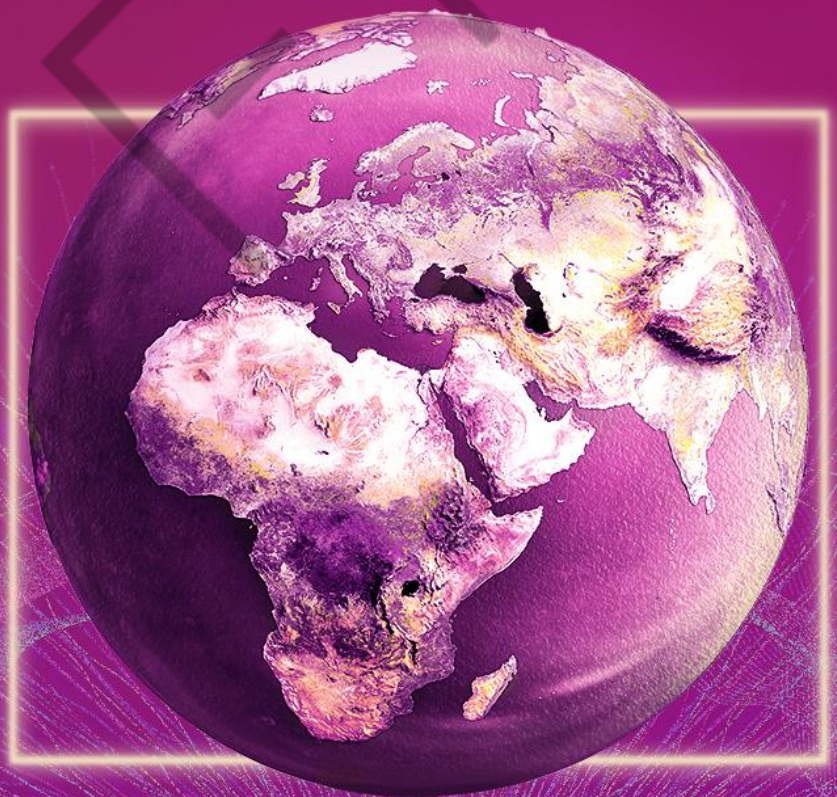
APP WORLD

INTERNACIONALIZAÇÃO E VALIDAÇÃO DE ENTRADA DE DADOS



10A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 INTERNACIONALIZAÇÃO E VALIDAÇÃO DE ENTRADA DE DADOS

1.1 Internacionalização da aplicação

Quando desenvolvemos uma aplicação para Android, temos que ter em mente o seu alcance regional, ou seja, ele pode ter um alcance local para usuários de um único país ou pode ter um alcance mais globalizado, onde pessoas com diferentes idiomas poderão utilizá-lo. Vamos aprender agora como tornar nosso aplicativo internacional. Vamos lá!

1.1.1 Nosso aplicativo de testes

Para tornar o nosso trabalho mais focado no que realmente interessa, vamos criar uma aplicação com uma única tela responsável pela autenticação do usuário, então, crie um projeto no Android Studio com o nome “Login”. Você já sabe como fazer isso, então sinta-se à vontade para definir o package e local de gravação. Após a criação do projeto, substitua o conteúdo do arquivo “MainActivity.kt” pelo código da listagem abaixo:

```
package br.com.fiap.loginprof

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
```

```
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import br.com.fiap.loginprof.ui.theme.LoginProfTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      LoginProfTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          Login()
        }
      }
    }
  }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Login() {

  var email by remember() {
    mutableStateOf("")
  }

  var password by remember {
    mutableStateOf("")
  }

  Column(modifier = Modifier.padding(16.dp)) {
    Text(
      text = "Login",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color.Blue
    )
    Text(text = "Por favor entre com seus dados")
    Spacer(modifier = Modifier.height(48.dp))
    Card(modifier = Modifier
      .fillMaxWidth()) {
        Column(modifier = Modifier
          .fillMaxWidth()
          .padding(32.dp)) {
          OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            modifier = Modifier.fillMaxWidth(),
            label = {
                    Text(text = "Digite o seu e-mail")
            },
            keyboardOptions = KeyboardOptions(
```

```
                keyboardType = KeyboardType.Email
              ),
            )
            Spacer(modifier = Modifier.height(16.dp))
            OutlinedTextField(
              value = password,
              onValueChange = { password = it },
              modifier = Modifier.fillMaxWidth(),
              label = {
                      Text(text = "Digite a sua senha")
              },
              keyboardOptions = KeyboardOptions(
                keyboardType = KeyboardType.Password
              ),
              visualTransformation = PasswordVisualTransformation()
            )
            Spacer(modifier = Modifier.height(32.dp))
            Button(onClick = { /*TODO*/ }) {
              Text(
                text = "ENTRAR",
                modifier = Modifier
                  .padding(8.dp)
                  .fillMaxWidth(),
                textAlign = TextAlign.Center
              )
            }
          }
      }
    }
}
```

