

# Kodular

Desenvolvimento Android sem código



© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

*Edição*

Vivian Matsui  
Sabrina Barbosa

[2021]

Casa do Código

Rua Vergueiro, 3185 - 8º andar  
04101-300 – Vila Mariana – São Paulo – SP – Brasil  
[casadocodigo.com.br](http://casadocodigo.com.br)





Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código é a editora do Grupo Alura, grupo que nasceu da vontade de criar uma plataforma de ensino com o objetivo de incentivar a transformação pessoal e profissional através da tecnologia.

Juntas, as empresas do Grupo constroem uma verdadeira comunidade colaborativa de aprendizado em programação, negócios, design, marketing e muito mais, oferecendo inovação na evolução dos seus alunos e alunas através de uma verdadeira experiência de encantamento.

Venha conhecer os cursos da Alura e siga-nos em nossas redes sociais.

 [alura.com.br](http://alura.com.br)

 @casadocodigo

 @casadocodigo

# ISBN

Impresso e PDF: 978-85-7254-042-1

EPUB: 978-85-7254-043-8

MOBI: 978-85-7254-044-5

Caso você deseje submeter alguma errata ou sugestão, acesse  
<http://erratas.casadocodigo.com.br>.

# PREFÁCIO

Com o grande crescimento dos aplicativos Android para celulares e tablets que vivemos em nossos dias e consequentemente a necessidade de formar programadores para este sistema operacional, há uma grande corrida para se preparar para este universo. Apesar de a IDE oficial para o desenvolvimento Android ser o Android Studio da empresa Google, percebemos algumas dificuldades em nosso dia a dia em sala de aula com a utilização desse ambiente como requisitos elevados de hardware e conhecimentos da complexa linguagem de programação Java.

Encontramos um caminho que transpassasse essas barreiras e apresentamos uma nova solução de desenvolvimento para o sistema operacional Android: a plataforma **Kodular**.

Esta plataforma é um ambiente *Drag and Drop*, ou seja, sua programação baseia-se em clicar e arrastar blocos de códigos para desenvolver seu app.

O Kodular é utilizado em mais de 200 países, com mais de 170.000 usuários ativos. Mais de 400.000 projetos já foram desenvolvidos e centenas de milhares de pessoas em todo o mundo usam um aplicativo feito com Kodular todos os dias!

Para demonstrar parte do potencial do Kodular, o projeto que será apresentado neste livro fará a integração do app com uma página de dados cadastrados na internet. Trata-se de um app colaborativo, onde todos os leitores e leitoras desenvolverão o mesmo projeto e as informações cadastradas serão compartilhadas com todos.

Este é um livro que pode ser considerado técnico e prático, pois no transcorrer dos capítulos, você será apresentado/a às funcionalidades do aplicativo que se integrará com um banco de dados online.

## A QUEM SE DESTINA ESTE LIVRO?

Todas as pessoas estão convidadas a lê-lo pois explicaremos passo a passo todos os caminhos para a resolução dos problemas. Sugerimos que para um melhor aproveitamento o/a leitor/a possua algum conhecimento de Lógica de Programação e um pouco de intimidade com a linguagem PHP pois assim seus aplicativos se destacarão.

Neste livro serão apresentadas técnicas diferentes de acesso a banco de dados daquelas abordadas em nosso livro **App Inventor: seus primeiros aplicativos Android**, publicado pela editora Casa do Código, portanto a partir de agora você conhecerá mais a fundo o potencial da plataforma Kodular para desenvolvimento de aplicativos Android.

## ORGANIZAÇÃO DO LIVRO

Capítulo 1: breve introdução à plataforma de desenvolvimento Kodular para Android.

Capítulo 2: explanação do projeto que será desenvolvido no decorrer da leitura deste livro.

Capítulos 3, 4, 5, 6 e 7: configuração e montagem de todos os leiautes das telas que utilizaremos no app. Passo a passo, serão

apresentadas as ferramentas que utilizaremos e suas configurações para um melhor aproveitamento.

Capítulo 8: demonstramos e explicamos os comandos do site que realiza o cadastro dos dados que será utilizado no app que será desenvolvido.

Capítulos 9, 10, 11 e 12: conheceremos os blocos de comandos que realizarão toda a programação necessária para o perfeito funcionamento do app e integração com o banco de dados online.

Capítulo 13: apresentamos e comentamos todas as linhas de comando dos arquivos em linguagem PHP que serão utilizados no projeto.

## SOBRE OS AUTORES

**Nelson Fabbri Gerbelli** é tecnólogo em processamento de dados, pós-graduado em Análise de Sistema, Pedagogo e licenciado nas áreas de Tecnologia da Informação e Matemática. Iniciou como docente na área de TI em 1995. Em 2001, ingressou no Centro Estadual de Educação Tecnológica Paula Souza, nas Escolas Técnicas do Estado de São Paulo, lecionando as disciplinas de desenvolvimento de softwares com o ambiente Microsoft Visual Studio.net, tecnologia de desenvolvimento de aplicativos mobile, programação para internet, entre outras. No ano de 2009, tornou-se corredor amador e desde então muitos dos problemas de desenvolvimento são resolvidos durante a prática das corridas.

**Valéria Helena P. Gerbelli** é tecnóloga em processamento de dados, pedagoga, licenciada em Tecnologia da Informação e pós-graduada em Educação Profissional e Tecnológica. Iniciou carreira na extinta Equipamentos Villares em São Bernardo do Campo como estagiária na área de Suporte e Desenvolvimento de Sistemas, onde permaneceu por 9 anos. Iniciou como docente na área de TI em 1995 e ingressou em 1998 no Centro Estadual de Educação Tecnológica Paula Souza, nas Escolas Técnicas do Estado de São Paulo, lecionando as disciplinas de desenvolvimento de softwares com o ambiente Microsoft Visual Studio.net, banco de dados, programação para internet entre outras.

Ambos são autores do livro **App Inventor: seus primeiros aplicativos Android** pela editora Casa do Código.

# Sumário

<b>1 Introdução</b>	<b>1</b>
1.1 O que vamos aprender	2
1.2 Tecnologias utilizadas	4
<b>2 Conhecendo o projeto</b>	<b>5</b>
2.1 Como funciona?	6
2.2 A estrutura do banco de dados do app	12
2.3 Descrição das tabelas e seus campos	12
2.4 Instale e conheça o app	15
<b>3 Iniciando o desenvolvimento</b>	<b>16</b>
3.1 Por que o nome Kodular?	16
3.2 Iniciando o desenvolvimento	17
3.3 Desenvolvendo a primeira tela do projeto	21
<b>4 Instalando o app</b>	<b>37</b>
4.1 Importando um projeto	39
<b>5 Desenvolvendo a tela Principal</b>	<b>42</b>
5.1 Acrescentando uma nova Screen	43

Sumário	
	Casa do Código
5.2 Uma barra de títulos personalizada	45
5.3 Área de exibição das corridas	47
5.4 Importando uma extensão	49
<b>6 Design da tela de Login e Cadastro</b>	<b>55</b>
6.1 Configurando a tela de Login	58
6.2 A tela de Cadastro de Usuários	64
<b>7 A tela de detalhes da corrida</b>	<b>70</b>
7.1 Objeto Rating Bar	75
7.2 Inserindo um Mapa na Screen	76
7.3 Preparando a área de avaliação	79
<b>8 Cadastrando as corridas</b>	<b>83</b>
8.1 Parte em HTML	85
8.2 Parte em PHP	87
<b>9 Programando a tela de Destaques</b>	<b>92</b>
9.1 Acessando a área dos blocos	93
9.2 Inserindo os primeiros blocos	95
9.3 Declarando variável de programação	102
9.4 Exibindo as corridas em Destaque	104
9.5 Selecionando uma corrida em destaque	115
9.6 O botão flutuante	120
<b>10 Programando a tela Principal</b>	<b>123</b>
10.1 Recebendo a lista de corridas	125
10.2 Declarando variáveis locais	127
10.3 Selecionando uma opção do CardView	136
10.4 Compartilhando uma informação	137

10.5 Fechando uma Screen	140
10.6 Filtrando os dados para exibição	141
<b>11 Programando o Login e o Cadastro</b>	<b>146</b>
11.1 Alternando a visibilidade das funções	146
11.2 Realizando o Login	148
11.3 Recebendo a confirmação do login	153
11.4 Cadastrando um novo usuário	157
11.5 Verificando se o cadastro foi realizado	162
<b>12 Programando a tela dos Detalhes</b>	<b>165</b>
12.1 Programando a inicialização da Screen	166
12.2 Exibindo os dados da corrida selecionada	168
12.3 Selecionando a Latitude e a Longitude	170
12.4 Procedure para exibir os comentários	175
12.5 Exibindo a localização no mapa	182
12.6 Inserindo uma avaliação	183
<b>13 Arquivos de back-end</b>	<b>192</b>
13.1 Conecta.php	192
13.2 Destaques.php	193
13.3 Query.php	194
13.4 Login.php	196
13.5 Cadastro.php	198
13.6 Seleciona_item.php	200
13.7 Seleciona_comentarios.php	201
13.8 Comentarios.php	202

Versão: 26.2.12

## CAPÍTULO 1

# INTRODUÇÃO

Hoje em dia é comum encontrar ferramentas que auxiliam os usuários sem muito conhecimento técnico a criar aplicativos Android para smartphones, tablets e outros dispositivos.

Uma dessas ferramentas é o **Kodular**, que permite converter nossas ideias em aplicativos Android utilizando uma plataforma online de programação visual e intuitiva de arrastar e soltar. Você não precisa ter muitos conhecimentos em linguagem de programação, pois o Kodular já traz em blocos os códigos necessários para programar.

Os projetos desenvolvidos no Kodular ficam hospedados com segurança no Google Cloud Platform, assim não é necessário ficar fazendo backups nem download. Outro ponto importante é que é gratuito, sem taxas ou planos para utilizá-lo.

## Onde tudo começou

O Kodular foi desenvolvido com base no **MIT App Inventor** para facilitar ainda mais a codificação de aplicativos Android. Ele recebe constantes atualizações disponibilizando novos recursos aos usuários e facilitando cada vez mais o desenvolvimento de aplicativos.

O App Inventor iniciou seu desenvolvimento no Google em 2007 e lá permaneceu até 2010, quando foi movido para o MIT (Massachusetts Institute of Technology). Ele permite que os recém-chegados à programação de computador criem aplicativos para o sistema operacional Android, usando uma interface gráfica, do tipo *Drag and Drop* (arrastar e soltar). Ele ainda recebe atualizações e continua sendo uma excelente plataforma para iniciantes desenvolverem seus primeiros aplicativos Android.

Kodular é propriedade da Makeroid, que é uma empresa registrada na Holanda.

Se você já tiver conhecimentos em uma das plataformas de desenvolvimento Android, como o MIT App Inventor ou Thunkable, não terá dificuldades em desenvolver com o Kodular, entretanto, qualquer pessoa que queira aprender Kodular não encontrará dificuldades.

## 1.1 O QUE VAMOS APRENDER

Sabe aquelas corridas que estão em alta, como a **Corrida de São Silvestre, Maratona do Rio de Janeiro, Volta Internacional da Pampulha** e tantas outras, onde há bastante participantes? Somos aficionados por elas. Não seria legal se existisse um app em que, ao participar de uma delas, o/a atleta pudesse avaliá-la e deixar suas impressões sobre o evento para todos os corredores?

Ao término da leitura e prática do projeto proposto pelos autores, você desenvolverá um aplicativo colaborativo sobre *Avaliação de Corridas de Rua*.

Esse tema é apenas um exemplo que adotamos para

demonstrar as ferramentas e potenciais presentes na plataforma Kodular, porém você poderá adaptar tudo o que será apresentado, desenvolvendo seus próprios apps de acordo com suas necessidades.

Veja alguns dos recursos que aprenderemos durante o desenvolvimento do app de Avaliação de Corridas de Rua.

1. Desenvolvimento de layout em diversas telas;
2. Instalação do app em seu dispositivo;
3. Criação de barra de título;
4. Listagem de registros;
5. Importação de extensão;
6. Login e cadastro;
7. Avaliação da corrida - comentários e nota;
8. Exibição de mapa;
9. Compartilhamento de informação, entre outros.

A imagem a seguir demonstra a utilização de alguns dos recursos citados.

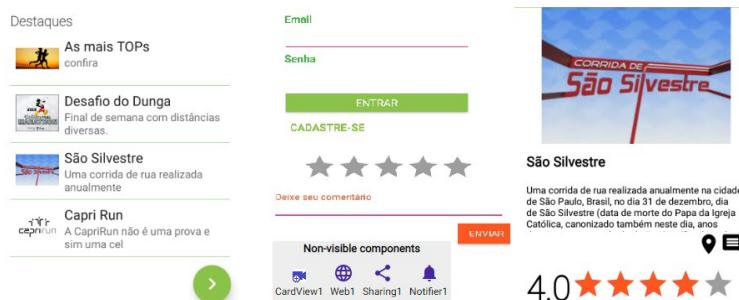


Figura 1.1: Visualização de alguns recursos

## 1.2 TECNOLOGIAS UTILIZADAS

Além de utilizar a plataforma Kodular para o desenvolvimento do leiaute de interatividade com o usuário e a sua programação em blocos, veremos também a parte que processa os dados enviados pelo app para o banco de dados que está hospedado em um servidor. Esse processamento se dará por programas escritos em linguagem PHP, que vão interagir com o banco de dados em MySQL.

No próximo capítulo, vamos começar a explicar em maiores detalhes como será o projeto do app que vamos desenvolver. Bons estudos.

## CAPÍTULO 2

# CONHECENDO O PROJETO

Antes de começarmos com o desenvolvimento de um aplicativo, temos que realizar uma análise minuciosa do que realmente desejamos, para que o produto esteja de acordo com o que foi idealizado e não apareçam alterações imprevistas durante a sua programação.

As modificações durante o desenvolvimento consomem muito tempo de replanejamento do programador, exigem diversas adaptações nos códigos e podem culminar até na alteração do objetivo final, com um aplicativo que se desviou do planejado inicialmente. A análise de sistema é uma prática que realmente deve ser realizada para ajudar a nos guiar durante o processo de desenvolvimento.

Vamos descrever o projeto de aplicativo que vamos desenvolver durante a leitura deste livro: o app **AVALIAÇÃO DE CORRIDAS DE RUA**, que tem como finalidade conhecer os detalhes de uma corrida de rua que foi cadastrada no sistema, seu local de realização exibido no mapa, a nota média de avaliação, exibição dos comentários dos demais usuários e a possibilidade de o próprio usuário realizar sua avaliação e comentários.

## 2.1 COMO FUNCIONA?

O pedestranismo ou a corrida é a modalidade mais tradicional do atletismo. Devido à quantidade de eventos esportivos realizados, fica muito difícil escolher ou obter informações sobre quais são as melhores e aquelas que apresentam algum tipo de problema em sua realização.

Para facilitar a obtenção dessas informações, após a realização da inserção de uma ou mais corridas de rua através de um site que já está online e que será descrito em um capítulo adiante, elas serão exibidas no app para conhecimento dos usuários e realização de avaliação e comentários sobre uma corrida. Portanto, não será necessário criar o site para o desenvolvimento do app.

Trata-se de um aplicativo colaborativo, que possibilitará também compartilhar através de suas redes e contatos a avaliação que está realizando no momento.

O usuário poderá visualizar todos os detalhes das corridas cadastradas, mas para realizar comentários e avaliações este deverá fazer seu cadastro e posterior login pelo próprio aplicativo. O app também apresentará as corridas que mais se destacaram pela avaliação dos usuários em uma tela específica para este fim.

Apresentaremos as telas do app para que possamos ter um roteiro a seguir durante o desenvolvimento.

A tela de abertura do app, mostrada na próxima imagem, exibirá uma listagem com as 3 corridas com melhores avaliações realizadas pelos usuários. Quando o usuário clicar em uma corrida específica, será exibida a tela com todos os detalhes desta corrida.



## Destaques

 As mais TOPs  
confira

---

 Desafio do Dunga  
Final de semana com distâncias  
diversas.

 São Silvestre  
Uma corrida de rua realizada  
anualmente

---

 Capri Run  
A CapriRun não é uma prova e  
sim uma cel



Figura 2.1: Tela inicial do app

Ao clicar no botão flutuante que se encontra no rodapé da tela na cor verde, será exibida uma segunda tela com a listagem de todas as corridas cadastradas. Veja demonstração na imagem a seguir.



## São Silvestre

Av. Paulista, 900

Uma corrida de rua realizada anualmente na  
cidade



Figura 2.2: Tela com todas as corridas cadastradas

Nela, ao clicarmos em uma corrida específica, todos os detalhes da corrida serão apresentados ao usuário.

No topo da página, temos um botão para retornar para a página principal e um botão que realizará um filtro de exibição de corridas. Note que abaixo de cada corrida existe um botão de compartilhamento. Ao clicar nele, sua função será compartilhar com seus contatos ou mídia social qual corrida o usuário está verificando.

A tela de detalhamento de corridas exibirá o logotipo da

corrida, o título, a descrição, os pontos da avaliação, os comentários dos demais usuários e um mapa do local da largada. Veja na imagem a seguir.



### São Silvestre

Uma corrida de rua realizada anualmente na cidade de São Paulo, Brasil, no dia 31 de dezembro, dia de São Silvestre (data de morte do Papa da Igreja Católica, canonizado também neste dia, anos depois, no quarto século da Era Cristã) e de onde vem o seu nome.<sup>[1]</sup>

A corrida, a mais famosa e tradicional do Brasil e a da América do Sul, tem um percurso atual de 15 km pelo centro de São Paulo e é uma corrida mista desde 1975, quando começou a participação oficial das mulheres. Entre 1925, ano de sua criação e 1944, foi disputada apenas por corredores brasileiros.



4.0 

Figura 2.3: Tela dos detalhes da corrida selecionada

Note que na tela de detalhes existe um marcador de local. Ao clicar nele, será exibido um mapa com o local da largada da corrida. Ao lado do botão de local, existe um botão para comentários, e ao clicar nele será exibida uma caixa onde o usuário poderá inserir um comentário sobre a corrida e realizar a sua avaliação. Veja na imagem a seguir esta opção.



Figura 2.4: Espaço para inserção da avaliação

Para poder avaliar uma corrida, o usuário deverá estar logado no app. Caso não esteja, será exibida a tela de login e cadastro de usuários, conforme imagem a seguir.



The screenshot shows a login screen with two input fields. The first field is labeled "Email" and the second is labeled "Senha", both in green text. Below these fields is a large green button with the white text "ENTRAR". At the bottom of the screen, there is a smaller green button with the white text "CADASTRE-SE".

Figura 2.5: Tela de login de usuário cadastrado

A tela de cadastro, primeiramente, solicitará que o usuário realize seu login. Caso o usuário ainda não tenha se cadastrado, basta clicar no botão **cadastre-se** para realizar a tarefa, conforme a próxima imagem.



**Nome**

---

**Email**

---

**Senha**

---

**SALVAR**

**Fazer Login**

Figura 2.6: Tela de cadastro de usuário

Após o cadastro, deverá ser realizado o login para acessar a área de avaliação e comentários das corridas.

## 2.2 A ESTRUTURA DO BANCO DE DADOS DO APP

As informações disponíveis para exibição do aplicativo estarão armazenadas em um banco de dados online, em um provedor hospedeiro previamente preparado para este fim. Portanto, todos os comentários inseridos estarão visíveis para os demais usuários.

Teremos um banco de dados e três tabelas para armazenamento desses dados. A estrutura das tabelas que vamos utilizar é a exibida a seguir.

u902122380_aval corridas	u902122380_aval usuarios	u902122380_aval comentarios
<ul style="list-style-type: none"><li>id : int(10) unsigned</li><li>nome : varchar(155)</li><li>descricao : text</li><li>foto : text</li><li>telefone : varchar(20)</li><li>google_maps : text</li><li>endereco : text</li><li>horarios : text</li><li># media : float</li></ul>	<ul style="list-style-type: none"><li>id : int(11)</li><li>nome : varchar(40)</li><li>email : varchar(40)</li><li>senha : varchar(10)</li></ul>	<ul style="list-style-type: none"><li>id : int(11)</li><li>descricao : varchar(140)</li><li># avaliacao : float</li><li># corrida_id : int(3)</li><li># user_id : int(3)</li></ul>

Figura 2.7: Estrutura do Banco de Dados

## 2.3 DESCRIÇÃO DAS TABELAS E SEUS CAMPOS

Vamos conhecer a estrutura de cada tabela do banco de dados e sua função. É importante lembrar que os campos devem ter o seu tipo de acordo com os valores que vão receber.

A tabela **usuarios**, que armazenará os dados de cadastro dos usuários do app, terá os seguintes campos para compor sua estrutura:

- O campo **id**, que é do tipo numérico e enumerado automaticamente, pois receberá um número para cada novo registro para identificar o usuário cadastrado.
- O campo é o **nome**, que deve ser do tipo *Varchar*, pois receberá qualquer tipo de caractere para cadastrar o nome do usuário do app.
- O campo **email** segue a mesma ideia do campo **nome** porque armazenará o e-mail do usuário cadastrado.
- O campo **senha**, que também possui o tipo *Varchar* para armazenar a senha digitada pelo usuário.

A tabela **corridas**, que armazenará as informações de cada corrida cadastrada para avaliação, será composta também por:

- Um campo **id**, do tipo numérico, e enumerado automaticamente para receber cada nova corrida cadastrada.
- Um campo **nome**, do tipo *Varchar*, que receberá o nome da corrida.
- O campo **descricao**, que será também do tipo *Varchar*.
- O campo **foto** do tipo *Varchar*, porque nele vamos armazenar o texto com o endereço do arquivo da foto que será exibida.
- O campo **telefone**, que armazenará o telefone de contato da organização da corrida.
- O campo **google\_maps**, que armazenará a longitude e

latitude do local da corrida para exibição no mapa.

- O campo **endereco**, para armazenar o endereço da corrida.
- O campo **horario** servirá para o horário em que se iniciará a corrida.
- O campo **media**, que indicará o valor da média de avaliação de todos os usuários.

Com exceção do campo **id**, todos os demais campos desta tabela serão do tipo *Varchar*, seguindo a mesma ideia das informações que serão armazenadas.

A tabela **comentarios** armazenará os comentários que cada usuário realizar e possuirá:

- O campo **id**, que tem a mesma função dos outros campos de mesmo nome das outras tabelas.
- O campo **descricao**, que será do tipo *Varchar*, e guardará o comentário realizado sobre a corrida.
- O campo **avaliacao**, que será do tipo *Float* pois armazenará a avaliação numérica realizada pelo usuário, podendo ser com até uma casa decimal.
- O campo **corrida\_id**, que servirá para realizar um relacionamento entre a tabela de **comentarios** com a tabela de **corridas**, pois assim conseguimos identificar que o comentário realizado pertence à corrida especificada.
- O campo **user\_id**, que tem a finalidade semelhante ao campo **corrida\_id**, porém seu relacionamento é com a

tabela de **usuarios**.

## 2.4 INSTALE E CONHEÇA O APP

O projeto final está disponível na Play Store da Google, através do link [https://play.google.com/store/apps/details?id=io.kodular.nelfabbri.Avaliacao\\_corrida\\_de\\_rua](https://play.google.com/store/apps/details?id=io.kodular.nelfabbri.Avaliacao_corrida_de_rua). Você já pode acessá-lo e instalá-lo no seu dispositivo para um melhor conhecimento do aplicativo que vamos desenvolver.

Todos os materiais utilizados para o desenvolvimento do App de Avaliação de Corridas de Rua estão disponíveis no link: [http://www.nelfabbri.com/avalia\\_run](http://www.nelfabbri.com/avalia_run).

Aqui apresentamos uma breve descrição das principais funções do projeto que vamos desenvolver. Nos próximos capítulos, vamos mostrar o ambiente de desenvolvimento do Kodular e começaremos a desenvolver a programação visual das telas, ou seja, vamos criar os leiautes do app de **AVALIAÇÃO DE CORRIDAS DE RUA** passo a passo, para posterior implementação da programação que dará a funcionalidade do app.

Essa é apenas uma sugestão de projeto que vamos desenvolver, porém ao término da leitura deste livro, você estará habilitado/a a desenvolver diversos outros temas utilizando os recursos aprendidos, tudo vai depender da sua criatividade e necessidade.

## CAPÍTULO 3

# INICIANDO O DESENVOLVIMENTO

Após a descrição do app no capítulo anterior, chegou a hora de começarmos a realizar o desenvolvimento do projeto. Como já dissemos, primeiramente vamos realizar a parte visual do aplicativo, deixando toda a programação que realiza a interatividade com o usuário reservada para capítulos posteriores.

Vamos conhecer a Kodular, a plataforma utilizada para o desenvolvimento de aplicativos Android, que já sabemos que nos facilitará a codificação através de blocos de comandos predefinidos. Quem já tem conhecimento no MIT App Inventor, não encontrará dificuldades com este ambiente e quem não o conhece terá uma excelente oportunidade de aprendizagem.

## 3.1 POR QUE O NOME KODULAR?

A palavra Kodular vem da junção de duas palavras:

**Kode** deriva de Code. Foi decidido substituir a letra C por um K, pois assim ficaria diferente da palavra original. Também por fornecer um serviço de pseudocódigo, (em blocos), então é melhor chamá-lo usando uma palavra diferente, mas ao mesmo tempo que

lembre a sua origem.

**Modular**, por ser oferecido uma “codificação modular”, ou seja, com vários módulos já disponíveis para utilização, sem a necessidade de criá-los com longos códigos.

Escolhemos essa nova plataforma de desenvolvimento pois, além de possuir todos os recursos do MIT App Inventor, apresenta diversos novos recursos e facilidades que vamos explorar nesta obra.

## 3.2 INICIANDO O DESENVOLVIMENTO

Vamos acessar a plataforma através do endereço <http://www.kodular.io>, preferencialmente com o navegador Google Chrome. Ao entrar na página, será exibida a seguinte tela:



Figura 3.1: Página inicial do Kodular

Para começarmos o desenvolvimento, necessitamos nos logar na plataforma. Para isso clique em um dos botões CREATE APPS! visíveis na tela e será exibida a próxima imagem. É necessário que você possua uma conta de e-mail do **Gmail** (que é um serviço gratuito de email do Google) para realizar o login e utilizar o

espaço.

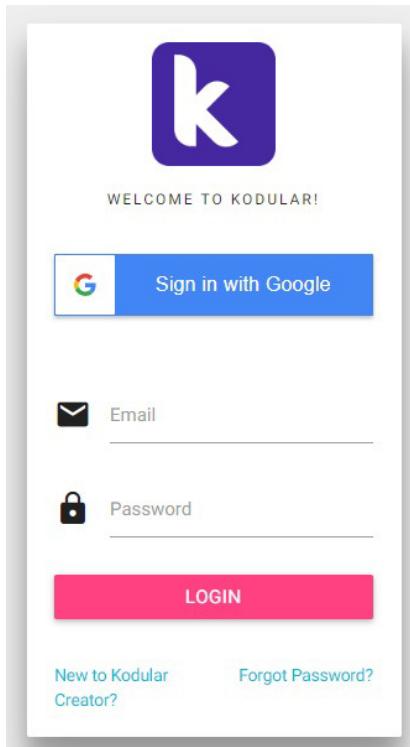


Figura 3.2: Página Ide login

Após o login, será exibida a imagem de boas-vindas mostrada na imagem a seguir e você estará no ambiente de desenvolvimento pronto para criar seus aplicativos Android.



Figura 3.3: Imagem de boas-vindas

Precisamos criar um novo projeto, que será o app de avaliação de corridas, então clique na opção `Create Project`. Será exibida a imagem a seguir, solicitando o nome do projeto. Informe o nome **Avalia\_run** e clique em `NEXT` para continuar.

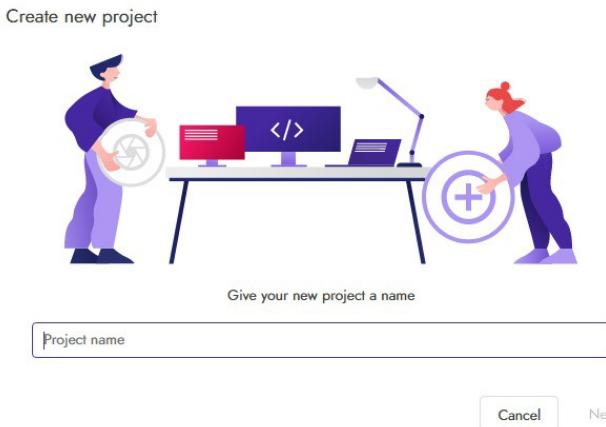


Figura 3.4: Nomeando o projeto

Uma nova janela será exibida para a realização de algumas configurações de seu projeto, neste momento simplesmente clique no botão `Finish` para prosseguir.

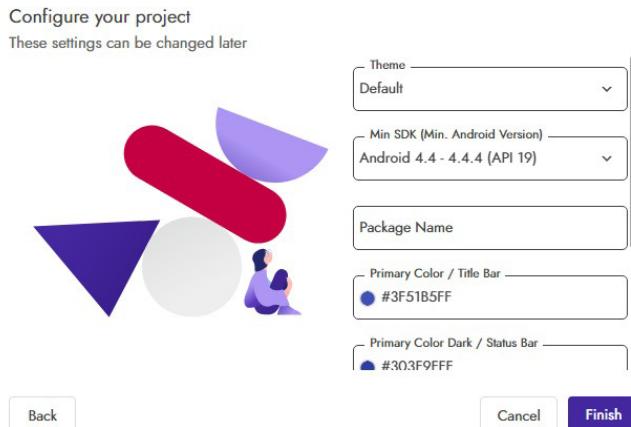


Figura 3.5: Nomeando o projeto

A tela exibida pelo Kodular lhe dará as boas-vindas, conforme demonstra a imagem a seguir. Clique na opção `fechar`, na janela central para continuar.

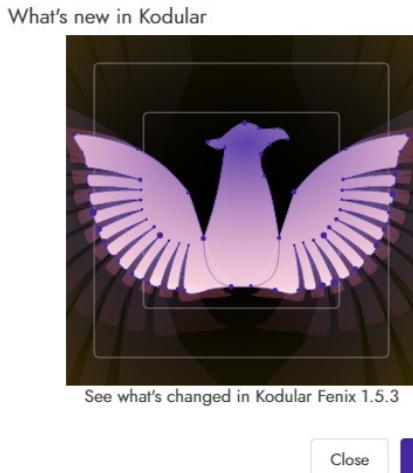


Figura 3.6: Janela de boas-vindas

A tela exibida na imagem a seguir lhe indica que tudo está pronto para começarmos o desenvolvimento do leiaute de nosso app. Vale lembrar que todos os projetos que você desenvolver ficarão salvos nas nuvens do ambiente do Kodular, isto é, de qualquer computador você pode acessar o ambiente com o seu login, e encontrar todos os seus projetos disponíveis para continuar os trabalhos.

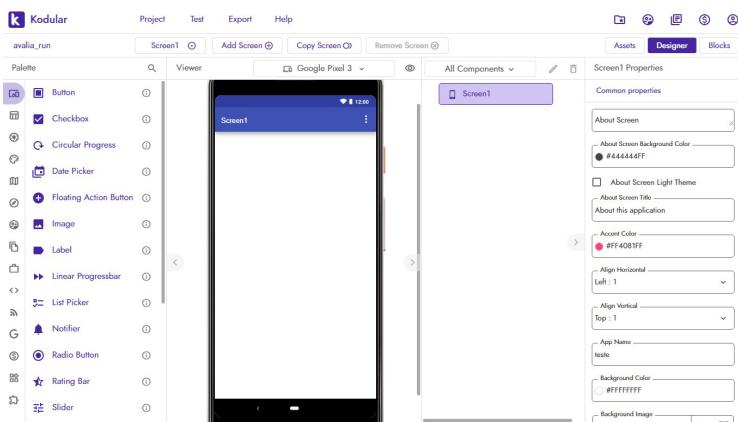


Figura 3.7: Tela para desenvolvimento do app

### 3.3 DESENVOLVENDO A PRIMEIRA TELA DO PROJETO

Observe, na imagem anterior, que temos vários elementos disponíveis visualmente na tela para o desenvolvimento do app, mas não se preocupe, pois vamos falar sobre eles no momento oportuno.

Começaremos defindindo algumas configurações do projeto. Para acessar essas configurações clique no ícone da engrenagem que se encontra no canto superior direito de sua tela.



Figura 3.8: Acessando as configurações do projeto

Após o acesso, a seguinte tela será exibida.

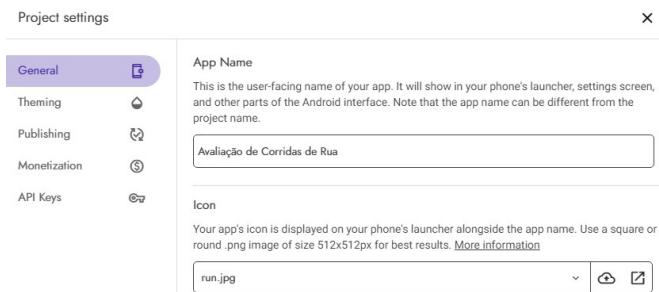


Figura 3.9: Exibindo a tela de configurações do projeto

A primeira mudança na configuração da tela que faremos será referente ao nome que será exibido quando o app for instalado em seu dispositivo Android. Para isso, identifique a propriedade **App Name**, e na linha abaixo da propriedade digite **Avaliação de Corridas de Rua**.

Outra alteração que devemos realizar é referente ao ícone que será exibido no app quando este for instalado. Devemos realizar um upload da imagem que será utilizada. Para isso, localize e clique na opção **Upload file** abaixo do nome **ICON** nessa mesma tela que está sendo mostrada.

#### Icon

Your app's icon is displayed on your phone's launcher alongside the app name. Use a square or round .png image of size 512x512px for best results. [More information](#)



Figura 3.10: Janela para upload de imagens

Nesse momento será exibida uma pequena janela conforme mostra a figura a seguir. Ao clicar no botão de `Upload asset`, surgirá uma nova janela pedindo para selecionar um arquivo para envio e posterior utilização.

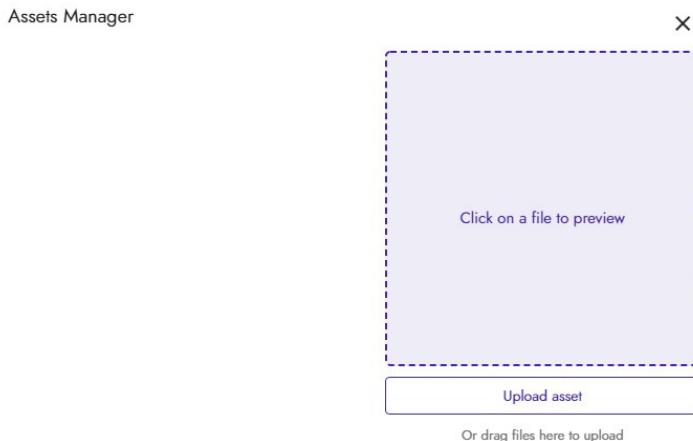


Figura 3.11: Selecionando imagens

Selecione a imagem `run.jpg` (que está disponível para download através do link [http://www.nelfabbri.com/avalia\\_run](http://www.nelfabbri.com/avalia_run)) e clique no botão `OK`, para confirmar. Após essa tarefa, o arquivo já está disponível para utilização no seu app. Selecione-o na lista, e clique em `OK` para confirmar a configuração do ícone. Observe que nada vai ocorrer ainda em sua tela, pois este efeito apenas

ocorrerá na instalação do app em seu dispositivo.

