# RESUMO –

## Variáveis – “palavra\_palavra”

Elas seguem o padrão de utilizar apenas letras minúsculas sem acento, e em caso de duas ou mais palavras, utilizar de “\_” para separar as palavras. Segue o exemplo:

“name\_user”.

“password\_user”

“name\_category”

“icon\_comunity”

## Funções – “palavraVerbo”

Elas seguem o padrão de utilizar letra minúsculas na primeira palavra e na segunda que deve ser um verbo Letra maiúscula na primeira letra, sem acento. Segue o exemplo:

“userRegister”

“userOpenProfile”

“categoryCreate”

“categoryClick”

## Classes – “PalavraPalavra”

Elas seguem o padrão de utilizar maiúscula na primeira letra de cada palavra. Segue o exemplo:

“UserModel”

“CategoryModel”

“ComunityController”

“PlaceModel”

**SUMÁRIO**

[RESUMO – 1](#_Toc176463100)

[Variáveis – “palavra\_palavra” 1](#_Toc176463101)

[Funções – “palavraVerbo” 1](#_Toc176463102)

[Classes – “PalavraPalavra” 1](#_Toc176463103)

[1. API – 4](#_Toc176463104)

[1.1. Variáveis – 4](#_Toc176463105)

[1.2. Funções – 5](#_Toc176463106)

[1.3. Classes – 6](#_Toc176463107)

[2. APLICAÇÃO 7](#_Toc176463108)

[2.1. CLASSES 7](#_Toc176463109)

[2.1.1. LoginActivity.kt 7](#_Toc176463110)

[2.1.3. InitialActivity.kt 7](#_Toc176463111)

[2.1.4. SplashActivity.kt 7](#_Toc176463112)

[2.1.5. CommonEvents.kt 7](#_Toc176463113)

[2.1.6 UserModel.kt 7](#_Toc176463114)

[2.1.7. ApiCliente.kt 7](#_Toc176463115)

[2.1.8 EndpointUser.kt 7](#_Toc176463116)

[2.2. VÁRIAVEIS 7](#_Toc176463117)

[2.2.1. LoginActivity.kt 7](#_Toc176463118)

[2.2.2. RegisterActivity.kt 8](#_Toc176463119)

[2.2.6. UserModel.kt 9](#_Toc176463120)

[2.2.7. ApiCliente.kt 9](#_Toc176463121)

[2.3. FUNÇÕES 9](#_Toc176463122)

[2.3.1. LoginActivity.kt 9](#_Toc176463123)

[2.3.2. RegisterActivity.kt 10](#_Toc176463124)

[2.3.3. CommonEvents 10](#_Toc176463125)

[2.3.4 EndpointUser.kt 10](#_Toc176463126)

# 1. API –

## 1.1. Variáveis –

Aquelas que representem dados de tabelas do banco, terão o mesmo nome dos campos que representam, como por exemplo: “name\_user”; “description\_category”. Em seu nome já há “descrição” do que faz no código.

**context.** Relativo à conexão com o banco de dados, servindo como um array formado por todas as tabelas do banco.

**check\_email.** Váriavel booleana referente ao retorno de uma função que verifica se o email digitado já está inserido no banco de dados.

**check\_login.** Váriavel booleana referente ao retorno de uma função que verifica se o login, ou seja, o @ digitado já está inserido no banco de dados.

**get\_users.** Guarda todos os objetos “User” retornados na função para puxar os usuários do banco de dados.

**categories.** Integra a tabela categories do banco de dados com a classe CategoryModel.

**users.** Integra a tabela users do banco, com a classe UserModel.

**cities.** Integra a tabela cities do banco, com a classe CityModel.

**comunities.** Integra a tabela comunities do banco, com a classe ComunityModel.

## 1.2. Funções –

**userCreate.** Retorna um usuário novo para o banco de dados com base nos parâmetros inseridos.

**userGet.** Retorna todos os usuários existentes no banco de dados.

**userGetID.**  Retorna apenas um usuário, aquele com o id do parâmetro.

**userChange.** Altera todos os dados de um usuário especificado pelo id.

**userDelete.** Desativa o usuário especificado.

**categoryCreate.** Retorna uma categoria nova para o banco de dados com base nos parâmetros inseridos.

**categoryGet.** Retorna todos os usuários existentes no banco de dados.

**categoryGetID.** Retorna uma categoria, aquela com o id do parâmetro.

**categoryChange.** Altera todos os dados de uma categoria especificado pelo id.

**categoryDelete.** Deleta uma categoria do banco de dados.

**cityGet.** Retorna todos as cidades existentes no banco de dados.

**cityGetID.** Retorna apenas uma cidade, aquele com o id do parâmetro.

## 1.3. Classes –

**ConnectionContext.** Integra o banco de dados ao programa, neste caso, a API

**CategoryController.** Classe que gera as URLs para manipulação da tabela categories do banco.

**CityController.** Classe que gera as URLs para manipulação da tabela cities do banco.

**ComunityController.** Classe que gera as URLs para manipulação da tabela comunities do banco.

**UserController.** Classe que gera as URLs para manipulação da users categories do banco.

**CategoryModel.** Classe para instanciar uma linha da tabela categories do banco.

**CityModel.** Classe para instanciar uma linha da tabela cities do banco.

**ComunityModel.** Classe para instanciar uma linha da tabela comunities do banco.

**UserModel.** Classe para instanciar uma linha da tabela users do banco.

# 2. APLICAÇÃO

## **2.1. CLASSES**

### 2.1.1. LoginActivity.kt

Classe para a tela de login.

### 2.1.2. RegisterActivity.kt

Classe para a tela de registro.

