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

MARCELO DE SOUZA MARTINS

DESENVOLVIMENTO DE APLICATIVO PARA DISPOSITIVO MÓVEL (ME LEMBRE)

AMERICANA

2018

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

MARCELO DE SOUZA MARTINS

DESENVOLVIMENTO DE APLICATIVO PARA DISPOSITIVO MÓVEL (ME LEMBRE)

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Americana como parte dos requisitos para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

AMERICANA
2018

M344d MARTINS, Marcelo de Souza

Desenvolvimento de aplicativo para dispositivo móvel (Me Lembre). / Marcelo de Souza Martins. – Americana, 2018.

77f.

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

Orientador: Profa. Ms. Clerivaldo José Roccia

1 Dispositivos móveis – aplicativos 2. Android – aplicativos 3. Banco de dados. I. ROCCIA, Clerivaldo José II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.519

Faculdade de Tecnologia de Americana

Marcelo de Souza Martins

Desenvolvimento de aplicativo para dispositivo móvel (Me Lembre)

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e desenvolvimento de sistemas pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana.
Área de concentração: Computação.

Americana, 04 de dezembro de defesa da banca de 2018.

Banca Examinadora:

Clerivaldo José Roccia
Mestre
Faculdade de tecnologia de Americana

Evandro Santaclara
Especialista
Faculdade de tecnologia de Americana

Wladimir costa
Mestre
Faculdade de tecnologia de Americana

RESUMO

A humanidade sempre precisou de métodos que auxiliassem quanto à lembrança de tarefas cotidianas, isto é um fato, a partir disso, algumas pessoas procuraram solucionar estes problemas, através da escrita ou desenhos. Na sociedade moderna, com a evolução e popularização dos dispositivos móveis o celular tornou-se um grande foco no quesito de desenvolvimento de aplicações devido a sua mobilidade e utilidade no dia a dia. Tendo isso em vista, muitos desenvolvedores já pensaram em aplicações que auxiliassem nos lembretes diários, porém, a utilização da maioria desses aplicativos é de difícil utilização e acaba não agradando os usuários devido ao tempo que levaria a aprender o funcionamento da aplicação. Levando esses fatos em consideração, o objetivo deste trabalho foi desenvolver um aplicativo de lembretes de fácil aprendizado e entendimento, para que o usuário não gaste muito de seu tempo aprendendo como utilizar a aplicação. Durante os capítulos foram apresentados os processos de desenvolvimento do aplicativo, detalhando e documentando todos os passos de desenvolvimento, explicando também ao leitor todas as técnicas empregadas para a elaboração e desenvolvimento do trabalho, para que ele sinta maior engajamento com o projeto desenvolvido.

Palavras Chave: Dispositivos móveis – aplicativos; Android – aplicativos; Banco de dados.

Abstract

Mankind has always needed methods that help with the recollection of everyday tasks, that is a fact, then, some people have sought to solve these problems through writing or drawings. In modern society, with the evolution and popularization of mobile devices the cell phone has become a major focus in the development of applications due to its mobility and utility in everyday life. With this in mind, many developers have thought of applications that would aid in daily reminders, but most of these applications are difficult to use and end up appealing to users because of the time it would take to learn how the application works. Considering this, the aim of this paper was to develop a reminder application that is easy to learn and understand, so that the user does not spend much of his time learning how to use the application. During the chapters, the application development processes were presented, detailing and documenting all the development steps, explaining to the reader all the techniques used for the elaboration and development of the work, so that it feels more engaged with the developed project.

Keywords: Mobile – applications; Applications; Android – applications; Database.

Lista de Figuras

Lista de tabelas

SUMÁRIO

1. Introdução

1.1 Contextualização

Com a frenesi do mundo atual, pessoas possuem um desempenho cada vez mais comprometido de certas capacidades cognitivas, que são de suma importância durante a vida.

A capacidade cognitiva abordada como plano de fundo deste trabalho para posteriormente gerar a criação de um aplicativo, é a memória. A memória, como mãe de muitas outras capacidades, realiza um processo ótimo, que nomeamos como esquecimento, porém, este mesmo processo às vezes pode ser um tanto quanto prejudicial, o que é algo inevitável na vida. Tendo isso em vista, muitas tecnologias são desenvolvidas buscando auxiliar no não esquecimento de tarefas diárias, contudo, são poucos que conseguem, de forma eficiente, sanar as necessidades de pessoas que sofrem com o esquecimento, tanto por conta de tarefas excessivas, quanto por conta de dificuldades de recordação por influências externas, como por exemplo, algum evento anormal no cotidiano que provoque distrações, desviando a atenção tarefas a serem realizadas.

1.2 Problemática

A humanidade sempre precisou de métodos que auxiliassem quanto à lembrança de tarefas cotidianas, isto é um fato, a partir disso, algumas pessoas procuraram solucionar estes problemas, desde pouco tempo atrás, através de agendas físicas, isto é, anotava-se o que era necessário realizar durante o dia, semana ou mês e, pelo mesmo, fazia-se o controle das tarefas, sendo possível visualizar quais já finalizou ou não.

Neste método existem inúmeros pontos negativos, como a danificação das folhas por acidente, ou até mesmo o fato de se esquecer a própria agenda em algum lugar. Atualmente existem agendas que são à prova d'água, mas, ainda assim, não são totalmente práticas.

A evolução da tecnologia colaborou muito para resolver os problemas que a agenda física possui através da criação de dispositivos de agendas virtuais, no

começo era bem simples, o usuário fazia uma anotação e pronto, sempre que desejasse consultar as tarefas que precisava realizar, tirava o dispositivo do bolso e instantaneamente já tinha uma visão ampla do que precisava ser feito, conseguia ver tarefas do dia inteiro, em segundos. Infelizmente esta versão inicial era bem escassa de recursos, sendo insuficiente em seu uso, era simples e eficiente dentro de suas limitações, mas era necessário mais implementações para torná-la precisa e perfeita.

Com o passar dos anos, houve o surgimento dos smartphones e aplicativos, porém, assim como toda tecnologia inicial, os recursos são escassos, havia poucos desenvolvedores e poucas pessoas interessadas em fazer uso do mesmo. Este cenário atualmente é totalmente o oposto do inicial, hoje as pessoas não vivem sem seu smartphone, consequentemente, também não vivem sem fazer uso de aplicativos, incluindo a agenda.