Ainda na tela de configurações do projeto, selecione no menu lateral que se encontra à sua direita a opção Theming .

Vamos modificar a cor da **barra de status** (a barra superior) e, para isso, devemos alterar a propriedade Primary Color Dark para a cor Light Green . A imagem a seguir mostra o resultado da modificação que acabamos de realizar após o clique em fechar a janela de configurações do projeto.



Figura 3.12: Alterando a cor da barra de status

Vamos ajeitar algumas configurações da tela principal: a **Screen1**. Essas configurações são chamadas de propriedades, ou *properties*. Para acessar a área das *properties* da Screen1, basta dar um clique com o mouse sobre o nome do objeto no qual deseja realizar as modificações. Veja na imagem a seguir o local para se clicar e acessar as propriedades da Screen1:

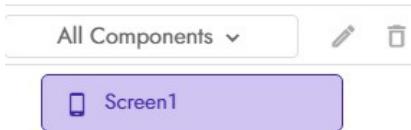


Figura 3.13: Acessando as propriedades

Ao clicar, será exibida a janela de propriedades da Screen1,

conforme demonstra a imagem a seguir:

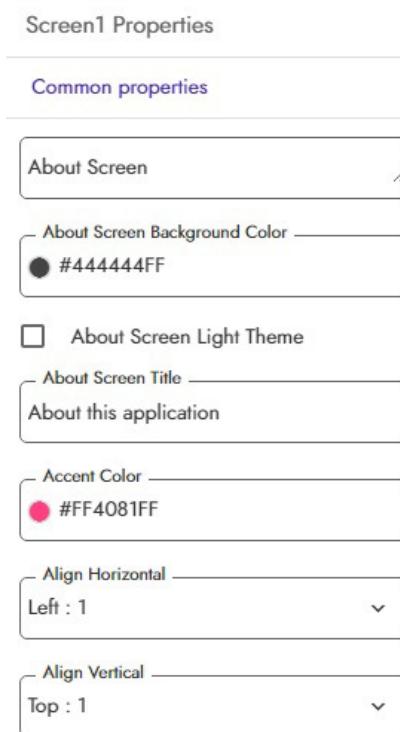


Figura 3.14: Janela de propriedades da Screen1

Vamos alterar o alinhamento de tudo que ficará visível na sua tela para centralização horizontal, pois nas configurações originais tudo ficaria alinhado à esquerda. Então, na propriedade Align Horizontal , selecione a opção Center: 3 .

Vamos também alterar a cor da barra de navegação do app. Clique na opção Default da propriedade Navigation Bar Color para exibir as opções de cores e então selecione a cor Light Green . Veja na imagem a seguir, como ficou a tela com

esta propriedade modificada.

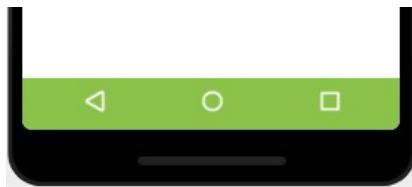


Figura 3.15: Alterando a cor da barra de navegação

Como estamos vendo na imagem anterior, existe uma barra de títulos com o nome **Screen1** em exibição. Vamos ocultar essa barra, pois não fará parte do design planejado. Para ocultá-la, localize a propriedade **Title Visible** e desligue sua visualização arrastando o botão para a esquerda.

Depois desses ajustes na tela principal, chegou o momento de inserirmos os componentes que vão compor a parte que dará interatividade do usuário com o app. Esses componentes também são chamados de **Objetos**, e são encontrados na guia **Palette** , que o leitor encontrará na parte esquerda da tela. A figura a seguir exibe a guia **Palette** .

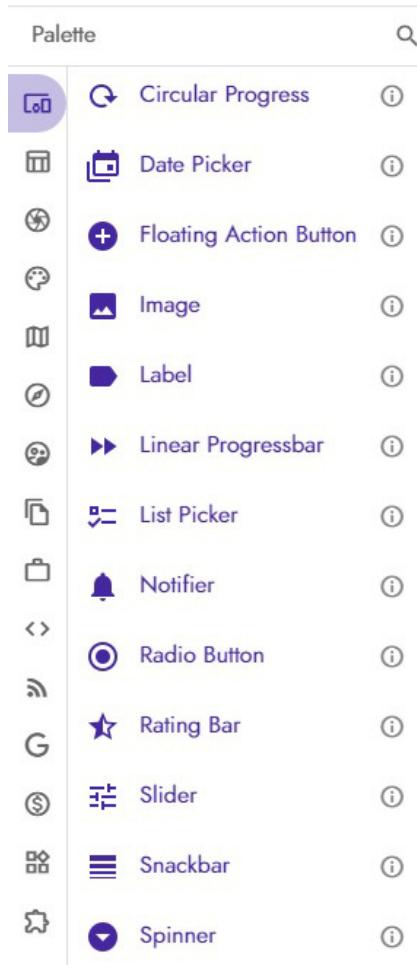


Figura 3.16: Objetos disponíveis da guia Palette

Observe que existem vários objetos disponíveis para utilização. Todos os componentes estão reunidos em categorias ou guias definidas por utilidades. A guia default é a `User Interface`.

Na tela principal do nosso app, vamos criar um efeito de

carrossel, ou seja, duas imagens ficarão sendo exibidas alternadamente de acordo com um tempo que vamos definir. O objeto responsável por esse efeito é o `View Flipper`. Localize-o na `Palette`, clique com o mouse e arraste-o até a área de `Viewer`. Após sua inserção, sua tela deverá estar conforme a imagem exibida a seguir.

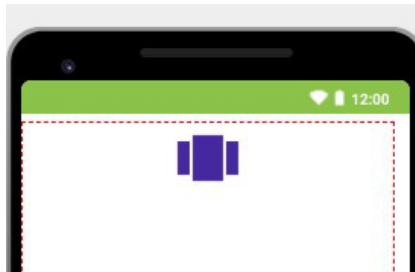


Figura 3.17: Objeto `View Flipper` inserido na tela do app

Precisamos realizar alguns ajustes em suas propriedades. Selecione a guia `Properties` e na sequência clique sobre o objeto `View_Flipper1`, para exibir todas suas propriedades.

Primeiramente, vamos definir o tempo da transição entre as imagens, que será de 2 segundos. Esse tempo é medido em milissegundos, então 2 segundos representam 2000 milissegundos. Para essa alteração, localize a propriedade `Flip Internal` e defina-a para 2000.

Suas próximas configurações referem-se ao seu tamanho de exibição na tela. Vamos alterar sua altura de exibição indicada pela propriedade de nome `Height` e a sua largura indicada pela propriedade `Width`. Localize a propriedade da altura `Height` e altere seu valor para `150 Pixels`, conforme demonstrado na imagem a seguir e logo após a digitação pressione a tecla `Enter`,

para o valor ser aceito e confirmado.



Figura 3.18: Alterando a altura do objeto

Vamos deixar a largura de exibição do objeto em tela ocupando todo espaço na horizontal, para isso, selecione na propriedade `width` a opção `Fill parent` clicando no ícone mais à direita dessa propriedade. A exibição da tela deverá estar igual à figura demonstrada a seguir.

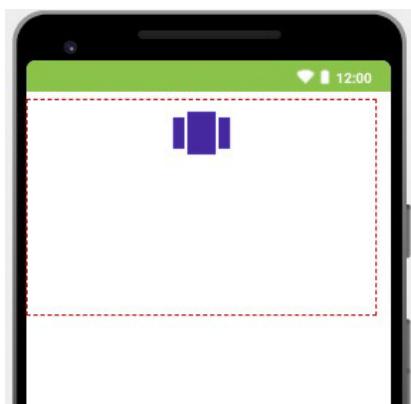


Figura 3.19: Alterando a largura do objeto View Flipper

Precisamos ainda indicar quais as figuras que serão exibidas no objeto `View_Flipper1`. Vamos realizar um upload das imagens que utilizaremos. Para isso, clique no botão `ASSETS` (indicado pelo ícone de figuras) que se encontra ao lado esquerdo do botão da configuração do projeto para que seja exibida a caixa de diálogo **Assets Manager**. Nela estão visíveis todos os arquivos que você

subiu, além do botão `UPLOAD ASSET`, que é o responsável pelo envio de novas imagens. Clique nele e selecione as imagens desejadas. Aqui nós utilizamos a `imagem1.png` e a `imagem2.png`. Realize o upload individualmente. A imagem a seguir exibe a janela `Assets Manager`.

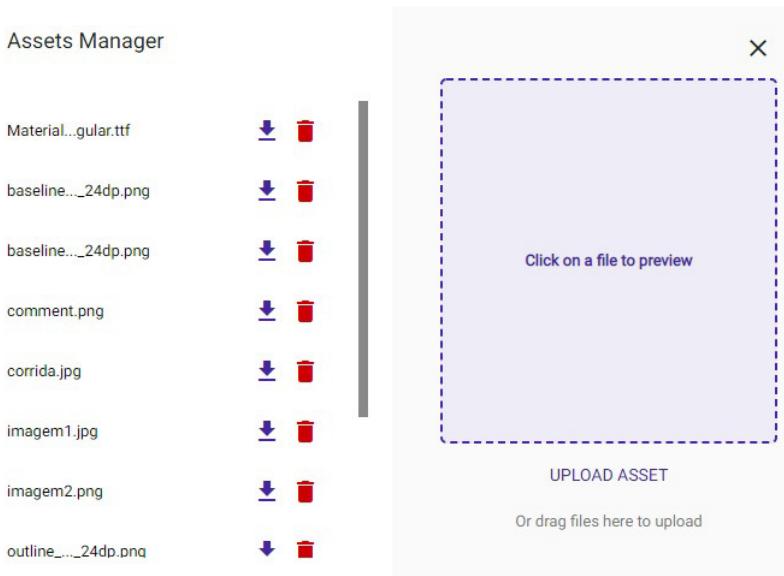


Figura 3.20: janela Assets Manager

Após o upload das imagens, clique no botão de fechar para prosseguirmos com as configurações.

Agora precisamos indicar no objeto `View_Flipper1` o nome das imagens que serão utilizadas para exibição. Para isso, na propriedade `Add Images From String`, digite os nomes das figuras cujo upload realizamos: `imagem1.png, imagem2.png`. Utilize uma vírgula (,) para separar as imagens. Vale lembrar que essas configurações apenas serão exibidas quando o app estiver

instalado.

O próximo passo será a inserção de um espaçamento que ficará entre o `View_Flipper1` e o próximo objeto que vamos inserir. A plataforma Kodular possui um objeto que realiza este espaçamento entre objetos, trata-se do objeto **Space**, que pode ser encontrado na

Palette na guia `Layout`. Para acessar a guia `Layout`, posicione o mouse totalmente ao lado esquerdo da Palette sobre os pequenos ícones que estão em exibição para ver as demais guias.

Clique sobre o ícone da guia `Layout` para a exibição dos objetos referente a esta guia. A imagem a seguir exibe a guia `Layout` e seus objetos disponíveis para utilização.

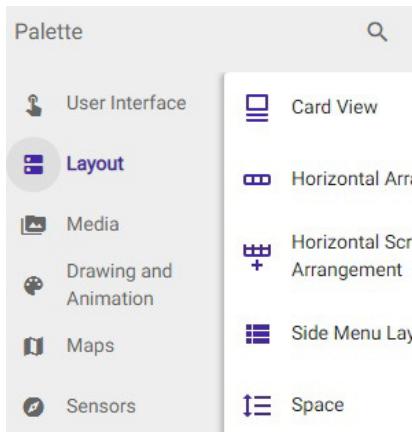


Figura 3.21: Acessando a guia de Layout

Localize o objeto `Space`, clique e arraste-o até o centro da sua tela, logo abaixo do objeto `View_Flipper1`.

A única configuração que deveremos realizar em suas

propriedades é o tamanho que ela ocupará na tela para separar os objetos. Vamos definir sua altura para 20 pixels. Acesse a propriedade `Height` do objeto `Space1` e digite `20` na opção `pixels` e depois clique no botão `OK` para a aceitação da propriedade.

## Inserindo Label para exibição de mensagem

Dando sequência ao desenvolvimento da tela, vamos incluir um objeto para exibir a mensagem que indicará a lista de corridas que estão com as melhores avaliações. Para isso devemos acrescentar um objeto que exibe mensagens em tela: `Label`. Na Palette de objetos, altere da guia `Layout` para a guia `User Interface` e nela localize o objeto `Label`. Clique sobre ele e arraste-o para a área de `Viewer`, imediatamente abaixo do componente `Space`.

Para dar uma maior visibilidade à mensagem que será exibida, vamos aumentar o tamanho da letra para 20 pontos. Acesse a guia de propriedades da `Label1` e localize a que realiza esta tarefa: `Font Size`. Digite o número `20`. Alteraremos também o nome que está em exibição na tela, pois não queremos que fique escrita a palavra "Label1" para o usuário e, sim, **Destaques**. Para tanto, na propriedade `Text`, digite a palavra "Destaques" sem as aspas e pressione a tecla `ENTER`. Vamos também alterar a cor de exibição da palavra "Destaques". Na propriedade da `Label1`, selecione em `Text Color` a opção de cor `Dark Gray`. Veja na imagem a seguir, como estará o designer desenvolvido até agora.

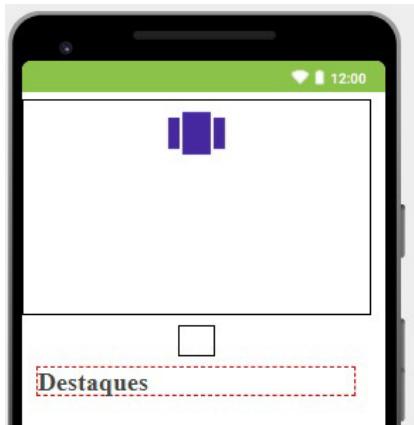


Figura 3.22: Exibindo a mensagem Destques

Para listar as corridas que estão em destaque, necessitamos acrescentar um componente, que, além de mostrar o nome da corrida, terá uma breve descrição e uma imagem. O componente `List View Image and Text` responsável por listar essas informações, encontra-se na guia `User Interface` da Palette . Insira-o abaixo da label na sua área `Viewer` .

Algumas configurações neste componente também se tornam necessárias para se ajustar ao leiaute do projeto. Altere o tamanho do comprimento lateral para 90% da tela. Para isso, na propriedade `Width` digite o valor `90` na opção `porcent` . Existirá uma linha que servirá para separar as corridas em destaque, vamos alterar a sua cor para verde, então acesse a propriedade `Divider Color` e selecione a cor `Green` . Após essas configurações, veja na imagem a seguir como estará a sua tela até o momento.

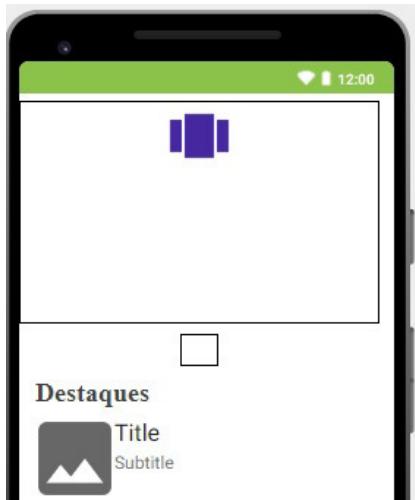


Figura 3.23: Exibindo a List View Image and Text

## Adicionando um botão flutuante

Acrescentaremos ainda um botão flutuante que terá a função de, quando clicado, abrir uma tela com todas as corridas disponíveis para avaliação. Esse objeto `Float Action Button` também se encontra na guia `User Interface` da Palette de objetos. Localize-o e insira um em sua tela. Observe que, ao inserir, você não visualizará este componente, pois esse é um **componente não visível**, isto é, ele apenas será exibido quando o seu aplicativo estiver instalado no dispositivo Android.

Porém, ao inseri-lo, ele ficará disponível na sessão de todos os componentes na área de design. Veja na imagem a seguir a área de todos os componentes com os objetos não visíveis.

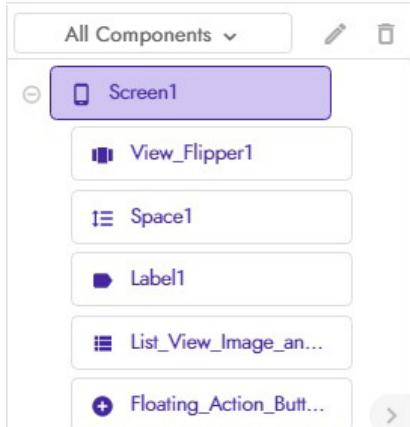


Figura 3.24: Área dos objetos não visíveis

O botão flutuante também necessitará de algumas configurações em suas propriedades. Primeiramente, vamos alterar a sua cor de fundo para o padrão de nosso app. Selecione na propriedade `Background Color`, a cor `Light Green`. O botão flutuante exibirá em seu interior um caractere do sinal maior que (`>`) indicando que abrirá mais corridas. Para incluir esse símbolo, acesse a propriedade `Icon Name (Material)` e digite o nome do símbolo: `keyboard_arrow_right`.

Precisamos para finalizar esta tela inicial de mais dois componentes não visíveis. Ainda na guia `User Interface`, acrescente a sua tela de desenvolvimento um objeto chamado `Notifier`. Agora vá até a `Palette` na guia `Connectivity` e inclua um objeto chamado `Web`. A imagem a seguir exibe todos os objetos não visíveis que temos na tela inicial de nosso app.

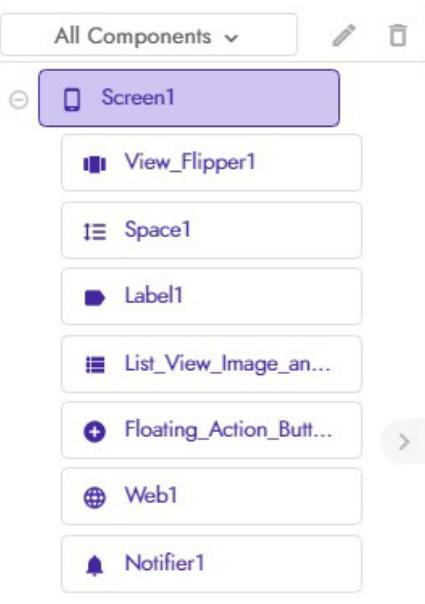


Figura 3.25: Exibindo os objetos não visíveis

Por ora, não vamos dizer mais detalhes sobre estes objetos (`Notifier` e `Web`), pois serão explicados mais adiante durante a programação das funcionalidades da tela.

Com isso, terminamos a tela inicial de nosso projeto. A programação, como dissemos anteriormente, ficará para capítulos posteriores. Caso você deseje, poderá baixar o leiaute desenvolvido neste capítulo através do link [http://www.nelfabbri.com/avalia\\_run](http://www.nelfabbri.com/avalia_run).

No próximo capítulo aprenderemos a instalar o aplicativo no nosso dispositivo para realizar os primeiros testes de visualização e importar o projeto que foi desenvolvido e baixado do site deste livro.

## CAPÍTULO 4

# INSTALANDO O APP

Sempre que quisermos testar nosso aplicativo, necessitaremos instalá-lo em um dispositivo com o sistema Android, então precisaremos de alguns pré-requisitos.

- Possuir acesso à internet;
- Possuir um leitor de QR code instalado em seu dispositivo.

Caso não possua um aplicativo em seu dispositivo que faça a leitura de um QR Code, recomendamos acessar a loja de aplicativos Google Play e realizar a procura e instalação do aplicativo gratuito **Lightning QR**.

Com o Lightning QR instalado, teremos que solicitar a geração de um código no Kodular com o endereço para realizar o download e a instalação de nosso app Avaliação de Corridas de Rua. Na plataforma de desenvolvimento Kodular, acesse no menu Export a opção Android App (.apk) e aguarde a geração e exibição do QR code em sua tela. A imagem a seguir, demonstra a opção de geração do QR Code.

-  Android App (.apk)
-  Android App Bundle (.aab)

Figura 4.1: Gerando o QR Code

Após a exibição do QR Code na tela de seu computador, abra o leitor Lightning QR no dispositivo móvel e aponte a sua câmera para o QR code exibido em sua tela para a leitura do endereço. Será exibida uma imagem semelhante à demonstrada a seguir, com um link em destaque.

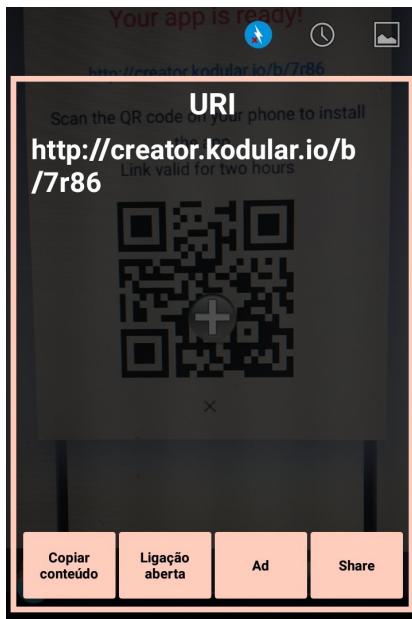


Figura 4.2: Exibição do link de instalação do app

Para realizar o download, pode-se apenas clicar no botão

Ligaçāo aberta e aguardar a realização do download, ou clicar no botāo Copiar conteúdo e abrir o seu aplicativo de navegação e colar o endereço para download.

Após o download, será exibida em seu dispositivo a mensagem conforme imagem a seguir.

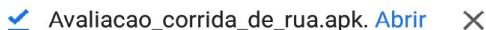


Figura 4.3: Exibição do fim do download

Clique no link **Abrir** para realizar a instalação e, caso seja solicitado, dê a permissão de instalação a aplicativos de fonte desconhecida.

Após a instalação basta procurar pelo seu app instalado e abri-lo para visualização. Observe que apenas temos o leiaute desenvolvido e nenhuma função estará disponível para utilização até o momento.

## 4.1 IMPORTANDO UM PROJETO

Caso o leitor deseje realizar a importação do projeto desenvolvido pelos autores, abra o link disponibilizado no início do livro e siga as seguintes orientações:

Um projeto para edição no Kodular possui a extensão de arquivo .aia .

O acrônimo aia significa **App Inventor App**, ou seja, o arquivo que contém o código de edição do projeto.

Clique no menu FILE e selecione a opção Import Project (.aia) from my computer , conforme demonstra a imagem a seguir.

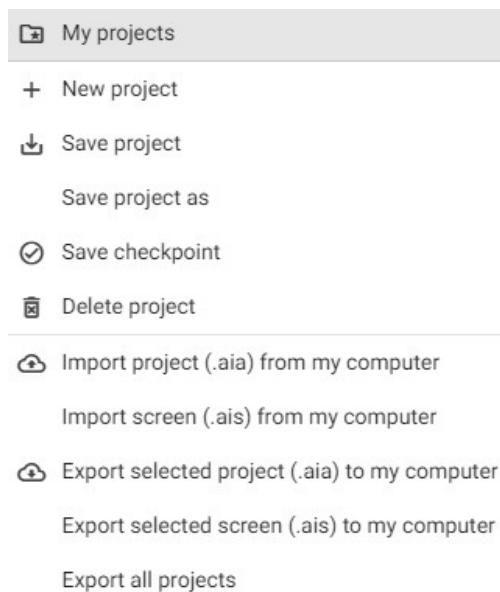


Figura 4.4: Tela para importar um projeto

Agora é só clicar no botão Escolher arquivo e selecionar o arquivo com extensão .aia para realizar a importação de um

projeto. Clique no botão `OK` para confirmar. A figura a seguir exibe a tela de importação de um projeto.

Import project



Figura 4.5: Selecionando o arquivo para importação

Feitos esses procedimentos, o projeto será exibido no seu ambiente de desenvolvimento.

No próximo capítulo, vamos dar continuidade à criação dos leiautes, desenvolvendo a tela que exibe todas as corridas cadastradas e disponíveis para avaliação.

## CAPÍTULO 5

# DESENVOLVENDO A TELA PRINCIPAL

Aqui vamos desenvolver o leiaute da segunda tela, onde serão listadas todas as corridas cadastradas pelos usuários e disponíveis para avaliações. Veja na imagem a seguir como ficará.



## São Silvestre

Av. Paulista, 900

Uma corrida de rua realizada anualmente na  
cidade



Figura 5.1: Essa é a tela principal que desenvolveremos

## 5.1 ACRESCENTANDO UMA NOVA SCREEN

Como essa é uma nova tela, será necessário inserir uma segunda Screen em nosso projeto. Para isso, note a existência de um botão no canto superior esquerdo da plataforma com a função que desejamos, ADD SCREEN . Clique nele, e será exibida a janela que solicitará o nome da nova Screen. Veja a janela de New Screen que surgirá na imagem a seguir.

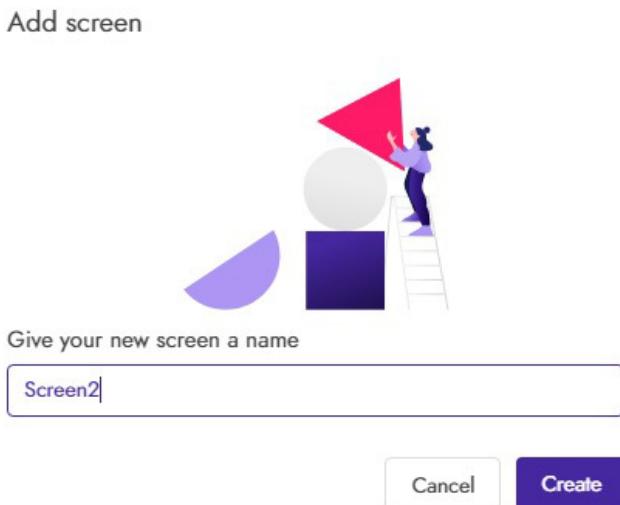


Figura 5.2: Criando uma nova Screen

Escreva o nome “Principal” (sem as aspas) no lugar onde se lê “Screen2” e clique no botão `OK`.

Uma nova Screen vazia surgirá em sua tela. Ela é parte integrante do mesmo projeto que estamos desenvolvendo.

Teremos que realizar algumas configurações nas propriedades dessa nova Screen. Então acesse a área das propriedades da tela **Principal** e façamos as alterações para que a exibição de todo o conteúdo fique centralizada na Screen. Para isso, na propriedade `Align Horizontal`, selecione a opção `Center`. Para que o título default (padrão) da Screen não apareça, altere a propriedade `Title Visible` apenas deslizando o marcador para a esquerda para desligá-lo.

## 5.2 UMA BARRA DE TÍTULOS PERSONALIZADA

Vamos desenvolver uma barra de títulos para esta nova Screen, mas como cada linha da tela permite exibir apenas um único objeto, necessitamos de algum recurso que possibilite um alinhamento de vários objetos na mesma linha.

Vá até o Palette de objetos e selecione a guia `Layout`. Nela, você encontrará o componente `Horizontal Arrangement`, que permitirá esse arranjo na tela. Selecione e inclua em sua Screen Principal um objeto `Horizontal Arrangement`. Necessitamos realizar algumas configurações em suas propriedades, para melhor aproveitamento de seu espaço interior. Deixaremos sua altura configurada para ter exatamente 50 pixels. Então acesse a propriedade `Height` e digite o valor 50 pixels. O comprimento horizontal deverá ocupar todo o espaço da tela. Para isso, acesse a propriedade `Width` e altere seu valor para `Fill Parent`.

Também alteraremos o alinhamento vertical para deixar todo o conteúdo alinhado na posição vertical. Então, acesse a propriedade `Align Vertical` e selecione a opção `Center`. Para deixarmos uma cor de fundo padronizada no objeto `Horizontal Arrangement`, altere a propriedade `Background Color` para `Light Green`.

Dentro desse `Horizontal Arrangement`, vamos incluir um objeto que realizará a função de dar um espaçamento entre a borda esquerda da tela e o próximo objeto. Da guia `Layout`, inclua um objeto `Space` e configure sua propriedade `Width` para 10 pixels.

Nessa barra de títulos, teremos um botão para retornar a tela inicial. Para criá-lo, acesse a guia da `User Interface` e inclua ao lado do `Space` um objeto **Button**. Vamos alterar suas propriedades relativas ao seu tamanho de exibição: `Height` e `Width` terão 35 pixels. Apagaremos o texto exibido na propriedade `Text`, pois exibiremos uma imagem em seu interior. Na propriedade `image`, realize o upload do arquivo `seta_voltar.png`, conforme demonstrado em capítulo anterior.

Ao lado direito do botão que acabamos de configurar, devemos inserir uma `Label` para exibir o título da página que estamos construindo. Nela exibiremos a palavra `CORRIDAS`. Após inserir a `Label`, vamos realizar as suas configurações. Deixaremos seu conteúdo formatado como negrito. Para esse efeito, acesse a propriedade `Font Bold` e deslize para o lado direito, para ativá-lo.

Alteraremos também o tamanho do texto a ser exibido, selecionando a propriedade `Font Size` e digitando o valor `20`.

Em relação ao tamanho horizontal que a `Label` ocupará na tela, vamos deixá-la ocupar todo o espaço disponível, bastando, para isso, selecionar a propriedade `Width` e selecionar a opção `Fill Parent`.

Agora, vamos alterar o texto da exibição. Selecione a propriedade `Text`, apague o conteúdo presente e digite a palavra `CORRIDAS`. Não podemos nos esquecer de centralizar o texto dentro da `Label`, bastando selecionar a propriedade `Text Alignment` e marcar a opção `Center`.

Na barra de títulos da Screen Principal, haverá um segundo

botão que realizará um filtro das corridas. Vamos adicionar mais um objeto `Button` ao lado da `Label` com o texto de `CORRIDAS` e realizar algumas modificações em suas propriedades, são elas:

Quanto ao tamanho do `Button`, altere as propriedades `Height` e `Width` para 35 pixels cada uma. Devemos também retirar o texto da propriedade `Text`, pois será exibida uma imagem em seu interior. Por falar em imagem, realize o upload do arquivo `filtro.png` através da propriedade `Image`, conforme demonstrado em capítulo anterior.

Da mesma maneira que demos o espaçamento entre a margem da tela e o primeiro botão, utilizando o objeto `Space`, vamos inserir outro `Space` para ficar ao lado direito do botão da filtragem. Após incluí-lo, altere sua propriedade `width` para 10 pixels. A imagem a seguir mostra o resultado das configurações que acabamos de realizar com a barra de títulos que criamos.



Figura 5.3: Barra de títulos personalizada

## 5.3 ÁREA DE EXIBIÇÃO DAS CORRIDAS

Após a criação da barra de títulos, devemos preparar a parte onde serão exibidas todas as corridas. Como teremos muitas corridas e o espaço de visualização do dispositivo pode ficar pequeno, devemos incluir um objeto que permita rolar a tela para

cima ou para baixo, possibilitando a total exibição dos dados. Vamos então até a guia `Layout` da `Palette`, para localizar e inserir o objeto `Vertical Scroll Arrangement`, responsável pelo efeito esperado. Inclua-o abaixo da barra de títulos.

Vamos realizar algumas configurações em suas propriedades em relação ao tamanho de sua exibição em tela: para `Height`, selecione `Fill Parent` e, em `Width`, altere para `90 percent`.

Precisamos inserir alguns objetos não visíveis que utilizaremos futuramente durante a criação da programação. Acesse a `Palette` dos objetos e na guia `Connectivity` insira um objeto `Web` na sua `Screen`. Note que, em qualquer lugar que o objeto for inserido, ele será posicionado automaticamente na área de todos os componentes.

Para o compartilhamento das informações, vamos inserir o objeto `Sharing`, que pode ser encontrado na guia `Social` na `Palette` dos objetos. Da mesma maneira que fizemos com o objeto `Web`, insira o `Sharing` dentro da sua `Screen`, e note que ele também será posicionado na mesma área dos objetos.

Por fim, acrescente um objeto `Notifier` da guia `User Interface` em sua `Screen`, o qual terá o mesmo comportamento dos demais objetos não visíveis.

Os objetos não visíveis que acabamos de inserir serão utilizados futuramente no momento da programação das funcionalidades do aplicativo. Por ora, não precisamos dizer mais detalhes sobre estes objetos, mas não se preocupe, em breve falaremos mais sobre eles. O importante para este momento é que eles estejam disponíveis em sua `Screen` e logo você estará

dominando tudo isso.

## 5.4 IMPORTANDO UMA EXTENSÃO

Apesar de o Kodular possuir muitas ferramentas para formatação e exibições de dados, o formato em que desejamos apresentar a lista de corridas cadastradas deverá possuir:

- uma `Image` com o logotipo da corrida;
- uma `Label` para a legenda com o nome;
- outra `Label` que possuirá o endereço da largada;
- outra `Label` que exibirá uma breve descrição da corrida;
- um `Button` para compartilhar informações da corrida.

Porém, para cada uma das corridas cadastradas no banco de dados do app seria necessário inserir esse conjunto de objetos listados. Assim sendo, ficaria praticamente impossível de inserirmos vários componentes para realizar esta tarefa, pois não sabemos quantas corridas estão cadastradas e quantas ainda poderão ser inseridas pelos usuários.

Para resolver este problema, existe uma extensão que realiza automaticamente todo esse trabalho de configuração e exibição, independente da quantidade de corridas que temos cadastradas.

Quando novas corridas forem inseridas no banco de dados, a extensão se encarregará de exibi-las sem a necessidades de realizar a atualizações ou programações adicionais no app.

Extensões são componentes desenvolvidos por terceiros a fim de executar uma determinada função que não está disponível na plataforma.

Utilizaremos a extensão **life.inventor.CardView.aix** também disponível para download no site do projeto, através do link [http://www.nelfabbri.com/avalia\\_run](http://www.nelfabbri.com/avalia_run).

Essa extensão **CardView** possibilita uma outra opção de listagem de conteúdos, utilizando um formato de cartão. No cartão, serão exibidos todos os objetos descritos anteriormente.

Para realizarmos a importação de uma extensão, selecione na Palette a guia **Extension**. Será exibida uma pequena janela, conforme imagem demonstrada a seguir.

