UNIVERSIDADE NOVE DE JULHO
DIRETORIA DOS CURSOS DE INFORMÁTICA

Caroline Paiva Toledo - RA 918202161

David Cunha dos Santos - RA 918207779

Edivan Pereira dos Santos - RA 918209251

Guilherme Henrique Canato - RA 918207807

Marcelo da Silva Felix - RA 918210890

Mauricio Martinelli - RA 919105260

Rafael de Moraes Bonfim - RA 918111598

Rafaela dos Santos Silva - RA 918208187

Rogério de Oliveira - RA 918205762

PROJETO PRÁTICO EM SISTEMAS

Criative S.A

SÃO PAULO
2020 – 4º SEMESTRE

Caroline Paiva Toledo - RA 918202161

David Cunha dos Santos - RA 918207779

Edivan Pereira dos Santos - RA 918209251

Guilherme Henrique Canato - RA 918207807

Marcelo da Silva Felix - RA 918210890

Mauricio Martinelli - RA 919105260

Rafael de Moraes Bonfim - RA 918111598

Rafaela dos Santos Silva - RA 918208187

Rogério de Oliveira - RA 918205762

PROJETO PRÁTICO EM SISTEMAS

Criative S.A

Trabalho apresentado à Universidade Nove de Julho, UNINOVE, em cumprimento parcial às exigências da disciplina de Projeto Prático em Sistemas, sob orientação do Prof. **Leandro Fernandes da Mota.**

SÃO PAULO
2020 – 4º SEMESTRE

ROTEIRO DO PROJETO

Para o desenvolvimento do Projeto é proposto a criação de um Sistema Móvel (APP Android), utilizando os conceitos aprendidos nas disciplinas de Sistemas Móveis (Android).

Os conceitos desenvolvidos nas disciplinas devem ser empregados em sua completude, ou seja, devem estar presentes em todas as etapas do projeto.

O projeto não restringe a utilização de outras tecnologias, mesmo que não tenham sido abordadas no curso.

RESUMO

Este aplicativo mostrará a interação do cliente com a empresa Criative S.A.

Palavras-chave: aplicativos móveis, sites, android.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Com o crescente aumento da tecnologia e na rapidez que as empresa propõem para o cliente agilidade no atendimento, os aplicativos moveis são uma facilidade no acesso na ponta dos dedos, onde e suprida as necessidades do usuário para ter um aumento significativo de melhor performance no atendimento e criação de sites, caso o usuário esteja com muita presa irá solicitar sua criação pelo aplicativo da empresa onde será adquirido pelas plataformas de aplicações.

1.1. MOTIVAÇÕES E OBJETIVO

O objetivo desse software de comércio de criação de site é reunir as ferramentas de que uma empresa precisa para cuidar dos seus clientes desde a pré-criação até a pós-criação e entre diferentes canais de comunicação e distribuição.

1.2. DESCRIÇÃO DO PRODUTO

Permitirá realizar solicitação de criação de um site personalizados, fornecendo controle ao cliente sobre pedidos realizadas, pedidos solicitados, bem como para os administradores da plataforma no que diz respeito ao controle de pedidos e solicitações.

1.3. PREMISSAS

Tivemos muita dificuldade ao criar o app, na ferramenta android studio, pois não tínhamos noção de como inicializar uma app, vimos vidões que nos auxiliou para o desenvolvimento do app, nos baseando no CRUD onde pedimos ajuda também para os professores Edson, Leandro e vídeo do professor Luís.

1.4. RECURSOS

Os recurso utilizados estão instalado no computador de cada integrante da equipe o Android Studio: https://developer.android.com/studio, e vídeo aulas de aplicações android e ajuda de professores qualificados, como o professor Edson Melo de Souza e o professor Luís Carlos dos Santos Junior ambos professores da Uninove.

1.5. DEFINIÇÃO DO NEGÓCIO

Realizará uma criação de site personalizados atendendo a necessidade de cada clientes da melhor forma possível. Onde o cliente verá no aplicativo os status de sua solicitação, a forma como ele quer o site, e a data de entrega do site.

