Sintaxe e componentes do React Native

Apresentação dos conceitos, das técnicas e das ferramentas ligadas ao desenvolvimento de aplicativos mobile para a plataforma Android com o uso do framework React Native.

Prof. Alexandre de Oliveira Paixão

1. Itens iniciais

Propósito

Compreender as especificidades da programação para dispositivos móveis nos sistemas Android mediante a utilização da biblioteca React Native, baseada, por sua vez, na sintaxe de uma linguagem popular, como o JavaScript.

Preparação

Para acompanhar o conteúdo e a codificação dos exemplos a serem apresentados ao longo deste estudo, você terá de utilizar uma IDE (sigla de *Integrated Development Environment*), sendo recomendado, para tal, o software gratuito Visual Studio Code. Além disso, você precisa configurar o ambiente de desenvolvimento e de testes no qual diferentes configurações e ferramentas podem ser usadas. Para mais detalhes a respeito dessa etapa, visite o site oficial do React Native.

Objetivos

- Definir o ambiente de desenvolvimento com algumas de suas possíveis configurações.
- Listar os componentes nativos do React Native.
- Esquematizar a depuração de aplicativos.

A crescente utilização de smartphones para a realização de tarefas anteriormente restritas a computadores fomentou um novo e vasto mercado: o de aplicativos para dispositivos móveis. Tal mercado possui características próprias e distintas, como a existência de diferentes sistemas operacionais utilizados para diversos dispositivos com variadas configurações de hardware, além da própria mobilidade, em que nem sempre estão disponíveis conexões de internet de alta velocidade.

Essas características trazem grandes desafios para os profissionais que atuam no desenvolvimento de aplicativos capazes de atender às mais diversas necessidades. Desse modo, a programação para dispositivos móveis tem recebido cada vez mais atenção, com a criação de novas bibliotecas e ferramentas para apoiar seu processo.

Ao longo deste conteúdo, apresentaremos, de maneira introdutória, uma das principais bibliotecas Javascript utilizadas atualmente no desenvolvimento mobile: a React Native. Sua principal característica é possibilitar a criação de aplicativos **multiplataformas**.

Conheceremos a sintaxe e os componentes do framework React Native para o desenvolvimento de aplicativos móveis, nos concentrando, para isso, naqueles que utilizam o sistema operacional Android. Falaremos sobre a configuração do ambiente de desenvolvimento e os conceitos da linguagem utilizada no framework, assim como algumas de suas características — incluindo seus principais componentes. Por fim, exploraremos a codificação e a depuração de aplicativos.

Multiplataformas

Em linhas gerais, o termo “multiplataforma”, no desenvolvimento mobile, se refere a aplicações que compartilham um mesmo código-fonte e que podem ser executadas em diferentes sistemas operacionais — nesse caso, Android e iOS.

Contextualizando

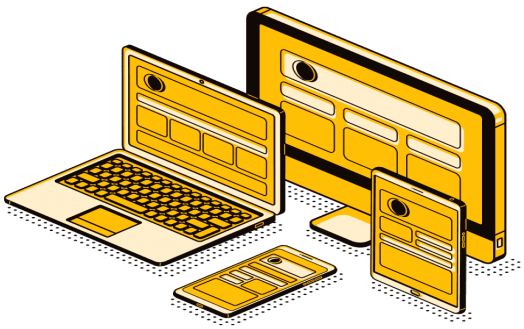
Como de praxe, no início do estudo de uma linguagem de programação ou do processo de desenvolvimento de um software/aplicativo, precisamos configurar nosso ambiente de desenvolvimento. Entretanto, antes disso, devemos introduzir alguns conceitos relacionados ao framework React Native.

O React Native é uma biblioteca Javascript integrante do ecossistema de outro framework, o React.js. Apesar de ambos terem sido criados pelo Facebook e compartilharem algumas semelhanças, eles possuem utilizações distintas. Veja!

React.js	React Native
É uma biblioteca voltada para o desenvolvimento web. Sua principal finalidade é simplificar o processo de confecção de interfaces ricas e responsivas. Além disso, os componentes gerados pelo React podem ser utilizados em qualquer plataforma.	É uma biblioteca voltada para o desenvolvimento mobile, cuja principal característica é possibilitar o desenvolvimento híbrido, ou seja, permitir que um único código rode em diferentes plataformas, como Android e iOS.

Após essa introdução, já estamos prontos para o próximo passo. Nesse ponto, além de escolher o editor a ser utilizado, deve-se configurar e instalar ferramentas adicionais, como bibliotecas e até mesmo, dependendo do caso, servidores de aplicação.

No ambiente mobile, algumas etapas extras se fazem necessárias — sobretudo no que tange à forma como os aplicativos desenvolvidos serão testados. É possível utilizar desde **emuladores** (softwares instalados em nosso computador que fazem o papel de um dispositivo móvel) até dispositivos móveis propriamente ditos, como os smartphones.



Para a biblioteca React Native, há ainda outra possibilidade: o Expo, um framework que permite a codificação e o teste de aplicativos de forma bastante simples. Embora possua algumas limitações, essa opção é interessante, já que consome menos recursos de hardware em relação às demais opções citadas.

Utilização do Expo

Veremos agora como definir e configurar nosso ambiente de desenvolvimento para podermos programar em React Native. A forma mais simples de se desenvolver nele é mediante a utilização do Expo.

> O Expo é um framework e uma plataforma composta por um conjunto de ferramentas e serviços que facilita as tarefas de desenvolvimento, construção e implantação de aplicativos Android, iOS e web. Ele possui como base um mesmo código JavaScript/TypeScript.

O primeiro passo para se poder usar o Expo — e que também é o ponto de partida de tudo relacionado ao React Native, como veremos em breve — consiste na instalação de um **gerenciador de pacotes**. Entre suas principais opções, destacam-se o NPM e o Yarn. No entanto, falaremos ainda de outro gerenciador: o **NODE.JS**.

Dica

Você pode escolher o gerenciador de sua preferência, uma vez que ambos desempenham o mesmo papel.

NPM

O NPM (de *node package manager*) é um gerenciador de pacotes lançado no biênio 2009-2010. Tal pacote faz parte da instalação padrão do ambiente de execução da linguagem JavaScript Node.js, sendo ambos instalados de forma conjunta.

O NPM possui três componentes. Vamos conhecê-los!

Site
É possível acessar nele sua documentação, além de pesquisar e navegar pelas diversas bibliotecas (*packages*) disponíveis.

CLI
Do inglês *command line interface* ou interface de linha de comando, o CLI possibilita a execução de comandos por meio de um terminal. O CMD no Windows ou o bash no Linux é um exemplo disso.

Repositório aberto
Local onde os pacotes (bibliotecas) ficam armazenados.

Curiosidade

A empresa chamada npm, Inc., além de mantenedora do framework, desenvolve soluções para o mercado empresarial.

Por fim, um ponto importantíssimo desse gerenciador de pacotes é o “package.json”. Trata-se de um arquivo por meio do qual o NPM armazena:

- As configurações do projeto.
- Alguns comandos a serem executados.
- Uma lista de dependências (bibliotecas externas) utilizadas no projeto.

A partir do arquivo “package.json” e da CLI NPM, é possível instalar todas as dependências de um projeto.