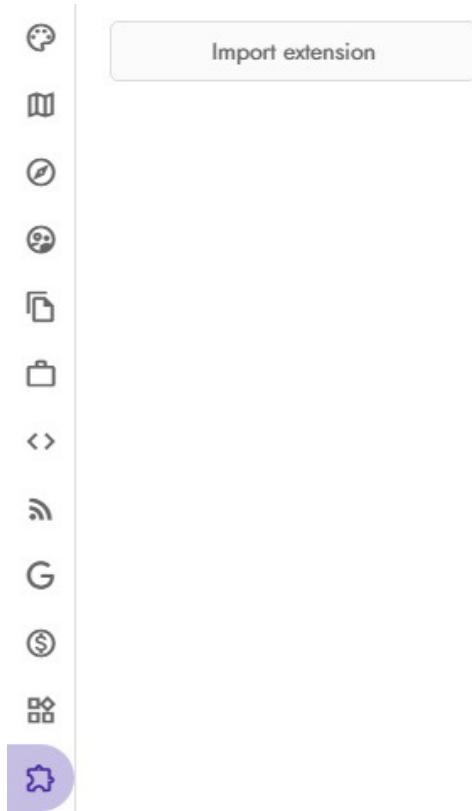


Figura 5.4: Adicionando extensão

Selecione a opção `Import extension`, para começar com a importação. Após o clique será exibida a próxima imagem.

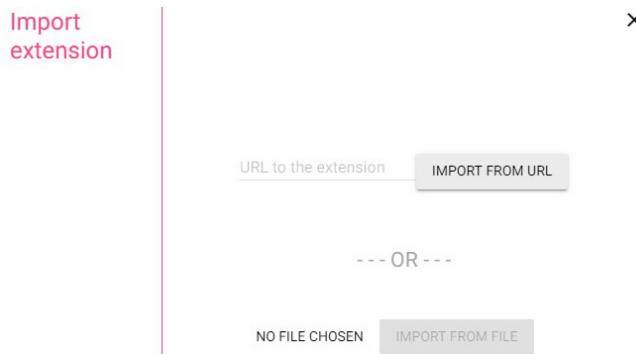


Figura 5.5: Selecionado uma extensão

Clique no botão `From my computer`, e selecione o arquivo `life.inventor.CardView.aix`. Confirme a importação clicando no botão `IMPORT`. Quando a extensão for enviada para o ambiente da plataforma Kodular, a guia `Extension` indicará a existência da sua extensão, conforme a imagem a seguir.

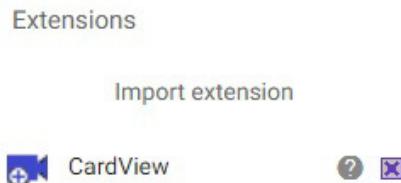


Figura 5.6: Término da importação da extensão

Após a importação da extensão, selecione e arraste-a para incluir um objeto `CardView` em sua Screen Principal. Ela ficará disponível nos componentes não visíveis. A imagem a seguir mostra todos os objetos não visíveis que incluímos na Screen Principal.

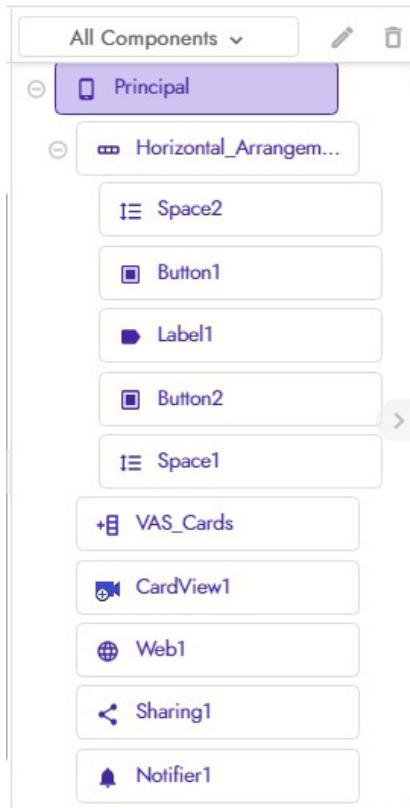


Figura 5.7: Área dos objetos não visíveis

Para finalizar, necessitamos realizar uma única, mas não menos importante, configuração na propriedade do objeto `CardView1`. É a propriedade que define uma fonte de ícones que serão utilizados no objeto durante a execução. Acesse as propriedades da `CardView1`, localize a propriedade `FontIcon` e realize o upload da fonte `MaterialIcons-Regular.ttf` que está disponível para download no link dos arquivos do livro.

Material é um sistema adaptável de diretrizes, componentes e ferramentas que suportam as melhores práticas de design de interface do usuário. Com o apoio do código-fonte aberto, o Material dinamiza a colaboração entre designers e desenvolvedores e ajuda as equipes a criar produtos bonitos rapidamente. Conheça mais no site oficial <http://material.io>.

No próximo capítulo, vamos criar e configurar o leiaute de login e cadastro de usuários, pois somente com o login efetuado, o usuário poderá realizar uma avaliação.

## CAPÍTULO 6

# DESIGN DA TELA DE LOGIN E CADASTRO

Neste capítulo, vamos construir o leiaute da tela de login e de cadastro de novo usuário, pois somente cadastrado e logado é que se poderá comentar sobre a corrida. As imagens a seguir demonstram as duas partes da terceira Screen que vamos criar. Poderíamos fazer duas Screens, uma para o login e outra para o cadastro de usuários, porém cada Screen é tratada pelo Android como sendo um novo app, consumindo mais recursos de memória e processamento de seu aparelho, podendo até fazer com que o seu app pare de funcionar abruptamente. Se é possível diminuir a quantidade de Screens utilizadas para uma melhor utilização e performance de nosso app, por que não o fazer?

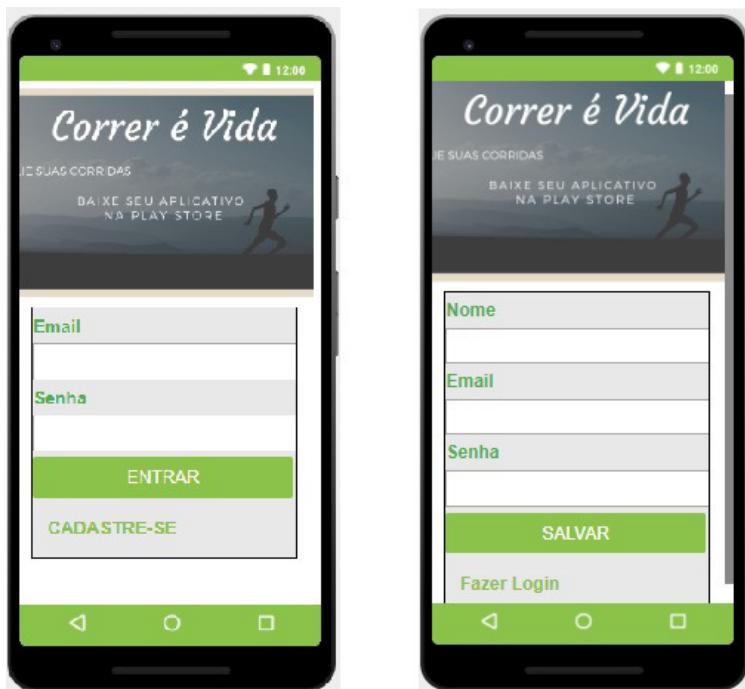


Figura 6.1: Tela de Login e tela de cadastro

Alternar entre os leiautes, de acordo com a necessidade do usuário, será o recurso que demonstraremos nessa Screen.

Vamos criar uma nova Screen no seu projeto do Kodular. Conforme já visto em capítulo anterior, adicione uma Screen e dê o nome de **LoginCadastro**.

Igualmente como fizemos na Screen1, necessitamos realizar as mesmas configurações iniciais referente ao design da Screen de **LoginCadastro**. Realize as configurações nas propriedades, conforme demonstra a tabela a seguir:

Propriedade	Alterar valor
Align Horizontal	Center
Align Vertical	Center
Navigation Bar Color	Light Green
Title Visible	Rolar para a esquerda

Dentro desta tela, deixaremos uma imagem sempre visível, como parte do design do app. Selecione na Palette a guia User Interface , e nela localize o objeto Image , e arraste-o para dentro da tela de LoginCadastro .

Realizaremos poucas alterações em suas propriedades. Primeiro vamos inserir uma imagem para exibição no interior do objeto. Para isso, localize a propriedade Picture e selecione a figura Imagem2.png , que está disponível, pois já foi enviada para o projeto anteriormente. Para o tamanho da exibição da figura, altere a propriedade Height para 40 percent .

Após a configuração das propriedades do objeto Image , veja na figura a seguir o resultado esperado.



Figura 6.2: Exibição de imagem

## 6.1 CONFIGURANDO A TELA DE LOGIN

Chegou o momento de criarmos as telas de login e cadastro, mas como dissemos anteriormente, utilizaremos apenas uma Screen para exibi-las, alternando sua visibilidade quando necessário.

Para criar a tela de login, vamos inserir um `Vertical Arrangement` da guia de `Layout` que conterá em seu interior a solicitação do e-mail e a senha do usuário, além de dois botões, sendo o primeiro para logar e outro para cadastrar um novo usuário.

Após inserir a `Vertical Arrangement` imediatamente abaixo do objeto `Image`, vamos alterar a propriedade `width` para `90 percent`.

Vamos agora inserir em seu interior uma `Label` para identificar o que o usuário deverá digitar.

Precisamos realizar algumas configurações nas propriedades da

`Label`. Na propriedade que deixa o texto em negrito `FontBold`, deslize o controle para a direita para ativá-lo. Na propriedade do tamanho da fonte, `FontSize`, vamos utilizar `17`. Já para o texto de exibição na `Label`, altere a propriedade `Text` e digite a palavra `E-mail`. Não podemos nos esquecer de alterar a cor do texto para que fique nos padrões do app, então acesse a propriedade `TextColor` e selecione a cor `Green`.

Com a `Label` já configurada, devemos preparar uma caixa para a digitação do e-mail do usuário. O objeto responsável por essa digitação é a caixa de texto `Textbox`. Ela está disponível na guia `User Interface` na `Palette` e deverá ser inserida logo abaixo da `Label` com o texto `E-mail`.

Como a outra, precisamos realizar algumas configurações de ajustes em suas propriedades. Alteraremos o tamanho da fonte para `17`. Seu comprimento horizontal `Width` deverá ser marcado como `Fill parent`.

Durante a digitação de valores em uma `Textbox`, surgirá uma linha que indicará qual `Textbox` está selecionada. Essa linha também necessita sofrer um ajuste em sua cor para que fique de acordo com as cores de nosso app. Para isso, localize a propriedade `Highlight Color` na `Textbox` e selecione a cor `Green`.

Outra propriedade de uma `Textbox`, é um texto que aparece como sugestão ao usuário durante a sua utilização. No nosso caso, não necessitamos fornecer maiores detalhes e a propriedade `Hint` deverá ter o seu texto apagado.

Para a digitação da senha do usuário, precisaremos inserir mais uma `Label` e uma `Textbox`. Insira primeiramente a `Label`

logo abaixo da `Textbox` que já está na tela. O procedimento adotado para essa `Label` é praticamente o mesmo que já vimos, portanto realize as mudanças nas propriedades, conforme exibe a tabela:

Propriedade	Novo valor
FontBold	Ativar
FontSize	17
Text	Senha
TextColor	Green

Agora, insira uma outra `Textbox` para receber a senha que o usuário vai digitar logo abaixo da `Label` inserida, e realize as configurações das propriedades conforme mostra a tabela a seguir. Observe que deixaremos os mesmos valores adotados para as propriedades da `Textbox` do e-mail, visto anteriormente:

Propriedade	Novo valor
FontSize	17
Width	Fill parent
Highlight Color	Green
Hint	Apagar o texto

Será necessário inserir dois botões na tela. O primeiro para efetuar o login, e o segundo para o usuário ativar a tela de cadastro caso necessário. O objeto `Button`, também pertence à guia `User Interface`. Insira um botão abaixo da `Textbox` da senha, ainda dentro da `Vertical Arrangement` e outro botão logo abaixo. Repare que cada botão recebe um nome de identificação: `Button1` e `Button2`. Para ficar mais fácil a identificação durante

a programação, uma boa prática que adotamos é renomear os objetos. Vamos trocar o nome de `Button1` para `Btn_Login`.

O local para renomear um objeto é na guia de propriedades. Ao entrar nessa área, note que existe um ícone de edição e exclusão (um lápis e uma lixeira). A imagem a seguir mostra essa área.

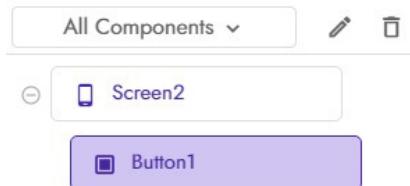


Figura 6.3: Edição e Exclusão de objeto

Clicando no ícone do lápis, será apresentada uma janela para edição, conforme demonstra a próxima imagem. Na opção `New name`, digite `Btn_Login`, e confirme clicando no botão `OK`.

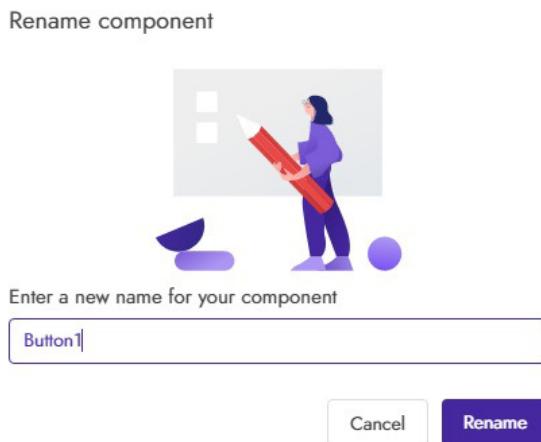


Figura 6.4: Renomeando um componente

Após a troca de nome, veja na imagem a seguir como ficará a identificação do botão.

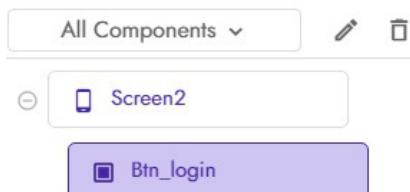


Figura 6.5: Novo nome do objeto

Renomeie o `Button2` da mesma maneira que já vimos, trocando o nome para `Btn_Cadastrar`.

Agora, vamos realizar algumas configurações para o `Btn_Login`. A primeira propriedade a ser alterada é a que exibe a sua cor de fundo. Vá até a `BackGround Color` e selecione a cor `Light Green`.

Vamos também padronizar o tamanho da fonte do texto que será exibido, alterando a propriedade `FontSize` para `17`. A propriedade `Width` que trata do tamanho do objeto na tela, deverá ser marcado como `Fill parent`.

O texto que será exibido no botão pode ser alterado através da propriedade `Text`, digite a palavra `Entrar`. Centralize o texto dentro do botão, para isso localize a propriedade `Text Align` e selecione a opção `Center`.

Para finalizar a `Vertical Arrangement` de login, vamos realizar algumas modificações nas propriedades do botão `Btn_Cadastrar`. De maneira semelhante demonstrado para o `Btn_Login`, configure o botão `Btn_Cadastrar` com as

informações exibidas na tabela a seguir.

Propriedade	Novo valor
BackGround Color	None
FontBold	Ativar
FontSize	17
Width	Fill parent
Text	CADASTRE-SE
Text Color	Light Green

Veja na figura a seguir, como o designer da sua tela deverá estar após todas as configurações realizadas.



Figura 6.6: Tela de login

## 6.2 A TELA DE CADASTRO DE USUÁRIOS

Para desenvolvermos o leiaute do Cadastro de Usuários, necessitamos tornar a `Vertical Arrangement1` **invisível**, para que tenhamos espaço suficiente para o desenvolvimento da tela de cadastro. Acesse a guia de propriedades da `Vertical Arrangement1` e localize a opção `Visible` e altere para falso. Isso fará com que todos os objetos que estão no interior da `Vertical Arrangement1` fiquem invisíveis. Foi essa a ideia que falamos no início deste capítulo: uma tela em que teremos duas funções diferentes. Para isso, trabalharemos com a alternância de visibilidade das `Vertical Arrangements`.

Vamos então inserir uma nova `Vertical Arrangement` da guia de `Layout` logo após o objeto `Image`, para servir como tela de cadastro de usuários.

Configure essa nova `Vertical Arrangement2` com tamanho lateral (`Width`) com 90 porcento.

Para facilitar a identificação dos objetos, altere também o nome das duas `Vertical Arrangements` que inserimos. A primeira referente à tela de Login, renomeie para `VA_Login` e a segunda referente à tela de Cadastro, altere para `VA_Cadastro`.

Vamos inserir alguns objetos dentro da `VA_Cadastro`, que servirão de identificação do que vamos digitar.

Insira uma `Label` que identificará o nome que o usuário deve digitar. Acessando a guia de propriedades, deixe a `Label` com a opção `Font Bold` ativa. Vamos alterar o tamanho da fonte através da opção `Font Size` e digitar o número 17. Na

propriedade `Text` digite a palavra `Nome` e, finalmente, na propriedade `TextColor` altere a cor do texto que será exibido para `Green`.

Logo abaixo da última `Label` inserida, acrescente um objeto `Textbox`, pois será dentro dele que o usuário digitará o nome para cadastro. Novamente, alteraremos a propriedade `Font Size` para `17`. O comprimento horizontal do objeto, deverá ser marcado como `Fill parent` na propriedade `Width`. Conforme visto anteriormente, também vamos alterar a cor da linha interna da `Textbox`, através da propriedade `Highlight Color` para `Green`. Apague o texto que é mostrado na opção `Hint` e renomeie o objeto `Textbox` para `TXT_Nome`.

Vamos inserir mais uma `Label` e uma `Textbox`, que servirão para o recebimento do e-mail do usuário. Após inseri-los, realize as configurações conforme tabela a seguir:

Objeto	Propriedade	Novo valor
Label4	FontBold	Ativar
	FontSize	17
	Text	E-mail
Textbox4	TextColor	Green
	FontSize	17
	Width	Fill parent
	Highlight Color	Green
	Hint	Apagar o texto
	Renomear	TXT_Email

Para finalizarmos o cadastro, vamos incluir mais uma `Label`

e logo abaixo dela, acrescentar uma `Textbox`, pois estes objetos servirão para a digitação da senha do usuário. Realize as configurações nas propriedades de cada objeto adicionado, conforme tabela a seguir:

Objeto	Propriedade	Novo valor
Label5	FontBold	Ativar
	FontSize	17
	Text	Senha
	TextColor	Green
Textbox5	FontSize	17
	Width	Fill parent
	Highlight Color	Green
	Hint	Apagar o texto
	Renomear	TXT_Senha

Quando o usuário terminar a digitação dos dados de cadastro, necessitará pressionar um botão que realize a gravação dos dados, portanto acrescente um botão que realizará essa função.

Insira um `Button` logo abaixo da `Textbox` referente à senha. Vamos alterar algumas de suas propriedades para ajustá-lo ao design planejado. Na propriedade que altera a cor de fundo do botão `BackGround Color` selecione a cor `Light Green`. O tamanho de texto que será exibido também necessita de alteração, então localize a opção `FontSize` e digite o tamanho `17`. O comprimento horizontal do botão, vamos deixar que ocupe a parte total da tela do dispositivo, para isso, altere a propriedade `Width` para `Fill parent`. Alteraremos também a propriedade `Text`

para exibir a função que o botão vai desempenhar, então digite a palavra `SALVAR` e a última propriedade a ser alterada deve ser para alinhar o texto do botão na propriedade `Text Align` para `Center`.

Abaixo do botão que salva os dados do usuário, vamos adicionar ainda outro objeto `Button` que terá a função de retornar para a tela de login. Após sua inserção, deveremos realizar algumas modificações em suas propriedades, veja na tabela a seguir quais modificações necessitamos realizar:

Propriedade	Novo valor
BackGround Color	None
Font Bold	Ativar
Font Size	17
Width	Fill parent
Text	Login
Text Color	Light Green

Após as configurações acima realizadas no leiaute `VA_Cadastro`, a sua Screen deverá ser exibida conforme a imagem a seguir.

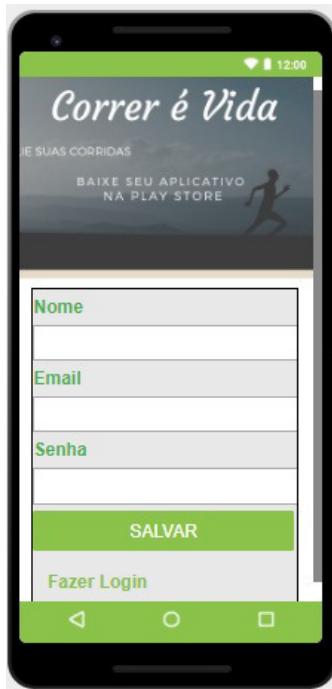


Figura 6.7: Tela de cadastro

Os últimos objetos que devemos inserir na tela de LoginCadastro são 4 objetos não visíveis, cuja funções serão explicadas mais adiante quando forem utilizados durante a programação. Localize a guia `Connectivity` e insira dois objetos `Web`. Altere seus nomes para `Web_Cadastro` e `Web_login`. Da guia `User Interface`, adicione um objeto `Notifier`, e da guia `Storage` acrescente um objeto `Tiny_DB`. A figura a seguir demonstra como ficará finalizada a área de todos os componentes com os objetos não visíveis.

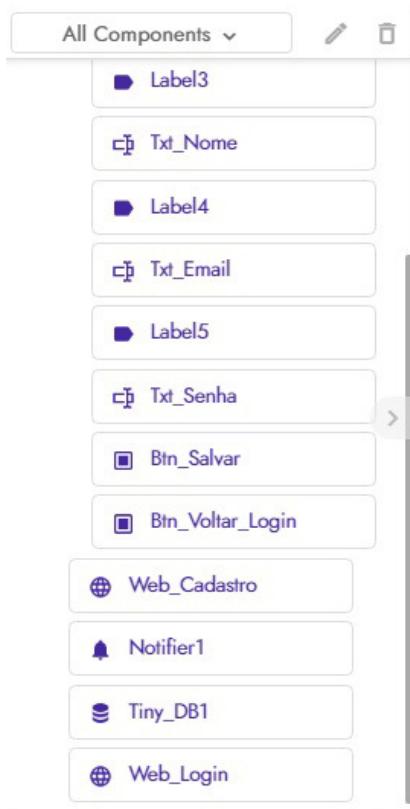


Figura 6.8: Objetos não visíveis

Com isso, finalizamos o leiaute da tela de Login e Cadastro de usuários. Não se esqueça, se quiser baixar o arquivo com todas as configurações realizadas no projeto até agora, ele está disponível no link descrito anteriormente.

## CAPÍTULO 7

# A TELA DE DETALHES DA CORRIDA

Chegamos ao desenvolvimento do leiaute da última tela do app de Avaliação de Corridas de Rua. É nela que o usuário verá a descrição completa da corrida, a nota de avaliação, os comentários dos demais usuários, o mapa do local da corrida e, caso esteja logado no app, poderá anotar sua avaliação e comentários.

Para isso, necessitamos de mais uma Screen. Acrescente-a como vimos anteriormente e renomeie-a para **Detalhes**. Será necessário realizar algumas modificações em suas propriedades para que fique com o design das demais telas. Faça as configurações nas propriedades conforme tabela exibida a seguir. Lembre-se de que já vimos como realizar essas modificações em capítulos anteriores.

Propriedade	Novo valor
Align Horizontal	Center
Navigation Bar Color	Light Green
Title Visible	Arrastar para a esquerda

Como já descrevemos os procedimentos da inserção e configuração de propriedades dos objetos que serão utilizados

nesta Screen, deixamos aqui um desafio para que o/a leitor/a possa desenvolver com seus conhecimentos já adquiridos a seguinte tela.



Figura 7.1: Tela de Detalhes

Veja a lista dos objetos que deverão ser inseridos:

- Image
- Space
- Label
- Space
- Label
- Horizontal Arrangement
- Button
- Button
- Space
- Horizontal Arrangement

- Label

Os valores das propriedades que utilizaremos em cada objeto estão demonstrados a seguir:

### Objeto Image

Propriedade	Novo valor
Height	200 pixels
Width	Fill parent
Picture	imagem1.png

### Objeto Space

Propriedade	Novo valor
Height	10 pixels

### Objeto Label

Propriedade	Novo valor
Font Bold	Ativar
Font Size	20
Width	90 percent
Text	Label_titulo
Renomear	Lbl_Titulo

### Objeto Space

Propriedade	Novo valor
Height	15 pixels

## Objeto Label

Propriedade	Novo valor
Font Size	14
Width	90 percent
Text	Label da descrição da corrida
Renomear	Lbl_Descricao

Abaixo da Label de descrição da corrida, como podemos visualizar na imagem a seguir, haverá uma área com dois botões. A função do primeiro é exibir o local da corrida e a do segundo, incluir uma avaliação e comentários sobre esta corrida.

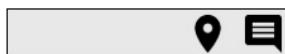


Figura 7.2: Botões de local e avaliação

## Objeto Horizontal Arrangement

Propriedade	Novo valor
Horizontal Align	Right
Height	35
Width	90

## Objeto Button

Propriedade	Novo valor
Height	35 pixels
Width	35 pixels
Text	Apagar o texto

Image	place1.png <sup>1</sup>
Renomear	Btn_Mapa

<sup>1</sup> A imagem place1.png está disponível para download através do link [http://nelfabbri.com/avalia\\_run](http://nelfabbri.com/avalia_run).

### Objeto **Button**

Propriedade	Alterar valor
Height	35 pixels
Width	35 pixels
Image	comment.png <sup>1</sup>
Text	Remova o texto exibido
Renomear o objeto	Btn_Comentar

<sup>1</sup> A imagem comment.png também está disponível para download no link do material deste livro. Essas imagens são apenas sugestões, podendo o/a leitor/a escolher outras conforme desejar.

### Objeto **Space**

Propriedade	Novo valor
Height	10 pixels

### Objeto **Horizontal Arrangement**

Propriedade	Novo valor
Width	90 percent

## Objeto Label

Propriedade	Novo valor
Font Size	50
Text	0.0
Font Typeface	Roboto Thin

## 7.1 OBJETO RATING BAR

Um recurso interessante que utilizaremos para mostrar a nota da avaliação é a utilização das estrelas, Rating Bar . Esse objeto é bem recente nessa plataforma, então vamos utilizá-la em nosso app. Insira o objeto Rating Bar ao lado direito da Label dentro da Horizontal Arrangement . Nas propriedades da Rating Bar , como esse objeto é apenas para a exibição da média de avaliação da corrida, ative a opção Is Indicator , bastando deslizar seu controle para a direita. Se não marcássemos essa propriedade, o usuário poderia alterar os valores exibidos e isso não é adequado para esse momento.

As avaliações serão dadas de meio em meio pontos e para habilitar essa pontuação devemos na sua propriedade Set Step Size alterar o valor para .5 . Para as estrelas ficarem com a mesma cor de nosso design, mude a propriedade Star Color para a cor Light Green .

Veja na imagem a seguir, como deverá estar momentaneamente o desenvolvimento de sua Screen Detalhes.



Figura 7.3: Tela de Detalhes

Logo abaixo necessitamos inserir mais um objeto para dar um espaçamento entre a horizontal que exibe a avaliação e o mapa que vamos inserir para mostrar a localização da corrida selecionada. Insira então um objeto Space abaixo da Horizontal Arrangement e configure sua propriedade Height para 10 pixels .

## 7.2 INSERINDO UM MAPA NA SCREEN

Estamos prontos para incluir um mapa com a indicação do local da corrida. Acesse a guia Maps da Palette e localize o objeto Map . Insira-o abaixo do espaço que acabamos de inserir. Algumas configurações em suas propriedades serão necessárias. Vamos definir a altura que o mapa terá na tela para 200 pixels ,

bastando alterar a propriedade `Height`. A largura do mapa deverá ter o espaço total do dispositivo, bastando para isso, definir a propriedade `Width` para `Fill Parent`.

O componente `Map`, necessita de um valor para a latitude e longitude para exibição antes de mostrar a corrida selecionada. Para isso, na propriedade `Center From String` digite os valores da latitude e longitude separados por uma vírgula. Como sugestão utilize as coordenadas `-23.637421, -46.577917`. Para o usuário poder alterar o efeito de zoom na exibição do mapa, vamos ativar a propriedade `Enable Zoom` deslizando para a direita. Vamos ajustar o nível de exibição do mapa, alterando a propriedade `Zoom Level` para tamanho `15`.

No mapa vamos precisar do marcador de local da corrida que está selecionada. O componente que cria o marcador está na guia `Maps` da `Palette`, e se chama `Marker`. Ele deverá ser inserido no mapa que está em exibição na sua `Screen Detalhes`. Vamos alterar algumas de suas propriedades. A cor do marcador que será exibido deverá ser definida como `Green` na propriedade `Fill Color`. O tamanho do marcador também deverá ser modificado, vamos definir as propriedades `Height` e `Width` para `35 pixels` cada uma. Para o marcador, necessitamos definir o local onde ele deverá ser exibido dentro do objeto mapa, bastando informar o valor `-23.637421` na propriedade `Latitude` e `-46.577917` na `Longitude`.

Veja na figura a seguir como deverá estar a visualização de sua tela até o momento:

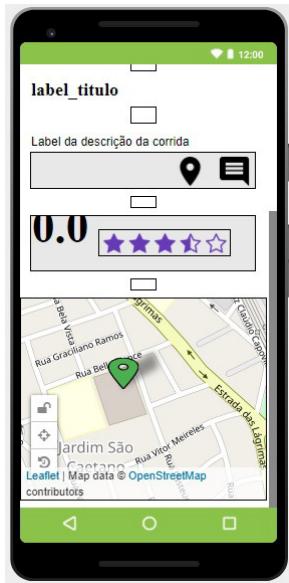


Figura 7.4: Exibindo o objeto Map

Para a exibição da lista de comentários efetuados por todos os usuários, vamos utilizar um objeto que realize a exibição da lista. O objeto `List View` é responsável por esta tarefa. Ele encontra-se na guia `User Interface`, e deverá ser posicionado logo abaixo do mapa em sua área `Viewer`. Acesse suas propriedades para realizar as modificações da cor de fundo da `List View`, que deverá ficar marcado como `White`, para isso acesse a propriedade `Background Color`. A propriedade `Divider Color`, que indica a cor da linha que separará um comentário de outro, deverá ser selecionada como `Light Green`. O tamanho do texto que exibirá os comentários deverá ficar em `18` em `Text Size`. A largura do `List View` deverá ser configurada para que a propriedade `Width` fique com o valor de `90 percent`.

## 7.3 PREPARANDO A ÁREA DE AVALIAÇÃO

Agora precisamos de um espaço para o usuário realizar a avaliação da corrida. Neste espaço ele indicará a nota através do objeto Rating Bar e poderá digitar um texto com a sua opinião. Todos esses controles estarão dispostos dentro de uma Vertical Arrangement , que deverá ser incluída abaixo da ListView1 que foi posicionada em sua Screen Detalhes. Na imagem a seguir, vemos como ficará a Vertical Arrangement , após a realização e configuração de seus objetos que serão realizadas a seguir.

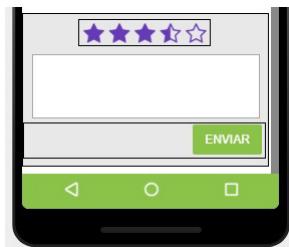


Figura 7.5: Área de avaliação Map

Vamos fazer alguns ajustes nessa Vertical Arrangement . Para que tudo em seu interior fique centralizado, altere a propriedade Align Horizontal para a opção Center . Para o tamanho que o objeto ocupará em tela, deixe a propriedade Width como Fill Parent .

Altere também o nome do objeto Vertical Arrangement para Va\_avaliacao .

Vamos inserir os objetos da avaliação dentro da Va\_avaliacao . O primeiro que devemos inserir é o Rating Bar . Altere o nome do objeto para Rating\_avaliar . A única

propriedade que deveremos alterar é a que indica a cor das estrelas selecionadas. Altere a propriedade `Star Color` para `Light Green`.

Devemos inserir uma caixa de texto logo abaixo do objeto `Rating_avaliar` (dentro da `Va_avaliação`), para o usuário deixar seu comentário registrado. Após sua inserção, vamos realizar algumas alterações nas propriedades da `Textbox`. Para a propriedade `Height` que define a sua altura, digite o valor `60 pixels` e para a largura digite `90 percent` em `Width`. Para a propriedade `Hint`, que mostra uma dica do que deverá ser digitado pelo usuário, digite a frase `Deixe seu comentário`. Essa informação também poderá ter a sua cor alterada, e é exatamente isso que a propriedade `HintColor` faz, então selecione a opção de cor como `Light Green`. Como a avaliação do usuário poderá ser maior do que uma linha, deveremos deixar a propriedade `MultLine` ativada, deslizando sua opção para a direita. Renomeie a `Textbox` para `Txt_comentar`.

Como mostra a figura exibida anteriormente, o botão de enviar está posicionado à direita da tela, para isso necessitamos inserir uma `Horizontal Arrangement`, abaixo do `Textbox` do comentário e realizar a configuração da propriedade `Align Horizontal` para `Rigth` e alterar seu tamanho `Width` para `Fill Parent`.

Dentro dessa `Horizontal`, insira um objeto `Button` que servirá para enviar sua avaliação e comentários para registro. Configure as propriedades do `Button`, para que fique como o nosso design. Primeiramente, altere a cor de fundo do botão através da propriedade `Background Color` e selecione a cor `Light Green`.

Para o seu texto de exibição, escreva a palavra `Enviar`. Não podemos nos esquecer de renomear o nome do objeto para `Btn_enviar_comentario`.

O objeto `VA_avaliar` só deverá ser exibido quando o usuário clicar no botão com essa função. Então selecione o `VA_avaliar` e deixe sua propriedade `visible` marcada como `falsa`, para isso basta deslizar seu marcador para a esquerda.

Para finalizar o leiaute da tela de Detalhes, vamos inserir os componentes não visíveis que farão parte do app e que serão explicados em capítulos futuros quando de sua utilização. Da guia `User Interface` adicione um componente `Bottom Sheet` e um `Notifier`. Da guia `Connectivity`, adicione três componentes `Web` e da guia `Storage` adicione um `Tiny Db`.

Vamos renomear cada objeto `Web`, com os seguintes nomes: `Web_get_Detalhes`, `Web_Comentarios` e `Web_busca_Comentarios`.

A imagem a seguir, mostra todos os objetos não visíveis que estamos utilizando na Screen Detalhes.

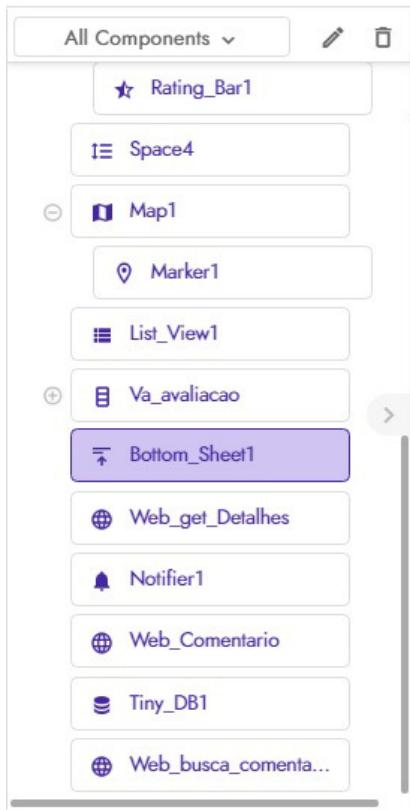


Figura 7.6: Objetos não visíveis

Terminamos assim a criação de todas as telas de nosso aplicativos de Avaliação de Corridas de Rua e estamos aptos para continuar com o desenvolvimento dos blocos de comando que darão a funcionalidade ao nosso app.