1.6. DEFINIÇÃO DA EQUIPE

Caroline Paiva Toledo - RA 918202161 email: c.toledo@uni9.edu.br

David Cunha dos Santos - RA 918207779 email: davidcunhasantos@uni9.edu.br

Edivan Pereira dos Santos - RA 918209251 email:edivan.santos@uni9.edu.br

Guilherme Henrique Canato - RA 918207807 email: canatogui@uni9.edu.br

Marcelo da Silva Felix - RA 918210890 email: marcelo.felix@uni9.edu.br

Mauricio Martinelli - RA 919105260 email: mauricio.martinelli@uni9.edu.br

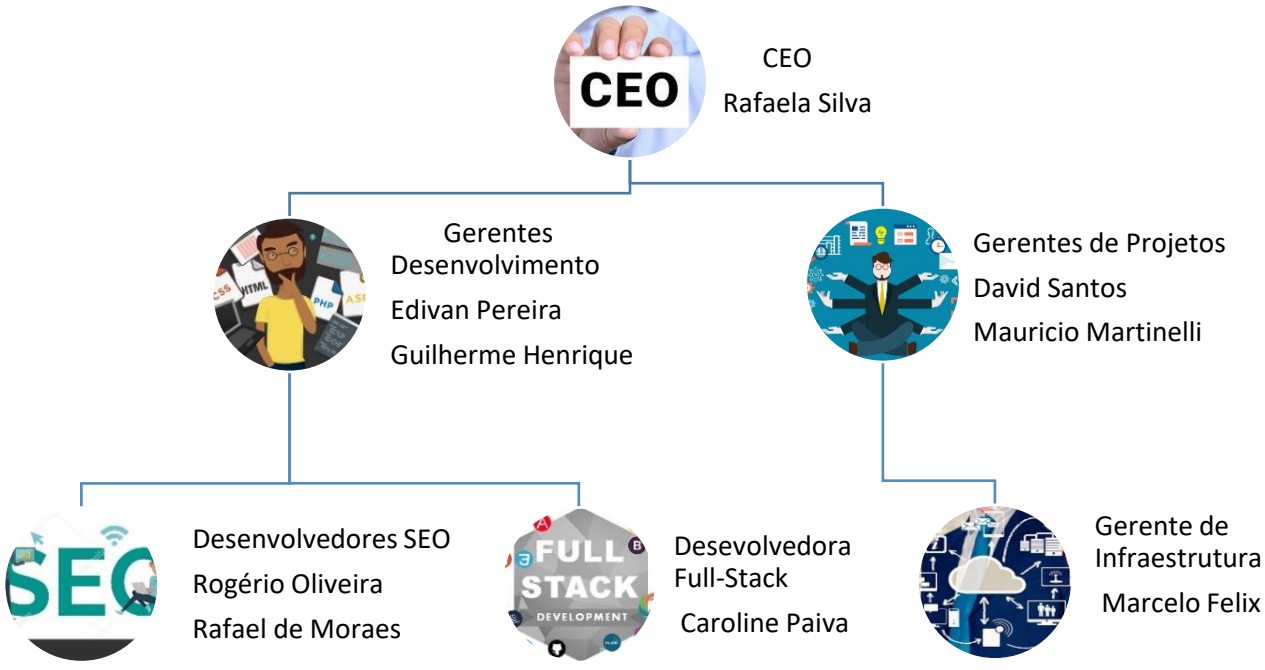
Rafael de Moraes Bonfim - RA 918111598 email: moraesbonfim@uni9.edu.br

Rafaela dos Santos Silva - RA 918208187 email: rafpromais@uni9.edu.br

Rogério de Oliveira - RA 918205762 email: rogerio.oliveira@uni9.edu.br

1.6.1. ORGANOGRAMA

FIGURA 1: ORGANOGRAMA.



2. Descrição do Sistema

2.1. Descrição Detalhada das Partes que Compõe o Sistema

O Crud Criative S.A com base no site de criação de sites vem com a finalidade de facilitar a inserção e manipulação de dados no sistema. Com Crud é possível inserir editar e excluir os dados que compõe no sistema.

2.1.1. Página Inicial

Na página inicial o Administrador encontrar 3 botões Inserir, Listar, Pesquisar, podendo administrar todos clientes e desenvolvedores.

2.1.2. Pesquisa

Depois de inserido é possível pesquisar os clientes editar excluir manipulando os dados como administrador da plataforma.

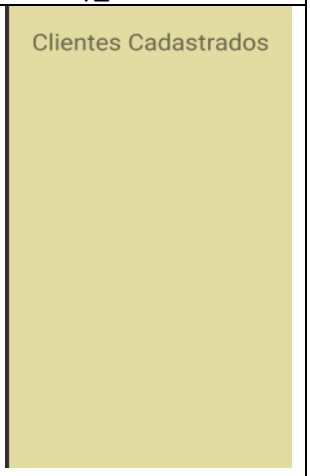
2.2. Requisitos Funcionais

O Software permite que o administrador do sistema possa fazer alterações como *Inserir, editar e Excluir* dados da plataforma.

2.3. Wireframes

É um guia visual usado em design de interface do android studio onde será mostrado os layouts do app.

Figura 2: Layouts do app no android studio.

<table>
<tr><td>Activity_spash</td><td>Activity_main</td><td>Activity_insert</td><td>Activity_list</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>Activity_details</td><td>Activity_edit</td><td>Activity_excluir</td><td>Activity_search</td></tr>
</table>


Detalhes do Cliente
ID:
CPF:
Telefone:
Nome:
E-mail:
Descrição:
EXCLUIR
EDITAR
Dados Clientes
ID:
CPF:
Telefone:
Nome:
E-mail:
Descrição:
SALVAR
Exclusão de Dados
ID:
CPF:
Telefone:
Nome:
E-mail:
Descrição:
EXCLUIR
Pesquisa por CPF
Digite o CPF
PESQUISAR

