Introdução ao Desenvolvimento para Dispositivos Móveis

[Conceitos Avançados]

Apostila do aluno

cepedi

Me. Ciro Daniel Gurgel de Moura
Vitor Rafael Queiroz Ferreira
Autor da apostila

Me. Ciro Daniel Gurgel de Moura
Vitor Rafael Queiroz Ferreira
Instrutor do curso

Autor

Ciro Daniel Gurgel de Moura

Titulação: Mestre em Ciência da Computação

Áreas de conhecimento: Experiência na área de Sistemas de Informação, com ênfase em Banco de Dados, Mineração de Dados, Geoinformática e Processamento de Imagens, atuando principalmente no Desenvolvimento de Sistemas Web e Aplicações Móveis.

Vitor Rafael Queiroz Ferreira

Titulação: Graduando em Análise e Desenvolvimento de Sistemas

Áreas de conhecimento: Conhecimento na área de Desenvolvimento de Softwares, com ênfase em Desenvolvimento Fullstack, atuando principalmente nos seguintes temas: Typescript; React; NextJS; VueJS; RestAPI. Portfolio: https://vitorrafael.com.br/

APRESENTAÇÃO

Olá! Boas-vindas ao curso de **Introdução ao Desenvolvimento para Dispositivos Móveis** oferecido pela FIT TECH Academy!

O desenvolvimento de aplicativos para dispositivos móveis é um campo tecnológico em constante evolução, que cresce exponencialmente, apresentando constantemente novas soluções e desafios. Além disso, a comunidade de profissionais nesse setor expande-se diariamente, reunindo programadores, designers e entusiastas.

Neste curso, você mergulhará no fascinante mundo dos dispositivos móveis, abrangendo todos os aspectos, desde a concepção inicial das interfaces de aplicativos e seu design visual até a implementação das funcionalidades que tornam os aplicativos úteis e interativos. Além disso, irá explorar a programação para diferentes plataformas, como Android e iOS.

O conteúdo deste curso é fundamentado em diversos materiais elaborados por profissionais especializados e respeitados na área. Para uma compreensão mais profunda, é importante que você leia atentamente este conteúdo e realize as atividades propostas ao longo de cada seção, além de explorar os materiais recomendados nas referências bibliográficas.

A apostila está organizada em cinco seções distintas: **Comunicação com API** - Onde serão dados conceitos sobre HTTP e seus métodos, além de como realizar a comunicação entre aplicativo e API usando a biblioteca Axios**, Depuração e Testes** - Que abordará conceitos e dicas de como realizar esses processos que são importantíssimos no desenvolvimento de aplicações, **Validação de Formulários** - Reforçando a importância de se gerenciar e validar os dados encaminhados via formulário, utilizando bibliotecas como Formik e Yup**, Introdução ao React Native Maps** - Que aponta os conceitos de geolocalização e traz exemplos de como integrar mapas em uma aplicação móvel e, por fim, **Notificações** - Onde serão apresentados conceitos sobre o envio de alertas ou informações para cada usuário que utiliza uma aplicação.

Desejo a você, estudante, que tenha um excelente curso!

Boa Leitura!

Indicação de ícones (opcional)

Saiba mais: *oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.*

Exemplos: *descreve exemplos sobre o assunto que está sendo abordado.*

Atenção: *indica pontos de maior relevância no texto.*

Avisos: *oferece avisos referente ao assunto..*

Sumário

1 Comunicação com APIs

Para desenvolver aplicações é essencial o dinamismo, mas, se fosse para desenvolver um aplicativo do zero com dinamismo sem comunicar com qualquer outra interface (API) seria praticamente impossível, já que os dados da aplicação seriam fixados no momento de *build*. Já com a implementação de uma API, os dados podem ser servidos de forma dinâmica, consumidos de um banco de dados, alterados por algum tipo de painel administrativo, etc.

1.1 Antes de tudo… O que é uma API?

Segundo Amazon (2024), as APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos. Por exemplo, o sistema de software do instituto meteorológico contém dados meteorológicos diários. A aplicação para a previsão do tempo em seu telefone “fala” com esse sistema por meio de APIs e mostra atualizações meteorológicas diárias no telefone.