Muitas empresas enxergaram a necessidade da agenda virtual em forma de aplicativo e muitas delas desenvolvem agendas boas, que atendem boa parte de seu público, porém, quando o assunto é recursos, alguns destes aplicativos acabam desenvolvendo muito mais que o necessário, enquanto outros são muito simples, isto é, não há um aplicativo com um ponto de equilíbrio harmonioso, onde a pessoa possa aprender de forma simples e eficaz o funcionamento do aplicativo, exigindo assim um tempo considerável para sua aprendizagem. Os aplicativos já desenvolvidos normalmente possuem uma interface complexa, alguns até possuem recursos interessantes como, por exemplo, mudar a cor do texto, da folha, inserir imagens e entre outros, porém acabam escapando muito da simplicidade, já que o objetivo é auxiliar a pessoa a lembrar de tarefas cotidianas e não acrescentar mais tarefas, como a de aprender de uma forma mais profunda a utilização de uma aplicação.

1.3 Hipótese

De acordo com o problema apresentado, a solução que se encaixa nos parâmetros requisitados é o desenvolvimento de uma aplicação para dispositivos móveis que possui o intuito de auxiliar os usuários a recordar de suas tarefas diárias, com um layout simples e com recursos fundamentais, solucionando o problema com eficiência e eficácia.

1.4 Motivação

Observando o meio acadêmico, conclui-se que a maioria das pessoas sofre em lembrar-se de tarefas cotidianas, muitas vezes devido à grande quantidade de atividades a serem realizadas, sejam elas complexas ou simples. Partindo deste pressuposto, ampliando o campo de análise para fora do meio acadêmico pode-se observar que mesmo se tratando de um público mais generalizado, ainda possui-se a mesma dificuldade quanto ao esquecimento das tarefas cotidianas, motivando assim o desenvolvimento de um aplicativo com objetivo de auxiliar as pessoas quanto a este problema.

1.5 Justificativa

Com o aplicativo móvel a ser desenvolvido, será possível auxiliar uma grande porcentagem do público que busca uma aplicação para ajudá-los quanto a sua organização pessoal, sem que os mesmos gastem muito tempo de seu dia com a aprendizagem de como utilizar corretamente a aplicação ou até mesmo de entendê-la, tornando-a eficiente e eficaz para seu propósito.

1.6 Objetivos

1.6.1 Objetivos Gerais

Desenvolver um para aplicativo móvel de auxílio para recordação de tarefas, assim evitando o esquecimento das mesmas.

1.6.2 Objetivos Específicos

- Desenvolvimento de uma parcela dos diagramas que compõem a UML
- Desenvolvimento do banco de dados SQLite e diagramas
- Desenvolvimento de aplicativo para smartphones utilizando Android versão 6.0
- Realização de testes de funcionalidade

2. Desenvolvimento teórico

Este capítulo tem como intuito demonstrar e dar uma noção básica para o leitor de todas as tecnologias que serão utilizadas posteriormente para o desenvolvimento do aplicativo para a plataforma Android versão 5.0.

2.1 O que é UML

A UML ou Unified Modeling Language (Linguagem de Modelagem Unificada) é uma linguagem adotada internacionalmente pela indústria de engenharia de software, sendo ela uma linguagem visual que define uma série de artefatos que nos ajuda a modelar e documentar os sistemas orientados a objetos que desenvolvemos, entretanto, vale ressaltar que esta linguagem é totalmente independente, podendo ser utilizada em diferentes processos de desenvolvimento, embora o número de diagramas possui o foco em desenvolvimento de softwares seja grande (GUEDES, 2009, p. 19; RIBEIRO, 2012).

Na área de desenvolvimento de software, a UML, explicando de um modo geral, tem a função de descrever de forma gráfica como, quando e o que o software irá realizar, a partir disso, transformamos esta forma gráfica em códigos (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 14).

2.1.1 Breve história da UML

O surgimento da UML não foi de forma repentina, com o aparecimento das linguagens orientada a objetos entre os anos de 1970 e 1980, o pessoal envolvido com a metodologia da construção de software começaram a desenvolver novas metodologias de análise e desenvolvimento de projeto para atender a demanda das novas linguagens, surgiram de 10 à 50 novas metodologias no período de 1989 à 1994 (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 2).

Durante o surgimento dessas novas metodologias, dentro das seis que mais se destacaram, três delas se sobressaíram, sendo elas, o método Booch, OMT de Jacobson e OOSE de Rumbaugh. Cada uma dessas três eram ótimas em certas partes do desenvolvimento, porém, não eram perfeitas, sendo assim os três criadores dos modelos supracitados começaram a unificar o sistema para fornecer ao mercado uma metodologia de construção de software com programação

orientado a objetos simples e eficaz (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 2 - 3).

Houve versões 0.8 e 0.9 da UML, sendo a primeira feita em 1994, e a segunda em 1996. Após demonstrarem a UML 0.9 para o mercado e aprovada pelos engenheiros de software, muitas empresas de desenvolvimento de software como IBM, Microsoft, Hewlett-Packard e entre outras, resolveram colaborar na elaboração de um sistema ainda melhor e mais sofisticado que atendesse a demanda de todos, assim então surge a UML 1.0, em 1997 (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 3 – 4).

A UML foi oferecida para a OMG (Object management group) para se tornar a linguagem de modelagem padrão, proposta que posteriormente foi aceita (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 3 – 4).