3. MODELAGEM UML

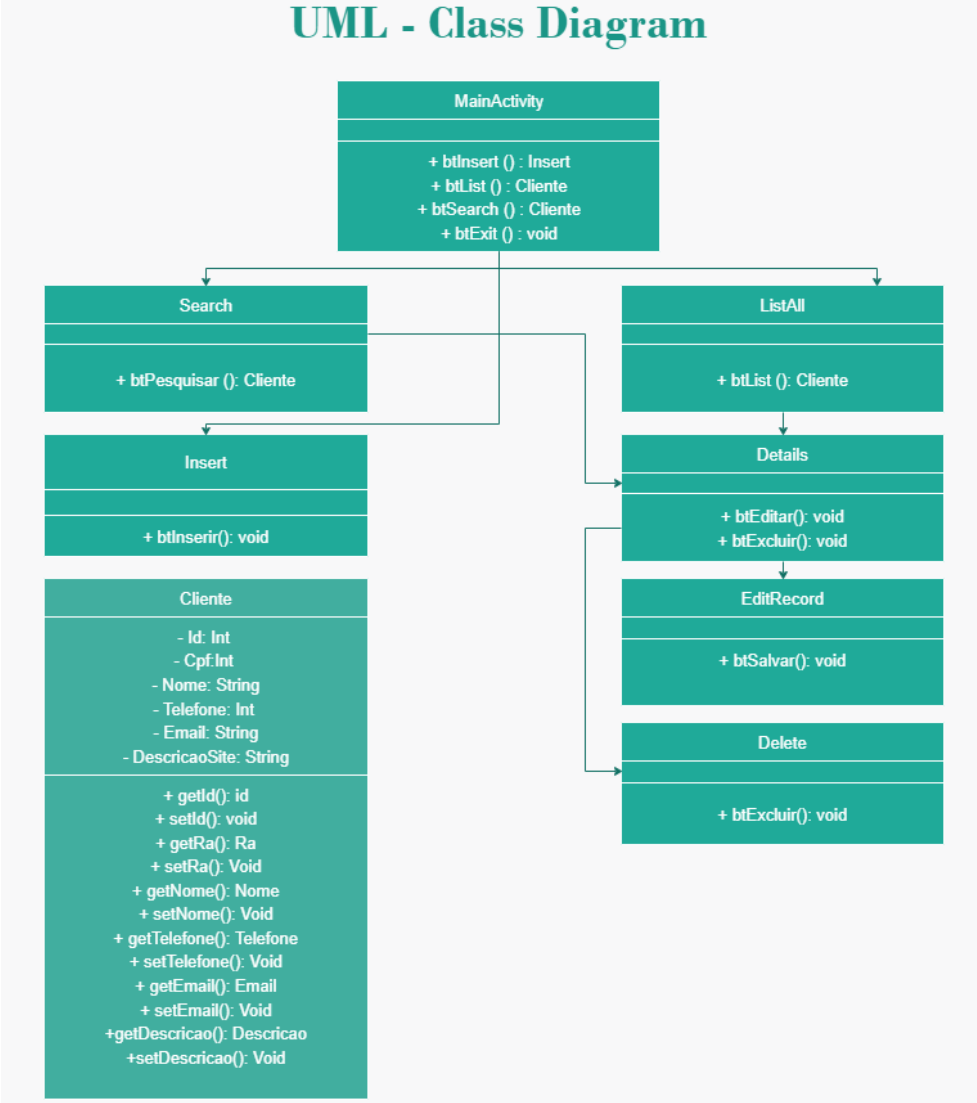
3.1. DIAGRAMAS DE CLASSES

Nesse tópico iremos detalhar as funcionalidades e estrutura das classes existentes dentro do aplicativo e a sua modelagem.

Iremos utilizar a UML por ser o modo mais utilizado para representar essa organização. Através dos Diagramas de Classe conseguimos demonstrar os relacionamentos existentes dentro do sistema. O diagrama mostra apenas uma visão estática do sistema.

A seguir temos a modelagem das classes existente dentro do sistema para demonstrar a estrutura do aplicativo e os relacionamentos.

FIGURA 3: DIAGRAMA DE CLASSE.



Classe “MainActivity” ela é a principal e responsável por chamar as outras classes de acordo com a opção selecionada pelo cliente quando iniciar o aplicativo. Entre as opções irão existir as opções de “Inserir”, “Pesquisar” e “Listar”.

Classe “Insert” responsável por fazer o cadastro dos dados do cliente conforme indicado nos campos pelo aplicativo. Os dados informados serão os mesmos dos atributos da Classe “Cliente”. Após a inserção os dos dados eles são gravados no banco de dados.

Classe “Search” é responsável por fazer a pesquisa de um cliente no banco de dados a partir do CPF e após a pesquisa ele poderá fazer a edição dos dados ou até mesmo excluir seu cadastro.

Classe “ListAll” é responsável por listar todos os clientes que realizaram o cadastro através do aplicativo. Assim sendo possível acessar um dos cadastros e então editar ou excluir os dados.

Classe “Details” ela é responsável por mostrar os dados após a busca feita através das classes “Search” ou “ListAll” e assim é possível editar ou excluir os dados exibidos.

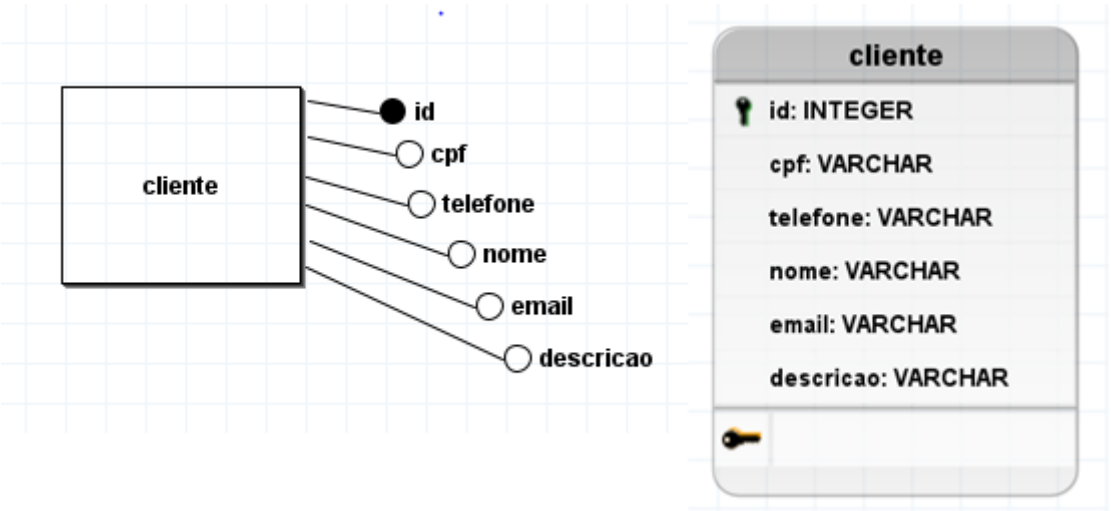
Classe “EditRecord” é chamada após a seleção de uma das opções na classe “Details”, nesse caso a opção de editar os dados. A classe é responsável por editar os dados que já estão gravados na base de dados.

Classe “Excluir” também é chamada após a seleção de uma das opções na classe “Details”, nesse caso a opção de excluir. Essa classe é responsável por excluir os dados na base de dado

4. MODELAGEM DO BANCO DE DADOS

4.1. DIAGRAMA E-R

FIGURA 4: DIAGRAMA ENTIDADE-RELACIONAMENTO.



4.2. IMPLEMENTAÇÃO FÍSICA

Para a criação do banco de dados, utilizamos os códigos:

```
db = openOrCreateDatabase("db_cliente", Context.MODE_PRIVATE, null);.
```

Para a criação da Tabela, foi feito o seguinte código e lógica.

