutenção de *software*. Neste sentido, esse recurso é muito importante para aplicações para dispositivos móveis.

Para saber mais sobre esse assunto, leia o Capítulo 19 da obra *Engenharia de Software*. Ian Sommerville, 9. ed., disponível na biblioteca Pearson.

2. INTEROPERABILIDADE DE SISTEMAS COM RESTFUL WEB SERVICE

Em aplicações de serviço Web RESTful, geralmente se utiliza o protocolo HTTP para requisitar recursos. As operações disponíveis no protocolo HTTP incluem os métodos predefinidos GET, POST, PUT, DELETE, entre outros. As requisições das operações são realizadas por meio da URI de um recurso disponível na web e provocam uma resposta que pode usar os formatos XML, HTML, JSON ou algum outro formato definido. A resposta à requisição pode confirmar que alguma alteração foi feita para o recurso armazenado ou pode fornecer *links* de hipertexto para outros recursos ou coleções de recursos relacionadas.

Basicamente, a implementação de um Web Service deve definir os seguintes aspectos:

- A URI base/raiz para o serviço web, tais como o contexto da aplicação e seus recursos.
- O tipo MIME dos dados de resposta que poderão ser: JSON, XML, texto puro, entre outros.
- O conjunto de operações suportadas pelo serviço, por exemplo, POST, GET, PUT ou DELETE.

Um recurso é uma entidade, ou seja, é um objeto com informações que será representado por meio de dados em XML, JSON etc. Em geral, a URL para acessar esse recurso será sempre a mesma, o que muda é o método HTTP de acesso (GET, POST, PUT, DELETE), e de acordo com o método escolhido o resultado da requisição será diferente. Veja os exemplos a seguir:

Operações sobre uma URI: http://exemplo.com/contatos/

- GET obtém a coleção
- POST adiciona um item à coleção
- PUT substitui a coleção
- DELETE remove a coleção

Operações sobre um item na URI: http://exemplo.com/contatos/17

- GET obtém o item
- POST adiciona o item
- PUT atualiza o item
- DELETE remove o item

Os formatos para transmitir os objetos de dados podem ser JSON, XML etc. Observe, a seguir, a descrição de cada um deles.

JSON (*JavaScript Object Notation*) é um formato de dados de padrão aberto criado para transmitir objetos de dados na internet.

Ao trocar dados entre um navegador e um servidor web, os dados só podem trafegar pela rede em formato texto. Em alguns casos é necessário transmitir objetos pela rede, assim é necessário converter objeto em texto e vice-versa. Utilizando o formato JSON, podemos converter qualquer objeto JavaScript em JSON e enviar JSON para o servidor. Também podemos converter qualquer JSON recebido do servidor em objetos JavaScript.

Já o XML (*Extensible Markup Language*) é uma linguagem padronizada de marcação capaz de descrever diversos tipos de dados. Ela foi criada para facilitar o compartilhamento de informações na internet.

Esses são os fundamentos básicos da tecnologia Restful. Na sequência, estudaremos o web service como aplicativo Ionic.

Para saber mais sobre os conceitos da tecnologia Restful, pesquise pelo termo Restful na web.

Sugerimos que aprofunde seus conhecimentos sobre JSON e XML, acessando os *links* a seguir:

- W3Schools. JSON – Introduction. Disponível em: <https://www.w3schools.com/js/js_json_intro.asp>. Acesso em: 27 ago. 2018.
- W3Schools. XML Tutorial. Disponível em: <https://www.w3schools.com/xml/default.asp>. Acesso em: 27 ago. 2018.

3. CONSUMINDO WEB SERVICE COM APLICATIVO IONIC

Agora que você já conhece os fundamentos de interoperabilidade e Web Service Restful, vamos desenvolver um projeto para colocar esse conhecimento em prática.

Nesse exemplo de aplicação, criaremos um projeto Ionic com a finalidadade de implementar um aplicativo que faz requisições a um serviço Restful para troca de dados. Para isso, será necessário criar o novo projeto baseado no template blank.