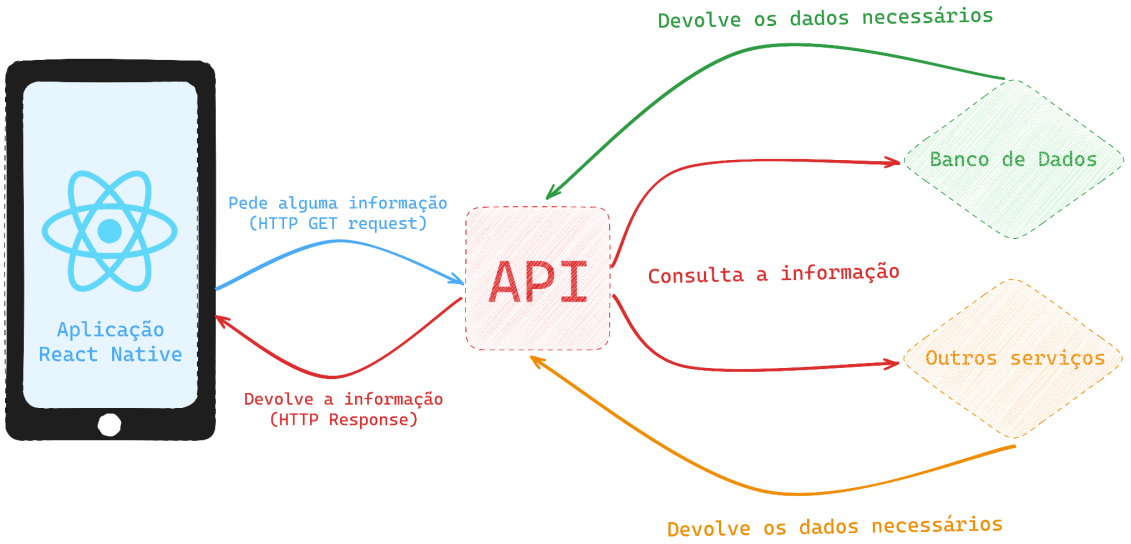


Figura 1 - Ilustração do funcionamento de uma API Simples.
Fonte: Reprodução dos autores

1.2 E o que é o protocolo HTTP?

Para Tanenbaum (2003), o Protocolo de Transferência de Hipertexto, ou HTTP (sigla em inglês para Hypertext Transfer Protocol), é um conjunto de regras e convenções que facilitam a comunicação entre computadores na World Wide Web. Ele permite a transferência de informações, como textos, imagens e outros recursos, entre um navegador web (como o Chrome, Firefox ou Safari) e um servidor web.

O HTTP funciona como um protocolo de solicitação e resposta, onde o navegador envia pedidos para o servidor, e o servidor responde fornecendo os recursos solicitados. Esses pedidos e respostas são trocados por meio de mensagens que seguem um formato padrão.

Além disso, existem métodos padrão para requisições HTTP, alguns destes são:

1.2.1 GET

O método GET é utilizado para solicitar dados de um recurso específico no servidor. Ele é usado para recuperar informações, e a requisição não deve ter efeitos colaterais no servidor.

1.2.2 POST

O método POST é utilizado para enviar dados ao servidor para serem processados. É comumente usado ao enviar informações de formulários HTML e pode ter efeitos colaterais no servidor, como a criação de um novo recurso.

1.2.3 PATCH

O método PATCH é utilizado para aplicar modificações parciais a um recurso. Em vez de enviar o recurso completo, o PATCH envia apenas as alterações que devem ser aplicadas ao recurso existente.

1.2.4 PUT

O método PUT é utilizado para atualizar um recurso no servidor ou criar um recurso se ele não existir. Ele envia todo o recurso atualizado para o servidor.

1.2.5 DELETE

O método DELETE é utilizado para solicitar a remoção de um recurso no servidor. Ele indica ao servidor que o cliente deseja excluir o recurso especificado.

1.3 Como o React Native consome uma API?

Existem diversas maneiras de se consumir uma API e durante o curso será utilizada a biblioteca Axios (2023). Essa biblioteca é um cliente HTTP que roda tanto no navegador, como no NodeJS. Através dele será possível enviar requisições HTTP para qualquer API que esteja sob domínio do desenvolvedor. Antes de utilizá-lo, é importante que consulte sua documentação no link acima. Veja no Exemplo 1 como realizar uma requisição POST com o mesmo.

Exemplo 1 - Realizando uma requisição POST com Axios

```
# A requisição está sendo realizada em uma URL fictícia na rota /user, ex: http://localhost:5555/user/
axios.post('/user', {
    firstName: 'Santos',
    lastName: 'Dumont'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.error(error);
  });
```

Dessa forma, ao realizar a requisição do tipo POST no *endpoint* (ou rota) '/user', através do uso da biblioteca Axios, será possível obter uma resposta (*response*) ou caso algo de errado aconteça, será retornado um erro (*error*).

2 Depuração e Testes

A depuração e os testes manuais são partes essenciais do processo de desenvolvimento de aplicativos React Native. Com as técnicas e ferramentas adequadas, os desenvolvedores podem identificar e corrigir problemas no código, garantir que o aplicativo funcione conforme o esperado e oferecer uma experiência de usuário de alta qualidade. Ao incorporar práticas de depuração e testes manuais em seu fluxo de trabalho de desenvolvimento, você pode garantir que seu aplicativo seja robusto, confiável e eficaz.