Crie a tabela se não existir, se já possui a tabela, carregue-a para uso.

```
db.execSQL("CREATE TABLE IF NOT EXISTS cliente(" +
            "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "cpf VARCHAR NOT NULL," +
            "telefone VARCHAR NOT NULL, " +
            "nome VARCHAR NOT NULL, " +
            "email VARCHAR NOT NULL, " +
            "descricao VARCHAR NOT NULL);");
```

Para a inserção de dados, utilizamos a seguinte lógica:

Coletamos os dados digitados nos campos abaixo, com um Objeto(value) que armazena o conteúdo.

```
ContentValues values = new ContentValues();
            values.put("cpf", cliente.getCpf());
            values.put("telefone", cliente.getTelefone());
            values.put("nome", cliente.getNome());
```

```
values.put("email", cliente.getEmail());
values.put("descricao", cliente.getDescricao());
```

Após isso utilizamos o comando:

```
db.insert("cliente", null, values);
```

Para fazer a busca de todos(ALL), utilizamos o comando:

```
db.rawQuery("SELECT * FROM cliente ORDER BY nome ASC", null);
```

Para fazer a busca de apenas um único registro, utilizamos o CPF como parâmetro de busca:

```
db.rawQuery("SELECT * FROM cliente WHERE cpf=?", new
String[ ]{cpf.getText().toString()});
```

Também fizemos o comando de Editar, caso o Cliente precise alterar algum dado do cadastro e o Excluir, caso queira deletar o seu cadastro.

Editar: usamos a mesma lógica de inserir, assim pegamos os dados digitados nos campos através do Objeto (Values) e damos o UPDATE na tabela.

```
db.execSQL("UPDATE cliente SET " +
                "cpf='" + novosDados.getCpf() + "'," +
                "telefone='" + novosDados.getTelefone() + "'," +
                "nome='" + novosDados.getNome() + "'," +
                "email='" + novosDados.getEmail() + "'," +
                "descricao='" + novosDados.getDescricao() + "' " +
                "WHERE id=" + cliente.getId()  );
```

EXCLUIR: Para a exclusão, utilizamos o ID como parâmetro.

```
db.execSQL("DELETE FROM cliente WHERE id=" + cliente.getId());
```

5. METODOLOGIA

5.1. DESENVOLVIMENTO

Para desenvolver o aplicativo foi necessário preparar o ambiente de desenvolvimento que conta com alguns softwares voltados para a criação desse tipo de aplicação. Os softwares utilizados foram:

- JDK (versão 8): O JDK é um ambiente de desenvolvimento para a criação de aplicativos, applets, e componentes usando a linguagem de programação Java.
- Android Studio: O Android Studio é um ambiente de desenvolvimento integrado oficial da Google para criar aplicações Android. Ele conta com diversos componentes que dão suporte ao desenvolvimento de aplicativos, tanto para smartphones e tablets, quanto para Android Wear e outras plataformas.

5.2. ESTRUTURA DO SISTEMA

Tomamos como base um CRUD pronto para adaptarmos ao nosso projeto e processo da empresa Criative. Como se trata de um CRUD, essencialmente as classes Details, EditRecord, Excluir, Insert, ListAll e Search fazem o processo, descrito a seguir:

a. **Classe Details:** utilizada para apresentar os detalhes de cada registro que foi apresentado (Nome, Telefone, CPF, Descrição).
b. **Classe EditRecord:** utilizada para editar o registro selecionado, e atualizar assim o banco de dados.
c. **Classe Excluir**: **:** utilizada para excluir o registro selecionado, e atualizar assim o banco de dados.
d. **Classe Insert:** utilizada para inserir um novo registro no banco de dados.
e. **Classe ListAll:** utilizada para apresentar todos os registros disponíveis no banco de dados.

f. **Classe Search:** utilizada para buscar um registro específico por CPF no banco de dados, possibilitando assim a visão dos detalhes do registro e a edição dele.

Esse processo foi feito observando diretamente um modelo já disponibilizado, pois dessa maneira sabemos qual é a estrutura correta devemos seguir para a criação do aplicativo baseado em nosso processo.

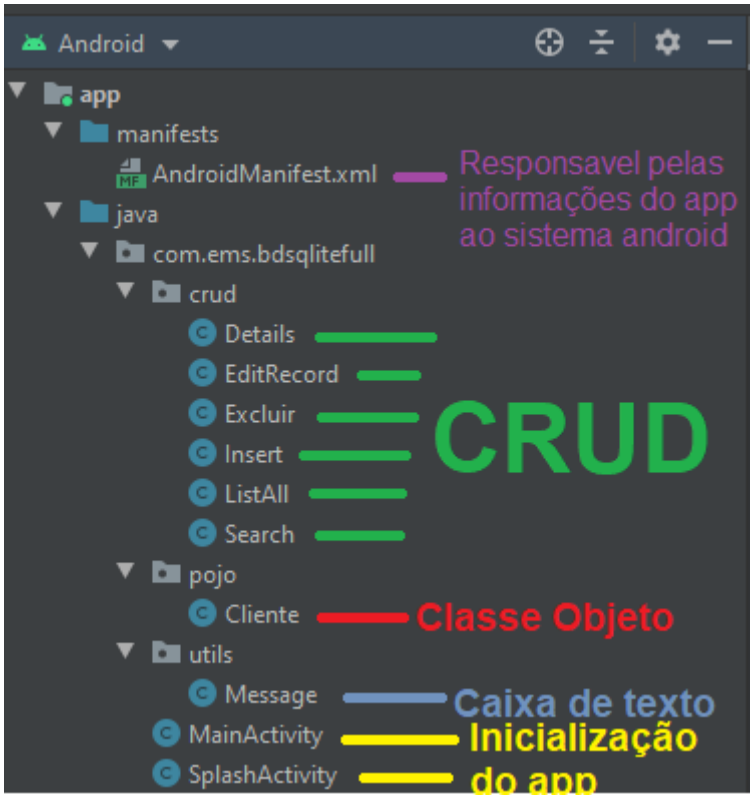
O link abaixo apresenta todo o código do aplicativo feito:
https://github.com/Rafyy2102/Projeto_faculdade_App

6. ARQUITETURA DE SOFTWARE

6.1. DESENVOLVIMENTO

O codigo fonte do projeto pode ser dividido em 3 grupos nos quais todas as funcionalidades estão divididas por responsabilidades. Ao passo que as estruturas vão sendo explicadas, alguns trechos de códigos serão mostrados para demonstrar como algumas funcionalidades destas estruturas funcionam para um melhor entendimento.