2.2 Por que modelar um software?

A documentação e modelação de software são de suma importância, mesmo que alguns questionem sua necessidade. Um grande questionamento é, por que fazer a documentação se eu sei o que quero no software? É neste ponto onde muitos pecam, um desenvolvedor até pode saber o que o software deve fazer, entretanto, os fatores tempo e dinamismo de software não permitem que esse argumento seja válido. Primeiramente porque, conforme o tempo passa, esse desenvolvedor provavelmente vai esquecer-se do que havia feito no software e qual foi seu raciocínio em certo momento, segundo a citação de Guedes:

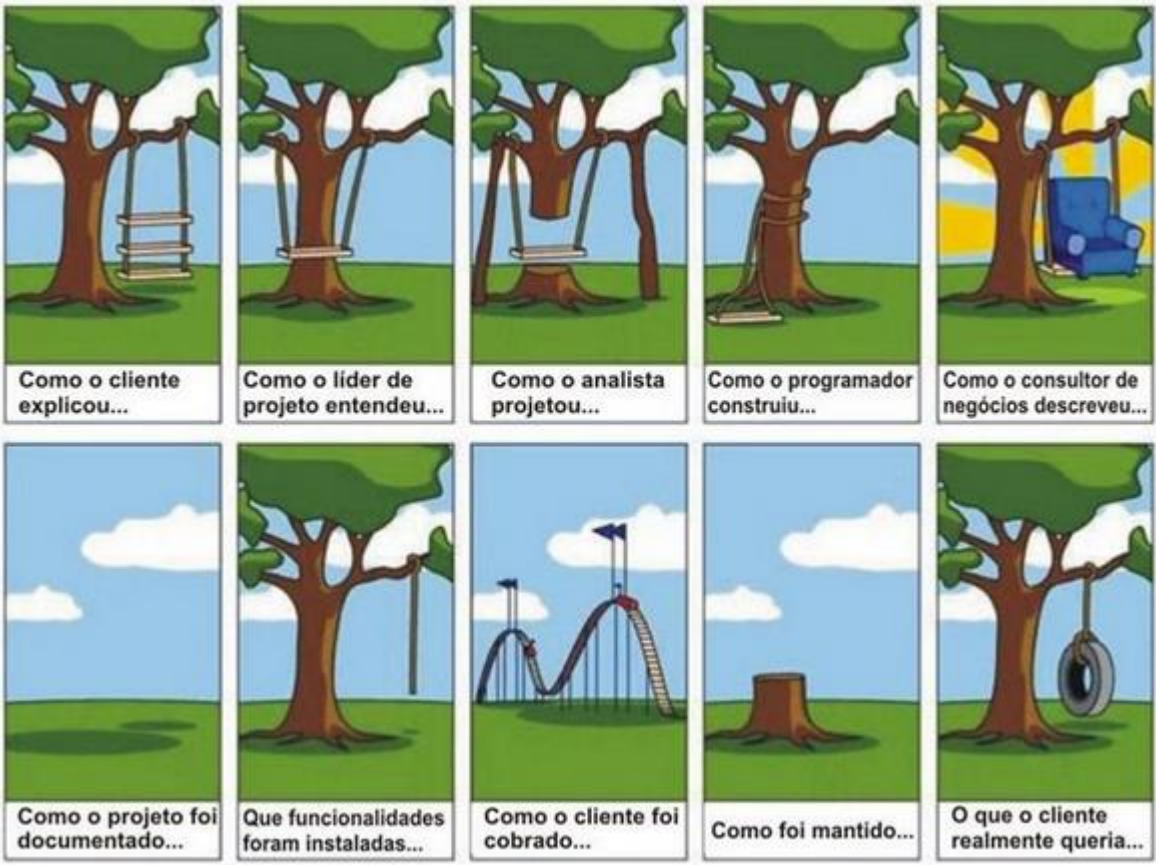
> Os clientes desejam constantemente modificações ou melhorias
> O mercado está sempre mudando, o que força a adoção de novas estratégias por parte das empresas e, consequentemente, de seus sistemas.
> O governo seguidamente promulga novas leis e cria impostos alíquotas ou, ainda, modifica as leis, os impostos e alíquotas já existentes, o que acarreta a manutenção do software (GUEDES, 2009, p. 21).

2.2.1 O que é levantamento de requisitos

Na construção de um software, existem algumas etapas para o desenvolvimento do mesmo, porém, neste trabalho a finalidade não é entrar em

detalhes sobre a UML, mas sim explicar o básico para que o leitor tenha entendimento do software que será apresentado nos capítulos futuros. Uma das etapas existentes na construção de um software é a análise de requisitos, considerada por muitos a parte mais importante, pois é nesse momento onde escutamos os clientes buscando entender a necessidade dele, esta etapa está encarregada de determinar o que o software irá realizar, tudo isso feito por meio de entrevistas. Deve ser realizada com calma e muita cautela, caso contrário, poderá acontecer o que vemos na Figura 1, além disso, toda e qualquer alteração irá afetar as próximas etapas (VERÍSSIMO, 2007; GUEDES, 2009, p. 22).

Figura 1 - Análise de requisitos



Fonte: https://goo.gl/ypRQRc (2009).

2.2.2 Tipos de requisitos

Existem dois tipos de requisitos no levantamento de requisitos, os funcionais e os não funcionais. Os requisitos funcionais dizem respeito a tudo o que o software

deve realizar, já os requisitos não funcionais são as restrições, condições, linguagem de programação, SGBD (Sistema de gerenciamento de banco de dados) e entre outros critérios, isto é, nada disso faz parte das funcionalidades do sistema (VERÍSSIMO, 2007; GUEDES, 2009, p. 23).