### 2.1.3. InitialActivity.kt

Classe para a tela de início.

### 2.1.4. SplashActivity.kt

Classe para a tela de splash.

### 2.1.5. CommonEvents.kt

Classe para agrupar funções personalizadas comuns entre os componentes visuais. Utilizando para redefinir certas funções padronizadas dos componentes como os componentes EditText, TextView, AppCompatButton e etc.

### 2.1.6 UserModel.kt

Classe para receber os dados do usuário como atributos e reutilizar o objeto instanciado no resto das telas, para assim puxar os dados do banco de dados com base nos atributos já armazenados.

### 2.1.7. ApiCliente.kt

Classe para vincular a API com a tela em que for instanciada.

### 2.1.8 EndpointUser.kt

Interface que define o endpoint de users da api, ou seja, vincula com a aplicação todas as rotas da parte de users da API.

## **2.2. VÁRIAVEIS**

### 2.2.1. LoginActivity.kt

1. **binding:** Tipo: ActivityLoginBinding. Objeto da classe ActivityLoginBinding. Utilizado para agrupar e referenciar todos os componentes visuais dentro do layout da LoginActivity.
2. **common\_events:** Tipo: CommonEvents. Objeto da classe CommonEvents.
3. **btn\_login:** Tipo: AppCompatButton. Objeto da classe AppCompatButton. Utilizado para guardar o componente btn\_login do layout da LoginActivity.
4. **et\_username:** Tipo: EditText. Objeto da classe EditText. Utilizado para guardar o componente et\_username do layout da LoginActivity.
5. **et\_password:** Tipo: EditText. Objeto da classe EditText. Utilizado para guardar o componente et\_password do layout da LoginActivity.
6. **api\_user:** Tipo: Endpoint. Objeto da classe personalizada Endpoint. Utilizado para chamar os Endpoints(URL’s) relacionadas a parte de usuários da API.

### 2.2.2. RegisterActivity.kt

1. **binding:** Tipo: ActivityRegisterBinding. Objeto da classe ActivityRegisterBinding. Utilizado para agrupar e referenciar todos os componentes visuais dentro do layout da RegisterActivity.
2. **common\_events:** Tipo: CommonEvents. Objeto da classe CommonEvents.
3. **btn\_continue:** Tipo: AppCompatButton. Objeto da classe AppCompatButton. Utilizado para guardar o componente btn\_login do layout da LoginActivity.
4. **et\_username:** Tipo: EditText. Objeto da classe EditText. Utilizado para guardar o componente et\_username do layout da RegisterActivity.
5. **et\_password:** Tipo: EditText. Objeto da classe EditText. Utilizado para guardar o componente et\_password do layout da RegisterActivity.
6. **api\_user:** Tipo: Endpoint. Objeto da classe personalizada Endpoint. Utilizado para chamar os Endpoints(URL’s) relacionadas a parte de usuários da API.

### 2.2.6. UserModel.kt

1. **id\_user:** Tipo: Int. Atributo da classe usuário que deve receber como valor, o id\_user do usuário no banco de dados.
2. **name\_user:** Tipo: String. Atributo da classe usuário que deve receber como valor, o name\_user do usuário no banco de dados.
3. **activate\_user:** Tipo: Boolean. Atributo da classe usuário que deve receber como valor, o activate\_user do usuário no banco de dados.
4. **email\_user:** Tipo: String. Atributo da classe usuário que deve receber como valor, o email\_user do usuário no banco de dados.
5. **password\_user:** Tipo: String. Atributo da classe usuário que deve receber como valor, o password\_user do usuário no banco de dados.
6. **login\_user:** Tipo: String. Atributo da classe usuário que deve receber como valor, o login\_user do usuário no banco de dados.
7. **profile\_picture\_user:** Tipo: String. Atributo da classe usuário que deve receber como valor, o profile\_picture\_user do usuário no banco de dados.

### 2.2.7. ApiCliente.kt

1. **URL:** Tipo: String. Atributo para definir a rota da API.

## **2.3. FUNÇÕES**

### 2.3.1. LoginActivity.kt

1. **componentsInitialize():** retorna vazio. Inicializa as variáveis da classe com seus respectivos valores. E altera seus eventos, a partir do objeto common\_events.
2. **enterClick():** retorna vazio. Verifica se os campos et\_username e et\_password estão vazios. Caso não estejam, chama a função userLogar().
3. **userLogar().** Retorna vazio. Verifica.

### 2.3.2. RegisterActivity.kt

1. **componentsInitialize():** retorna vazio. Inicializa as variáveis da classe com seus respectivos valores. E altera seus eventos, a partir do objeto common\_events.

### 2.3.3. CommonEvents

1. **touchListenerGet(button : AppCompatButton):** retorna a função personalizada OnTouchListener da classe View. Recebe como parâmetro um objeto da classe AppCompatButton. Verifica se o botão dado como parâmetro está sendo clicado. Se sim, alterna a cor do texto do botão para a variável laranja\_splash, e altera o shape do background para o edit\_text\_selected na pasta Drawnable. Se não, altera tudo para o padrão normal.
2. **focusChangedListenerGet(edit\_text : EditText):** retorna a função personalizada OnFocusChangeListener da classe View. Recebe como parâmetro um objeto da classe EditText. Verfica se o objeto passado como parâmetro está em foco(se o usuário clicou nele). Se sim, altera o shape para o edit\_text\_selected. Se não, altera tudo para o padrão.

### 2.3.4 EndpointUser.kt

1. **userLogin(login\_user : String, password\_user : String):** retorna uma Call(vou pesquisar oq é)