2.1 Depuração

Em React Native, conforme aponta a documentação oficial (REACT NATIVE, 2023) existem várias técnicas e ferramentas disponíveis para facilitar o processo de depuração. Algumas dessas são descritas a seguir.

2.1.1 console.log()

Uma das técnicas mais simples e eficazes de depuração é usar console.log() para imprimir valores e mensagens no console do seu ambiente de desenvolvimento. Isso permite que você acompanhe o estado do aplicativo e identifique possíveis problemas.

2.1.2 React Developer Tools

Esta é uma extensão para o Chrome e Firefox que permite inspecionar a hierarquia de componentes React em seu aplicativo, bem como visualizar e modificar o estado e as props de cada componente.

2.1.3 Reactotron

É uma ferramenta de depuração para aplicativos React e React Native. Ele oferece recursos como monitoramento de estado, visualização de logs, inspeção de rede e muito mais, facilitando a depuração de aplicativos React Native.

2.1.4 Remote Debugging

O React Native oferece suporte ao remote debugging, que permite depurar aplicativos React Native em dispositivos físicos usando o Chrome Developer Tools ou o React Native Debugger. Isso é especialmente útil para depurar problemas que ocorrem apenas em dispositivos reais.

2.2 Testes manuais

Os testes manuais desempenham um papel importante no processo de desenvolvimento de aplicativos React Native, permitindo que os desenvolvedores verifiquem manualmente se o aplicativo funciona conforme o esperado e se atende aos requisitos de design e funcionalidade. Aqui estão algumas práticas comuns para testes manuais em aplicativos React Native:

2.2.1 Testes de Interface do Usuário (UI)

Teste a interface do usuário do aplicativo em diferentes dispositivos e tamanhos de tela para garantir que os elementos da interface estejam bem posicionados e se comportem conforme o esperado.

2.2.2 Testes de funcionalidades

Teste as principais funcionalidades do aplicativo para garantir que elas funcionem conforme o esperado, como preenchimento de formulários, navegação entre telas e interações com elementos interativos.

2.2.3 Testes de navegação

Teste a navegação entre telas do aplicativo para garantir que os usuários possam navegar facilmente pelo aplicativo e acessar todas as funcionalidades.

2.2.4 Testes de desempenho

Teste o desempenho do aplicativo em diferentes condições, como conexões de rede lentas ou dispositivos com recursos limitados, para garantir que o aplicativo seja responsivo e rápido em todas as situações.

2.2.5 Testes de compatibilidade

Teste o aplicativo em diferentes dispositivos, sistemas operacionais e versões de navegador para garantir que ele funcione corretamente em uma variedade de ambientes.

3 Validação de Formulários

Os formulários são componentes essenciais em aplicações, permitindo a coleta de dados dos usuários. Gerenciar o estado de cada campo de um formulário, lidar com validações, tratamento de erros e feedbacks podem ser tarefas complexas e trabalhosas.

Dessa forma, há uma necessidade de lidar com esses pontos, além de tratar para que o código não seja repetitivo no momento de controlar os campos e também deve-se implementar as regras de validação de um forma que seja fácil prover a manutenção e escalabilidade.

O Formik é uma biblioteca popular em React e React Native que simplifica o gerenciamento de formulários (FORMIK, 2023), enquanto o Yup é uma biblioteca de validação de esquemas JavaScript (YUP, 2023). Juntos, oferecem uma solução poderosa para validar e gerenciar formulários de maneira eficiente e robusta.

3.1 Integrando o Formik com Yup

A integração do Formik com o Yup é direta e eficaz. Nas próximas subseções serão dados detalhes de como essas duas bibliotecas podem ser usadas juntas para validar um formulário em um aplicativo React Native.

3.1.1 Instalando as dependências e construindo o formulário

Certifique-se de ter as dependências necessárias instaladas no seu projeto:

```
npm install formik yup
```

Após a instalação, será possível utilizar o Formik e o Yup em seu projeto, conforme apresenta o Exemplo 2. É importante destacar que deve-se importar as duas bibliotecas quando for utilizá-las.