Código-fonte 1 – Código da tela Login
Fonte: Elaborado pelo autor (2023)

Execute a aplicação em um emulador. O resultado esperado dever ser como o exibido na figura “Tela Login”:

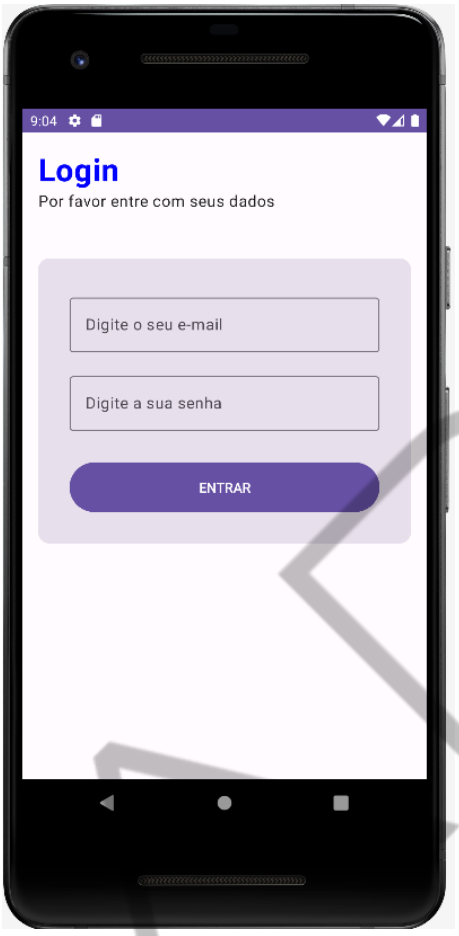


Figura 1 – Tela Login
Fonte: Google Imagens (2023)

1.1.2 O arquivo strings.xml

Independentemente se você deseja ou não que a sua aplicação seja internacional, a verdade é que nunca devemos inserir os textos literais em nossa aplicação. É muito comum, ao passar do tempo, que haja a necessidade de alterarmos alguma expressão que está contida em diversas telas e o esquecimento da alteração de uma delas torna nossa aplicação inconsistente e estranha ao usuário.

Então, se faz necessário a centralização de todo o conteúdo textual da aplicação em um único lugar, e qualquer alteração efetivada neste local deverá refletir em toda a aplicação. Para essa finalidade, o Android disponibiliza um arquivo chamado “strings.xml”, que é encontrado na pasta “res” do nosso projeto, de acordo com a figura “Localização do arquivo strings.xml”:

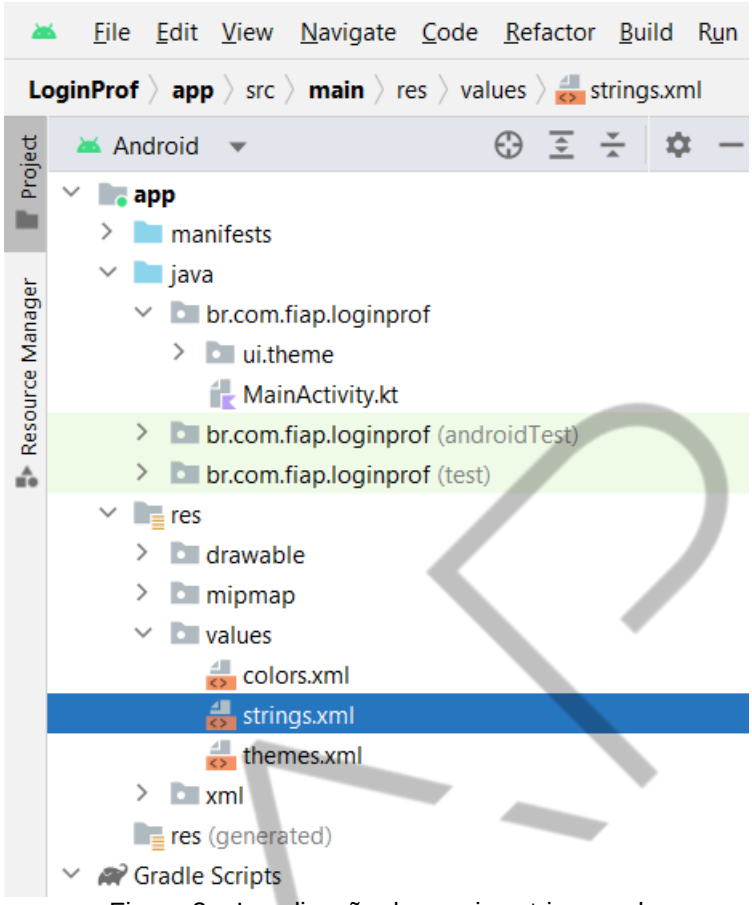


Figura 2 – Localização do arquivo strings.xml
Fonte: Google Imagens (2023)

A outra função do arquivo “strings.xml” é permitir que a nossa aplicação seja multi-idioma, ou seja, nosso aplicativo poderá manter arquivos “strings.xml” para os mais diversos idiomas para os quais desejamos a tradução.

