FACULDADE DE TECNOLOGIA DE AMERICANA
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCIO GABBAI GONÇALVES
LEVI DE SOUZA DOS SANTOS

DESENVOLVIMENTO DE UM APLICATIVO ANDROID PARA O GERENCIAMENTO DE CONFEITARIAS

AMERICANA, SP
2018

FACULDADE DE TECNOLOGIA DE AMERICANA
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCIO GABBAI GONÇALVES
LEVI DE SOUZA DOS SANTOS

DESENVOLVIMENTO DE UM APLICATIVO ANDROID PARA O GERENCIAMENTO DE CONFEITARIAS

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Esp. Antônio Alfredo Lacerda.
Área de concentração: Programação Orientada a Objetos.

AMERICANA, SP
2018

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

G627d GONÇALVES, Márcio Gabbai

Desenvolvimento de um aplicativo Android para o gerenciamento de confeitarias. / Márcio Gabbai Gonçalves, Levi de Souza dos Santos. – Americana, 2018.

130f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Esp. Antônio Alfredo Lacerda

1. Dispositivos móveis – aplicativos 2. Android - aplicativos I. SANTOS, Levi de Souza dos II. LACERDA, Antônio Alfredo. III. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.519

MARCIO GABBAI GONÇALVES
LEVI DE SOUZA DOS SANTOS

DESENVOLVIMENTO DE UM APLICATIVO ANDROID PARA O GERENCIAMENTO DE CONFEITARIAS

Trabalho de graduação apresentado como exigência parcial para obtenção de título de Tecnólogo em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia – FATEC/Americana.
Área de concentração: Programação Orientada a Objetos.

Americana, 03 de dezembro de 2018.

Banca Examinadora:

Antônio Alfredo Lacerda
Especialista
Faculdade de Tecnologia – FATEC/Americana

Kleber de Oliveira Andrade
Doutor
Faculdade de Tecnologia – FATEC/Americana

Diógenes de Oliveira
Mestre
Faculdade de Tecnologia – FATEC/Americana

Dedicamos esse trabalho a nossa família, amigos, professores que nos auxiliaram nesse processo e a todos que diretamente ou indiretamente nos ajudaram.

AGRADECIMENTOS

Nossos agradecimentos a todos que contribuíram, direta ou indiretamente, para a realização deste trabalho, em especial:

Aos professores e coordenadores do Curso de Análise e Desenvolvimento de Sistemas, pela dedicação e competência na ajuda, além de transferir os conhecimentos adquiridos por toda a vida a nós. Também a instituição Centro Paula Souza que nos deu a oportunidade de ter uma formação em tecnologia e nos avaliando através desse trabalho.

Aos nossos familiares, por sempre nos apoiar e incentivar na elaboração do trabalho.

Aos nossos amigos, que nos ajudaram e aguentaram todos os dias.

A microempresária Anáile Murielle Cruz Gonçalves que se ofereceu para mostrar as necessidades do mercado onde nosso aplicativo irá atuar.

Por fim, gostaríamos de agradecer a Deus, que durante toda a nossa vida tem nos ajudado e acompanhado nos momentos difíceis.

“A tarefa não é tanto ver aquilo que ninguém viu, mas pensar o que ninguém ainda pensou sobre aquilo que todo mundo vê.”

(Arthur Schopenhauer)

RESUMO

Este trabalho trata-se do desenvolvimento de um aplicativo de gerenciamento para microempresários, que realizará o controle de gastos, retiro de margem de lucro por pedidos, agendará pedidos realizado por clientes e publicação em redes sociais para divulgação. Mesmo tendo dificuldades, a aplicação foi desenvolvida com sucesso, para isso contamos com o auxílio de uma microempresária no ramo do aplicativo demonstrando objetivos aguardados e as necessidades do mercado para o desenvolvimento do aplicativo.

PALAVRAS-CHAVE: DESENVOLVIMENTO DE SOFTWARE, DISPOSITIVOS MÓVEIS – APLICATIVOS.

ABSTRACT

This work is about development of a management application for microentrepreneurs, which performs expense control, withdraws profit margins by ordering, scheduling requests for publication and publishing on social networks for disclosure. Even though having difficulties, a mobile application was successfully obtained, for this we have the help of a micro businesswoman in the field of application demonstrating awaited objectives and the market needs for this application development.
KEYWORDS: SOFTWARE DEVELOPMENT, MOBILE - APPLICATIONS.

SUMÁRIO

LISTA DE FIGURAS E DE TABELAS

INTRODUÇÃO

Todos os momentos estamos cercados por pessoas que buscam um aumento de renda, a realização de seus sonhos, um complemento na renda e ou a realização de ter seu próprio negócio. Essas e outras coisas, motivam as pessoas a utilizarem seus talentos para alcançar esses objetivos pessoais e muitas vezes familiares.

Vemos essas pessoas cursando o ensino superior ou se especializando em uma área técnica, que muitas vezes não possuem recursos suficientes para se manter durante o período educacional em que se encontram. Para manter-se, economicamente, essas pessoas na maior parte das vezes iniciam-se a vender alimentos de fabricação própria aos seus colegas, amigos, familiares e desconhecidos. Vemos então surgirem vendedores de trufas, sonhos, ovos de pascoa, entre outros.

O sistema aqui proposto, tem como objetivo auxiliar esses indivíduos, com uma ferramenta móvel especializada e focada em sua área de negócio.

> Um sistema de informação pode ser definido tecnicamente como um conjunto de componentes inter-relacionados que coletam (ou recuperam), processam, armazenam e distribuem informações destinadas a apoiar a tomada de decisões, a coordenação e o controle de uma organização (LAURON, 2010).

Este trabalho abordará a engenharia e a implementação desse sistema proposto. Abordará também conceitos teóricos e técnicos relacionados ao desenvolvimento *mobile*, os desafios enfrentados para se desenvolver na plataforma *Android* e os recursos implementados no sistema como a utilização de APIs de redes sociais, calendário (manipulação própria, não nativo do sistema), Código de Endereçamento Postal e cálculos.

OBJETIVO

O objetivo para a realização desse trabalho acadêmico para a conclusão do curso de Análise e Desenvolvimento de Sistemas (ADS) da Faculdade de Tecnologia de Americana (FATEC-AM), foi tornar os integrantes do grupo mais preparados para o mercado de trabalho atual.

Tendo o desejo de crescimento e desenvolvimento nessa área de desenvolvimento *mobile*, surgiu à oportunidade de realizar um projeto para microempreendedores na área de venda de doces. Estudando um pouco sobre esse mercado encontramos os seguintes dados em uma pesquisa do SEBRAE em 2017 sobre a atividade (área de negócio) em que os microempreendedores se encontram inscritos:

Figura 1 – Quantidade de pessoas que possuem o MEI e suas áreas de negócio.

Fonte: SEBRAE – Dados eletrônicos.

Dentre a divisão de 40 áreas de negócio feita na pesquisa pelo SEBRAE, a venda de doces está entre as 16 áreas de negócios com o maior número de pessoas que micro empreendem.

Em conversa com a microempresária Anáile, descobrimos que a carência desse mercado de ter uma ferramenta focalizada em seu empreendimento é muito grande, e o maior número de empreendedores nessa área não possuem um MEI registrado, o que torna essa área de trabalho mais extensa do que a apresentada na pesquisa, ou seja, mais clientes e maior divulgação que resultam em maiores desafios de melhoria do projeto e maior visibilidade no mercado de trabalho em nossa área.

1 ENGENHARIA DE SOFTWARES

Neste capítulo serão abordados os conceitos sobre metodologia ágil, linguagem de programação e marcação, banco de dados e as *APIs*.

1.1 METODOLOGIA ÁGIL

Metodologia significa um “corpo de regras e diligências estabelecidas para realizar uma pesquisa” (*GOOGLE* DICIONÁRIO). Ágil significa “eficiente, rápido no trabalho; diligente, trabalhador.” (*GOOGLE* DICIONÁRIO).

Sendo assim,

> [...] uma metodologia ágil serve para auxiliar os times a pensar e trabalhar mais eficazmente tomando decisões melhores. Muitas metodologias ágeis são conhecidas e utilizadas como: *Scrum*, *Extreme Programming – XP*, *Lean* e *Kanban*.”(STELLMAN; GREENE, 2014).

A maioria do uso dessas metodologias se estende a todas as áreas de negócio, elas vêm sendo muito utilizadas em projetos de desenvolvimento de *software*, implementação de projetos de cerimonias de casamento, implementação de novos serviços internos e externos de uma empresa, entre outros.

1.1.1 Metodologia ágil Scrum

A metodologia ágil escolhida para o projeto de desenvolvimento do *software* proposto foi a *Scrum*. Essa metodologia surgiu em 1986 por um artigo publicado pela *Harvard Business Review*, descrevendo como empresas como a “*Honda*, *Canon* e *Fuji-Xerox* estavam crescendo de forma escalável em nível global” (RUBIN, 2012). Desde então foram publicados artigos e livros conforme o envolvimento dessa metodologia com o gerenciamento de projetos e desenvolvimento de *softwares*.

> *Scrum* é uma metodologia ágil para desenvolvimento de produtos e serviços inovadores. Com uma metodologia ágil, você inicia criando um *backlog de* produtos – uma lista de recursos priorizados e outras capacidades necessária para o desenvolvimento de um produto de sucesso. Guiado por um *backlog* de produtos, você sempre trabalha com os itens mais importantes ou de alta prioridade primeiro. (RUBIN, 2012. Tradução livre do Autor).

As atribuições de nosso grupo em relação a está metodologia foram as seguintes, Marcio Gabbai Gonçalves como PO (*Product Owner)* – pessoa responsável em ver os requerimentos do sistema com base no mercado onde o produto irá atuar, uma de suas tarefas foi de manter-se atualizado com as necessidades dos futuros clientes. Levi de Sousa dos Santos como SM (*Scrum Master*) – pessoa responsável em realizar o desbloqueios das ações e tomada de decisões em meio a impasse técnico, também procura manter o foco do time, estabelecendo metas, prazos e conduzindo as reuniões chamadas pela metodologia de cerimonias e ambos Marcio e Levi atuaram como DT (*Development Team*) – responsáveis pela implementação técnica dos requerimentos levantados pelo PO. Vale ressaltar que essa metodologia exige mais pessoas para compor um time e divisão de tarefas. Durante este trabalho procuramos sentir um pouco como funciona essa metodologia e aprender por meio de aplicação pratica um pouco do Ágil.

1.2 LINGUAGEM DE PROGRAMAÇÃO

É a notação utilizada para descrever a execução de algoritmos em computadores. “Onde os principais pilares dos algoritmos giram em torno de sintaxe e semântica. Sendo sintaxe: as regras formais da linguagem. E a semântica: a forma de se escrever na linguagem”. (SILVA, 1988)

1.2.1 Orientação ao Objeto

Muitas linguagens de programação hoje em dia são o que chamamos de orientada ao objeto. “Em programação o conceito de um objeto e similar ao objeto no mundo real” (CROSS, 2018). Por exemplo, uma caneta é um objeto que contém atributos como cor da tinta, quantidade de tinta, tamanho, entre outros. Possui também ações como escrever, desenhar ou rabiscar, dependendo do que é feito com ela. Semelhantemente um objeto em um *software* possui ações que chamamos de métodos, esses métodos fazem ações mediante o que passamos para eles – chamados parâmetros, como na caneta, se apertarmos ela contra um papel e direciona-la para muitas posições, será a ação de rabiscar, já se rabiscar fosse um método ele receberia os parâmetros, posição da mão e apertando contra um papel.

1.3 LINGUAGEM DE MARCAÇÃO

Linguagem de marcação, tem como objetivo criar convenções para a formatação de textos. Uma linguagem de marcação possui como características especificar “quais marcas são utilizadas, quais são os parâmetros de exigência, como se dá a distinção entre o que é marca e o que é conteúdo do texto e qual a função da marcação” (ALMEIDA, 2018). Estas características podem ser representadas “por diversos símbolos ou palavras-chave diferentes, dependendo da linguagem de marcação” (SOUZA, 2018).

1.4 BANCO DE DADOS

Toda aplicação que gera dados e necessita guarda-los para uso posterior possui o que é chamado de banco de dados. “O propósito de um banco de dados é armazenar e reter informações relacionadas entre si” (ORACLE).

1.5 *API*

API – Application Programming Interface ou Interface de Programação da Aplicação é o nome utilizado para “especificação de protocolos, procedimentos e serviços que podem ser utilizados [...] para implementar um requerimento” (SAMOYLOV, 2018). Atualmente quase todos os programas utilizão operações especificas, ou troca de informações com outros serviços, por exemplo o *Facebook*, são utilizadas *APIs* que recebem as requisições do sistema (ações a serem feitas como, por exemplo quando um sistema faz um *login* em uma aplicação ou *site* com o *login* do *Facebook*), essas aplicações que usam as *APIs* possuem uma chave especifica de segurança, onde é possível saber quem fez a requisição.

> Estamos conectados com o mundo e uns aos outros como nunca antes. Você pode comprar mercadorias, postar informações, carregar fotos ou marcar uma reserva em um restaurante através de um dispositivo móvel ou um computador. Toda essa conectividade é possível pelo uso do que chamamos de *APIs* [...] uma *API* é um conjunto de sub-rotinas, definições, protocolos e ferramentas que podem ser utilizadas para o desenvolvimento de aplicações. Ela é uma coleção de elementos construtivos que abstraem e sublinham a implementação de uma tecnologia, expondo objetos específicos ou ações que um desenvolvedor pode realizar, simplificando a programação ao fazê-lo. (VITANTONIO; MAHLER, 2018. Tradução livre do Autor).

Vale ressaltar que as *APIs* aqui apresentadas são apenas para manipulação de dados e que todos os dados são recuperados em *JSON*.

1.5.1 API Facebook

A API utilizada do Facebook possibilita a criação de uma função de compartilhamento, permitindo que os usuários "acrescentem mensagens personalizadas a sua publicação antes de compartilhá-las em sua linha do Tempo no Facebook" (FACEBOOK).

Por questão de pré-requisito e de segurança tanto o Facebook quanto Instagram pedem para que os desenvolvedores vinculem uma chave única de identificação ao seu aplicativo "e enviem um relatório sobre o aplicativo para eles, assim, quando validados que os requisitos foram atendidos, seja possível utilizar os recursos descritos" (FACEBOOK) em ambas *APIs*.

A classe abaixo exibe o código de manipulação de dados com a *API* do *Facebook*:

```
private class FacebookDAO extends AsyncTask<String, Void, String> {
  @Override
  protected void onPreExecute() {

  }

  @Override
  protected void onPostExecute(String jsonStr) {


    try {
      JSONObject jsonObject = new JSONObject(jsonStr);
      JSONArray jsonRoot = jsonObject.getJSONArray("data");

      for (int i = 0; i < jsonRoot.length(); i++) {


        RedeSocialModel redeSocialModel = new RedeSocialModel();
        redeSocialModel.setType(1);
        JSONObject jsonItem = jsonRoot.getJSONObject(i);

        redeSocialModel.setPicture(jsonItem.isNull("picture") ? "" : jsonItem.getString("picture"));
        redeSocialModel.setFullPicture(jsonItem.isNull("full_picture") ? "" :
```

```
jsonItem.getString("full_picture"));
            redeSocialModel.setId(jsonItem.isNull("id") ? "" : jsonItem.getString("id"));
            redeSocialModel.setCreatedTime(jsonItem.isNull("created_time") ? "" :
jsonItem.getString("created_time"));
            redeSocialModel.setDescription(jsonItem.isNull("description") ? "" :
jsonItem.getString("description"));
            JSONObject jsonObject1 = jsonItem.getJSONObject("likes");
            JSONObject jsonObject2 = jsonObject1.getJSONObject("summary");

            redeSocialModel.setLikes(jsonObject2.isNull("total_count") ? -1 : jsonObject2.getInt("total_count"));

            jsonObject1 = jsonItem.getJSONObject("comments");
            jsonObject2 = jsonObject1.getJSONObject("summary");

            redeSocialModel.setComments(jsonObject2.isNull("total_count") ? -1 :
jsonObject2.getInt("total_count"));


            list.add(redeSocialModel);


        }
        asyncTaskSharedPreferences.incrementaCounter();


        if(asyncTaskSharedPreferences.terminou()){

            Collections.sort(list, new Comparator<RedeSocialModel>() {
                @Override
                public int compare(RedeSocialModel redeSocialModel, RedeSocialModel redeSocialModel2) {
                    DateFormat f = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
                    try {
                        return
f.parse(redeSocialModel2.getCreatedTime()).compareTo(f.parse(redeSocialModel.getCreatedTime()));
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }

                    return 0;

                }
            });

            redeSocialListaPublicacaoView.getListaPublicacoesListView().setAdapter(new
RedeSocialAdapter(context, list));
```

```
            redeSocialListaPublicacaoView.getProgressoProgressBar().setVisibility(View.INVISIBLE);
            redeSocialListaPublicacaoView.getListaPublicacoesListView().setVisibility(View.VISIBLE);
        }

    } catch (JSONException e) {
        e.printStackTrace();
    }

}

@Override
protected String doInBackground(String... dados) {

    return requestDados(URL_FACEBOOK);
}
}
```

1.5.2 API Instagram

Diz-se do *Instagram*: “se o seu aplicativo criar/publicar fotos e seus usuários quiserem compartilhar usando o Instagram, você pode usar o Android Intents para enviar sua mídia no fluxo de compartilhamento do Instagram”. (INSTAGRAM. Tradução livre do Autor). Semelhante ao *Facebook* tendo a chave anexa e fixa no sistema podemos fazer a integração com essa rede social.

Segue o código do da manipulação de dados dessa API:

```
private class InstagramDAO extends AsyncTask<String, Void, String> {
    @Override
    protected void onPreExecute() {

    }

    @Override
    protected void onPostExecute(String jsonStr) {

        try {
            JSONObject jsonObject = new JSONObject(jsonStr);
            JSONArray jsonRoot = jsonObject.getJSONArray("data");

            for (int i = 0; i < jsonRoot.length(); i++) {
```

```
            RedeSocialModel redeSocialModel = new RedeSocialModel();
            redeSocialModel.setType(2);
            JSONObject jsonItem = jsonRoot.getJSONObject(i);

            redeSocialModel.setId(jsonItem.isNull("id") ? "" : jsonItem.getString("id"));
            redeSocialModel.setCreatedTime(jsonItem.isNull("created_time") ? "" :
jsonItem.getString("created_time"));
            redeSocialModel.setDescription(jsonItem.isNull("caption") ? "" : jsonItem.getString("caption"));


            JSONObject jsonObject1 = jsonItem.getJSONObject("images");
            JSONObject jsonObject2 = jsonObject1.getJSONObject("thumbnail");

            redeSocialModel.setPicture(jsonObject2.isNull("url") ? "" : jsonObject2.getString("url"));


            jsonObject1 = jsonItem.getJSONObject("images");
            jsonObject2 = jsonObject1.getJSONObject("standard_resolution");

            redeSocialModel.setFullPicture(jsonObject2.isNull("url") ? "" : jsonObject2.getString("url"));

            jsonObject1 = jsonItem.getJSONObject("likes");
            redeSocialModel.setLikes(jsonObject1.isNull("count") ? -1 : jsonObject1.getInt("count"));

            jsonObject1 = jsonItem.getJSONObject("comments");
            redeSocialModel.setComments(jsonObject1.isNull("count") ? -1 : jsonObject1.getInt("count"));


            list.add(redeSocialModel);


        }


        asyncTaskSharedPreferences.incrementaCounter();

        if(asyncTaskSharedPreferences.terminou()){

            Collections.sort(list, new Comparator<RedeSocialModel>() {
                @Override
                public int compare(RedeSocialModel redeSocialModel, RedeSocialModel redeSocialModel2) {
                    DateFormat f = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
                    try {
                        return
f.parse(redeSocialModel2.getCreatedTime()).compareTo(f.parse(redeSocialModel.getCreatedTime()));
```

```
                } catch (ParseException e) {
                    e.printStackTrace();
                }

                return 0;

            }
        });

        redeSocialListaPublicacaoView.getListaPublicacoesListView().setAdapter(new
RedeSocialAdapter(context, list));
        redeSocialListaPublicacaoView.getProgressoProgressBar().setVisibility(View.INVISIBLE);
        redeSocialListaPublicacaoView.getListaPublicacoesListView().setVisibility(View.VISIBLE);
      }

    } catch (JSONException e) {
      e.printStackTrace();
    }

  }

  @Override
  protected String doInBackground(String... dados) {
    return requestDados(URL_INSTAGRAM);
  }
}
```