FIGURA 5: ESTRUTURA DE ARQUIVOS.



Inicialização do app: responsável por iniciar a aplicativo quando é instalado no celular, esta inicialização tem 2 atividades:

Figura 6: Inicialização da atividade da tela splash.

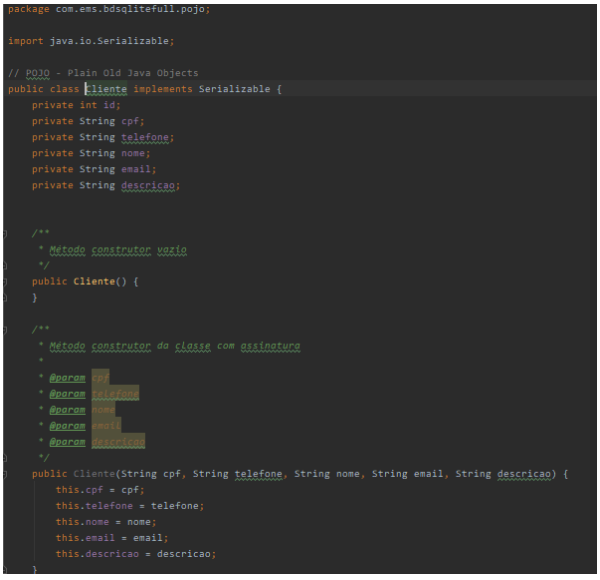
```java
package com.ems.bdsqlitefull;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;

public class SplashActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        new Handler().postDelayed(() -> {
                //Esse metodo será executado para a tela de splash e inicializando a tela de MainActivity
                //chamar indica a tela que esta e qual vai ser chamada.
                Intent i = new Intent( packageContext: SplashActivity.this, MainActivity.class);
                startActivity(i);

                //encerrando a tela splash
                finish();
        }, delayMillis: 2000);

    }
}
```

Figura 7: Inicialização da atividade da tela Main.

```java
public class MainActivity extends AppCompatActivity {

    // Declaração dos botões
    Button btInsert, btList, btSearch, btExit;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Associa o botão de inserção e configura o evento do clique para abrir a tela de inclusão
        btInsert = findViewById(R.id.btMainInsert);
        btInsert.setOnClickListener((v) -> {
                Intent insert = new Intent(getApplicationContext(), Insert.class);
                startActivity(insert);
        });

        // Associa o botão e configura a ação para abre a tela de buscas
        btList = findViewById(R.id.btMainList);
        btList.setOnClickListener((v) -> {
                Intent insert = new Intent(getApplicationContext(), ListAll.class);
                startActivity(insert);
        });

        // Associa o botão pesquisa a ação para abre a tela de pesquisa
        btSearch = findViewById(R.id.btMainSearch);
        btSearch.setOnClickListener((v) -> {
                Intent insert = new Intent(getApplicationContext(), Search.class);
                startActivity(insert);
        });

        // Associa e configura o botão para sair da aplicação
        btExit = findViewById(R.id.btMainExit);
        btExit.setOnClickListener((v) -> {
                // Finaliza a aplicação e remove da pilha
                finishAffinity();
```

Classe objeto: responsável pelos atributos getters e setters:

FIGURA 8: CLASSE CLIENTE.

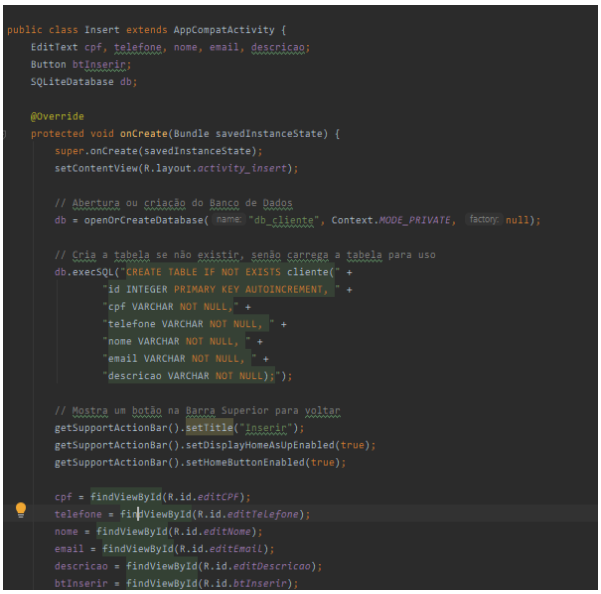


```java
package com.ems.bdsqlitefull.pojo;

import java.io.Serializable;

// POJO - Plain Old Java Objects
public class Cliente implements Serializable {
    private int id;
    private String cpf;
    private String telefone;
    private String nome;
    private String email;
    private String descricao;


    /**
     * Método construtor vazio
     */
    public Cliente() {
    }

    /**
     * Método construtor da classe com assinatura
     *
     * @param cpf
     * @param telefone
     * @param nome
     * @param email
     * @param descricao
     */
    public Cliente(String cpf, String telefone, String nome, String email, String descricao) {
        this.cpf = cpf;
        this.telefone = telefone;
        this.nome = nome;
        this.email = email;
        this.descricao = descricao;
    }
```