Exemplo 2 - Criando um formulário com Formik e Yup

```
# Comece criando um formulário simples com o Formik. O Formik
fornece os componentes <Formik> e <Field> para envolver e
renderizar campos de formulário. Por exemplo:

import React from "react";
```

```
import { View, TextInput, Button } from "react-native";
import { Formik, Field } from "formik";

const MyForm = () => {
  return (
    <Formik
      initialValues={{ email: "", password: "" }}
      onSubmit={(values) => console.log(values)}
    >
      {({ handleChange, handleSubmit, values }) => (
        <View>
          <Field
            name="email"
            component={TextInput}
            placeholder="Email"
            onChangeText={handleChange("email")}
            value={values.email}
          />
          <Field
            name="password"
            component={TextInput}
            placeholder="Password"
            secureTextEntry
            onChangeText={handleChange("password")}
            value={values.password}
          />
          <Button title="Submit" onPress={handleSubmit} />
        </View>
      )}
    </Formik>
  );
};

export default MyForm;
```

Exemplo 2.1 - Adicionando validação ao formulário

```
# Agora, adicione validação ao nosso formulário usando Yup. Você pode definir um esquema de validação e aplicá-lo ao seu formulário. Por exemplo:

import * as Yup from 'yup';

const validationSchema = Yup.object().shape({email: Yup.string().email('Email inválido').required('Campo obrigatório'),
  password: Yup.string().min(6, 'A senha deve ter pelo menos 6 caracteres').required('Campo obrigatório'),
});

# Dentro do componente Formik
<Formik
  initialValues={{ email: '', password: '' }}
  validationSchema={validationSchema}
  onSubmit={values => console.log(values)}
></Formik>
```

Exemplo 2.2 - Exibindo mensagens de erro

```
# Agora que você configurou a validação, é possível exibir mensagens de erro para os campos do formulário. O Formik fornece uma propriedade errors que contém os erros de validação. Por exemplo:

{({ handleChange, handleSubmit, values, errors }) => (
  <View>
    <Field
      name="email"
      component={TextInput}
      placeholder="Email"
      onChangeText={handleChange('email')}
      value={values.email}
    />
          {errors.email && <Text style={{ color: 'red' }}>{errors.email}</Text>}
    {/* Exibir mensagens de erro para outros campos */}
    <Field
      name="password"
      component={TextInput}
      placeholder="Password"
      secureTextEntry
```

```
        onChangeText={handleChange('password')}
        value={values.password}
      />
        {errors.password && <Text style={{ color: 'red' }}>{errors.password}</Text>}
      <Button title="Submit" onPress={handleSubmit} />
    </View>
)}
```

4 Introdução ao React Native Maps

Realizar a integração das aplicações móveis com mapas e geolocalização é uma prática cada vez mais essencial e desafiadora, devido à crescente demanda por funcionalidades baseadas em localização. A capacidade de incorporar mapas interativos e serviços de geolocalização em aplicativos permite uma variedade de recursos valiosos, desde direções e navegação até informações personalizadas com base na localização do usuário.

O React Native Maps (2023) surge como uma ferramenta fundamental para simplificar e agilizar esse processo, oferecendo uma biblioteca robusta e flexível para a integração de mapas e geolocalização em aplicativos React Native.

O React Native Maps é uma biblioteca que permite integrar mapas interativos em aplicativos React Native. Ele fornece componentes que facilitam a exibição de mapas e a interação com eles, incluindo marcadores, polígonos, polilinhas e muito mais. Com o React Native Maps, os desenvolvedores podem adicionar funcionalidades de mapa poderosas e personalizadas aos seus aplicativos.