1.5.3 API Twitter

A API utilizada do Twitter permite que publiquemos “conteúdos de forma pública no Twitter, de forma a facilitar ao cliente o compartilhamento de informações e artigos” (TWITTER). Segue abaixo o código de manipulação de dados dessa rede social:

```
private class TwitterDAO{

  public void execute(String task){

    ResponseList<Status> listTwitter = null;
    try {
      listTwitter = twitterApp.getmTwitter().getUserTimeline(twitterSessao.getUsername());
    } catch (TwitterException e) {
      e.printStackTrace();
    }
```

```
        for (Status each : listTwitter) {


            RedeSocialModel redeSocialModel = new RedeSocialModel();
            redeSocialModel.setType(3);

            if(each.getMediaEntities().length > 0){


                redeSocialModel.setPicture(each.getMediaEntities()[0].getMediaURLHttps());
                redeSocialModel.setFullPicture(each.getMediaEntities()[0].getMediaURLHttps());
            }
            else{

redeSocialModel.setPicture("https://pbs.twimg.com/profile_images/875135141135302656/eiM2Wz66_400x40
0.jpg");

redeSocialModel.setFullPicture("https://pbs.twimg.com/profile_images/875135141135302656/eiM2Wz66_400
x400.jpg");
            }

            redeSocialModel.setLikes(each.getFavoriteCount());
            redeSocialModel.setComments(each.getRetweetCount());
            redeSocialModel.setDescription(each.getText());



            System.out.println(each.getCreatedAt().toString());
            redeSocialModel.setCreatedTime(each.getCreatedAt().toString());
            list.add(redeSocialModel);

            System.out.println("Sent by: @" + each.getUser().getScreenName()
                    + " - " + each.getUser().getName() + "\n" + each.getText()
                    + "\n");
        }

        asyncTaskSharedPreferences.incrementaCounter();


        if(asyncTaskSharedPreferences.terminou()) {

            Collections.sort(list, new Comparator<RedeSocialModel>() {
                @Override
                public int compare(RedeSocialModel redeSocialModel, RedeSocialModel redeSocialModel2) {
                    DateFormat f = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
                    try {
```

```
                return
f.parse(redeSocialModel2.getCreatedTime()).compareTo(f.parse(redeSocialModel.getCreatedTime()));
            } catch (ParseException e) {
                e.printStackTrace();
            }

            return 0;

        }
    });

    redeSocialListaPublicacaoView.getListaPublicacoesListView().setAdapter(new
RedeSocialAdapter(context, list));
    redeSocialListaPublicacaoView.getProgressoProgressBar().setVisibility(View.INVISIBLE);
    redeSocialListaPublicacaoView.getListaPublicacoesListView().setVisibility(View.VISIBLE);
  }

 }
}
```

1.5.4 API de CEP

No Sistema proposto, para facilidade do usuário cadastrar o endereço de entrega do pedido de seus clientes, foi realizada uma conexão com uma API permite recuperar os dados do Código do Endereçamento Postal, segue o código:

```
private String requestCep(String cep) {
  Uri builtUri = Uri.parse("https://viacep.com.br/ws/" + cep.toString().replaceAll("[^\\d.]", "") +
"/json/").buildUpon()
      .build();

  try {
    URL url = new URL(builtUri.toString());
    HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
    urlConnection.setRequestMethod("GET");
    urlConnection.connect();

    InputStream inputStream = urlConnection.getInputStream();
    StringBuffer buffer = new StringBuffer();
    if (inputStream == null) {
      return null;
    }
    BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

    String line;
```

```
        while ((line = reader.readLine()) != null) {

            buffer.append(line + "\n");
        }

        if (buffer.length() == 0) {

            return null;
        }
        String jsonStr = buffer.toString();


        return jsonStr;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

1.5.5 API de Calendário

Devido a grande diferença nos aparelhos *mobiles* decidiu-se implementar um calendário resultante de informações de um API especifica, tornando-o mais dinâmico e compatível, segue o código de manipulação de dados do calendário:

```
public class AgendaCalendarView extends FrameLayout implements
StickyListHeadersListView.OnStickyHeaderChangedListener {

    private static final String LOG_TAG = AgendaCalendarView.class.getSimpleName();

    private CalendarView mCalendarView;
    private AgendaView mAgendaView;
    private FloatingActionButton mFloatingActionButton;

    private int mAgendaCurrentDayTextColor, mCalendarHeaderColor, mCalendarBackgroundColor,
mCalendarDayTextColor, mCalendarPastDayTextColor, mCalendarCurrentDayColor, mFabColor;
    private CalendarPickerController mCalendarPickerController;

    private ListViewScrollTracker mAgendaListViewScrollTracker;
    private AbsListView.OnScrollListener mAgendaScrollListener = new AbsListView.OnScrollListener() {
        int mCurrentAngle;
        int mMaxAngle = 85;

        @Override
        public void onScrollStateChanged(AbsListView view, int scrollState) {
```

```
        }

        @Override
        public void onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int totalItemCount) {
            int scrollY = mAgendaListViewScrollTracker.calculateScrollY(firstVisibleItem, visibleItemCount);
            if (scrollY != 0) {
                mFloatingActionButton.show();
            }
            Log.d(LOG_TAG, String.format("Agenda listView scrollY: %d", scrollY));
            int toAngle = scrollY / 100;
            if (toAngle > mMaxAngle) {
                toAngle = mMaxAngle;
            } else if (toAngle < -mMaxAngle) {
                toAngle = -mMaxAngle;
            }
            RotateAnimation rotate = new RotateAnimation(mCurrentAngle, toAngle,
mFloatingActionButton.getWidth() / 2, mFloatingActionButton.getHeight() / 2);
            rotate.setFillAfter(true);
            mCurrentAngle = toAngle;
            mFloatingActionButton.startAnimation(rotate);
        }
    };

    // region Constructors

    public AgendaCalendarView(Context context) {
        super(context);
    }

    public AgendaCalendarView(Context context, AttributeSet attrs) {
        super(context, attrs);

        TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.ColorOptionsView, 0, 0);
        mAgendaCurrentDayTextColor =
a.getColor(R.styleable.ColorOptionsView_agendaCurrentDayTextColor,
getResources().getColor(R.color.theme_primary));
        mCalendarHeaderColor = a.getColor(R.styleable.ColorOptionsView_calendarHeaderColor,
getResources().getColor(R.color.theme_primary_dark));
        mCalendarBackgroundColor = a.getColor(R.styleable.ColorOptionsView_calendarColor,
getResources().getColor(R.color.theme_primary));
        mCalendarDayTextColor = a.getColor(R.styleable.ColorOptionsView_calendarDayTextColor,
getResources().getColor(R.color.theme_text_icons));
        mCalendarCurrentDayColor =
a.getColor(R.styleable.ColorOptionsView_calendarCurrentDayTextColor,
```

```
getResources().getColor(R.color.calendar_text_current_day));
        mCalendarPastDayTextColor = a.getColor(R.styleable.ColorOptionsView_calendarPastDayTextColor,
getResources().getColor(R.color.theme_light_primary));
        mFabColor = a.getColor(R.styleable.ColorOptionsView_fabColor,
getResources().getColor(R.color.theme_accent));

        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        inflater.inflate(R.layout.view_agendacalendar, this, true);

        setAlpha(0f);
    }

    // endregion

    // region Class - View

    @Override
    protected void onFinishInflate() {
        super.onFinishInflate();
        mCalendarView = (CalendarView) findViewById(R.id.calendar_view);
        mAgendaView = (AgendaView) findViewById(R.id.agenda_view);
        mFloatingActionButton = (FloatingActionButton) findViewById(R.id.floating_action_button);
        ColorStateList csl = new ColorStateList(new int[][]{new int[0]}, new int[]{mFabColor});
        mFloatingActionButton.setBackgroundTintList(csl);

        mCalendarView.findViewById(R.id.cal_day_names).setBackgroundColor(mCalendarHeaderColor);
        mCalendarView.findViewById(R.id.list_week).setBackgroundColor(mCalendarBackgroundColor);

        mAgendaView.getAgendaListView().setOnItemClickListener((AdapterView<?> parent, View view, int
position, long id)->{
            mCalendarPickerController.onEventSelected(CalendarManager.getInstance().getEvents().get(position));
        });

        BusProvider.getInstance().toObserverable()
                .subscribe(event -> {
                    if (event instanceof Events.DayClickedEvent) {
                        mCalendarPickerController.onDaySelected(((Events.DayClickedEvent) event).getDay());
                    } else if (event instanceof Events.EventsFetched) {
                        ObjectAnimator alphaAnimation = new ObjectAnimator().ofFloat(this, "alpha", getAlpha(),
1f).setDuration(500);
                        alphaAnimation.addListener(new Animator.AnimatorListener() {
                            @Override
                            public void onAnimationStart(Animator animation) {
```

```
                    }

                    @Override
                    public void onAnimationEnd(Animator animation) {
                        long fabAnimationDelay= 500;
                        // Just after setting the alpha from this view to 1, we hide the fab.
                        // It will reappear as soon as the user is scrolling the Agenda view.
                        new Handler().postDelayed(() -> {
                            mFloatingActionButton.hide();
                            mAgendaListViewScrollTracker = new
ListViewScrollTracker(mAgendaView.getAgendaListView());
                            mAgendaView.getAgendaListView().setOnScrollListener(mAgendaScrollListener);
                            mFloatingActionButton.setOnClickListener((v) -> {
                                mAgendaView.translateList(0);

mAgendaView.getAgendaListView().scrollToCurrentDate(CalendarManager.getInstance().getToday());
                                new Handler().postDelayed(() -> mFloatingActionButton.hide(), fabAnimationDelay);
                            });
                        }, fabAnimationDelay);
                    }

                    @Override
                    public void onAnimationCancel(Animator animation) {

                    }

                    @Override
                    public void onAnimationRepeat(Animator animation) {

                    }
                });
                alphaAnimation.start();
            }
        });
    }

    // endregion

    // region Interface - StickyListHeadersListView.OnStickyHeaderChangedListener

    @Override
    public void onStickyHeaderChanged(StickyListHeadersListView stickyListHeadersListView, View header, int
position, long headerId) {
        Log.d(LOG_TAG, String.format("onStickyHeaderChanged, position = %d, headerId = %d", position,
headerId));
```

```
        if (CalendarManager.getInstance().getEvents().size() > 0) {
            CalendarEvent event = CalendarManager.getInstance().getEvents().get(position);
            if (event != null) {
                mCalendarView.scrollToDate(event);
                mCalendarPickerController.onScrollToDate(event.getInstanceDay());
            }
        }
    }

    // endregion

    // region Public methods

    public void init(List<CalendarEvent> eventList, Calendar minDate, Calendar maxDate, Locale locale,
CalendarPickerController calendarPickerController) {
        mCalendarPickerController = calendarPickerController;

        CalendarManager.getInstance(getContext()).buildCal(minDate, maxDate, locale);

        // Feed our views with weeks list and events
        mCalendarView.init(CalendarManager.getInstance(getContext()), mCalendarDayTextColor,
mCalendarCurrentDayColor, mCalendarPastDayTextColor);

        // Load agenda events and scroll to current day
        AgendaAdapter agendaAdapter = new AgendaAdapter(mAgendaCurrentDayTextColor);
        mAgendaView.getAgendaListView().setAdapter(agendaAdapter);
        mAgendaView.getAgendaListView().setOnStickyHeaderChangedListener(this);
        CalendarManager.getInstance().loadEvents(eventList);

        // add default event renderer
        addEventRenderer(new DefaultEventRenderer());
    }

    public void addEventRenderer(@NonNull final EventRenderer<?> renderer) {
        AgendaAdapter adapter = (AgendaAdapter) mAgendaView.getAgendaListView().getAdapter();
        adapter.addEventRenderer(renderer);
    }