Caso deseje, baixe o projeto desenvolvido até aqui para estudos, no link dos materiais do livro.

## CAPÍTULO 8

# CADASTRANDO AS CORRIDAS

Por lógica, faz-se necessário dar uma pausa do desenvolvimento no ambiente do Kodular e iniciarmos o desenvolvimento da página de cadastro de corridas utilizando a linguagem de formatação **HTML** e a de programação web **PHP**, pois estes dados inseridos e cadastrados serão utilizados para exibição no app.

As corridas que aparecerão no aplicativo serão cadastradas por uma página que já está disponível online através do endereço: [http://www.nelfabbri.com.br/avalia\\_run](http://www.nelfabbri.com.br/avalia_run). Ao acessá-la o/a leitor/a verá a tela exibida na imagem a seguir.

## Cadastro de Corridas de Rua

The screenshot shows a registration form for a race. The fields are as follows:

- Nome da Corrida: [Text input field]
- Descrição: [Text input field]
- Foto: [File input field] with buttons "Escolher arquivo" (Select file) and "Nenhum arquivo selecionado" (No file selected)
- Telefone: [Text input field]
- Latitude e Longitude: [Text input field]
- Endereço: [Text input field]
- Horários: [Text input field]

At the bottom are two buttons: "Cadastrar" (Register) and "Limpar" (Clear).

Figura 8.1: Tela de cadastro online

A tela foi criada utilizando HTML e formatada com CSS (*Cascade Style Sheet*). Decidimos não exibir e nem explicar os códigos em CSS, para que o código fique mais simples para o seu entendimento, visto que esse não é o foco do livro. Os códigos demonstrados a seguir funcionarão perfeitamente.

No formulário, o usuário deverá inserir obrigatoriamente os seguintes dados: o nome, uma descrição, selecionar a imagem que vai aparecer no app, um telefone de contato, as coordenadas da latitude e longitude separadas por uma vírgula, o endereço e o horário da corrida.

Após o preenchimento, ao clicar no botão `Cadastrar`, as informações serão enviadas para o banco de dados e estarão disponíveis para avaliação de todos os usuários através do app.

Vamos conhecer o arquivo que executa o recebimento das informações pelo usuário e posterior cadastro em banco de dados.

O arquivo que realiza essas tarefas é o `index.php` que está online e não precisará ser digitado por você, apenas estamos exibindo e comentando seus códigos para um melhor entendimento.

Note que as duas sequências de códigos exibidas a seguir fazem parte de um único arquivo, apenas separamos a parte inicial que está na linguagem HTML da segunda que está em PHP, para uma melhor explicação.

## 8.1 PARTE EM HTML

```
1.   <html>
2.   <head> <title> App Corridas de Rua </title> </head>
3.   <body>
4.   <center><h1>Cadastro de Corridas de Rua</h1>
5.   <form name="form" action="index.php" method="POST" enctype=
 "multipart/form-data">
6.   <table width="40%" border="0" >
7.   <tr><td> Nome da Corrida</td><td> <input type='text' name='
 nome'> <br></td></tr>
8.   <tr><td> Descrição:</td><td> <input type='text' name='descri
 icao'> <br></td></tr>
9.   <tr><td> Foto: </td><td><input type='file' name='foto'> <br>
</td></tr>
10.  <tr><td> Telefone:</td><td> <input type='text' name='fone'>
 <br></td></tr>
11.  <tr><td> Latitude e Longitude:</td><td> <input type='text'
 name='google_maps'> <br></td></tr>
12.  <tr><td> Endereço: </td><td><input type='text' name='endereco'>
 <br></td></tr>
13.  <tr><td> Horários:</td><td> <input type='text' name='horarios'>
 <br></td></tr>
14. <tr><td colspan=2> <center> <input type="Submit" value="Cadas
```

```
trar" name="cadastre">
15.    <input type="Reset" value="Limpar"></td></tr>
16.  </table>
17.  </form>
18.  </body>
19. </html>
```

Vamos entender os códigos do arquivo apresentado, que é responsável pelo recebimento dos dados digitados pelo usuário para o cadastro de uma nova corrida.

Linha 1 - Abertura da estrutura do arquivo em HTML.

Linha 2 - Seção do cabeçalho do documento e seu título, preparando que seja exibida na barra de títulos do seu navegador a mensagem **App Corridas de Rua**.

Linha 3 - TAG de inicialização do corpo do documento, isto é, a partir deste ponto as informações serão exibidas no navegador.

Linha 4 - Formatação e exibição do título centralizado **Cadastro de Corridas de Rua**.

Linha 5 - Comando que inicia o formulário para digitação das informações da corrida. Quando o usuário clicar no botão **Cadastrar**, as informações serão enviadas através do método **POST** para o próprio arquivo **index.php**.

Linha 6 - Criação de uma tabela com largura de 40% da tela, que armazenará os espaços para a digitação das informações.

Linhos de 7 a 13 - Cada linha está preparando a digitação de um dado específico da corrida. Cada informação ficará armazenada em uma variável que é identificada após a propriedade **name** encontrada em cada TAG.

Linha 14 - Criação do botão `Cadastrar`. Quando ele for clicado pelo usuário, as informações serão enviadas para o próprio arquivo, porém elas serão processadas pela parte preparada em PHP. O valor `cadastre` será utilizado para indicar o início do processamento em PHP.

Linha 15 - Criação do botão de `Limpar` os dados digitados no formulário.

Linha 16 - Finaliza a estrutura da TAG da tabela.

Linha 17 - Término da TAG do formulário de cadastro.

Linha 19 - Encerramento da estrutura do arquivo em HTML.

Como já dissemos, os códigos apresentados a seguir fazem parte do mesmo arquivo `index.php` analisado anteriormente e são os responsáveis por adicionar os dados da corrida no banco de dados e estão na linguagem PHP.

## 8.2 PARTE EM PHP

```
1.      <?php
2.      if (isset($_POST['cadastre'])) {
3.          mysqli_set_charset('utf8');
4.          require "Db.class.php";
5.          $db = new Db();
6.          $nome = $_POST["nome"];
7.          $descricao=$_POST["descricao"];
8.          $telefone=$_POST["fone"];
9.          $google_maps=$_POST["google_maps"];
10.         $endereco=$_POST["endereco"];
11.         $horarios=$_POST["horarios"];
12.         $foto=$_FILES['foto']['tmp_name'];
13.         $media=0;
14.         if ($nome=="" or $descricao=="" or $telefone=="" or $google
```

```

_maps=="" or $endereco=="" or $horarios=="" or $foto=="") {
15.     echo "Preencha todos os dados";
16.     return;
17.     $tamanho_permitido = 512000;
18.     $pasta = 'imagens';
19.     if (!empty($foto)) {
20.         $file = getimagesize($foto);
21.         if($_FILES['image']['size'] > $tamanho_permitido){
22.             echo "erro - arquivo muito grande";
23.             exit();
24.             if(!preg_match('/^image\/(?:gif|jpg|jpeg|png)$i', $file['
mime'])) {
25.                 echo "erro - extensão não permitida";
26.                 exit();
27.                 $extensao = str_ireplace("/", "", substr($file['mime'], "/"));
28.                 $novoDestino = "{$pasta}/foto_arquivo_".uniqid('', true) .
'.' . $extensao;
29.                 move_uploaded_file ($foto , $novoDestino );
30.                 $novoDestino = "http://www.nelfabbri.com/avalia_run/" . $
novoDestino;
31.                 $sql_Cadastro = "INSERT INTO corridas (nome, descricao, fo
to, telefone, google_maps, endereco, horarios, media) VALUES ('$ nome', '$descricao', '$novoDestino', '$telefone', '$google_maps', '$en
dereco', '$horarios', '$media')";
32.                 $resultado = $db->query($sql_Cadastro);
33.                 if ($resultado) {
34.                     echo "Cadastrado com sucesso";
35.                 } else{
36.                     echo "Erro ao salvar informações no Banco de Dados";
37.                 }
38.             ?>
```

Vamos descrever as linhas do código do programa em PHP.

Linha 1 - Abertura da estrutura do arquivo em PHP.

Linha 2 - Verifica se há objeto `cadastre` no formulário.

Linha 3 - Define a tabela utilizada para acentuação para UTF-8 .

Linha 4 - Executa o arquivo `conecta.php`, que é responsável pela conexão com o banco de dados.

Linha 5 - Cria novo objeto de nome `$db`.

Linhas de 6 a 12 - Armazenam nas variáveis locais em PHP os valores digitados pelo usuário no formulário em HTML.

Linha 13 - Inicializa variável `$media` com o valor 0.

Linha 14 - Verifica se todos os valores do formulário foram preenchidos pelo usuário.

Linha 15 - Caso algum campo não tenha sido preenchido, será exibida a mensagem **Preencha todos os dados**.

Linha 16 - Retorna o comando para o usuário completar todos os dados da corrida.

Linha 17 - Define em 512 Kb o tamanho máximo que o arquivo da imagem deverá possuir.

Linha 18 - Define uma variável com o nome da pasta no servidor, onde serão armazenadas as imagens das corridas.

Linha 19 - Verifica se a variável `$foto` não está vazia, pois sem a imagem não podemos cadastrar uma nova corrida.

Linha 20 - Identifica o tamanho da imagem que foi enviada pelo usuário.

Linha 21 - Verifica se o tamanho do arquivo é maior que o permitido.

Linha 22 - Caso o tamanho seja maior que o permitido, será

exibida a mensagem para o usuário - **erro - arquivo muito grande.**

Linha 23 - Retorna o comando para o usuário completar todos os dados da corrida.

Linha 24 - Verifica se a extensão do arquivo da imagem enviada é uma das permitidas - gif , jpg , jpeg ou png .

Linha 25 - Caso a extensão não seja nenhuma das testadas, exibirá a mensagem **erro - extensão não permitida.**

Linha 26 - Retorna o comando para o usuário completar todos os dados da corrida.

Linha 27 - Captura a extensão do arquivo enviado.

Linha 28 - Prepara o novo destino para salvar a imagem enviada.

Linha 29 - Comando que realiza o upload do arquivo para o servidor.

Linha 30 - Atualiza o valor da variável de destino do arquivo para gravar no banco de dados.

Linha 31 - Armazena na variável o valor do comando da linguagem em SQL que armazenará os campos no banco de dados.

Linha 32 - Executa a inserção dos valores no banco de dados. O resultado da execução deste comando retornará o valor verdadeiro ou falso.

Linha 33 - Verifica se o resultado do comando foi verdadeiro.

Linha 34 - Caso seja verdadeiro, exibirá a mensagem para o

usuário **Cadastrado com sucesso.**

Linha 35 - O comando `else` trata a opção falso da linha 32.

Linha 36 - Se for falso, exibirá a mensagem **Erro ao salvar informações no Banco de Dados.**

Linha 37 - Encerramento da estrutura do primeiro `IF` aberto na linha 2.

Linha 38 - Encerramento do arquivo em PHP.

Após o entendimento do arquivo demonstrado anteriormente, já podemos continuar o desenvolvimento do app no ambiente do Kodular. No próximo capítulo, veremos a programação dos comandos que darão funcionalidade ao app Avaliação de Corridas de Rua.

## CAPÍTULO 9

# PROGRAMANDO A TELA DE DESTAQUES

Após terminado o desenvolvimento do leiaute de todas as telas do app, chegamos ao momento em que vamos dar a funcionalidade às telas que criamos, ou seja, realizaremos a programação para que cada uma delas execute o que planejamos para o app.

Vamos aprender a utilizar alguns dos objetos não visíveis que inserimos nos capítulos anteriores e finalmente veremos sua utilidade.

A funcionalidade desta tela será a de exibir as 3 melhores corridas avaliadas e que estão disponíveis em um banco de dados. Como poderá levar um tempo para o carregamento, para que o usuário não tenha a impressão de que o app está travado, vamos colocar uma notificação que mostre a ele que os dados estão sendo carregados enquanto a requisição está sendo feita.

A primeira tela que vamos programar será a tela inicial, a Screen1. Caso você não esteja com essa tela na visualização, deve primeiro posicioná-la para evitar que programemos em lugar errado. Localize o botão no menu superior esquerdo que exibe a

tela na qual estamos trabalhando no momento (no caso, estamos na tela **Detalhes**), e selecione o nome da tela que desejamos exibir (Screen1). A imagem a seguir exibe essa opção.



Figura 9.1: Selezionando a tela para exibição

## 9.1 ACESSANDO A ÁREA DOS BLOCOS

Para acessar a área dos códigos da programação, deveremos localizar e clicar no botão **BLOCKS**, que se encontra no canto superior direito de sua tela. A figura a seguir exibe o botão que dá acesso aos blocos de programação.



Figura 9.2: Botão de acesso aos blocos de programação

Após seu clique, será exibida a tela conforme a próxima figura demonstra.

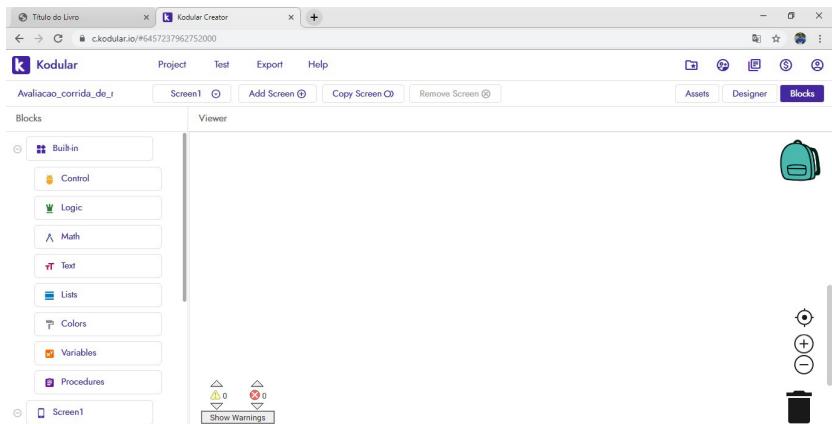


Figura 9.3: Tela de programação dos blocos

Nela serão inseridos todos os comandos que estão representados através de blocos e que darão funcionalidade ao app desenvolvido. Na imagem exibida a seguir, localizamos a área **Blocks** à esquerda da tela e é exatamente nela que encontraremos os blocos de comandos.

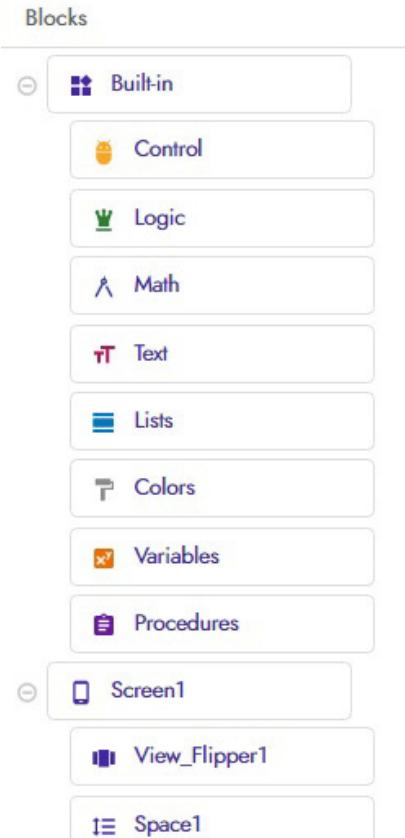


Figura 9.4: Área de seleção dos blocos de comando

## 9.2 INSERINDO OS PRIMEIROS BLOCOS

Como já falamos na descrição do projeto, a Screen1 vai apresentar um efeito de carrossel com as imagens. Para que isso ocorra, além de já ter inserido e configurado o objeto View\_Flipper , será necessário ativar o efeito de transição das imagens. Essa ativação se dará no exato momento em que o aplicativo for aberto pelo usuário, ou seja, será preciso programar

o **evento** de inicialização da Screen1.

Um **EVENTO** é o resultado de uma ação, que pode provocar uma reação ou um conjunto de reações.

Para se ter acesso aos blocos dos eventos e dos comandos, localize na área de blocos o nome Screen1 e dê um clique sobre ele. Nesse exato momento, serão exibidos todos os blocos de controle que esse objeto possui. A figura a seguir exibe a opção de seleção de eventos e comandos.

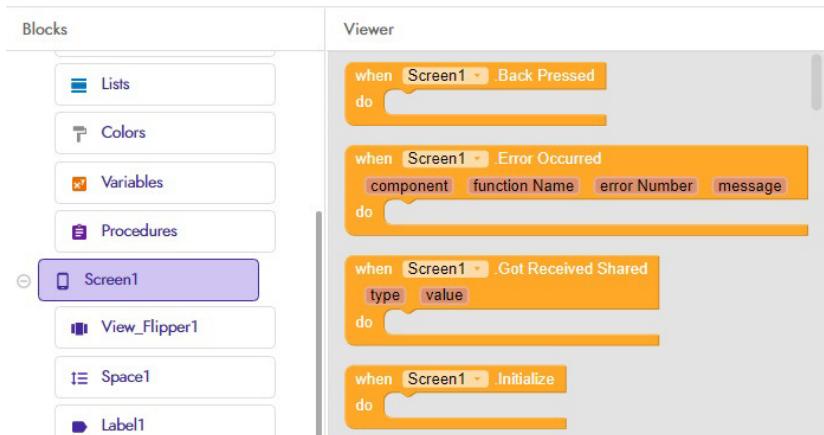


Figura 9.5: Seleção de eventos e comandos

Como desejamos programar o evento de inicialização da Screen para que as imagens fiquem em movimento, devemos primeiramente inserir o bloco que representa a inicialização da tela e arrastá-lo para a área de programação ao lado direito. Para tanto, localize o bloco do evento Initialize da Screen1, clique e

arraste para a área `Viewer`. Veja na figura a seguir o bloco inserido.

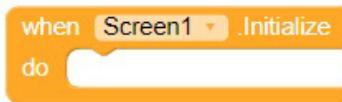


Figura 9.6: Bloco initialize

Todos os demais blocos de comando que serão executados quando o app for inicializado deverão ficar agrupados dentro do evento `Initialize`.

A primeira programação que vamos inserir é a que ativa a transição das figuras do `View_Flipper1`, o bloco de comando responsável por esta tarefa é o `call View_Flipper1.Start Flipping`. Para localizá-lo, acesse a guia dos blocos e encontre o objeto `View_Flipper1`. Dê um clique sobre ele para ver a lista disponível de todos os seus blocos de comando e localize o bloco `call View_Flipper1.Start Flipping`, clique sobre ele, arraste e encaixe dentro do bloco de inicialização da `Screen1`.

A imagem a seguir demonstra como está o desenvolvimento até aqui.



Figura 9.7: Bloco de inicialização da `Screen1`

Após terminada a programação do objeto `View_Flipper1`, vamos implementar um recurso que exibe ao usuário uma

mensagem enquanto os dados estão sendo buscados e processados para futura exibição. O objeto que exibe avisos ao usuário é o `Notifier`. Ele foi inserido durante a montagem do leiaute e é um dos objetos não visíveis da Screen1, pois sua exibição depende de uma chamada durante a programação.

Na guia dos blocos, ao clicar sobre o `Notifier1`, serão mostrados todos os comandos disponíveis deste objeto. Localize o bloco de comando `Call Notifier1.ShowProgress Dialog` que é responsável pela exibição da mensagem, clique sobre ele e posicione-o logo abaixo do bloco de comando `View_Flipper1.Start Flipping`, conforme demonstra a figura a seguir.



Figura 9.8: Bloco Call Notifier1.ShowProgress Dialog

Observe que, no último bloco que inserimos, existem dois locais para encaixar algumas informações. O primeiro indica uma mensagem que será exibida para o usuário e o segundo, o título da mensagem. Em ambos os casos, devemos inserir um bloco de texto. Para ter acesso aos blocos de texto, localize na guia de blocos a seção `Built-in`. Nela você encontrará a opção `Text` e ao clicar nela surgirão as possíveis opções de trabalho com um texto. A primeira opção demonstra um texto vazio, e é disso que necessitamos no momento. Veja na imagem a seguir o local dos blocos de texto na `Built-in`.

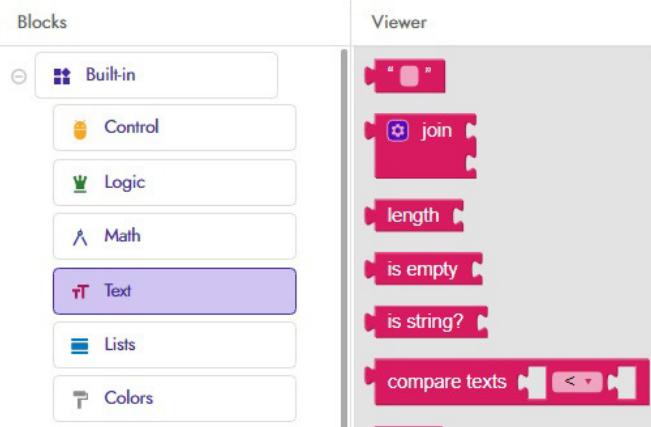


Figura 9.9: Comandos de texto na seção Built-in

Selecione o bloco de texto vazio e arraste-o para a área **Viewer**. Posicione-o para encaixar na opção **message** da **Notifier1**. Repita o processo para inserir mais um bloco de texto vazio e posicione-o no espaço referente ao título da **Notifier1**.

No espaço de texto para a mensagem, clique em seu interior e digite a informação que será exibida ao usuário assim que o aplicativo for inicializado - **Buscando Dados**. Veja na imagem a seguir o resultado dos blocos até o momento.

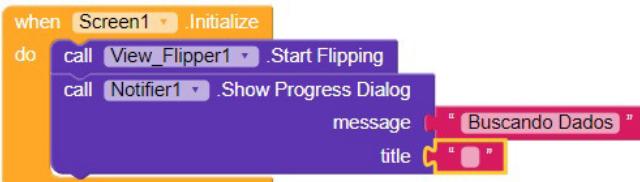


Figura 9.10: Comandos do bloco de inicialização

Observe que não inserimos nenhum texto para o título da notificação, pois a exibição de um título é uma tarefa opcional.

Como o objeto `Notifier1` não possui um bloco de comando sem a opção do título e não podemos deixar nenhum bloco de comando incompleto, a saída foi deixar o bloco de texto vazio.

Enquanto a caixa de diálogo da notificação estiver sendo exibida, precisamos realizar o procedimento para que se exibam as 3 corridas com melhor avaliação pelos usuários.

Para exibir as 3 corridas mais bem avaliadas, vamos utilizar um arquivo PHP que realiza um filtro entre todas as corridas cadastradas. Você pode tentar criá-lo por conta própria se entender de PHP, mas se preferir, siga utilizando o arquivo que pré-disponibilizamos, pois assim criaremos um app colaborativo, onde todas as informações estarão disponíveis para todos/as os/as leitores/as que estão desenvolvendo este app. Subimos nosso arquivo `destaques.php` para o site [http://www.nelfabbri.com/avalia\\_run/destaques.php](http://www.nelfabbri.com/avalia_run/destaques.php), então precisamos passar este endereço para nosso app poder buscar as informações lá.

Precisamos que nosso app envie uma solicitação de recebimento destes dados. O componente que realiza funções para solicitações HTTP `GET` , `POST` , `PUT` e `DELETE` é o objeto não visível `WEB` . Na guia de blocos, localize o objeto `Web1` e inclua o bloco `set Web1.URL` to que indicará o local onde estará hospedado seu arquivo em PHP. Posicione-o também dentro do evento de inicialização da `Screen1`, logo abaixo o bloco `Notifier1` .

Para indicar o endereço do arquivo em PHP, acrescente e encaixe um bloco de texto vazio na frente do bloco `set Web1.URL to`. Dentro do bloco de texto vazio, devemos digitar o endereço `http://www.nelfabbri.com/avalia_run/destaques.php`.

O arquivo `destaques.php` encontra-se disponível online e suas funções estão comentadas no capítulo final deste livro.

A imagem a seguir exibe o atual estágio do bloco de inicialização da tela com o endereço do arquivo `destaques.php` configurado.

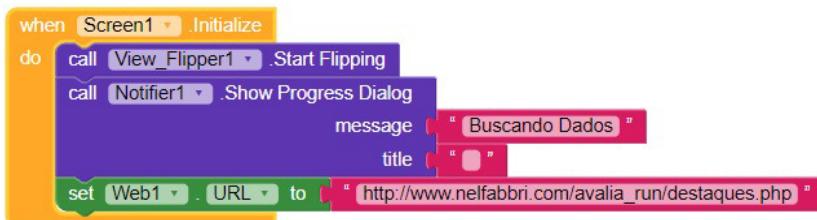


Figura 9.11: Comandos do bloco de inicialização

Após informarmos o endereço do arquivo em PHP, necessitamos executar o procedimento para receber esses dados. O bloco do objeto `Web1` que realiza esta função é o `Web1.Get`, localize-o na guia de blocos e insira-o logo abaixo da configuração do endereço do arquivo em PHP. A figura a seguir mostra o bloco de inicialização da `Screen1` concluída.



Figura 9.12: Bloco de inicialização concluído

## 9.3 DECLARANDO VARIÁVEL DE PROGRAMAÇÃO

Variável de programação é uma **região da memória** do seu dispositivo previamente identificada com um nome, cuja função é armazenar dados ou informações por um determinado espaço de tempo.

Necessitamos de uma variável que armazene no app todas as corridas que serão retornadas da consulta do banco de dados através arquivo em PHP realizada pelo objeto `Web1`. Na guia de blocos `Built-in`, localize a seção de `Variables` para exibir as opções disponíveis de trabalho com variáveis. A imagem a seguir demonstra as opções das variáveis.

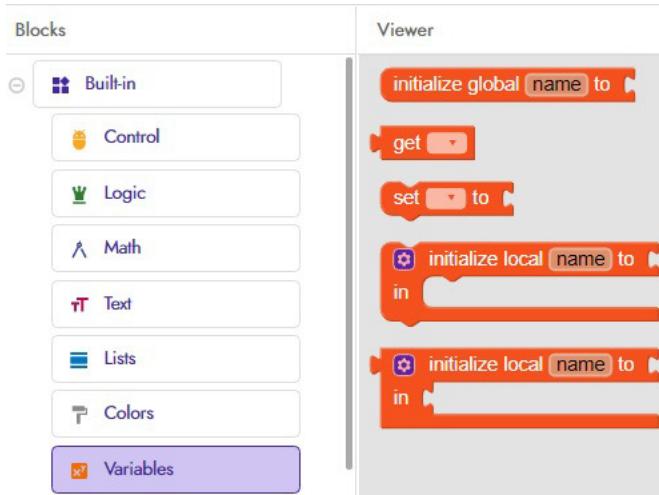


Figura 9.13: Seção de comandos das variáveis

Inclua na área **Viewer** dos blocos uma inicialização de variável global, conforme demonstra imagem a seguir.



Figura 9.14: Comando de inicialização de variável

Para dar nome à variável, clique sobre a palavra `name` e digite a palavra `corridas`.

Toda variável necessita ser definida com algum tipo ou formato para poder armazenar informações. Como receberemos uma lista de três corridas do arquivo em PHP, vamos definir a variável como sendo do tipo lista. Para isso, acesse a área **Built-in** na seção de listas conforme exibido na imagem a seguir.

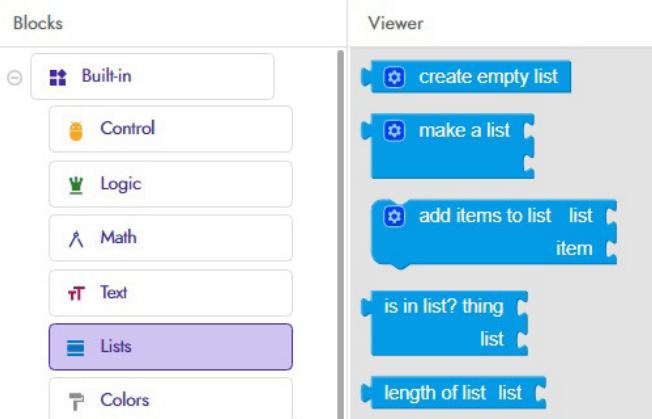


Figura 9.15: Seção de comandos de listas

Identifique a opção `create empty list` que cria uma lista vazia, e encaixe-a no bloco de criação de variável, conforme demonstrado na imagem a seguir.

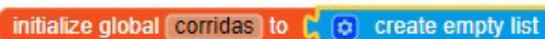


Figura 9.16: Declaração de variável do tipo lista

## 9.4 EXIBINDO AS CORRIDAS EM DESTAQUE

O próximo passo é tratar o evento que indica que o app recebeu as informações solicitadas através do objeto `Web1`. Esse evento é o `Got Text`. Ele é responsável pelo recebimento dos dados do arquivo em PHP após o seu processamento.

Na guia de blocos do objeto `Web1`, localize o evento `When Web1.Got Text` e insira-o na sua área de programação. A imagem a seguir exibe o bloco que vamos programar a partir de agora.

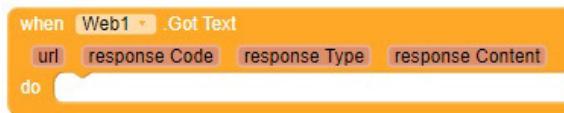


Figura 9.17: Bloco do evento When Web1.Got Text

Quando o objeto Web1 receber as informações do banco de dados, vamos armazená-las na variável global corrida através do bloco de comando set global corridas to . Para localizar esse comando, passe o ponteiro do mouse sobre o nome da variável no bloco de inicialização para que sejam exibidas as opções, conforme imagem demonstrada a seguir.



Figura 9.18: Gerando o comando set global corridas to

Selecione o comando set global corridas to , pois é através do comando set que indicamos que a variável receberá um determinado valor, e posicione-o dentro do evento Got Text do objeto Web1 , conforme a próxima imagem.

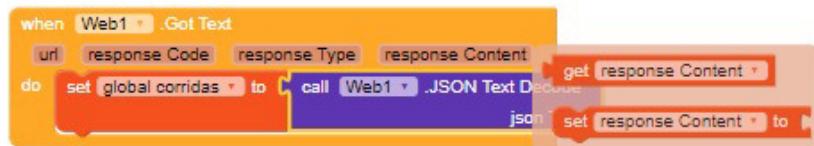


Figura 9.19: Variável que receberá um valor

As informações que chegarem como retorno do PHP estarão codificados no formato JSON e por esse motivo, necessitaremos de um bloco que faça a sua **decodificação** e então utilizá-las no app. Esse bloco de comando é o `Call Web1.JSON Text Decode`. Encontre-o na guia de blocos referente ao objeto `Web1` e o posicione no bloco da variável corridas, conforme demonstra a imagem a seguir.

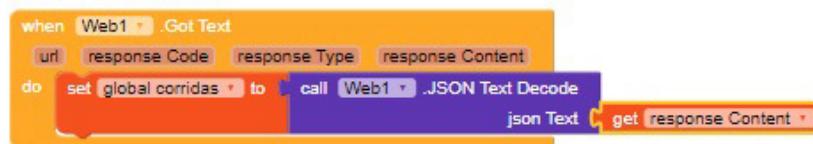


Figura 9.20: Posicionado o bloco Call Web1.JSON Text Decode

**JSON** JavaScript Object Notation. É basicamente um formato leve de troca de informações/dados entre sistemas.

O retorno da informação que teremos através da codificação JSON é exibida no seguinte formato:

```
[{"id": "7", "nome": " dado de cadastro", "descricao": " dado de cadastro"}]
```

O formato JSON sempre retornará uma lista com as informações com o nome de um campo e o seu respectivo valor à sua frente.

Todos os valores do resultado do processamento do arquivo `detalhes.php` estarão disponíveis para utilização em nosso app através do bloco de comando `get response Content` que

poderá ser obtido posicionando o ponteiro do mouse sobre a opção response Content do evento Got Text da sua Web1 . A imagem a seguir demonstra esse processo.

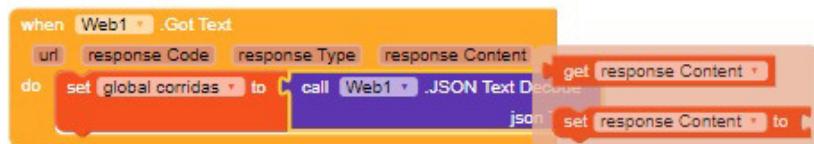


Figura 9.21: Gerando o bloco do comando get response Content

Selecione a opção get response Content clique e arraste para encaixar no bloco de decodificação, conforme demonstra a figura a seguir.

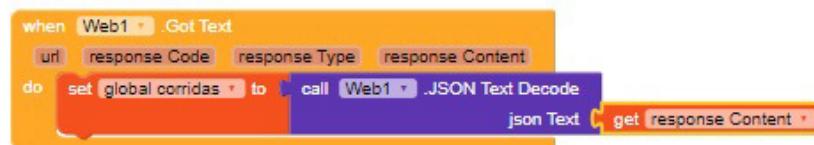


Figura 9.22: Recebendo os valores decodificados enviados pelo arquivo PHP

Após a decodificação realizada anteriormente, precisamos de uma lista para exibir as informações das corridas. O responsável por essa função é o objeto List\_View\_Image\_and\_Text1 . Vamos inseri-lo acessando a guia de blocos e localizando o objeto List\_View\_Image\_and\_Text1 . Em seus blocos de comando identifique o call List\_View\_Image\_and\_Text1.add item , posicione-o logo abaixo do recebimento dos dados demonstrado anteriormente. Este bloco de comando tem a finalidade de inserir um único item em sua lista. Como primeira linha da lista, vamos acrescentar uma identificação que servirá de título.

Observe que o bloco que inserimos possui três opções de encaixe: `image` , `title` e `subtitle` . Em cada uma das suas posições, insira um bloco de texto vazio. O primeiro espaço é referente à imagem que aparecerá na lista, digite no bloco de texto o nome da figura que será exibida no topo da lista: `corrida.jpg` . Na opção de título da `List_View_Image_and_Text1` digitaremos `As mais TOPs` e, no espaço reservado ao subtítulo, digite `confira` . Veja na figura a seguir a montagem do bloco que acabamos de descrever.

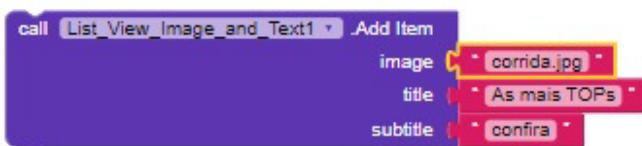


Figura 9.23: call List\_View\_Image\_and\_Text1.add item

Teremos que inserir cada uma das três corridas em cada uma das linhas da `List_View_Image_and_Text1` , para isso vamos realizar uma leitura em todo o resultado recebido pelo `Web1` e fazer a separação. Dentro de cada registro retornado, ainda precisamos separar os campos para exibição, ou seja, queremos identificar o campo para exibir **a foto, o nome da corrida e a sua descrição**. Para percorrermos por cada registro retornado, utilizaremos o comando `for each` , e indicaremos a lista de registros que foi recebida pelo objeto `Web1` .