Um cenário muito comum é disponibilizar — diretamente ou por meio de um repositório/versionador — apenas os códigos-fonte do aplicativo ao lado do arquivo de configuração. Aliás, essa é a opção indicada, uma vez que os pacotes usados no projeto podem consumir bastante espaço em disco. Com isso, bastará a quem for utilizar nosso código baixar o projeto e executar o comando “npm install” para que todas as dependências sejam instaladas e o aplicativo esteja funcional.

YARN

O YARN (sigla de *yet another resource negotiator*) foi lançado em 2016 pelo Facebook com outras empresas — entre elas, a Google. Sua criação teve como premissa resolver alguns problemas de segurança existentes no NPM à época, além de tornar mais rápido o processo de instalação de dependências.

> Outra característica própria do YARN é a forma como a gestão de dependências é realizada: por intermédio de um arquivo de lock denominado yarn.lock, é guardada a versão exata de cada dependência, garantindo, assim, uma igualdade em todas as instalações.

Por mais que o NPM atualmente também dê suporte a tal parametrização, o YARN faz isso de forma automática. Na comparação entre ambos, alguns *benchmarks* apontam diferenças, vantagens e desvantagens de um em relação ao outro.

No final das contas — e como é bastante comum em ferramentas “concorrentes” —, cada nova atualização deixa ambos muito parecidos. O mais importante, nesse caso, é que os desenvolvedores têm em mãos duas excelentes alternativas para realizar a tarefa de gestão de dependências.

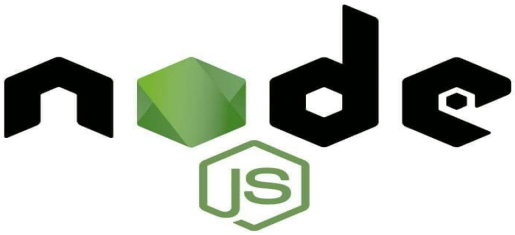
Tanto o NPM quanto o YARN cumprem a mesma função: gerenciar a instalação de dependências de um projeto React Native. Embora isso se dê com processos diferentes, ambos utilizam o arquivo “**package.json**” para anotar as dependências e suas versões, além de baixarem e salvarem as dependências/bibliotecas na pasta “**node_modules**”.

Atenção

Na escolha do gerenciador é fundamental usar apenas um deles no projeto. Isso evita comportamentos inadequados e/ou até mesmo falhas e conflitos de dependências. Ao longo deste estudo, utilzaremos o NPM.

NODE.JS

Antes de voltarmos a falar do Expo e de vermos como instalá-lo, precisamos tratar de outro assunto: o Node.js. O Node pode ser definido como um ambiente *server-side* para a execução de códigos escritos utilizando a linguagem JavaScript. Com ele, é possível criar quaisquer tipos de aplicações no back-end utilizando uma linguagem até então restrita ao front-end, desde servidores web, estáticos ou dinâmicos até robustas APIs ou softwares baseados em microsserviços.



Logo do NODE.JS.

A importância do Node em relação à abordagem de nosso conteúdo se dá pelo fato de ele ser um dos requisitos necessários para criarmos nossas aplicações com o React Native. Afinal, precisamos utilizar o NPM para isso — e ele faz parte ou é instalado com o Node.

Instalação

O primeiro passo é instalar o Node.js, tendo consequentemente o NPM instalado. Ele pode ser instalado de duas maneiras:

- Instaladores (disponíveis para os sistemas operacionais Windows e Linux).
- NVM (de *node version manager*).

Conforme consta na documentação oficial do Node, deve-se dar preferência à segunda forma, já que os instaladores costumam armazenar os arquivos do NPM em diretórios com permissões apenas locais, e isso poderá causar alguns erros quando quisermos instalar e utilizar **pacotes a nível global**.

Pacotes a nível global

As ferramentas de gerenciamento de dependências permitem a instalação de pacotes/dependências de maneira local ou global. A local limita os pacotes aos projetos nos quais foram instalados, enquanto a global permite sua utilização em todos os projetos.

Saiba mais

Mais detalhes sobre o processo de instalação do Node.js e do NPM podem ser encontrados na página oficial do projeto.

Após realizar as instalações do Node.js e do NPM, certifique-se de que ambos já estão rodando e disponíveis por meio das seguintes linhas de comando no terminal, onde trabalharemos a maior parte do tempo nas tarefas de instalação, *build* e *deploy*:

- CMD (no Windows)

- Bash (no Linux)

  Caso o processo de instalação tenha sido completado com sucesso, estes comandos retornam as versões instaladas:

```
javascript

node -v
npm -v
```

Com o NPM instalado, já podemos realizar a instalação do Expo. Para isso, ainda no terminal, digite o comando a seguir:

```
javascript

npm install -g expo-cli
```

Antes de prosseguirmos, temos de fazer algumas observações sobre o comando *npm install -g expo-cli*. Em linhas gerais, trata-se da sintaxe de instalação de pacotes (dependências) utilizando o NPM. Inicialmente, temos, na ordem, os seguintes elementos:

1. O executável, ou seja, "*npm*".

2. A instrução "*install*".

3. O nome do pacote a ser instalado (em nosso caso, "*expo-cli*").

Já a opção "*-g*" indica que esse pacote será instalado de forma global.

Dica

Ao utilizar o terminal, observe as questões de permissão em seu sistema operacional. Pode haver a necessidade de executar o terminal ou os comandos com permissões de administrador.

Teste

Após a execução do comando de instalação — e não tendo ocorrido nenhum erro —, já temos a interface cliente do Expo disponível em nosso computador. A partir disso, já poderemos criar nossa primeira aplicação. Para isso, faça o seguinte:

1. Crie uma pasta em seu computador para armazenar os códigos-fonte de suas aplicações.

2. Navegue, utilizando o terminal, até a raiz dessa pasta.

3. Para criar uma aplicação, digite no terminal: expo init nome_da_aplicacao.

4. Serão apresentados os templates disponíveis para a criação da aplicação. Escolha a primeira opção, “blank”, e tecle Enter.

Ao final do processo, uma pasta com o nome definido para a aplicação será criada e, dentro dela, a estrutura básica do projeto. Ainda no terminal, navegue para dentro da pasta do projeto. Em seguida, para iniciar a aplicação, digite o comando:

```
javascript

npm start
```

O comando **npm start** abrirá automaticamente uma janela do navegador padrão de sua máquina. Com o Expo, você tem a opção de visualizar no próprio navegador. Outra maneira de fazer isso é instalar no seu smartphone o Expo Client e escanear o QR Code disponível na página aberta no navegador para executar o aplicativo em seu dispositivo móvel.

Atenção

É imprescindível que seu dispositivo móvel esteja na mesma rede em que o Expo foi inicializado. Em alguns casos, é possível resolver alguns problemas de conexão alterando a opção Connection para Tunnel nas configurações disponíveis, logo acima do QR Code, na página aberta no navegador.

Utilização de um dispositivo virtual ou físico

Embora o Expo seja uma excelente opção, sobretudo para iniciarmos nossos passos no desenvolvimento mobile, ele ainda possui algumas limitações. A principal delas é a impossibilidade de incluir módulos e componentes nativos (Android ou iOS) nos aplicativos. Nesse caso, há outra opção para configurar o ambiente, que, além de mais robusta, é mais pesada em termos de requisitos de hardware.