Vamos criar e preparar o projeto. Para isso, realize os procedimentos a seguir:

- acesse a pasta dev;
- crie um novo projeto com o nome “appRestful”;
- acesse a pasta do projeto;
- crie uma nova página com o nome “editusers”;
- crie um novo provider com o nome “RestfulService”;
- registre a página editusers e o provider restful-service no arquivo app.module.ts.

Após concluído e configurado o projeto, é hora de fazer uma chamada de API REST no provedor de serviços web. Para tornar esse exemplo mais prático, usaremos a API de REST gratuita disponível para testes na internet.

Veremos a declaração da URL do provedor de teste na implementação do Código 1.

Abra o arquivo restful-service.ts e faça as alterações para que a implementação fique como é mostrado no Código 1.

Código 1 Código do arquivo restful-service.ts

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import 'rxjs/add/operator/map';

@Injectable()
export class RestfulServiceProvider {

  data: any;
  apiUrl = 'https://jsonplaceholder.typicode.com';

  constructor(public http: Http) {
    console.log('Hello RestfulServiceProvider Provider');
  }

  getUsers() {
    if (this.data) {
      return Promise.resolve(this.data);
    }

    return new Promise(resolve => {
      this.http.get(this.apiUrl+'/users')
        .map(res => res.json())
        .subscribe(data => {
          this.data = data;
          resolve(this.data);
```

```
            });
        });
    }

    saveUser(data) {
      return new Promise((resolve, reject) => {
        this.http.post(this.apiUrl+'/users', JSON.stringify(data))
          .subscribe(res => {
            resolve(res);
          }, (err) => {
            reject(err);
          });
      });
  }

  }
```

Fim código 1

Como você pôde observar no Código 1, o Web Service Restful responde às requisições do cliente URI declarada na variável: `apiUrl = 'https://jsonplaceholder.typicode.com'`. O método `getUsers()` recebe os dados que representam a coleção de usuários por meio do método GET no comando: `this.http.get(this.apiUrl+'/users')`. Esse comando faz uma concatenação juntando o endereço global do Web Service com o endereço específico da coleção de usuários.

Os dados da coleção de usuários são obtidos, na resposta da requisição, no formato JSON, conforme o comando: `map(res => res.json()` e, em seguida, são transferidos para a variável "data".

O método `saveUser()` realiza o processo no sentido oposto, cliente para o servidor, ou seja, envia os dados para serem armazenados por meio do método POST, no comando: `this.http.post(this.apiUrl+'/users', JSON.stringify(data))`. Os dados são enviados no format JSON.

Salve o arquivo e verifique se não ocorreram erros de código. Abra o arquivo home.ts e faça as alterações para que a implementação fique como é mostrado no Código 2.

Código 2 Código do arquivo home.ts

```
import { Component } from '@angular/core';

import { IonicPage, NavController, NavParams } from 'ionic-angular';

import { RestfulServiceProvider } from '../../providers/restful-
service/restful-service';

import { EditusersPage } from '../../pages/editusers/editusers';


@IonicPage()

@Component({

  selector: 'page-home',

  templateUrl: 'home.html'

})

export class HomePage {


  users: any;


  constructor(public navCtrl: NavController,

              public navParams: NavParams,

              public restServ: RestfulServiceProvider) {

    this.getUsers();

  }


  ionViewDidLoad() {

    console.log('ionViewDidLoad UsersPage');

  }


  getUsers() {

    this.restServ.getUsers()

    .then(data => {

      this.users = data;

      console.log(this.users);

    });

  }
```

```
    userSelected(user: any){
      console.log('User selecionado: '+user.name);
      this.navCtrl.push(EditusersPage, user);
    }
  }
```

Fim código 2

Observe no código 2 que o método `getUsers()` faz uma chamada ao método `getUsers()` do provider RestfulProvider. O método `userSelected()` aciona a navegação para a página EditUser passando o usuário selecionado como parâmetro.

Salve o arquivo e verifique se não ocorreram erros de código. Abra o arquivo home.html e faça as alterações para que a implementação fique como é mostrado no Código 3.

Código 3 Código do arquivo home.html