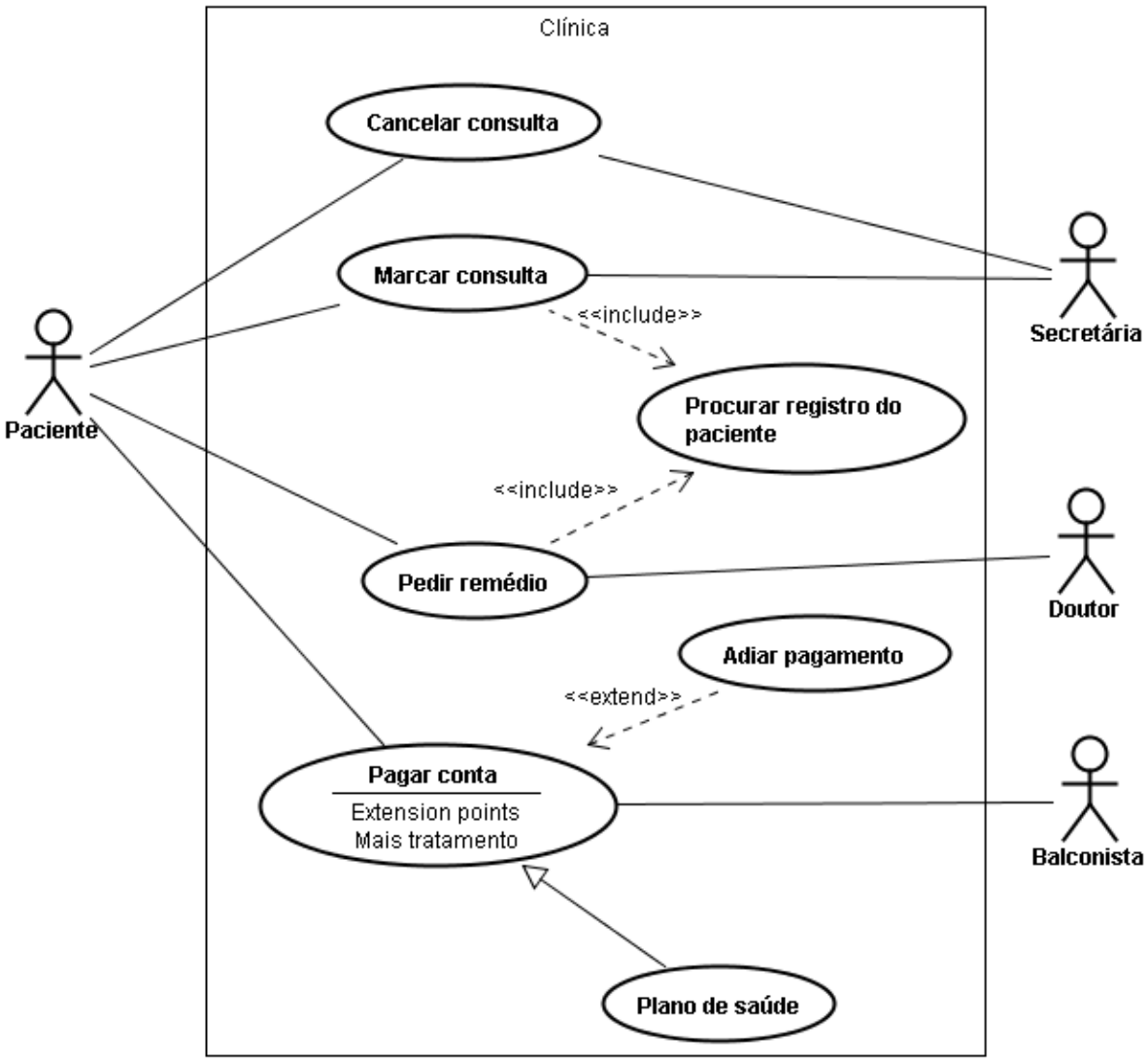
Alguns exemplos de requisitos funcionais seriam, o sistema deve cadastrar médicos profissionais, o paciente pode consultar alguns dados do sistema, o sistema deve gerar boleto, resumindo, tudo que se refere a funcionalidade. Já os exemplos de requisitos não funcionais seriam, o sistema deve imprimir um relatório em até 5 segundos, somente maiores de idade poderá obter uma conta, para uma conta será necessário um CPF válido, entre outros, isto é, não se é uma funcionalidade, mas sim regras e condições (VERÍSSIMO, 2007).

2.3 Diagramas da UML

Como supracitado, a UML é uma linguagem visual, isto significa que há uma forma gráfica de representar como o sistema irá se comportar, quais funcionalidades vão existir e entre outros. Os diagramas são a forma utilizada pela UML de representar os sistemas graficamente, o número de diagramas necessário para representar um software é extenso, por esse motivo, iremos resumir o seguinte trabalho baseando-se em 4 diagramas, o diagrama de classe, caso de uso, atividade e de sequência (GUSTAVO, 2009).

O diagrama de caso de uso é o diagrama mais geral da UML, utilizado normalmente na fase de levantamento de requisitos, mas servirá de base para todos os outros diagramas e será constantemente consultado, sua finalidade é deixar claro e objetivo as funcionalidades do sistema, o que e como seus atores e usuários vão interagir com o sistema, e até mesmo demonstrar algum hardware especial. Este diagrama busca ser o mais simples possível para seu usuário final, um exemplo prático de como ele é representado é mostrado na figura 2. (GUEDES, 2009, p. 31).

Figura 2 - Exemplo de diagrama de caso de uso



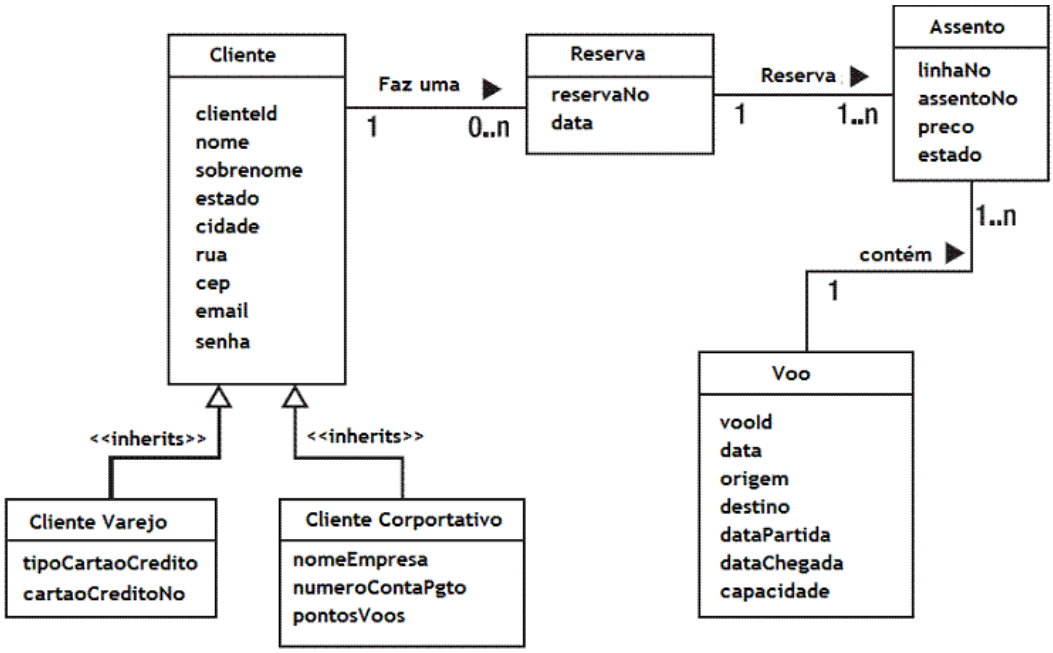
Fonte: https://goo.gl/TJRJBU (2014).