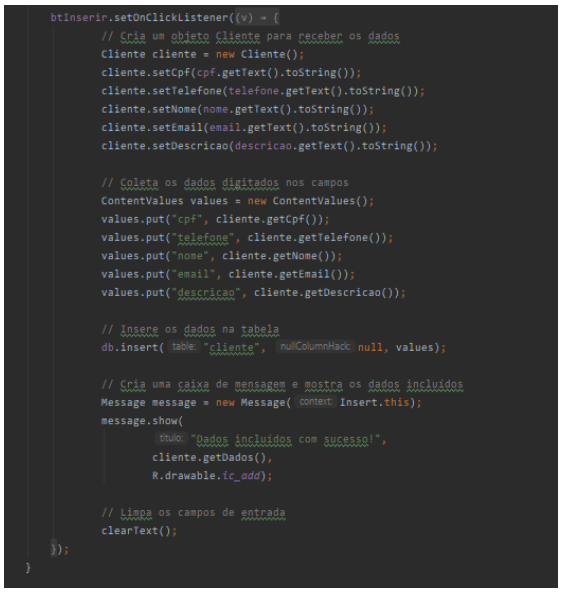
```java
// Getters and Setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getCpf() { return cpf; }

public void setCpf(String cpf) { this.cpf = cpf; }

public String getTelefone() { return telefone; }

public void setTelefone(String telefone) { this.telefone = telefone; }

public String getNome() { return nome; }

public void setNome(String nome) { this.nome = nome; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public String getDescricao() { return descricao; }

public void setDescricao(String descricao) { this.descricao = descricao; }


/**
 * Método sobrescrito para retornar o nome do aluno na ListView
 *
 * @return
 */
@Override
public String toString() { return nome; }
```

CRUD: responsável por Create (Criação), Read (Consulta), Update (Atualização) e Delete (Destruição).

FIGURA 9: CREATE (CRIAÇÃO) TELA DE INSERT.



```java
public class Insert extends AppCompatActivity {
    EditText cpf, telefone, nome, email, descricao;
    Button btInserir;
    SQLiteDatabase db;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_insert);

        // Abertura ou criação do Banco de Dados
        db = openOrCreateDatabase( name: "db_cliente", Context.MODE_PRIVATE,  factory: null);

        // Cria a tabela se não existir, senão carrega a tabela para uso
        db.execSQL("CREATE TABLE IF NOT EXISTS cliente(" +
                "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "cpf VARCHAR NOT NULL," +
                "telefone VARCHAR NOT NULL, " +
                "nome VARCHAR NOT NULL, " +
                "email VARCHAR NOT NULL, " +
                "descricao VARCHAR NOT NULL);");

        // Mostra um botão na Barra Superior para voltar
        getSupportActionBar().setTitle("Inserir");
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setHomeButtonEnabled(true);

        cpf = findViewById(R.id.editCPF);
        telefone = findViewById(R.id.editTelefone);
        nome = findViewById(R.id.editNome);
        email = findViewById(R.id.editEmail);
        descricao = findViewById(R.id.editDescricao);
        btInserir = findViewById(R.id.btInserir);
```



```java
    btInserir.setOnClickListener((v) -> {
            // Cria um objeto Cliente para receber os dados
            Cliente cliente = new Cliente();
            cliente.setCpf(cpf.getText().toString());
            cliente.setTelefone(telefone.getText().toString());
            cliente.setNome(nome.getText().toString());
            cliente.setEmail(email.getText().toString());
            cliente.setDescricao(descricao.getText().toString());

            // Coleta os dados digitados nos campos
            ContentValues values = new ContentValues();
            values.put("cpf", cliente.getCpf());
            values.put("telefone", cliente.getTelefone());
            values.put("nome", cliente.getNome());
            values.put("email", cliente.getEmail());
            values.put("descricao", cliente.getDescricao());

            // Insere os dados na tabela
            db.insert( table: "cliente",  nullColumnHack: null, values);

            // Cria uma caixa de mensagem e mostra os dados incluídos
            Message message = new Message( context: Insert.this);
            message.show(
                    titulo: "Dados incluídos com sucesso!",
                    cliente.getDados(),
                    R.drawable.ic_add);

            // Limpa os campos de entrada
            clearText();
    });
}
```

FIGURA 10: READ (CONSULTA) TELAS DE SEARCH E DETAILS.

A tela Search é uma consulta específica, por CPF do cliente.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_search);

    // Mostra um botão na Barra Superior para voltar
    getSupportActionBar().setTitle("Pesquisa");
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    getSupportActionBar().setHomeButtonEnabled(true);

    cpf = findViewById(R.id.pesquisaCPF);
    btnPesquisar = findViewById(R.id.btnPesquisar);
    btnPesquisar.setOnClickListener((v) -> {
            db = openOrCreateDatabase( name: "db_cliente", Context.MODE_PRIVATE, factory: null);

            // Alterei o modo de realização da consulta, como havia falado e ontem e funcionou.
            // Pode ser algo no Android Studio que estava sendo executado, pois tive que baixar uns addons diferentes
            // Enfim, está funcionando.
            Cursor c = db.rawQuery( sql: "SELECT * FROM cliente WHERE cpf=?", new String[]{cpf.getText().toString()});
            while (c.moveToNext()) {
                cliente = new Cliente(
                        c.getInt( columnIndex: 0),
                        c.getString( columnIndex: 1),
                        c.getString( columnIndex: 2),
                        c.getString( columnIndex: 3),
                        c.getString( columnIndex: 4),
                        c.getString( columnIndex: 5));
            }

            Intent search = new Intent(getApplicationContext(), Details.class);
            search.putExtra( name: "objCliente", cliente);
            startActivity(search);
    });
}
```

A tela Details consulta uma listagem de clientes

```
public class Details extends AppCompatActivity {
    Button btEditar;
    TextView id, cpf, telefone,  nome, email, descricao;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);

        // Mostra um botão na Barra Superior para voltar
        getSupportActionBar().setTitle("Detalhes");
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setHomeButtonEnabled(true);

        id = findViewById(R.id.id);
        cpf = findViewById(R.id.cpf);
        telefone = findViewById(R.id.telefone);
        nome = findViewById(R.id.nome);
        email = findViewById(R.id.email);
        descricao = findViewById(R.id.descricao);
        btEditar = findViewById(R.id.btSalvar);