    // endregion
}
```

2 FERRAMENTAS DE DESENVOLVIMENTO E LINGUAGENS

Neste capítulo será descrito brevemente sobre: *UML,* as ferramentas de desenvolvimento, Sistema Gerenciado de Banco de Dados, as linguagens de programação e marcação utilizados durante o trabalho.

2.1 FERRAMENTAS DE DESENVOLVIMENTO

2.1.1 Android Studio

O *Android Studio* é uma *IDE – Integrated Development Environment*, ou seja, um ambiente de desenvolvimento integrado, “oficial para o desenvolvimento de aplicativos *Android* e distribuído pela empresa *Google”* (*Google* Developers, 2018).

2.2 FERRAMENTA DE MODELAGEM

2.2.1 Astah Community

Astah é a ferramenta de modelagem para *UML* utilizada para criar todos os diagramas do projeto, dentre eles o Diagrama de Classe, Caso de Uso (ASTAH, 2018a). Sendo o *Astah Community* uma versão grátis.

2.3 SISTEMA GERENCIADOR DE BANCO DE DADOS (SGBD)

2.3.1 SQLite

No desenvolvimento *mobile*, existem duas formas de armazenamento de dados. Primeira, os dados são armazenados na nuvem (enviar os dados via *internet* a serviços de armazenamento *online*, os mais conhecidos são *Amazon* S3, *Google Cloud, Azure Cloud* e *IBM Cloud*). Segundo os dados são armazenados localmente, em um banco de dados no próprio dispositivo.

> Em termos simples, o *SQLite* é um pacote de *software* em um domínio público que fornece um sistema de gerenciamento de banco de dados relacional ou RDBMS (*Relational Database Management System*). O sistema de banco de dados relacional é utilizado para armazenar registros definidos do usuário em grandes tabelas. Além do mais, para armazenamento e gerenciamento dados, um mecanismo de banco de dados pode processar consultas complexas que

> combinam os dados de múltiplas tabelas para gerar relatórios e resumos. (KREIBICH, 2010. Tradução livre do Autor).

2.4 LINGUAGEM DE MARCAÇÃO

2.4.1 XML

O desenvolvimento *mobile – Android* – utiliza a linguagem de marcação *XML*, que significa *Extensible Markup Language* em tradução livre "Linguagem de Marcação Extensível". Fora criada em 1996 pela W3C, com o propósito de ser mais simples que a *SGML*, outra linguagem de marcação da época (ALMEIDA, p. 7). É uma linguagem usada para representar dados de um texto que são "marcados" a fim de descrever as propriedades dos dados. "O uso de marcação permite que o texto seja interpretado juntamente com o seu conteúdo e formato" (GRAVES, 2003, p. 2)

2.5 *UML*

UML é a abreviação de *Unified Modeling Language*, que em tradução livre significa "Linguagem de Modelagem Unificada". Que diferentemente das linguagens de programações e de marcação, a *UML* tem por definição a seguinte descrição: "[...] uma linguagem visual para especificar, construir e documentar os artefatos dos sistemas." (KRUCTHEN, 2005, p. 39), sistemas esses de preferência orientados a objeto.

2.6 LINGUAGENS DE PROGRAMAÇÃO

2.6.1 Java

Java é uma linguagem de programação orientada a objeto, sendo utilizada em diversas áreas, inclusive para desenvolvimento *mobile*, pode ser um ponto de partida para os iniciantes no mundo da programação ou ser uma grande ferramenta nas mãos de programadores já consagrados no mercado de trabalho, "o *Java* é uma poderosa linguagem de programação que pode ser utilizada tanto para implementar aplicativos baseados na *Internet* ou para o desenvolvimento de *softwares* que dependam ou não de rede" (DEITEL, 2010).

2.7 ARQUITETURA E ORGANIZAÇÃO DA APLICAÇÃO

2.7.1 MVC – Model-View-Controller

"*MVC* é um padrão de arquitetura usado na engenharia de *software*, cujo princípio fundamental é baseado na ideia de que a lógica de uma aplicação deva estar separada de sua apresentação" (KILIÇDAĞI; YILMAZ, 2014.Tradução livre do autor).

O padrão *MVC* estrutura o projeto em 3 camadas. 1ª camada: Manipulação de dados (*model*); 2ª camada: Interação com o usuário (*view*); 3ª camada: Controle entre as camadas (*controller*).

3 Desenvolvimento de um aplicativo *Android* para o gerenciamento de confeitarias

3.1 Requisitos

"Um requisito é um atributo essencial ou uma característica para um sistema ou um elemento de um sistema" (GRADY, 2013).

Por convenção, a referência a requisitos é feita através do nome da subseção onde eles estão descritos, seguidos do identificador do requisito, de acordo com a especificação a seguir:

[nome da subseção. identificador do requisito]

Por exemplo, o requisito funcional [Recuperação de dados. RF016] deve estar descrito em uma subseção chamada "Recuperação de dados", em um bloco identificado pelo número [RF016]. Já o requisito não-funcional [Confiabilidade. NF008] deve estar descrito na seção de requisitos não-funcionais de Confiabilidade, em um bloco identificado por [NF008].

Os requisitos devem ser identificados com um identificador único. A numeração inicia com o identificador [RF001] ou [NF001] e prossegue sendo incrementada à medida que forem surgindo novos requisitos.

Para estabelecer a prioridade dos requisitos, foram adotadas as denominações “essencial”, “importante” e “desejável”.

- **Essencial** é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis, que têm que ser implementados impreterivelmente.
- **Importante** é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.
- **Desejável** é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

As principais funcionalidades que o sistema deverá cumprir são:

- Login e cadastro de novos usuários.
- Gerenciamento de Produtos.
- Gerenciamento de Clientes.
- Gerenciamento de pedidos.
- Gerenciamento de usuário.
- Integração com as redes sociais (*Facebook*, *Instagram* e *Twitter*).

Entretanto, haverá limitações, dentre as quais:

- A integração será somente com as redes sociais citadas acima.
- O sistema será apenas *mobile*, não sendo possível acessa-lo através de *sites* e aplicações *desktop*.
- O sistema em sua versão inicial estará apenas acessível para *Android* a partir da versão da *API* 15 (*ice cream sandwich*).
- Em sua versão inicial o sistema apenas terá suporte a língua portuguesa.

O principal e único usuário do aplicativo será o microempreendedor vendedor de doces, que possui uma série de diversos problemas, dentre os quais:

- Dificuldade de se colocar preços justos e lucrativos em seus produtos.
- Visualizar a divulgação dos produtos em várias redes sociais rapidamente.
- Gerenciamento de clientes, tanto com nomes, endereços, valores a cobrar e agendamento de entregas.
- Cálculo de valores de negócio.

3.1.1 Requisitos Funcionais

Requisitos funcionais são declarações das funções – ações – que o sistema deve fazer para satisfazer as necessidades de negócio proposta no projeto de desenvolvimento.

> Os requisitos funcionais especificam o que o produto deve fazer. Eles descrevem as ações que o produto deve executar para satisfazer as razões fundamentais de sua existência. Por exemplo, um requisito funcional descreve a ação que o produto deve executar no trabalho pelo qual ele se destina. A intenção é entender os requisitos funcionais e assim transmitir aos desenvolvedores o que é requerido que o produto faça [...] (ROBERTSON, J.; ROBERTSON, S., 2006. Tradução livre do Autor)

***Login* e cadastro**

[*Login* e Cadastro. RF001] – Conexão do usuário com seu *login* e senha.

Prioridade: Essencial

Entradas e pré-condições: Para realizar o *login* local o usuário já deverá ter criado uma conta, ou caso contrário será informado que a conta informada não existe.

Saídas e pós condições: Continuar na tela de *login* caso o *login* e senha informados estejam incorretos, ou, entrar no sistema caso o *login* e senha informados estejam corretos.

[*Login* e Cadastro. RF002] – Processo de cadastro de conta invocado logo após o clique no botão “Cadastre-se agora!” na inicialização do sistema.

Prioridade: Essencial

Entradas e pré-condições: As entradas serão os dados de *login* e senha do usuário, incluindo a confirmação de senha.

Saídas e pós condições: Se todos os dados estiverem corretos, o sistema principal será carregado. Caso contrário, um alerta deverá ser feito durante o processo de cadastrar a nova conta.

Clientes

[Clientes. RF001] – Deverá listar todos os clientes cadastrados e fornecer acesso as telas de adição, alteração e remoção de clientes.

Prioridade: Importante.

Entradas e pré-condições: Para a listagem dos clientes, basta que a funcionalidade de clientes tenha sido chamada que será aberta uma lista de todos os clientes existentes. Para adicionar o cliente, é necessário adicionar os dados como nome, e-*mail* e telefone. Para alterar o cliente, é necessário informar os mesmos dados, porém os dados já estarão preenchidos, sendo necessário apenas realizas as alterações desejadas. Para excluir um cliente, é necessário que ele exista na lista de clientes. Ao excluir é pedido uma confirmação de exclusão verificando se o usuário realmente deseja realizar esta ação.

A pré-condição é que o usuário esteja conectado no sistema.

Saídas e pós condições: Ao ser adicionado, alterado ou excluído um cliente, ocorre o redirecionamento para a tela de listagem de clientes. Caso algum dado informado esteja incorreto na adição ou alteração do cliente, um alerta será feito na respectiva tela.

Usuário

[Usuário. RF001] – Ao usuário estar conectado no sistema, devera exibir a tela de alteração de usuário.

Prioridade: Essencial.

Entradas e pré-condições: Caso o usuário deseje alterar o perfil, um formulário com os dados já preenchidos será chamado, e caberá ao usuário realizar

as alterações desejadas e então clicar no botão de salvar. Caso o usuário clique em configurações, o usuário poderá customizar algumas configurações locais no aplicativo.

A pré-condição é que o usuário esteja conectado no sistema.

Saídas e pós condições: Ao usuário clicar em salvar, ele será direcionado novamente a tela inicial com as informações atualizadas. Caso o usuário clique em sair/desconectar, ele será redirecionado para o *login* do sistema.

Pedidos

[Pedidos. RF001] – Deverá exibir um calendário que lista todos os pedidos cadastrados como eventos e fornecer acesso as telas de adição, alteração e remoção de pedidos.

Prioridade: Essencial.

Entradas e pré-condições: Para a listagem dos pedidos, basta que a funcionalidade de pedidos tenha sido chamada que será aberta uma lista em formato de calendário com todos os pedidos existentes. Para adicionar o pedido, é necessário adicionar os dados do pedido e selecionar o cliente previamente cadastrado. Para alterar o pedido, é necessário informar os mesmos dados, porém os dados já estarão preenchidos, sendo necessário apenas realizas as alterações desejadas. Para excluir um pedido, é necessário que ele exista na lista de pedidos. Ao excluir é pedido uma confirmação de exclusão verificando se o usuário realmente deseja realizar esta ação.

A pré-condição deste caso de uso é que o usuário esteja conectado no sistema e já possua o cliente cadastrado.

Saídas e pós condições: Ao ser adicionado, alterado ou excluído um pedido, ocorre o redirecionamento para a tela de listagem de pedidos. Caso algum dado seja informado incorretamente na tela de alteração ou adição, um alerta será informado na respectiva tela. Caso o usuário não possua nenhum cliente registrado, será necessário acessar a funcionalidade de clientes para adiciona-lo.

Redes Sociais

[Redes Sociais. RF001] – Processo de integração

Prioridade: Essencial.

Entradas e pré-condições: A pré-condição obrigatória é a conexão com a internet para o funcionamento da rede social. O usuário deverá informar seu *login* e senha na respectiva rede social para integra-la ao sistema.

Saídas e pós condições: Ao usuário integrar a rede social no sistema, ele poderá visualizar suas publicações em suas respectivas redes.

[Redes Sociais. RF002] – Interação com as principais redes sociais para o microempreendedor, *Facebook*, *Twitter*, *Instagram.*

Prioridade: Essencial.

Entradas e pré-condições: A pré-condição obrigatória é que o usuário possua acesso à *internet* e já possua a respectiva rede social integrada. O usuário poderá realizar publicações informando textos e/ou imagens.

Saídas e pós condições: Ao usuário realizar uma publicação ele será enviado de volta a tela de listagem de suas publicações realizadas.

3.1.2 Requisitos não funcionais

Requisito não funcional é aquele que descreve não o que o sistema fará, mas como ele fará para satisfazer as necessidades de negócio proposta no projeto de desenvolvimento.

Requisitos nãos funcionais

> [...] devem ser compreendidos para facilitar o *design* e o desenvolvimento do modelo operacional de destino. Isto inclui servidores, redes e plataformas, incluindo o tempo de execução do aplicativo nos ambientes. Estes são pontos críticos para execução de testes de *benchmark*. Eles também afetam o *design* técnicos dos componentes da aplicação (PARADKAR, 2017. Tradução livre do Autor)

Usabilidade

[Usabilidade. NF001] – Todas as telas deverão funcionar independentemente do tamanho do dispositivo móvel utilizado com a orientação vertical padrão da tela.

Prioridade: Importante.

[Usabilidade. NF002] – Telas deverão fornecer um guia rápido de ajuda ao usuário

Prioridade: Desejável.

Confiabilidade

[Confiabilidade. NF001] – O processo de cálculo deverá apresentar o padrão do mercado no quesito de cálculo, obtendo em sua grande maioria dos casos uma boa precisão.

Prioridade: Essencial.

Desempenho

[Desempenho. NF001] – Os usuários não devem sofrer impactos maiores do que de 2 segundos em qualquer interação com a interface. Dentre as interações, algumas delas:

- Adicionar item no banco de dados.
- Atualizar recursos gráficos.
- Alterar ou excluir itens no banco de dados.

Prioridade: Desejável.

Segurança

[Segurança. NF001] – Toda senha deverá ser salva em seu formato *SHA*-256.

Prioridade: Essencial.

[Segurança. NF002] - Apenas usuários cadastrados poderão acessar o sistema.

Prioridade: Essencial.

Distribuição

[Distribuição. NF001] – O sistema deverá ser distribuído na *Google Play Store*.

Prioridade: Essencial

Padrões

[Padrões. NF001] – Todas as variáveis e nomes deverão ser definidos conforme padrão estabelecido pela equipe de desenvolvimento, buscando utilizar como boa pratica o *Clean Code* e *CamelCase*.

Prioridade: Desejável

[Padrões. NF002] – As variáveis relativas a widgets (botões, calendários, campos de edição de texto, dentre outros) deverão iniciar pela sua respectiva nomenclatura seguida pelo seu tipo.

Exemplo:

editarUsuarioButton – Botão de edição do usuário.

Prioridade: Desejável

[Padrões. NF003] – Os pacotes deverão ser organizados de forma a facilitar a manutenção e criação de código, de preferência seguindo a estrutura organizacional das sessões de casos de uso.

