UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

DANIEL YUDI UEDA

APLICAÇÃO ANDROID COM FIREBASE PARA ADMINISTRAR O USO DE MEDICAÇÕES

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2020

DANIEL YUDI UEDA

APLICAÇÃO ANDROID COM FIREBASE PARA ADMINISTRAR O USO DE MEDICAÇÕES

Trabalho de Conclusão de Curso, apresentado ao Curso de Especialização Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Profa. Andreia Scariot Beulke

PATO BRANCO
2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

APLICAÇÃO ANDROID COM FIREBASE PARA ADMINISTRAR O USO DE MEDICAÇÕES

por

DANIEL YUDI UEDA

A avaliação deste trabalho de conclusão de curso foi realizada em 09 de março de 2020, como requisito parcial para a obtenção do título de especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Andréia Scariot Beulke (orientadora), Beatriz Terezinha Borsoi e Vinicius Pegorini, membros de banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Andréia Scariot Beulke
Profa. Orientadora (UTFPR)

_____
Beatriz Terezinha Borsoi
(UTFPR)

_____
Vinicius Pegorini
(UTFPR)

_____
Vinicius Pegorini
Coordenador do curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

Os medicamentos são utilizados na prevenção, no diagnóstico, no tratamento de doenças e, também, no controle de sintomas, como dor, por exemplo. Esses medicamentos são adquiridos em farmácias com prescrição médica ou por automedicação. A ingestão de medicamentos deve ser realizada a cada período de tempo, seja em horas, diariamente ou semanalmente por exemplo, ou ainda, em dose única, dependendo da sua finalidade. A administração na dosagem e periodicidade indicada de cada medicamento é bastante importante para que o resultado esperado seja satisfatório e pode acontecer em diferentes lugares, como, em casa, no trabalho, ou mesmo quando a pessoa está em seu momento de lazer. A solução proposta por meio do desenvolvimento deste trabalho é para dispositivos móveis Android. O aplicativo possibilita que o usuário possa cadastrar e consultar os medicamentos que utiliza e que tenha ingerido em determinado dia e horário. Para o desenvolvimento do sistema foi utilizada a tecnologia Java e o banco de dados Firebase.

Palavras-chave: Java. Android. Firebase. Medicamentos.

ABSTRACT

Medicines are used in the prevention, diagnosis, treatment of diseases and also in the control of symptoms, such as pain, for example. These drugs are purchased at pharmacies with a prescription or self-medication. An intake of medications should be performed in specific time, such as in hours, daily or weekly for example, or even in a single dose, depending on its use. The administration of these medications is very important for the expected result and can be satisfactory and can occur in different places, such as at home, at work place or even when the person is at leisure. A solution proposed through the development of this work is for Android mobile devices. The developed application allows the user to register and consult the medications he uses and which has already been determined its use in specific times. For the development of the system, Java technology and Firebase database were used.

Keywords: Java. Android. Firebase. Medicamentos.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGO

LISTA DE SIGLAS

API	*Application Programming Interface*
BO	*Bussines Object*
GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
JVM	*Java* Virtual Machine
OMS	Organização Mundial de Saúde
SO	Sistema Operacional
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho que abrange as considerações iniciais, os seus objetivos e a justificativa. O capítulo é finalizado com a apresentação do texto por meio da descrição sumária dos capítulos que compõem o texto.

1.1 CONSIDERAÇÕES INICIAIS

O uso de medicamentos seja contínuo, temporário ou controlado é uma prática bastante comum entre as pessoas e vem crescendo a cada ano. Dados da Guia 2018 Interfarma (2020) apontam que de 2013 até 2017 houve um crescimento no varejo farmacêutico de 36 para 57 bilhões no Brasil, com 162 bilhões de doses comercializadas. Esses dados evidenciam que cada vez mais as pessoas estão adquirindo medicamentos seja por prescrição médica ou por automedicação. Esses medicamentos podem ser usados na prevenção, no diagnóstico, no tratamento de doenças e, também, no controle de sintomas, como dor e febre, por exemplo.

Uma recomendação clássica da Organização Mundial de Saúde (OMS) é que o uso adequado de medicamentos acontece quando os pacientes recebem os medicamentos apropriados à sua condição de saúde, em doses adequadas às suas necessidades individuais, por um período de tempo adequado (OMS, 1985).

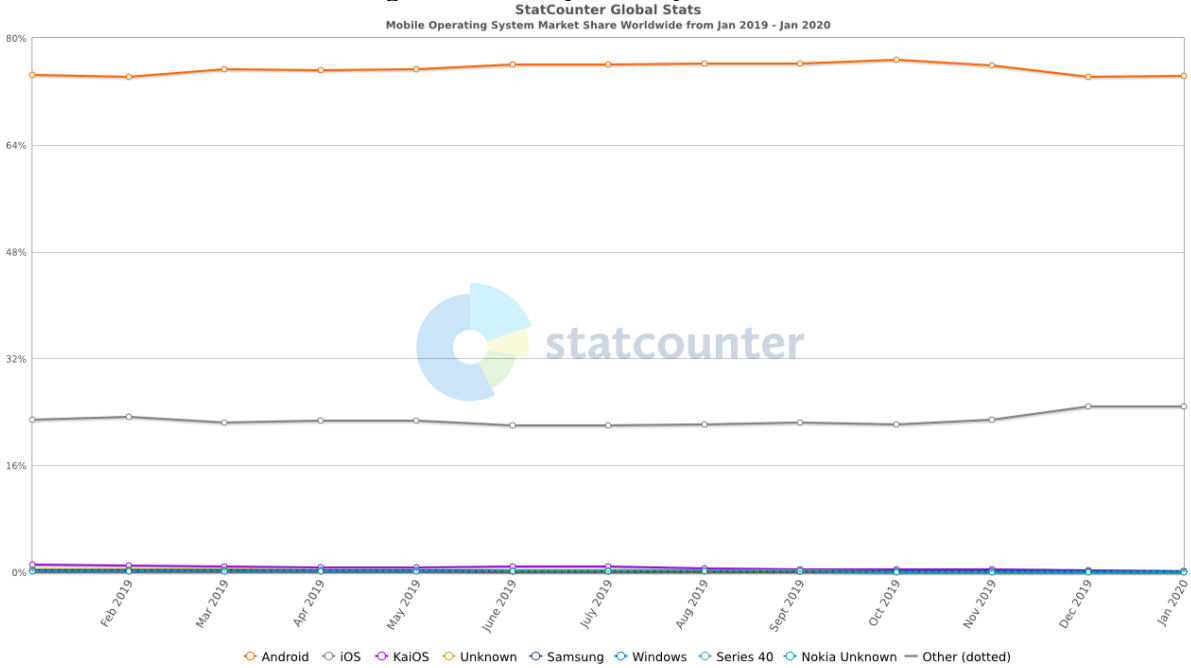
A ingestão de medicamentos deve ser realizada a cada período de tempo, seja em horas, diariamente ou semanalmente por exemplo, ou ainda, em dose única, dependendo da sua finalidade. A administração desses medicamentos é bastante importante para que o resultado esperado seja satisfatório e pode acontecer em diferentes lugares, como, em casa, no trabalho, ou mesmo quando a pessoa está em seu momento de lazer.