        Intent itCliente = getIntent();
        final Cliente cliente = (Cliente) itCliente.getExtras().getSerializable( key: "objCliente");
        id.setText(String.valueOf(cliente.getId()));
        cpf.setText(cliente.getCpf());
        telefone.setText(cliente.getTelefone());
        nome.setText(cliente.getNome());
        email.setText(cliente.getEmail());
        descricao.setText(cliente.getDescricao());

        btEditar.setOnClickListener((v) -> {
                Intent editar = new Intent(getApplicationContext(), EditRecord.class);
                editar.putExtra( name: "objCliente", cliente);
                startActivity(editar);
        });
```

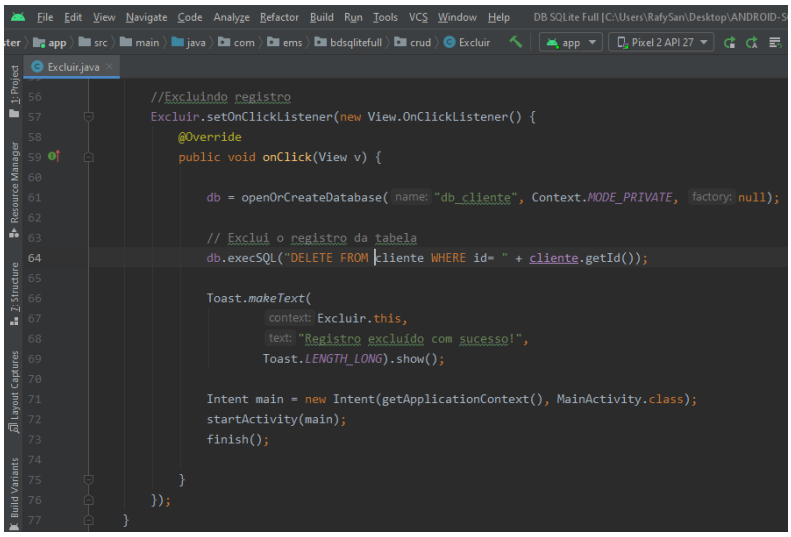
Faz atualização dos dados cadastros no app, caso tenha alguma alteração.

FIGURA 11: UPDATE (ATUALIZAÇÃO) TELA DE EDITRECORD.

```
        // Coleta os dados digitados nos campos
        ContentValues values = new ContentValues();
        values.put("cpf", cpf.getText().toString());
        values.put("telefone", telefone.getText().toString());
        values.put("nome", nome.getText().toString());
        values.put("email", email.getText().toString());
        values.put("descricao", descricao.getText().toString());

        Cliente novosDados = new Cliente();
        novosDados.setCpf(cpf.getText().toString());
        novosDados.setTelefone(telefone.getText().toString());
        novosDados.setNome(nome.getText().toString());
        novosDados.setEmail(email.getText().toString());
        novosDados.setDescricao(descricao.getText().toString());

        // Atualiza os dados na tabela
        db = openOrCreateDatabase( name: "db_cliente", Context.MODE_PRIVATE,  factory: null);
        db.execSQL("UPDATE cliente SET " +
                "cpf='" + novosDados.getCpf() + "'," +
                "telefone='" + novosDados.getTelefone() + "'," +
                "nome='" + novosDados.getNome() + "'," +
                "email='" + novosDados.getEmail() + "'," +
                "descricao='" + novosDados.getDescricao() + "' " +
                "WHERE id=" + cliente.getId()
        );

        // Cria uma caixa de mensagem e mostra os dados incluídos
        Message message = new Message( context: EditRecord.this);
        message.show(
                 titulo: "Dados Atualizados com Sucesso!",
                novosDados.getDados(),
                R.drawable.ic_add);
        ;
        Intent main = new Intent(getApplicationContext(), MainActivity.class);
        startActivity(main);
});
```

Exclui a informação, o registro seleciona no campo de details.

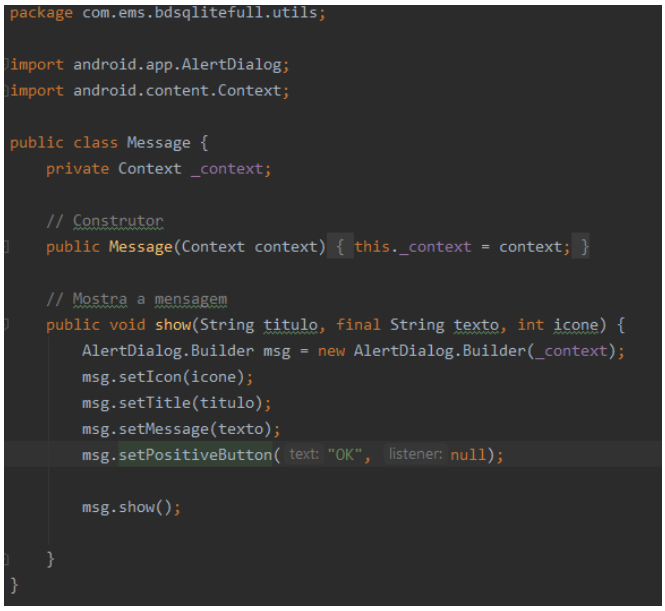
FIGURA 12: DELETE (DESTRUIÇÃO) TELA EXCLUIR.