Prioridade: Desejável

Hardware* e *software

[*Hardware* e *software*. NF001] – Deverá ser utilizado banco de dados local

Prioridade: Essencial.

[*Hardware* e *software*. NF002] – O dispositivo deverá conter 50Mb de espaço livre para instalação.

Prioridade: Desejável.

[Hardware e software. NF003] – O dispositivo deverá conter 100Mb de memória *RAM* livres para a execução do aplicativo.

Prioridade: Desejável.

3.2 Diagramas de Caso de Uso

Os diagramas de caso de uso descrevem um cenário de funcionalidades do ponto de vista do usuário, catalogando os requisitos funcionais do sistema. Dentro do diagrama são retratados os atores (representado pelos bonecos), as funcionalidades (representadas pelos balões com a ação escrita por dentro) e as relações (representadas pelas linhas).

Spence e Bittner (2002) descrevem os diagramas de caso de uso como uma

> [...] descrição da sequência dos eventos que realizados juntos, levam a um sistema que faz algo útil. [...] Casos de uso reduzem a ambiguidade dos requisitos do sistema por especificar quando e sobre quais condições certos comportamentos ocorrem. (Tradução livre do Autor)

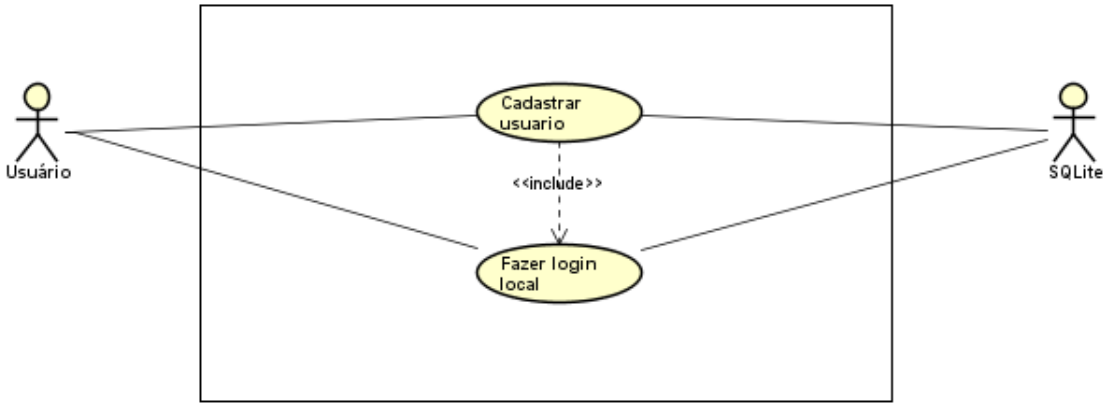
Os atores que interagem com o sistema são:

- **Usuário** é o ator que representa os utilizadores deste aplicativo. Um ator pode, por exemplo, se cadastrar, fazer *login*, agendar entregas, adicionar clientes, dentre outros.
- ***API Facebook*** representa o ator da *API* que permite a interação entre o aplicativo e o *Facebook*, por exemplo, o *login* com as credenciais da rede social.
- ***API Instagram*** representa o ator da *API* que permite a interação entre o aplicativo e o *Instagram*, por exemplo, o *login* com as credenciais da rede social.
- ***API Twitter*** representa o ator da *API* que permite a interação entre o aplicativo e o *Twitter*, por exemplo, o *login* com as credenciais da rede social.

SQLite representa o banco de dados em tempo real, onde o sistema armazena as informações de usuários, clientes, dentre outros.

Caso de uso 1:

Figura 2 – Diagrama de caso de uso do login de usuário.



Fonte: Elaborado pelos autores

Caso de uso 2:

Figura 3 – Caso de uso de funcionalidades do aplicativo.

Fonte: Elaborado pelos autores

Cada funcionalidade dos diagramas de casos de uso acima será descrita nas tabelas 2 a 8.

O caso de uso “Fazer *Login* Local” utiliza o banco de dados *SQLite* para autenticar o *login* do usuário e sua senha. A base confirma o *login*. A Tabela 2 detalha o passo a passo da execução deste caso de uso.

Tabela 1 – Caso de uso “Fazer *login* local”.

Nome do caso de uso	Fazer *login* local
Atores envolvidos	Usuário e *SQLite*
Objetivo	Este caso de uso descreve os passos do *login*
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. O usuário informa os dados	
2. Usuário clica em *login*	
	3. O sistema leva os dados para o *SQLite*, que se encarrega do processo de verificação de conta e retorna com as informações.
	4. Ao retornar com as informações do usuário, o *SQLite* autêntica o usuário em sua base de dados.
	5. Após a autenticação, o sistema redireciona para a página inicial do aplicativo.
Validações	Para o *login* seja efetuado, o usuário deve entrar com seu usuário e senha do *SQLite*.

Fonte: Elaborado pelos autores

O caso de uso “Cadastrar Usuário” utiliza o banco de dados *SQLite* para registrar o *login* do usuário e sua senha. É solicitado criação do usuário, o *SQLite* a cria em sua base, registrando o *e-mail* e senha, a data em que foi criada a conta, a última vez o usuário que foi conectado e uma *User ID* (*UID*) gerada pelo *SQLite*. O Tabela 3 detalha o passo a passo da execução deste caso de uso.

Tabela 2 – Caso de uso “Cadastrar Usuário”.

Nome do caso de uso	Cadastrar usuário
Atores envolvidos	Usuário e *SQLite*
Objetivo	Este caso de uso descreve os passos para cadastrar usuário
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. O usuário clica em cadastrar	
2. Usuário informa seus dados	
	3. O sistema leva os dados para o *SQLite* registrar as informações do usuário, caso o *e-mail* já seja cadastrado não será permitido o novo registro.
	4. Após o registro usuário é enviado para o caso de uso “Fazer *login* local”
Validações	Para que o cadastro seja realizado o *e-mail* necessita ser valido e único. A senha será informada e confirmada

Fonte: Elaborado pelos autores

No caso de uso “Manter clientes” aparecerá uma lista com os clientes que o usuário cadastrou, a possibilidade de adicionar, alterar, buscar e excluir clientes. Ao usuário clicar sobre um item da lista as seguintes opções serão fornecidas:

- Mostrar dados do cliente.
- Alterar Cliente.
- Excluir Cliente.

No topo da tela, na barra de ação do aplicativo, aparecerá um campo de busca que irá filtrar a lista conforme o que o usuário está digitando. Também haverá um ícone de adicionar cliente no topo da tela. A tabela 4 descreve o caso de uso “Manter Cliente”.

Tabela 3 – Caso de uso “Manter clientes”.

Nome do caso de uso	Manter Cliente
Atores envolvidos	Usuário e *SQLite*
Objetivo	Fazer o gerenciamento dos clientes
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. O usuário clica na opção de clientes.	
	2. O sistema irá buscar os dados cadastrados dos clientes no SQLite e apresenta-los na tela em forma de lista.
3. O usuário clica na opção de adicionar clientes	
	4. O sistema apresentará uma tela de cadastro.
5. O usuário preenche o formulário com os dados do cliente (nome, e-mail, telefone) e clica em “Cadastrar”.	
	6. O sistema leva os dados para o SQLite que se encarregará de salvar os dados.
	7. O sistema deverá recarregar a listagem de clientes com as informações

	atualizadas.
8. O usuário clica sobre um item específico do menu e clica em “Mostrar cliente”	
	9. O sistema busca no banco de dados através do SQLite as informações do respectivo cliente através de seu identificador único.
	10. O sistema mostra na tela as informações que foram buscadas.
11. O usuário escreve alguma das informações do cliente no campo de busca.	
	12. O sistema pega o evento de escrita no campo de busca e o compara com a lista carregada na memória do celular, ocorrendo a partir desta lista a filtragem.
	13. O sistema apresenta a lista filtrada ao usuário.
14. O usuário clica sobre um item específico do menu e clica em “Alterar cliente”	
	15. Um formulário de alteração é apresentado ao usuário.
16. O usuário preenche os campos do formulário e clica em “Alterar”.	

	17. O SQLite recebe os dados do formulário e altera o respectivo registro do cliente.
	18. 18. A lista de clientes é atualizada.
19. O usuário clica sobre um item específico do menu e clica em “Excluir cliente”	
	20. O sistema fornece uma mensagem de se o usuário realmente deseja excluir o cliente.
21. O usuário informa se deseja ou não excluir o cliente.	
	22. Caso o usuário deseje excluir o cliente, o sistema irá excluir o registro do banco de dados a utilizando o SQLite e em seguida carregar a lista de clientes atualizada. Caso contrário, nenhuma ação será feita.
Validações	Para adicionar e alterar os clientes, será verificado se todos os dados informados são válidos e não vazios. Na exclusão ocorrerá uma validação de se o usuário realmente deseja excluir o cliente. As buscas serão realizadas em uma lista na memória local do celular do usuário, não ocorrendo chamadas excessivas ao banco. A cada cliente inserido deverá ser utilizado um identificador único novo.

Fonte: Elaborado pelos autores

No caso de uso “Manter usuário” o usuário poderá fazer alterações em seu perfil e configurar opções de segurança. A tabela 5 descreve o caso de uso “Manter Usuário”.

Tabela 4 – Caso de uso “Manter usuário”.

Nome do caso de uso	Manter usuário
Atores envolvidos	Usuário e *SQLite*
Objetivo	Este caso de uso faz as configurações do perfil do usuário
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. O usuário clica em sua foto de perfil.	
	2. O sistema abrirá a edição de perfil.
	3. O sistema mostrará uma tela com os dados anteriormente definidos prontos para a edição, no início da página aparecerá um botão para salvar e outro para o menu.
5. O usuário clica em salvar ou menu	
	6 . Caso ele clique em menu o sistema exibe o menu lateral para navegação no sistema, ao ser selecionado outra tela sem salvar as alterações elas serão perdidas. Caso ele clique em salvar o sistema salva no banco de dados e retorna como salvo e volta a página inicial.
7. O usuário clica em configuração	

	8. O sistema entre na tela de configuração.
9. O usuário clica para sair do sistema.	
	10. O sistema desconecta da conta e retorna a página de *login*
Validações	Para que as configurações da conta sejam salvas os dados necessitam ser salvos no *SQLite*.

Fonte: Elaborado pelos autores

No caso de uso “Manter Pedidos” aparecerá um calendário que pode adicionar, alterar, buscar e excluir pedidos importantes. Ao usuário clicar sobre uma data do calendário as seguintes opções serão fornecidas:

- Alterar pedido da Agenda.
- Excluir pedido Agenda.

A tabela 6 descreve o caso de uso “Manter Pedidos”.

Tabela 5 – Caso de uso “Manter Pedidos”.

Nome do caso de uso	Manter **Pedidos**
Atores envolvidos	Usuário e *SQLite*
Objetivo	Fazer o gerenciamento dos pedidos
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. O usuário clica na opção de pedidos.	

	2. O sistema irá buscar os dados salvos dos pedidos no *SQLite* e apresenta na tela em forma de calendário.
3. O usuário clica na data que deseja adicionar um pedido.	
	4. O sistema apresentará uma tela para cadastrar pedidos, cliente e horário do pedido.
5. O usuário deve preencher os campos pedidos, cliente e horário e clica em salvar.	
	6. O sistema leva os dados para o *SQLite* que se encarregará de salvar os dados.
	7. O sistema deverá recarregar o calendário com as informações atualizadas.
8. O usuário clica sobre uma data onde há um ou mais pedidos salvos.	
	9. O sistema busca no banco de dados através do *SQLite* as informações da respectiva data.
	10. O sistema mostra na tela as informações que foram buscadas.
11. O usuário seleciona o pedido que deseja alterar.	
	12. O sistema mostra na tela as informações que foram buscadas.
13. O usuário altera os campos pedidos, cliente e horário e clica em salvar.	

	14. O sistema leva os dados para o *SQLite* que se encarregará de salvar os dados.
	15. O sistema deverá recarregar o calendário com as informações atualizadas.
16. O usuário clica sobre uma data onde há um ou mais pedidos salvos.	
	17. O sistema busca no banco de dados através do *SQLite* as informações da respectiva data.
	18. O sistema mostra na tela as informações que foram buscadas.
19. O usuário seleciona o pedido que deseja excluir.	
	20. O sistema mostra na tela um *pop-up* perguntando se o usuário deseja excluir realmente aquele pedido.
21. O usuário clica em confirmar.	
	22. O sistema deverá recarregar o calendário com as informações atualizadas.
Validações	Para adicionar e alterar os pedidos, será verificado se todos os dados informados são válidos e não vazios. Na exclusão ocorrerá uma validação de se o usuário realmente deseja excluir o pedido. As buscas serão realizadas em uma lista na memória local do celular do usuário, não ocorrendo chamadas excessivas ao banco. A cada pedido inserido deverá ser utilizado um identificador único novo.

Fonte: Elaborado pelos autores

No caso de uso “Manter Redes Sociais” o usuário terá a possibilidade de se conectar com redes sociais, tais como, *Facebook*, *Twitter* e *Instagram*. Com essa conexão será possível que o microempreendedor consiga facilmente divulgar e publicar fotos de seu negócio sem a necessidade de ter que usar vários aplicativos para isso, para que essa conexão funcione, nesse caso de uso o usuário poderá:

- Conectar-se a uma rede social por meio de uma *API* especifica de cada fornecedor.
- Manter o sigilo de seu usuário e senha nas suas redes sociais.
- Usar para publicar fotos, eventos, comentários relacionados a seu negócio.

A tabela 7 descreve o caso de uso “Manter Redes Sociais”.

Tabela 6 – Caso de uso “Manter Redes Sociais”.

Nome do caso de uso	Manter Redes Sociais
Atores envolvidos	Usuário, *SQLite* e *API* de Redes sociais
Objetivo	Conexão e integração com as principais redes sociais.
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. O usuário abre a tela de configurações para escolher qual das redes sociais ele deseja vincular os aplicativos (podendo escolher mais de uma)	
	2. O sistema fornece ao usuário três opções por meio de ícones das redes

	sociais para que o usuário escolha.
3. O usuário escolhe a rede social	
	4. O sistema verifica se há conexão com a internet
	5. Sem conexão retorna ao usuário que deve se conectar à internet para conseguir se vincular a uma rede e utilizar os recursos que esse vínculo oferece
	6. Com conexão, o sistema faz uma chamada na *API* da rede social escolhida
	7. Conecta com os serviços da rede social
	8. *API* Solicita usuário e senha.
9. O usuário digita o usuário e senha	
	10. *API* fornece uma 'chave', que são salvos em banco de dados (*SQLite*) para que o sistema mantenha a conexão com a rede social sem que seja necessário solicitar novamente usuário e senha
	11. Fecha a *API*.
	12. Retorna na tela de configurações.

Fonte: Elaborado pelos autores

No caso de uso “Manter *Facebook*, Manter *Twitter*, Manter *Instagram*” é descrito a maneira como as *APIs* irão se ‘comunicar’ com o aplicativo.

A tabela 8 descreve o caso de uso “Manter *Facebook*, Manter *Twitter*, Manter *Instagram*”.

Tabela 7 – Caso de uso “Manter *Facebook*, Manter *Twitter*, Manter *Instagram*”.

Nome do caso de uso	Manter *Facebook*, Manter *Twitter*, Manter *Instagram*
Atores envolvidos	Usuário, *SQLite* e *API* de Redes sociais
Objetivo	Descrição das funcionalidades da integração do aplicativo com as redes sociais
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Usuário seleciona um *toggle button*	
	2. A *API* abre a tela de conexão de login e senha para que o usuário faça a conexão e gere a ‘chave’ de acesso a rede social.
3. Ao abrir para inserir login e senha o usuário insere os devidos valores.	
	4. O sistema entrega a chave de acesso a *API* juntamente com a solicitação da

	ação que será feita.
	5. A *API* faz a conexão com a rede social e interage com o aplicativo.
	6. *API* realiza busca por publicações do usuário naquela respectiva rede social.
	7. *API* retorna ao sistema se o procedimento foi bem-sucedido
	8. O sistema retorna a tela de configurações

Fonte: Elaborado pelos autores

3.3 Diagrama de Entidade e Relacionamento (DER)