Para o uso correto dos medicamentos que necessita, é importante que o paciente seja bastante organizado adotando alguns procedimentos que o auxiliem, como, fazer uma lista de medicamentos, com o nome, a quantidade e o horário de cada um, ou fazer uma agenda, usando aplicativos como o Google Agenda, por exemplo. Contudo, isso pode não ser tão eficaz no controle de horários que os medicamentos devem ser ingeridos, pois o paciente pode esquecer a lista ou não verificar a agenda.

Nesse contexto, um aplicativo Android que permita que o paciente possa cadastrar os medicamentos que necessita, com dados como o dia, o horário e o período que o medicamento deve ser utilizado, pode auxiliar para que o paciente use adequadamente os medicamentos que necessita.

O sistema operacional Android é bastante utilizado em dispositivos móveis. De acordo com a pesquisa realizada pela StatCounter (2020), o sistema operacional Android está presente em 74,3% dos aparelhos do mundo, conforme mostra a Figura 1.

Figura 1 – Utilização de dispositivos móveis



Fonte: Statcounter (2020)

Devido ao alto percentual de dispositivos móveis Android esse sistema operacional foi escolhido para o desenvolvimento do aplicativo. Assim, por meio da realização deste trabalho foi desenvolvido um aplicativo para Android para auxiliar na administração adequada de medicamentos.

1.2 OBJETIVOS

O objetivo geral apresenta o resultado principal obtido da realização deste trabalho e os objetivos específicos o complementam.

1.2.1 Objetivo Geral

Desenvolver um aplicativo Android para auxiliar na administração adequada de medicamentos.

1.2.2 Objetivos Específicos

Os objetivos específicos do trabalho são:

- Possibilitar que o usuário possa cadastrar um tratamento medicamentoso.
- Permitir que o usuário possa consultar os tratamentos cadastrados por ele.

1.3 JUSTIFICATIVA

O uso de medicamentos deve ser o mais adequado possível respeitando os intervalos e a dosagem estabelecidos para a doença e o paciente. As diversas formas de tomar cada remédio podem causar certa confusão de horários e dosagem, principalmente quando o paciente toma vários remédios com intervalos de tempos diferentes. No entanto, seguir a posologia de cada medicamento é essencial para que o tratamento alcance os melhores resultados, além de evitar os riscos associados ao consumo excessivo das substâncias encontradas nos remédios. A posologia varia em função do paciente, da doença que está sendo tratada e do tipo de medicamento utilizado. Respeitar a posologia é fundamental para a eficácia do tratamento e evitar complicações devido à superdosagem.

Um aplicativo Android que auxilie o usuário na administração adequada do uso de seus medicamentos e que notifique o usuário do horário para o uso de cada medicamento pode trazer benefícios para o paciente, pois poderá ser mais fácil respeitar à posologia e os intervalos orientados pelo médico para que os remédios façam o efeito esperado. É comum que medicamentos precisem ser tomados em períodos determinados pelo profissional da saúde.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos, dos quais este é o primeiro e apresenta as considerações iniciais e contextualização do trabalho, incluindo os objetivos e a justificativa.

No Capítulo 2 estão os materiais e os métodos utilizados no desenvolvimento deste. Os materiais se referem às tecnologias e ferramentas utilizadas e o método contém a metodologia adotada para o desenvolvimento deste trabalho, visando alcançar os objetivos propostos. O Capítulo 3 apresenta o sistema desenvolvido, contendo a modelagem das mais diversas partes do sistema e fornece uma visão geral das principais funcionalidades, apresentando detalhes do código fonte e da implementação. Por fim, o Capítulo 4 conclui o trabalho, apresentando os resultados obtidos, as análises acerca dos resultados e considerações sobre as possibilidades de continuidade e melhorias do projeto em trabalhos futuros.

2 MATERIAIS E MÉTODO

Este capítulo está subdividido em duas seções, sendo uma para os materiais e outra para o método.

O Quadro 1 apresenta as tecnologias e as ferramentas utilizadas para o desenvolvimento do trabalho.

Quadro 1 - Ferramentas e tecnologias utilizadas no desenvolvido do aplicativo

Ferramenta	Versão	Link	Funcionalidade
Android Studio	3.3.2	https://developer.android.com/studio	*Integrated Development Environment* (IDE) para desenvolvimento do aplicativo.
Firebase		https://firebase.google.com/?hl=ptbr	Armazenamento de *login*.
Gradle	3.3.2	https://gradle.org/	Gerenciador de dependências.
Java	1.8	https://www.java.com/pt_BR/about/	Linguagem de programação.
Astah Community	8.2	http://astah.net/editions/community	Modelagem do sistema: desenvolvimento do diagrama de casos de uso.
SQLite	3.31.1	https://www.sqlite.org/index.html	Banco de dados.

Fonte: Autoria própria (2020)

A seguir são descritas as principais tecnologias que foram utilizadas no desenvolvimento do sistema.

2.1.1 ANDROID STUDIO

A história do Android iniciou em 2005, quando a Google adquiriu a empresa Android Inc. Em 2007 foi apresentado o Sistema Operacional (SO) Android (PRIMORAC; RUSSO, 2012). Esse sistema é definido como um conjunto de serviços de software, especialmente portados para dispositivos móveis (GUANA *et al*., 2012). O Android oferece uma *Application Programming Interface* (API) que provê a capacidade de acessar informações de hardware (como localização *Global Positioning System* (GPS), câmera, microfone e alto-falante), leitura do estado do telefone, leitura/escrita de dados do usuário, modificar configurações do telefone, entre outros.