Antes de decifrar o conceito do diagrama de classes, é necessário ter o conhecimento sobre o que é uma classe, portanto, a definição de classe em programação trata-se de uma estrutura que abstrai um conjunto de características similares, sendo nela definida seu nome, seus métodos e atributos (WIKIBOOKS, 2017; DOUGLAS, 2016).

Estabelecido o conceito de classes, podemos dar continuidade ao diagrama de classes, o objetivo do diagrama de classes é apresentar uma visão dos valores armazenados pelos objetos (este conceito será trabalhado nos capítulos posteriores), sendo assim, ele representa de forma gráfica a estrutura e ligações

entre todas as classes de um sistema de uma forma visualmente limpa, como podemos ver na figura 3 (GUEDES, 2009, p. 34).

Figura 3 - Exemplo de diagrama de classe

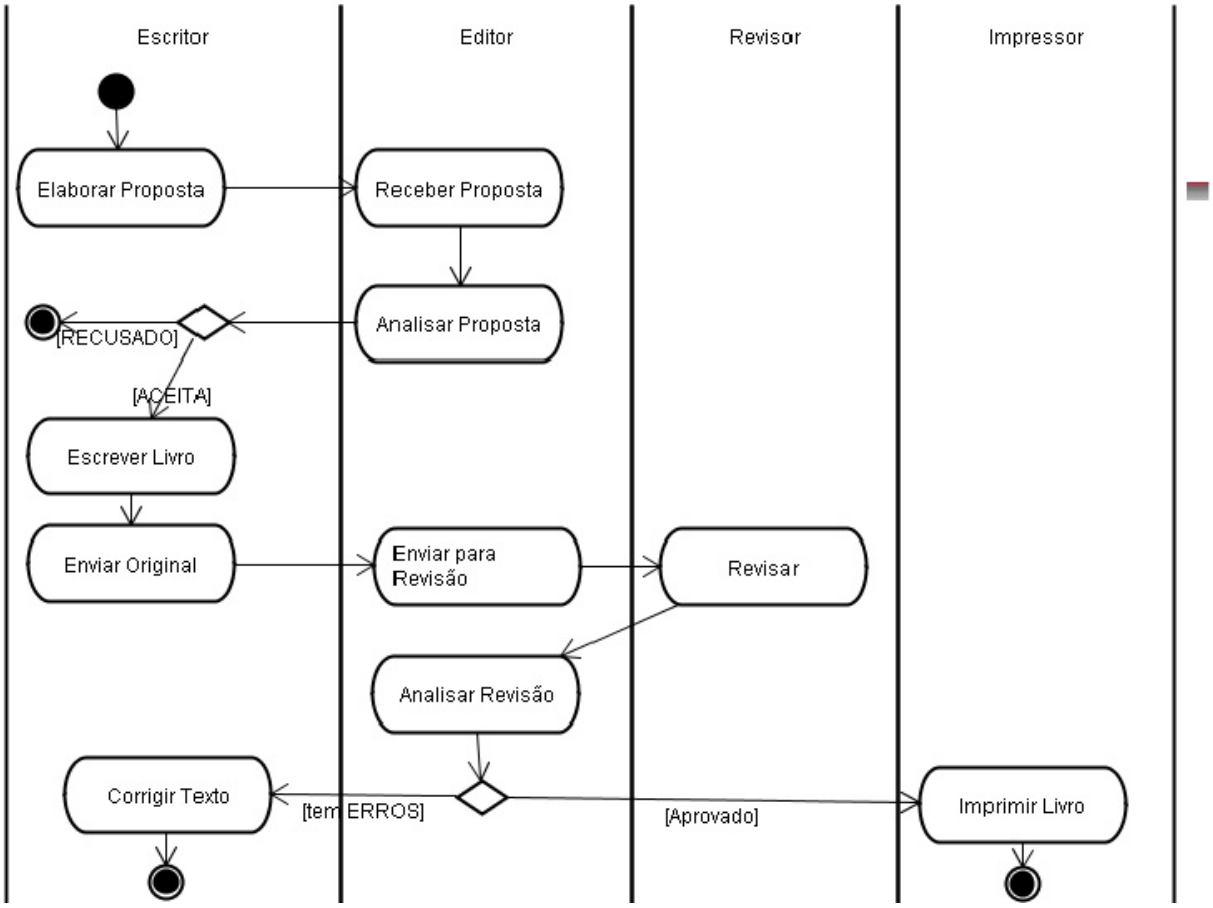


Fonte: https://goo.gl/P4unta (2010).

O diagrama de atividade é muito parecido com um fluxograma, preocupa-se em descrever os passos a serem percorridos quando se solicita uma função no sistema, concentra-se na representação do fluxo de controle de uma atividade, demonstrado seus atores, podendo ser de uma visão macro ou micro (GUEDES, 2009, p.38; VENTURA, 2016).

Podemos ver um exemplo deste diagrama na figura 4.