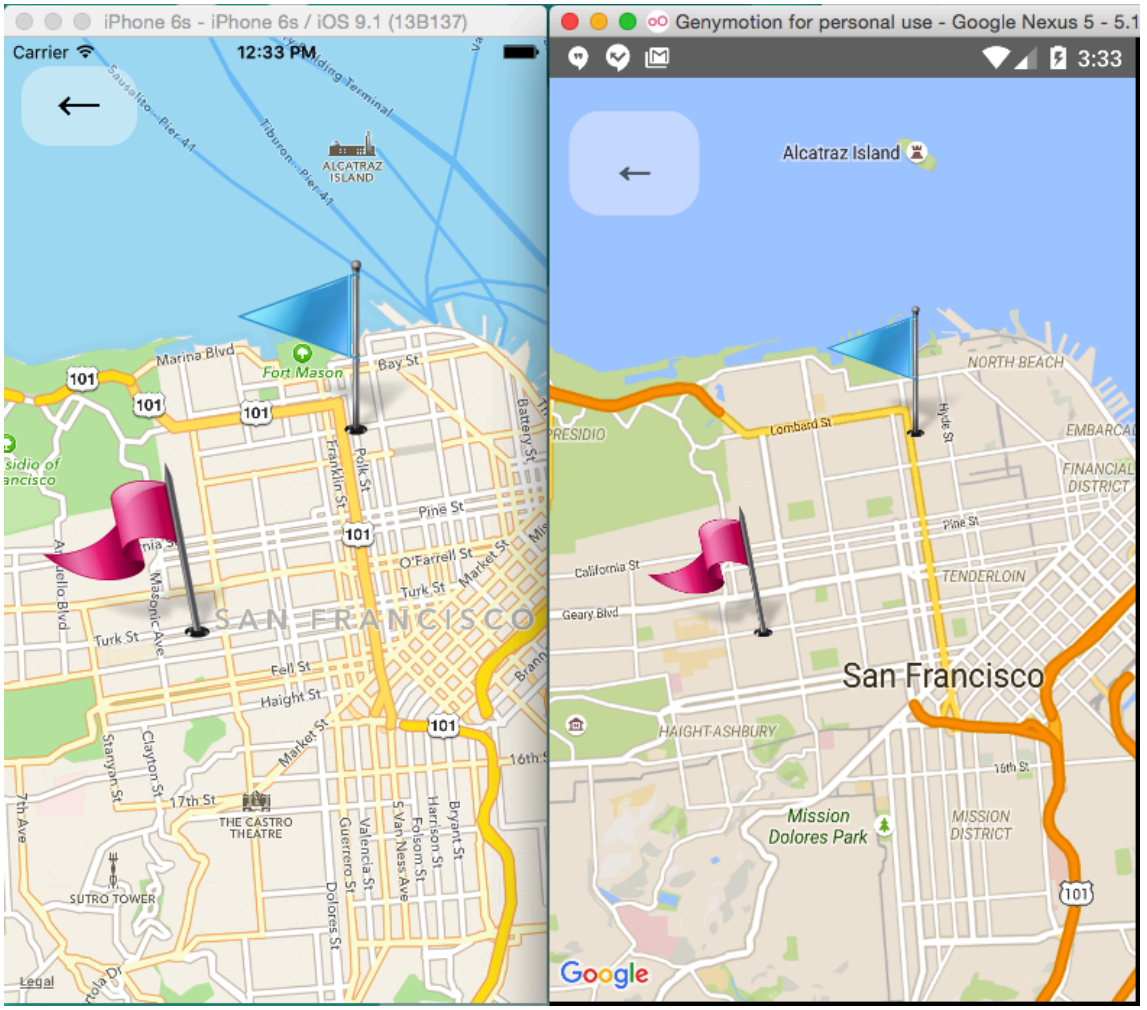


Figura 3: Demonstração de funcionalidade da biblioteca React Native Maps no Android.

Fonte: [React Native Map components for iOS + Android]

4.1 Configuração inicial da biblioteca

Para começar a usar o React Native Maps em seu projeto, você precisa configurar as dependências necessárias. Aqui estão os passos básicos:

4.1.1 Instalação

```
npx expo install react-native-maps
```

4.1.2 Criando chave de API do Google

No Android, para poder receber os dados de mapas reais é necessário utilizar a API do Google Maps, para isso é preciso ter uma chave de acesso. Veja abaixo como criá-la.

- Vá até a página inicial em [Using API Keys | Maps SDK for Android | Google for Developers], lá terá um breve tutorial de como criar uma credencial;

- Na página, utilize o botão azul, pode ser necessário fazer login ou criar uma nova conta;

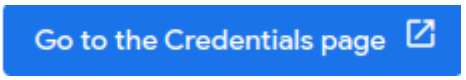


Figura 2: Representação do botão na página do Google. Fonte: reprodução do autor

- Na página de seleção de projetos clique em **CRIAR PROJETO;**
- Coloque o **NOME DO PROJETO** que desejar e em **LOCAL** é possível deixar na opção "*Sem organização*"
- Após criar o projeto, você será redirecionado para as configurações dele, na **barra lateral** vá em "*Chaves e credenciais*"
- **OBS:** Pode ser necessário vincular uma conta de faturamento com o Google, isso é feito apenas por questões de segurança da empresa, mas para uso pessoal e sem fins comerciais a chave se torna grátis
- Ao navegar até "*Chaves e credenciais*" será aberto um questionário, você pode o preencher informando que o projeto não terá fins comerciais e apenas será de uso pessoal.
- Após concluir o formulário, sua **chave de API** será apresentada, copie e a guarde.
- Em seu Android Studio, na tela inicial, vá em "**More Actions**" e clique em "**SDK Manager**", lá haverá três abas referentes ao seu SDK, clique na segunda (**SDK Tools**).
- Em *SDK Tools*, marque todos os pacotes referentes ao Google e clique em "Apply", eles irão baixar.
- Caso seu emulador esteja aberto, reinicie.