A Figura 2 apresenta a estrutura do sistema operacional Android que é dividida em cinco camadas que possuem funcionalidades e comportamentos específicos: aplicações, *frameworks*, bibliotecas, *runtime* e kernel Linux.

Figura 2 – Estrutura do sistema operacional Android

Fonte: Traduzido de Android Developers (2016) e Android Architecture (2016).

2.1.2 FIREBASE

O Firebase, desenvolvido pela Google, é um conjunto de produtos distribuído gratuitamente, com um limite de utilização (FIREBASE, 2017). Entre esses produtos, existem serviços de hospedagem, armazenamento em nuvem e banco de dados. No caso do banco, a utilização gratuita permite até 100 acessos simultâneos. Essa ferramenta permite, com poucas linhas de código, adicionar o banco de dados em aplicações *web*, Android e iOS para que se conectem ao mesmo banco, sem, contudo, requerer conhecimentos sobre a infraestrutura do sistema.

2.1.3 JAVA

A linguagem Java foi implementada pela Sun Microsystems, em 1990, para ser executada nas mais diversas plataformas de hardware visando à Internet (FEDELI, 2003). É uma linguagem de programação orientada a objetos, multiplataforma, e roda em qualquer sistema operacional que possua um interpretador instalado, denominado de *Java Virtual Machine* (JVM), é um programa que converte o código Java em comandos que o sistema operacional possa executar. Ao compilar o código na linguagem Java, são gerados arquivos objetos chamados de *bytecodes* que podem ser executados por meio de interpretadores desenvolvidos para cada tipo de plataforma (FEDELI, 2003).

Com a linguagem Java é possível desenvolver sistemas para diversos ambientes, como *web*, *desktop* e dispositivos móveis. Assim, qualquer dispositivo que tenha a JVM instalada pode executar programas escritos em Java.

2.2 MÉTODO

O método consiste nas atividades de levantamento de requisitos, modelagem e implementação e testes. No desenvolvimento deste trabalho, essas atividades foram baseadas em um ciclo de planejamento iterativo de análise, prototipação e implementação de requisitos. A seguir é descrita sucintamente a realização dessas etapas.

Levantamento de requisitos: a ideia da implementação do aplicativo surgiu observando pessoas de idade com dificuldade de lembrar quais remédios tomavam e em qual período. O levantamento dos requisitos iniciou com a listagem dos requisitos considerados essenciais para a aplicação, enfatizando funcionalidades e aspectos de qualidade considerados

relevantes. A partir dessa listagem foram definidos os requisitos funcionais e a eles foram agregados requisitos complementares, descrições adicionais e requisitos não funcionais. Posteriormente desenvolveu-se um esboço inicial do padrão das telas para o sistema.

Análise e projeto: a modelagem dos requisitos identificados foi realizada gerando os diagramas para a representação das funcionalidades. Em seguida o diagrama do banco de dados com a definição das tabelas, seus campos e relacionamento, foi realizado. Os dados serão armazenados localmente, no banco de dados do dispositivo móvel e não há necessidade de controle de permissões de acesso ou de restrição a funcionalidades, pois haverá apenas um perfil de acesso.

Implementação: a implementação foi realizada utilizando o ambiente de desenvolvimento Android Studio com a linguagem Java e os bancos de dados Firebase e SQLite.

Testes: foram realizados os testes relacionados ao código. Esses testes foram informais e realizados pelo autor deste trabalho visando identificar erros de codificação, verificação e validação dos requisitos definidos para o aplicativo.

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho. O capítulo inicia com a descrição do escopo do sistema, seguido da modelagem, apresentação e implementação do sistema proposto.

3.1 ESCOPO DO SISTEMA

Utilizando o aplicativo desenvolvido como resultado desse trabalho, o usuário poderá administrar seus tratamentos realizados com o uso medicamentos de maneira mais eficiente do que anotar horários e dosagens em papéis, por exemplo.

Para a utilização do aplicativo, que é para Android, o usuário deverá realizar o seu cadastro informando os dados de e-mail e senha. Esses dados devem ser válidos e são registrados no Firebase. Após realizar a autenticação no aplicativo, o usuário poderá cadastrar os medicamentos que serão utilizados no seu tratamento, seja por um tempo determinado ou indeterminado. Esses dados correspondem ao nome do medicamento, a dosagem, os dias e horários que o medicamento deve ser ingerido e um campo para observações. Ao cadastrar um medicamento, será gerado um registro no banco de dados local que conterá todos os medicamentos cadastrados por ele naquele dispositivo.

Os registros gerados podem ser consultados para que o usuário possa administrar os dias e horários de uso da medicação de acordo com o período cadastrado.

3.2 MODELAGEM DO SISTEMA

Essa seção apresenta os requisitos funcionais e não funcionais, diagrama de caso de uso e o diagrama do banco de dados.

No Quadro 2 estão os requisitos funcionais (RF) definidos para o sistema.

Quadro 2 – Requisitos Funcionais

Identificação	Nome	Descrição
RF1	Criar login	O sistema só é acessado por meio de autenticação com os dados de e-mail e senha.
RF2	Manter medicamento	O usuário pode cadastrar os medicamentos utilizados em seus tratamentos médicos.

RF3	Manter tratamento	Após o cadastro do medicamento ser realizado é possível verificá-los para conferir se está de acordo com o receituário médico.

Fonte: Autoria própria (2020).