Essa opção consiste — nos limitando ao desenvolvimento de apps Android — na instalação de um **emulador**. Disponível com o **Android Studio**, ele permite a utilização dessa ferramenta para emular virtualmente um dispositivo móvel ou até mesmo usar um real por meio de uma conexão USB com o computador.

Android Studio

IDE a partir da qual é possível desenvolver aplicativos para o S.O. Android. O Android Studio conta com uma série de ferramentas, como o editor de layout e o emulador.

Instalação das dependências

Como dissemos anteriormente, para a construção de aplicações que utilizam um código nativo, é preciso configurar o ambiente mediante a instalação dos seguintes softwares adicionais:

- Node.js

- React Native CLI
- Java JDK
- Android Studio

As indicações para a instalação do Node já foram vistas quando tratamos do Expo. Em relação ao React Native CLI, é recomendado utilizar o NPX (executor de pacotes do NPM) já instalado com o Node.js em vez de realizar uma instalação global dele.

Java JDK e Android Studio

Há diferentes formas de instalar esses dois softwares. Além de baixar os instaladores nos respectivos sites oficiais, é possível fazer essa instalação por intermédio de gerenciadores de software, como o Chocolatey, no Windows, ou de gerenciadores nativos próprios das diferentes distribuições, como o Linux, por exemplo.

Comentário

Como esse processo pode variar muito, já que depende das escolhas realizadas, assim como das diferentes versões de sistemas operacionais de cada usuário, entre outras particularidades, não o abordaremos em detalhes neste conteúdo. Recomenda-se, nesse caso, a leitura do site oficial do React Native ou a consulta de fontes alternativas, como vídeos ou tutoriais disponíveis na internet.

Configuração das variáveis de ambiente

Além da instalação das dependências citadas, será necessário realizar a configuração de algumas variáveis de ambiente. Tal processo, muito comum no desenvolvimento de softwares mediante a utilização de algumas linguagens de programação, consiste na inserção de variáveis e de seus respectivos valores nas configurações de nosso sistema operacional.

Isso permite, por exemplo, o acesso a executáveis e outros recursos dos softwares instalados a partir de qualquer ponto em nosso sistema operacional, não ficando restritos às pastas em que foram instalados.

Em nosso caso, devemos configurar as seguintes variáveis de ambiente:

JAVA_HOME

O valor deve ser o endereço da pasta JDK do Java.

ANDROID_HOME

O valor deve ser o endereço da pasta SDK do Android Studio.

Path

O valor deve ser o endereço da pasta platform-tools do Android Studio Endereço da pasta bin do Java JDK.

É importante destacar que, as configurações citadas podem variar de acordo com o processo de instalação adotado na etapa anterior e as respectivas particularidades do ambiente de cada usuário.

Teste do ambiente

Após o cumprimento dos passos descritos anteriormente, chegou a hora de testar o ambiente utilizando um dispositivo virtual ou físico. Para isso, criaremos um projeto por meio do React Native CLI. Estabeleceremos a seguir o passo a passo da criação desse projeto:

1. Crie/utilize uma pasta em seu computador para armazenar os códigos-fonte de suas aplicações.

2. Navegue, utilizando o terminal, até a raiz dessa pasta.

3. Para criar uma aplicação, digite no terminal: npx react-native init nome_da_aplicacao.

Dica

Há outras formas, além da apresentada, de criar um aplicativo React Native, por exemplo, definindo uma versão específica da biblioteca ou determinado template.

Ao final do processo de download do template e instalação das dependências básicas, nosso projeto estará pronto para ser testado. Nesse ponto, será necessário realizar alguns passos adicionais de acordo com a forma escolhida para testar o aplicativo (por meio de um dispositivo virtual ou físico). O recomendado é seguir os passos indicados na documentação oficial do React Native ou em fontes alternativas.

Após ter configurado o dispositivo e estando no terminal, acesse a pasta do projeto criado. Para iniciá-lo, digite o seguinte comando:

```
javascript

npx react-native run-android
```

O comando apresentado realizará o build da aplicação e a abrirá no dispositivo:

Físico	Virtual
Caso você tenha escolhido essa opção.	Um software emula a tela de um smartphone em seu computador.

IDE

Há várias opções disponíveis de ferramenta para a edição/confecção do código. Elas variam desde as mais simples, como os editores (entre eles, o Notepad++), até as IDEs, como o VS Code. Todas essas opções são gratuitas.

> A IDE é a mais recomendada, sobretudo pela gama de plug-ins que o VS Code possui. Voltados para o desenvolvimento em React Native, eles facilitam e agilizam todo o processo de desenvolvimento.

Além dessas opções, existem outras gratuitas e comerciais. Nesse ponto, caso você não tenha nenhuma preferência pessoal, comece utilizando o VS Code e seus plug-ins voltados para o React e o React Native.

Primeiros passos no React Native

Após termos realizado a configuração de nosso ambiente com o Expo CLI ou o React Native CLI, estamos prontos para iniciar o processo de desenvolvimento. Você pode dar seus próximos passos ao analisar a estrutura de pastas criadas por default nas aplicações usadas como teste e até mesmo modificando o código gerado inicialmente (para isso, edite o arquivo App.js).

Neste vídeo, nosso especialista irá apresentar o ambiente necessário para programação em React Native, com uma introdução à sua sintaxe.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Os gerenciadores de pacotes/dependências possuem um importante papel no desenvolvimento de aplicativos. Com base nisso, escolha a alternativa correta.

A Embora os gerenciadores de pacote sejam importantes, existe a opção de realizar as tarefas desempenhas por eles de forma manual. Isso nos permite um maior controle sobre o que é instalado em nossos ambientes de desenvolvimento.

B Os gerenciadores de pacote para desenvolvimento na plataforma Android só estão disponíveis nativamente no sistema operacional Linux. No Windows, é preciso instalar o Java para ter acesso aos gerenciadores de pacote.

C Os gerenciadores facilitam a gestão (instalação, controle de versão etc.) de pacotes. Entretanto, em relação ao React Native, realizar a instalação deles se torna uma tarefa complicada independentemente do sistema operacional utilizado, já que há apenas um gerenciador disponível, assim como uma única forma de instalação.

D Os gerenciadores NPM e YARN desempenham o mesmo papel. Logo, não havendo nenhum tipo de restrição ou recomendação específica inerente a determinado projeto específico, o desenvolvedor poderá optar por um dos dois. O único cuidado é evitar o uso de ambos num mesmo projeto.

E O YARN, lançado posteriormente ao NPM, além de mais seguro, conta com uma variedade maior de pacotes disponíveis.

A alternativa D está correta.

O NPM e o YARN são gerenciadores de pacotes que cumprem a mesma função, além de possuírem uma vasta base de pacotes disponíveis e ferramentas semelhantes, como o cliente para a utilização via terminal. Embora possuam pequenas diferenças em termos de busca e indexação de pacotes, entre outras, ambos, no final, cumprem o mesmo propósito e com a mesma eficiência.

Questão 2

Um dos problemas no desenvolvimento mobile, no que tange ao ambiente de trabalho, é o alto consumo de recursos de hardware. Uma alternativa para essa situação é utilizar o framework Expo. A respeito dele, marque a alternativa correta.

A O Expo é um framework que, para ser utilizado, depende da biblioteca React Native. Logo, para que possa ser usado, é necessário instalar primeiramente o React Native por intermédio de um gerenciador de pacotes.