Exemplo 3 - Adicionando chave da API

```
# No seu arquivo app.json, o objeto “android” deve ficar parecido com o seguinte:

"android": {
 "config": {
   "googleMaps": {
     "apiKey": "SUA CHAVE API AQUI"
   }
 },
 "adaptiveIcon": {
   "foregroundImage": "./assets/adaptive-icon.png",
   "backgroundColor": "#ffffff"
 },
 "package": "com.anonymous.vagacerta"
},
```

Exemplo 3.1 - Adicionando MapView

```
# Com a chave da API configurada, basta inserir o componente MapView do react-native-maps em algum lugar do aplicativo que deseja utilizar.

import MapView, { PROVIDER_GOOGLE } from "react-native-maps";

<MapView
 provider={PROVIDER_GOOGLE}
 style={{
   width: "100%",
   height: "100%",
   flex: 4,
 }}
 initialRegion={{
   latitude: 37.78825,
   longitude: -122.4324,
   latitudeDelta: 0.0922,
   longitudeDelta: 0.0421,
 }}
/>
```

4.2 Como visualizar o mapa?

Caso tenha concluído os passos anteriores e tenha notado que não houve qualquer visualização no seu emulador, você precisa gerar um *build* da sua aplicação localmente. Considerando que esta utilizando android, utilize o comando:

```
npx expo run:android
```

Com isso, o Expo irá gerar um *build* de desenvolvimento da sua aplicação, permitindo com que seja possível visualizar o mapa sem maiores problemas.

5 Introdução às Notificações Push

As notificações push são mensagens que podem ser enviadas aos dispositivos móveis dos usuários, mesmo quando eles não estão interagindo diretamente com o aplicativo. Essas mensagens podem ser usadas para informar sobre novos conteúdos, eventos importantes ou atualizações relevantes.

No React Native, as notificações push são tratadas por meio de bibliotecas e serviços específicos, como o Firebase Cloud Messaging (FCM) para dispositivos Android e o Apple Push Notification Service (APNs) para dispositivos iOS.

Nas próximas subseções serão detalhados os passos de como configurar e realizar a comunicação entre um aplicativo e o Firebase. Dessa forma, serão dados os detalhes voltados para uma aplicação desenvolvida para Android.

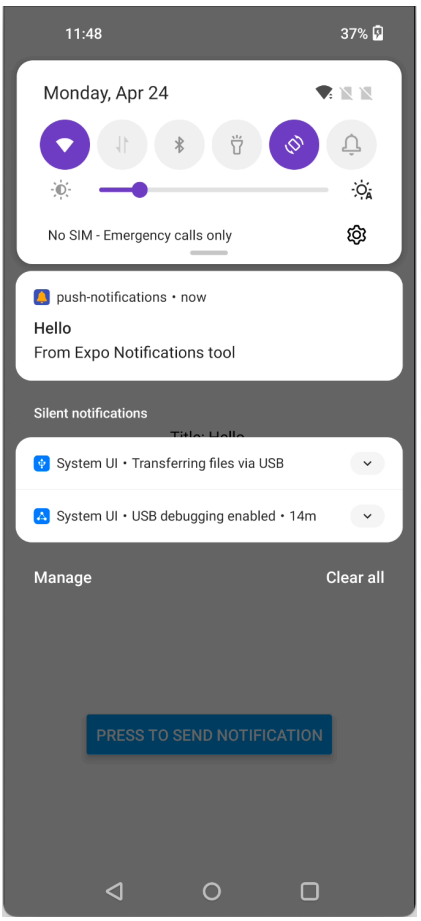


Figura 4: Exemplo de notificação push com Expo Push Notifications. Fonte: Push notifications setup - Expo Documentation

5.1 Configuração inicial do Firebase Cloud Messaging

- Acesse o Firebase Console e crie um novo projeto.
- Vá para as configurações do projeto no Firebase Console.
- Na aba Cloud Messaging, ative a API Legado clique no ícone de três pontos verticais e em seguida em Gerenciar.
- Ative a API e siga as instruções fornecidas.

5.2 Configuração inicial do projeto Expo

- Acesse sua conta Expo.
- Crie um novo projeto na interface web (com o mesmo nome do qual você criou em sua máquina, caso já tenha criado).
- Copie os comandos fornecidos de acordo com seu caso (caso já tenha criado um projeto local ou não)

5.3 Comunicação Firebase e Expo

- No seu projeto local, utilize o comando:

```
eas build
```

- Aguarde aparecer o texto "*Build in progress*" e então cancele.
- No Expo, adicione uma nova chave ao Android em "Credentials", na parte de FCM.
- Copie a chave fornecida pelo Firebase e adicione-a ao seu projeto, na parte de FCM, no Expo.
- No Firebase, registre o aplicativo Android, usando o pacote definido no arquivo app.json na chave "android.package" do seu projeto.
- Baixe o arquivo google-services.json fornecido pelo Firebase e coloque-o na raiz do seu projeto React Native.
- No arquivo app.json, adicione "googleServicesFile": "./google-services.json" abaixo de "package".