O comando `for each item in list` está disponível acessando a guia `Built-in` na seção `control` . Insira-o abaixo da criação da linha de cabeçalho.

Clique no nome `item` do bloco de comando `for each item`

`in list`, e altere seu nome para `corridas`, pois assim torna-se mais fácil sua identificação, indicando que cada item do comando `for each item in list` é uma corrida.

Como já falamos anteriormente, os dados retornados do banco de dados através do arquivo `destaques.php` estão codificados em JSON, por esse motivo necessitamos realizar a decodificação novamente. Veja na imagem a seguir como deverá estar programado o comando do bloco `for each`.



Figura 9.24: Comando do bloco `for each`

Agora que já preparamos a leitura da lista das corridas, vamos acrescentar as informações na `List_View_Image_and_Text1`. Adicione o bloco `call List_View_Image_and_Text1.Add Item`, dentro do bloco de comando `for each` conforme demonstra a imagem a seguir.

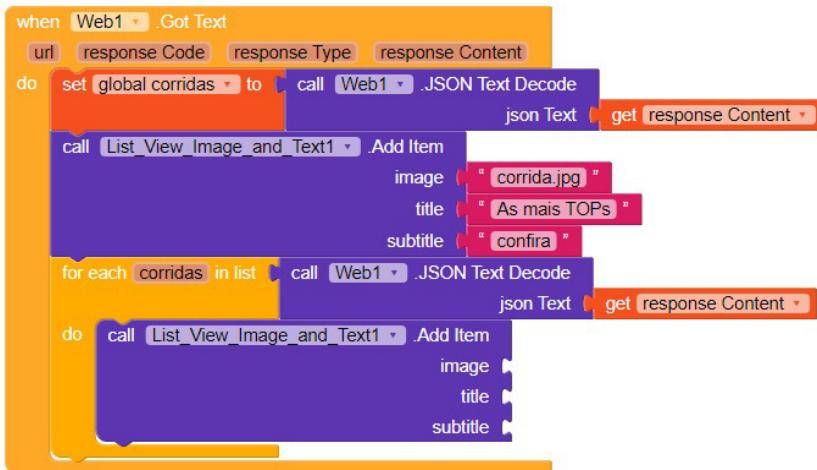


Figura 9.25: Bloco de comandos do evento When Web1.Got Text

E vamos adicionar os campos desejados nas opções da lista. Como já selecionamos uma corrida através do comando `for each`, precisamos agora selecionar os campos para adicioná-los ao `List_View_Image_and_Text1`. O formato que temos de uma corrida decodificada é como demonstra o exemplo:

```
[{"**id**": "7",      "***nome***":      "dados de
cadastro", "***descricao***":      "dados de
cadastro", "***foto***": "foto.jpg"}] .
```

Note que existe um padrão de exibição dos dados: **nome do campo** e o seu **valor** imediatamente exibido a seguir. Precisamos realizar a separação para exibir o valor desejado.

O bloco de comando que realiza a procura do campo e retorna o seu valor, é o `look up in pairs`, que está disponível na guia `Built-in` na seção `Lists`. A imagem a seguir exibe o bloco `look up in pairs`.



Figura 9.26: Bloco de comandos look up in pairs

O bloco do comando `look up in pairs` deverá ser posicionado na opção `image`, do bloco do comando `call List_View_Image_and_Text1`, e nele devemos acrescentar um bloco de texto vazio na opção `key`. Essa opção indicará o nome do campo onde será buscada a informação para exibição. Como temos um campo chamado `foto` no banco de dados e a informação que ele armazena é a imagem da corrida, digite no interior do bloco de texto o nome `foto`. Para ser exibida a imagem, na opção `pairs`, selecione a variável `get corridas` do `for each`. Para encontrá-la, basta posicionar o ponteiro do mouse sobre o nome `corridas` do `for each` e selecionar o comando desejado.

Note que há uma opção `notFound`, isto é, não encontrado, entre as opções do `look up in pairs`. Porém, ela não é necessária para o nosso caso, porque seria apenas exibida caso não fosse encontrado o nome do campo procurado, então, vamos apagar o seu conteúdo. A imagem a seguir demonstra a seleção da imagem através do comando `look up in pairs`.

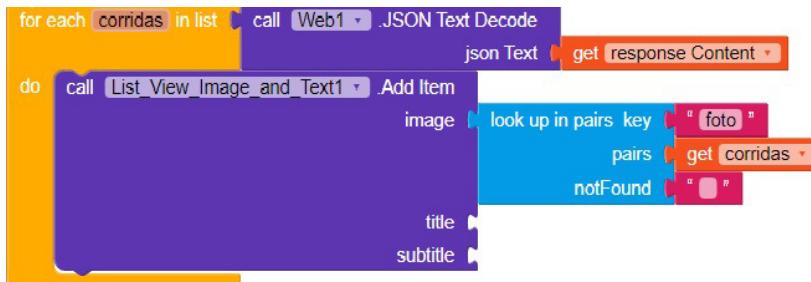


Figura 9.27: Bloco de exibição da imagem da corrida

Para a opção `title`, repita a operação que acabamos de fazer com a imagem, porém o campo que agora será pesquisado é o campo `nome`.

Para a última opção, temos o subtítulo da corrida. Durante o cadastro das corridas, há um limite de 255 caracteres para a sua descrição, porém, aqui na tela de destaque, vamos mostrar apenas os 40 primeiros caracteres do campo descrição. Para selecionar um trecho de texto, precisamos de um comando que indique onde começa o texto a ser selecionado e a quantidade de caracteres que serão exibidos. Para efetuar este trabalho, no `Built-in`, seção `Text`, existe um bloco de comando chamado `segment text`, conforme imagem exibida a seguir. Ela deverá ser conectada a opção `subtitle`, para realizar sua tarefa.

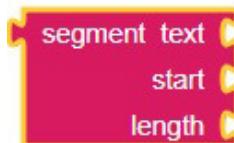


Figura 9.28: Bloco do segmento de texto

Assim como fizemos com a exibição da foto e com o nome da corrida, necessitaremos de mais um bloco de comando `look up in pairs`, configurado com o campo `descricao` para indicar no comando `segment text`.

Nas opções `start` e `length` do bloco de comando `segment text`, vamos indicar onde **inicia** e onde **termina** a seleção do texto para exibição. Como queremos mostrar o início do texto cadastrado, devemos inserir da `Built-in`, seção `Math`, um bloco numérico com valor `1` em seu interior, representando que queremos selecionar o texto a partir do primeiro caractere da descrição da corrida. Como comprimento do texto a ser exibido, inclua mais um bloco numérico e digite o número `40`, que indicará que serão exibidos os primeiros 40 caracteres da descrição da corrida. Veja na imagem a seguir, as configurações de todo bloco de comando para adicionar itens na `List_View_Image_and_Text1`.

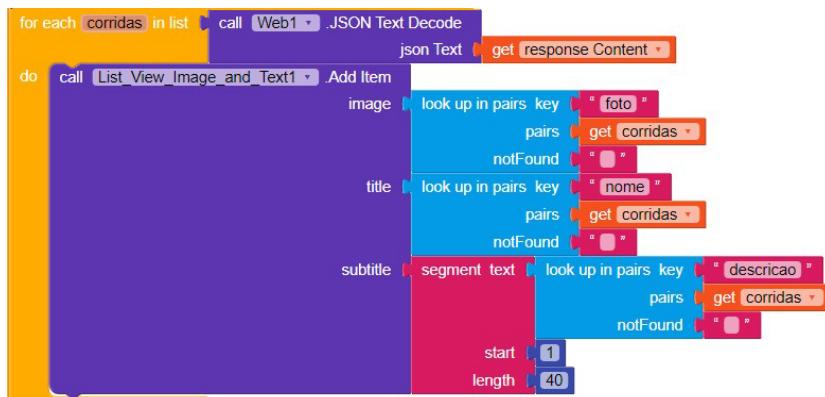


Figura 9.29: Bloco do comando de adicionar itens na `List_View_Image_and_Text`

Após a exibição de todos os dados retornados do banco de

dados pelo componente Web1 , precisamos cancelar a exibição da mensagem de **Buscando Dados** que foi exibido quando a Screen1 foi inicializada. Para isso, acesse a guia de blocos e localize o objeto **notifier1** e na sequência identifique o procedimento **call Notifier1.Dismiss Progress Dialog** e insira-o logo abaixo do bloco de comando **For each corridas** , pois assim, quando terminar a exibição dos dados a mensagem deverá desaparecer. A imagem a seguir exibe o bloco **When Web1.Got Text** finalizado.

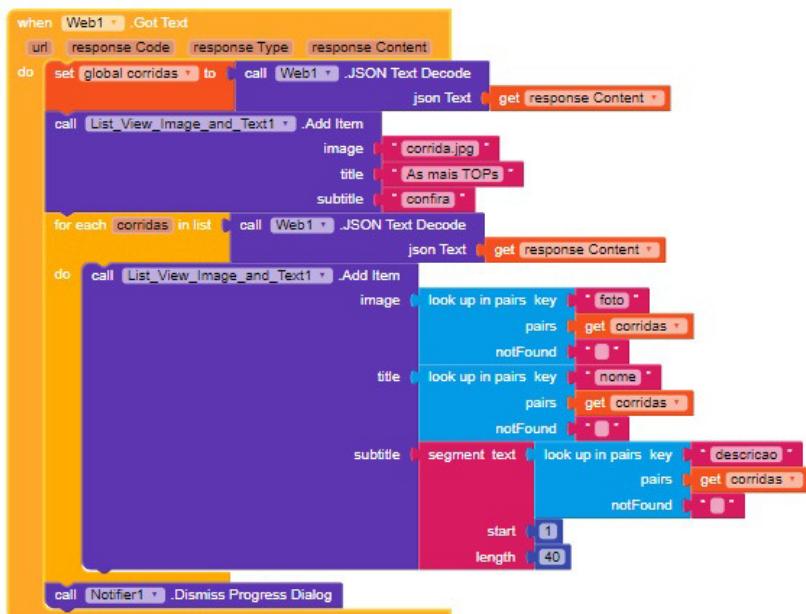


Figura 9.30: Bloco When Web1.Got Text finalizado

## 9.5 SELECIONANDO UMA CORRIDA EM DESTAQUE

Como vimos na descrição do projeto no início deste livro, quando o usuário clicar em uma das corridas listadas como destaque, será exibida uma tela com todos os detalhes desta corrida e suas avaliações realizadas pelos demais usuários. Para que isso aconteça, precisamos identificar em qual corrida o usuário clicou. Existe um evento que identifica o clique do usuário na `List_View_Image_and_Text1`. É ao evento `click` que estamos nos referindo e vamos programá-lo a partir de agora.

Localize na guia de blocos o objeto `List_View_Image_and_Text1` e na sequência identifique o bloco do evento `click` e insira-o na área `Viewer`. A figura a seguir exibe o evento `click`.

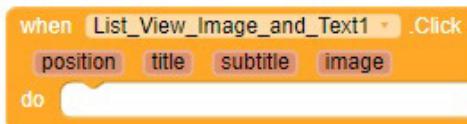


Figura 9.31: Bloco do evento click do objeto `List_View_Image_and_Text1`

Como a primeira linha da `List_View_Image_and_Text1` é utilizada para o título e não contém nenhuma das corridas em destaque, quando essa linha for clicada não deverá realizar nenhuma tarefa. É isso que vamos verificar logo no início da programação do evento `click` da `List_View_Image_and_Text1`.

Um objeto `List_View_Image_and_Text` sempre enumera suas linhas de registro a partir da linha de número 0, sendo assim

teremos que verificar se o usuário clicou na linha 0. Basta verificar se o número da linha que foi clicado é maior que 1 e caso seja deverá realizar as tarefas de abrir a exibição da corrida selecionada.

Para realizar essa verificação, utilizaremos um bloco de comando de decisão IF que se encontra na guia Built-in na seção de control . Insira um bloco de decisão IF conforme demonstra a figura a seguir.



Figura 9.32: Bloco do comando IF

Como o comando IF realiza uma comparação, precisamos de um bloco matemático que faça a comparação entre dois números - o número da linha clicada e o número 1. Acesse a guia Built-in na seção Math e insira um bloco de comparação com o sinal de igual ( = ). Encaixe o bloco de comparação logo à frente do bloco IF . Altere no próprio bloco que está encaixado no IF, o sinal de = para o símbolo de maior e/ou igual , conforme demonstra a imagem a seguir.

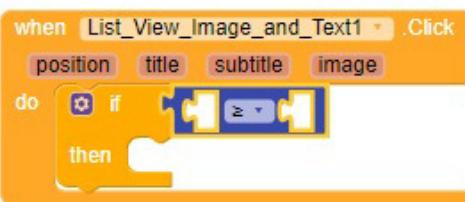


Figura 9.33: Bloco do comando IF com o comando de comparação matemática

Como queremos comparar a posição da linha que foi clicada pelo usuário, posicione o mouse sobre a opção `position`, para que sejam exibidas as opções de trabalho disponíveis. Vamos utilizar a posição que foi clicada pelo usuário, então selecione a opção `get position`, conforme demonstrada na imagem a seguir, e posicione-a no primeiro espaço logo após ao bloco de comando `IF`.



Figura 9.34: Gerando o bloco de comando que identifica a opção do usuário

Como queremos realizar a exibição dos detalhes da corrida que estão após a linha 0, devemos inserir o número 1 no espaço final do bloco `IF`. Para isso, na guia `Built-In` e seção `Math`, selecione o primeiro bloco numérico com o valor 0, e posicione no final do bloco do comando `IF`. Após o encaixe, altere o valor de seu interior, digitando sobre o número 0 o número 1. Veja na imagem a seguir como está até o momento o desenvolvimento da `List_View_Image_and_Text1.Click`.



Figura 9.35: Comando de decisão IF configurada

Se a comparação que acabamos de realizar indicar que a posição clicada for maior ou igual a 1, devemos exibir a tela **Detalhes** e, juntamente com a abertura dessa tela, devemos informar o número da corrida para exibição de seus detalhes. Para realizar a abertura de nova tela, devemos incluir um comando que realize esta tarefa.

Vá a guia **Built-in** na seção **Control** e selecione o comando responsável pelo que desejamos realizar - **open another screen with start value screenName**. Posicione este bloco de comando logo na frente do comando **then** do bloco **IF**, pois a abertura só será executada quando a comparação da posição clicada for maior ou igual a 1.

Precisamos ainda indicar o nome da tela que deveremos abrir. Para isso, encaixe um bloco de texto vazio na frente da opção **screenName** e digite o nome da Screen que será aberta - **Detalhes**. A figura a seguir mostra o atual estágio do bloco que estamos desenvolvendo.

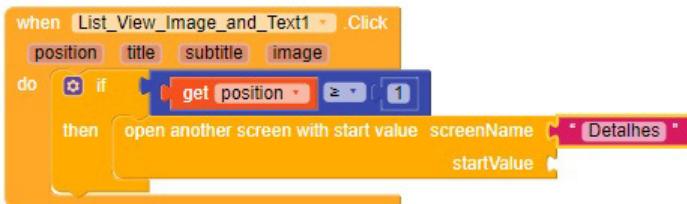


Figura 9.36: Comando de abertura de outra screen

Quando a tela de detalhes for exibida, devemos enviar para ela o valor que está armazenado no campo **ID** da corrida, para que esta seja exibida por completo na nova tela, quando for inicializada.

Já que recebemos anteriormente todas as informações das corridas e as decodificamos do formato JSON e armazenamos na variável `global corridas`, é nela que vamos buscar o seu ID.

Acrescente um bloco do comando `look up pairs` na opção `start value`, pois é ela quem envia uma informação para a tela `Detalhes`. Na opção de localizar a chave para a busca (`key`), insira um bloco de texto vazio e digite o campo que desejamos localizar `ID`.

Para localizarmos o campo `ID` da corrida na qual o usuário clicou, necessitamos identificar o número da linha que foi clicado e buscá-la na lista de corridas que se encontra na variável `global corridas`. Isso é feito adicionando um bloco de comando `select list item`, presente na seção `list` da guia `Built-in` e posicionando-a na opção `pairs`. Como queremos buscar na lista de corridas, precisamos utilizar o bloco `get global corridas` para indicar o valor da lista. Para a opção `index`, selecione o comando `get position` presente na `List_View_Image_and_Text1`.

A próxima imagem exibe o bloco final da seleção de uma corrida em destaque.

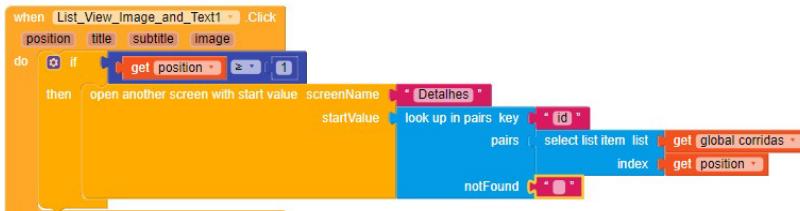


Figura 9.37: Bloco final da seleção de corrida em destaque

## 9.6 O BOTÃO FLUTUANTE

Durante a programação do leiaute, nós inserimos um botão flutuante que ficou posicionado na seção dos objetos invisíveis. Sua função será a de abrir a tela Principal com todas as corridas disponíveis para análise. Agora necessitamos programar o evento click do objeto `Float_Action_Button1`.

Na guia de blocos, localize o objeto `Float_Action_Button1` e na sequência identifique o evento When `Float_Action_Button1.click` e insira-o na área `Viewer`.

Resta-nos agora inserir um comando que realize a abertura de uma outra tela que já está com o seu leiaute pronto, a tela `Principal`.

O comando para abertura de outra tela encontra-se na guia de blocos, na seção `Built-in`. Estando na `Built-in`, selecione a seção `control`, localize o comando `open another screen` `screenName` e arraste-o para dentro do evento `click` do botão `Float_Action_Button1`.

Para finalizar este evento, vamos informar qual a tela gostaríamos de exibir. Inclua um bloco de texto vazio após o comando de abertura da tela e digite o nome da tela que desejamos exibir - `Principal`.

Digite o nome exatamente igual ao da tela que foi criada, respeitando as letras maiúsculas e minúsculas, pois se alguma das letras estiver invertida, sua tela não será exibida.

Veja na figura a seguir o botão flutuante programado.

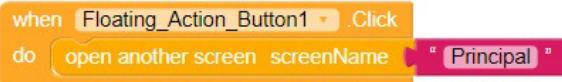


Figura 9.38: Bloco do botão flutuante programado

Uma situação que encontraremos quando o usuário retornar de uma outra tela para a Screen1 , é que o objeto que realiza a mudança das imagens, o View\_Flipper1 , estará desativado e as imagens não serão mais trocadas. Como não queremos que a imagem fique fixa, vamos ativá-lo novamente.

Na guia de blocos, selecione o evento BackPressed da sua Screen1 e insira o evento na área Viewer . Em seu interior posicione o procedimento que inicializa os movimentos de troca das imagens - call View\_Flipper1.StartFlipping . A imagem a seguir mostra o evento finalizado.



Figura 9.39: Programando o botão voltar do dispositivo

Finalizamos aqui a programação da Screen1 , que exibe as corridas mais bem avaliadas pelos usuários do app. Você já poderá neste momento instalar o app no seu dispositivo Android para realizar a visualização da implementação realizada.

Caso deseje, baixe o app desenvolvido até o momento no link [http://nelfabbri.com/avalia\\_run](http://nelfabbri.com/avalia_run) e realize o upload na plataforma do Kodular.

No próximo capítulo, veremos a programação em blocos da segunda tela do app de avaliação de corridas que exibirá todas as corridas cadastradas para que o usuário possa selecionar a que desejar e ver todas as suas informações.

## CAPÍTULO 10

# PROGRAMANDO A TELA PRINCIPAL

Neste capítulo, vamos aprender a programar a tela Principal do App de Avaliação de Corridas de Rua, onde estarão listadas todas as corridas disponíveis para verificação e análise do usuário. Esta tela será acessada quando o usuário clicar no botão flutuante da Screen1.

Ao ser exibida, o evento de inicialização já disparará uma consulta ao arquivo em PHP para solicitar os dados das corridas cadastradas no banco de dados e então serão exibidas em um objeto **Cardview**.

Mas, para começarmos, necessitamos acessar a Screen **Principal**. Faça conforme visto em capítulo anterior.

Visualizando a Screen Principal, acesse a área de blocos e insira o evento de inicialização que realizará a solicitação dos dados, assim que a Screen Principal for exibida. Estamos nos referindo ao bloco de comando `Principal.Initialize`, exibido na imagem a seguir, que o leitor deverá inserir em sua área `Viewer`.

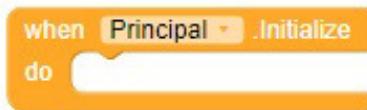


Figura 10.1: Bloco de comando de inicialização da Screen Principal

No capítulo anterior, vimos muitos procedimentos que serão repetidos deste ponto em diante. Para não ficar redundante, daremos ênfase aos comandos que ainda não são conhecidos.

Exibiremos neste evento uma caixa de diálogo, informando que os dados estão sendo processados e que o usuário deverá aguardar. Insira um `call Notifier1.Show Progress Dialog` dentro do evento `Initialize` da Screen Principal e insira a mensagem utilizando um bloco de texto, digitando em seu interior `Buscando Dados`. No espaço reservado ao título da caixa de texto, insira um bloco de texto vazio. A imagem a seguir mostra o atual estágio do evento `When Principal.Initialize`.



Figura 10.2: Evento When Principal.Initialize

O arquivo em PHP, que seleciona todas as corridas para exibição chama-se `query.php` e está detalhado no capítulo final deste livro.

Precisamos indicar no objeto `Web1` o endereço deste arquivo. Insira um comando de configuração do endereço `Set Web1.URL`

to e acrescente um bloco de texto para digitar o local do arquivo em seu interior:

[http://www.nelfabbri.com/avalia\\_run/query.php](http://www.nelfabbri.com/avalia_run/query.php)

Após a configuração da URL do arquivo query.php seja executado no provedor e retorne os valores para exibição. Insira o bloco de comando que realiza essa chamada - call Web1.get . A imagem a seguir mostra o bloco de inicialização da Screen Principal finalizado.



Figura 10.3: Bloco de inicialização da Screen Principal

É necessário criar uma variável de nome corridas do tipo lista, para armazenar a lista de corridas resultantes da consulta do banco de dados. Após a criação da variável veja o resultado:

initialize global [corridas v] to [create empty list]

Figura 10.4: Criação da variável corridas

## 10.1 RECEBENDO A LISTA DE CORRIDAS

Após a solicitação dos dados pelo objeto Web1 , devemos tratar as informações recebidas através do evento Web1.Got Text .

É possível que, nas primeiras utilizações de seu app, não haja informações de corridas cadastradas no banco de dados e, para que não ocorram erros durante o processo de exibição dos dados, deveremos verificar se a lista que retornou do arquivo em PHP não está vazia.

Para isso, insira primeiramente um evento `When Web1.Got Text` que trata o recebimento de informações do objeto `Web1` em sua área `Viewer`.

Dentro deste bloco de evento, iniciaremos a verificação para saber se a lista que retornou não está vazia. Para isso, adicione um comando `IF` da seção `Control` na guia `Built-in`.

Para saber se a lista **NÃO** está vazia, precisamos de um bloco de comando lógico que faça essa negação. Novamente, acesse a `Built-in`, acesse a seção `Logic` e selecione o comando lógico de negação `not`, encaixando-o ao bloco `IF`. Na sequência, precisamos de um bloco de lista, que verificará se ela está vazia.

Observe que estamos montando uma sequência de comandos e até agora o comando está informando `Se não tiver uma lista vazia`.

Agora devemos indicar qual a lista que queremos verificar que não está vazia. Como o objeto `Web1` retorna dados codificados no formato JSON teremos que realizar a sua decodificação. Encaixe o comando `call Web1.JSON Text Decode` na frente do comando que verifica se a lista está vazia. Para finalizar o comando de decisão, utilize o bloco de comando (`get response content`) que recebeu a lista de corridas do arquivo em PHP e encaixe na área para a decodificação. Seu bloco de comandos da decisão

deverá estar igual à figura a seguir.

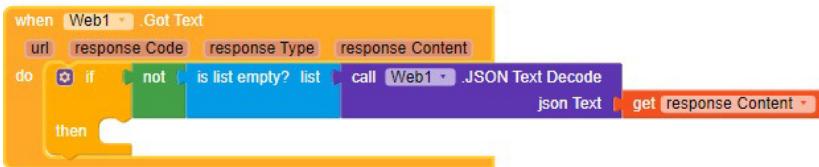


Figura 10.5: Verificação se a lista não está vazia

Se a lista retornada **não estiver vazia**, precisamos armazená-la decodificada na variável `set global corridas`. A figura a seguir demonstra o processo já visto anteriormente.

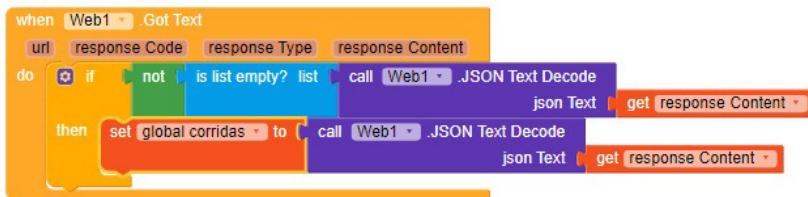


Figura 10.6: Armazenando a lista decodificada

## 10.2 DECLARANDO VARIÁVEIS LOCAIS

Variáveis locais são identificadas pelo app somente no evento em que estão sendo declaradas. Com isso, diminuímos um pouco o consumo da memória do dispositivo, já que, depois de utilizar o evento específico, as variáveis locais não estarão mais reservadas, sendo assim, não ocuparão espaço na memória.

Para declarar variáveis locais, acesse na guia `Built-in` a mesma seção de `Variables` e selecione a inicialização da variável local. Veja a imagem deste bloco a seguir.



Figura 10.7: Declarando variável local

Insira esse bloco, logo abaixo do comando `set global corridas to .`

Defina o nome da variável local para `imagens` do tipo lista vazia, pois armazenará uma lista com todas as imagens das corridas cadastradas. A imagem a seguir, demonstra a criação da variável local `imagens`.



Figura 10.8: Declarando variável local do tipo lista

Como será necessário exibir o nome, local e a descrição da corrida, teremos que criar mais 3 variáveis locais para armazenamento das listas das respectivas informações.

Para declararmos mais variáveis locais, basta clicar no símbolo de configuração, conforme demonstra a imagem a seguir, para exibir uma janela com mais opções.



Figura 10.9: Declarando outras variáveis locais

Arraste a opção `name` que está à esquerda para dentro de `local names`, que está à direita nessa janela de configuração. A figura a seguir demonstra o processo de criação de mais variáveis locais.



Figura 10.10: Declarando outras variáveis locais

Necessitamos ao todo de 4 variáveis locais. Repita o procedimento mais duas vezes. Nomeie as variáveis para `- titulos`, `subtitulos` e `conteudos`. Todos os tipos das variáveis deverão ser uma lista vazia, pois como já dito, elas vão armazenar várias informações das corridas. A imagem a seguir mostra a definição das variáveis locais finalizadas.



Figura 10.11: Variáveis locais finalizadas

O procedimento que fizemos na Screen1 para a listagem das corridas deverá ocorrer aqui também, com a diferença de que os dados serão exibidos no objeto `CardView1`.

Vamos lá, inclua um bloco de comando `for each in list` dentro dos blocos das variáveis locais e adicione uma decodificação do texto JSON recebido pelo objeto `Web1`, alterando o nome do `for each` para `corridas`. A figura a seguir demonstra o atual momento do desenvolvimento do evento `Web1.Got Text`.

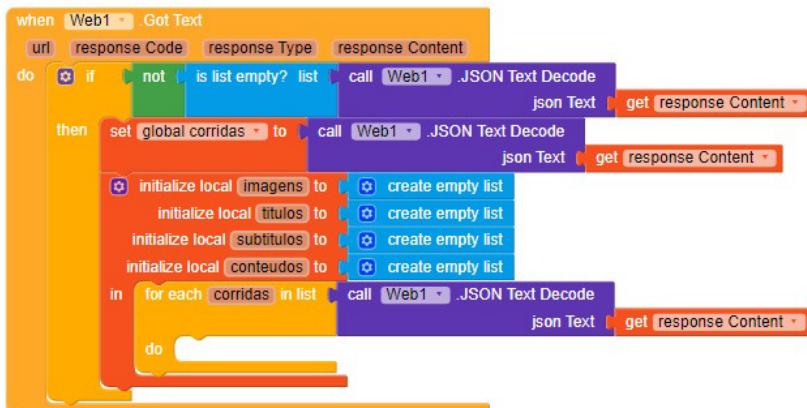


Figura 10.12: Evento `Web1.Got Text`

Como vimos, o comando `for each` percorrerá toda a lista

carregada com as informações das corridas. A cada item ou registro lido da lista, deveremos separar as informações em listas distintas para sua futura exibição no objeto `CardView1`. Para isso, vamos adicionar da guia `Built-in` e seção `Lists`, o bloco de comando que adiciona um item à lista desejada (`add items to list`).

O primeiro campo que armazenaremos na variável local `imagens` é a foto, e ela será exibida na `CardView1`. Em sua opção `List`, adicione a variável local `get imagens`, que é obtida posicionando o ponteiro do mouse sobre o nome da variável no bloco de sua inicialização.

Já indicamos a variável que armazenará a lista de imagens para exibição na `CardView1`, porém falta receber os dados das fotos que o objeto `Web1` recebeu. Vamos procurar pelo campo `foto` e armazenar seu conteúdo na lista de imagens. Para isso, é utilizado o bloco de comando `look up in pairs key`. Adicione um bloco de texto na posição da `key` para a procura e digite o nome do campo `foto`. Pegue também a variável de controle do `for each` `corridas` e encaixe na opção `pairs`. No caso de não ser encontrada a informação, devemos deixar em branco a opção `notFound`, para que não ocorram erros durante a execução do app. A figura a seguir mostra a programação para adicionar as fotos das corridas na lista `imagens`.

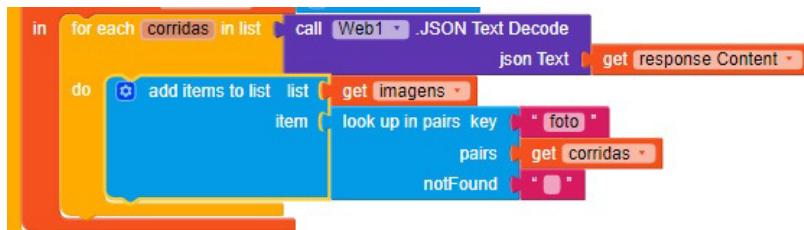


Figura 10.13: Exibindo as fotos na lista de imagens

Da mesma maneira que fizemos com a lista de imagens, precisamos realizar a separação das demais informações - `titulo`, `subtitulo` e `conteudo`.

Realize essas programações adicionando para cada uma das variáveis locais um comando de `add items to list`, lembrando que os campos que retornaram do banco de dados e deverão ser utilizados na opção `Key` do comando `look up in pairs` são: `nome`, `endereco` e `descricao`, respectivamente.

Para a descrição, devemos limitar o tamanho do texto para 50 caracteres, pois todas as informações complementares serão exibidas na Screen Detalhes. Para isso, utilize o bloco de texto `segment text`. A imagem a seguir exibe o bloco `for each` finalizado.

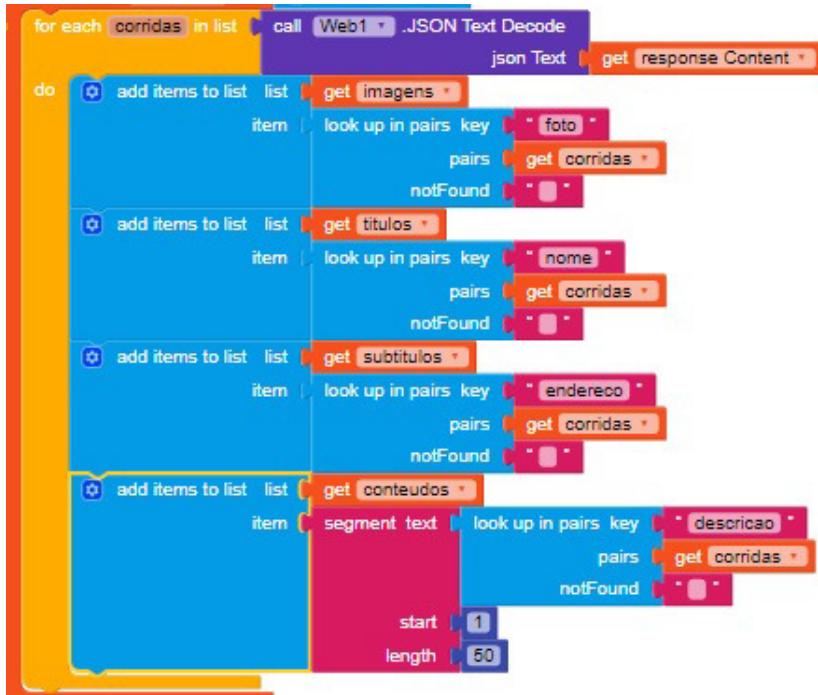


Figura 10.14: Bloco for each finalizado

Com todas as informações separadas em suas listas específicas, devemos agora solicitar a exibição no componente `CardView`. Primeiramente, na guia de blocos, localize o objeto `Cardview1` e então selecione o bloco que representa o procedimento `call CardView1.Type1`. O procedimento `Type1` é o que nos fornece o maior número de opções para trabalhar com a `CardView1`.

Vamos analisar as opções disponíveis na `CardView1`: na opção `input`, inserimos o objeto onde será exibida a `CardView`. Neste caso, precisamos selecionar o `VAS_Cards` na guia de blocos e posicioná-lo na opção `input`.

Nas demais opções, `images`, `titles`, `subTitles` e `content`, devemos inserir as variáveis locais com os nomes `get imagens`, `get titulos`, `get subtitulos` e `get conteudos`.

Para a opção `buttons`, vamos criar uma lista com um botão de compartilhamento para cada corrida. Insira na opção `buttons` um bloco `make a list` da guia `Built-in` e seção `Lists` e na sequência insira um bloco de texto digitando em seu interior o nome do botão que deverá aparecer abaixo da exibição das corridas. Neste caso, digite a palavra `share`. Veja na figura a seguir a programação completa do objeto `CardView1`.



Figura 10.15: Bloco de comandos da CardView

Para obter outros nomes de ícones para exibição nos botões, acesse o site <https://material.io/tools/icons>.

Após a exibição da `CardView1`, necessitamos encerrar a visualização do aviso ao usuário dos dados que estavam sendo buscados. Para isso, insira um `Call` `notifier1.dismiss Progress Dialog`, logo abaixo do bloco de decisão `if`. Veja na imagem o bloco completo do evento `Got Text` do objeto `Web1`.