```
<ion-header>

  <ion-navbar>
    <ion-title>users</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>

  <h2>Lista de Usuários</h2>
  <ion-list>
    <button ion-item *ngFor="let user of users;let i=index"
(click)="userSelected(user)">
      {{user.name}}
    </button>
  </ion-list>

</ion-content>
```

Fim código 3

Observe no Código 3 que a página home.html mostra uma lista (List) de usuários que vem do Web Service. Cada item da lista é um botão (button) e contém dados de um usuário em particular, como nome, email etc.

Quando uma pessoa que usa esse aplicativo clica em uma linha da lista, ela está selecionando a linha. Lembrando que se trata de uma lista de usuários, e cada linha contém dados de um usuário em particular, na verdade a pessoa está selecionando o usuário, e os dados do usuário são passados no método `(click)="userSelected(user)"`.

Salve o arquivo e verifique se não ocorreram erros de código. Abra o arquivo editusers.ts e faça as alterações para que a implementação fique como é mostrado no Código 4.

Código 4 Código do arquivo editusers.ts

```
    import { Component } from '@angular/core';

    import { IonicPage, NavController, NavParams } from 'ionic-angular';

    import { RestfulServiceProvider } from '../../providers/restful-
service/restful-service';



    @IonicPage()

    @Component({

      selector: 'page-editusers',

      templateUrl: 'editusers.html',

    })

    export class EditusersPage {


      user = { name: '', username: '', email: '', phone: '', website: '',
address: { street: '', suite: '', city: '', zipcode: '', geo: { lat: '',
lng: '' } }, company: { name: '', bs: '', catchPhrase: '' }};


      constructor(public navCtrl: NavController,

                  public navParams: NavParams,

                  public restServ: RestfulServiceProvider) {

          this.user = this.navParams.data;

      }


      ionViewDidLoad() {

        console.log('ionViewDidLoad EditusersPage');

      }
```

```
  saveUser() {
    this.restServ.saveUser(this.user).then((result) => {
      console.log(result);
    }, (err) => {
      console.log(err);
    });
  }

  }
```

Fim código 4

Observe a variável “user” no código 4; ela foi declarada no formato JavaScript object para receber dados do formato JSON. Essa variável recebe os dados do usuário por meio do comando: `this.user = this.navParams.data`.

Salve o arquivo e verifique se não ocorreram erros de código.

Em seguida, abra o arquivo editusers.html e faça as alterações para que a implementação fique como é mostrado no Código 5.

Código 5 Código do arquivo editusers.html

```
  <ion-header>

    <ion-navbar>
      <ion-title>editusers</ion-title>
    </ion-navbar>

  </ion-header>

  <ion-content padding>

    <h2>Adicionar Usuário</h2>

      <ion-item>
        <ion-label>Name</ion-label>
        <ion-input type=”text” [(ngModel)]=”user.name” name=”name”></
ion-input>
```

```
          </ion-item>

          <ion-item>

            <ion-label>Username</ion-label>

            <ion-input type=”text” [(ngModel)]=”user.username”
name=”username”></ion-input>

          </ion-item>

          <ion-item>

            <ion-label>Email</ion-label>

            <ion-input type=”email” [(ngModel)]=”user.email” name=”email”></
ion-input>

          </ion-item>

          <ion-item>

            <ion-label>Phone</ion-label>

            <ion-input type=”text” [(ngModel)]=”user.phone” name=”phone”></
ion-input>

          </ion-item>

          <ion-item>

            <ion-label>Website</ion-label>

            <ion-input type=”url” [(ngModel)]=”user.website”
name=”website”></ion-input>

          </ion-item>

          <ion-item-divider color=”light”>Address</ion-item-divider>

          <ion-item>

            <ion-label>Street</ion-label>

            <ion-input type=”text” [(ngModel)]=”user.address.street”
name=”street”></ion-input>