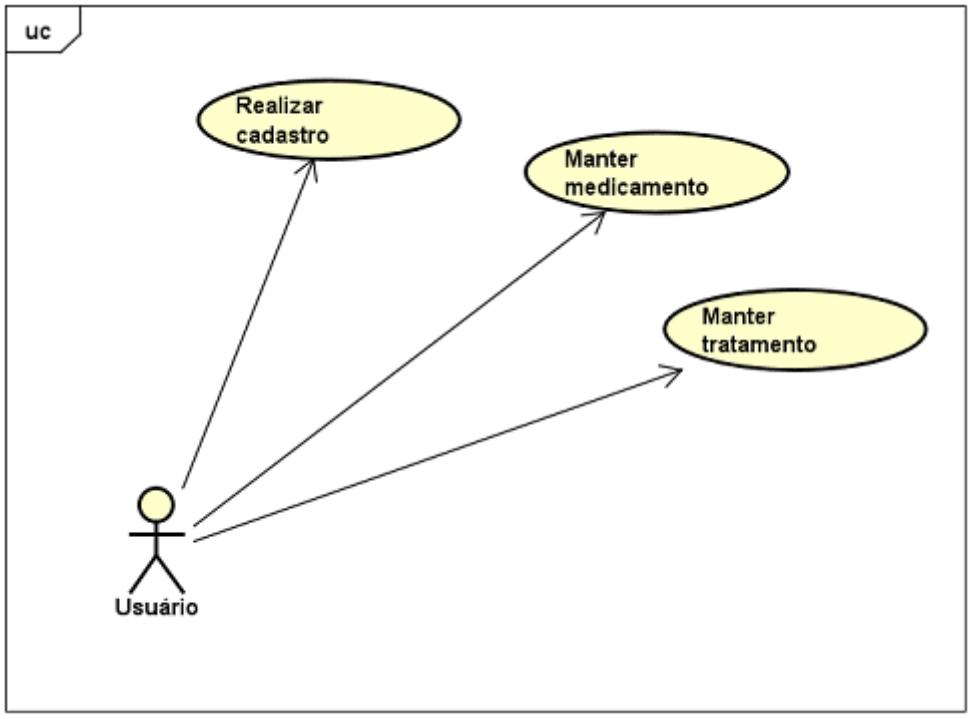
No Quadro 3 estão os requisitos não funcionais (RNF) do sistema e estão relacionados ao acesso ao sistema, validação de campos e de relatórios.

Quadro 3 – Requisitos não funcionais

Identificação	Nome	Descrição
RNF01	Efetuar *login*	O acesso ao sistema será realizado por meio de *login* e senha.
RNF02	Campos de preenchimento obrigatório	Os campos que são de preenchimento obrigatório são validados por meio de funções do sistema.

Fonte: Autoria própria (2020).

Figura 3 – Diagrama de caso de uso

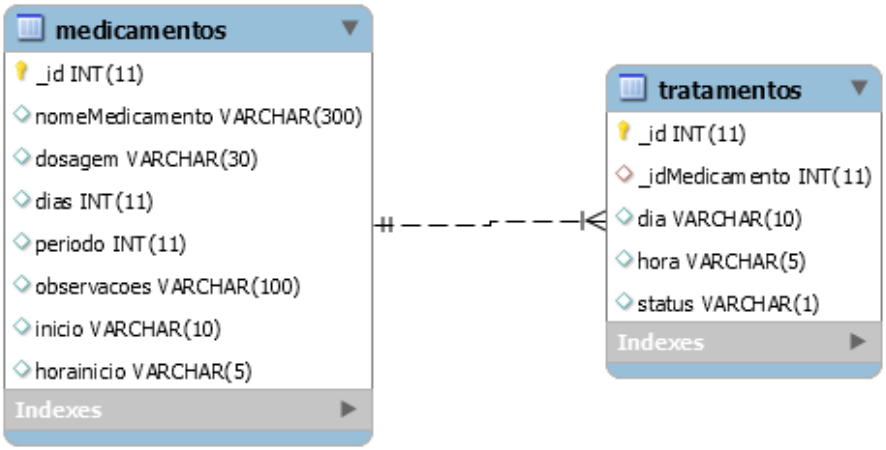


Fonte: Autoria própria (2020).

3.2.1 MODELAGEM DO BANCO DE DADOS

Na Figura 4 é exibido o diagrama entidade relacionamento do sistema, que contém a tabela que mantém o cadastro dos medicamentos localmente no dispositivo do usuário e a tabela que armazena o tratamento de acordo com as informações cadastradas com os tipos e tamanhos de cada atributo.

Figura 4 – Entidade Relacionamento



Fonte: Autoria própria (2020).

3.3 APRESENTAÇÃO DO SISTEMA

Essa seção apresenta a descrição das principais funcionalidades que poderão ser visualizadas por meio de *prints* das telas do sistema.

A Figura 5 exibe a tela de autenticação do usuário, sendo que o acesso é realizado com os dados de *e-mail* e senha. Caso o usuário não possua uma conta para o acesso ele deverá clicar no botão “registrar” para criar a sua conta de acesso. Caso o usuário informe os dados incorretos são exibidas mensagens de erro de validação ao usuário.

Figura 5 - Tela de acesso ao sistema

Fonte: Autoria própria.

A Figura 6 exibe o menu de opções do aplicativo, no qual a opção de cadastrar medicamento direciona para outra *Activity* e a opção “Medicamentos Cadastros” exibe um componente de alerta com os medicamentos cadastrados.

Figura 6 - Menu do sistema

Fonte: Autoria própria (2020).

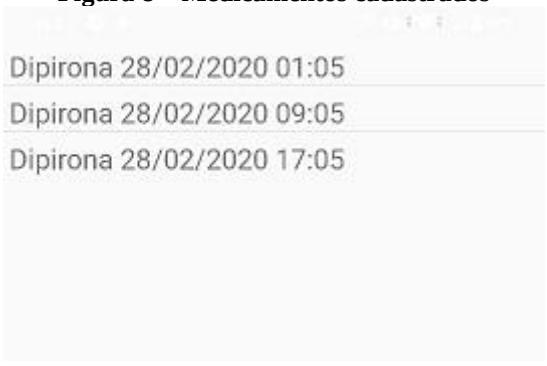
A Figura 7, exibe os campos para o preenchimento das informações do tratamento medicamentoso. O usuário deverá preencher esses campos de acordo com as informações da receita médica.

Figura 7 - Cadastro de medicamentos

Fonte: Autoria própria (2020).

A Figura 8 demonstra os medicamentos que o usuário deverá tomar no dia em questão, de acordo com o que foi cadastrado previamente. Por exemplo, o usuário cadastrou um determinado medicamento e que deve ser tomado a cada período de tempo (8 horas, por exemplo).

Figura 8 – Medicamentos cadastrados



Fonte: Autoria própria (2020).

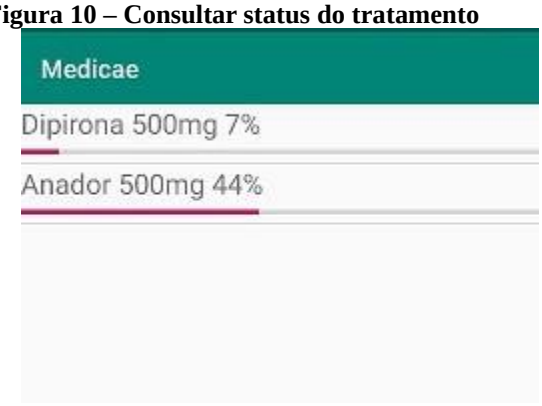
Para que o usuário tenha um controle mais efetivo da medicação que foi tomada no período indicado ele deverá tocar e segurar sobre o horário correspondente e o sistema atualiza o *status* daquele registro ficando na cor verde que indica que o usuário e já ingeriu determinado medicamento no horário marcado. Na Figura 9 é exibido como se comporta o registro após realizar o toque longo.