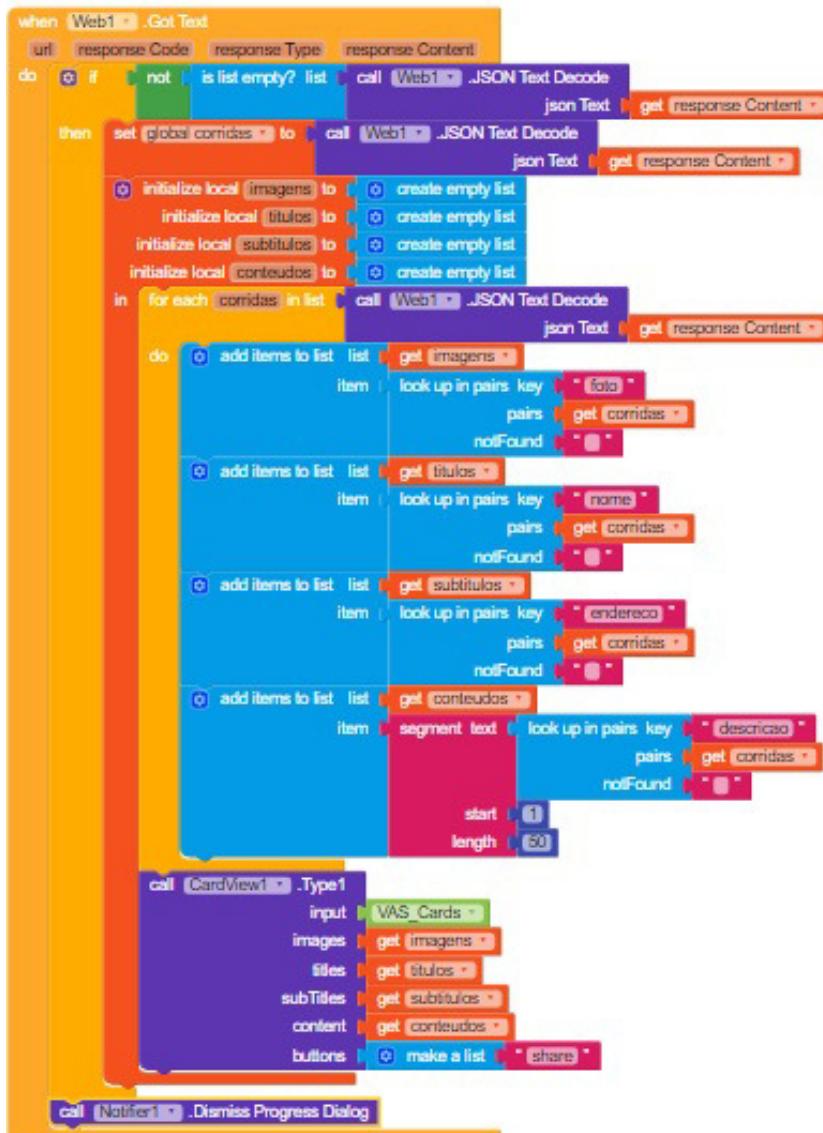


Figura 10.16: Bloco de comandos do evento GotText finalizado

## 10.3 SELECIONANDO UMA OPÇÃO DO CARDVIEW

Após a exibição de todas as corridas no objeto `CardView1`, o usuário terá a possibilidade de conhecer mais detalhes da corrida, tais como a nota de avaliação, comentários de outros usuários, visualização do local no mapa e postar sua avaliação e comentários. Para que isso ocorra, o usuário deverá clicar sobre a corrida sobre a qual desejar obter maiores informações. Nesse momento será exibida a Screen Detalhes com a corrida já selecionada.

Para que isso aconteça, precisamos programar o evento `AfterPicking` para abrir a nova Screen. Insira um evento `When CardView1.AfterPicking` na área `Viewer`. Insira o bloco do comando `open another screen start value` em seu interior para exibir nova Screen. Em `screenName` insira um bloco de texto e digite o nome da tela que deverá ser exibida - `Detalhes`. A imagem a seguir, exibe o evento `AlterPicking` do `CardView1`.



Figura 10.17: Bloco de comandos do evento GotText finalizado

Como dissemos, precisamos exibir a tela `Detalhes` e enviar o código de identificação (ID) da corrida selecionada. Para isso, o processo é idêntico ao realizado no capítulo anterior, quando clicamos em uma corrida em destaque. Vamos revê-lo? Insira o bloco `Look up in pairs` para procurarmos pela chave `ID`. Devemos procurar o `ID` em uma lista de corridas que estão

armazenadas na variável `get global corridas`. O índice que queremos selecionar é a posição da linha que foi clicada no `CardView1`. Para obter o índice selecionado utilize a variável `get position` do `CardView1.After Picking` e a posicione na opção `index`. Veja na figura a seguir o desenvolvimento do bloco para selecionar o `ID` da corrida desejada.



Figura 10.18: Bloco de seleção do ID da corrida

A imagem a seguir, mostra o bloco `When CardView1.AfterPicking` finalizado.

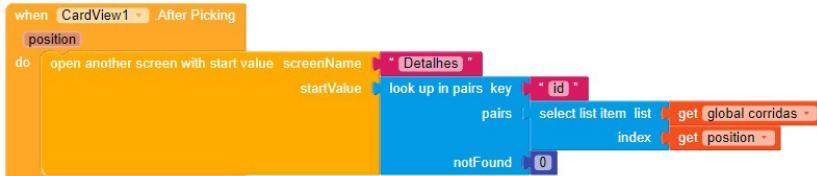


Figura 10.19: Bloco When CardView1.AfterPicking finalizado

## 10.4 COMPARTILHANDO UMA INFORMAÇÃO

Como vimos durante o processo de exibição dos dados pelo objeto `CardView1`, foi inserido um botão de compartilhamento de informações da corrida. Precisamos programar o evento `click` do botão da `CardView1` para identificar os dados da corrida a serem compartilhados. Insira o evento `When`

`CardView1.Button Click` em sua área `Viewer` para ativarmos a função desejada.

Durante o processo de criação do leiaute da Screen Principal, inserimos um objeto invisível chamado de `Sharing`. Ele é o objeto responsável por ativar o compartilhamento de informações utilizando seu dispositivo. Inclua a chamada do procedimento `call Sharing1.Share Message` no interior do evento `Click` do `Cardview1`, conforme demonstra a imagem a seguir.

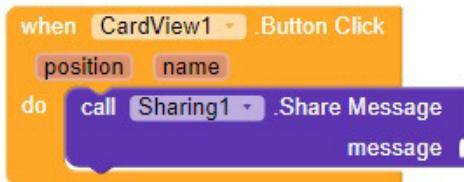


Figura 10.20: Evento Click do CardView1

A mensagem que vamos compartilhar terá a aparência da exibida a seguir. Na imagem foi utilizado o aplicativo WhatsApp para o envio, mas poderia utilizar qualquer outra ferramenta de compartilhamento instalada em seu dispositivo Android.

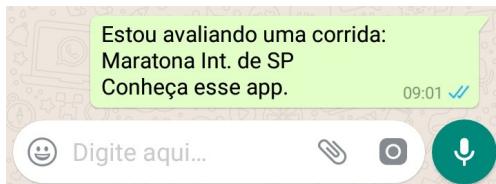


Figura 10.21: Texto de compartilhamento

A opção `Message` do procedimento `Sharing` possui apenas uma entrada para o texto, e como estamos enviando mais do que

uma linha de informação precisamos utilizar um bloco de texto que faça a união de vários textos. O bloco de comando que realiza essa tarefa é o `join` da guia `Built-in` da seção `Text`. Por padrão, o bloco de junção `join` possibilita apenas duas entradas de textos, mas como precisamos de mais, clique no ícone de configuração do `join` e insira mais duas entradas. A imagem a seguir exibe o bloco de texto `join` com 4 entradas.

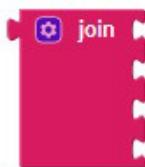


Figura 10.22: Bloco de comando Join com 4 entradas

Para a primeira entrada do bloco `join`, vamos colocar a frase dentro de um bloco de texto - `Estou avaliando uma corrida`  
`\n`.

Observe que inserimos os caracteres `\n` ao término da frase. Sua função é realizar uma quebra de linha, isto é, ao encontrar o comando `\n` o próximo texto será exibido na linha debaixo.

No segundo espaço de entrada do bloco `join` devemos identificar a corrida na qual clicamos para realizar o compartilhamento de seu nome. Para isso, precisamos utilizar o bloco de comando já visto `look up in pairs`. Vamos procurar pelo campo `nome` na lista `get global corridas` pelo índice `get position` do evento `When CardView1.Button Click`.

No terceiro espaço de entrada do bloco `join`, incluiremos mais um bloco de texto com os caracteres de quebra de linha `\n`.

No quarto e último espaço, inclua um bloco de texto, convidando a pessoa para conhecer o aplicativo desenvolvido. Digite em um bloco de texto a frase `Conheça esse aplicativo`.

A imagem a seguir exibe a programação do evento clique no botão de compartilhamento da `CardView1`.

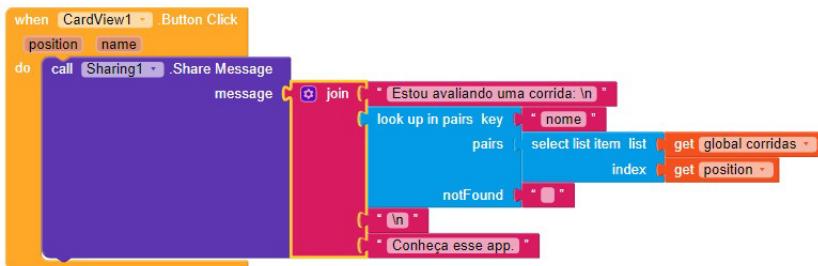


Figura 10.23: Bloco do botão de compartilhamento da `CardView1`

## 10.5 FECHANDO UMA SCREEN

O botão de voltar que está visível no menu superior da Screen Principal nada mais é do que um comando para fechar a Screen atual após o botão ser clicado. Para isso, insira um evento `Click` do `Button1` em sua área `Viewer`.

Em seu interior, necessitamos inserir o comando `close screen` que está disponível na guia `Built-in` na seção `Control`. A figura a seguir demonstra o botão de voltar programado.

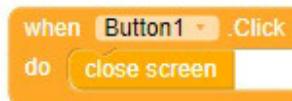


Figura 10.24: Bloco do botão retornar

## 10.6 FILTRANDO OS DADOS PARA EXIBIÇÃO

Como o cadastro de corridas para avaliação pode se tornar muito amplo em quantidade de registros, decidimos criar um botão para realizar uma filtragem para exibição dos nomes das corridas em ordem alfabética crescente ou decrescente.

O botão `Button2` será o responsável pela ativação deste recurso, para tanto, insira um bloco do comando `When Button2.Click`. No seu interior, necessitamos de um bloco de comando que forneça as opções de filtro para o usuário. O comando que realiza a exibição da solicitação de digitação para o usuário do app é o `call Notifier1.Show Choose Dialog`, que se encontra na seção `Built-in` na guia do objeto `Notifier1`.

Ao inserir o objeto na área de `Viewer`, observe que ele possui algumas entradas que vamos programar. Para as opções `message`, `title`, `button1 Text` e `button2 Text` inclua um bloco de texto vazio. Nela vamos preparar a exibição da mensagem que queremos que seja exibida ao nosso usuário. No espaço reservado, a opção `message`, digite a frase `Exibir as corridas em ordem alfabética`. Para a opção `title` não digitaremos nada. Na opção `button1 Text`, digitaremos a primeira opção do filtro que será exibida ao usuário. Insira a palavra `Crescente`, pois quando o usuário clicar nessa opção, os registros da corrida serão exibidos em ordem crescente. Para a opção do `button2 Text`, vamos digitar a palavra `Decrescente`, para que as corridas sejam exibidas em ordem decrescente. Nessa tela de mensagem, existe uma última opção que permite exibir o botão `Cancelar` a operação. Para nosso exemplo, vamos desativar a sua exibição, bastando alterar na opção `cancelable` de `true` para `false`. A

imagem a seguir exibe o botão de filtragem de corridas programado.

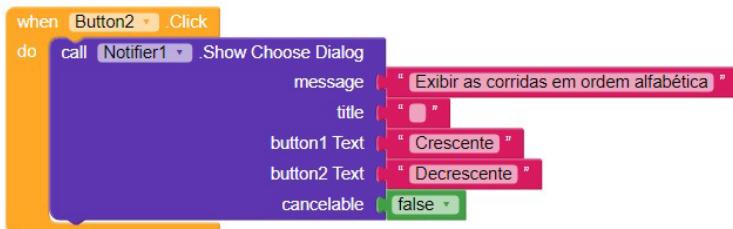


Figura 10.25: Bloco da seleção do filtro da listagem

Precisaremos de um evento que verifique o que o usuário selecionou. O evento responsável por essa verificação é o `When Notifier After Choosing` que deverá ser inserido na área do Viewer. A imagem a seguir exibe o bloco inserido.



Figura 10.26: Bloco da seleção do filtro da listagem

Como temos duas opções de resposta, necessitaremos realizar uma decisão utilizando o bloco de comando `if`. Configure esse bloco para que possua a opção `else`. A imagem a seguir exibe o bloco do evento `After Choosing` com o comando `if` configurado.



Figura 10.27: Bloco do comando if

O comando `if` deverá verificar se o usuário clicou em crescente. A programação para essa verificação deverá conter primeiramente um bloco `compare texts` da seção `Text` logo após o bloco de comando `if` na sua área `Viewer`. Não se esqueça de alterar o sinal do bloco do comando `compare texts` para o símbolo `=`. No primeiro espaço do bloco que compara os textos, devemos inserir a variável `get choice`, disponível no próprio bloco do `Notifier1.After Choosing` em sua área de `Viewer`. A variável `get choice` armazena a opção de escolha do usuário.

Para o segundo espaço do comando `compare texts`, devemos inserir um bloco de texto e em seu interior digitar o texto que desejamos comparar. Digite no espaço reservado a palavra `Crescente`. A imagem a seguir, demonstra a programação do comando `if` comparando a escolha do usuário.



Figura 10.28: Programação do comando if para o filtro.

No caso da pergunta que acabamos de realizar na programação com o bloco `if`, devemos enviar uma informação para o arquivo em PHP, sobre qual consulta desejamos realizar. Aqui definimos que, ao enviarmos uma variável chamada `tipo` com o valor igual a `1`, desejamos que a consulta seja realizada de forma alfabética crescente e se o valor for igual a `2` indicará que a consulta será em ordem decrescente. Portanto, se a resposta do bloco `if` for verdadeira, devemos adicionar um bloco do comando `call Web1.Post Text` e na propriedade `text` inserir um bloco de texto com o valor `tipo=1` que será enviado para o arquivo em PHP. A imagem a seguir demonstra o bloco `call Web1.Post Text`.



Figura 10.29: Comando `call Web1.Post Text` com valor passado ao arquivo PHP

Para a opção que exibe as corridas em ordem decrescente, necessitaremos inserir outro comando `call Web1.Post Text` em frente à opção `else` do bloco `if` e configurar a sua entrada de texto com a informação `tipo=2`. A imagem a seguir exibe o bloco do comando que envia o tipo de filtro que desejamos exibir.

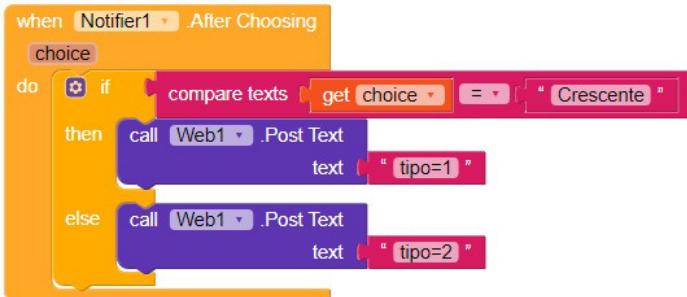


Figura 10.30: Comando call Web1.Post Text finalizado

Após o envio da solicitação dos dados pelo objeto Web1 ao servidor, as informações de retorno também serão tratadas pelo bloco Web1.Got Text já programado anteriormente neste capítulo e não necessitará de nenhuma alteração, pois quem realizará o filtro será o arquivo em PHP.

## Concluindo

Neste capítulo, aprendemos as técnicas para adicionar valores em uma CardView , selecionar os dados retornados do objeto Web , realizar um filtro dos dados, identificar o clique em uma corrida específica e compartilhar informações para seus contatos.

No próximo capítulo vamos aprender a criar a programação da tela de login e cadastro, pois somente se o usuário estiver logado, poderá postar comentários e avaliar as corridas.

## CAPÍTULO 11

# PROGRAMANDO O LOGIN E O CADASTRO

Como foi definido anteriormente no capítulo do projeto que iríamos desenvolver, o usuário só poderá comentar e avaliar uma corrida se já tiver realizado um cadastro e estiver logado no aplicativo.

Neste capítulo, vamos desenvolver a lógica e a programação dos blocos de comandos para realizarmos essas duas tarefas.

Primeiramente, é necessário selecionar a Screen LoginCadastro e, na sequência, acessar a área de programação dos blocos de comando.

## 11.1 ALTERNANDO A VISIBILIDADE DAS FUNÇÕES

Durante o desenvolvimento do leiaute, tínhamos criado dois objetos Vertical Arrangement : um para realizar o login e outro para o cadastro de usuário, sendo que apenas um deverá ficar visível. Iniciaremos a programação para exibir primeiro o Vertical Arrangement referente ao Login, portanto vamos programar a inicialização da tela de LoginCadastro . Para tanto,

insira o evento `Initialize` da Screen `LoginCadastro` em sua área `Viewer` e em seu interior adicionaremos os controles que realizarão a visibilidade dos objetos na ordem desejada.

A `Vertical Arrangement` que ficará visível para o usuário será a `VA_Login`. Portanto insira o bloco `VA_Login.Visible` to e adicione um bloco lógico `true`. Já para a `VA_Cadastro`, deixe a opção `visible` marcada com o bloco de comando lógico `false`, pois ela somente será exibida quando o usuário clicar no botão cadastrar. A imagem a seguir exibe a programação do bloco de inicialização da tela `LoginCadastro`.

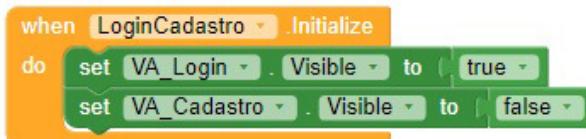


Figura 11.1: Bloco de inicialização da tela `LoginCadastro`.

Quando o usuário clicar no botão `Btn_Cadastrar`, devemos realizar o procedimento inverso ao demonstrado anteriormente, ou seja, o `VA_Cadastro` ficará marcado com a visibilidade verdadeira e o `VA_Login` será marcado como falso. A figura a seguir exibe o evento `Btn_Cadastrar.Click` programado.



Figura 11.2: Evento Click do `Btn_Cadastrar`.

Na `Vertical Arrangement` do Cadastro, existe um botão

que retornará para o `VA_Login`. Nele também devemos trabalhar com a alternância da visibilidade dos componentes. Após inserir o evento `click` do botão `Btn_Voltar_Login`, deixaremos o `VA_Login` visível e tornaremos o `VA_Cadastro` invisível. A imagem a seguir demonstra o resultado da programação do `Btn_Voltar_Login`.



Figura 11.3: Programação do `Btn_Voltar_Login`

## 11.2 REALIZANDO O LOGIN

Para a verificação da existência do cadastro do usuário, precisamos que sejam digitados o login e a sua senha, para depois, ao clicar no botão `Btn_Login`, seja feita a verificação junto ao banco de dados online; se as informações estiverem corretas o usuário poderá comentar e avaliar as corridas.

Primeiramente, devemos inserir o bloco do evento `click` do `Btn_Login` na área `Viewer` e verificar se o usuário realmente digitou as informações nas caixas de texto; caso contrário poderão ocorrer erros de consulta ao banco de dados. Para realizar essas verificações, insira da guia `Built-in` da seção `Control` um bloco de comando de decisão `if`. No bloco de decisão devemos tratar o resultado verdadeiro e falso da condição testada, ou seja, se a decisão for verdadeira, o aplicativo realizará a consulta dos dados, mas se não for, devemos informar ao usuário que os dados devem ser preenchidos.

Após inserir o bloco `if`, vamos configurá-lo para exibir a opção `else`, pois é ela quem executará os comandos se a decisão `for false`. Clicando no ícone de configuração no próprio bloco `if`, arraste a opção `else` para dentro do bloco principal. A imagem a seguir exibe o bloco `if` configurado.



Figura 11.4: Bloco do comando if preparado

Para verificar se o login e a senha não estão vazias, ou seja, sem a digitação de valores, precisamos adicionar no bloco `if` um comando `and` da guia `Built-In` seção `Logic`. A figura a seguir exibe o bloco `if` com a opção `and` configurada.

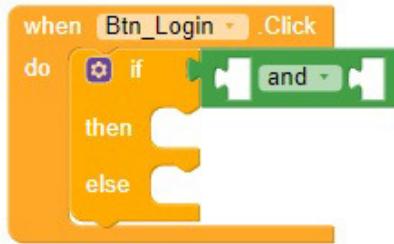


Figura 11.5: Bloco do comando if com and configurado

A programação dos blocos de comandos seguirá a seguinte lógica: insira da seção `Logic` um comando de negação `not`. Da

guia `Text`, adicione um bloco que verifica se o objeto está vazio - `is empty` e na sequência indique o objeto que deverá ser testado - `Text_Box1.Text`. O procedimento de verificação realizado com a senha que está no objeto `Text_Box2.Text` deverá ser o mesmo. Após a sua realização, veja na imagem a seguir os blocos preparados para a decisão.



Figura 11.6: Blocos preparados para inserir na verificação

Esses dois blocos deverão ser posicionados nos espaços que estão reservados dentro do comando `if`. Um antes do bloco `and` e o outro após.

Talvez você não esteja vendo o bloco conforme a figura a seguir. Isso porque, para que toda a imagem possa ser visualizada nesta página, há uma configuração extra que realizamos, mas que não altera em nada a lógica do app.

Para se conseguir essa visualização, clique com o botão direito do mouse sobre o comando `and` e selecione a opção `External Inputs`. Agora seu bloco deverá estar igual ao exibido na figura a seguir.

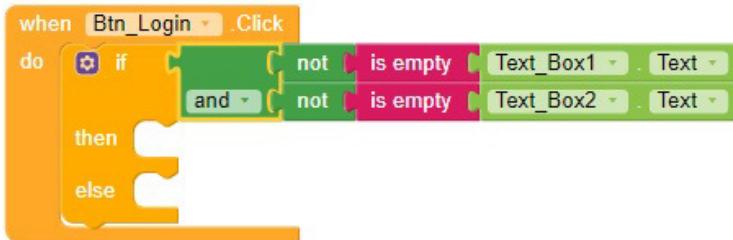


Figura 11.7: Bloco do comando if configurado

Com o login e a senha já digitados pelo usuário, precisamos indicar ao bloco de comando `Set Web_Login.URL to` o endereço do site onde se encontra o arquivo `login.php` - [http://www.nelfabbri.com/avalia\\_run/login.php](http://www.nelfabbri.com/avalia_run/login.php).

**LEMBRETE:** o arquivo `login.php` , está disponível e comentado no último capítulo deste livro.

Logo abaixo do bloco de comando `Set Web_Login.URL to` , vamos preparar as informações do email e da senha para enviar ao arquivo `login.php` . Utilizaremos o método `POST` para enviá-las, então inclua o bloco do processamento `call Web_Login.Post Text` .

O envio dos dados obedecerá a seguinte estrutura e será enviado no formato de uma única linha, conforme exemplo a seguir.

`Nome_da_variável_1=valor_enviado_1&nome_da_variável_2=valor_enviado_2`

Para montarmos essa sequência de informação, precisamos adicionar na opção Text do call Web\_Login.Post Text um bloco de texto join para unir as informações. Configure o bloco join para possuir 4 entradas de texto.

Para a primeira entrada, adicione um bloco de texto e, em seu interior, informe o nome da primeira variável que levará a informação ao arquivo login.php - email= .

Na segunda entrada, devemos adicionar o bloco do objeto que contém o email digitado pelo usuário - Text\_Box1.text .

Na terceira entrada do join , insira um bloco de texto e digite o nome da segunda variável que levará o valor da senha para o arquivo em php - &senha= .

Na última entrada, indique o bloco do objeto que contém a senha digitada pelo usuário, o Text\_Box2.text .

A imagem a seguir exibe o bloco com os comandos que acabamos de descrever.

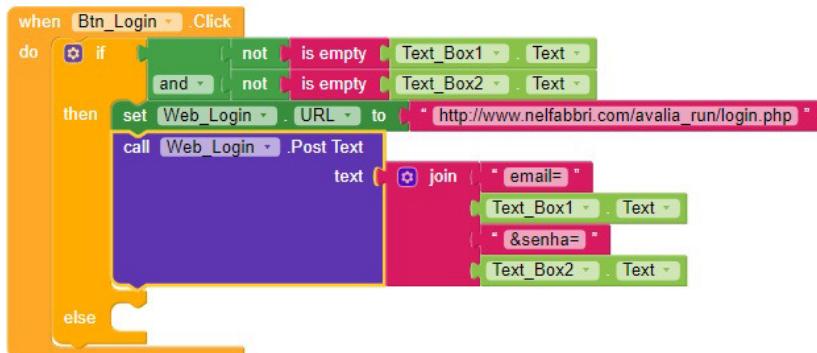


Figura 11.8: Btn\_Login.Click

No caso de o resultado da condição ( if ) ser falsa, necessitamos informar ao usuário do app que todos os dados deverão ser preenchidos para poder realizar o login. Insira o bloco call Notifier1.Show Alert para a exibição da mensagem e digite em uma caixa de texto a mensagem Todos os campos são obrigatórios . A imagem a seguir exibe o bloco do botão Btn\_Login.Click por completo.

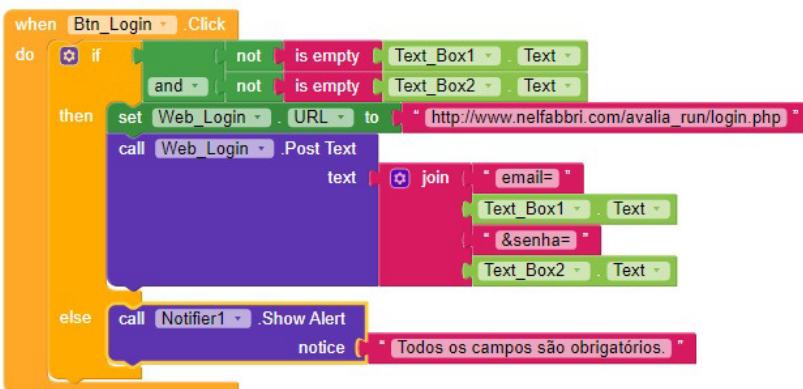


Figura 11.9: Blocos de comandos do botão Btn\_Login.Click finalizado

### 11.3 RECEBENDO A CONFIRMAÇÃO DO LOGIN

Após o envio das informações do email e da senha pelo objeto Web\_Login ao arquivo login.php , este processará as informações no banco de dados e retornará alguma informação para o nosso aplicativo, e é essa informação que vamos analisar para saber se o login ocorreu de maneira satisfatória ou não.

O evento que trata o recebimento da informação do processamento em PHP é o bloco Web\_Login.GotText que

deverá ser inserido em sua área `Viewer` .

Se o PHP retornar a palavra `nada` , é uma indicação de que o login não foi realizado com sucesso e isso deverá ser informado ao usuário. Caso contrário, devemos armazenar uma informação no seu dispositivo para que, nas próximas vezes em que for utilizar este app, não seja necessário realizar novo login.

Agora vamos verificar o recebimento das informações que o arquivo PHP retornou, que ocorre dentro do evento `GotText` do `Web_Login` . Para isso, insira um bloco de comando `if` e configure-o para ter a opção `else` .

Para comparar um texto, precisamos inserir um bloco de comandos especial para essa função. Na guia `Built-in` seção `Text` , localize o `compare texts` e posicione para completar o comando de decisão `if` .

Nos espaços do comando `compare texts` , adicione o bloco com o valor retornado `get response Content` que está disponível no próprio bloco `GotText` . No outro espaço da comparação, insira um bloco de texto e digite a palavra `nada` em seu interior. A palavra `nada` será enviada pelo arquivo em PHP, caso o login esteja errado. A imagem a seguir, exibe o bloco com a comparação dos valores retornados pelo arquivo `login.php` .

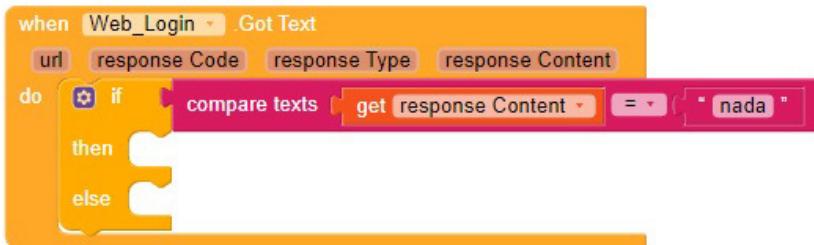


Figura 11.10: Comandos de comparação no evento Got Text

Para exibir a mensagem de que o login não foi bem-sucedido, vamos utilizar o comando `call Notifier1.Show Alert` e incluir em uma caixa de texto a mensagem: Usuário não existe!. A figura a seguir demonstra o desenvolvimento dos blocos até o momento.

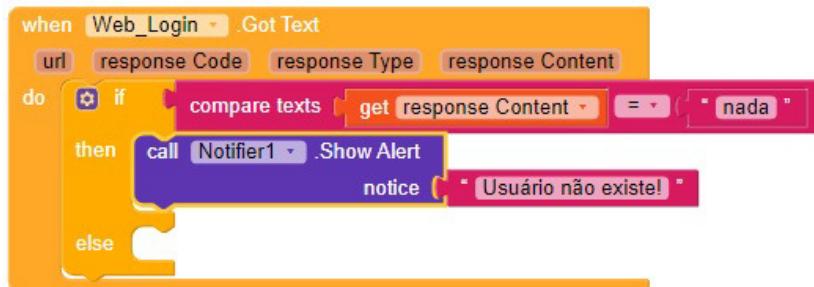


Figura 11.11: Verificação do cadastro do usuário

Caso os dados estejam corretos, utilizaremos a opção `else` para armazenar as informações do usuário em seu dispositivo e não precisar realizar o login em utilizações futuras. Para esse armazenamento, utilizamos o objeto não visível `Tiny_DB`. Insira o bloco de comando que adiciona um valor ao `Tiny_DB`, o procedimento `call Tiny_DB1.Store Value`. Nele temos duas

opções de entradas de valores: a primeira é uma `tag`, isto é, uma variável que será salva em seu dispositivo enquanto o app estiver instalado. E a segunda opção `value To Store` nos indicará o valor que deverá ser armazenado.

Na primeira opção, inclua um bloco de comando de textos e digite o nome – `usuario`. Veja na figura a seguir o bloco `Tiny_DB1` sendo desenvolvido.

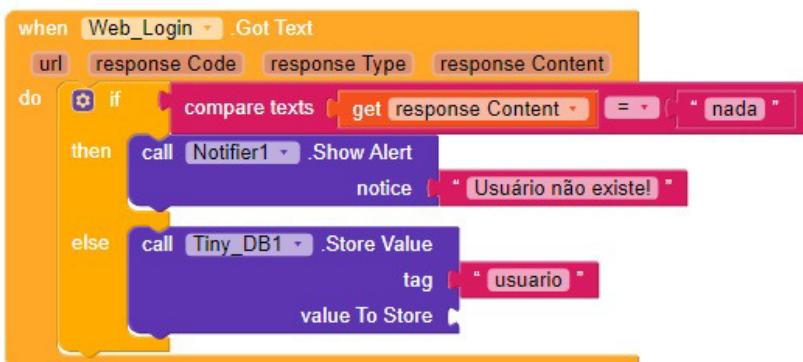


Figura 11.12: Bloco de comando `Tiny_DB1`

A identificação do usuário (`id`) é o que deverá ser armazenado no objeto `Tiny_DB1`, portanto é preciso identificar o campo `id` no texto decodificado. Para isso, na opção `value To Store` do `Tiny_DB1`, acrescente o bloco de procura `look up in pairs` e vamos realizar a busca do valor do campo `id`.

Na opção `key` adicione um bloco de texto com o nome do campo `id`. Na opção `pairs` insira o bloco de comando que realiza a decodificação do formato JSON dos dados recebidos – `call Web_Login.JSON Text Decode` – e informe utilizando a variável `get response Content` indicando os dados a serem

decodificados.

Para finalizarmos o processo de login, após o armazenamento da informação no dispositivo, devemos fechar a Screen LoginCadastro e retornar para a página que a solicitou. Vamos incluir um comando `close screen` da guia Built-in seção Control para realizar essa tarefa.

A imagem a seguir demonstra o comando de armazenamento local `Tiny_DB1` configurado juntamente com o evento `GotText` completo.

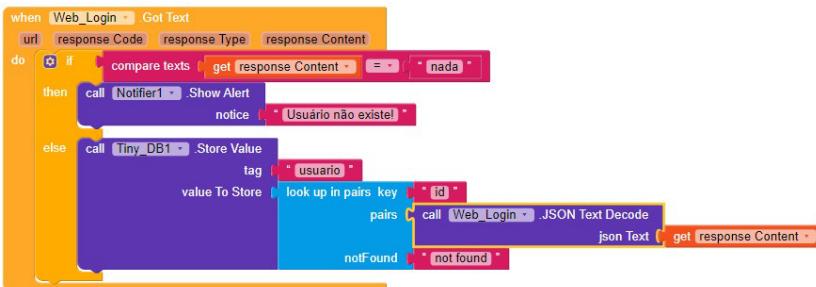


Figura 11.13: Evento GotText do Web\_Login finalizado

## 11.4 CADASTRANDO UM NOVO USUÁRIO

Vamos programar o botão que realiza o cadastro de um novo usuário e você verá um procedimento parecido com o que foi adotado para realizar o login. Vamos começar inserindo na área Viewer o bloco de comando do evento `Btn_Salvar.Click`.

Antes de cadastrar o nome, email e a senha do usuário, necessitamos verificar se esses valores foram preenchidos. Para a verificação, utilizamos o bloco do comando `if`. Configure-o para ter a opção `else`. Como teremos três valores para comparar (um

para o `Txt_Nome`, outro para o `Txt_Email` e mais um para o `Txt_Senha`), necessitamos preparar o comando `if` para ter dois comandos do bloco lógico `and`. A imagem a seguir demonstra como deve estar o desenvolvimento do bloco do `Btn_Salvar.click` até o momento.