B O Expo é uma ferramenta que facilita o desenvolvimento e o teste de aplicações escritas em React Native. Sua principal limitação é não permitir a utilização dos componentes nativos do React Native.

C O Expo é um framework desenvolvido pelo Facebook. Ele facilita o desenvolvimento de aplicativos em React Native, mas sua principal função é resolver problemas contidos no NPM.

D O Expo restringe as funcionalidades disponíveis para a plataforma Android. Ou seja, com ele, não é possível desenvolver ou testar o código React Native para a plataforma iOS.

E Com o Expo CLI, é possível desenvolver aplicações da mesma forma que é feita com o React Native CLI. Ou seja, pode-se acessar os mesmos recursos e funcionalidades, exceto a câmera e o microfone do dispositivo que executa a aplicação criada.

A alternativa B está correta.

O Expo fornece uma série de vantagens — principalmente no início do aprendizado de desenvolvimento mobile. Entre elas, destaca-se a facilidade de instalação, de uso e de acesso a recursos, como API e hardware do dispositivo no qual a aplicação está rodando, microfone, câmera e player de música, entre outros. Por outro lado, a principal desvantagem de sua utilização é não poder acessar os componentes nativos de cada plataforma — no caso, Android e iOS.

Componentes

Na engenharia de software, alguns conceitos são muito utilizados para se definir o que são os componentes. Tais conceitos se referem tanto aos aspectos mais técnicos quanto aos mais práticos.

Tomando como base essa segunda abordagem, ou seja, de ordem prática, podemos enxergar os **componentes** como insumos, artefatos ou simplesmente "coisas" que facilitam o processo de desenvolvimento, uma vez que eles tornam dispensável que uma única pessoa, equipe ou até mesmo empresa (de software) tenha de desenvolver todas as "peças" do software que está escrevendo ou que tenha que começar todo projeto do zero.

Isso ocorre por conta de uma das principais características de um componente: ser integrado por pequenos pedaços de software que desempenham uma função (ou poucas funções) específica.

Ao pensarmos na codificação de um software com base em componentes, devemos ter em mente que, em vez de sempre fazermos a construção, podemos realizar também a composição. Isto é, podemos construir pequenos pedaços de código (os componentes), os quais, quando reunidos, formarão o software como um todo. Tais princípios se aplicam a situações nas quais são desenvolvidas tanto as aplicações de back-end quanto as aplicações e/ou os aplicativos de front-end.

Tendo isso em mente, veremos a seguir alguns dos **componentes nativos Android** disponibilizados pelo framework React Native.

JSX

De maneira simples e, ao mesmo tempo, completa, podemos inicialmente definir o JSX (sigla de JavaScript XML) como uma sintaxe de extensão da **linguagem JavaScript** bastante familiar da **linguagem de marcação XML**.

Os componentes disponíveis em React Native são escritos utilizando JSX.

Recomendação

Além de configurar uma escolha natural, você pode utilizar o JSX na construção dos componentes React ou React Native.

Aprofundando um pouco os conceitos, o JSX também é conhecido como JavaScript XML. Extensão semelhante ao XML para a especificação ECMAScript, ele combina a lógica de componentes (JavaScript) e o mark-up (DOM/modelo de objeto de documento ou Native UI/interface de usuário Nativa) em um único arquivo/código.

Este fragmento de código mostra a forma de um elemento JSX:

```
javascript

var element = (




);
```

Sintaxe

Em termos de sintaxe, a especificação JSX define que:

- Os elementos JSX podem ser “self-opening” <JSXElement></JSXElement> ou “self-closing” <JSXElement />.
- Os atributos podem ser declarados como uma expressão <Component attr={atributo}> ou um string <Component attr=”atributo”>.
- Os elementos filhos podem ser textos, expressões ou elementos.

Funcionamento das aplicações escritas com React Native

Na escrita de um aplicativo fazendo uso do framework React Native, opta-se por um desenvolvimento chamado de híbrido. Ou seja, um único código será compilado para poder rodar em dispositivos Android ou iOS.

Isso é possível pelo fato de os dispositivos possuírem dois núcleos. Vejamos!

Núcelo nativo

De acordo com a linguagem nativa de cada sistema operacional (Java ou Kotlin para Android e Swift ou Objective-C para iOS).

Núcleo JavaScript

Chamado de JavaScriptCore.

Com isso, o que o **React** faz é compilar (na verdade, transpilar, otimizar e minificar) um aplicativo-base, o qual, rodando no JavaScriptCore, acessará os componentes nativos de cada S.O. Agora, entenda a diferença entre transpilar e minificar!

Transpilar

O processo de transpilação é bastante parecido com o de compilação. A diferença é que, na transpilação, o resultado do processo não é um código de mais baixo nível, e sim um código com uma linguagem de alto nível, sendo normalmente diferente daquela na qual o software foi construído.

Minificar

O processo de minificar um código-fonte é muito comum em linguagens que rodam no lado cliente, como o JS e o CSS, por exemplo. Tal processo consiste em reduzir o tamanho final do código-fonte, removendo os espaços e as linhas e diminuindo o comprimento dos nomes das variáveis e das funções, além de outras funções.

Componentes nativos

Um dos principais pilares — provavelmente o principal — do React Native é a utilização de componentes, ou seja, **coleções de dados** e **elementos de interface gráfica** (UI *elements*) que compõem as views e, de forma geral, os aplicativos em si. Embora exista a flexibilidade de desenvolver os próprios componentes customizados, o framework React Native já disponibiliza, no momento da instalação, uma série de componentes chamados de **componentes nativos** (*native components*). Outro conceito associado a ele é o de *core components*.

No desenvolvimento específico para Android e iOS, as views são construídas utilizando respectivamente o Kotlin (ou Java) e o Swift (ou Objective-C). Graças ao framework React Native, é possível invocar essas views por meio dos componentes React escritos com JavaScript.

Em React, os componentes são escritos utilizando o JSX e estão agrupados em diferentes categorias. Os elementos principais estão destacados na tabela a seguir. Os componentes correspondentes em cada tecnologia que constam nela, por sua vez, serão descritos em detalhes. Confira!

Componente UI React Native	Componente Android	Componente iOS	Elemento HTML
<View>	<ViewGroup>	<UIView>	
<Text>	<TextView>	<UITextView>	<p>
<Image>	<ImageView>	<UIImageView>	<img>
<TextInput>	<EditText>	<UITextField>	<input type=”text”>
<ScrollView>	<ScrollView>	<UIScrollView>	

Tabela: Componentes.
Alexandre de Oliveira Paixão.

View

A View é o principal componente na construção de uma **interface gráfica de usuário** (GUI, do inglês *graphical use interface*). Esse componente se relacionará diretamente com seu equivalente nas plataformas em que o aplicativo React estiver rodando (veja a tabela anterior). Em termos de organização do layout, ele pode ser utilizado de forma aninhada com outras views, podendo ainda ter como filhos elementos de qualquer tipo.

O fragmento de código adiante demonstra, de forma simples, a utilização de uma view como contêiner de outra view e de um elemento Text:

```
javascript

import React from "react";
import { View, Text } from "react-native";

const ViewExemplo = () => {
  return (


          Olá, mundo!

  );
};

export default ViewExemplo;
```

O componente View ainda possui vários atributos, além de poder “ouvir e responder” a alguns eventos.