Figura 9 – Medicamentos cadastrados atualizados

Fonte: Autoria própria (2020).

O usuário pode consultar o *status* do seu tratamento, possuindo assim uma estimativa do curso do tratamento através do percentual exibido e de uma barra de progresso conforme na Figura 10.

Figura 10 – Consultar status do tratamento



Fonte: Autoria própria (2020).

3.4 IMPLEMENTAÇÃO DO SISTEMA

Para a codificação do sistema, foi realizada a instalação e a configuração do ambiente de desenvolvimento juntamente com os recursos utilizados (*frameworks* e *plugins*, por exemplo.)

Para a utilização do banco de dados SQLite foi criada uma classe que realizará todas as operações relacionadas ao banco de dados local. Essa classe estende a *SQLiteOpenHelper*, e é responsável por manipular as informações do SQLite, conforme exibido na Listagem 1.

Listagem 1 - Classe DatabaseHandler

```
public class DatabaseHandler extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "banco";
    private static final int VERSION = 8;

    public DatabaseHandler(Context c){
        super(c,DATABASE_NAME,null,VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL("CREATE TABLE IF NOT EXISTS medicamentos ( " +
                "_id               INTEGER       PRIMARY KEY AUTOINCREMENT,"
+
                "nomeMedicamento VARCHAR (300)," +
                "dosagem          VARCHAR (100)," +
                "dias             BIGINT," +
                "periodo          INTEGER," +
                "observacoes      VARCHAR (500)," +
                "inicio           TEXT,"+
                "horainicio       TEXT)"
        );

        db.execSQL("CREATE TABLE IF NOT EXISTS tratamentos ( " +
                "_id               INTEGER       PRIMARY KEY AUTOINCREMENT,"
+
                "_idMedicamento    INTEGER," +
                "dia               TEXT," +
                "hora              TEXT," +
                "status            VARCHAR(1))"
        );


    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        onCreate(db);
    }

    public void incluir(Medicamento medicamento){
        SQLiteDatabase banco = this.getWritableDatabase();

        ContentValues registro = new ContentValues();
        registro.put("nomeMedicamento", medicamento.getNomeMedicamento());
        registro.put("dosagem", medicamento.getDosagem());
        registro.put("dias", medicamento.getDias().longValue());
        registro.put("periodo", medicamento.getPeriodo());
        registro.put("observacoes", medicamento.getObservacoes());
        registro.put("inicio", medicamento.getInicio().toString());
        registro.put("horainicio", medicamento.getHoraInicio().toString());

        banco.insert("medicamentos", null, registro);
```

```
        Integer qtd = 24/medicamento.getPeriodo();
        LocalDate diaInicio = medicamento.getInicio();
        LocalTime horaInicio = medicamento.getHoraInicio();
        Integer id = ultimoIdMedicamento();

        for (int i = 0; i < medicamento.getDias().longValue(); i++) {
            for (int j = 0; j < qtd; j++) {
                Tratamento tratamento = new Tratamento();
                tratamento.setStatus("n");
                tratamento.set_idMedicamento(id);

tratamento.setDia(diaInicio.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"
)));
                tratamento.setHora(horaInicio);
                horaInicio =
horaInicio.plusHours(medicamento.getPeriodo());
                this.incluir(tratamento);
            }
            diaInicio = diaInicio.plusDays(1);
        }
    }
```

Fonte: Autoria própria (2020).

A Listagem 2 é contém o método de inserção do tratamento e os métodos que retornam os dados para manipulação em tela.

Listagem 2 - Classe DatabaseHandler

```
public void incluir(Tratamento tratamento){

        SQLiteDatabase banco = this.getWritableDatabase();
        ContentValues registro = new ContentValues();
        registro.put("_idMedicamento", tratamento.get_idMedicamento());
        registro.put("status", tratamento.getStatus());
        registro.put("dia", tratamento.getDia().toString());
        registro.put("hora", tratamento.getHora().toString());

        banco.insert("tratamentos", null, registro);

    }

    public Cursor listar(){

        SQLiteDatabase banco = this.getWritableDatabase();

        Cursor registros = banco.query("medicamentos", null,null,null,
                null,null,null,null);

        return registros;
    }

    public ArrayList<MedicamentoTratamento> getAllMedicamentoTratamentos()
{
        SQLiteDatabase db = this.getReadableDatabase();
```

```
        ArrayList<MedicamentoTratamento> listItems = new
ArrayList<MedicamentoTratamento>();
        Cursor cursor = db.rawQuery("SELECT t._id, t.dia, " +
                                    "t.hora, t.status,
m.nomeMedicamento," +
                                    " m.observacoes " +
                                    "from tratamentos t " +
                                    "inner join medicamentos m on m._id
= t._idMedicamento " +
                                    "where t.dia = strftime('%d/%m/%Y',
'now')",
                new String[] {});

        if (cursor.moveToFirst()) {
            do {
                MedicamentoTratamento medicamentoTratamento = new
MedicamentoTratamento();


medicamentoTratamento.setId(cursor.getInt(cursor.getColumnIndex("_id")));

medicamentoTratamento.setDia(cursor.getString(cursor.getColumnIndex("dia"))
);

medicamentoTratamento.setHora(cursor.getString(cursor.getColumnIndex("hora"
)));

medicamentoTratamento.setStatus(cursor.getString(cursor.getColumnIndex("sta
tus")));

medicamentoTratamento.setNomeMedicamento(cursor.getString(cursor.getColumnI
ndex("nomeMedicamento")));

medicamentoTratamento.setObservacoes(cursor.getString(cursor.getColumnIndex
("observacoes")));

                listItems.add(medicamentoTratamento);
            } while (cursor.moveToNext());
        }

        cursor.close();

        return listItems;
    }

    public Integer ultimoIdMedicamento(){
        Integer aux = 0;
        SQLiteDatabase banco = this.getReadableDatabase();

        Cursor dados = banco.rawQuery("SELECT MAX(_ID) ID FROM
MEDICAMENTOS",null);

        if (dados.getCount() != 0)
        {
            dados.moveToNext();
            aux = dados.getInt(dados.getColumnIndex("ID"));
        }

        return aux;
    }
```

```
    public void atualizaStatus(Integer id){
        SQLiteDatabase db = this.getReadableDatabase();

        db.execSQL("UPDATE tratamentos SET status = 's' where _id = " +
String.valueOf(id));
    }

public ArrayList<MedicamentoStatus> getAllMedicamentoTratamentosStatus() {
    SQLiteDatabase db = this.getReadableDatabase();
    ArrayList<MedicamentoStatus> listItems = new
ArrayList<MedicamentoStatus>();
    Cursor cursor = db.rawQuery("SELECT m.nomeMedicamento || ' ' ||
m.dosagem nomeMedicamento, count(t.status) total, (SELECT
count(t1.status)\n" +
                    "         from tratamentos t1\n" +
                    "         where t1.status <> 'n'\n" +
                    "         and t1._idMedicamento = t._idMedicamento)
parcial\n" +
                    "         from tratamentos t  \n" +
                    "         inner join medicamentos m on m._id =
t._idMedicamento\n" +
                    "group by m._id",
            new String[] {});

    if (cursor.moveToFirst()) {
        do {
            MedicamentoStatus medicamentoStatus = new MedicamentoStatus();

medicamentoStatus.setNomeMedicamento(cursor.getString(cursor.getColumnIndex
("nomeMedicamento")));

medicamentoStatus.setParcial(cursor.getInt(cursor.getColumnIndex("parcial")
));

medicamentoStatus.setTotal(cursor.getInt(cursor.getColumnIndex("total")));

            listItems.add(medicamentoStatus);
        } while (cursor.moveToNext());
    }

    cursor.close();

    return listItems;
}

}
```