Exemplo 4 - Hook de notificações personalizado

```
# Esse é um hook de exemplo com base na documentação do expo-push-notifications. Caso esteja utilizando um emulador, troque o Device.isDevice para !Device.isDevice.

# O hook pode ser testado em modo de desenvolvimento ou em build

import * as Device from "expo-device";
import * as Notifications from "expo-notifications";
import { useEffect, useRef, useState } from "react";

import Constants from "expo-constants";

import { Platform } from "react-native";

export interface PushNotificationState {
 expoPushToken?: Notifications.ExpoPushToken;
 notification?: Notifications.Notification;
 isLoading: boolean;
 sendPushNotification: (
   title: string,
   body: string,
   token: string,
   data?: any
 ) => Promise<void>;
}

export const usePushNotifications = ():
PushNotificationState => {
 Notifications.setNotificationHandler({
   handleNotification: async () => ({
     shouldPlaySound: false,
     shouldShowAlert: true,
     shouldSetBadge: false,
   }),
 });

 const [isLoading, setIsLoading] = useState(false);

 const [expoPushToken, setExpoPushToken] = useState<
   Notifications.ExpoPushToken | undefined
 >();
```

```
 const [notification, setNotification] = useState<
   Notifications.Notification | undefined
 >();

 const notificationListener =
useRef<Notifications.Subscription>();
 const responseListener =
useRef<Notifications.Subscription>();

 async function registerForPushNotificationsAsync() {
   setIsLoading(true);
   let token;
   if (Device.isDevice) {
     const { status: existingStatus } =
       await Notifications.getPermissionsAsync();
     let finalStatus = existingStatus;

     if (existingStatus !== "granted") {
       const { status } = await
Notifications.requestPermissionsAsync();
       finalStatus = status;
     }
     if (finalStatus !== "granted") {
       alert("Failed to get push token for push
notification");
       return;
     }

     token = await
Notifications.getExpoPushTokenAsync({
       projectId:
Constants.expoConfig?.extra?.eas.projectId,
     });
   } else {
     alert("Must be using a physical device for Push
notifications");
   }

   if (Platform.OS === "android") {

Notifications.setNotificationChannelAsync("default", {
       name: "default",
```

```
        importance: Notifications.AndroidImportance.MAX,
        vibrationPattern: [0, 250, 250, 250],
        lightColor: "#FF231F7C",
      });
    }

    setIsLoading(false);

    return token;
  }

  async function sendPushNotification(
    title: string,
    body: string,
    token: string,
    data?: any
  ) {
    const message = {
      to: token,
      sound: "default",
      title,
      body,
      data,
    };

    await fetch("https://exp.host/--/api/v2/push/send",
{
      method: "POST",
      headers: {
        Accept: "application/json",
        "Accept-encoding": "gzip, deflate",
        "Content-Type": "application/json",
      },
      body: JSON.stringify(message),
    });
  }

  useEffect(() => {
    registerForPushNotificationsAsync().then((token) =>
{
      setExpoPushToken(token);
    });
```

```
    notificationListener.current =

Notifications.addNotificationReceivedListener((notifica
tion) => {
        setNotification(notification);
      });

    responseListener.current =

Notifications.addNotificationResponseReceivedListener((
response) => {
        console.log(response);
      });

    return () => {
      Notifications.removeNotificationSubscription(
        notificationListener.current!
      );


Notifications.removeNotificationSubscription(responseLi
stener.current!);
    };
  }, []);

  return {
    expoPushToken,
    notification,
    sendPushNotification,
    isLoading,
  };
};
```

Através das informações do retornadas pelo hook, é possível visualizar o token do expo, a notificação e enviar novas notificações.

Conclusão

Esta apostila abrange uma extensa gama de tópicos, proporcionando uma visão abrangente do desenvolvimento de aplicações móveis usando React Native. Desde os conceitos de Validação de Formulário, passando por pontos importantes como API, seus métodos e forma de comunicação, além de Geolocalização. Dessa forma, cada leitor(a) é guiado(a) através de uma jornada educativa que visa não apenas apresentar informações, mas também cultivar uma compreensão e prática.

A organização em seções distintas facilita a absorção progressiva de conhecimento, permitindo que os(as) leitores(as) se aprofundem gradualmente em conceitos mais complexos. A seção inicial fornece uma sólida introdução aos fundamentos de API, enquanto cada seção subsequente expande e aprofunda áreas específicas do desenvolvimento móvel.

A inclusão de tópicos como Notificações e Geolocalização reflete a difusão de conceitos considerados avançados para o desenvolvimento de aplicativos, abrangendo desde a Validação de Formulário até aspectos importantes para realização de Depuração e Testes.

Assim, o conteúdo do curso Introdução ao Desenvolvimento para Dispositivos Móveis familiariza o leitor com o mundo tecnológico, em um campo tecnológico que está sempre evoluindo, crescendo exponencialmente e apresentando constantemente novas soluções e desafios.

Obrigado por fazer parte do curso e ter realizado a leitura desta apostila. Espero que o interesse por Desenvolvimento Mobile apresentado neste curso esteja aguçado, para que você pratique e conheça ainda mais as maravilhas do Desenvolvimento para Dispositivos Móveis.