          </ion-item>

          <ion-item>

            <ion-label>Suite</ion-label>

            <ion-input type=”text” [(ngModel)]=”user.address.suite”
name=”suite”></ion-input>
```

```
        </ion-item>

        <ion-item>

          <ion-label>City</ion-label>

          <ion-input type="text" [(ngModel)]="user.address.city"
name="city"></ion-input>

        </ion-item>

        <ion-item>

          <ion-label>Zip Code</ion-label>

          <ion-input type="text" [(ngModel)]="user.address.zipcode"
name="zipcode"></ion-input>

        </ion-item>

        <ion-item>

          <ion-label>Geo</ion-label>

          <ion-input type="text" [(ngModel)]="user.address.geo.lat"
name="lat" placeholder="Latitude"></ion-input>

          <ion-input type="text" [(ngModel)]="user.address.geo.lng"
name="lng" placeholder="Longitude"></ion-input>

        </ion-item>

        <ion-item-divider color="light">Company</ion-item-divider>

        <ion-item>

          <ion-label>Name</ion-label>

          <ion-input type="text" [(ngModel)]="user.company.name"
name="name"></ion-input>

        </ion-item>

        <ion-item>

          <ion-label>Business</ion-label>

          <ion-input type="text" [(ngModel)]="user.company.bs"
name="bs"></ion-input>

        </ion-item>

        <ion-item>

          <ion-label>Catch Phrase</ion-label>
```

```
          <ion-input type="text" [(ngModel)]="user.company.catchPhrase"
name="catchPhrase"></ion-input>