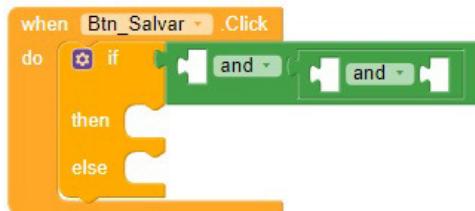


Figura 11.14: Bloco de comandos do `Btn_Salvar.click`

Como a `Txt_Senha`, `Txt_Email` e a `Txt_Nome` não poderão estar sem informações para serem cadastradas, necessitamos dos comandos que realizem essa tarefa. Conforme explicado anteriormente quando realizamos o login, organize as três linhas de comando:

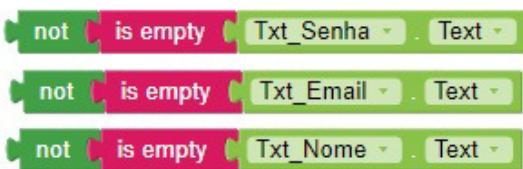


Figura 11.15: Comando `if` preparado para a verificação

Encaixe cada bloco de linha de comando que acabamos de criar em um dos espaços reservados dentro dos blocos lógicos `and`.

Para uma melhor exibição dos blocos, alteramos a opção para External Inputs do bloco and . A figura a seguir demonstra o bloco do comando de decisão if programado para verificar se as informações não estão vazias.

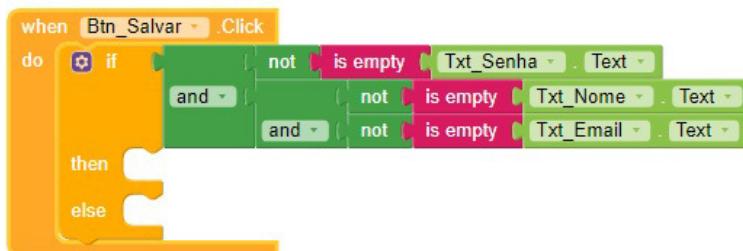


Figura 11.16: Blocos preparados para o comando if

Antes de enviar as informações digitadas para a realização do cadastro, necessitamos configurar o objeto Web1 com o endereço do arquivo `cadastro.php`. Para tanto, insira o bloco do comando `Web_Cadastro.URL to`, e adicione nele um bloco de texto com o endereço - [http://www.nelfabbri.com/avalia\\_run/cadastro.php](http://www.nelfabbri.com/avalia_run/cadastro.php).

O arquivo `cadastro.php`, também se encontra disponível e comentado no capítulo final deste livro.

A seguir a figura demonstra a configuração do endereço no bloco de comando `Set Web_cadastro.URL to`.



Figura 11.17: Configuração do endereço no objeto Web\_Cadastro

Para enviar as informações para realizar o cadastro do usuário, utilizaremos o método `POST` do objeto `Web_Cadastro`.

Acrescente o método `call Web_Cadastro.Post Text` juntamente com um objeto de texto `join` para realizar a junção dos dados que serão enviados. No comando `join`, necessitamos configurá-lo para ter 6 opções de entradas de textos. A tabela a seguir mostra as entradas de cada opção do comando `join` que deveremos realizar.

OPÇÃO	COMPONENTE	VALOR
1 <sup>a</sup>	Bloco de texto	<code>nome=</code>
2 <sup>a</sup>	Caixa de texto	<code>Txt_Nome.text</code>
3 <sup>a</sup>	Bloco de texto	<code>&amp;email=</code>
4 <sup>a</sup>	Caixa de texto	<code>Txt_Email.text</code>
5 <sup>a</sup>	Bloco de texto	<code>&amp;senha=</code>
6 <sup>a</sup>	Caixa de texto	<code>Txt_Senha.text</code>

Veja na figura a seguir como ficará o seu bloco de comando para enviar as informações através do método `POST` ao arquivo `cadastro.php`.

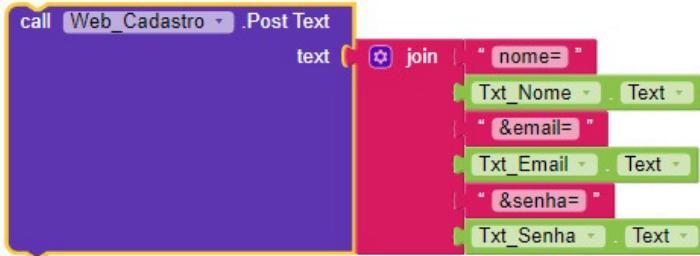


Figura 11.18: Blocos do comando call Web\_Cadastro.Post Text

Essas informações serão enviadas se todos os dados forem preenchidos corretamente. Mas caso alguma das informações não tenha sido digitada, devemos emitir um aviso ao usuário alertando-o sobre o esquecimento. Vamos programar a opção `else` do bloco de comando `if` para emitir esse aviso. Adicione um `call Notifier1.Show Alert` na opção `else` e inclua um bloco de texto com a frase `Todos os campos são obrigatórios`. A imagem a seguir exibe o bloco do evento `Btn_Salvar.click` finalizado.

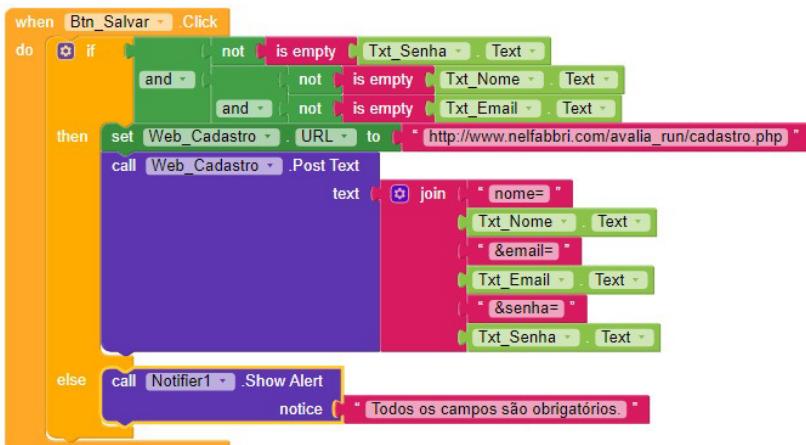


Figura 11.19: Bloco do evento `Btn_Salvar.click` finalizado

## 11.5 VERIFICANDO SE O CADASTRO FOI REALIZADO

Após o processamento do arquivo `cadastro.php`, o objeto `Web_Cadastro` receberá uma informação de retorno que nos informará se o cadastro foi realizado, caso contrário, será enviada para o app a mensagem `Não Salvo`.

Insira em sua área `Viewer` o bloco de comando que trata o evento `Got Text` do objeto `Web_Cadastro`. Como vamos verificar se o texto que retornou foi a frase `Não Salvo`, teremos que inserir um bloco de comando `if` e configurá-lo para possuir a opção `else`.

Inclua na sequência do comando `if`, um bloco de comando de comparação de textos `compare texts` para verificar se o texto retornado é igual a `Não Salvo`, portanto em seus espaços, inclua a variável `get response Content` e no outro espaço um bloco de texto com o valor `Não Salvo`. A imagem a seguir, demonstra os comandos preparados até o momento.



Figura 11.20: Blocos do evento Got Text

Caso os dados não tenham sido salvos, uma notificação deverá ser emitida ao usuário, e para isso utilizaremos um `call`

`Notifier1.ShowAlert` com uma caixa de texto, contendo a informação `Não Salvo`.

Caso os dados do usuário tenham sido gravados, necessitamos armazenar no objeto `Tiny_DB1` o número de identificação do usuário que foi gerado automaticamente pelo arquivo em PHP. O processo para armazenar o número de identificação do usuário é idêntico ao adotado quando o login foi realizado com sucesso.

Insira o procedimento `call Tiny_DB1.Store Value` na opção `else`. Em sua opção `tag` acrescente um bloco de texto e digite o valor `usuario`, para guardar o número de identificação necessitamos procurar o campo `id` no texto que foi retornado pelo arquivo em PHP. Na opção `value To Store` insira um bloco de comando `look up um pairs` e na opção `key` adicione um bloco de texto com o valor `id`. Por último, na opção `pairs`, insira um bloco do procedimento `call Web_Cadastro.JSON text Decode`, para decodificar o texto recebido no formato JSON e posicione no espaço reservado ao texto JSON a variável `get response Content`.

Após salvar a informação no `Tiny_DB1`, deixaremos a `Vertical Arrangement` de cadastro invisível e tornaremos a `Vertical Arrangement` de login visível para o usuário, pois assim que for realizado o novo cadastro, o usuário deverá realizar o seu login no app. Adicione os comandos `set VA_Cadastro.Visible to` e `set VA_Login.Visible to` e defina a visibilidade de cada objeto com um bloco de comando lógico `false` e `true`, respectivamente.

A imagem a seguir, exibe o bloco do evento `Web_Cadastro.Got Text` com sua programação completa.

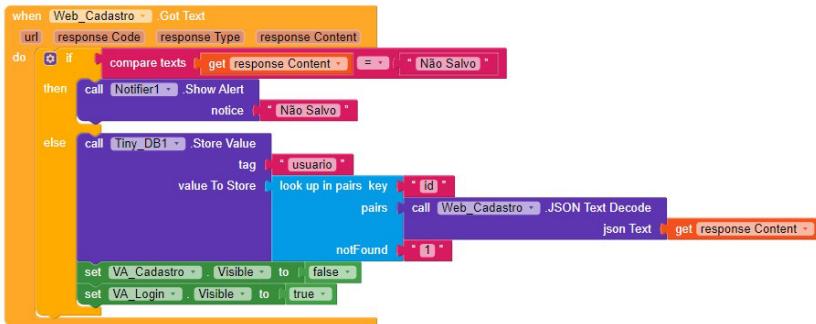


Figura 11.21: Programação completa do evento Web\_Cadastro.Got Text

## Conclusão

Aqui terminamos mais uma parte de nosso app de Avaliação de Corridas de Rua, cadastrando um usuário e realizando seu login. Vale lembrar que esta tela só poderá ser testada no momento em que o usuário do app desejar comentar ou avaliar uma corrida, o que será explicado e desenvolvido no próximo capítulo.

O arquivo desenvolvido até o momento está disponível para download e análises no site informado no início do livro.

## CAPÍTULO 12

# PROGRAMANDO A TELA DOS DETALHES

Chegamos à Screen final de nosso app. É nela que o usuário verá todos os detalhes da corrida selecionada: seu local, horário de largada, a nota média das avaliações dos demais usuários e todos os comentários realizados. Caso o usuário já tenha realizado o cadastro e o login, poderá postar os seus próprios comentários e avaliações, caso contrário, será exibida a tela de login e cadastro já visto no capítulo anterior.

Selecione a Screen Detalhes e acesse a área dos blocos, através do botão específico já demonstrado anteriormente.

Começaremos definindo algumas variáveis que utilizaremos mais tarde durante o decorrer da programação. Inicialize 4 novas variáveis globais com os seguintes nomes - `latitude` , `longitude` , `avaliacao` e `corrida` , todas elas deverão ser inicializadas com o valor zero (0) exceto a variável `corrida` , que deverá ser do tipo lista vazia. Como o próprio nome das variáveis já sugere, elas armazenarão informações referente a latitude e longitude do local da corrida, a nota de avaliação e o nome da corrida. A figura a seguir demonstra como deverão ficar as suas variáveis.



Figura 12.1: Declaração das variáveis

## 12.1 PROGRAMANDO A INICIALIZAÇÃO DA SCREEN

Quando a Screen Detalhes tiver sua solicitação de exibição realizada por uma das telas (Screen1 ou Principal), devemos programar no evento de inicialização algumas funções importantes para o seu bom funcionamento.

Durante o desenvolvimento do leiaute desta Screen, inserimos um objeto não visível chamado `Bottom_Sheet`, que, quando acionado pelo usuário, exibirá na parte inferior da tela um espaço reservado, sobrepondo aos demais elementos, para que possamos realizar nossas avaliações e comentários, porém, precisamos informar qual será o objeto que realizará essa função de sobreposição. No leiaute, preparamos uma `Vertical Arrangement` que renomeamos para `Va_avaliacao` para fazer esta função.

Insira na área de `Viewer` o evento `Initialize` do objeto da Screen `Detalhes`. Dentro dele, vamos configurar a exibição do `Bottom_Sheet` inserindo o bloco de comando `call Bottom_Sheet1.Register Layout As Dialog`. Em sua opção

`layout` indique o objeto que exibirá a sobreposição - `Va_avaliacao`. A imagem a seguir exibe o bloco de programação da configuração do `Bottom_Sheet` durante a inicialização da tela `Detalhes`.



Figura 12.2: configuração do Bottom\_Sheet

Os demais procedimentos da inicialização da tela `Detalhes` são praticamente idênticos aos utilizados na inicialização das demais telas. Vamos rever a lógica: necessitamos emitir ao usuário o aviso

`Buscando os Dados`. Para tanto, utilizamos o bloco de programação `call Notifier1.Show Progress Dialog` com a referida informação. Enquanto o aviso estiver visível ao usuário, o objeto `Web_get_Detalhes` é configurado com o endereço de solicitação dos dados na internet através do link - [http://www.nelfabbri.com/avalia\\_run/seleciona\\_item.php?id=/](http://www.nelfabbri.com/avalia_run/seleciona_item.php?id=/).

Note que estamos preparando o envio do número de identificação da corrida (`id`) que será consultada, porém essa informação será recebida de outra Screen, que poderá ser recuperada através do bloco de comando `get start value` da guia `Built-in` na seção `control`.

O arquivo `seleciona_item.php` está disponível e comentado para análises no capítulo final deste livro.

Após a configuração do endereço, vamos realizar a solicitação dos dados através do evento `call Web_get_Detalhes.Get`. A figura a seguir demonstra o bloco de inicialização da tela Detalhes concluída.



Figura 12.3: Evento Initialize completo

## 12.2 EXIBINDO OS DADOS DA CORRIDA SELECIONADA

Após o envio da solicitação dos dados da corrida pelo objeto `Web_get_Detalhes`, trataremos as informações recebidas que estão no formato JSON através do evento `Got Text` para exibi-las em tela.

Iniciaremos incluindo na área de `Viewer` o evento `Web_get_Detalhes.Got Text`. Não podemos esquecer de desativar a mensagem `Buscando os dados`, bastando acrescentar o procedimento `call Notifier1.Dismiss Custom Dialog`. Como já dissemos, os dados pesquisados são recebidos pelo app no formato JSON e deverão ser decodificados e armazenados em uma variável. Para tal, insira o bloco de comando `set global corridas to` e posicione o bloco de comando de decodificação

call Web\_get\_Detalhes.JSON Text Decode com as informações recebidas e que estão armazenadas na variável get response Content . A figura a seguir demonstra os procedimentos tratados até o momento.



Figura 12.4: Web\_get\_Detalhes.Got Text

Agora que já recebemos os dados e os decodificamos, vamos exibir cada informação em seu devido lugar. Para exibir o nome da corrida selecionada na label Lbl\_Titulo , insira o bloco de comando set Lbl\_titulo.text to abaixo da variável corrida.

Para exibir o título devemos procurar pelo campo nome através do comando look up in pairs e exibir o valor correspondente. Na opção key do look up in pairs , insira um bloco de texto com a identificação do campo que vamos procurar - nome . Em pairs insira a decodificação dos dados recebidos. A imagem a seguir, demonstra a exibição do nome da corrida na Lbl\_Titulo .

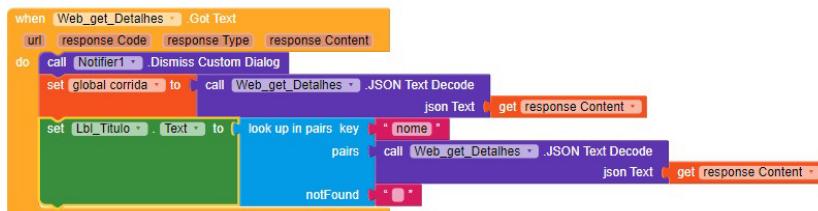


Figura 12.5: Comando para exibir o nome da corrida.

Para a exibição da imagem da corrida, insira o comando `set Image1.Picture to`, logo abaixo, para a descrição, inclua um comando `set Lbl_Descricao.Text to`.

Para o objeto `Image1` e para a `Lbl_Descricao`, os comandos para buscar os dados são os mesmos apresentados para exibição do título da corrida. Realize a procura utilizando um bloco de comando `look up in pairs` para cada objeto que acabamos de inserir e faça as configurações para exibi-los. A única diferença em relação à exibição do nome da corrida, são os nomes dos campos que devemos procurar na opção `key`. Para a imagem utilize um bloco de texto com o campo `foto` e para a sua descrição utilize um bloco de texto com o campo `descricao`. A seguir, é exibida a imagem com todos os blocos do evento `Got Text`, desenvolvido até o momento.

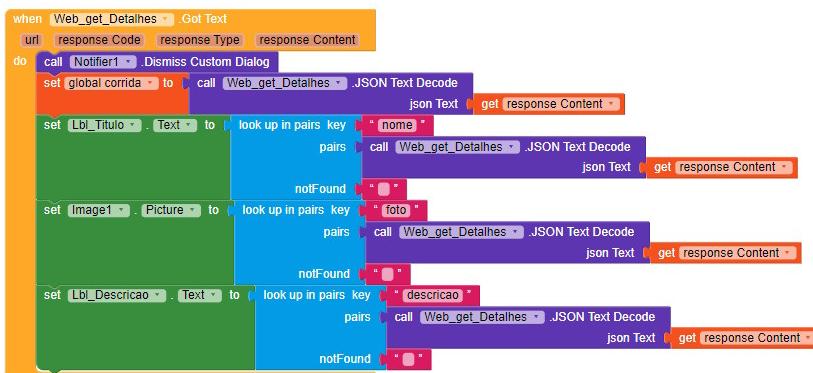


Figura 12.6: Comandos do evento GotText do objeto Web\_get\_Detalhes

## 12.3 SELECCIONANDO A LATITUDE E A LONGITUDE

Quando cadastramos uma nova corrida através do site, o

campo `google_maps` recebeu a latitude e a longitude digitadas no formato: `-23.637160,-46.577337`. Poderíamos ter cadastrado em dois campos distintos, mas não o fizemos para demonstrar o processo onde devemos separar as informações da latitude e longitude nas variáveis que foram criadas anteriormente para este fim.

A lógica da separação é a seguinte: sabendo que o valor da latitude e longitude estão separados por uma vírgula, devemos realizar a separação procurando por este símbolo. Esta separação gerará uma lista com os dois valores, sendo que o primeiro valor é a latitude e o segundo valor a longitude.

Na prática, vamos programar essa lógica primeiramente inserindo um bloco do comando `set global latitude to` para receber o valor da latitude que será o primeiro item da lista a ser separado. Da guia `Built-in`, seção `List`, insira o comando `select list item`, pois é ele quem realizará a seleção de uma das partes. Como queremos a primeira parte da separação, na opção `index`, insira o bloco numérico com o valor `1` em seu interior. Na opção `list`, devemos incluir o comando do bloco de texto que realizará a separação das informações. E da guia `Built-in` e seção `Text`, adicione o comando `split text`. Na opção de `at` indique o caractere vírgula `( , )` em um bloco de texto que deverá ser encontrado para servir de separação entre as opções latitude e longitude. Na opção `text` do comando `split`, devemos indicar o campo do banco de dados que possui essa informação. Novamente devemos utilizar o bloco `look up in pairs` para localizar a informação desejada. O campo que vamos inserir na opção `key` dentro de um bloco de texto é `google_maps`, que será procurado nos valores retornados do

objeto `Web_get_Detalhes` e deverá ser decodificado aqui também. Veja na imagem a seguir o comando que realiza a separação da latitude.

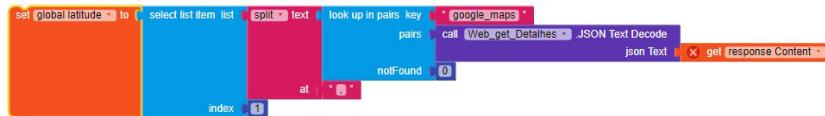


Figura 12.7: Separando o valor da latitude

Devemos realizar o mesmo procedimento para receber o valor da longitude. A única diferença será a seleção do item da lista: a longitude terá que ser informada como `index` de número `2`. A figura a seguir mostra todo o bloco do evento `Got Text` preparado até o momento.

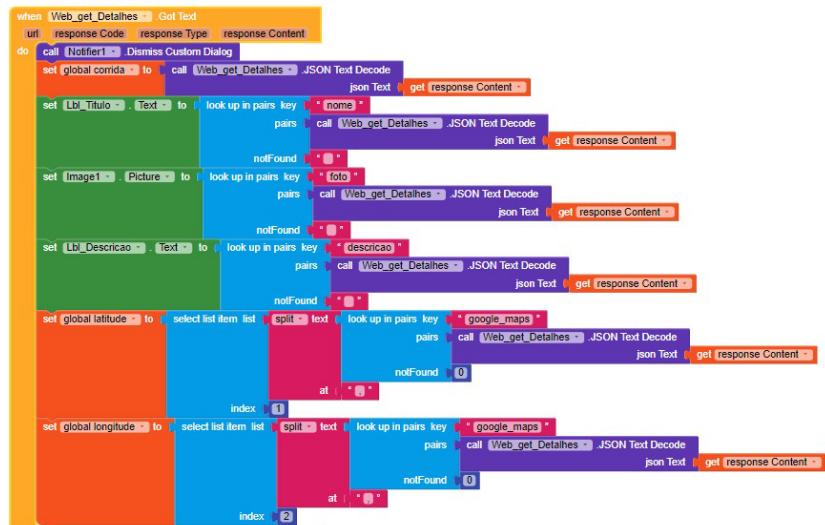


Figura 12.8: Bloco do evento Got Text

Para a exibição do valor da nota de avaliação na label e exibição da quantidade de estrelas recebidas no objeto Rating , vamos verificar se os valores estão zerados. Se estiverem, vamos configurar sua exibição com o valor zero (0). Insira um bloco de comando de decisão if , configurando-o para apresentar a opção else . Adicione um bloco de comparação de textos para realizarmos a verificação do valor da média, que deverá ser procurado através de um bloco do comando look up in pairs . O nome do campo a ser utilizado é media e seu conteúdo está no retorno das informações recebidas pelo objeto Web\_get\_Detalhes e deverá ser decodificado do formato JSON. Esse valor retornado deverá ser comparado com um bloco de texto com o valor zero (0) digitado em seu interior. Se essa comparação for verdadeira, devemos utilizar os blocos de comando set label3.text to e set Rating\_Bar1.Set rating to para exibir um bloco numérico com o valor zero (0) em seu interior, conforme demonstra a imagem a seguir.

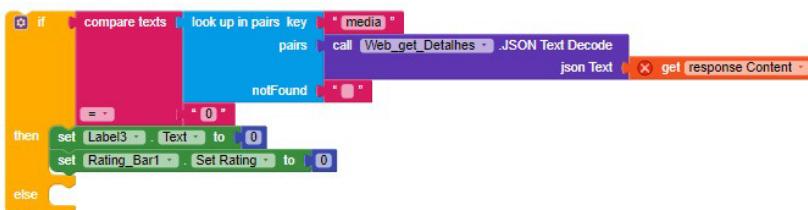


Figura 12.9: Verificando se a média de avaliação da corrida é igual a 0

Caso a comparação retorne um valor diferente de zero, vamos exibir esse número na label3 . Porém, como o cálculo da média pode ter mais que uma casa decimal e não queremos exibir todos esses números, vamos formatá-la para exibir apenas uma casa após a vírgula.

Essa programação deverá ser realizada no espaço reservado ao comando `else`. Vamos inserir o bloco de comando `set label3.text to`. Para configurar a formatação da quantidade de casas decimais que queremos exibir, acesse a guia `Built-in` e, na seção `Math`, selecione o bloco do comando `format as decimal number`. Na sua opção `places` insira um bloco numérico com a quantidade de casas decimais que deseja exibir. Em nosso caso, utilizamos o número `1`. Para selecionar o valor da média, utilize o bloco de comando `look up in pairs`. Veja na imagem a seguir como ficou o comando de exibição da nota de avaliação na `label3`.



Figura 12.10: Exibição da nota média de avaliação da corrida.

Para a exibição do número de estrelas recebidas pela corrida, utilizaremos o comando que ajusta essa quantidade - `set Rating_Bar1.SetRating to`. O valor da média também é obtido através do bloco de comando `look up in pairs`. Veja o bloco do comando exibido na figura a seguir.

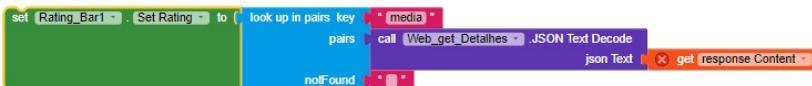


Figura 12.11: Configuração do Rating\_Bar1

## 12.4 PROCEDURE PARA EXIBIR OS COMENTÁRIOS

Os comentários referentes a essa corrida estão armazenados em outra tabela no banco de dados e por esta razão necessitamos realizar uma nova consulta de dados e realizar o correto tratamento para exibi-las. Para essa nova consulta, vamos aprender a declarar uma *procedure* que terá a função de chamar a exibição dos comentários.

*Procedure* é uma coleção de instruções implementadas que poderão ser solicitadas quantas vezes forem necessárias, bastando utilizar um comando de chamada com o nome definido.

Acesse a guia *Built-in* na seção de *procedures* e insira um bloco *to procedure* na área *Viewer*. Altere o nome da *procedure* para *buscar\_comentarios* para sua identificação. O bloco da *procedure buscar\_comentarios* deverá ser exibida conforme imagem a seguir.

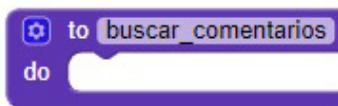


Figura 12.12: Bloco da procedure *buscar\_comentarios*

A *procedure buscar\_comentarios* possuirá os blocos de configuração do objeto *Web\_buscar\_comentarios*, com o

endereço do arquivo em PHP e o número de identificação da corrida. Para realizar a chamada dos comentários necessitamos inserir o bloco `set Web_busca_comentarios.URL to` e nele posicionar um bloco de texto `join` que realizará a junção de três valores. O primeiro bloco de texto a ser utilizado levará o endereço base dos arquivos em PHP - [http://www.nelfabbri.com/avalia\\_run](http://www.nelfabbri.com/avalia_run). No segundo espaço do comando de texto `join` devemos inserir o nome do arquivo que buscará os comentários `seleciona_comentarios.php` seguido do campo `?id=`, que indicará ao arquivo em PHP qual corrida ele deverá buscar os comentários. Veja na figura a seguir como deve estar o desenvolvimento da *procedure* `buscar_comentários`.



Figura 12.13: Procedure `buscar_comentarios`

No último espaço da `join`, necessitamos indicar qual corrida queremos buscar para exibir os comentários. Faremos isso identificando o valor do campo `id`, através do comando `look up in pairs`. A imagem a seguir exibe esta programação.



Figura 12.14: Identificando o código da corrida para pesquisar os comentários

Para finalizar os comandos da *procedure*, vamos realizar a

chamada do método `call Web_buscar_comentarios.get` para que sejam buscados os comentários desejados. A seguir a figura da *procedure* `buscar_comentarios` finalizada.

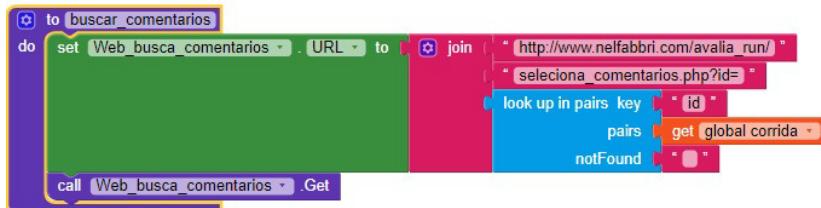


Figura 12.15: Procedure `buscar_comentarios` finalizada

Terminada a programação da procedure `buscar_comentarios`, é preciso utilizá-la em algum lugar, pois apenas a criamos. E qual seria o local onde devemos fazer isso?

Vamos utilizá-la como o último dos blocos, após a exibição de todos os detalhes da corrida selecionada pelo evento `Got text` do objeto `Web_get_Detalhes`.

Para chamar a execução da *procedure* criada, acesse a guia `Built-in` na seção `procedures` e selecione a chamada `call buscar_comentarios`. A imagem a seguir exibe o final dos blocos dos comandos do evento `Got text` com a chamada da *procedure* `buscar_comentarios`.

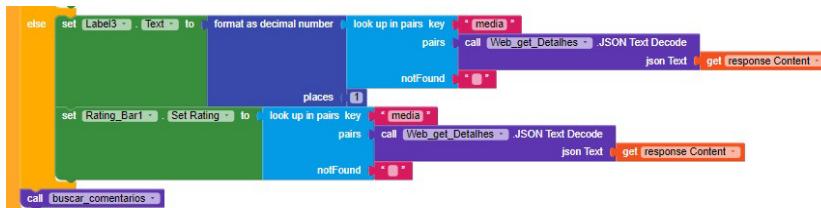


Figura 12.16: Chamado da execução da procedure

Após a execução da *procedure* buscar\_resultado todos os comentários serão retornados para o aplicativo no formato de texto JSON. Porém, precisamos verificar se a corrida possui algum comentário realizado. Caso não possua, não precisamos realizar nenhuma tarefa.

Para essa verificação, insira um bloco de comando Web\_busca\_comentarios.Got Text na área de Viewer e insira um bloco de comando de decisão *if*. O bloco do comando *if* verificará se os valores que retornaram no formato decodificado JSON é do tipo lista e, caso sejam, indica que temos algum comentário para exibição. Isso poderá ser programado inserindo o bloco de comando *is a list? thing* da guia Built-in seção Lists . Nele, devemos inserir o bloco de decodificação do retorno das informações do banco de dados. A imagem a seguir exibe o bloco de recebimento dos dados com o comando *if* preparado.

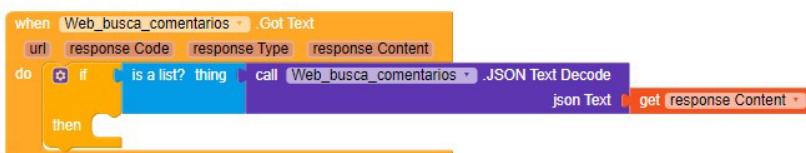


Figura 12.17: Verificação se há comentários realizados na corrida

Caso o resultado da condição do comando *if* seja verdadeiro, necessitamos inicializar uma variável local chamada comentarios para armazenar todos os comentários realizados para esta corrida. Como podemos possuir inúmeros comentários, o tipo da variável que utilizaremos deverá ser do tipo lista vazia (*create empty list* ).

Assim como todos os eventos em que tratamos a exibição dos

dados retornados de um arquivo em PHP, precisamos utilizar o bloco de comando `for each item in list` para percorrer essa lista e separar as informações desejadas. A imagem a seguir exibe a programação realizada até o momento com o evento `Got Text` do objeto `Web_busca_comentarios`.

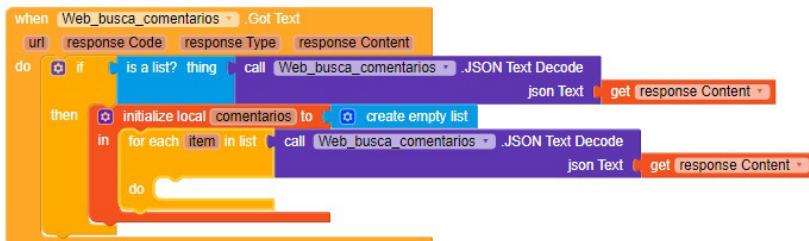


Figura 12.18: Estágio atual do objeto `Web_busca_comentarios.Got Text`

Poderá ocorrer casos onde o usuário apenas realizou a avaliação numérica da corrida e não escreveu nenhuma postagem, portanto ao percorrer a lista necessitaremos verificar se o campo `descricao` **não é igual a um valor vazio**. Como estamos tratando de uma verificação, acrescentaremos um bloco de comando `if` e na sequência um bloco lógico de negação. Para comparar os valores, devemos inserir um bloco que realiza a comparação de textos - `compare texts`. No primeiro espaço da comparação de textos vamos comparar a `descricao` com um bloco de texto vazio, utilizando o conhecido bloco de comando `look up in pairs`. A imagem a seguir demonstra esse procedimento.

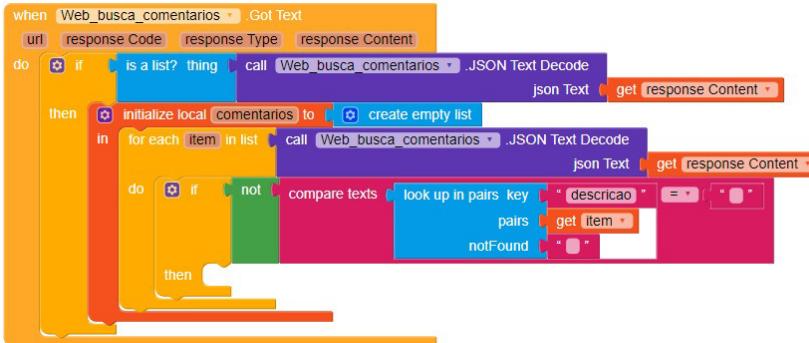


Figura 12.19: Verificação da existência de comentário

Caso o campo da descrição não esteja vazio, vamos adicionar o nome do usuário e seu comentário em uma lista para posterior exibição. Como já visto, o bloco de comando `add items to list` realiza a função de adicionar um item na lista. Adicione-o logo à frente do comando `then`. Na opção `list` adicione a variável local `get comentarios`. Em `item`, adicione um bloco de texto `join` e configure-o para receber 4 entradas. Essas entradas receberão o nome do usuário que ficará formatado em negrito e na sequência será exibida a opinião da corrida.

A primeira entrada do bloco `join` receberá um bloco de texto com o comando `<strong>` da linguagem HTML que deixa um texto em negrito. Na segunda entrada, devemos localizar o campo nome do usuário utilizando o recurso do `look up in pairs`. Na penúltima entrada do `join`, insira mais um bloco de texto e digite o comando em HTML que encerra o comando de negrito juntamente com uma mudança de linha - `</strong><br>`. Na última entrada, vamos acrescentar a descrição feita pelo usuário do aplicativo. Para isso repita a inserção de um bloco de comando `look up in pairs` e configure-o para localizar o campo

descricao . Veja na imagem a seguir o evento Got Text com toda a programação vista até o momento.

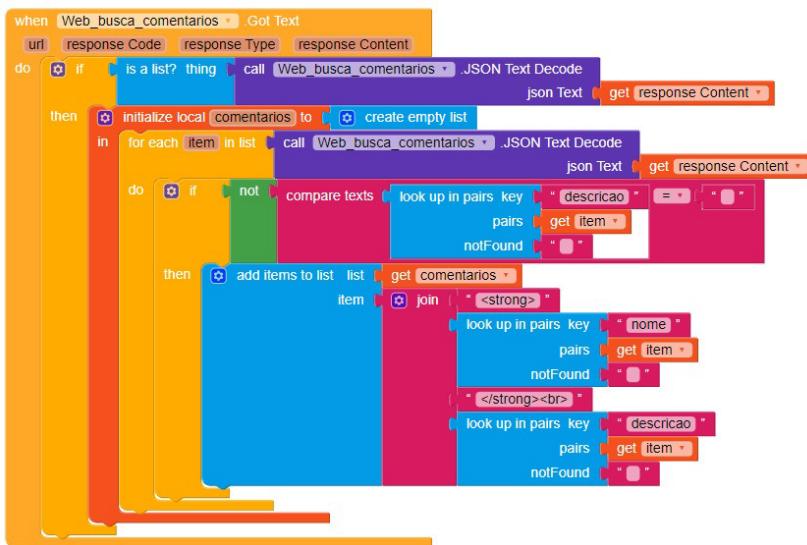


Figura 12.20: Estágio atual do objeto Web\_busca\_comentarios.Got Text

Após carregar todos os itens na variável `comentarios` , devemos exibir a lista no objeto `List_View` . Para tanto, após a finalização do comando `for each` , acrescente o bloco de comando `set List_view1.Elements to` e adicione a lista dos comentários que está na variável local `get comentarios` .