O ideal é que o arquivo “strings.xml” default contenha os textos em um idioma padrão, que na maioria das vezes é o inglês.

1.1.3 Editando o arquivo strings.xml

Todos os textos que vemos em nossa aplicação estão inseridas de forma literal. Como já falamos, isso não é uma boa prática, então, vamos editar o arquivo “strings.xml” default da aplicação adicionando os textos inicialmente em inglês.

Vale lembrar que o arquivo “strings.xml” utiliza a notação “XML”, onde nós temos as “tags” que identificam os valores. O arquivo “strings.xml” do seu projeto deverá se parecer com a listagem abaixo:

```
<resources>
    <string name="app_name">Login Prof</string>

    // Strings personalizadas para o inglês
    // que será o idioma padrão da aplicação
    <string name="login">Login</string>
    <string name="subtitle">Please, provide your information</string>
    <string name="email">Enter your e-mail</string>
    <string name="password">Enter your password</string>
    <string name="enter">Enter</string>

</resources>
```

Código-fonte 2 – Arquivo strings.xml
Fonte: Elaborado pelo autor (2023)

Já temos o arquivo “strings.xml” com o conteúdo textual da aplicação em inglês, que será o idioma padrão da nossa aplicação.

1.1.4 Utilizando o arquivo strings.xml

Agora vamos substituir todas as entradas de texto da nossa aplicação para que utilize o arquivo “strings.xml”. Vamos começar pelos textos de título e subtítulo da aplicação. Abra o arquivo “MainActivity.kt” e faça os ajustes conforme a listagem abaixo:

```
. . . trecho omitido
Text(
  text = stringResource(id = R.string.login),
  fontSize = 32.sp,
  fontWeight = FontWeight.Bold,
  color = Color.Blue
)
Text(text = stringResource(id = R.string.subtitle))
Spacer(modifier = Modifier.height(48.dp))
. . . trecho omitido
```

Código-fonte 3 – Utilização do arquivo strings.xml
Fonte: Elaborado pelo autor (2023)

Nos trechos de código em destaque, utilizamos a função “stringResource” para fazermos referência aos textos que estão inseridos no arquivo “strings.xml”. A função “stringsResource” recebe como parâmetro a identificação do recurso de String que estamos queremos utilizar.

Execute a aplicação no emulador. O resultado esperado deve se parecer com a figura “Utilização arquivo strings.xml”:

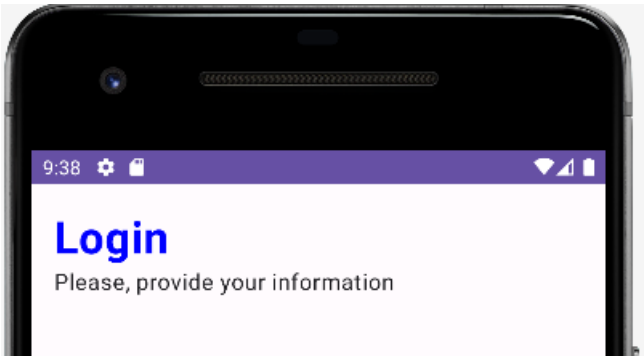


Figura 3 – Utilização do arquivo strings.xml
Fonte: Google Imagens (2023)

Observe que nossa aplicação está com o idioma em inglês. Isso ocorre devido ao fato de termos apenas o arquivo padrão que tem o texto neste idioma. Mas, e se quisermos que a aplicação utilize o idioma português? Simples, vamos criar outro arquivo “strings.xml” com o mesmo conteúdo do arquivo padrão, mas com as frases traduzidas.

Faça os ajustes para os textos das caixas de entrada e botão. Ao executar a aplicação, todo o texto deverá estar em inglês, conforme a figura “Aplicação traduzida para o inglês”:

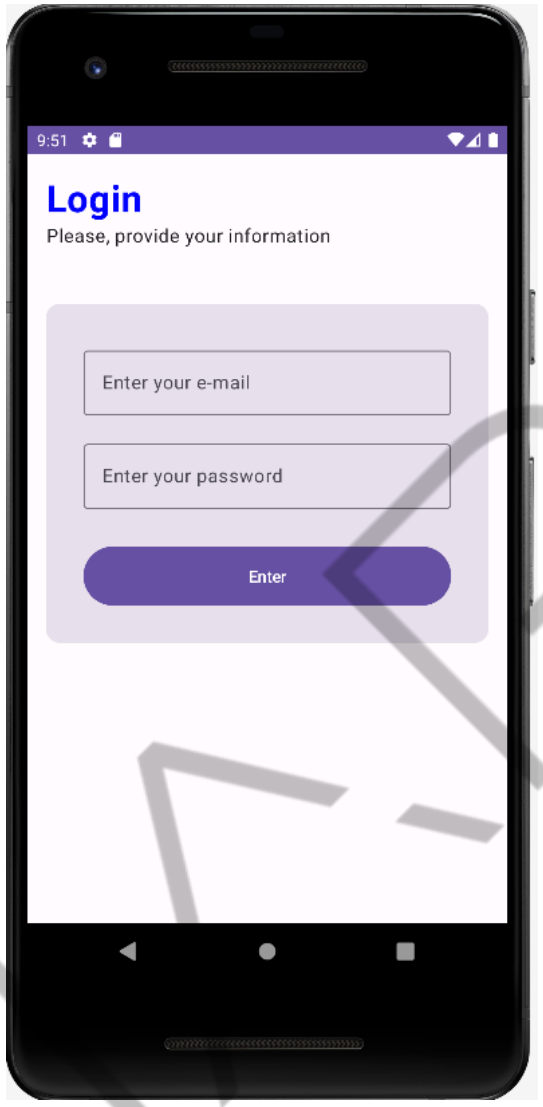


Figura 4 – Aplicação traduzida para o inglês
Fonte: Google Imagens (2023)

1.1.5 Traduzindo a aplicação para o idioma português

Agora, vamos traduzir a aplicação para o idioma português. Para isso, precisamos de outro arquivo “strings.xml” que deve conter as mesmas *tags* do arquivo padrão, mas com o valor traduzido. Então, clique com o botão direito do mouse na pasta “res”, aponte para “New” e clique em “Android Resource File”, de acordo com a figura “Criar arquivo de recurso”:

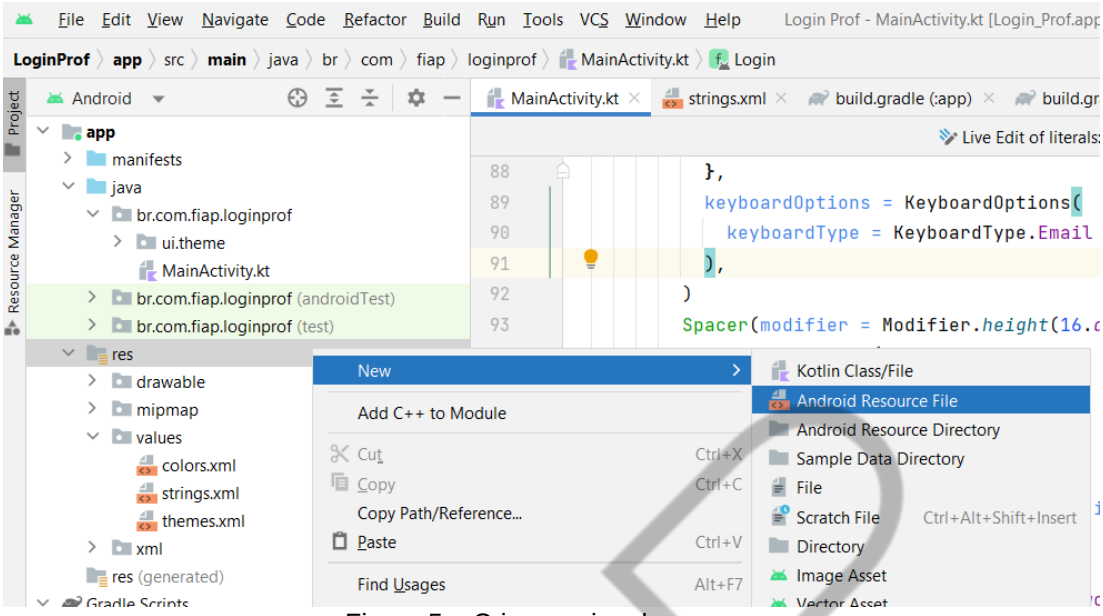


Figura 5 – Criar arquivo de recurso
Fonte: Google Imagens (2023)

Na janela “New Resource File” digite o nome do arquivo, que deve ser obrigatoriamente “strings”. Em “Available qualifiers”, selecione “Locale” e clique no botão “>>”, conforme a figura “Janela criar novo recurso”:

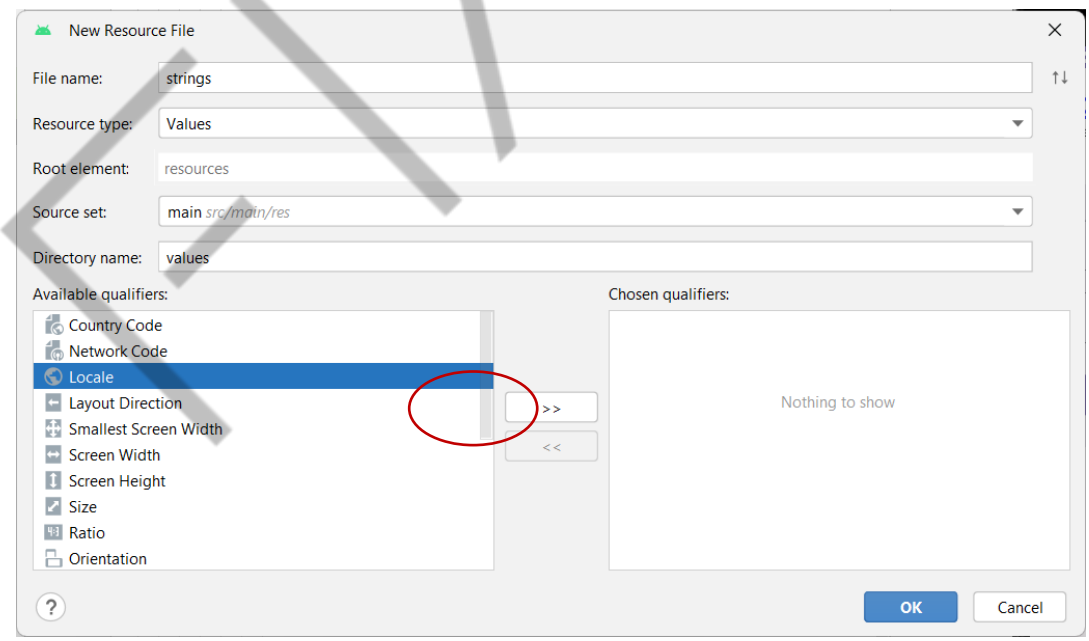


Figura 6 – Janela criar novo recurso
Fonte: Google Imagens (2023)

Na janela “New Resource File”, selecione o idioma “pt: Portuguese” e a região “BR: Brazil”, conforme a figura “Seleção de Idioma e País”. Clique no botão “OK” para

confirmar. A estrutura do projeto deverá ficar como na figura “Arquivo strings.xml (pt-rBR)”.

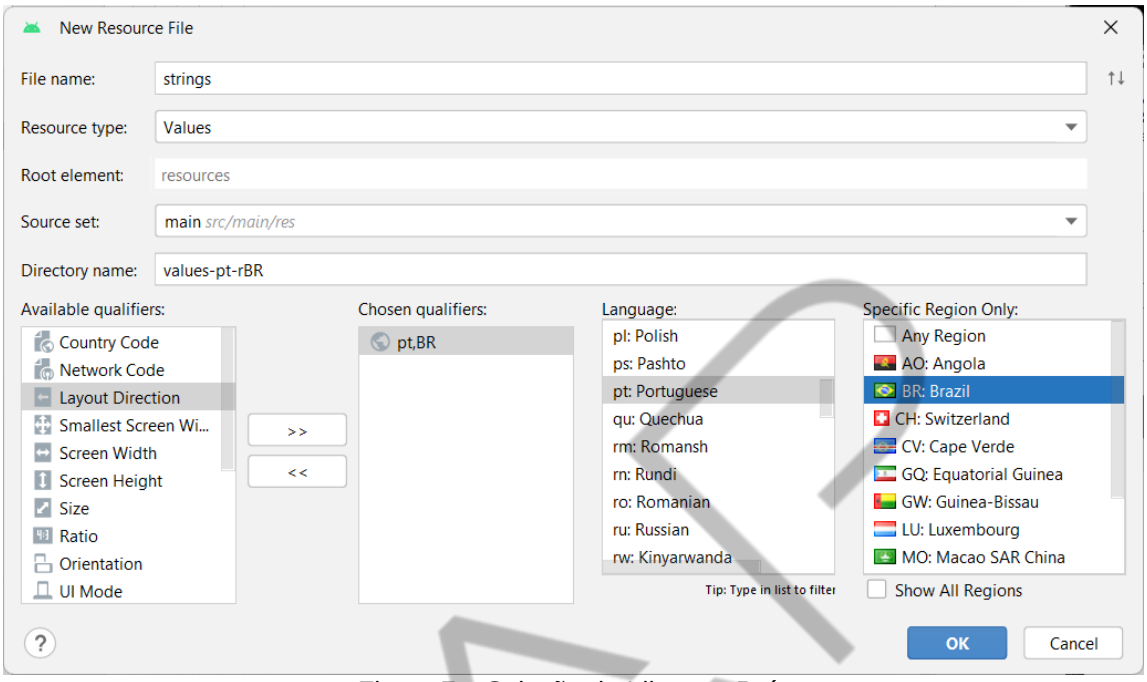


Figura 7 – Seleção de Idioma e País
Fonte: Google Imagens (2023)