Figura 4 - Exemplo de diagrama de atividade



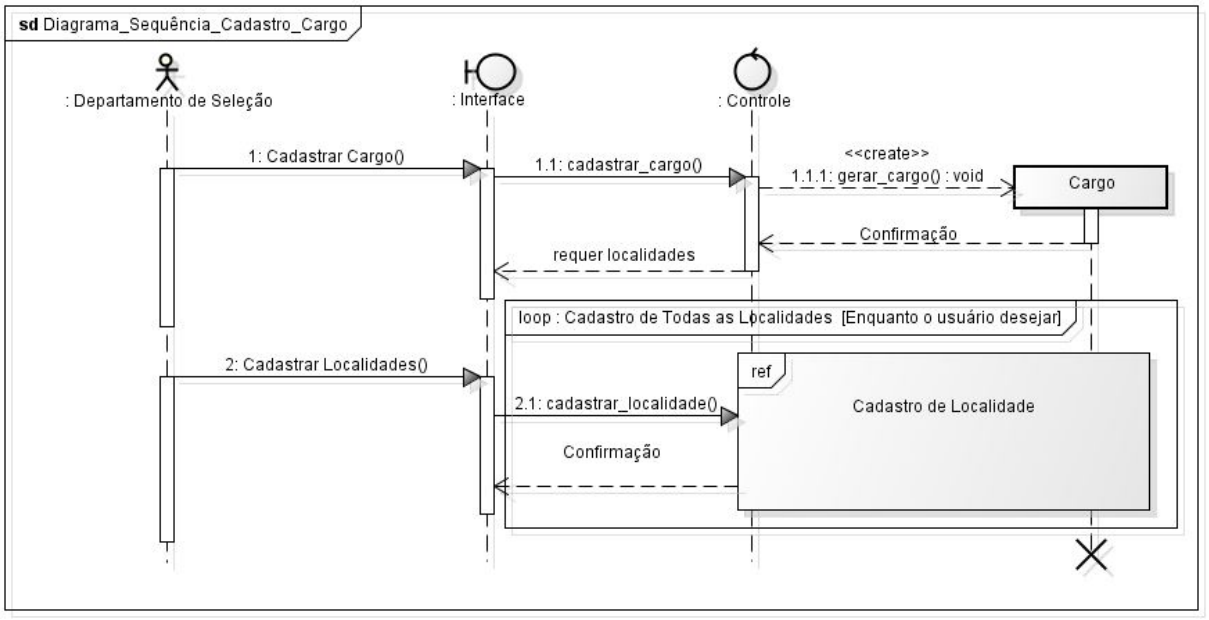
Fonte: https://goo.gl/1VJPQG (2006).

O diagrama de sequência se preocupa com a ordem temporal das mensagens trocadas entre os objetos envolvidos em uma ação, costumando também identificar seu autor, como cita GUEDES:

> Um diagrama de sequência costuma identificar o gerador do processo modelado, bem como o ator responsável por esses ventos, e determina como o processo deve se desenrolar e ser concluído por meio da chamada de métodos disparados por mensagens enviadas entre os objetos (GUEDES, 2009, p. 35).

Um exemplo deste diagrama pode ser observado na figura 5.

Figura 5 - Exemplo de diagrama de sequência



Fonte: https://goo.gl/U9KkRk (2012).

2.4 O que é um Banco de Dados?

Como o próprio nome já induz, banco de dados é uma coleção de dados, segundo Korth “é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico”, sempre que informações podem ser agrupadas e tratam de um mesmo assunto, possui-se um banco de dados. Um exemplo do que pode ser considerado um banco de dados, seria a lista telefônica, onde temos muitas informações em que todas elas se relacionam (RICARDO, 2006 apud KORTH 1986). Existem vários modelos de banco de dados, porém, neste trabalho iremos dar total ênfase ao banco de dados no modelo relacional.

2.4.1 Sistema Gerenciador de Banco de Dados

Muitas pessoas confundem o conceito de banco de dados com o conceito de sistema gerenciador de banco de dados ou SGBD, ambos são tratados genericamente como banco de dados, porém, isso é um equívoco (TONSIG, 2006, p. 18).

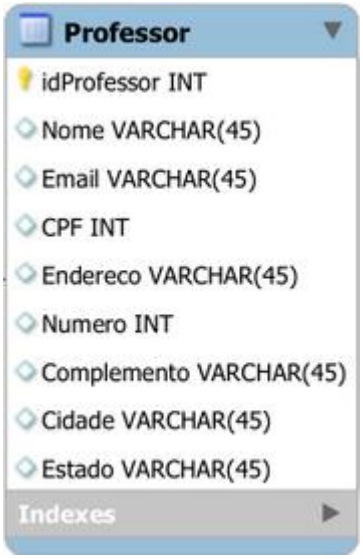
O sistema gerenciador de banco de dados se define como um software manipulador do banco de dados, isto é, ele manipula os dados como de acordo com o que o usuário que tem acesso a esse banco de dados desejar, podendo excluir, alterar, criar, modificar e pesquisar dados quando necessário (TONSIG, 2006, p. 18; RICARDO, 2006).

O objetivo do SGBD é isolar o usuário que está utilizando o software dos detalhes do banco de dados que o software utiliza, o banco de dados busca a independência da aplicação, ou seja, ele existe mesmo sem a aplicação em que está sendo utilizado (TONSIG, 2006, p. 18; RICARDO, 2006).

O SGBD também tem a função classificar cada tipo de dado que o banco de dados possui, ele realiza tudo isso através de tabelas, mais conhecidas como entidades, onde se encontra os identificadores e os tipos de dados (TONSIG, 2006, p. 18; RICARDO, 2006).

Podemos ver um exemplo na figura 6 de uma entidade em que o SGBD pode interagir.

Figura 6 - Exemplo de entidade



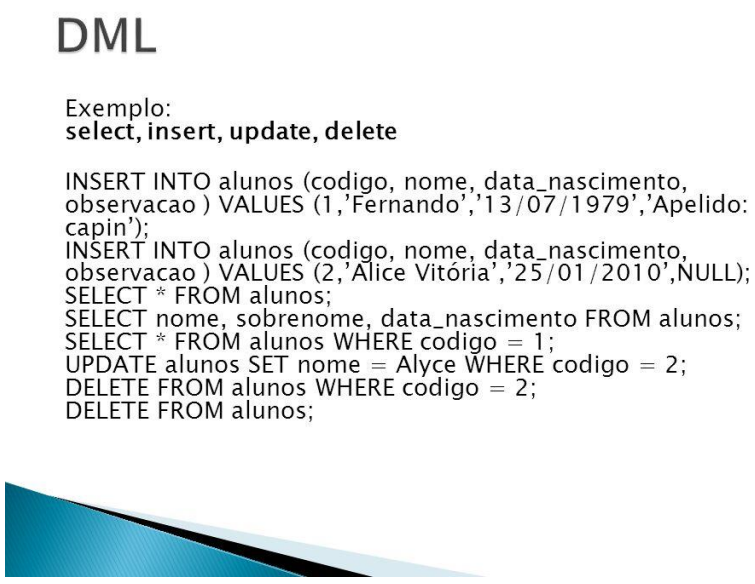
Fonte: Própria autoria

Dentro do SGBD, existem duas definições de linguagem, uma denominada DML (Data definition language) e outra denominada DDL (Data manipulation language). A DML tem a função de definir objetos dentro do banco de dados, como

tabelas, views, procedures e entre outros, a DML permite excluir, alterar e criar todos os objetos citados. A DDL é a parte em que se interage com os dados na tabela criada pela DML, isto é, conseguimos inserir, alterar, excluir e consultar dados dentro da tabela (PANDEY, 2017).