Referências

AMAZON. Amazon Web Services: API (Application Programming Interface). Disponível em: https://aws.amazon.com/pt/what-is/api/. Acesso em janeiro de 2024.

AXIOS. Documentação do Axios. Disponível em: https://axios-http.com/ptbr/docs/intro. Acesso em dezembro de 2023.

EXPO. Documentação do Expo. Disponível em: https://docs.expo.dev. Acesso em dezembro de 2023.

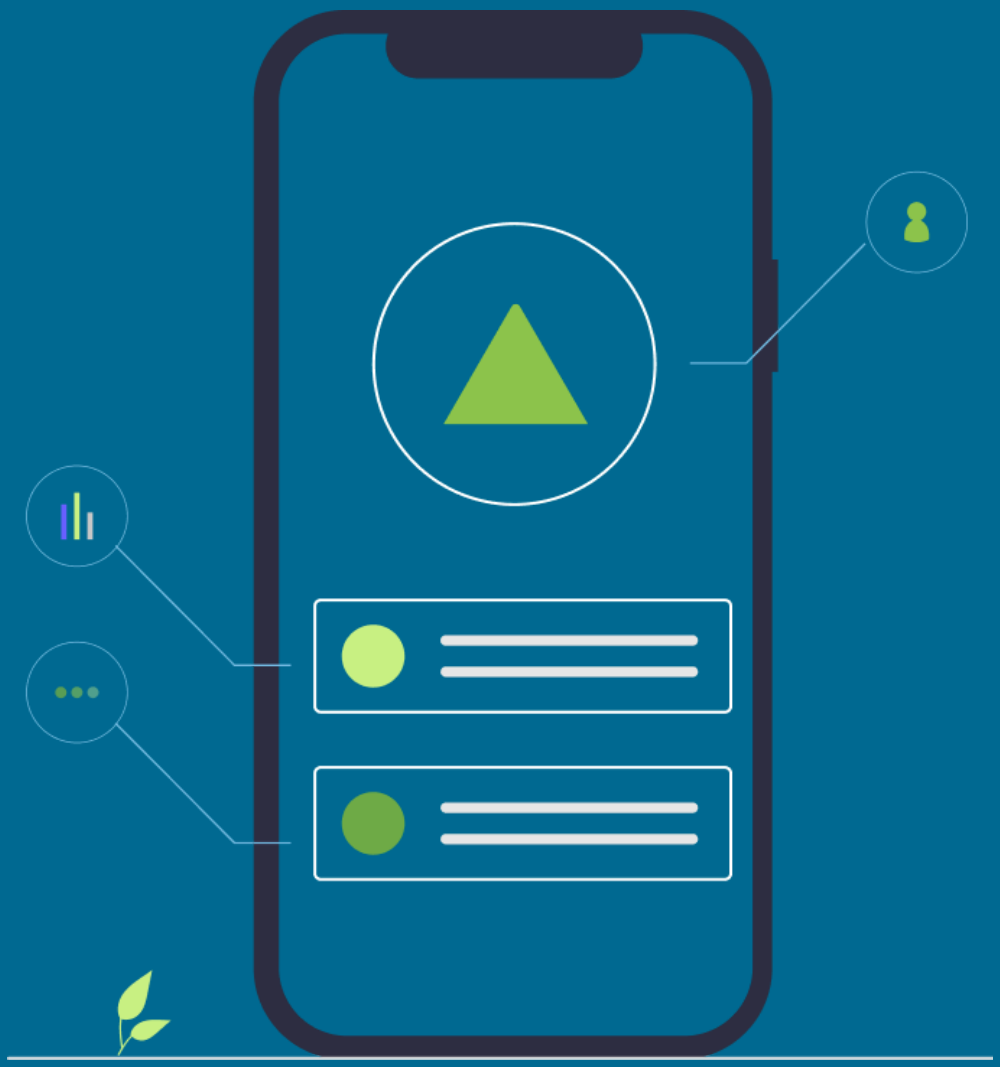
FORMIK. Guia para React Native. Disponível em: https://formik.org/docs/guides/react-native. Acesso em dezembro de 2023.

REACT NATIVE. Documentação oficial. Disponível em: https://reactnative.dev. Acesso em dezembro de 2023.

REACT-NATIVE-MAPS. Repositório do GitHub. Disponível em: https://github.com/react-native-maps/react-native-maps. Acesso em dezembro de 2023.

TANENBAUM, Andrew S. Redes de Computadores. 4. ed. São Paulo: Pearson Education do Brasil, 2003.

YUP. Repositório do GitHub. Disponível em: https://github.com/jquense/yup. Acesso em dezembro de 2023.



BOM CURSO!

cepedi