Diagrama Entidade Relacionamento (DER) é um modelo diagramático que descreve o modelo de dados de um sistema com alto nível de abstração. Sua maior aplicação é para visualizar o relacionamento entre tabelas de um banco de dados, no qual as relações são construídas através da associação de um ou mais atributos destas tabelas.

Carpenter (2013) descreve o Diagrama de Entidade de Relacionamento como:

> […] Uma representação lógica de seu banco de dados apresentado como um modelo. Um DER é um modelo lógico e pode também ser um modelo físico. Um modelo logico não é especificamente um sistema; também não é exclusivo de um sistema gerenciador de banco de dados como o SQL Server ou Oracle.

Abaixo segue o DER do sistema proposto:

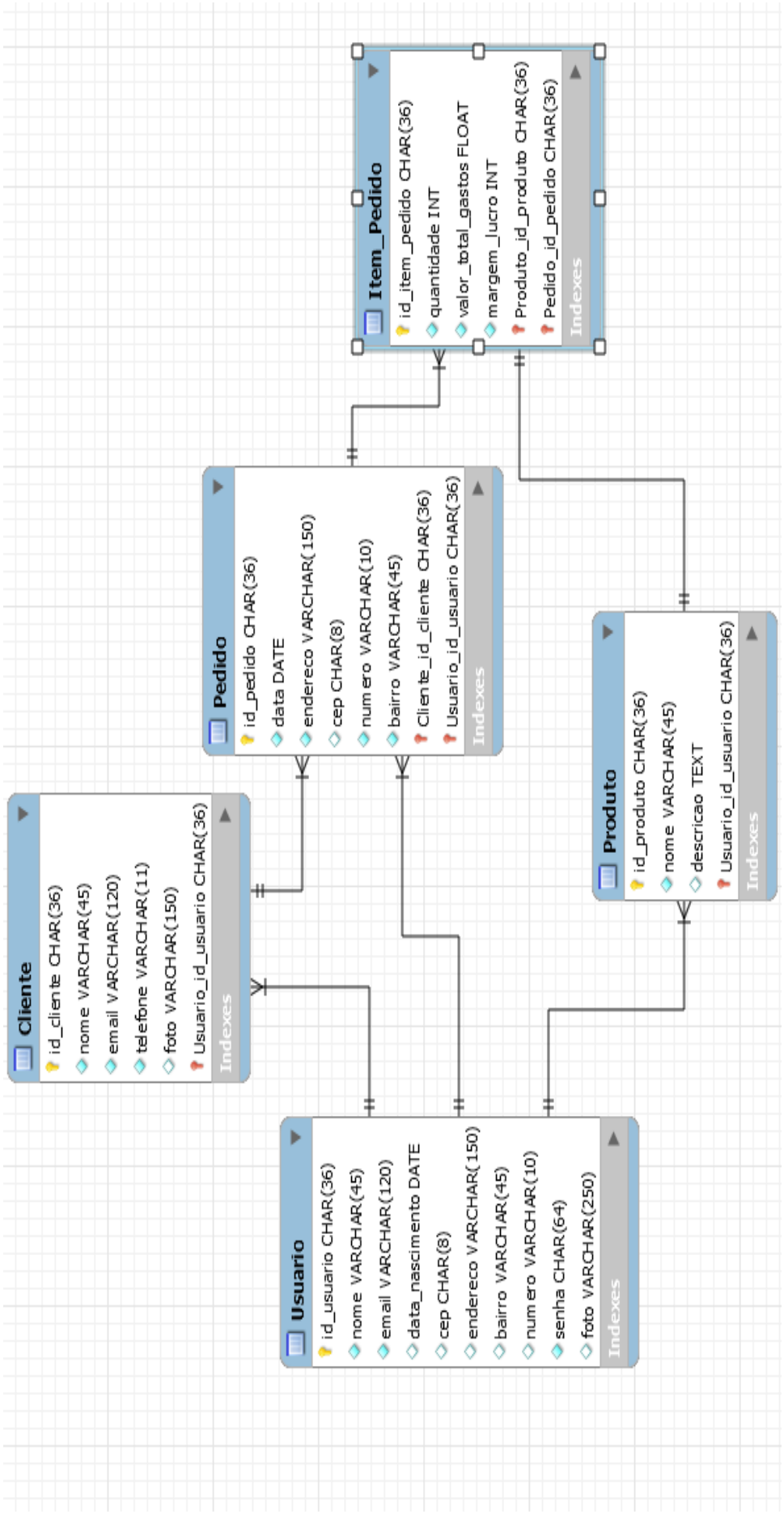


Figura 4 – Diagrama de Entidade e Relacionamento.

3.4 Dicionário de Dados

O Dicionário de Dados (DD) consiste em uma lista organizada de todos os elementos de dados que são pertinentes para o sistema.

“Um Dicionário de dados prove informações sobre cada atributo e se refere aos campos de um modelo dados” (BRANDENBURG, 2018).

As tabelas devem conter os seguintes campos:

Entidade: é o nome da entidade que foi definida no Modelo de Entidade e Relacionamento. A entidade é uma pessoa, objeto ou lugar que será considerada como objeto pelo qual temos interesse em guardar informações a seu respeito.

Atributo: Os atributos são as características da entidade Cliente que desejamos guardar.

Classe: as classes podem ser: simples, composto, multivalorado e determinante. Simples indica um atributo normal. Composto indica que ele poderá ser dividido em outros atributos, como por exemplo, o endereço. Multivalorado é quando o valor do atributo poderá não ser único e determinante é um atributo que será usado como chave, como CPF, Código do cliente, etc.

Domínio: podem ser numéricos, texto, data e booleano (verdadeiro ou falso). Podemos chamar também de tipo do valor que o atributo irá receber. A definição desses tipos deve seguir um processo lógico, exemplo: nome é texto, salário é numérico, data de nascimento é data e assim por diante.

Tamanho: define a quantidade de caracteres que serão necessários para armazenar o seu conteúdo. Geralmente o tamanho é definido apenas para atributos de domínio texto.

Descrição: é opcional e pode ser usado para descrever o que é aquele atributo ou dar informações adicionais que possam ser usadas futuramente pelo analista ou programador do sistema.

As tabelas 9 a 13 apresentam o DD de cada entidade.

Tabela 8 – Dicionário de Dados da entidade Usuário.

Entidade: Usuario				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Id_usuario	Determinante	Texto	36	*UUID*.
Nome	Simples	Texto	45	
Email	Simples	Texto	120	
Data_nascimento	Simples	Data	Não se aplica	
Cep	Simples	Texto	8	
Endereco	Simples	Texto	150	
Bairro	Simples	Texto	45	
Numero	Simples	Texto	10	
Senha	Simples	Texto	64	Senha criptografada
Foto	Simples	Texto	250	

Fonte: Elaborado pelos autores

Tabela 9 – Dicionário de Dados da entidade Cliente.

Entidade: Cliente				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Id_cliente	Determinante	Texto	36	*UUID*.

Nome	Simples	Texto	45	
Email	Simples	Texto	120	
Telefone	Simples	Texto	11	
Foto	Simples	Texto	150	
Usuario_id_usuario	Determinante	Texto	36	*UUID*.

Fonte: Elaborado pelos autores

Tabela 10 – Dicionário de Dados da entidade Produto.

Entidade: Produto				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Id_produto	Determinante	Texto	36	*UUID*.
Nome	Simples	Texto	45	
Descrição	Simples	Texto	Não se aplica	
Usuario_id_usuario	Determinante	Texto	36	*UUID*.

Fonte: Elaborado pelos autores

Tabela 11 – Dicionário de Dados da entidade Pedido.

Entidade: Pedido				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Id_pedido	Determinante	Texto	36	*UUID*.

Data	Simples	Data	Não se aplica	
Endereço	Simples	Texto	150	
Cep	Simples	Texto	8	
Número	Simples	Texto	10	
Bairro	Simples	Texto	45	
Cliente_id_cliente	Simples	Texto	36	*UUID*.
Usuario_id_usuario	Determinante	Texto	36	*UUID*.

Fonte: Elaborado pelos autores

Tabela 12 – Dicionário de Dados da entidade Item_Pedido.

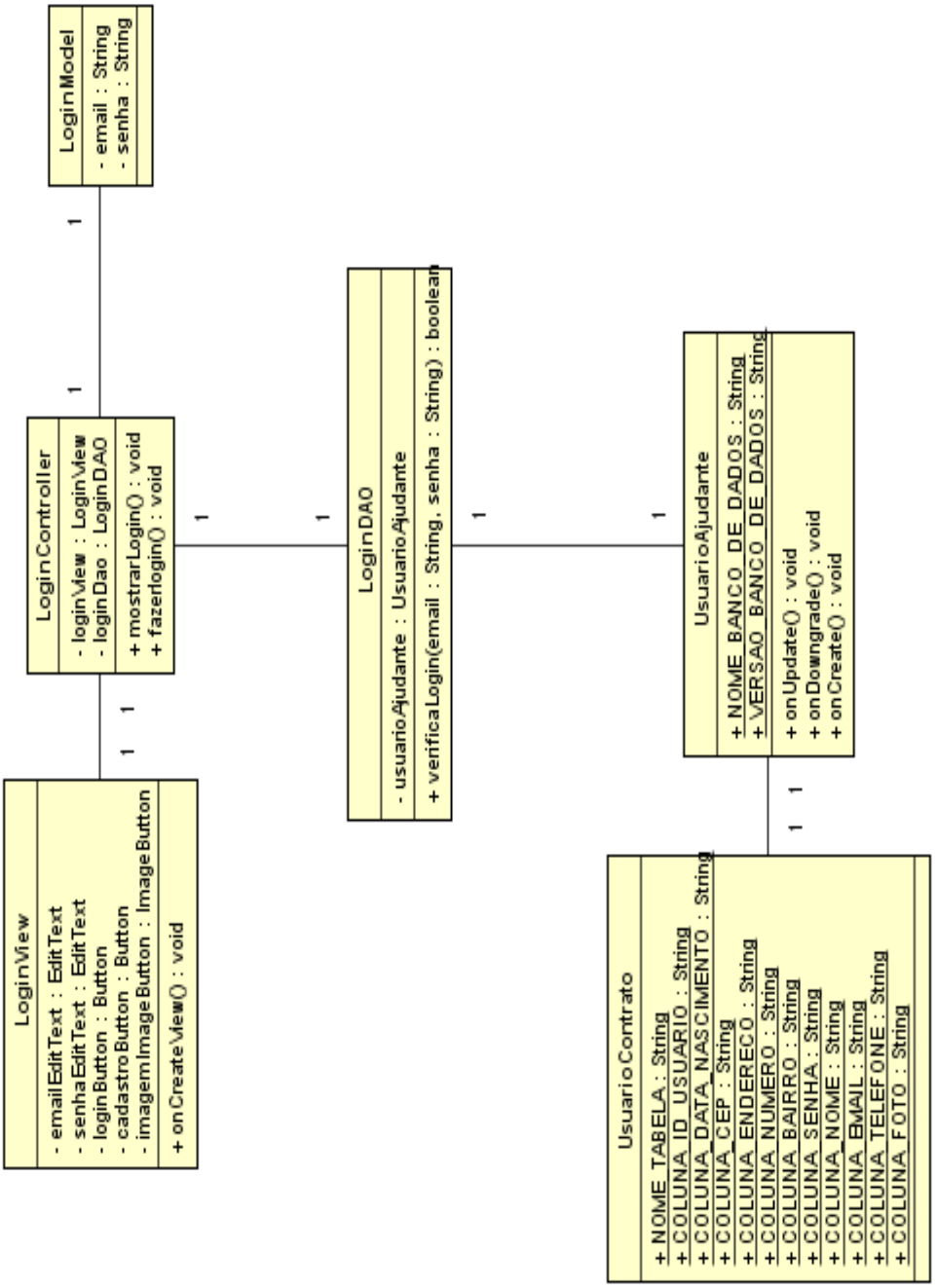
Entidade: Item_Pedido				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Id_item_pedido	Determinante	Texto	36	*UUID*.
Quantidade	Simples	Numérico	Não se aplica	
Valor_total_gastos	Simples	Numérico	Não se aplica	
Margem_lucro	Simples	Numérico	Não se aplica	
Produto_id_produto	Determinante	Texto	36	*UUID*.
Produto_id_pedido	Determinante	Texto	36	*UUID*.

Fonte: Elaborado pelos autores

3.5 Diagrama De Classes

Diagrama de Classes é a representação da estrutura e relação das classes que servem de modelo para os objetos. As Figuras 5 a 13 apresentam os diagramas de Classe.

Figura 5 – Protótipo de *login*.



Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

LoginView: Tem por objetivo ser a interface da tela de *login*. Os métodos identificados para esta classe foram:

Tabela 13 – Descrição dos Métodos da Classe: *LoginView*

Método	**Descrição**
onCreateView	Responsável pela criação da interface gráfica da tela de login.

Fonte: Elaborado pelos autores

LoginController: É a controladora da funcionalidade de *login*, sendo responsável por apresentar a interfaces e atualizar o modelo de dados, juntamente a escutar pelos eventos realizados na interface. Os métodos identificados para esta classe foram:

Tabela 14 – Descrição dos Métodos da Classe: *LoginControler*

Método	**Descrição**
mostrarLogin	Invocará a tela de interface, presente na classe de *LoginView*.
fazerLogin	Realiza o processo de login do sistema, consultando a partir do *e-mail* e senha digitados pelo usuário se ele existe no sistema. Utiliza o *LoginDao* para a consulta ao banco de dados.

Fonte: Elaborado pelos autores

LoginModel: Classe que servirá como modelo para os dados serem atualizados.

LoginDAO: Tem por acessar o banco de dados relativo a parte de consulta se um usuário existe ou não. Os métodos identificados para esta classe foram:

Tabela 15 – Descrição dos Métodos da Classe: *LoginDAO*

Método	**Descrição**
verificaLogin	Responsável pela verificação se o *e-mail* e senha passados como parâmetro existem no banco de dados. Retorna um booleano, sendo que *true* – verdadeiro – significa que o *login* está correto, e *false* – falso – se a conta não existe ou que os dados estão incorretos.

Fonte: Elaborado pelos autores

UsuarioContrato: Classe que representa a tabela do banco de dados do usuário.

UsuarioAjudante: Tem por objetivo fornecer o acesso direto ao banco de dados. Os métodos identificados para esta classe foram:

Tabela 16 – Descrição dos Métodos da Classe: UsuarioAjudante

Método	**Descrição**
onUpdate	Responsável pela atualização da tabela de usuários no banco de dados.
onDowngrade	Responsável pela diminuição da versão da tabela de

	usuários no banco de dados.
onCreate	Responsável pela criação da tabela de usuários no banco de dados.

Fonte: Elaborado pelos autores

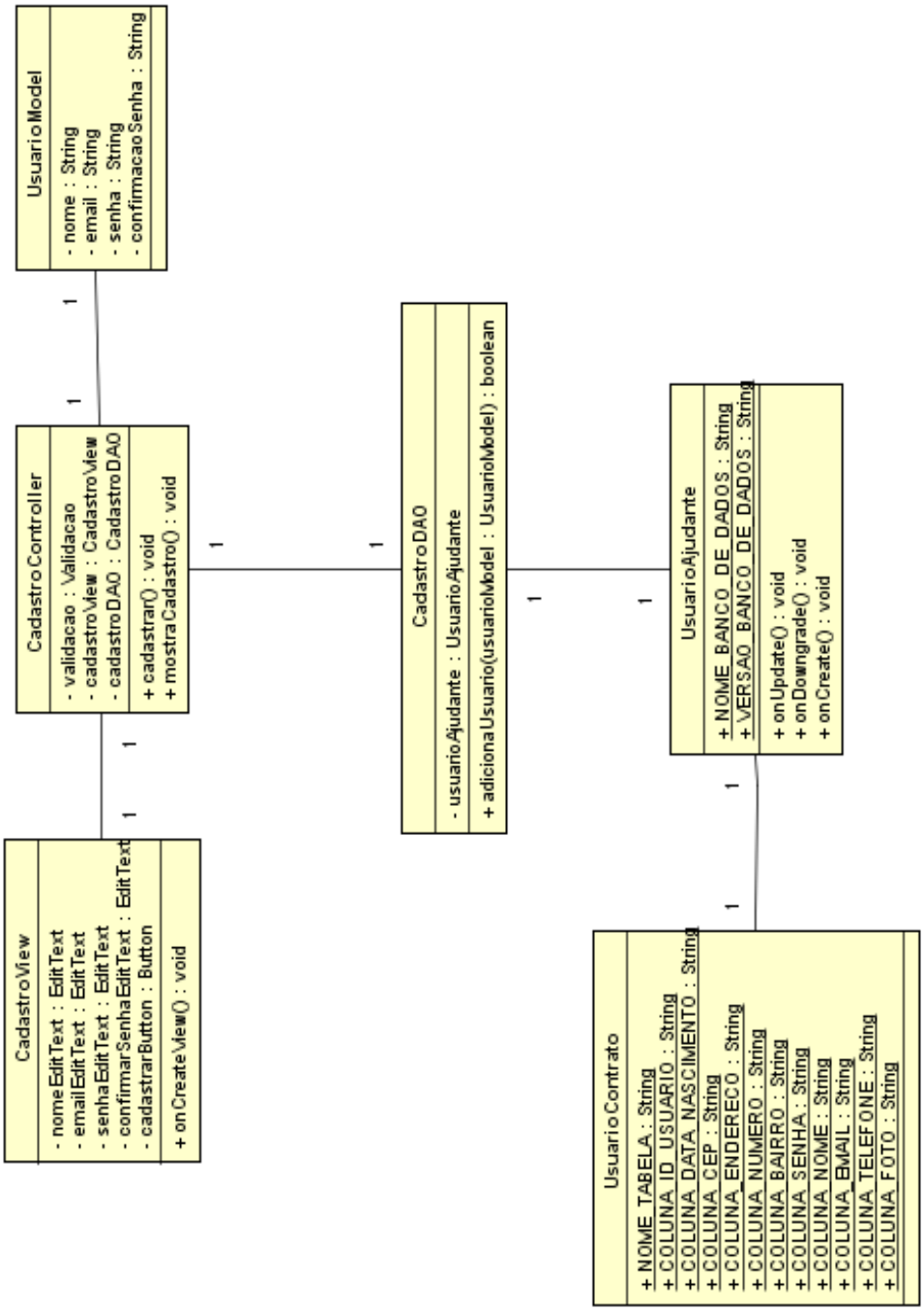


Figura 6 – Protótipo de cadastro.

Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

CadastroView. Tem por objetivo ser a interface da tela de *login*. Os métodos identificados para esta classe foram:

Tabela 17 – Descrição dos Métodos da Classe: *CadastroView*

Método	**Descrição**
onCreateView	Responsável pela criação da interface gráfica da tela de cadastro.

Fonte: Elaborado pelos autores

CadastroController: É a controladora da funcionalidade de cadastro, sendo responsável por apresentar a interfaces e atualizar o modelo de dados, juntamente a escutar pelos eventos realizados na interface. Os métodos identificados para esta classe foram:

Tabela 18 – Descrição dos Métodos da Classe: *CadastroController*

Método	**Descrição**
cadastrar	Realiza o processo de cadastro do usuário invocando a classe de *cadastroDao* para a realização do registro no banco de dados.
mostraCadastro	Responsável por mostrar a tela de cadastro do sistema.

Fonte: Elaborado pelos autores

UsuarioModel: Classe que servirá como modelo para os dados serem atualizados.

CadastroDAO: Tem como objetivo acessar o banco de dados relativo a parte de adição de um usuário no sistema. Os métodos identificados para esta classe foram:

Tabela 19 – Descrição dos Métodos da Classe: *CadastroDAO*

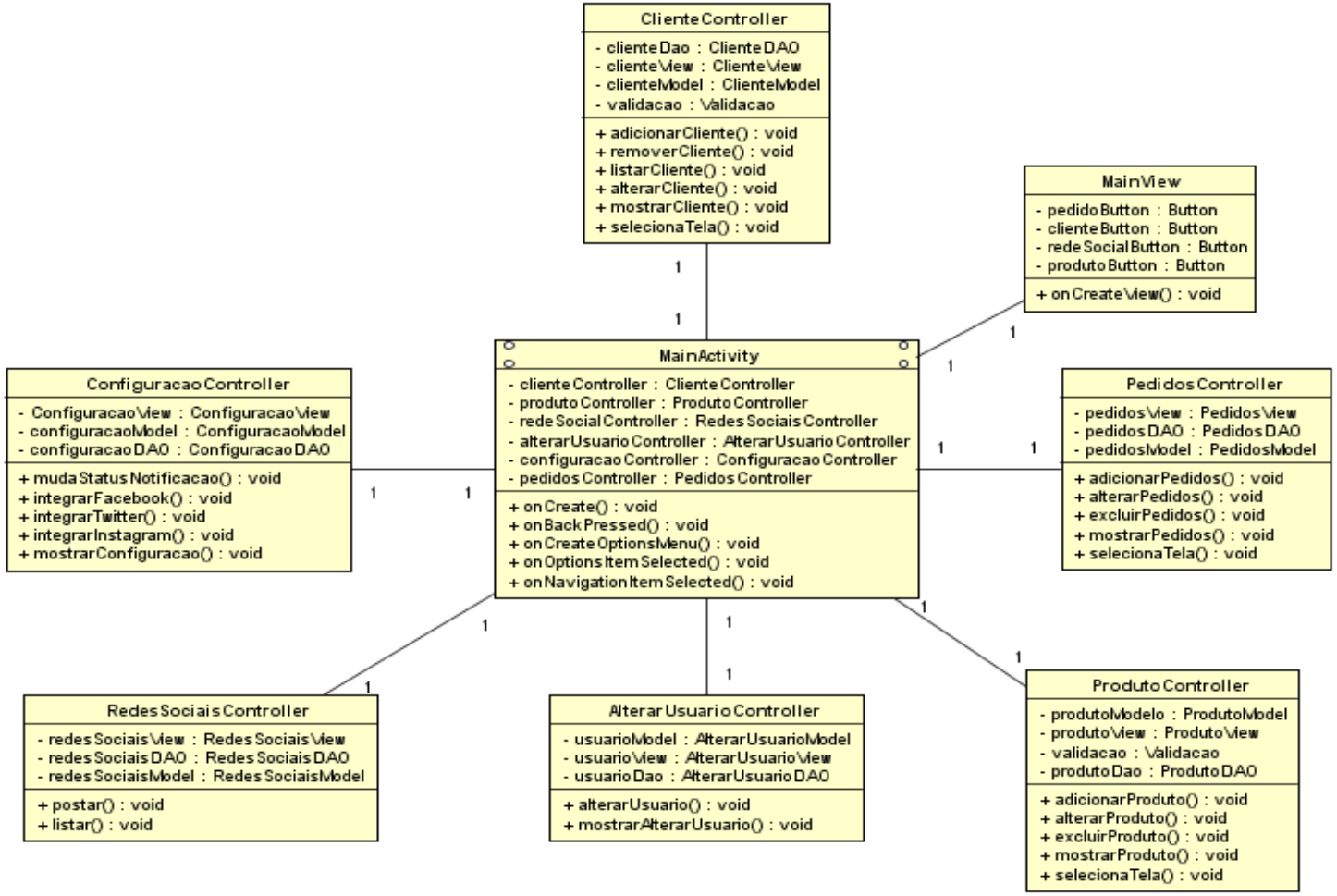
Método	**Descrição**
adicionaUsuario	Adiciona o usuário no banco de dados através da classe UsuarioAjudante. Recebe como parâmetro um modelo de usuário. Retorna um booleano informando se o usuário foi adicionado com sucesso ou não.

Fonte: Elaborado pelos autores

UsuarioContrato: Classe já descrita anteriormente.

UsuarioAjudante: Classe já descrita anteriormente.

Figura 7 – Protótipo da tela de início do sistema.



Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

MainActivity: Possui a responsabilidade de servir como uma base para o carregamento do *layout*, assim como carregar o menu e as respectivas telas de funcionalidades no sistema, como produtos, clientes, usuário, redes sociais, dentre outras. Os métodos identificados para esta classe foram:

Tabela 20 – Descrição dos Métodos da Classe: *MainActivity*

Método	**Descrição**
onCreate	Responsável pela criação da interface gráfica da tela principal.
onBackPressed	Captura o evento de clique de voltar do celular. Responsável por carregar a tela anterior a atual.
onCreateOptionsMenu	Responsável pela criação do menu do sistema.
onOptionsItemSelected	Responsável pela captura do item do menu clicado.
onNavigationItemSelected	Responsável por realizar a navegação e transição para o item selecionado.

Fonte: Elaborado pelos autores

MainView: Tela de visualização inicial do sistema, contendo botões que possibilitam acesso mais rápido a algumas funcionalidades. Os métodos identificados para esta classe foram:

Tabela 21 – Descrição dos Métodos da Classe: *MainView*

Método	**Descrição**
onCreateView	Responsável pela criação da interface gráfica da tela inicial.

Fonte: Elaborado pelos autores

ClienteController: Será abordada mais adiante.

PedidosController: Será abordada mais adiante.

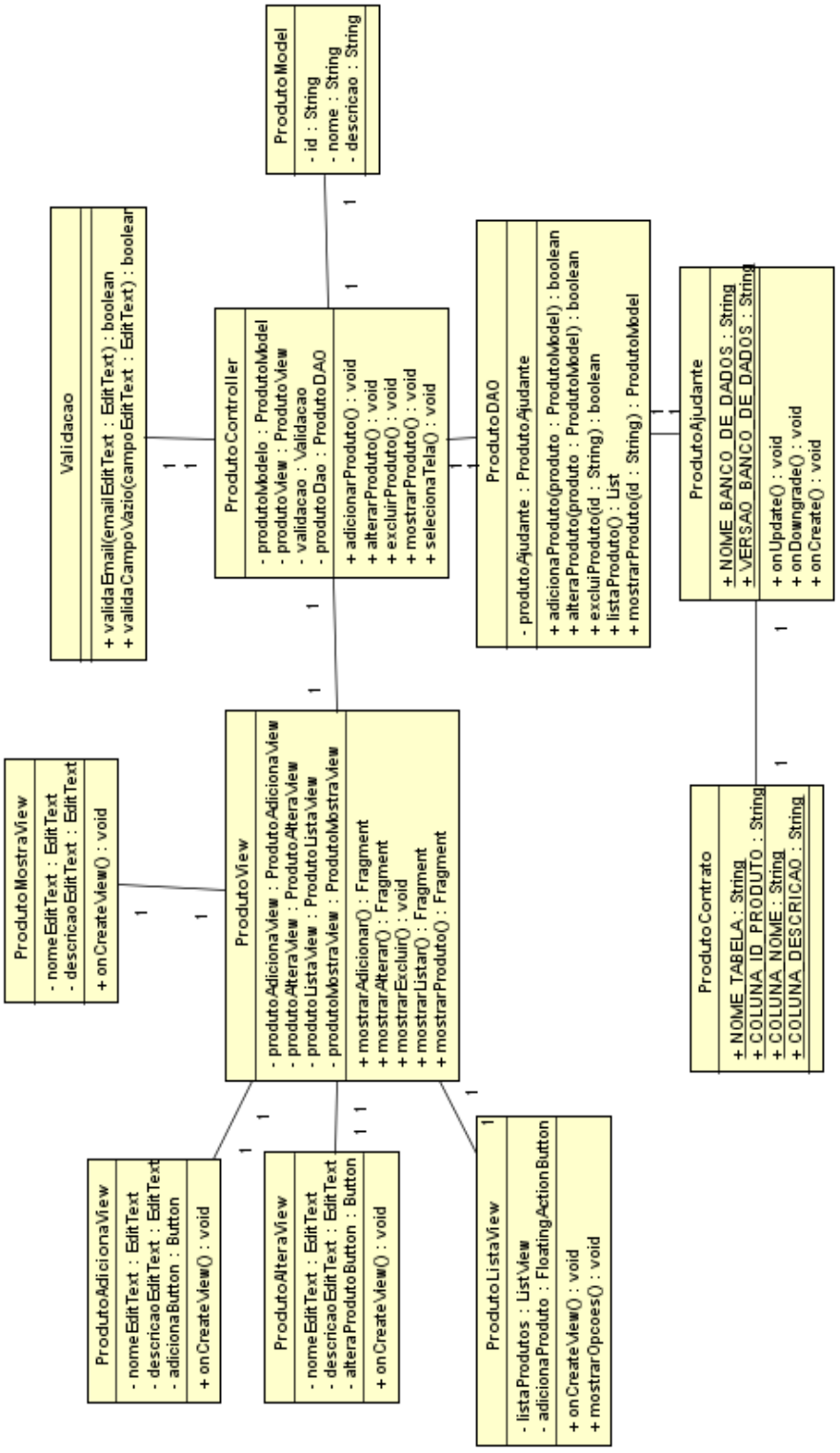
ProdutoController: Será abordada mais adiante.

AlterarUsuarioController: Será abordada mais adiante.

RedesSociaisController: Será abordada mais adiante.

ConfiguracaoController: Será abordada mais adiante.

Figura 8 – Protótipo de Produto.



Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

ProdutoView: Interface do sistema responsável pelo gerenciamento das telas de adicionar, alterar, deletar, mostrar e listar produtos. Os métodos identificados para esta classe foram:

Tabela 22 – Descrição dos Métodos da Classe: *ProdutoView*

Método	**Descrição**
mostrarAdicionar	Responsável pelo carregamento da tela de adição de produtos no sistema. Retorna um fragmento.
mostrarAlterar	Responsável pelo carregamento da tela de alteração de produtos do sistema. Retorna um fragmento.
mostrarExcluir	Responsável pelo carregamento da mensagem de confirmação e em sequência exclusão de um produto.
mostrarListar	Responsável pelo carregamento da tela de listar produtos.
mostrarProduto	Responsável por mostrar um produto em específico cadastrado.

Fonte: Elaborado pelos autores

ProdutoAdicionaView: Interface do sistema responsável pela adição do produto. Os métodos identificados para esta classe foram:

Tabela 23 – Descrição dos Métodos da Classe: *ProdutoAdicionaView*

Método	**Descrição**

onCreateView	Cria a interface da tela de adicionar produtos no sistema.

Fonte: Elaborado pelos autores

ProdutoAlteraView: Interface do sistema responsável pela alteração do produto. Os métodos identificados para esta classe foram:

Tabela 24 – Descrição dos Métodos da Classe: *ProdutoAlteraView*

Método	**Descrição**
onCreateView	Cria a interface da tela de alterar produtos no sistema.

Fonte: Elaborado pelos autores

ProdutoListaView: Interface do sistema responsável pela listagem dos produtos. Os métodos identificados para esta classe foram:

Tabela 25 – Descrição dos Métodos da Classe: *ProdutoListaView*

Método	**Descrição**
onCreateView	Cria a interface da tela de listagem de produtos no sistema.
mostrarOpcoes	Método que é responsável por mostrar as opções relativas a cada produto (mostrar, alterar e excluir produto).

Fonte: Elaborado pelos autores

ProdutoMostraView: Interface do sistema responsável pela visualização de um produto específico cadastrado na lista de produtos. Os métodos identificados para esta classe foram:

Tabela 26 – Descrição dos Métodos da Classe: *ProdutoMostraView*

Método	**Descrição**
onCreateView	Cria a interface da tela de mostrar produtos no sistema.

Fonte: Elaborado pelos autores

Validacao: Classe de validação do sistema que garante que os campos não estarão vazios e que o *e-mail* será válido. Os métodos identificados para esta classe foram:

Tabela 27 – Descrição dos Métodos da Classe: Validacao

Método	**Descrição**
validaEmail	Recebe como parâmetro um campo de edição de texto, o qual deverá ser validado e, se estiver válido o *e-mail* informado, nenhum erro será alertado ao usuário. Retorna um valor booleano.
validaCampoVazio	Recebe como parâmetro um campo de edição de texto responsável por validar se o conteúdo do campo é nulo ou está vazio. Retorna um valor booleano.

Fonte: Elaborado pelos autores

ProdutoController: Classe responsável por atualizar o modelo de dados assim como ouvir os eventos realizados pela interface gráfica. Os métodos identificados para esta classe foram:

Tabela 28 – Descrição dos Métodos da Classe: *ProdutoController*

Método	**Descrição**
adicionarProduto	Realiza a adição do produto utilizando a classe ProdutoDAO.
alterarProduto	Realiza a alteração do produto utilizando a classe ProdutoDAO.
excluirProduto	Realiza a exclusão do produto utilizando a classe *ProdutoDAO*.
mostrarProduto	Mostra produto utilizando a classe *ProdutoDAO*.
selecionaTela	Responsável pela seleção da tela de produtos no sistema.
listarProduto	Lista os produtos utilizando a classe *ProdutoDAO*.

Fonte: Elaborado pelos autores

ProdutoModel: Classe de modelo do produto, possuindo como atributos o *id*, nome e descrição.

ProdutoDAO: Classe que realiza a adição, alteração, exclusão, listagem e mostra os produtos, possuindo acesso direto ao banco de dados através da classe ProdutoAjudante. Os métodos identificados para esta classe foram:

Tabela 29 – Descrição dos Métodos da Classe: *ProdutoDAO*

Método	**Descrição**
adicionaProduto	Recebe como parâmetro um modelo de produto e realiza a gravação dele no banco de dados na tabela de produtos.