Saiba mais

Consulte a documentação do React Native para conhecer mais detalhes sobre ele.

Text

Este componente é utilizado para a **apresentação de textos**. Ele suporta aninhamento, estilização e manuseio de toque.

O exemplo a seguir mostra a utilização aninhada de dois elementos Text. Além disso, nesse exemplo, o componente é estilizado com uso do StyleSheet:

```javascript
import React, { useState } from "react";
import { Text, StyleSheet } from "react-native";

const TextoAninhado = () => {
  const [titulo, setTitulo] = useState("Texto do elemento filho");

  const modificaTexto = () => {
        setTitulo("Esse texto está sendo exibido pois o primeiro elemento de texto foi
pressionado/tocado");
  };

  return (


                {titulo}
                {"\n"}
                {"\n"}


  );
};

const styles = StyleSheet.create({
  baseText: {
        fontFamily: "Verdana",
        marginTop:50,
        marginLeft:10
  },
  titulo: {
        marginTop:10,
        fontSize: 18,
        fontWeight: "bold"
  }
});

export default TextoAninhado;
```

Recomendação

Embora seja possível utilizar estilos inline, como ocorreu no exemplo visto na View, é recomendado, sempre que possível, dar preferência ao StyleSheet, já que sua utilização facilita a separação e a leitura do código, tornando-o mais fluido.

Da mesma forma que a observada no código anterior — e isso vale para todos os códigos utilizados como exemplo —, pode-se copiar o código acima e rodá-lo (no Expo ou no dispositivo virtual ou físico) para ver o Text funcionando na prática. Repare que, além desse elemento, novas funcionalidades do React são introduzidas a cada exemplo.

Nesse último, o destaque fica por conta do evento “onPress”. Ele demonstra que, ao se tocar no texto inicialmente exibido, um novo é carregado em seu lugar. Experimente realizar outras modificações no código, isso vai ajudá-lo a compreender melhor o comportamento de cada componente.

Image

Assim como a tag HTML <img>, este componente permite a exibição de diferentes tipos de imagens com origens distintas — e aqui o destaque fica por conta da possibilidade de utilização até mesmo das imagens armazenadas no próprio dispositivo móvel. O Image herda as propriedades do componente View, além de possuir uma série de outros atributos. Vejamos um exemplo de sua utilização!

```
javascript

import React from 'react';
import { View, Image, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
        paddingTop: 50,
  },
  imagem: {
        width: 50,
        height: 50,
        alignSelf: 'center'
  }
});

const ComponenteSimplesImage = () => {
  return (



  );
}

export default ComponenteSimplesImage;
```

TextInput

Este componente permite a entrada de textos por meio do teclado, provendo ainda uma série de **funcionalidades**, por exemplo, autocorreção, autocapitalização e utilização de diferentes tipos de teclado, assim como apenas do teclado numérico (digite algum texto no segundo input no exemplo). Observemos um exemplo simples de TextInput!

```
javascript

import React from "react";
import { SafeAreaView, StyleSheet, TextInput } from "react-native";

const MeuTextInput = () => {
  const [texto, setTexto] = React.useState(null);
  const [numero, setNumero] = React.useState(0);

  return (




  );
};

const styles = StyleSheet.create({
  meutextinput: {
        marginTop:100,
        height: 40,
        margin: 12,
        borderWidth: 1,
  },
});

export default MeuTextInput;
```

Em relação às suas **propriedades**, **atributos** e **eventos**, destacam-se dois eventos disponíveis (e muito utilizados quando trabalhamos com formulários): **focus** e **blur**. Ambos ocorrem quando o elemento respectivamente ganha e perde foco, ou seja, quando o cursor fica sobre eles e sai. Eles são úteis para validar informações inseridas ou aplicar máscaras nos valores digitados, como números de telefone e CPF.

ScrollView

Este componente também é um contêiner, sendo, a exemplo da View, utilizado para armazenar conteúdo — e outros elementos —, permitindo a interação na tela por **meio de rolagem** (*scrolling*). Logo, o ScrollView, para funcionar corretamente, precisa ter uma altura limitada/definida, já que sua serventia é justamente conter elementos filhos com altura ilimitada. Teste o código a seguir, modificando o tamanho do texto (aumentando-o e o diminuindo) a fim de visualizar, na prática, como tal componente se comporta:

javascript

```
import React from 'react';
import { StyleSheet, Text, SafeAreaView, ScrollView, StatusBar } from 'react-native';

const Lista = () => {
  return (



                  Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                                        Pellentesque id dui sed nulla imperdiet
scelerisque.
                                        Integer malesuada facilisis nibh varius eleifend.
                                        Cras a velit laoreet dui interdum consectetur.
                                        Pellentesque volutpat placerat mauris in interdum.
                                        Pellentesque non egestas sem. Suspendisse
malesuada at augue
                                        sit amet pretium.
                                        Praesent odio nisl, semper vitae purus a,
elementum ultrices arcu.
                                        Praesent blandit lectus et aliquet posuere.
                                        Nulla dictum, nisi id feugiat suscipit, mi sem
maximus turpis,
                                        vel aliquet massa ex sit amet sem.
                                        Sed ullamcorper enim non elit vestibulum, feugiat
euismod elit
                                        consectetur. In et pulvinar eros.



  );
}

const styles = StyleSheet.create({
  safecontainer: {
        flex: 1,
        paddingTop: StatusBar.currentHeight,
  },
  containerScrollView: {
        backgroundColor: 'grey',
        marginHorizontal: 20,
  },
  text: {
        fontSize: 26,
  },
});

export default Lista;
```

Recomendação

Quando desenvolvemos uma aplicação que consome conteúdo externo por meio de uma API, por exemplo, nem sempre sabemos a quantidade de informações ou de elementos filhos que serão carregados. Nesse caso, por questões de melhor performance, devemos utilizar outro componente com funcionalidade semelhante à do ScrollView: a FlatList.

Outros componentes

Além daqueles já apresentados, o React Native possui outros componentes nativos. Apontaremos três deles a seguir:

- Button
- Switch
- FlatList

Saiba mais

Consulte a documentação oficial para obter mais detalhes sobre outros componentes. Pratique também seu conhecimento com os códigos apresentados neste material, combinando suas utilizações e gerando, com isso, interfaces mais ricas e completas.

Componentes do React Native

Neste vídeo, nosso especialista apresentará exemplos de componentes do React Native.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Um dos principais fatores que dificulta o desenvolvimento de aplicativos mobile é a diferença existente entre cada plataforma e seus respectivos sistemas operacionais. Logo, tais diferenças naturalmente fazem com que seja necessário escrever/repetir basicamente o mesmo código para atender a cada S.O. Uma solução para resolver tal problema seria

A fazer um movimento que envolva empresas e desenvolvedores para sensibilizar os fabricantes de dispositivos móveis a fim de que eles adotem uma plataforma em comum.

B escolher um sistema operacional e desenvolver aplicativos apenas para os dispositivos que o utilizem.

C no lugar de desenvolver aplicativos que utilizem componentes nativos, deve-se dar preferência ao desenvolvimento de websites ou de SPA (em inglês, *single-page application* ou, em português, aplicação web que roda em uma única página).

D utilizar bibliotecas que permitam a escrita de um único código-fonte, ficando ela responsável por transpilar o código criado a fim de que ele rode nos diferentes sistemas operacionais.