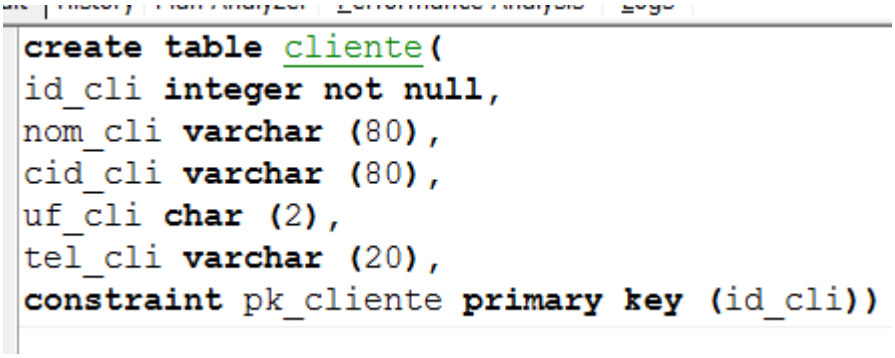
Todo o software necessita de uma linguagem para conseguir realizar suas tarefas, nos SGBDS utilizamos a linguagem SQL, em português, linguagem padrão de consulta estruturada. Alguns comandos utilizados dentro do SQL na DML são CREATE, usado para criar uma nova tabela, ALTER, usado para alterar a estrutura de uma tabela e DROP, usado para excluir uma tabela do banco de dados, como podemos ver na figura 7. O mesmo acontece com a DDL, porém com comandos distintos, na DDL, para criarmos um registro usamos o comando INSERT, para alterarmos usamos UPDATE, para deletar usamos DELETE e para selecionar registros utilizamos o SELECT, como podemos ver na figura 8 (ALENCAR, 2014).

Figura 7 - Exemplo da utilização da DML



Fonte: https://goo.gl/UCzVUb (2014)

Figura 8 - Exemplo da utilização da DDL



```
create table cliente(
id_cli integer not null,
nom_cli varchar (80),
cid_cli varchar (80),
uf_cli char (2),
tel_cli varchar (20),
constraint pk_cliente primary key (id_cli))
```

Fonte: https://goo.gl/cynBVk (2006)

2.4.2 Diagrama para Banco de Dados

Assim como a UML, que veio para facilitar a vida de muitas pessoas deixando mais claro o que acontece de forma gráfica para a parte de banco de dados, temos também um gráfico muito importante, denominado de DER ou diagrama de entidade e relacionamento. O objetivo deste diagrama é mostrar de forma clara e efetiva como os dados estão sendo estruturados dentro das entidades, mostrando seu nome e o relacionamento que ele possui com outras entidades (RODRIGUES, 2014).

Entretanto, esta etapa gráfica só é possível, após concluída a parte do MER ou modelo entidade relacionamento, o MER tem como objetivo, desenvolver um modelo conceitual sobre o banco de dados a ser criado, isto é, decidir as entidades existentes, decidir os campos que a entidade vai possuir, incluindo o tipo do campo e o tamanho do mesmo, verificar se a entidade é uma entidade fraca, forte ou associativa, decidir qual atributo será chave primária PK e qual será estrangeira FK e entre outros fatores, após tudo isso ser decidido podemos partir para o DER (RODRIGUES, 2014).

A chave primária no banco de dados, significa que aquele atributo jamais pode ser repetido na tabela, ou seja, o valor daquele campo não pode se repetir na mesma tabela. A chave estrangeira diz respeito aos relacionamentos entre tabelas, como cita Gasparotto:

> De forma sucinta, a chave estrangeira é uma referência, em uma tabela, a uma chave primária de outra tabela. Para facilitar a compreensão, tomemos como exemplo duas tabelas: Pessoa e Carro. Para montarmos um relacionamento entre elas, poderíamos ter, na tabela Carro, o campo ID_Pessoa fazendo referência à chave primária da tabela Pessoa (GASPAROTTO, 2017).

A entidade forte é quando a entidade é independente, ou seja, não necessita de nenhuma outra entidade para existir que é totalmente a oposta da entidade fraca, que necessita de alguma outra entidade para ela existir, pois individualmente elas não fazem sentido. A entidade associativa é quando a entidade surge de um relacionamento entre as entidades de uma relação muitos para muitos, também como citado por Gasparotto:

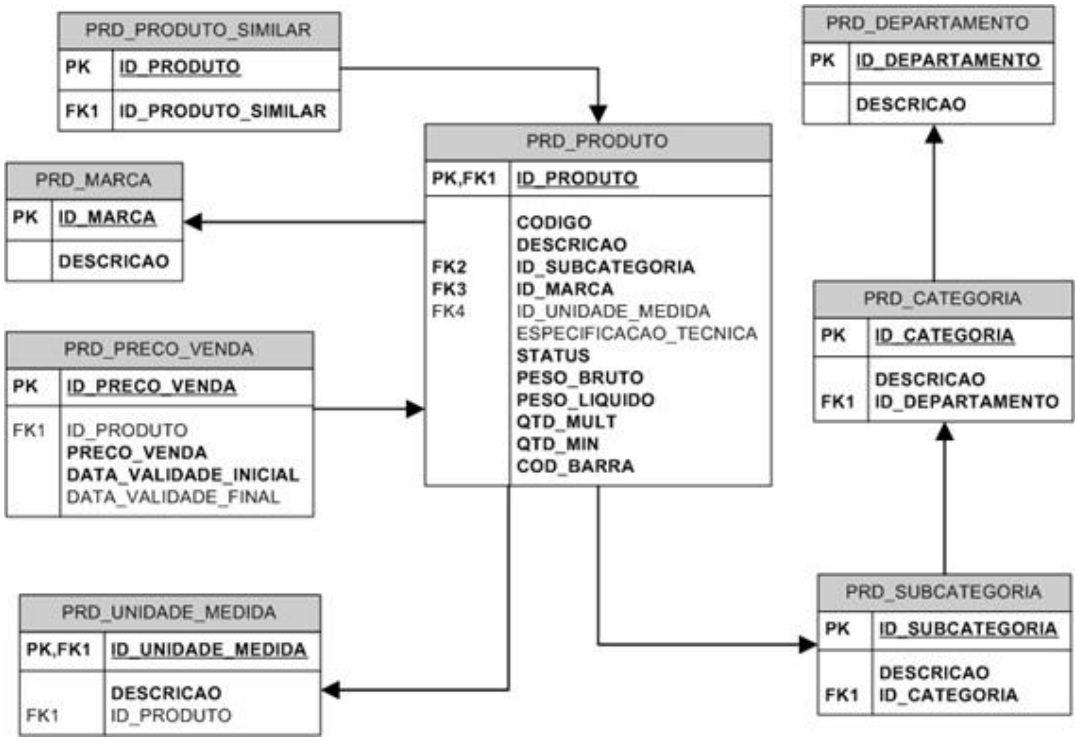
> Para melhor compreender esse conceito, tomemos como exemplo uma aplicação de vendas em que existem as entidades Produto e Venda, que se relacionam na forma muitos-para-muitos, uma vez que em uma venda pode haver vários produtos e um produto pode ser vendido várias vezes (no caso, unidades diferentes do mesmo produto). Em determinado momento, a empresa passou a entregar brindes para os clientes que comprassem um determinado produto. A entidade Brinde, então, está relacionada não apenas com a Venda, nem com o Produto, mas sim com o item da venda, ou seja, com o relacionamento entre as duas entidades citadas anteriormente. Como não podemos associar a entidade Brinde com um relacionamento, criamos então a entidade associativa "Item da Venda", que contém os atributos identificadores das entidades Venda e Produto, além de informações como quantidade e número de série, para casos específicos. A partir daí, podemos relacionar o Brinde com o Item da Venda, indicando que aquele prêmio foi dado ao cliente por comprar aquele produto especificamente (GASPAROTTO, 2017).