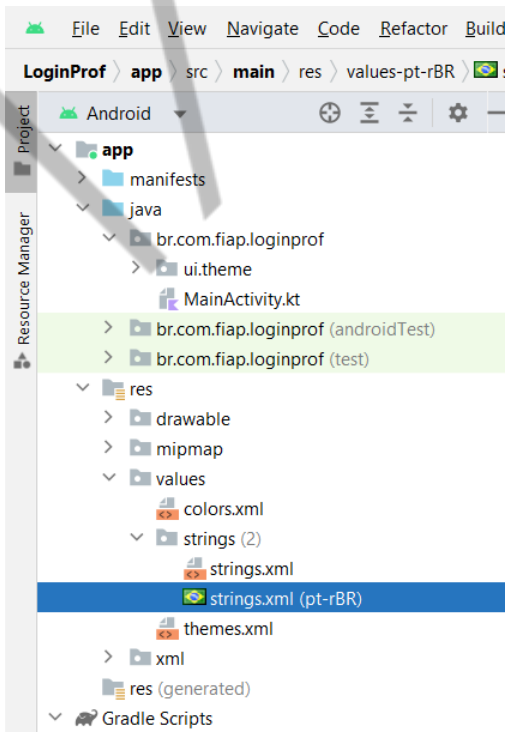


Figura 8 – Arquivo strings.xml (pt-rBR)
Fonte: Google Imagens (2023)

Copie todo o conteúdo do arquivo “strings.xml” default, abra o arquivo “strings.xml (pt-rBR)” e cole o conteúdo. O arquivo “strings.xml (pt-rBR)” deverá se parecer com a listagem abaixo:

```
<resources>

    <string name="app_name">Login Prof</string>

    // Strings personalizadas para o inglês
    // que será o idioma padrão da aplicação
    <string name="login">Login</string>
    <string name="subtitle">Por favor, informe os seus dados</string>
    <string name="email">Digeite o seu e-mail</string>
    <string name="password">Digite a sua senha</string>
    <string name="enter">Entrar</string>

</resources>
```

Código-fonte 4 – Arquivo strings.xml (pt-rBR)
Fonte: Elaborado pelo autor (2023)

Para testar a aplicação, troque o idioma do dispositivo para português do Brasil, conforme a figura “Idioma do dispositivo”:

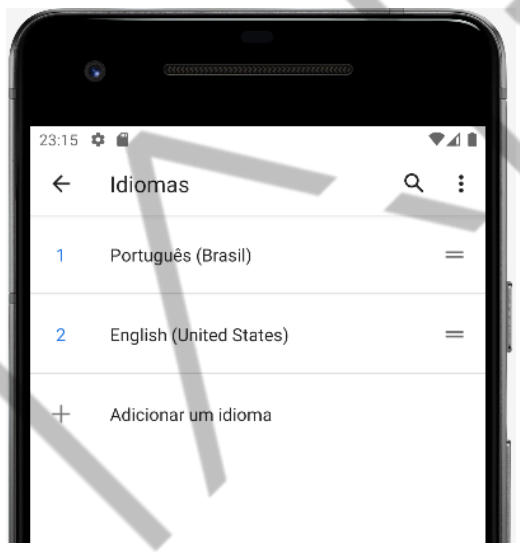


Figura 9 – Idioma do dispositivo
Fonte: Google Imagens (2023)

Com o idioma do dispositivo configurado para o idioma português, execute novamente a aplicação. O texto deverá estar em português, conforme a figura “Aplicativo com idioma português”:

Figura 10 – Aplicativo com idioma português
Fonte: Google Imagens (2023)

IMPORTANTE: caso o dispositivo utilize um idioma para o qual o aplicativo não tenha tradução, será utilizado o arquivo padrão, por isso é importante que o arquivo “strings.xml” default tenha seu conteúdo traduzido para o inglês, que será o idioma padrão do aplicativo.

2 VALIDAÇÃO DE ENTRADA DO USUÁRIO

Nem sempre o usuário insere as informações em um formulário da maneira como gostaríamos, e, muitas vezes, é necessário que a entrada de dados obedeça a alguma regra, como uma senha que deve ter no mínimo 8 caracteres. Também é comum o desenvolvedor somente habilitar alguma funcionalidade caso todos os campos obrigatórios tenham sido preenchidos. Vamos aprender agora como garantir que o usuário está inserindo os dados do modo que a aplicação espera para que não ocorram erros ou funcionamento inesperados.

2.1 Limitando o tamanho da senha

Podemos utilizar de vários artifícios para garantir que o usuário preencha uma caixa de entrada com os dados corretos. Uma forma que deve ser utilizada é abrir o teclado numérico quando o que esperamos do usuário sejam números, por exemplo.