E desenvolver o mesmo aplicativo, replicando/duplicando o código-fonte para que ele seja compatível com cada sistema operacional.

A alternativa D está correta.

O processo de desenvolvimento de aplicativos pode se tornar muito custoso caso optemos por desenvolver um mesmo código diversas vezes, adaptando-o para que ele se adeque às particularidades de cada sistema operacional, rodando, assim, nos mais diversos dispositivos móveis. Além de custoso, esse processo também dificulta a manutenção do código e sua evolução. Desse modo, utilizar uma biblioteca que possibilite escrever um único código e rodá-lo em diferentes sistemas operacionais traz inúmeros benefícios.

Questão 2

O React Native possui uma série de componentes nativos, como, por exemplo, <View> e <Text>. Considerando o que vimos sobre esses componentes e analisando o fragmento de código a seguir, marque a alternativa correta quanto ao que será exibido na tela do dispositivo.

```
javascript

export default function App() {
                return (



                                                Change code in the editor and watch it change on
your phone! Save to get a
        shareable url.




                );
        }
```

A Será exibido um erro informando que a variável “Div” não existe.

B Será exibido, independentemente da plataforma do dispositivo, o texto contido pelo componente <Text>, uma vez que seu contêiner poderá ser um componente <View> ou o seu equivalente em HTML, como a <Div>.

C Em dispositivos Android, o texto será exibido corretamente, sem nenhum erro. Entretanto, isso não acontecerá em dispositivos iOS, já que essa plataforma não possui um componente nativo equivalente à <Div>.

D O código acima apresentará comportamento distinto se for executado no Expo ou em um dispositivo virtual/físico.

E Como o componente <Div> não faz parte dos componentes nativos do React Native, será preciso utilizar uma classe de estilos (StyleSheet) para a renderização correta do conteúdo do contêiner em questão.

A alternativa A está correta.

O React Native possui alguns componentes nativos que são transpilados para os componentes equivalentes em cada plataforma onde o aplicativo é executado. Além disso, pode-se criar os próprios componentes customizados. Por outro lado, caso se utilize um componente não existente entre os nativos ou que não tenha sido criado por nós mesmos (ou seja, um importado para o projeto), a aplicação retornará um erro, informando que o elemento em questão não existe.

Depuração

Uma depuração ou um debug (termo em inglês comumente utilizado na área de desenvolvimento de software) é o processo de identificar erros (bugs) ou problemas no código-fonte de um software.

Ao longo de tal processo, o código-fonte é inspecionado e analisado durante sua execução, a fim de que qualquer erro existente possa ser identificado e corrigido.

O processo de debug pode ser iniciado de duas formas. Confira!

Indiretamente

A partir de um erro gerado na execução do aplicativo.

Iniciativa do desenvolvedor

De maneira estruturada e previamente organizada, tal processo pretende testar o software em diferentes situações de uso.

Como depurar um software

Para depurar um software, normalmente é utilizada uma IDE e/ou, no caso de aplicações React Native (e aplicações web em geral), o próprio console do navegador ou suas bibliotecas/ferramentas adicionais. Tal processo consiste geralmente na observação, independentemente da ferramenta utilizada, de partes do código, como:

- Estado de objetos.
- Valores de variáveis ou propriedades.
- Validação de instruções condicionais ou laços de repetição.

Nesse processo, é possível, entre algumas opções, realizar as seguintes ações. Veja!

1

Criar pontos de observação (break points) nas IDEs

Um pouco mais rebuscada, essa forma pode variar a partir de recursos nativos e de plug-ins disponíveis na IDE utilizada.

2

Inserir instruções simples

Como, por exemplo, “console.log”. Um pouco menos refinada, ela também é útil em muitas situações.

Ferramentas de depuração de código

Há várias ferramentas e técnicas disponíveis para o debug de aplicações React. Ao longo deste conteúdo, apresentaremos algumas delas, já que elas podem nos auxiliar no processo de depuração de aplicações escritas em React Native.

A principal e mais simples ferramenta a ser apresentada é o **console do navegador**. Bastante útil em aplicações que utilizam o JavaScript e rodam no navegador, ele também pode ser usado para depurar os aplicativos React.

No caso do uso do JavaScript, basta acionar, a partir do navegador, a opção Inspecionar Elementos ou Inspecionador de Elementos. Em seguida, você só precisa acessar a aba **Console** para ter acesso a ele.

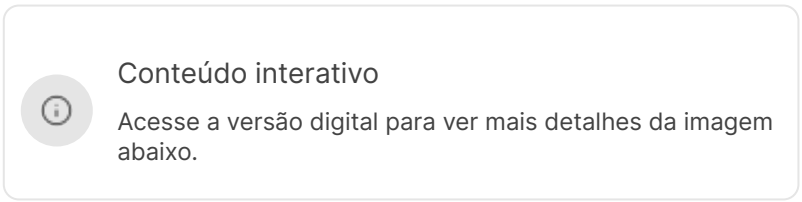
No React Native, contudo, o passo a passo é um pouquinho diferente. Nesse ambiente, você pode acessá-lo de algumas formas, a saber:

- A partir da janela do Metro, pressione a tecla “d”.
- React Developer Tools.
- In-App Developer Menu.
- Depuração de código nativo.

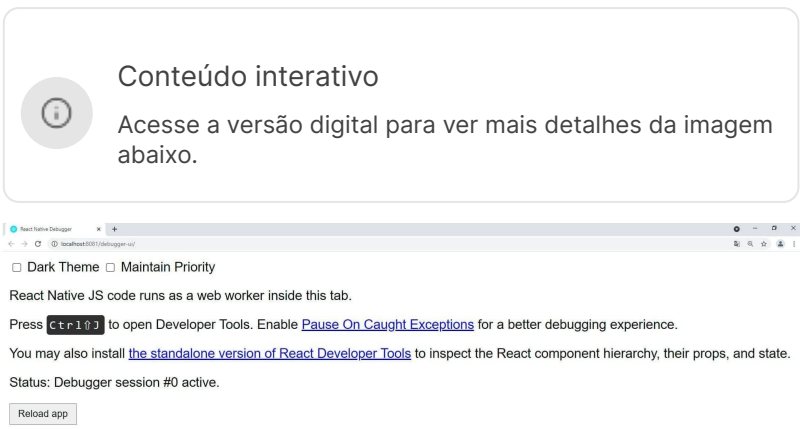
A seguir, veremos mais detalhes de cada uma das formas apresentadas.

A partir da janela do Metro, pressione a tecla “d”

Na janela do Metro, pressione a tecla “d”.



Em seguida, um novo menu será exibido no dispositivo virtual ou físico que você está utilizando. Entre as opções existentes, escolha Debug. Então uma nova janela do navegador será exibida. A partir dela, já é possível realizar a depuração.



Essa tela contém algumas informações adicionais para direcioná-lo. Você pode, a partir dela, abrir o console pressionando as telas **CTRL + J** (repare que é a letra J maiúscula, ou seja, combine **CTRL + SHIFT + J**). Com o console aberto, será possível analisar alguns aspectos do seu aplicativo.

Na prática, você verá que o console é bastante útil quando estamos trafegando dados externos em nossos aplicativos, já que é possível visualizar a chamada (*request*) e a resposta (*response*) de cada recurso. Além disso, a saída da instrução “console.log”, quando utilizada em nosso código, também pode ser vista nessa janela.