alteraProduto	Recebe como parâmetro um modelo de produto e realiza a alteração do produto no banco de dados em sua respectiva tabela.
excluiProduto	Realiza a exclusão de um produto no banco de dados. O parâmetro recebido é o *id* do produto.
listaProduto	Realiza a listagem de todos os produtos cadastrados no banco de dados.
mostraProduto	Realiza a consulta de um produto específico no banco de dados a partir do *id* que é recebido como parâmetro.

Fonte: Elaborado pelos autores

ProdutoContrato: Classe que representa a tabela do banco de dados do produto.

ProdutoAjudante: Tem por objetivo fornecer o acesso direto ao banco de dados. Os métodos identificados para esta classe foram:

Tabela 30 – Descrição dos Métodos da Classe: ProdutoAjudante

Método	**Descrição**
onUpdate	Responsável pela atualização da tabela de produtos no banco de dados.

onDowngrade	Responsável pela diminuição da versão da tabela de produtos no banco de dados.
onCreate	Responsável pela criação da tabela de produtos no banco de dados.

Fonte: Elaborado pelos autores

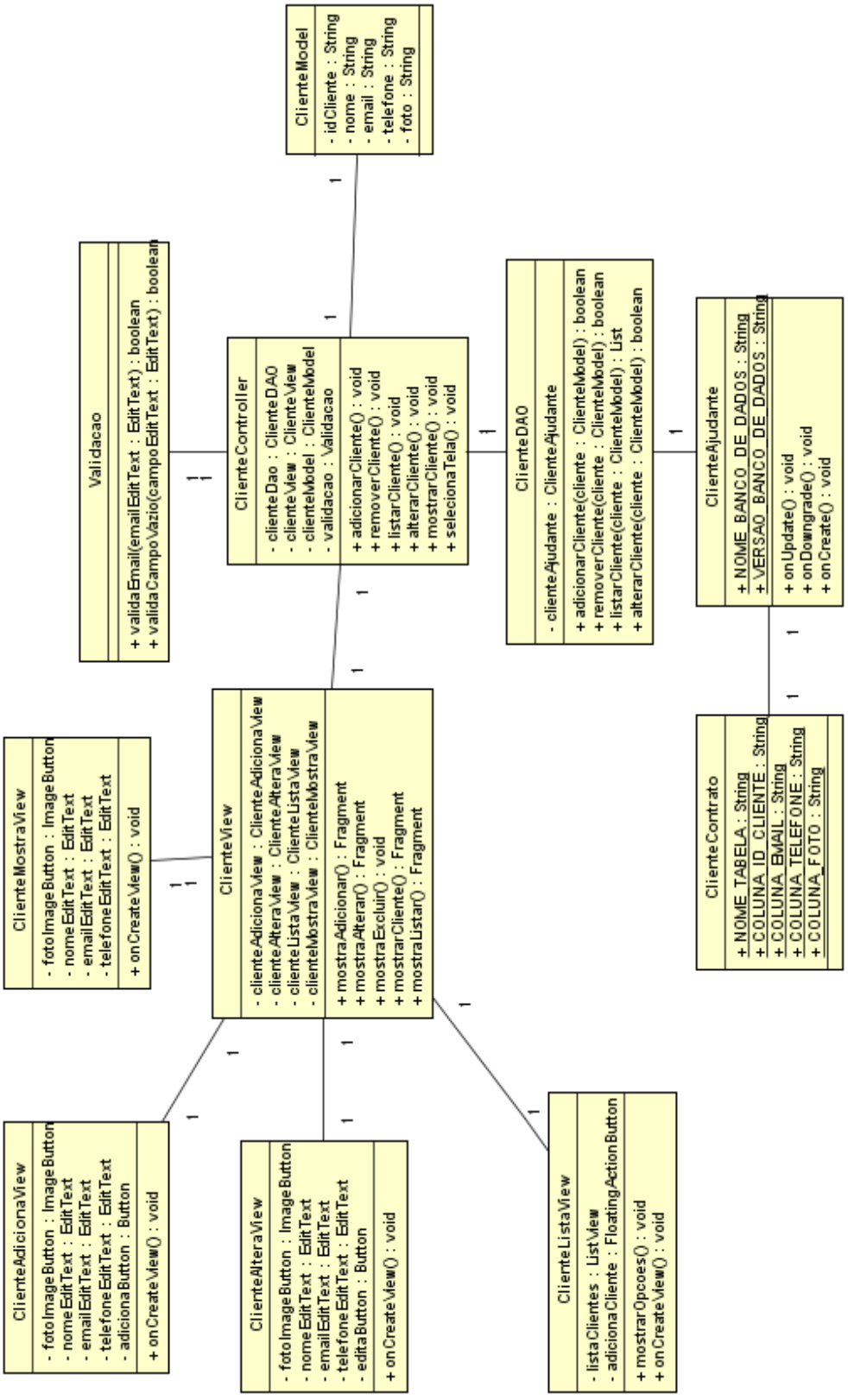


Figura 9 – Protótipo de Produto.

Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

ClienteView: Interface do sistema responsável pelo gerenciamento das telas de adicionar, alterar, deletar, mostrar e listar clientes. Os métodos identificados para esta classe foram:

Tabela 31 – Descrição dos Métodos da Classe: *ClienteView*

Método	**Descrição**
mostraAdicionar	Responsável pelo carregamento da tela de adição de clientes no sistema. Retorna um fragmento.
mostraAlterar	Responsável pelo carregamento da tela de alteração de clientes do sistema. Retorna um fragmento.
mostraExcluir	Responsável pelo carregamento da mensagem de confirmação e em sequência exclusão de um cliente.
mostraListar	Retorna a tela de listagem de clientes.
mostraCliente	Responsável por mostrar um cliente em específico cadastrado.

Fonte: Elaborado pelos autores

ClienteAdicionaView: Interface do sistema responsável pela adição do cliente. Os métodos identificados para esta classe foram:

Tabela 32 – Descrição dos Métodos da Classe: *ClienteAdicionaView*

Método	**Descrição**
onCreateView	Cria a interface da tela de adicionar clientes no sistema.

Fonte: Elaborado pelos autores

ClienteAlteraView: Interface do sistema responsável pela alteração do cliente. Os métodos identificados para esta classe foram:

Tabela 33 – Descrição dos Métodos da Classe: *ClienteAlteraView*

Método	**Descrição**
onCreateView	Cria a interface da tela de alterar clientes no sistema.

Fonte: Elaborado pelos autores

ClienteListaView: Interface do sistema responsável pela listagem dos clientes. Os métodos identificados para esta classe foram:

Tabela 34 – Descrição dos Métodos da Classe: *ClienteListaView*

Método	**Descrição**
onCreateView	Cria a interface da tela de listagem de clientes no sistema.
mostrarOpcoes	Método que é responsável por mostrar as opções relativas a cada cliente (mostrar, alterar e excluir cliente).

Fonte: Elaborado pelos autores

ClienteMostraView: Interface do sistema responsável pela visualização de um cliente específico cadastrado na lista de clientes. Os métodos identificados para esta classe foram:

Tabela 35 – Descrição dos Métodos da Classe: *ClienteMostraView*

Método	**Descrição**
onCreateView	Cria a interface da tela de mostrar clientes no sistema.

Fonte: Elaborado pelos autores

Validacao: Classe anteriormente explicada.

ClienteController: Classe responsável por atualizar o modelo de dados assim como ouvir os eventos realizados pela interface gráfica. Os métodos identificados para esta classe foram:

Tabela 36 – Descrição dos Métodos da Classe: *ClienteController*

Método	**Descrição**
adicionarCliente	Adiciona um cliente sendo auxiliado pela classe *ClienteDAO*.
alterarCliente	Altera um cliente sendo auxiliado pela classe *ClienteDAO*.
removerCliente	Remove um cliente utilizando a classe *ClienteDAO*.
mostrarCliente	Mostra um cliente a partir da classe *ClienteDAO*.
selecionaTela	Seleciona a tela de clientes (adicionar, alterar, listar, mostrar).
listarCliente	Responsável pela listagem de clientes.

Fonte: Elaborado pelos autores

ClienteModel: Classe de modelo do cliente, possuindo como atributos o *idCliente*, nome, *e-mail*, telefone e foto. Os métodos identificados para esta classe foram:

ClienteDAO: Classe que realiza a adição, alteração, exclusão, listagem e mostra os clientes, possuindo acesso direto ao banco de dados através da classe ClienteAjudante. Os métodos identificados para esta classe foram:

Tabela 37 – Descrição dos Métodos da Classe: *ClienteDAO*

Método	**Descrição**
adicionarCliente	Recebe como parâmetro um modelo de produto e realiza a gravação do mesmo no banco de dados na tabela de produtos.
removerCliente	Realiza a exclusão de um cliente no banco de dados. O parâmetro recebido é o *id* do cliente.
listarCliente	Realiza a listagem de todos os clientes cadastrados no banco de dados.
alterarCliente	Recebe como parâmetro um modelo de cliente e realiza a alteração do cliente no banco de dados em sua respectiva tabela.
mostrarCliente	Realiza a consulta de um cliente específico no banco de dados a partir do *id* que é recebido como parâmetro. É retornado um modelo do cliente.

Fonte: Elaborado pelos autores

ClienteContrato: Classe que representa a tabela do banco de dados do cliente.

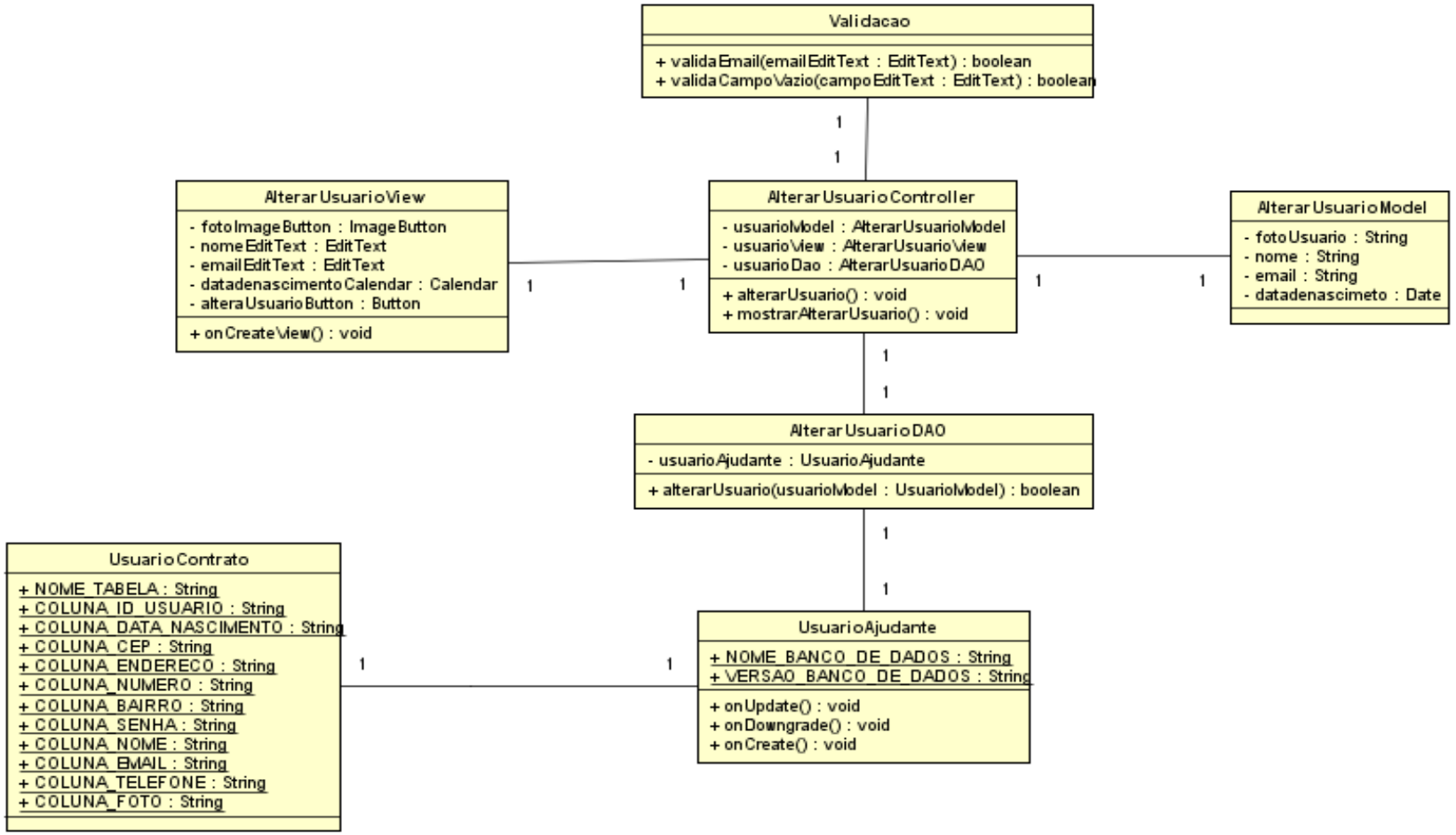
ClienteAjudante: Tem por objetivo fornecer o acesso direto ao banco de dados. Os métodos identificados para esta classe foram:

Tabela 38 – Descrição dos Métodos da Classe: ClienteAjudante

Método	**Descrição**
onUpdate	Responsável pela atualização da tabela de clientes no banco de dados.
onDowngrade	Responsável pela diminuição da versão da tabela de clientes no banco de dados.
onCreate	Responsável pela criação da tabela de clientes no banco de dados.

Fonte: Elaborado pelos autores

Figura 10 – Protótipo de Usuário.



Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

AlterarUsuarioView: Interface do sistema responsável pelo gerenciamento da tela de alterar usuários. O método identificado para esta classe foi:

Tabela 39 – Descrição dos Métodos da Classe: *AlteraUsuarioView*

Método	**Descrição**
onCreateView	Cria a interface das informações do usuário no sistema.

Fonte: Elaborado pelos autores

AlterarUsuarioController: Classe responsável por atualizar o modelo de dados assim como ouvir os eventos realizados pela interface gráfica. O método identificado para esta classe foi:

Tabela 40 – Descrição dos Métodos da Classe: *AlterarUsuarioController*

Método	**Descrição**
alterarUsuario	Realiza a alteração do usuário a partir da classe *AlteraUsuarioDAO*.
mostraAlterarUsuario	Mostra a tela de alteração de usuário.

Fonte: Elaborado pelos autores

Validacao: Classe anteriormente explicada.

AlterarUsuarioModel: Classe de modelo do Usuario, possuindo como atributos o fotoUsuario, nome, e-mail e datadenascimento. Essa classe não possui métodos.

AlterarUsuarioDAO: Classe que realiza a alteração do usuário possuindo acesso direto ao banco de dados através da classe UsuarioAjudante. O método identificado para esta classe foi:

Tabela 41 – Descrição dos Métodos da Classe: *AlteraUsuarioDAO*

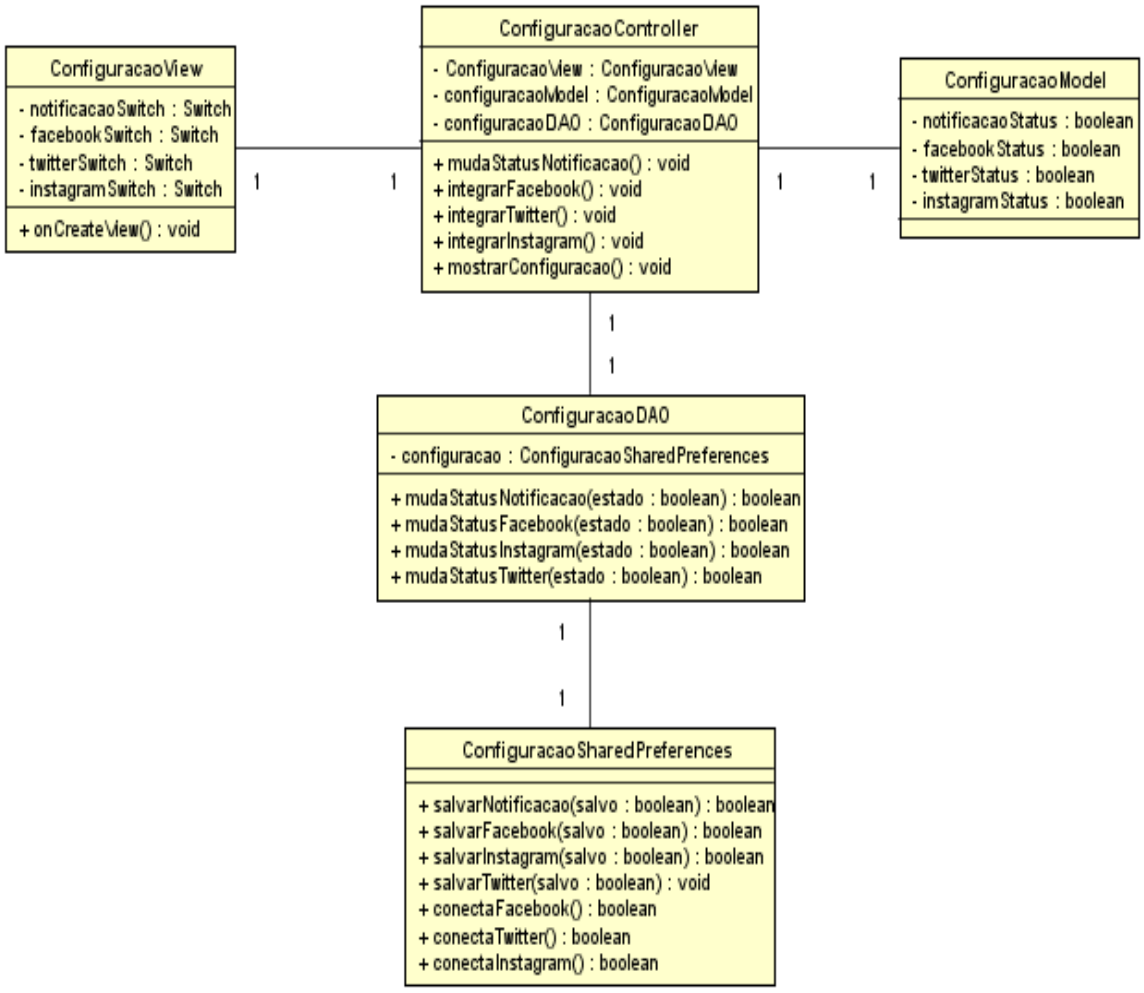
Método	**Descrição**
alterarUsuario	Recebe como parâmetro um modelo de usuário e realiza a alteração do usuário no banco de dados em sua respectiva tabela.

UsuarioContrato: Classe anteriormente explicada.

UsuarioAjudante: Classe anteriormente explicada.

Fonte: Elaborado pelos autores

Figura 11 – Protótipo de configurações.



Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

ConfiguracaoView: Interface do sistema responsável pelo gerenciamento das notificações do sistema e conexão com redes sociais. O método identificado para esta classe foi:

Tabela 42 – Descrição dos Métodos da Classe: *ConfiguracaoView*

Método	**Descrição**

onCreateView	Cria a interface do *status* das configurações e das redes sociais do usuário no sistema.

Fonte: Elaborado pelos autores

ConfiguracaoController: Classe responsável por atualizar o modelo de dados assim como ouvir os eventos realizados pela interface gráfica. Os métodos identificados para esta classe foram:

Tabela 43 – Descrição dos Métodos da Classe: *ConfiguracaoController*

Método	**Descrição**
mudaStatusNotificacao	Altera o status da notificação no sistema
integrarFacebook	Faz a integração do *Facebook* com o sistema (mudando o status para ativo ou falso).
integrarTwitter	Faz a integração do *Twitter* com o sistema (mudando o status para ativo ou falso).
integrarInstagram	Faz a integração do *Instagram* com o sistema (mudando o status para ativo ou falso).
mostrarConfiguracao	Mostra a tela de configuração.

Fonte: Elaborado pelos autores

ConfiguracaoModel: Classe de modelo das configurações, possuindo como atributos o configuracaoStatus, *facebookStatus*, *instagramStatus* e *twitterStatus*. Essa classe não possui métodos.

ConfiguracaoDAO: Classe que realiza a alteração nas configurações e nas conexões das redes sociais conectadas utilizando

ConfiguracaoSharedPreference. Os métodos identificados para esta classe foram:

Tabela 44 – Descrição dos Métodos da Classe: *ConfiguracaoDAO*

Método	**Descrição**
mudaStatusNotificacao	Salva a alteração do *status* da notificação utilizando a classe *ConfiguracaoSharedPreference*
mudaStatusFacebook	Salva a alteração do *status* da conexão com o *Facebook* utilizando a classe *ConfiguracaoSharedPreference*
mudaStatusTwitter	Salva a alteração do *status* da conexão com o *Twitter* utilizando a classe *ConfiguracaoSharedPreference*
mudaStatusInstagram	Salva a alteração do *status* da conexão com o *Instagram* utilizando a classe *ConfiguracaoSharedPreference*

Fonte: Elaborado pelos autores

ConfiguracaoSharedPreference: Classe que salva no sistema o estado de notificações e conexões com redes sociais (*Facebook*, *Twitter* e *Instagram*) em formato booleano. Os métodos identificados para esta classe foram:

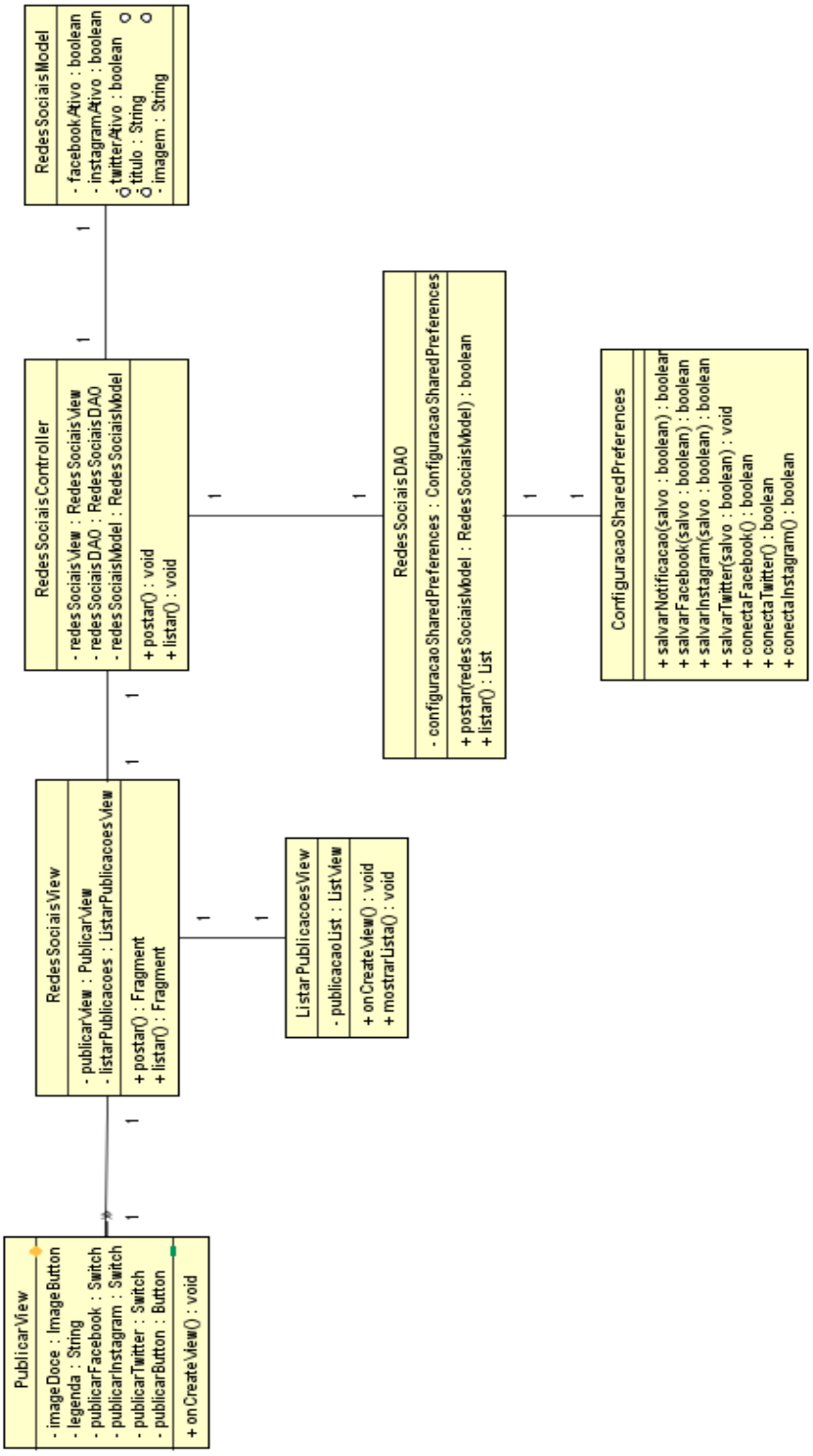
Tabela 45 – Descrição dos Métodos da Classe: *ConfiguracaoSharedPreference*

Método	**Descrição**
salvarNotificacao	Recebe como parâmetro o *status* salvo e salva a

	alteração do status da notificação
salvarFacebook	Recebe como parâmetro o *status* salvo e salva a alteração do status do *Facebook*
salvarTwitter	Recebe como parâmetro o *status* salvo e salva a alteração do status do *Twitter*
salvarInstagram	Recebe como parâmetro o *status* salvo e salva a alteração do status do *Instagram*
conectaFacebook	Realiza a conexão com o *Facebook*.
conectaTwitter	Realiza a conexão com o *Twitter*.
conectaInstagram	Realiza a conexão com o *Instagram*.

Fonte: Elaborado pelos autores

Figura 12 – Protótipo de redes sociais.



Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

RedesSociaisView: Interface do sistema responsável pelo gerenciamento das conexões do aplicativo e as redes sociais. Os métodos identificados para esta classe foram:

Tabela 46 – Descrição dos Métodos da Classe: *RedesSociaisView*

Método	**Descrição**
Listar	Invoca a tela de listagem em *ListarPublicacoesView*
Postar	Invoca a tela de postagem presente em *PublicarView*

Fonte: Elaborado pelos autores

PublicarView: Interface do sistema responsável pela publicação nas respectivas redes sociais. O método identificado para esta classe foi:

Tabela 47 – Descrição dos Métodos da Classe: *PublicarView*

Método	**Descrição**
onCreateView	Responsável pela criação da tela de publicação

Fonte: Elaborado pelos autores

ListarPublicacoesView: Interface do sistema responsável pela listagem das publicações realizadas nas redes sociais. Os métodos identificados para esta classe foram:

Tabela 48 – Descrição dos Métodos da Classe: *ListarPublicacoesView*

Método	**Descrição**
onCreateView	Responsável pela criação da tela de listagem
mostrarLista	Mostra a lista de postagens carregadas

Fonte: Elaborado pelos autores

RedesSociaisController: Classe responsável por atualizar o modelo de dados assim como ouvir os eventos realizados pela interface gráfica. Os métodos identificados para esta classe foram:

Tabela 49 – Descrição dos Métodos da Classe: *RedesSociaisController*

Método	**Descrição**
Postar	Realiza a ação de postagem utilizando a classe RedesSociaisDAO.
Listar	Realiza a ação de listar as postagens utilizando a classe RedesSociaisDAO.

Fonte: Elaborado pelos autores

RedesSociaisModel: Classe de modelo das redes sociais, possuindo como atributos o *facebookAtivo*, *instagramAtivo*, *twitterAtivo*, título e imagem. Essa classe não possui métodos.

ConfiguracaoSharedPreferences: Anteriormente explicada.

RedesSociaisDAO: Classe que realiza a alteração redes sociais como sua conexão com acesso direto usando as classes *APIFacebook*, *APIInstagram* e *APITwitter*. Os métodos identificados para esta classe foram:

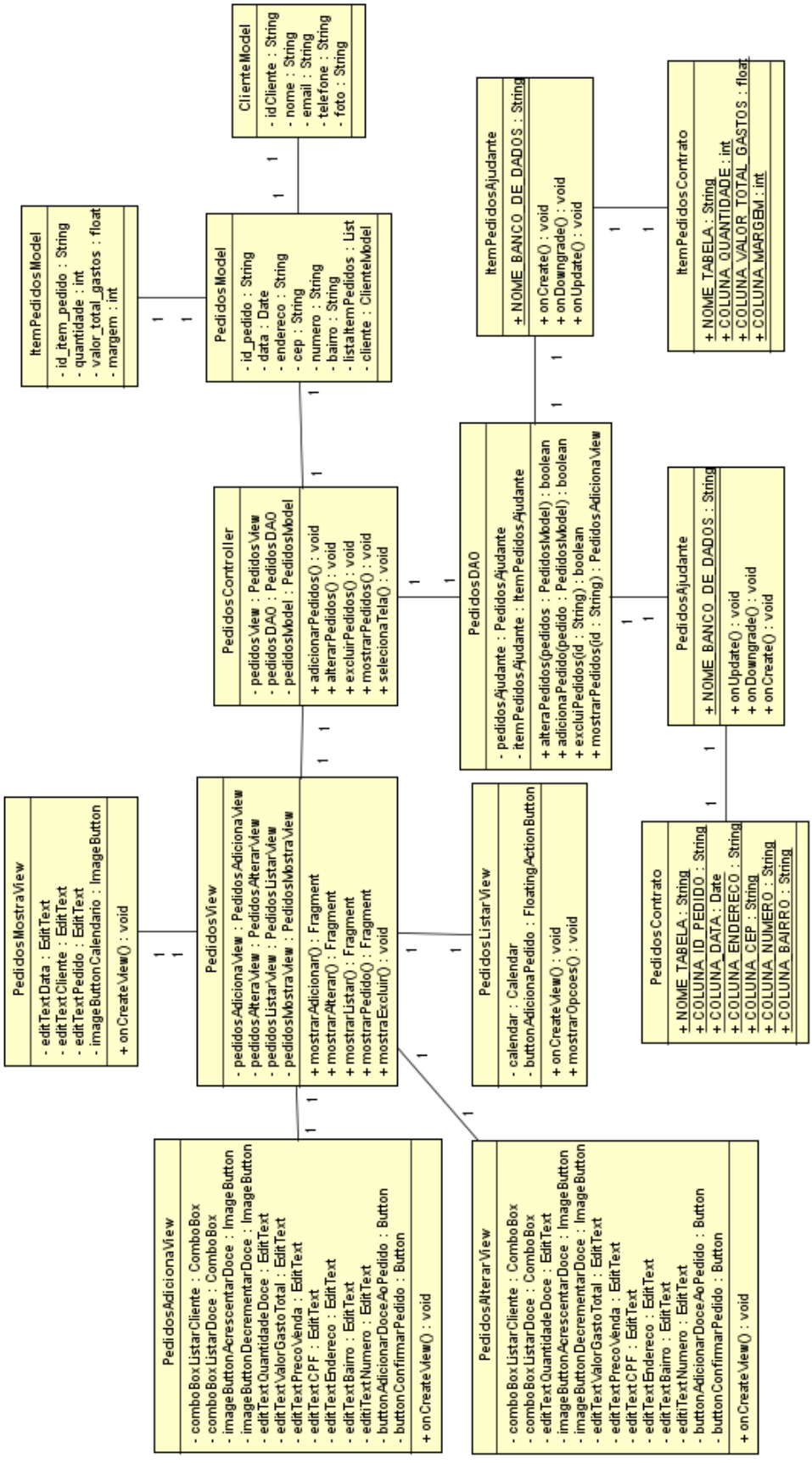
Tabela 50 – Descrição dos Métodos da Classe: *RedesSociaisDAO*

Método	**Descrição**
Postar	Realiza a postagem diretamente na respectiva rede social.
Listar	Lista as postagens das redes sociais integradas

Fonte: Elaborado pelos autores

ConfiguracaoSharedPreferences: Anteriormente explicada.

Figura 13 – Protótipo de Pedidos.



Fonte: Elaborado pelos autores

Neste modelo foram identificadas as seguintes classes:

PedidosView: Interface do sistema responsável pelo gerenciamento das telas de adicionar, alterar, deletar, mostrar e listar pedidos. Os métodos identificados para essa classe foram:

Tabela 51 – Descrição dos Métodos da Classe: *PedidosView*

Método	**Descrição**
mostrarAdicionar	Responsável pelo carregamento da tela de adição de pedidos no sistema. Retorna um fragmento.
mostrarAlterar	Responsável pelo carregamento da tela de alteração de pedidos do sistema. Retorna um fragmento.
mostrarExcluir	Responsável pelo carregamento da mensagem de confirmação e em sequência exclusão de um pedido.
mostrarPedido	Responsável por mostrar um pedido em específico cadastrado.

Fonte: Elaborado pelos autores

PedidosAdicionaView: Interface do sistema responsável pela adição do pedido. O método identificado para esta classe foi:

Tabela 52 – Descrição dos Métodos da Classe: *PedidosAdicionaView*

Método	**Descrição**
onCreateView	Cria a interface da tela de adicionar pedidos no sistema.

Fonte: Elaborado pelos autores

PedidosAlterarView: Interface do sistema responsável pela alteração dos pedidos. O método identificado para esta classe foi:

Tabela 53 – Descrição dos Métodos da Classe: *PedidosAlterarView*

Método	**Descrição**
onCreateView	Cria a interface da tela de alterar pedidos no sistema.