Vamos validar a caixa de texto da senha para que permita a digitação de no máximo 8 caracteres. Então, na função “Login” do arquivo “MainActivity.kt”, vamos adicionar uma variável que irá armazenar o tamanho máximo que este campo permitirá. O código deverá se parecer com a listagem abaixo:

```
@Composable
fun Login() {

  var email by remember() {
    mutableStateOf("")
  }

  var password by remember {
    mutableStateOf("")
  }

  var tamanhoSenha = 8

. . . trecho de Código omitido
```

Código-fonte 5 – Criando variável
Fonte: Elaborado pelo autor (2023)

Para limitar a quantidade de texto que poderá ser digitado no campo senha, vamos implementar uma regra no parâmetro “onValueChange” do

“OutlinedTextField”, responsável por capturar a digitação da senha. O código deverá se parecer com a listagem abaixo:

```
OutlinedTextField(
  value = password,
  onValueChange = { if (it.length <= tamanhoSenha) password = it },
  modifier = Modifier.fillMaxWidth(),
  label = {
    Text(text = stringResource(id = R.string.password))
  },
  keyboardOptions = KeyboardOptions(
    keyboardType = KeyboardType.Password
  ),
  visualTransformation = PasswordVisualTransformation()
)
```

Código-fonte 6 – Limitando senha
Fonte: Elaborado pelo autor (2023)

Lembre-se que a cada tecla digitada em um campo de texto, disparamos a função “onValueChange”, que nos retorna o valor desta caixa de texto através da variável “it”. O que estamos fazendo neste exemplo é testando se o tamanho do texto retornado por “it” é menor ou igual à variável “tamanhoSenha”. Se o teste lógico resultar em “true”, a variável de estado “password” recebe o valor da variável “it”, caso contrário, a variável “password” não é mais atualizada e consequentemente o estado deste componente, ou seja, não conseguimos digitar mais.

Execute a aplicação e verifique se este comportamento está ocorrendo.

2.2 Validando o campo e-mail

O que vamos fazer agora é verificar se o campo e-mail foi preenchido ao ocorrer o clique no botão “Entrar”. Caso o e-mail não tenha sido preenchido, vamos exibir uma mensagem ao usuário logo abaixo deste campo. Para facilitar a nossa vida, o composable “OutlinedTextField” e suas variações possuem um atributo “isError”, que é um booleano que quando recebe o valor “true” modifica a aparência do componente. Vamos adicionar este parâmetro com o valor “true” para observar o seu comportamento. O código da aplicação deve se parecer com a listagem abaixo:

```
OutlinedTextField(
  value = email,
  onValueChange = { email = it },
  modifier = Modifier.fillMaxWidth(),
  label = {
    Text(text = stringResource(id = R.string.email))
  },
  isError = true,
  keyboardOptions = KeyboardOptions(
    keyboardType = KeyboardType.Email
  ),
)
```

Código-fonte 7 – Parâmetro isError
Fonte: Elaborado pelo autor (2023)

Ao executar a aplicação em um emulador, notamos que o campo referente ao e-mail está com as bordas na cor vermelha, indicando um problema, como podemos observar na figura “Testando o parâmetro isError”:

Figura 11 – Testando o parâmetro isError
Fonte: Google Imagens (2023)

Para que este comportamento ocorra de uma forma mais adequada, vamos criar uma variável de estado do tipo booleano que manterá o estado de erro para este campo. Ao clicarmos no botão, se o campo estiver vazio, mudamos o valor da variável para “true”. O código desta implementação está disponível na listagem abaixo:

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Login() {

  var email by remember() {
    mutableStateOf("")
  }

  var password by remember {
    mutableStateOf("")
  }

  var emailError by remember {
    mutableStateOf(false)
  }

  var tamanhoSenha = 8

  Column(modifier = Modifier.padding(16.dp)) {
    Text(
      text = stringResource(id = R.string.login),
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color.Blue
    )
    Text(text = stringResource(id = R.string.subtitle))
    Spacer(modifier = Modifier.height(48.dp))
    Card(
      modifier = Modifier
        .fillMaxWidth()
    ) {
      Column(
        modifier = Modifier
          .fillMaxWidth()
          .padding(32.dp)
      ) {
        OutlinedTextField(
          value = email,
          onValueChange = { email = it },
          modifier = Modifier.fillMaxWidth(),
          label = {
            Text(text = stringResource(id = R.string.email))
          },
          isError = emailError,
          keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Email
          ),
        )
        Spacer(modifier = Modifier.height(16.dp))
        OutlinedTextField(
          value = password,
          onValueChange = {
            if (it.length <= tamanhoSenha) password = it
          },
          modifier = Modifier.fillMaxWidth(),
          label = {
            Text(text = stringResource(id = R.string.password))
          },
          keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Password
          ),
```