Observe que, conforme já mencionamos, há alguns links e algumas indicações de outras ferramentas de debug na janela aberta no navegador da imagem acima — entre elas, a ferramenta React Developer Tools, que será vista a seguir.

Dica

Outra forma de acessar a opção de debug é, no dispositivo físico, sacudir o aparelho. Isso fará com que o menu seja apresentado, possibilitando que a opção seja selecionada.

React Developer Tools

Esta ferramenta é uma biblioteca que, ao ser instalada, permite, por meio do navegador, a depuração da hierarquia de:

- Componentes
- Estilos
- Propriedades
- Estados do aplicativo

Para instalar essa ferramenta, execute o comando abaixo no terminal:

```
javascript
npm install -g react-devtools
```

Recomendação

Por se tratar de uma ferramenta, recomenda-se a instalação da biblioteca de forma global. Por isso, utilizamos a opção “-g”.

Após realizar a instalação, já no terminal, você precisará executar na pasta do projeto o seguinte comando:

```
javascript
react-devtools
```

O comando abrirá uma nova janela (mostrada abaixo). A seguir, você poderá acessar o DevTools, por meio do menu Developer (teclando “d” na janela do Metro; CTRL + M, no emulador; ou sacudindo o dispositivo físico), na opção Show Inspetor. Lembre-se de que o aplicativo também precisa estar rodando.

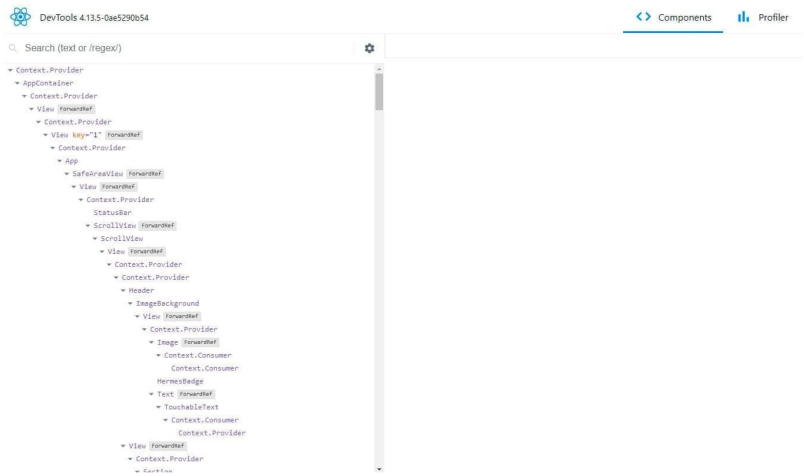
Dica

É possível que haja erros no processo em curso dependendo do dispositivo que você estiver usando para testar sua aplicação e/ou das versões das bibliotecas instaladas. Fique atento às janelas do terminal e do DevTools, pois elas exibirão os possíveis erros encontrados.

Ao final das etapas descritas, a janela do DevTools exibirá a hierarquia de componentes de seu aplicativo conforme a imagem a seguir!

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.

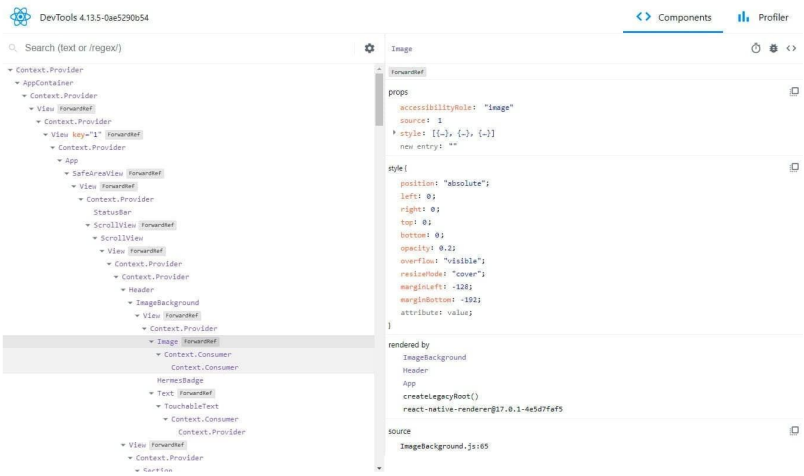


Hierarquia de componentes.

É possível obter mais informações sobre cada um dos elementos que compõem o aplicativo graças ao painel à esquerda da janela na qual eles são exibidos. Para isso, clique em um elemento e veja detalhes sobre ele no painel da direita. A imagem adiante exibe detalhes de um componente Image.

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Detalhes de componente.

Outra forma de debugar o aplicativo por meio do DevTools é no In-App Developer Menu (combinação de teclas apresentada anteriormente). Nele, é possível clicar, a partir de um dispositivo físico, em Toggle Inspector. Isso permite a obtenção das mesmas informações exibidas na janela do DevTools — e diretamente do dispositivo.

Como já vimos, essa ferramenta é muito detalhada, apresentando várias informações sobre o aplicativo e possuindo uma série de opções. Logo, além de navegar e analisar as informações e o conteúdo do aplicativo exibidos nela, recomendamos a leitura do site oficial para a obtenção de mais orientações.

In-App Developer Menu

Acessível tanto no dispositivo virtual quanto no físico (teclando “d” na janela do Metro; CTRL + M, no emulador; ou sacudindo o dispositivo físico), o In-App Developer Menu apresenta uma série de outras opções bastante úteis para a depuração de aplicativos.

Além das já mencionadas anteriormente, destacam-se ainda:

Fast Refresh

Permite a visualização mais rápida de mudanças feitas no código.

Sampling Profiler e Perf Monitor

Quando habilitados, ambos exibem informações detalhadas sobre o código JavaScript (*threads*) em execução e a performance do aplicativo.

Depuração de código nativo

Restrito a códigos nativos e não disponível em aplicações criadas utilizando o Expo, esse tipo de depuração acessa os logs detalhados do sistema. Para ter acesso a eles, você precisa fazer a execução destes comandos em três diferentes janelas do terminal:

1. Rodar o aplicativo a partir da pasta dele: npx react-native run-android.

2. Quando o aplicativo estiver rodando, habilitar estes logs:

- npx react-native log-android
- adb logcat *:S ReactNative:V ReactNativeJS:V

3. Os logs são exibidos na janela de terminal do Metro.

Como organizar o processo de depuração

Após ter decidido qual conjunto de ferramentas será utilizado na depuração de seu aplicativo, a etapa seguinte consiste em organizar o processo em si. Ou seja, tendo em mãos o conjunto de ferramentas necessário, precisamos agora decidir como usá-lo.

Este passo a passo contém algumas dicas para ajudá-lo ao longo dessa etapa! Vamos lá!

Confrontar o resultado esperado com o resultado obtido

Nosso aplicativo normalmente realiza uma série de ações, como obter dados externos ou realizar cálculos, por exemplo. Por isso, há uma série de resultados esperados para cada uma dessas ações. O primeiro passo consiste, portanto, em isolar uma das ações realizadas no aplicativo e analisá-la, a fim de verificar se o resultado dela corresponde ao esperado (conforme as definições realizadas na fase de análise e planejamento do software). Tendo feito isso para uma ação, devemos repetir o mesmo procedimento para as demais etapas.