Após a exibição de todas as informações desta Screen, não podemos esquecer que a mensagem Buscando os dados ainda está ativa e visível para o usuário, portanto, necessitamos encerrar a sua visualização. Acrescente o comando que encerra sua exibição através do bloco `call Notifier1.Dismiss Progress Dialog` . A figura a seguir demonstra o bloco completo do `Web_busca_comentarios.Got Text` .

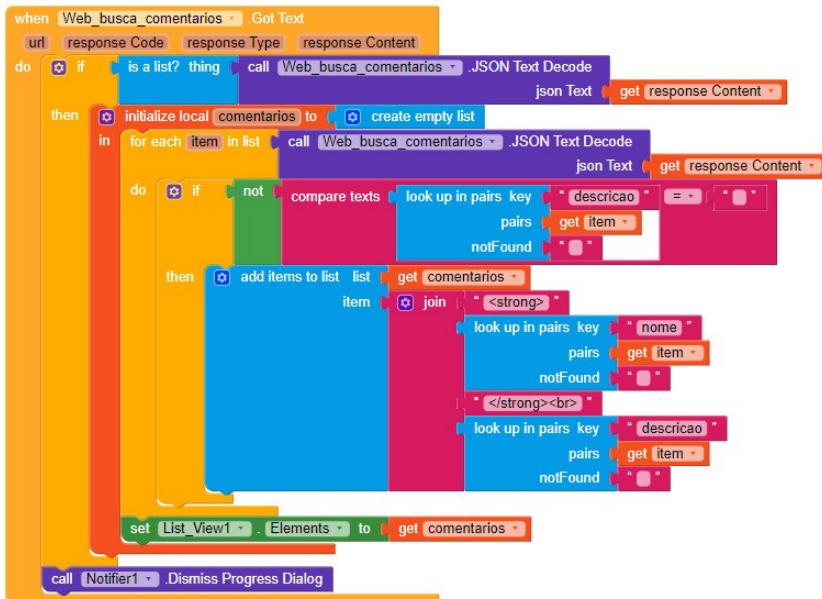


Figura 12.21: Objeto Web\_busca\_comentarios.Got Text finalizado

## 12.5 EXIBINDO A LOCALIZAÇÃO NO MAPA

A exibição do local de largada da corrida no mapa ocorrerá ao clicar no botão `Btn_Mapa`. A lógica de programação que será a seguinte: após clicar no botão, devemos criar um marcador de exibição (pin) para o local com as coordenadas da latitude e longitude e posteriormente posicionar o mapa para ser exibido nessas coordenadas.

Adicione o bloco de comando `Btn_Mapa.Click` na sua área `Viewer`. Para a criação do marcador, adicione o bloco de comando `call Marker1.Set Location` e posicione os valores das variáveis `get global latitude` e `get global longitude` nas opções de mesmo nome.

Com o marcador criado, precisamos centralizar a exibição do mapa nessas coordenadas. Insira o bloco de comando `set Map1.Center From String`. Adicione também um bloco de texto `join` e configure-o para possuir três entradas. Na primeira opção coloque a variável que armazena a latitude. Na segunda entrada, adicione um bloco de texto com apenas uma vírgula em seu interior, pois servirá para o mapa identificar onde termina a latitude e começa a longitude. Na última entrada do `join`, insira a variável que contém o valor da longitude.

A imagem a seguir exibe o bloco com a programação completa para a exibição do local da corrida no mapa.

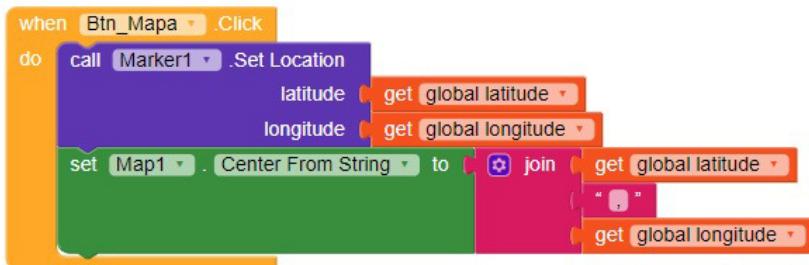


Figura 12.22: Exibindo o local da corrida no mapa

## 12.6 INSERINDO UMA AVALIAÇÃO

Chegamos ao momento em que o usuário do app poderá realizar sua avaliação dando uma nota e realizando um comentário sobre a corrida, porém, essa avaliação só será possível se o usuário estiver logado e, caso não esteja, ele será direcionado automaticamente para a tela `LoginCadastro`.

Insira na área `Viewer`, o bloco `Btn_Comentar.Click` e para verificar se o usuário está ou não logado, insira um bloco do

comando `if` e configure para possuir a opção `else`.

No comando `if` insira o bloco de comparação de textos `compare texts` para verificar se a tag `usuario` do objeto `Tiny_DB1` está ou não vazia. Na Screen `LoginCadastro` depois de realizado o login, o objeto `Tiny_DB1` armazenou o nome do usuário. Como dissemos anteriormente esse objeto armazena em seu dispositivo informações que podem ser recuperadas em outra Screen, portanto vamos utilizar o procedimento `call Tiny_DB1.Get Value` que vai ler a informação gravada na tag `usuario` e comparar se o conteúdo está vazio. A seguir veja o bloco do `Btn_Comentar.Click` programado com a decisão.

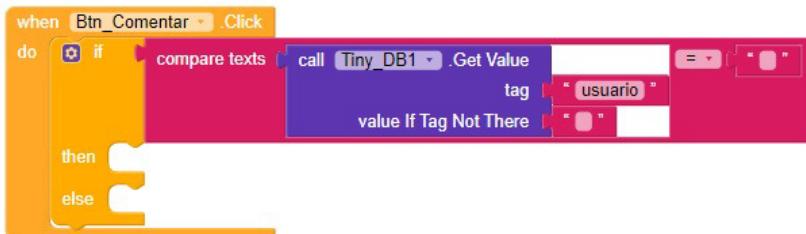


Figura 12.23: `Btn_Comentar.Click` com a verificação de login

Se o resultado da comparação for verdadeiro, isso indica que o usuário ainda não se logou ou se cadastrou no sistema e assim deverá ser exibida para o usuário a Screen `LoginCadastro`. Utilize o comando `open another screen screenname` para isso.

Mas caso o resultado da verificação seja falso, devemos habilitar a exibição da `Vertical Arrangement` da avaliação, alterando a sua propriedade `Visible` para verdadeira. Para tanto, na opção `else` insira o bloco `set Va_avaliacao.Visible to` e deixe-a marcada com o bloco lógico `true`.

Agora já podemos ativar a exibição da área de avaliação utilizando a chamada da propriedade `call Bottom_Sheet1.Show Dialog`. A figura a seguir exibe o evento clique do `Btn_Comentar` finalizado.



Figura 12.24: `Btn_Comentar.Click` finalizado

Agora o usuário já poderá realizar a sua avaliação. Vamos armazenar na variável `set global avaliacao to` o número de estrelas recebidas do usuário. Adicione na área `Viewer` o evento que será acionado após o usuário clicar em uma das estrelas, ou seja, o bloco de comando `Rating_avaliar.Changed`. Para armazenar o valor da avaliação, inclua o bloco do comando `set global avaliacao to`. O valor recebido está disponível na variável `rating` do próprio evento. A imagem a seguir demonstra o processo descrito.

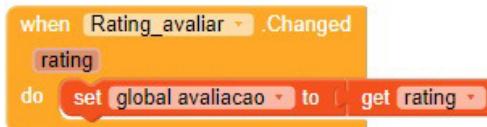


Figura 12.25: Recebendo a nota de avaliação do usuário.

O número de estrelas da corrida já foi armazenado e, caso o

usuário tenha realizado um comentário na `Txt_comentar.Text` enviaremos essas informações para o banco de dados quando o `Btn_enviar_comentario` for clicado. Como vimos anteriormente informe o caminho do arquivo em PHP que executará a tarefa de inserir as informações no banco de dados.

Insira um controle do objeto `Btn_enviar_comentario.Click` e em seu interior configure o endereço no objeto `Web_Comentario` para executar a gravação das informações. Acrescente um bloco de junção de texto `join`. Nele, insira dois blocos de textos. No primeiro, digite o endereço base do site que tem todos os arquivos em PHP de nosso app - [http://www.nelfabbri.com/avalia\\_run](http://www.nelfabbri.com/avalia_run). No segundo, espaço de texto digite o nome do arquivo - `comentario.php`.

O arquivo `comentario.php` está disponível e analisado no último capítulo deste livro.

A imagem a seguir exibe a configuração do objeto `Web_Comentario`.



Figura 12.26: Configuração do objeto `Web_Comentario`

As informações que serão enviadas para o arquivo `comentario.php` são: o comentário, sua nota de avaliação, o

código de identificação da corrida (id) e o nome do usuário.

Para realizar o envio de todos esses dados, utilizaremos o método Post do objeto Web\_Comentario . Nele vamos inserir um bloco de junção join com 8 entradas configuradas.

Na primeira opção de entrada inclua um bloco de texto com a informação descricao= . Na segunda opção de entrada adicione o objeto que está armazenando os comentários, ou seja, devemos inserir o bloco do objeto Txt\_comentar.Text . Precisamos adicionar a nota de avaliação do usuário, então adicione um bloco de texto com a informação &avaliacao= . Na quarta opção de entrada adicione o valor da avaliação que está armazenado na variável get global avaliacao . Na quinta posição, insira o número da corrida incluindo um bloco de texto com o valor &corrida\_id= . A imagem a seguir exibe o bloco de envio das informações até o presente momento do desenvolvimento.

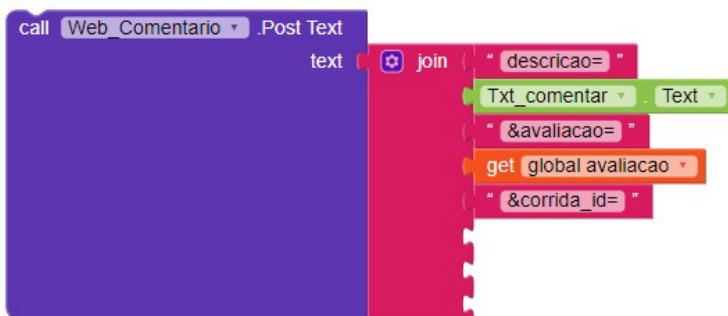


Figura 12.27: Preparando o bloco de envio das informações

Na sexta posição de entrada do comando join , utilizaremos o bloco de comando look up in pairs para verificar o id da corrida que está na lista da variável get global corrida .

Na sétima opção de entrada, insira um bloco de texto com a informação do nome do usuário que será enviado ao arquivo para processamento - `&user_id=`.

Como o nome do usuário está armazenado no componente `Tiny_DB1`, usaremos o evento que possibilita a leitura desta informação. Insira o `Tiny_DB1.Get Value`, e na opção `tag` digite o nome do campo que foi armazenado a informação quando realizamos o login - `usuario`. A figura a seguir demonstra o objeto `Web_Comentario.Post Text` preparado para o envio dos dados.

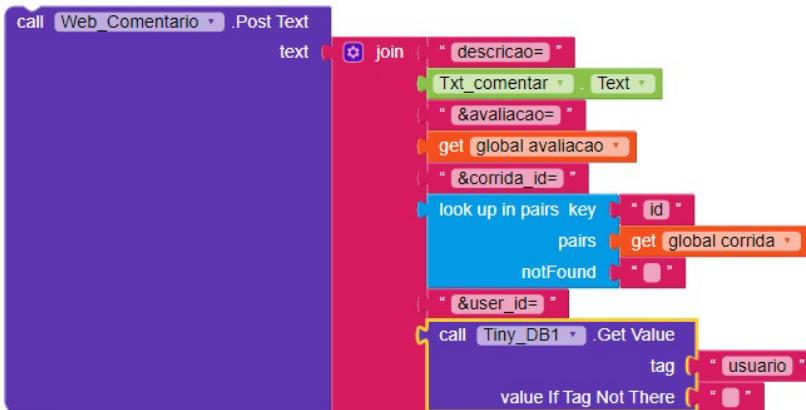


Figura 12.28: `Web_Comentario.Post Text` preparado para o envio dos dados.

Após o envio das informações de avaliação, deveremos desativar a exibição da área de avaliação, utilizando para isso o procedimento `call Bottom_Sheet1.Hide Dialog`. A seguir temos a exibição completa do `Btn_enviar_comentario.Click`.

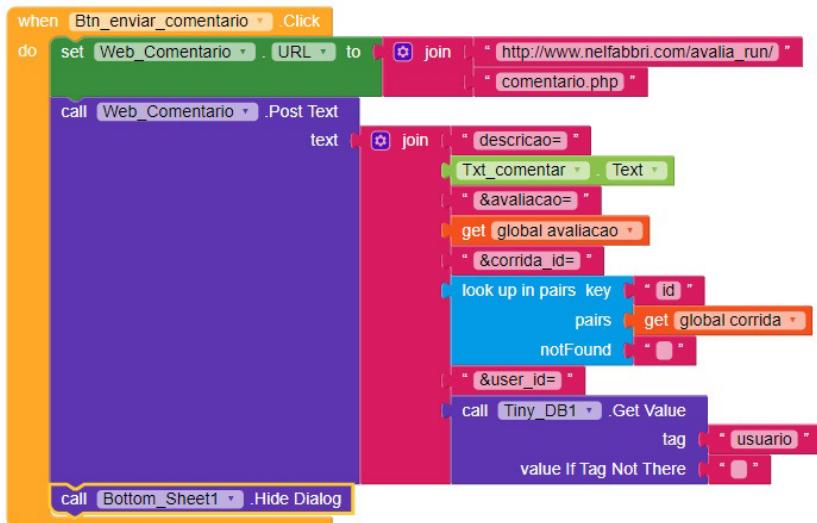


Figura 12.29: Btn\_enviar\_comentario.Click finalizado.

Após o envio das informações pelo método `Post`, o arquivo `comentario.php` retornará uma informação para o app, informando que o comentário foi adicionado ou se ocorreu algum problema durante o processo. Através de uma notificação exibiremos o resultado do processamento.

Insira o evento `Got Text` do objeto `Web_Comentario` com o procedimento `call Notifier1.Show Alert` para exibir o resultado do processamento que está disponível na variável `get response Content`. A seguir veja a figura com o bloco completo do evento `Got Text` do objeto `Web_Comentario`.

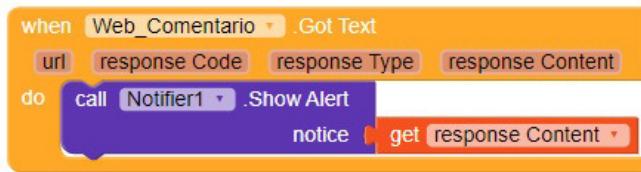


Figura 12.30: Web\_Comentario.Got Text concluído.

Quando desabilitamos a exibição do objeto `Bottom_Sheet1`, através do bloco de comando mostrado anteriormente `call Bottom_Sheet1.Hide Dialog`, precisamos deixar a visualização da `Vertical Arrangement Va_avaliacao` invisível para o usuário. Portanto, insira o bloco que é executado apenas quando o `Bottom_Sheet1` é fechado. Esse evento é o `Bottom_Sheet1.Closed`. Em seu interior adicione o bloco que trata da visibilidade do objeto `set Va_avaliacao.Visible to` e defina-o com o bloco lógico `false`. Veja na imagem a seguir como deve estar a programação do botão.

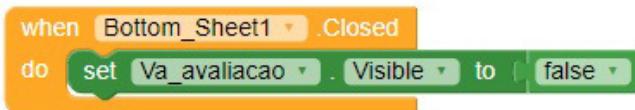


Figura 12.31: Evento Bottom\_Sheet1.Closed finalizado

Quando o usuário pressionar a tecla voltar do próprio dispositivo, a Screen Detalhes deverá ser fechada retornando para a tela que a chamou anteriormente. Inclua o evento `Detalhes.Back Pressed` e adicione o comando `close screen` que fecha a tela e está disponível na guia `Built-in` da seção `Control`. Veja na figura a baixo o bloco de programação do evento `Back Pressed` da Screen Detalhes.

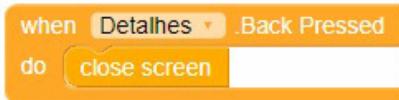


Figura 12.32: Blocos do comando Detalhes.Back Pressed

## Conclusão

Aqui finalizamos a programação do aplicativo de Avaliação de Corridas de Rua. Você poderá realizar sua instalação seguindo as orientações apresentadas anteriormente e executar os testes desejados.

O arquivo final de nosso projeto está disponível para download conforme link informado no início do livro, mas será muito mais gratificante que você realize o seu próprio aplicativo conhecendo cada fase do processo que realizamos.

# ARQUIVOS DE BACK-END

Os termos **front-end** e **back-end** na tecnologia da informação referem-se às etapas iniciais e finais de um processo. O **front-end** é responsável por receber a entrada das informações do usuário e processá-las para adequá-las a uma especificação que o **back-end** possa utilizar. Ou seja, a parte do front-end foi tratada dentro dos blocos do aplicativo na plataforma Kodular, já a parte do back-end será tratada por arquivos na **linguagem PHP** que estão hospedados em um servidor web, pois somente assim nosso aplicativo poderá ser utilizado de maneira colaborativa.

A seguir, vamos exibir e comentar cada linha dos arquivos na linguagem PHP utilizados durante o desenvolvimento do app de Avaliação de Corridas de Rua.

Lembramos que esses arquivos não precisarão ser digitados pelo usuário para a criação de nosso app, pois o foco deste livro é a programação em blocos na plataforma Kodular, entretanto o/a leitor/a poderá desenvolver seus próprios arquivos em PHP ou realizar alterações para melhorar os demonstrados a seguir.

## 13.1 CONECTA.PHP

Esse arquivo tem a finalidade de realizar a conexão com o

banco de dados hospedado na web. Ele será utilizado em todos os demais programas de nosso aplicativo, para reduzir a repetição da abertura do banco de dados.

```
1.  <?php
2.      $conexao = mysqli_connect('localhost', 'seu_usuario', 'su
a_senha', 'nome_de_seu_banco_de_dados');
3.  ?>
```

Linha 1 - Linha de abertura padrão e obrigatória da tag PHP.

Linha 2 - A variável \$conexao armazena a definição da conexão com o banco de dados, o comando mysqli\_connect é o responsável por essa conexão. Nele deverão ser informados alguns parâmetros como o local do banco de dados, a identificação do usuário, a senha de acesso e o nome do banco de dados no servidor.

Linha 3 - Encerramento obrigatório da indicação da tag PHP.

## 13.2 DESTAQUES.PHP

Sua finalidade é procurar pelas três primeiras corridas mais bem avaliadas e enviá-las ao aplicativo para exibição na Screen1 , que é a tela de destaque.

```
1.  <?php
2.      require "conecta.php";
3.      $sqlBusca = "select * from corridas order by media DESC
LIMIT 3 ";
4.      $resultado = mysqli_query($conexao,$sqlBusca);
5.      while ($array = mysqli_fetch_assoc($resultado)) {
6.          $dados[] = $array; }
7.      echo json_encode($dados);
8.  ?>
```

Entendendo as linhas de programação:

Linha 1 – Linha de abertura padrão e obrigatória da tag PHP.

Linha 2 – Importa o arquivo `conecta.php`, que é responsável pela conexão com o banco de dados.

Linha 3 – Armazena na variável `$sqlBusca` a instrução SQL que seleciona as 3 corridas mais bem avaliadas pelos usuários na tabela do banco de dados `corridas`. Observe que o campo que estamos selecionando é o campo `media` e vamos ordená-los da maior avaliação para a menor.

Linha 4 – Executa o comando que está na variável `$sqlBusca` e armazena os valores das 3 corridas na variável `$resultado`.

Linha 5 – Obtém uma linha do conjunto de resultados como uma matriz associativa. Quando nos referimos a associações, queremos dizer que, em vez de usarmos números como índices, podemos usar strings (nomes). Dessa forma, fica muito mais simples e intuitivo obter um valor de um array, pois nomes fazem mais sentido do que números.

Linha 6 – A cada iteração do comando `while`, uma linha com registro será adicionada à variável `$array`, que é do tipo lista.

Linha 7 – Realiza a codificação dos dados para o formato JSON das corridas que estão armazenados na variável `$dados` para serem retornadas para o aplicativo.

Linha 8 – Encerramento da tag PHP.

### 13.3 QUERY.PHP

Esse arquivo destina-se à exibição de todas as corridas para

exibição na Screen Principal e pode também realizar o filtro que é descrito no capítulo.

```
1. <?php
2.     require "conecta.php";
3.     $tipo = $_POST['tipo'];
4.     if ($tipo ==1){
5.         $sqlBusca = "select * from corridas order by nome asc";
6.     }
7.     if ($tipo==2){
8.         $sqlBusca = "select * from corridas order by nome desc"
;
9.     }
10.    if ($tipo==''){
11.        $sqlBusca = "select * from corridas";
12.    }
13.    $resultado =mysqli_query($conexao,$sqlBusca);
14.    while ($array = mysqli_fetch_assoc($resultado)) {
15.        $dados[] = $array;
16.    }
17.    echo json_encode($dados);
18. ?>
```

Entendendo as linhas de programação:

Linha 1 – Linha de abertura padrão e obrigatória da tag PHP.

Linha 2 – Importa o arquivo `conecta.php`, que é responsável pela conexão com o banco de dados.

Linha 3 - Recebe a variável do app que indica a ordem em que deverão ser exibidas as corridas (alfabética crescente ou decrescente).

Linhos 4 a 6 - Se o tipo da corrida for igual a 1, ocorrerá uma busca no banco de dados pelo nome das corridas em ordem alfabética crescente.

Linhos 7 a 9 - Se o tipo da corrida for igual a 2, ocorrerá uma

busca no banco de dados pelo nome das corridas em ordem alfabética decrescente.

Linhas 10 a 12 - Se o tipo da corrida estiver vazio, a consulta ao banco de dados será realizada pela ordem em que foram cadastradas.

As demais linhas terão as mesmas funções apresentadas a partir da linha 4 do arquivo destaque.php demonstrado anteriormente.

## 13.4 LOGIN.PHP

A finalidade do arquivo login.php é receber o email e a senha que são enviadas pelo app e verificar se o usuário está cadastrado no banco de dados para poder realizar uma avaliação de uma corrida.

```
1.      <?php
2.      require "conecta.php";
3.      $email=$_POST['email'];
4.      $senha= $_POST['senha'];
5.      $sql = "SELECT * FROM usuarios WHERE email='".$email' and
6.              senha='".$senha."'";
7.      $resultado = mysqli_query($conexao,$sql);
8.      $dados = [];
9.      while($linha = mysqli_fetch_assoc($resultado))
10.         $dados[]=$linha;
11.      if (count($dados) > 0)
12.      { echo json_encode($dados[0]); }
13.      else
14.      { echo "nada"; }
```

Linha 1 – Linha de abertura padrão e obrigatória da tag PHP.

Linha 2 – Importa o arquivo conecta.php , que é responsável

pela conexão com o banco de dados.

Linha 3 – A variável local `$email` recebe o valor do `email` enviado pelo app através do método `POST` do objeto `Web`.

Linha 4 – *Idem* à linha anterior, porém recebe o valor da senha e armazena na variável local `$senha`.

Linha 5 – A variável local `$sql` armazena uma instrução SQL, que realiza uma busca na tabela `usuarios` com os valores das variáveis do `email` e `senha` que foram enviadas pelo app e recebidas pelo arquivo de `login.php`.

Linha 6 – Após a definição da instrução que está na variável `$sql`, o comando `mysqli_query` executa o comando de consulta do login e senha no banco de dados e armazena os valores na variável `$resultado`.

Linha 7 – Define uma variável chamada `$dados` do tipo array. O tipo array em PHP é similar ao tipo lista utilizado no aplicativo.

Linha 8 – O comando `mysqli_fetch_assoc` retornará o conjunto de resultados da linha da consulta.

Linha 9 – Armazena na variável `$dados` a informação que retornou da consulta SQL.

Linha 10 – Através do comando `count` estamos verificando se o total de registros retornados é maior do que zero, ou seja, se houver um registro é porque o usuário acertou o login e a senha.

Linha 11 – No caso de o login e senha estarem corretos todas as informações do usuário serão codificadas no formato JSON e enviadas para o app.

Linha 12 – O comando `else` indica que o login ou a senha estão incorretos, neste caso será executada a informação que está na linha 13.

Linha 13 – A mensagem `nada` será retornada para o app caso os dados da tentativa de login estiverem errados.

Linha 14 – Encerramento da tag PHP.

Lembramos que as informações que estamos enviando como retorno para o app serão recebidas pelo evento `Got Text` do objeto `Web`.

## 13.5 CADASTRO.PHP

Caso o usuário não tenha realizado seu cadastro, o arquivo a seguir realizará a tarefa de incluí-lo no banco de dados. Vejamos:

```
1.      <?php
2.          $nome = $_POST['nome'];
3.          $email =$_POST['email'];
4.          $senha =$_POST['senha'];
5.          require "conecta.php";
6.          $sql = "insert into usuarios (nome, email, senha) values
    ('$nome','$email','$senha')";
7.          $resultado = mysqli_query($conexao,$sql);
8.          if ($resultado){
9.              $sql = "SELECT * FROM usuarios where nome = '$nome'";
10.             $resultado = mysqli_query($conexao,$sql);
11.             while ($array = mysqli_fetch_assoc($resultado))  {
12.                 $dados[] = $array;          }
13.                 echo json_encode($dados[0]);  }
14.             else{
15.                 echo "Não Salvo";}
16.         ?>
```

Linha 1 – Identificação da abertura da tag PHP.

Linha 2 – Recebe e armazena o valor do nome digitado no app e enviado pelo objeto web através do método POST .

Linha 3 – Recebe e armazena o valor do email enviado pelo app.

Linha 4 – Recebe e armazena o valor da senha , que também foi enviado pelo objeto Web .

Linha 5 – Importa o arquivo conecta.php , que é responsável pela abertura da conexão com o banco de dados.

Linha 6 – Armazena na variável \$sql a instrução que realiza o cadastro dos dados: nome, e-mail e senha na tabela usuarios no banco de dados.

Linha 7 – O comando mysqli\_query executa o cadastro do novo usuário no banco de dados. O retorno da execução do comando fica armazenado na variável \$resultado , sendo que o valor poderá ser verdadeiro (*true*) se a inclusão ocorreu sem problemas ou falso (*false*) se ocorreu algum erro no processo de cadastro.

Linha 8 – Verifica se a variável \$resultado possui o valor verdadeiro.

Linha 9 – Caso a inclusão tenha ocorrido com sucesso, realizamos uma nova consulta para armazenar no aplicativo o código do usuário. Preparamos a variável \$sql para realizar uma consulta ao banco de dados pesquisando pelo nome que foi digitado para cadastro.

Linha 10 – Executa o comando armazenado na variável \$sql e armazena o retorno das informações na variável \$resultado .

Linha 11 - O comando `mysqli_fetch_assoc` retornará o conjunto de resultados da linha da consulta.

Linha 12 – As informações que retornaram da consulta SQL serão armazenadas na variável `$dados` .

Linha 13 – Realiza a codificação para o formato JSON dos valores que estão na variável `$dados` e os envia para o aplicativo continuar com o processamento.

Linha 14 – Caso a variável `$resultado` não possua o valor verdadeiro, será executado o comando da linha 15.

Linha 15 – Será enviada para o app a informação `Não Salvo` .

Linha 16 - Encerramento da tag PHP.

## 13.6 SELECCIONA\_ITEM.PHP

A finalidade deste arquivo é exibir as informações complementares da corrida na Screen Detalhes. O app envia o `id` da corrida para consulta e o PHP executa a seleção dos dados e retorna ao app. Veja nos códigos a seguir o conteúdo do arquivo `seleciona_item.php` .

```
1.  <?php
2.    require "conecta.php";
3.    $ID = $_GET['id'];
4.    $sqlBusca = "select * from corridas where id=" . $ID;
5.    $resultado = mysqli_query($conexao,$sqlBusca);
6.    while ($array = mysqli_fetch_assoc($resultado)) {
7.      $dados[] = $array; }
8.    echo json_encode($dados[0]);
9.  ?>
```

Linha 1 – Linha de abertura padrão e obrigatória da tag PHP.

Linha 2 – Importa o arquivo `conecta.php`, que é responsável pela conexão com o banco de dados.

Linha 3 – O arquivo PHP recebe o `id` da corrida do objeto Web que foi enviado pelo app, e armazena-o na variável local `$ID`.

Linha 4 – A variável `$sqlBusca` armazena o comando SQL que realizará uma consulta pelo número da corrida que está indicado na variável `$ID` na tabela `corridas`.

Linha 5 - O comando `mysqli_query` executa o código SQL e armazena as informações da consulta na variável `$resultado`.

Linha 6 - O comando `mysqli_fetch_assoc` retornará o conjunto de informações da corrida selecionada.

Linha 7 – As informações que retornaram da consulta SQL serão armazenadas na variável `$dados`.

Linha 8 – Realiza a codificação para o formato JSON dos valores que estão na variável `$dados` e os envia para o aplicativo continuar o processamento.

Linha 9 - Encerramento da tag PHP.

## 13.7 SELECCIONA\_COMENTARIOS.PHP

Esse arquivo tem por finalidade selecionar todos os comentários da corrida para exibição na Screen Detalhes. A seguir são exibidos os códigos do arquivo `seleciona_comentarios.php`.

1.     `<?php`

```

2.      require "conecta.php";
3.      $corrida_id = $_GET['id'];
4.      $sql ="SELECT comentarios.user_id, comentarios.descricao,
   usuarios.id, usuarios.nome from comentarios INNER JOIN usuarios
   ON comentarios.user_id = usuarios.id where comentarios.corrida
   _id=$corrida_id";
5.      $resultado = mysqli_query($conexao,$sql);
6.      while ($array = mysqli_fetch_assoc($resultado)) {
7.          $dados[] = $array;
8.      echo json_encode($dados);
9.  ?>

```

Observe que esse arquivo é praticamente idêntico ao `seleciona_item.php`. Sua única diferença está na linha de número 4 onde é definida a instrução que seleciona todos os comentários e os nomes dos usuários que as realizaram. Todos esses dados estão na tabela `comentarios` e serão filtrados pelo número da corrida selecionada que foi enviada pelo app.

## 13.8 COMENTARIOS.PHP

O objetivo deste arquivo é cadastrar o comentário realizado pelo usuário na tabela `comentarios`, calcular a média das avaliações e realizar a atualização do valor da média das corridas na tabela `corridas`. Vejamos as linhas do arquivo `comentarios.php` a seguir.

```

1.  <?php
2.  require "conecta.php";
3.  $descricao =$_POST['descricao'];
4.  $avaliacao =$_POST['avaliacao'];
5.  $corrida_id =$_POST['corrida_id'];
6.  $user_id =$_POST['user_id'];
7.  $sql = "insert into comentarios (descricao, avaliacao, corr
   ida_id,user_id) values ('$descricao',$avaliacao,$corrida_id,$user
   _id')";
8.  $resultado = mysqli_query($conexao,$sql);
9.  if ($resultado) {

```

```
10.      echo "Avaliado com sucesso"; }
11.    else
12.      { echo "Não Salvo"; }
13.    $ID = $corrida_id;
14.    $sqlBusca = "select avg(avaliacao) as 'valor' from comentar
ios where corrida_id=" . $ID ;
15.    $resultado = mysqli_query($conexao,$sqlBusca);
16.    while($linha = mysqli_fetch_object($resultado))
17.      $media= $linha->valor;
18.    $sqlAtualiza = "UPDATE corridas SET media= $media where i
d=" . $ID ;
19.    $resultado = mysqli_query($conexao,$sqlAtualiza);
20.    ?>
```

Linha 1 – Linha de abertura padrão e obrigatória da tag PHP.

Linha 2 – Importa o arquivo conecta.php , que é responsável pela conexão com o banco de dados.

Linha 3 – Nesta linha armazenamos na variável local \$descricao o comentário realizado pelo usuário sobre a corrida e que foi enviada pelo app através do método POST do objeto Web .

Linha 4 – Recebe e armazena o número da avaliação fornecido pelo usuário no app.

Linha 5 – Recebe e armazena o número da corrida que está sendo avaliada ( id ).

Linha 6 – Recebe e armazena o número do usuário que está realizando a avaliação.

Linha 7 – A variável \$sql armazena uma instrução SQL que insere a descrição, avaliação, número da corrida e número do usuário na tabela comentarios .

Linha 8 - O comando mysqli\_query executa a instrução SQL

e realiza o cadastro do novo comentário no banco de dados. Como vimos anteriormente, o status da execução do comando fica armazenado na variável `$resultado`. O status poderá ser um valor verdadeiro caso a inclusão ocorra, ou falso, se algo de errado ocorreu.

Linha 9 – Verifica se a variável `$resultado` possui o valor verdadeiro.

Linha 10 – Caso a inclusão do comentário tenha ocorrido, será enviada para o app a informação `Avaliado com sucesso`.

Linha 11 – Comando que inicia a estrutura de negação do comando `if`.

Linha 12 – Caso a inclusão não tenha sido realizada, será enviada a mensagem `Não Salvo` para o aplicativo.

Linha 13 – Armazena o número da corrida na variável `$ID`.

Linha 14 – Definimos uma instrução SQL que calcula a média de todos os valores de avaliações dos usuários da corrida indicada pela variável `$ID` e que estão na tabela `comentarios`.

Linha 15 – O comando `mysqli_query` executa a instrução SQL calculando a média.

Linha 16 – O comando `mysqli_fetch_object` retorna o valor da média da corrida selecionada.

Linha 17 – A variável `$media` recebe o valor calculado da média para posterior atualização.

Linha 18 – A variável `$sqlAtualiza` armazena a instrução

SQL com o novo valor da média da corrida selecionada para atualização do registro.

Linha 19 – O comando `mysqli_query` executa a instrução SQL atualizando a média na tabela `corridas`.

Linha 20 – Encerramento da tag PHP.

## Conclusão

Este aplicativo é apenas uma sugestão de utilização e demonstração do poder que a plataforma de desenvolvimento Kodular para Android possui e pode realizar. Com base no app apresentado e desenvolvido no decorrer deste livro, muitos outros poderão surgir, bastando disposição e criatividade.

Esperamos que você possa desenvolver os seus próprios aplicativos e que seus usuários fiquem satisfeitos com o seu trabalho.

Bons estudos.