```
//Excluindo registro
Excluir.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        db = openOrCreateDatabase( name: "db_cliente", Context.MODE_PRIVATE, factory: null);

        // Exclui o registro da tabela
        db.execSQL("DELETE FROM cliente WHERE id= " + cliente.getId());

        Toast.makeText(
                context: Excluir.this,
                text: "Registro excluído com sucesso!",
                Toast.LENGTH_LONG).show();

        Intent main = new Intent(getApplicationContext(), MainActivity.class);
        startActivity(main);
        finish();

    }
});
}
```

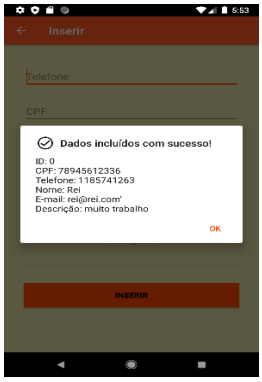
Mostra as mensagens de na tela de inserir quando inseri um novo cliente, na tela de editar quando salva as informações alteradas.

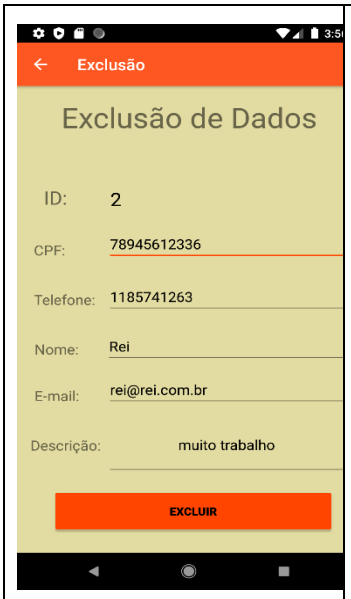
FIGURA 13: MESSAGE - CAIXA DE TEXTO.

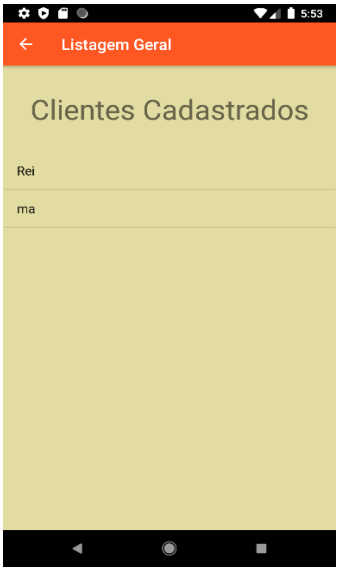


```
package com.ems.bdsqlitefull.utils;

import android.app.AlertDialog;
import android.content.Context;

public class Message {
    private Context _context;

    // Construtor
    public Message(Context context) { this._context = context; }

    // Mostra a mensagem
    public void show(String titulo, final String texto, int icone) {
        AlertDialog.Builder msg = new AlertDialog.Builder(_context);
        msg.setIcon(icone);
        msg.setTitle(titulo);
        msg.setMessage(texto);
        msg.setPositiveButton( text: "OK",  listener: null);

        msg.show();

    }
}
```

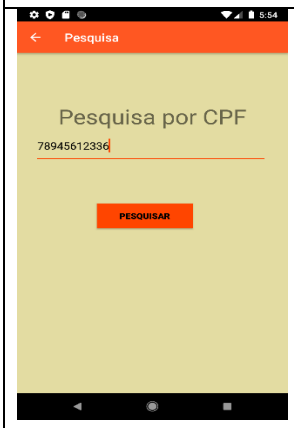
FIGURA 14: TELAS DO APP.

Tela Inicial	Tela Menu	Tela Inserir
		
Tela dados	**Tela dados inseridos**	**Tela detalhes**
		
Tela exclusão	**Tela edição**	**Tela Listagem**





Tela de pesquisa



7. FERRAMENTAS UTILIZADAS

Para a construção deste projeto foi utilizado o Android Studio (versão 4.0 for Windows 64-bit), uma IDE que fornece as ferramentas necessárias para a construção de aplicativos para todos tipos de aparelhos Android (ANDROID STUDIO 2020). Desenvolvido pela Google tendo como base a IDE IntelliJ IDEA da JetBrains. Dentre as diversas funcionalidades disponíveis no Android Studio serão ressaltadas apenas algumas mais relevantes:

Emulador: o emulador talvez seja a ferramenta mais interessante dessa IDE, pois ela faz com que o desenvolvedor não precise estar com um smartphone por perto para poder visualizar e testar seu aplicativo, ele também conta com diversos aparelhos de diversas configurações e tamanhos disponíveis para o teste do aplicativo em diferentes cenários;

Editor de layout: ao trabalhar com o layout em XML, o Android Studio fornece ferramenta que facilita a criação do layout utilizando um método "drag and drop", em que o usuário arrasta os elementos visuais em um posicionamento de sua preferência, e o editor já projeta esse layout para outros tamanhos de telas e gera o código para o mesmo;

Analisador APK: ferramenta que analisa o conteúdo e tamanho de componentes em seu aplicativo informando quais componentes podem ser alterados. Dentre outras funções ele ajuda a reduzir o tamanho de sua APK;

Editor de linguagens: ferramenta capaz de automatizar o processo de traduzir seu aplicativo para diversos idiomas, basta o usuário preencher um "dicionário"com as palavras e suas respectivas traduções e apontar para o identificador dessa palavra no código que o sistema do escolhe a versão correta.

8. CONCLUSÃO

Este projeto teve como objetivo a criação do app da empresa fictícia Criative S.A, onde foram baseados no site da empresa. Este aplicativo mostrou o quão grande é o mercado de ti, o quanto temos que nos preparar, pois as dificuldades para a criação foram muitas, mas não desistimos e criamos um app baseado no que aprendemos no decorrer do curso como banco de dados e trabalho em equipe.

O app está bem simples e fácil manuseio, pois não tivemos uma base concreta da material de desenvolvimento de aplicativos móveis, assim finalizamos o app com muita ajuda do professor Edson Melo de Souza (arquitetura de software), somos muito gratos por este professor, que nos orientou e mostrou como que tínhamos que fazer.

9. BIBLIOGRAFIA

https://developer.android.com/studio
https://drive.google.com/drive/folders/1GywXamYBf6GwHFx0b9ut7xxMbOCCJesY
https://github.com/Rafyy2102/Projeto_faculdade_App
https://developer.android.com/studio/intro
https://developer.android.com/studio/run
https://developer.android.com/studio/run/managing-avds?utm_source=android-studio
http://theclub.com.br/Restrito/Revistas/201207/andr1207.aspx
https://www.youtube.com/watch?v=nc25f459t0Y
https://material.io/
https://developer.ibm.com/br/tutorials/j-introtojava1/
Professor: Luís Carlos dos Santos Junior :
https://classroom.google.com/u/0/c/njuxoti2nze0mdfa
https://www.tutorialspoint.com/android/index.htm