Analisar os (eventuais) erros obtidos

Ao se deparar com um erro durante o processo de depuração, é preciso analisar o que levou à sua ocorrência. As causas podem ser várias, desde erro no código-fonte até motivos externos. Por exemplo, se o aplicativo depender da obtenção de dados provenientes de uma API externa e ela estiver indisponível, teremos um erro em nosso software. Esse tipo de erro pode ser rapidamente identificado pelos logs do console no inspecionador de elementos. Além disso, outros erros de código JavaScript também podem ser diagnosticados por intermédio do console. Na análise dos erros, é bastante comum esquecer a instanciação das variáveis ou não as utilizar dentro do seu real escopo (variáveis locais versus globais). Nesse caso — e conforme apontamos no passo anterior —, o erro não será tão evidente, não será possível vê-lo de forma destacada no console, mas é possível identificá-lo ao se obter um resultado diferente daquele esperado.

Isolar cenários de execução

Durante o planejamento de software, é comum definir diferentes fluxos de execução para a aplicação. Em cada funcionalidade, existem: fluxos básicos e fluxos alternativos. Por conta disso, é importante isolar tais cenários e analisar se os erros acontecem em todos eles ou apenas em algum(s) específico(s).

Utilizar pontos de interrupção

Em muitas situações, existe a necessidade de depurar o código desenvolvido por outros programadores. Nesses casos, pode-se não ter em mãos a documentação ou sequer conhecer os fluxos e o funcionamento da aplicação. Uma boa estratégia aqui é identificar o ponto de entrada da aplicação, ou seja, o fluxo pelo qual a aplicação começa a ser executada. Normalmente, existe uma tela de login; após seu processamento, em caso de sucesso, determinado fluxo é executado. Isso geralmente é um bom ponto de partida. Fazendo uso de uma IDE, deve-se inserir pontos de interrupção (ou pontos de pausa; nas IDEs, eles normalmente são chamados de *breakpoints*) nesse fluxo e começar a depurar o código a partir deles. As IDEs fornecem meios de seguir o fluxo de execução da aplicação ao analisarem sua sequência de forma automatizada.

Saiba mais

Ao longo deste módulo, apresentamos algumas ferramentas e técnicas para a depuração de aplicativos. Entretanto, tal conteúdo é extenso, contando ainda com várias outras ferramentas — entre elas, a utilização de plug-ins em diferentes IDEs. Tendo isso em vista — e após ter visto na prática e testado as ferramentas aqui esquematizadas —, procure aprofundar seu conhecimento. Um bom ponto de partida é a documentação do próprio React Native, que possui vários tópicos sobre esse assunto.

Ferramentas e técnicas de depuração

Neste vídeo, nosso especialista apresentará conceitos, ferramentas e técnicas de depuração de programas em React Native.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

A depuração de software é um processo que procura garantir a identificação e a correção de erros ou funcionamento inadequado. Com base nessa afirmação, assinale a alternativa correta que complementa tais conceitos.

A O debug de aplicações só pode ser feito por programadores experientes com vivência em diferentes linguagens de programação e conhecimento de diversas IDEs.

B A depuração de aplicativos React Native é restrita a aplicações criadas com o uso do Expo CLI, uma vez que essa biblioteca facilita o processo em questão, já que o aplicativo não roda em um dispositivo físico.

C Para realizar a depuração de aplicativos React Native, é necessária a aquisição de ferramentas adicionais — a maioria delas, comercial —, pois não há opções nativas disponíveis.

D A observação de pontos específicos do código é restrita à utilização de IDEs em que se pode inserir pontos de pausa. Em outras palavras, não é possível fazer essa observação utilizando outras ferramentas de debug disponíveis para React Native.

E Ao desenvolvermos um aplicativo usando o React Native, contamos com uma série de ferramentas de fácil acesso nativamente disponíveis, como logs detalhados, debug por meio do console do navegador e bibliotecas voltadas para aplicações que usam tal tecnologia.

A alternativa E está correta.

A depuração é uma tarefa importante no desenvolvimento de um software para encontrar e corrigir erros ou mau funcionamento, garantindo, assim, sua qualidade. Tal tarefa, dependendo do ambiente de desenvolvimento ou da linguagem de programação utilizada, pode ser difícil e trabalhosa. Entretanto, em aplicativos escritos em React Native, existe uma série de ferramentas disponíveis. Algumas delas são muito simples e acessíveis, como o próprio navegador web — independentemente de a aplicação estar rodando por meio do Expo, de um dispositivo virtual ou de um físico. Essa particularidade permite que até desenvolvedores iniciantes consigam realizar o debug de seu código-fonte.

Questão 2

Em relação ao processo de depuração, é correto afirmar que

A esse processo elimina todos os erros presentes no software, além de certificar que todos os requisitos dele foram atendidos.

B é recomendado utilizar apenas uma ferramenta ou técnica para a identificação de erros existentes no código. Isso reduz o tempo de debug e evita conflitos nos resultados dos testes.

C o processo de debug consiste em localizar e corrigir defeitos em uma aplicação. Tais defeitos dizem respeito a erros provenientes da etapa de codificação.

D a depuração é o processo de localizar e corrigir defeitos em uma aplicação, enquanto debug é o nome de uma das ferramentas que pode ser utilizada nessa tarefa.

E uma das grandes vantagens de se depurar um software é que um erro corrigido nunca gera novos erros ou comportamentos inesperados na aplicação.

A alternativa C está correta.

A existência de erros é inerente ao processo de desenvolvimento de um software independentemente da experiência do programador. Pode haver erros sintáticos (falhas na aplicação da sintaxe da linguagem utilizada), semânticos (uso incorreto de declarações) e de lógica (o programa não faz o que deveria fazer) durante uma codificação. Desse modo, identificá-los e corrigi-los, ou seja, depurar o software, é um processo que deve acontecer a partir da verificação e da validação do funcionamento do software em que as falhas são identificadas, localizadas no código-fonte e reparadas. A partir do momento que a correção

de uma falha consiste na escrita/alteração de códigos-fontes, o programador tem de ficar atento para que novas falhas não sejam geradas.

Considerações finais

Ao longo deste conteúdo, abordamos a programação para dispositivos móveis. Usando o framework React Native, delineamos os passos iniciais para o desenvolvimento de aplicativos da plataforma Android.

Em seguida, definimos o ambiente de desenvolvimento, passando ainda pela descrição de alguns componentes disponíveis nesse framework. Por fim, também falamos sobre a esquematização e o processo de depuração de aplicativos.

Os conhecimentos adquiridos neste material são introdutórios e essenciais, seja você um estudante da área ou um profissional iniciante no desenvolvimento de sistemas para dispositivos móveis.

Podcast

Para encerrar, ouça sobre sintaxe e componentes do React Native.

Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore+

Para saber mais sobre as diferenças entre React.js e React Native, há vários sites que as explicam — entre eles, o Devmedia.

DEVMEDIA. **Qual a diferença entre react e React Native?** Publicado em: 27 dez. 2017. Consultado na internet em: 9 jul. 2021.

O site oficial do React Native possui um tutorial no formato *learn the basics*, que pode ser muito útil para desenvolvedores iniciantes da plataforma.

REACT NATIVE. **Tutorial**. Consultado na internet em: 9 jul. 2021.

Referências

REACT NATIVE. **Docs**. Consultado na internet em: 9 jul. 2021.