Fonte: Autoria própria (2020).

A Listagem 3 exibe o *model* do medicamento, que contém os campos criados na tabela medicamentos do banco de dados, com os métodos de acesso às propriedades.

Listagem 3 – Classe Medicamento

```
public class Medicamento implements Serializable {

    private int _id;
    private String nomeMedicamento;
    private String dosagem;
    private BigInteger dias;
    private int periodo;
    private String observacoes;
```

```
    private LocalDateTime inicio;

    public int get_id() {
        return _id;
    }

    public void set_id(int Iid) {
        this._id = _id;
    }

    public String getNomeMedicamento() {
        return nomeMedicamento;
    }

    public void setNomeMedicamento(String nomeMedicamento) {
        this.nomeMedicamento = nomeMedicamento;
    }

    public String getDosagem() {
        return dosagem;
    }

    public void setDosagem(String dosagem) {
        this.dosagem = dosagem;
    }

    public BigInteger getDias() {
        return dias;
    }

    public void setDias(BigInteger dias) {
        this.dias = dias;
    }

    public int getPeriodo() {
        return periodo;
    }

    public void setPeriodo(int periodo) {
        this.periodo = periodo;
    }

    public String getObservacoes() {
        return observacoes;
    }

    public void setObservacoes(String observacoes) {
        this.observacoes = observacoes;
    }

    public LocalDateTime getInicio() {
        return inicio;
    }

    public void setInicio(LocalDateTime inicio) {
        this.inicio = inicio;
    }
}
```

Fonte: Autoria própria.

A Listagem 4 contém o método responsável por adicionar um medicamento novo na base de dados do usuário, criando um objeto Medicamento e vinculando as informações digitadas pelo usuário nesse objeto, sendo persistido ao final do processo.

Listagem 4 – Classe CadastrarActiviy

```
public class CadastrarActivity extends AppCompatActivity {

    private EditText etDosagem, etNome, etDias, etPeriodo, etObs;
    private TimePicker timepicker;
    private DatePicker datePicker;
    private DatabaseHandler banco;

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

        etNome = findViewById(R.id.etNome);
        etDosagem = findViewById(R.id.etDosagem);
        etDias = findViewById(R.id.etDias);
        etPeriodo = findViewById(R.id.etPeriodo);
        etObs = findViewById(R.id.etObs);
        datePicker = findViewById(R.id.datePicker);
        timepicker= findViewById(R.id.timePicker);
        timepicker.setIs24HourView(true);

    }

    @android.support.annotation.RequiresApi(api = Build.VERSION_CODES.O)
    public void btIncluirOnClick(View view) {

        Medicamento cadastro = new Medicamento();
        cadastro.setNomeMedicamento(etNome.getText().toString());
        cadastro.setDosagem(etDosagem.getText().toString());

cadastro.setDias(BigInteger.valueOf(Long.parseLong((etDias.getText().toStri
ng()))));

cadastro.setPeriodo(Integer.parseInt(etPeriodo.getText().toString()));
        cadastro.setObservacoes(etObs.getText().toString());

        int day = datePicker.getDayOfMonth();
        int month = datePicker.getMonth() + 1;
        int year = datePicker.getYear();
        cadastro.setInicio(LocalDate.of(year,month,day));

        int hour = timepicker.getHour();
        int min = timepicker.getMinute();
        cadastro.setHoraInicio(LocalTime.of(hour,min));

        banco = new DatabaseHandler(this);

        banco.incluir(cadastro);
        Toast.makeText(this,"Sucesso", Toast.LENGTH_LONG).show();
        onBackPressed();
    }
}
```

Fonte: Autoria própria

A Listagem 5 exibe o método responsável por exibir os medicamentos previamente cadastrados pelo usuário ao selecionar a opção Medicamentos Cadastrados no menu principal.

Listagem 5 – Exibir medicamentos cadastrados

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

    ListView listView = (ListView) findViewById(R.id.list);
    banco = new DatabaseHandler(this);
    listItems = banco.getAllMedicamentoTratamentos();

    adapter = new TratamentoArrayAdapter(this, listItems);

    listView.setAdapter(adapter);

    listView.setOnItemLongClickListener(clickListener);

}

public AdapterView.OnItemLongClickListener clickListener = new
AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
                                   int arg2, long arg3) {
        // Assigning the item position to our global variable
        // So we can access it within our AlertDialog below
        banco.atualizaStatus(listItems.get(arg2).getId());

        Toast.makeText(MedicamentoCadastradoActivity.this,"Sucesso",
Toast.LENGTH_LONG).show();
        return false;
    }
};
```

Fonte: Autoria própria

O método responsável por realizar as chamadas a partir das opções disponíveis no menu principal, pode ser visto na Listagem 6.

Listagem 6 – Menu

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.new_medicamento) {
        Intent i = new Intent(this, CadastrarActivity.class);
        startActivityForResult(i,1);
    } else if (id == R.id.cad_medicamento) {
        Intent i = new Intent(this, MedicamentoCadastradoActivity.class);
        startActivityForResult(i, 1);
    }

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
```

```
    return true;
}
```

Fonte: Autoria própria.