Fonte: Elaborado pelos autores

PedidosMostraView: Interface do sistema responsável pela visualização de um pedido específico cadastrado na lista de pedidos. O método identificado para esta classe foi:

Tabela 54 – Descrição dos Métodos da Classe: *PedidosMostraView*

Método	**Descrição**
onCreateView	Cria a interface da tela de mostrar pedidos no sistema.

Fonte: Elaborado pelos autores

PedidosListarView: Interface do sistema responsável pela listagem dos pedidos em uma interface de calendário. O método identificado para esta classe foi:

Tabela 55 – Descrição dos Métodos da Classe: *PedidosListarView*

Método	**Descrição**
onCreateView	Cria a interface da tela de listagem de pedidos no sistema.
mostraOpcoes	Mostra as opções de alterar, excluir e alterar um pedido.

Fonte: Elaborado pelos autores

PedidosController: Realiza a comunicação entre a interface gráfica do sistema, juntamente ao modelo de dados, ouvindo eventos realizados pela interface. Também possui acesso direto as classes de manipulação do banco de dados (*DAOs*). Os métodos identificados para essa classe foram:

Tabela 56 – Descrição dos Métodos da Classe: *PedidosController*

Método	**Descrição**
adicionarPedidos	Adiciona um pedido no sistema utilizando a classe *PedidosDAO*.
alterarPedidos	Altera um pedido no sistema utilizando a classe *PedidosDAO*.
excluirPedidos	Exclui um pedido específico no sistema utilizando a classe *PedidosDAO*.
mostrarPedidos	Mostra um pedido específico no sistema utilizando a classe *PedidosDAO*.
selecionaTela	Responsável pela seleção de tela em produtos.

Fonte: Elaborado pelos autores

PedidosModel: Classe de modelo dos pedidos, possuindo como atributos o *id*, data, endereço, cep, número e bairro.

PedidosDAO: Classe que realiza a adição, alteração, exclusão e mostrar os pedidos, possuindo acesso direto ao banco de dados através da classe PedidosAjudante. Os métodos identificados para esta classe foram:

Tabela 57 – Descrição dos Métodos da Classe: *PedidosDAO*

Método	**Descrição**
adicionaPedidos	Recebe como parâmetro um modelo de pedido e realiza a gravação do mesmo no banco de dados na tabela de pedidos.
alteraPedidos	Recebe como parâmetro um modelo de pedidos e realiza a alteração dos pedidos no banco de dados em sua respectiva tabela.
excluiPedidos	Realiza a exclusão de um pedido no banco de dados. O parâmetro recebido é o *id* do pedido.
mostraPedidos	Realiza a consulta de um pedido específico no banco de dados a partir do *id* que é recebido como parâmetro.

Fonte: Elaborado pelos autores

PedidosContrato: Classe que representa a tabela do banco de dados dos pedidos.

PedidosAjudante: Tem por objetivo fornecer o acesso direto ao banco de dados. Os métodos identificados para esta classe foram:

Tabela 58 – Descrição dos Métodos da Classe: PedidosAjudante

Método	Descrição
onUpdate	Responsável pela atualização da tabela de pedidos no banco de dados.
onDowngrade	Responsável pela diminuição da versão da tabela de pedidos no banco de dados.
onCreate	Responsável pela criação da tabela de pedidos no banco de dados.

Fonte: Elaborado pelos autores

ItemPedidosModel: Classe de modelo dos pedidos, possuindo como atributos o *id*, valor total gastos, quantidade e margem.

ItemPedidosContrato: Classe que representa a tabela do banco de dados dos itens de pedidos.

ItemPedidosAjudante: Tem por objetivo fornecer o acesso direto ao banco de dados. Os métodos identificados para esta classe foram:

Tabela 59 – Descrição dos Métodos da Classe: ItemPedidosAjudante

Método	Descrição
onUpdate	Responsável pela atualização da tabela de item de pedidos no banco de dados.
onDowngrade	Responsável pela diminuição da versão da tabela de item de pedidos no banco de dados.
onCreate	Responsável pela criação da tabela de item de pedidos no banco de dados.

Fonte: Elaborado pelos autores

3.6 Telas do Aplicativo

3.6.1 Tela de *Login*

A Figura 14 apresenta a tela de login do sistema, na qual são apresentados os campos de e-mail e senha. Há também um botão de login responsável pela ação de entrar no sistema, verificado se os dados informados no login e senha do aplicativo já estão cadastrados. Caso os dados não estejam cadastrados, uma mensagem será informada alertando o usuário que a conta não existe. Caso os dados estejam cadastrados, o usuário será redirecionado para a tela principal do sistema (painel). Também é possível criar um usuário clicando em cadastrar usuário.

Figura 14 – Tela de *Login*.

Fonte: Elaborado pelos autores

A tela de login apresentada na Figura 14 é composta por:

- **Campo de e-mail:** Para a digitação do login do usuário. O login deverá ser um e-mail.
- **Campo de senha:** Para a digitação da senha do usuário.
- **Botão de login:** Botão que realiza a ação de login no sistema.
- **Texto “Não possui usuário? Cadastre-se agora!”:** Redireciona o usuário para a tela de cadastro do sistema.

3.6.2 Tela de Cadastro

A Figura 15 apresenta a tela de cadastro de nova conta no sistema, na qual são apresentados os campos de e-mail, nome, senha e confirmação de senha. Há também um botão de cadastro responsável pela ação de cadastrar os dados informados pelo usuário no sistema.

Figura 15 – Tela de Cadastro.

Fonte: Elaborado pelos autores

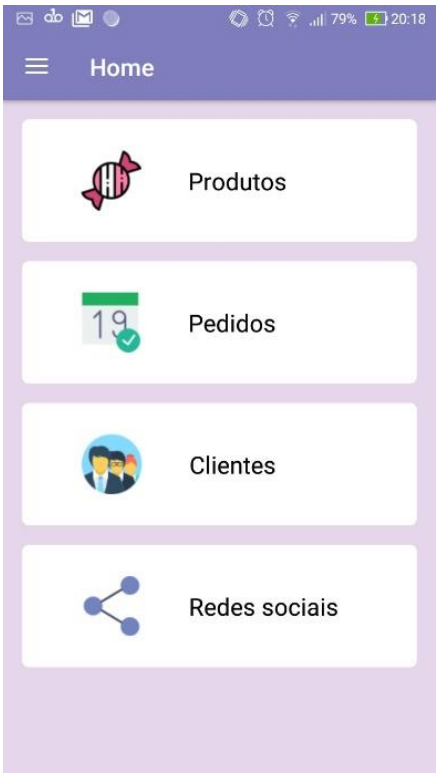
A tela de cadastro apresentada na Figura 15 é composta por:

- **Campo de nome:** Para a digitação do nome do usuário.
- **Campo de e-mail:** Para a digitação do login do usuário. O login deverá ser um e-mail.
- **Campo de senha:** Para digitação da senha da conta do usuário.
- **Campo de confirmação de senha:** Para a digitação da senha da conta do usuário (confirmação).
- **Botão "Cadastrar":** Cadastra o usuário no sistema com base nas informações preenchidas.

3.6.3 Tela de Painel Inicial

A Figura 16 apresenta a tela de painel inicial do sistema. A tela de painel inicial é a tela inicial do sistema que é carregada logo após o login ou cadastro com sucesso do usuário. A tela possui uma série de atalhos para as outras funcionalidades do sistema.

Figura 16 – Tela de Painel Inicial.



Fonte: Elaborado pelos autores

A tela de painel inicial apresentada na Figura 16 é composta por:

- **Botão de produtos:** Direciona o usuário para a tela de listagem de produtos.
- **Botão de pedidos:** Direciona o usuário para a tela de listagem de pedidos.
- **Botão de clientes:** Direciona o usuário para a tela de listagem de clientes.
- **Botão de redes sociais:** Direciona o usuário para as postagens realizadas nas telas de *Facebook*, *Twitter* e *Instagram*.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.4 Tela de Alteração de Usuário

A Figura 17 apresenta a tela de Alterar Usuário. A tela é apresentada campos para adição de foto, nome do usuário, e-mail, data de nascimento, CEP, endereço, bairro, número e botão de edição.

Figura 17 – Tela de Alteração de Usuário.

Fonte: Elaborado pelos autores

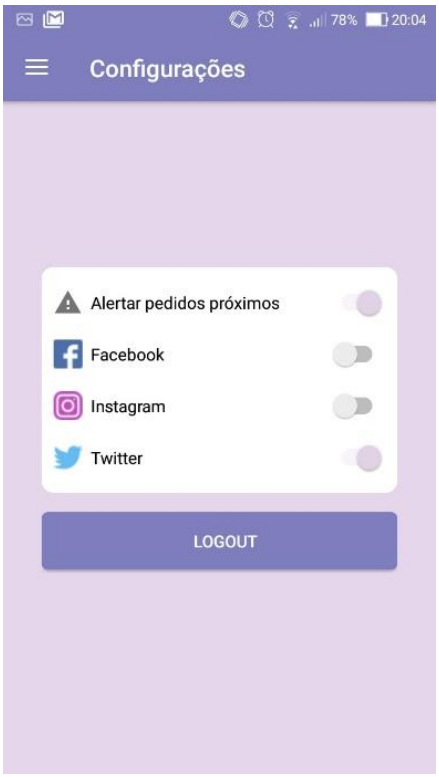
A tela de alterar usuário apresentada na Figura 17 é composta por:

- **Campo de foto:** Para que o usuário possa inserira a sua foto ou logo da empresa.
- **Campo nome:** Para que o usuário adicione o seu nome.
- **Campo e-mail:** Para que o usuário insira seu e-mail.
- **Campo data de nascimento:** Para que o usuário insira sua data de nascimento.
- **Campo CEP:** Para que o usuário insira seu CEP.
- **Campo endereço:** Para que o usuário insira seu endereço.
- **Campo bairro:** Para que o usuário insira seu bairro.
- **Campo número:** Para que o usuário insira o número de sua residencia.
- **Botão editar:** Salva as informações digitadas em todos os campos.
- **Icone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.5 Tela de Configurações

A Figura 18 apresenta a tela de Configurações. A tela é apresentada 4 botões de switch.

Figura 18 – Tela de Configurações.



Fonte: Elaborado pelos autores

A tela de configurações apresentada na Figura 18 é composta por:

- **Botão de Switch ativar alarmes:** Quando este estiver ativado o celular recebe notificações de agendamentos de pedidos feitos próximo a suas datas de entrega.
- **Botão de Switch Facebook:** Quando este estiver ativado significa que o facebook está conectado ao aplicativo, caso seja a primeira conexão o usuário será direcionado para tela de login do Facebook.
- **Botão de Switch do Twitter:** Quando este estiver ativado o Twitter está conectado ao aplicativo, caso seja a primeira conexão o usuário será direcionado para tela de login do Twitter.
- **Botão de Switch do Instagram:** Quando este estiver tivado o Instagram está conectado ao aplicativo, caso seja a primeira

conexão o usuário será direcionado para tela de login do Instagram.

- **Icone de menu:** Utilizado para o acesso ao menu do aplicativo

3.6.6 Tela de Configurações: *Login* em Rede Social

A figura 19 apresenta a tela de login da rede social *Twitter*. Nessa tela é esperado que seja pedido ao usuário os campos de *login* e senha, pois o processo é realizado através da *API* do próprio *Twitter* e pode variar conforme haja atualização.

Figura 19 - Tela de Configurações: Login em Rede Social.

Fonte: Elaborado pelos autores

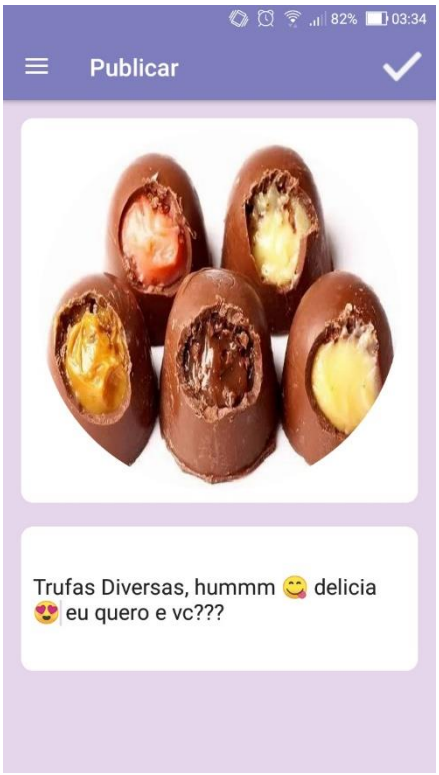
A tela de alterar usuário apresentada na Figura 19 é composta por:

- A tela apresentada é uma resposta proveniente da *API* do *Twitter*, sendo a *API* responsável por mostrar os componentes por ela necessários.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.7 Tela de Publicação

A figura 20 apresenta a tela de publicações. Nessa tela é exibido todas as publicações já feitas pelo usuário. Caso não haja uma rede social conectada exibe a tela de alerta de que não há uma rede social conectada, informando ao usuário os passos para que seja feito a conexão.

Figura 20 – Tela de Publicação.



Fonte: Elaborado pelos autores

A tela de publicações apresentada na Figura 20 é composta por:

- **Campo de imagem:** Mostra a imagem publicada na rede social (Facebook, Twitter ou Instagram)
- **Campo de Descrição:** Exibe a descrição da imagem publicada na rede social (Facebook, Twitter ou Instagram)
- **Botão Publicar:** Abre a tela de publicação.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.8 Tela de Produtos

A Figura 21 apresenta a tela de listar produtos. É exibido uma lista de todos os produtos cadastrados no sistema com um botão adicionar produto.

Figura 21 – Tela de Produtos.

Fonte: Elaborado pelos autores

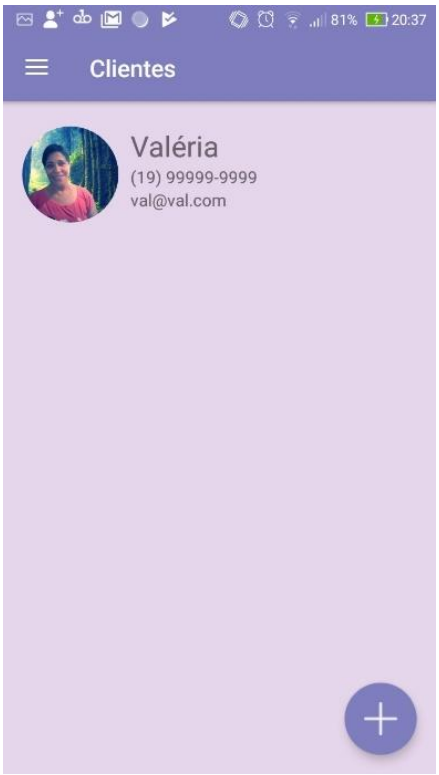
A tela de listar produtos apresentada na Figura 21 é composta por:

- **Lista de produtos:** Uma lista com todos os produtos cadastrados, quando o usuário clica em algum item este expande e mostra a descrição do produto clicado.
- **Botão adicionar produto:** Quando este é clicado o usuário e direcionado para a tela de adicionar produto.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.9 Tela de Clientes

A Figura 22 apresenta a tela de listagem de clientes cadastrados no sistema. Os clientes são apresentados em uma lista, contendo o nome, e-mail e telefone, juntamente a uma foto do cliente. Há também um botão de busca na barra de ação do aplicativo que realiza a busca pelos dados do cliente. Também há um botão com um ícone de adicionar cliente, que é responsável por enviar o usuário para a tela de adicionar cliente. No canto superior esquerdo, há um ícone de menu o qual o usuário poderá acessar o menu do aplicativo.

Figura 22 - Tela de Clientes.



Fonte: Elaborado pelos autores

A tela de adicionar clientes apresentada na Figura 22 é composta por:

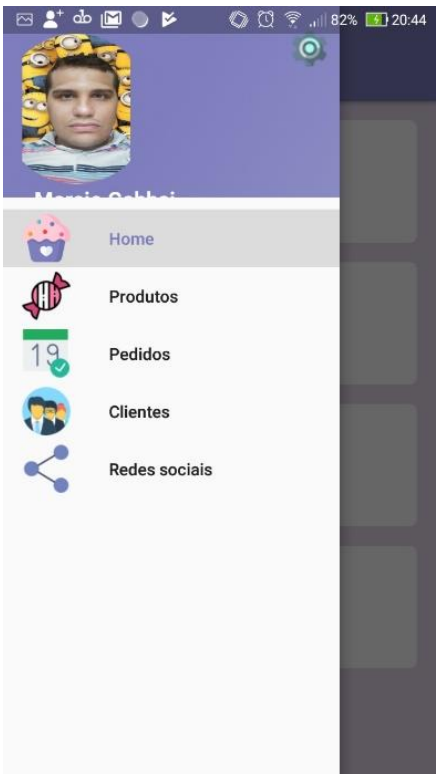
- **Campo de foto:** Campo para o usuário selecionar a foto do cliente.

- **Campo de nome:** Para a digitação do nome do cliente.
- **Campo de e-mail:** Para a digitação do e-mail do cliente.
- **Campo de telefone:** Para a digitação do telefone do cliente.
- **Botão com ícone adicionar cliente:** Adiciona o cliente no sistema, emite um alerta e direciona o usuário para a tela de listagem de clientes.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.10 Tela de Menu

A Figura 23 apresenta a tela de menu do sistema, responsável pela navegação do usuário. A partir dela, é possível acessar o painel, produtos, pedidos, clientes e redes sociais. Ao clicar sobre a foto do usuário, é possível editar o perfil da conta. Ao clicar na engrenagem é possível editar as configurações do usuário.

Figura 23 - Menu.



Fonte: Elaborado pelos autores

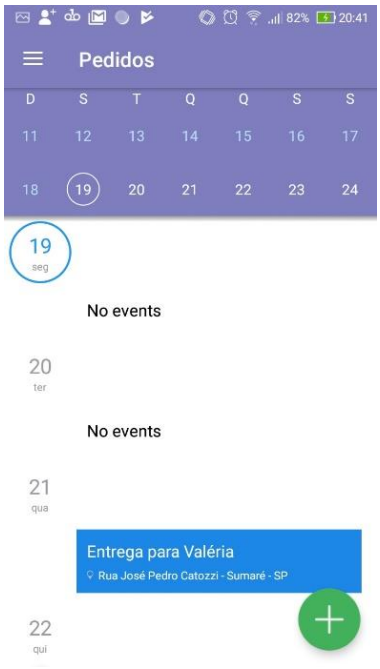
A tela de listagem de clientes apresentada na Figura 23 é composta por:

- **Foto do usuário:** É a foto do usuário, ao clicar sobre ela é possível a edição do perfil da conta.
- **Textos de nome e e-mail:** Textos com nome e e-mail do usuário.
- **Ícone de engrenagem:** Ícone que ao ser clicado leva o usuário as opções de configuração.
- **Item de home:** Leva o usuário a home do sistema.
- **Item de produtos:** Leva o usuário a listagem de produtos do sistema.
- **Item de pedidos:** Leva o usuário a listagem de pedidos do sistema.
- **Item de clientes:** Leva o usuário a listagem de clientes do sistema.
- **Item de redes sociais:** Leva o usuário as postagens realizadas no *Facebook*, *Twitter* e *Instagram*.

3.6.11 Tela de Pedidos

A Figura 24 apresenta a tela de pedidos do sistema, na qual é apresentado um calendário com todos os pedidos, que são carregados pelo sistema. Há também um botão de adicionar pedido responsável pela ação de chamar a tela de adicionar pedidos no sistema.

Figura 24 - Tela de Pedidos.



Fonte: Elaborado pelos autores

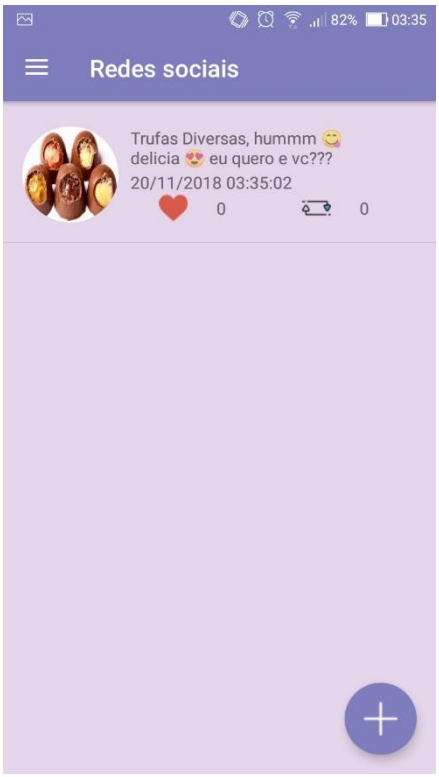
A tela de listar pedidos apresentada na Figura 24 é composta por:

- **Botão de adicionar pedido:** Botão que realiza a ação de chamar a tela de adicionar pedido no sistema.
- **Calendário:** Responsável por mostrar os pedidos cadastrados.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.12 Tela de Redes Sociais: Publicações

A figura 25 apresenta a tela de publicações. Nessa tela é exibido todas as publicações já feitas pelo usuário. Caso não haja uma rede social conectada exibe a tela de alerta de que não há uma rede social conectada, informando ao usuário os passos para que seja feito a conexão.

Figura 25 – Tela de Redes Sociais: Publicações.



Fonte: Elaborado pelos autores

A tela de publicações apresentada na Figura 25 é composta por:

- **Campo de imagem:** Mostra a imagem publicada na rede social (Facebook, Twitter ou Instagram)
- **Campo de Descrição:** Exibe a descrição da imagem publicada na rede social (*Facebook*, *Twitter* ou *Instagram*)
- **Botão Publicar:** Abre a tela de publicação.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.13 Tela de Cadastro de Pedidos