```
            visualTransformation = PasswordVisualTransformation()
          )
          Spacer(modifier = Modifier.height(32.dp))
          Button(onClick = {
            if(email.isEmpty()){
              emailError = true
            }
          }) {
            Text(
              text = stringResource(id = R.string.enter),
              modifier = Modifier
                .padding(8.dp)
                .fillMaxWidth(),
              textAlign = TextAlign.Center
            )
          }
        }
      }
    }
  }
```

Código-fonte 8 – Validação do campo e-mail
Fonte: Elaborado pelo autor (2023)

Nos trechos em destaque do código acima, efetuamos as seguintes implementações:

1 - Criamos uma variável de estado chamada “emailError” que é inicializada com o valor “false”;

2 – Atribuímos a variável “emailError” ao parâmetro “isError” do componente responsável pelo e-mail;

3 – Na função “onClick” do botão “Entrar”, verificamos se a variável de estado “email” está vazia. Se o teste lógico resultar em “true”, alteramos o valor da variável “emailError” para “true”;

4 – A caixa de texto responsável pela entrada do e-mail sofrerá a recomposição, tendo agora a borda vermelha.

Para nosso tratamento de erro ficar ainda mais interessante, vamos incluir uma mensagem de texto logo abaixo da caixa de texto do e-mail, que deverá aparecer somente quando o e-mail estiver vazio. A listagem de código abaixo exemplifica esta implementação:

```
. . . trecho omitido
OutlinedTextField(
  value = email,
  onValueChange = { email = it },
  modifier = Modifier.fillMaxWidth(),
  label = {
    Text(text = stringResource(id = R.string.email))
  },
  isError = emailError,
  keyboardOptions = KeyboardOptions(
    keyboardType = KeyboardType.Email
  ),
)
if(emailError){
  Text(
    text = "E-mail é obrigatório!",
    modifier = Modifier.fillMaxWidth(),
    color = Color.Red,
    textAlign = TextAlign.End
  )
}
Spacer(modifier = Modifier.height(16.dp))
. . . trecho omitido
```

Código-fonte 9 – Mensagem de erro personalizada
Fonte: Elaborado pelo autor (2023)

No código acima, inserimos após o e-mail uma condicional que verifica o valor da variável “emailError”. Se esta variável for “true”, o componente de texto com a mensagem de erro será exibido.

Quando clicamos no botão “Entrar” sem preenchermos o e-mail, deveremos ver a mensagem de erro conforme a figura “Mensagem de erro personalizada”:

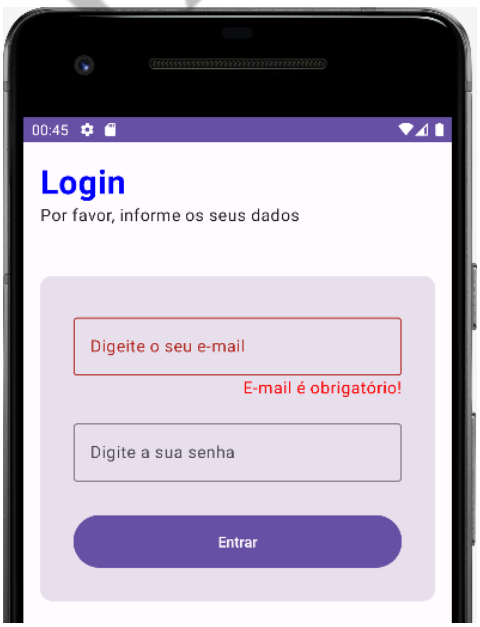


Figura 12 – Mensagem de erro personalizada
Fonte: Google Imagens (2023)

Queremos que a mensagem desapareça quando o usuário começar a digitar o e-mail. A listagem de código abaixo exemplifica a implementação:

```
OutlinedTextField(
  value = email,
  onValueChange = {
    email = it
    if (email.length > 0) emailError = false
  },
  modifier = Modifier.fillMaxWidth(),
  label = {
    Text(text = stringResource(id = R.string.email))
  },
  isError = emailError,
  keyboardOptions = KeyboardOptions(
    keyboardType = KeyboardType.Email
  ),
)
```

Código-fonte 10 – Verificando a variável emailError durante digitação
Fonte: Elaborado pelo autor (2023)

Execute a aplicação em um emulador e clique no botão “Entrar” com o e-mail vazio. Neste caso, a mensagem de erro deverá aparecer. Comece a digitar o e-mail e a mensagem de erro deverá sumir.

CONCLUSÃO

Tornar a aplicação acessível por usuários de diferentes idiomas é sem dúvida alguma um dos recursos mais importantes da plataforma Android, e que poderá levar a sua aplicação para além das fronteiras da sua região.

Além disso, as aplicações Android manipulam, basicamente, dados que são fornecidos pelos usuários, e saber como impedir que informações incorretas ou fora do padrão sejam inseridos nos permite melhorar bastante a experiência do usuário evitando erros e comportamentos inesperados.

Agora só depende de você explorar os recursos que aprendemos neste capítulo adicionando novos idiomas ao aplicativo e validações de entrada!

REFERÊNCIAS

DEVELOPERS. **Recursos de String**. 2023. Disponível em: <https://developer.android.com/guide/topics/resources/string-resource?hl=pt-br>. Acesso em: 7 jul. 2023.