Para a realizar a autenticação do usuário, a plataforma *Firebase* disponibiliza com a IDE AndroidStudio a integração que gera os códigos necessários para a validação. Na Listagem 7 é apresentado um código assíncrono, responsável pelo *login* ou registro do usuário.

Listagem 7 – Login ou Registro

```
public class UserLoginTask extends AsyncTask<Void, Void, Boolean> {

    private final String mEmail;
    private final String mPassword;

    UserLoginTask(String email, String password) {
        mEmail = email;
        mPassword = password;
    }

    @Override
    protected Boolean doInBackground(Void... params) {

        try {
            // Simulate network access.
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            return false;
        }

        for (String credential : DUMMY_CREDENTIALS) {
            String[] pieces = credential.split(":");
            if (pieces[0].equals(mEmail)) {
                // Account exists, return true if the password matches.
                return pieces[1].equals(mPassword);
            }
        }

        // TODO: register the new account here.
        return true;
    }

    @Override
    protected void onPostExecute(final Boolean success) {
        mAuthTask = null;
        showProgress(false);

        if (success) {
            finish();
        } else {

mPasswordView.setError(getString(R.string.error_incorrect_password));
            mPasswordView.requestFocus();
        }
    }

    @Override
```

```
    protected void onCancelled() {
        mAuthTask = null;
        showProgress(false);
    }
}
```

Fonte: Autoria própria.

Outro método disponibilizado, no qual foram realizadas alterações com o propósito de auxiliar o usuário no entendimento dos erros, é o método que realiza a tentativa de *login* ou de registro do usuário, apresentado na Listagem 8. Caso o parâmetro seja informado como um novo usuário o sistema tentará cadastrá-lo na base de dados do Firebase, se houver sucesso, o usuário poderá realizar o *login*. Se o parâmetro não for de novo usuário o sistema tentará realizar o *login*, de acordo com as validações pré-existentes.

Listagem 8 - Tentativa de login ou registro

```
private void attemptLoginOrRegister( boolean isNewUser) {
    if (mAuthTask != null) {
        return;
    }

    // Reset errors.
    mEmailView.setError(null);
    mPasswordView.setError(null);

    // Store values at the time of the login attempt.
    String email = mEmailView.getText().toString();
    String password = mPasswordView.getText().toString();

    boolean cancel = false;
    View focusView = null;

    // Check for a valid password, if the user entered one.
    if (!TextUtils.isEmpty(password) && !isPasswordValid(password)) {
        mPasswordView.setError(getString(R.string.error_invalid_password));
        focusView = mPasswordView;
        cancel = true;
    }

    // Check for a valid email address.
    if (TextUtils.isEmpty(email)) {
        mEmailView.setError(getString(R.string.error_field_required));
        focusView = mEmailView;
        cancel = true;
    } else if (!isEmailValid(email)) {
        mEmailView.setError(getString(R.string.error_invalid_email));
        focusView = mEmailView;
        cancel = true;
    }

    if (cancel) {
            focusView.requestFocus();
    } else {
            showProgress(true);

        if (isNewUser) {
```

```
auth.createUserWithEmailAndPassword(email,password).addOnCompleteListener(L
oginActivity.this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    showProgress(false);
                    Toast.makeText(getApplicationContext(), "Usuário
cadastrado com sucesso. Agora você pode se autenticar com suas
credenciais", Toast.LENGTH_LONG).show();
                }
            });
        } else {


auth.signInWithEmailAndPassword(email,password).addOnCompleteListener(Login
Activity.this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    showProgress(false);
                    if (task.getResult().getUser() != null ) {
                        Intent i = new
Intent(LoginActivity.this,MainActivity.class);
                        startActivity(i);
                        finish();
                    } else {
                        Toast.makeText(LoginActivity.this, "Usuário não
cadastrado", Toast.LENGTH_LONG).show();
                    }
                }
            });

        }

    }
}
```

Fonte: Autoria própria.

Foi necessário criar uma classe que estende da *BaseAdapter* para realizar a listagem dos medicamentos cadastros, conforme a Listagem 9.

Listagem 9 – Adapter

```
public class TratamentoArrayAdapter extends BaseAdapter {

    private final Context context;
    private final ArrayList<MedicamentoTratamento> values;

    @Override
    public int getCount() {
        return values.size();
    }

    @Override
    public Object getItem(int position) {
        return values.get(position);
    }

    @Override
    public long getItemId(int position) {
```

```
        return 0;
    }

    public TratamentoArrayAdapter(Context context,
ArrayList<MedicamentoTratamento> values) {

        this.context = context;
        this.values = values;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View rowView = inflater.inflate(R.layout.rowlayout, parent, false);

        TextView textView = (TextView) rowView.findViewById(R.id.label);

        textView.setText(values.get(position).getNomeMedicamento() + " " +
                values.get(position).getDia() + " " +
                values.get(position).getHora());

        if (!values.get(position).getStatus().equals("n")) {
            textView.setBackgroundColor(Color.GREEN);
        }

        return rowView;
    }
}
```

Fonte: Autoria própria.

Para a manipulação das informações dos itens referentes ao tratamento, foi criada uma classe que armazena as informações, visível na Listagem 10.

Listagem 10 – Classe Tratamento.

```
public class Tratamento implements Serializable {

    private int _id;
    private int _idMedicamento;
    private String dia;
    private LocalTime hora;
    private String status;

    public int get_id() {
        return _id;
    }

    public void set_id(int _id) {
        this._id = _id;
    }

    public int get_idMedicamento() {
        return _idMedicamento;
    }

    public void set_idMedicamento(int _idMedicamento) {
        this._idMedicamento = _idMedicamento;
    }
```

```
    public String getDia() {
        return dia;
    }

    public void setDia(String dia) {
        this.dia = dia;
    }

    public LocalTime getHora() {
        return hora;
    }

    public void setHora(LocalTime hora) {
        this.hora = hora;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
```

Fonte: Autoria própria.