A Figura 26 apresenta a tela de adicionar pedido no sistema, na qual são apresentados os campos de cliente, doce, quantidade, valor total dos gastos, preço de venda, margem de lucro, cep, endereço, bairro e número; como ícone

temos adicionar um novo doce e confirmar o cadastro do pedido; também há um calendário e dois botões de mais e menos.

Figura 26 – Tela de Cadastro de Pedidos.

Fonte: Elaborado pelos autores

A tela de adicionar pedido apresentada na Figura 26 é composta por:

- **Campo de doce:** Para a selecionar o doce cadastrado no sistema.
- **Campo de cliente:** Para a selecionar o nome do cliente.
- **Campo de quantidade:** Para digitação da quantidade.
- **Campo de valor total de gasto:** Para a digitação do total no valor gasto para a produção do doce.
- **Campo de preço de venda:** Para a digitação do preço a se cobrar do cliente.

- **Campo de margem de lucro:** Para a seleção da margem de lucro que se almeja ter com a venda.
- **Ícone de “adicionar um novo doce”:** Serve para acrescentar mais um doce ao pedido.
- **Ícone “confirmar o cadastro do pedido”:** Cadastra o pedido no sistema com base nas informações preenchidas.
- **Botão de mais:** Adiciona mais um na quantidade do pedido.
- **Botão de menos:** Retira um na quantidade do pedido.
- **Calendário:** Responsável por mostrar a data dos pedidos cadastrados.
- **Campo de CEP:** Para digitação do cep do pedido.
- **Campo de endereço:** Para digitação do endereço do pedido.
- **Campo de bairro:** Para a digitação do bairro do pedido.
- **Campo de número:** Para digitação de número do pedido.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.14 Tela de Cadastro de Clientes

A Figura 27 apresenta a tela de adicionar de clientes. A tela é apresentada em formato de formulário, contendo campos para foto, nome, e-mail e telefone. Também há um botão com um ícone de adicionar cliente, que é responsável adicionar o cliente no sistema. No canto superior esquerdo, há um ícone de menu o qual o usuário poderá acessar o menu do aplicativo.

Figura 27 – Tela de Cadastro de Clientes.

Fonte: Elaborado pelos autores

A tela de adicionar clientes apresentada na Figura 27 é composta por:

- **Campo de foto:** Campo para o usuário selecionar a foto do cliente.
- **Campo de nome:** Para a digitação do nome do cliente.
- **Campo de e-mail:** Para a digitação do e-mail do cliente.
- **Campo de telefone:** Para a digitação do telefone do cliente.
- **Botão com ícone adicionar cliente:** Adiciona o cliente no sistema, emite um alerta e direciona o usuário para a tela de listagem de clientes.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

3.6.15 Tela de Cadastro de Produtos

A Figura 28 apresenta a tela de Adicionar Produtos. A tela é apresentada em formato de formulário, contendo campos para nome do produto, descrição e um botão adicionar.

Figura 28 – Tela de Cadastro de Produtos.

Fonte: Elaborado pelos autores

A tela de adicionar produto apresentada na Figura 28 é composta por:

- **Campo nome do produto:** Neste campo o usuário digita o nome do produto que deseja adicionar.
- **Campo descrição:** Neste campo o usuário digita a descrição do produto que deseja adicionar.
- **Botão adicionar:** Este botão salva o produto no sistema.
- **Ícone de menu:** Utilizado para o acesso ao menu do aplicativo.

4 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo final, desenvolver um aplicativo para a plataforma *Android* que auxiliasse microempresários no controle sobre sua agenda de pedidos, o controle de custos e lucro e ajuda-los a divulgar de forma mais fácil e rápida nas redes sociais os seus produtos. Sendo estes requisitos atendidos pelo sistema aqui proposto, utilizando a *IDE Android Studio* para o desenvolvimento do aplicativo, juntamente com a linguagem de programação *Java* e o SGBD *SQLite*.

Ao longo dos capítulos deste trabalho foram apresentados conceitos necessário para o desenvolvimento do aplicativo, sendo eles: Sistema de Informação, Engenharia de *Software*, metodologia ágil, em especial a metodologia *Scrum,* utilizada de modo experimental para a criação do aplicativo, fora mostrado a arquitetura e organização de aplicação MVC usado no desenvolvimento e demostrou-se todos os diagramas utilizados para a criação do aplicativo desenvolvido e demonstrado no trabalho.

No desenvolvimento deste sistema houveram desafios interessantes, sendo um deles, a forma como iria ser implementado o calendário para os usuários poderem agendar os pedidos dos seus clientes, foram várias tentativas sem sucessos até chegar no atual resultado. Outro desafio foi a implementação das *APIs* de rede socias, onde, *Facebook* e *Instagram* se demonstraram muito exigentes com quem está apenas aprendendo a utilizar as *APIs,* pois se exige o envio de um relatório relatando sobre a funcionalidade do aplicativo e após o envio seria emitido um consentimento ou não de uso das *APIs* das empresas, fato este, que no *Twitter* não houve a necessidade. Fora estes dois grandes desafios, houve outros menores, mas que foram facilmente sanados por serem apenas problemas de adaptação, valendo citar o MVC, que a equipe sentiu uma certa resistência em implementar.

Vale ressaltar que a equipe de desenvolvimento se mostrou receptivo ao uso da metodologia *Scrum,* pois as reuniões diárias, existentes na metodologia, sanaram dúvidas e ajudou a equipe a procurar soluções mais facilmente para problemas encontrados no meio do desenvolvimento.

Para futuras implementações e melhorias foram levantas algumas ideias e ouvidas algumas ideias. Sendo elas:

- Expandir o aplicativo para outras plataformas, *IOS* e *web*;
- A criação de um aplicativo para os clientes poderem realizar pedidos, não havendo só a parte de microempreendedores;
- Implementar um sistema de localização de microempreendedores de doces perto dos usuários, um sistema similar ao do aplicativo *iFood*.
- Melhorias no Cadastro de Usuário.
- Melhorias no ajuste das imagens.
- Melhorias de na publicação nas Redes sociais.

REFERÊNCIAS

ADITYA, Sunny Kummar; KARN, Vikash Kummar. **Android SQLite Essentials**. Packt Publishing, 2014. Disponível em: <https://www.safaribooksonline.com/library/view/android-sqlite-essentials/9781783282951/ch01.html> Acesso em: 8 jun. 2018.

ASTAH. **About Astah**. Astah, Inc. Disponível em: <http://astah.net/about-us> Acesso em: 10 jun. 2018.

_____. **Astah Community**. Astah, Inc. Disponível em: <http://astah.net/editions/community> Acesso em: 10 jun. 2018.

_____. **Astah Professional**. Astah, Inc. Disponível em: <http://astah.net/editions/professional> Acesso em: 10 jun. 2018.

ALMEIDA, Maurício Barcellos; **Uma introdução ao XML, sua utilização na Internet e alguns conceitos complementares**. Disponível em: <http://www.scielo.br/pdf/ci/v31n2/12903>. p. 7. Acesso em: 06 jun. 2018, 17h09min.

ANSOFF, H. I., A nova estratégia empresarial. São Paulo: Atlas, 1990. p.58.

CATHO. **Programador**. Disponível em: <https://www.catho.com.br/vagas/?q=programador&regiao_id%5B0%5D=16&pais_id=31&faixa_sal_id_combinar=1&perfil_id=1&order=score&where_search=1&how_search=2> Acesso em: 22/05/2018 às 10h38m.

CROSS, James. **Learning Java 9 – Object Oriented Programming**. Safari Books, 2018. Disponível em: < https://www.safaribooksonline.com/library/view/learning-java-9/9781788623933/video3_1.html>. Acesso em: 06 jun. 2018.

DEITEL, Paul; DEITEL, Harvey. **Java** : Como programar.Trad. Edson Furmankiewicz. 8ª ed. São Paulo : Pearson Prentice Hall, 2010. p. 2-3 e 6-7.

Eclipse Foundation. **Eclipse for Android Developers.** Disponível em: <https://www.eclipse.org/downloads/packages/eclipse-android-developers/neonm6>. Acesso em: 04 jun. 2018, 23h03min.

FACEBOOK. **Como compartilhar no Android**. Facebook. Disponível em: <https://developers.facebook.com/docs/sharing/android > Acesso em: 14 nov. 2018.

GOOGLE DEVELOPERS. **Conheça o Android Studio**. Disponível em: <https://developer.android.com/studio/intro/?hl=pt-br>. Acesso em: 05 jun. 2018, às 22h28min.

GOOGLE DICIONÁRIO. **Ágil**. Disponível em: <https://www.google.com.br/search?q=Dicion%C3%A1rio#dobs=Agil> Acesso em: 09 jun. 2018.

_____.**Metodologia**. Disponível em: <https://www.google.com.br/search?q=Dicion%C3%A1rio#dobs=metodologia> Acesso em 09 jun. 2018.

GRADY, Jeffrey O.. **System Requirements Analysis**. Elsevier, 2013. 2ª ed. Disponível em: <https://www.safaribooksonline.com/library/view/system-requirements-analysis/9780124171077/xhtml/CHP002.html#ST0015> Acesso em 17 nov. 2018.

GRAVES, Mark; **Projeto de Banco de dados com XML**. Trad. Aldair José Coelho Corrêa da Silva. São Paulo : Pearson Education, 2003. p. 2.

GUSMÃO; Gustavo, **Google lança versão 1.0 do IDE de código aberto Android Studio:** Ambiente de desenvolvimento integrado serve como substituto ao Eclipse e foi anunciado no Google I/O de 2013; atualizações seguirão padrão do Chrome : São Paulo : Editora Abril, 2014. Disponível em: <https://exame.abril.com.br/tecnologia/google-lanca-versao-1-0-do-ide-de-codigo-aberto-android-studio/>. Acesso em: 04 jun. 2018, 23h11min.

HENLEY, A.J.; WOLF, Dave. **Java EE Web Application Primer**: Building Bullhorn: A Messaging App with JSP, Servlets, JavaScript, Bootstrap and Oracle. Apress, 2017. Disponível em:

<https://www.safaribooksonline.com/library/view/java-ee-web/9781484231951/A458483_1_En_5_Chapter.html> Acesso em 16 nov. 2018.

INSTAGRAM. **Android Intents**. INSTAGRAM. Disponível em: <https://www.instagram.com/developer/mobile-sharing/android-intents/> Acesso em: 14 nov. 2018.

INSTITUTO ELDORADO. **Vagas e Oportunidades**. Disponível em: <http://www.eldorado.org.br/oportunidades/?term=42&orderby=name&order=ASC> Acesso em: 22/05/2018 às 10h38m.

KILIÇDAĞI, Arda; YILMAZ, H. İbrahim. **Laravel Design Patterns and Best Practices**. Packt Publishing, 2014. Disponível em: <https://www.safaribooksonline.com/library/view/laravel-design-patterns/9781783287987/ch01s02.html> Acesso em 16 nov. 2018.

KREIBICH, Jay A. **Using SQLite**. O'Reilly Media, Inc., 2010. Disponível em: <https://www.safaribooksonline.com/library/view/using-sqlite/9781449394592/ch01.html> Acesso em: 08 jun. 2018.

KRUCTHEN, Philippe; **Utilizando UML e padrões**: Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. Trad. Rosana T. Vaccare Braga. Porto Alegre : ARTMED EDITORA S.A. Disponível em: <https://books.google.com.br/books?hl=ptBR&lr=&id=hzl2tmT8QkUC&oi=fnd&pg=PR7&dq=UML&ots=miQQQuJFwo&sig=eln5DHUtiNTWJdS1agq80Of3OXY#v=onepage&q&f=true>. 2005. p. 39. Acesso em: 08 jun. 2018, 15h37min.

LAUDON, Kenneth; LAUDON, Jane. **Sistemas de Informação Gerenciais**. 9ª.ed . Trad. Luciana do Amaral Teixeira. São Paulo : Pearson Prentice Hall, 2010. p. 12

LINKEDIN. **Desenvolvedor**. Disponível em: <https://www.linkedin.com/jobs/search/?keywords=desenvolvedor&location=Bras%C3%ADlia%20e%20Regi%C3%A3o%2C%20Brasil> Ambos acessos em 22/05/2018 às 10h38m.

ORACLE. **Introduction to the Oracle Database**. Disponível em: < https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm> Acesso em: 8 jun. 2018.

PARADKAR, Sameer. **Mastering Non-Functional Requirements**. Packt Publishing, 2017. Disponível em: <https://www.safaribooksonline.com/library/view/mastering-non-functional-requirements/9781788299237/aed6d7c8-7b98-4d84-8520-f93ac12f6df2.xhtml> Acesso em 20 nov. 2018.

PEPPERS, Jonathan. **Xamarin Cross-platform Application Development**. Packt Publishing, 2015. 2ª ed. Disponível em: <https://www.safaribooksonline.com/library/view/xamarin-cross-platform-application/9781784397883/ch01.html> Acesso em 07 jun. 2018.

RAMOS, Allan. **O que é MVC?**. Tableless, 2015. Disponível em: <https://tableless.com.br/mvc-afinal-e-o-que/> Acesso em 16 nov. 2018.

ROBERTSON, James; ROBERTSON, Suzanne. **Mastering the Requirements Process**. Addison-Wesley Professional, 2006. 2ª Ed. Disponível em: <https://www.safaribooksonline.com/library/view/mastering-the-requirements/0321419499/ch07.html> Acesso em: 20 nov. 2018.

RUBIN, Kenneth S. **Essential Scrum**: A Practical Guide to the Most Popular Agile Process. Addison-Wesley Professional, 2012. Disponível em: <https://www.safaribooksonline.com/library/view/essential-scrum-a/9780321700407/ch01.html> Acesso em: 09 jun. 2018.

SAMOYLOV, Nick. **Introduction to Programming**. Packt Publishing, 2018. Disponível em: <https://www.safaribooksonline.com/library/view/introduction-to-programming/9781788839129/4738e05d-c2cb-4541-ad0f-cf5b61d2d62b.xhtml> Acesso em 16 nov. 2018.

SEBRAE. **Pesquisa MEI 2017**: Microempreendedor Individual. Disponível em: <http://www.sebrae.com.br/Sebrae/Portal%20Sebrae/UFs/SP/Pesquisas/SEBRAE-SP%20-%20MEI%202017%20-%20Relatorio%20Final-Imprensa.pdf> Acesso em 22/05/2018.

SILVA, José Carlos G.; ASSIS, Fidelis Sigmaringa G. de. **Linguagens de Programação** : Conceitos e Avaliação. McGraw-Hill, 1988. p. 4-5.

SOUZA, Tiago Daniel de; **Guia Prático de HTML – Parte 1:** Aprenda HTML através deste guia. O HTML é uma Linguagem de marcação de texto. Mais especificamente, uma linguagem de marcação de hipertexto. Linhadecódigo. Disponível em: <http://www.linhadecodigo.com.br/artigo/1184/guia-pratico-de-html-parte-1.aspx>. Acesso em: 06 jun. 2018, 16h45min.

SPENCE, Ian; BITTNER, Kurt. **Use Case Modeling**. Addison-Wesley Professional, 2002. Disponível em: <https://www.safaribooksonline.com/library/view/use-case-modeling/0201709139/pr03.html#pref02lev1sec1> Acesso em: 17 nov. 2018.

STELLMAN, Andrew; GREENE, Jennifer. **Learninig Agile**. O'Reilly Media, Inc., 2014. Disponível em: <https://www.safaribooksonline.com/library/view/learning-agile/9781449363819/ch01.html#what_is_agile_question> Acesso em: 09 jun. 2018.

TWITTER. **Sobre as APIs do Twitter**. Twitter, Inc. Disponível em: <https://help.twitter.com/pt/rules-and-policies/twitter-api> Acesso em: 14 nov. 2018.

VASIĆ, Miloš. **Mastering Android Development with Kotlin**. Packt Publishing, 2017. Disponível em: < https://www.safaribooksonline.com/library/view/mastering-android-development/9781788473699/76a90970-bb5b-42a5-bede-6c7d823c7f5a.xhtml> Acesso em: 07 jun. 2018.

VITANTONIO, Juan Ignacio; MAHLER, Martin. **Mastering Qlik Sense**. Packt Publishing, 2018. Disponível em: <https://www.safaribooksonline.com/library/view/mastering-qlik-sense/9781783554027/f3e41416-fa58-4286-aa8d-78c263c64e2f.xhtml> Acesso em 16 nov. 2018.

WOJDA, Igor; MOSKALA, Marcin. **Android Development with Kotlin**. Packt Publishing, 2017. Dispinível em:

<https://www.safaribooksonline.com/library/view/android-development-with/9781787123687/f932c352-a794-4a31-b7a5-f11935885692.xhtml > Acesso em: 07 jun. 2018.