        </ion-item>

        <button ion-button (click)="saveUser()" round>Add User</button>


    </ion-content>
```

Fim código 5

Observe no código 5 que a página editusers.html utiliza as tags <ion-item> e <ion-input> para apresentar um conjunto de campos para entrada de dados do usuário e utiliza a diretiva ngModel para fazer a transferência de valores das variáveis entre a View e o Controlador.

Verifique se todos os arquivos alterados estão salvos e se não ocorreram erros de código. Para testar o aplicativo, abra o terminal ou o prompt de comando e execute a aplicação com o comando ionic serve.

Após todas as alterações de código, a página home.html é apresentada conforme demonstra a Figura 1.

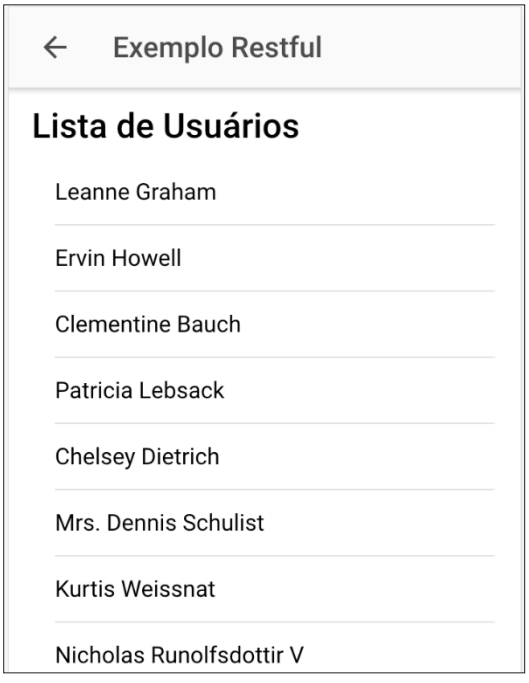


Figura 1 *Lista de Usuários do Web service*.

Clique em uma linha da lista e os dados do usuário serão mostrados em um formulário, conforme demonstra a Figura 2.

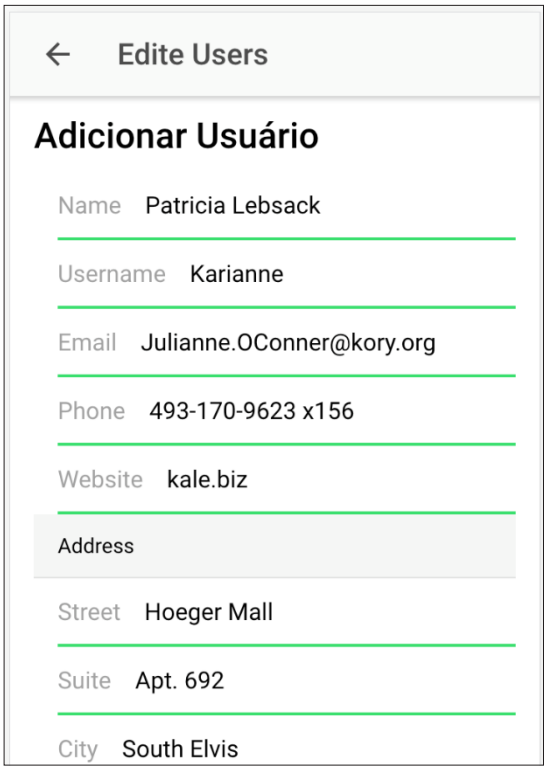


Figura 2 *Dados do Usuários no formulário*.

Concluímos nossos estudos com a criação de um projeto Ionic. Esperamos que tenha compreendido todos os conteúdos abordados nesta unidade e sugerimos que retome os conteúdos estudados sempre que surgirem dúvidas. Agora, realize as questões autoavaliativas para testar seus conhecimentos.

4. QUESTÕES AUTOAVALIATIVAS

Revise o conteúdo estudado e realize as questões propostas para fixar o seu aprendizado.

1) Em aplicações que fornecem serviços web por meio da tecnologia REST, qual comando do protocolo HTTP é utilizado para se realizar atualização de dados?
   a) DELETE
   b) POST
   c) GET
   d) PUT
   e) UPDATE

2) Qual o nome do formato de dados baseado na notação Javascript e de padrão aberto, criado para transmitir objetos de dados na internet?
   a) XML
   b) HTML
   c) MIME
   d) JSON
   e) XHTML

Gabarito

Confira, a seguir, as respostas corretas para as questões autoavaliativas propostas:

1) Alternativa D.

2) Alternativa D.

5. CONSIDERAÇÕES

Chegamos ao final da quarta unidade e com isso concluímos os estudos introdutórios relacionados ao desenvolvimento de aplicações móveis com Ionic.

Nesta unidade, apresentamos conceitos fundamentais para a utilização de Serviços Web baseados na tecnologia REST, e você pôde acompanhar e implementar um exemplo prático de aplicativo que utiliza essa tecnologia, com isso pôde entender as vantagens e as desvantagens dessa estratégia de armazenamento externo e comparar com as estratégias de armazenamento interno abordadas na Unidade 3.

É fundamental que você dê continuidade em seus estudos e experimente construir uma aplicação que combine as duas estratégias, ou seja, armazenar e gerenciar dados internamente e na nuvem. Isso ajudará a ampliar seus conhecimentos em uma área muito pesquisada no meio acadêmico, que envolve o sincronismo de dados entre sistemas de armazenamento.

Outros exemplos de aplicação são aqueles que utilizam os sensores e a câmera do dispositivo. Geralmente, as aplicações que utilizam esses recursos são aplicativos de jogos, de mapas etc. Existem muitos exemplos e tutoriais de aplicações para Ionic nessas áreas disponíveis na internet, basta uma pesquisa para encontrá-los.

Esperamos que você tenha aproveitado os estudos abordados neste material e que os conteúdos aqui apresentados sejam úteis na construção do seu conhecimento. Desejamos que você tenha sucesso em sua carreira profissional.

6. E-REFERÊNCIAS

JSONPlaceholder. REST API for Testing and Prototyping. Disponível em: <https://jsonplaceholder.typicode.com/>. Acesso em: 17 nov. 2017.

W3Schools. Tutoriais para estudo e treinamento JSON. Disponível em: <https://www.w3schools.com/js/js_json_intro.asp>. Acesso em: 27 ago. 2018.

W3Schools. Tutoriais para estudo e treinamento XML. Disponível em: <https://www.w3schools.com/xml/default.asp>. Acesso em: 27 ago. 2018.