Para a correta utilização do *Adapter* e exibição das informações, uma classe para auxílio foi utilizado o padrão*Bussines Object* (BO) cujo código é apresentado na Listagem 11.

Listagem 11 – BO MedicamentoTratamento

```
public class Tratamento implements Serializable {

    private int _id;
    private int _idMedicamento;
    private String dia;
    private LocalTime hora;
    private String status;

    public int get_id() {
        return _id;
    }

    public void set_id(int _id) {
        this._id = _id;
    }

    public int get_idMedicamento() {
        return _idMedicamento;
    }

    public void set_idMedicamento(int _idMedicamento) {
        this._idMedicamento = _idMedicamento;
    }

    public String getDia() {
        return dia;
    }

    public void setDia(String dia) {
```

```
        this.dia = dia;
    }

    public LocalTime getHora() {
        return hora;
    }

    public void setHora(LocalTime hora) {
        this.hora = hora;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
```

Fonte: Autoria própria.

Para a exibição do *status* foi necessário criar um objeto com as informações que seriam exibidas, demonstrado na Listagem 12.

Listagem 12 - Objeto MedicamentoStatus

```
public class MedicamentoStatus implements Serializable {

    private String nomeMedicamento;
    private Integer total;
    private Integer parcial;

    public String getNomeMedicamento() {
        return nomeMedicamento;
    }

    public void setNomeMedicamento(String nomeMedicamento) {
        this.nomeMedicamento = nomeMedicamento;
    }

    public Integer getTotal() {
        return total;
    }

    public void setTotal(Integer total) {
        this.total = total;
    }

    public Integer getParcial() {
        return parcial;
    }

    public void setParcial(Integer parcial) {
        this.parcial = parcial;
    }
}
```

Fonte: Autoria própria (2020).

Foi necessária também a criação de um *adapter* para a listagem do progresso do tratamento, conforme a Listagem 13.

Listagem 13 - Adapter StatusTratamento

```
public class StatusTratamentoAdapter extends BaseAdapter {
    private final Context context;
    private final ArrayList<MedicamentoStatus> values;

    @Override
    public int getCount() {
        return values.size();
    }

    @Override
    public Object getItem(int position) {
        return values.get(position);
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    public StatusTratamentoAdapter(Context context,
ArrayList<MedicamentoStatus> values) {

        this.context = context;
        this.values = values;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View rowView = inflater.inflate(R.layout.tratamentorow, parent,
false);

        TextView textView = (TextView)
rowView.findViewById(R.id.tratamento);
        ProgressBar progressBar = (ProgressBar)
rowView.findViewById(R.id.tratamentoProgressBar);

        Integer vlr =
Math.round((Float.valueOf(values.get(position).getParcial()) /
Float.valueOf(values.get(position).getTotal())) * 100);

        textView.setText(values.get(position).getNomeMedicamento() + " " +
vlr + "%");

        progressBar.setMax(100);
        progressBar.setProgress(vlr);

        return rowView;
    }
}
```

Fonte: Autoria própria (2020).

A tela responsável pela exibição do progresso dos tratamentos cadastrados possui o código disponibilizado na Listagem 14.

Listagem 14 - Consultar Progresso

```
public class ConsultarTratamentoActivity extends AppCompatActivity {

    DatabaseHandler banco;
    ArrayList<MedicamentoStatus> listItems;
    StatusTratamentoAdapter adapter;

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

        ListView listView = (ListView) findViewById(R.id.listTratamento);

        banco = new DatabaseHandler(this);
        listItems = banco.getAllMedicamentoTratamentosStatus();

        adapter = new StatusTratamentoAdapter(this, listItems);

        listView.setAdapter(adapter);

    }
}
```

4 CONCLUSÃO

Este trabalho teve como objetivo desenvolver um aplicativo Android que possibilita ao usuário cadastrar e consultar os medicamentos utilizados em seu tratamento. Foram utilizados os componentes nativos da plataforma Android em conjunto com a plataforma Firebase para o armazenamento dos dados de acesso. Além da utilização do banco de dados SQLite para a persistência local dos dados inseridos pelo usuário.

Com a realização deste trabalho foi possível colocar em prática e aprimorar os conhecimentos adquiridos durante o período do curso e notar o amplo mercado disponível para a exploração comercial da plataforma Android.

O auxílio da ampla documentação disponível da plataforma e dos *framerworks* utilizados nesse trabalho, tiveram foram importantes para a conclusão do projeto.

Como trabalho futuro é possível realizar melhorias visuais, acrescentar funcionalidades não definidas neste projeto e buscar uma forma de atender ao requisito de notificar o usuário de ingerir a medicação, sem a utilização de um servidor para o envio das notificações.

REFERÊNCIAS

ANDROID ARCHITRECTURE. Disponível em: <http://www.tutorialspoint.com/android/android_architecture.htm>. Acesso em: 25 fev. 2020.

ANDROID DEVELOPERS. Disponível em: <http://developer.android.com>. Acesso em: 25 fev. 2020.

ASSOCIAÇÃO DA INDÚSTRIA FARMACÊUTICA DE PESQUISA. **Guia 2018 Interfarma.** Disponível em < https://www.interfarma.org.br/guia/guia-2018/dados_do_setor#varejo_brasileiro>. Acesso em: 24 fev. 2020.

FEDELI, Ricardo Daniel; POLLONI, Eurico Giulio Franco; PERES, Fernando Eduardo. **Introdução à ciência da computação**. São Paulo: Pioneira Thomson Learning, 2003.

FIREBASE. Firebase. 2017. **Documentação do Firebase**. Disponível em: < https://firebase.google.com/docs/ >. Acesso em: 25 fev. 2020.

GUANA, Victor; ROCHA, Fabio; HINDLE, Abram; STROUL, Eleni. **Do the stars align? Multidimensional analysis of Android's layered architecture**. MSR 2012, p. 124-127.

ORGANIZAÇÃO MUNDIAL DE SAÚDE. Conferência Mundial sobre Uso Racional de Medicamentos. Nairobi, 1985.

PRIMORAC, Sanja; RUSSO, Mladen. **Android application for sending SMS messages with speech recognition interface**. In: 5th International Convention MIPRO, 2012, p. 1763-1767.

STATCOUNTER. Disponível em: < https://gs.statcounter.com/os-market-share/mobile/worldwide>. Acesso em: 18 fev. 2020.