Os relacionamentos entre as entidades podem ser de 3 tipos, um para um, onde cada uma das entidades envolvidas referenciam obrigatoriamente apenas a uma unidade da outra, um para muitos, onde cada uma das entidades envolvidas de um lado podem se envolver uma ou mais vezes, porém do outro lado, pode se envolver somente uma vez, e por último existe o relacionamento muitos para muitos,

onde ambas podem se relacionar várias para ambos os lados, “por exemplo, em um sistema de biblioteca, um título pode ser escrito por vários autores, ao mesmo tempo em que um autor pode escrever vários títulos. Assim, um objeto do tipo autor pode referenciar múltiplos objetos do tipo título, e vice-versa” (GASPAROTTO, 2017).

Após estabelecidos esses conceitos, podemos agora entender como isso funciona através da figura 9.

Figura 9 - Exemplo de diagrama entidade relacionamento



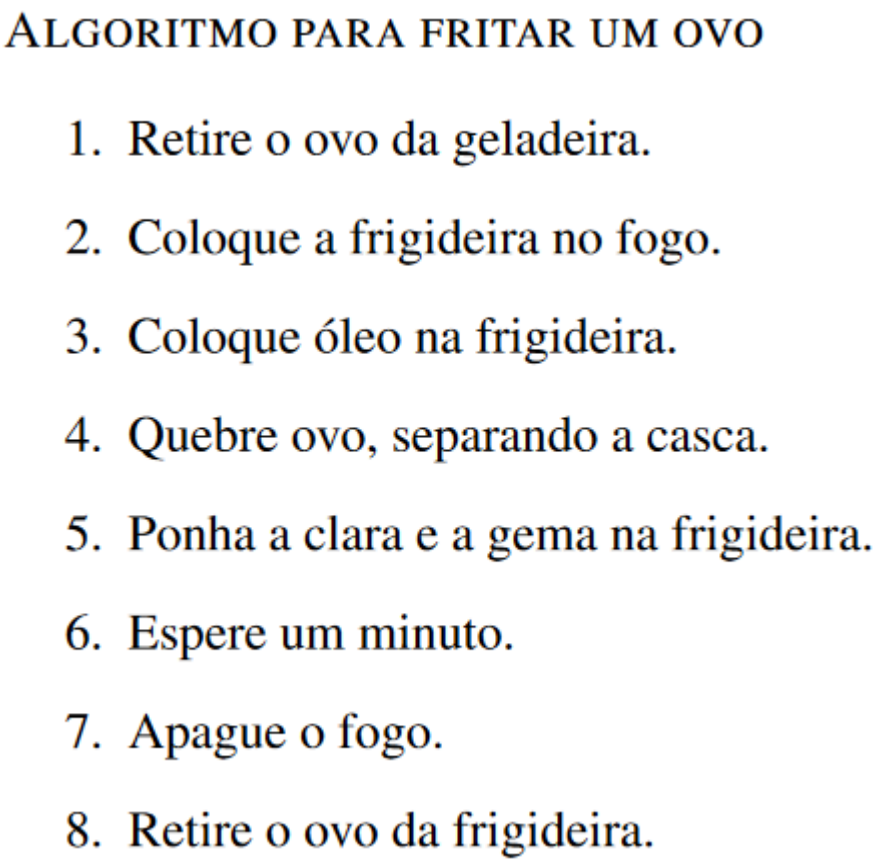
Fonte: https://goo.gl/K6MqX2 (2014)

2.5 O que é linguagem de programação

Para tudo na vida é necessário uma linguagem para que ocorra a comunicação, seja ela corporal, escrita ou falada, e com os computadores não é diferente (SOUSA; JÚNIOR; FORMIGA, 2014, p. 2). Ao contrário do que muitos dizem, o computador não é inteligente, ele somente obedece a instruções que foram previamente escritas para serem seguidas, classificamos essas instruções passadas para o computador como algoritmo, pois, para ser um algoritmo é necessário um conjunto não ambíguo de passos executáveis para um processo finito, isto é, não é

aplicável somente para a área de computação, podemos ver um exemplo de algoritmo na figura 10 (SOUSA;JÚNIOR;FORMIGA, 2014, p. 3).

Figura 10 - Exemplo de algorítmo



Fonte: Adaptado (SOUSA; JÚNIOR; FORMIGA, 2014).

O algoritmo que está presente na figura anterior foi um algoritmo simples e sem condicionais ou repetição de passos com a linguagem em português do Brasil, porém, existem outras formas de se representar algoritmos, como através de fluxogramas, ou como supracitado, podemos demonstrar de forma gráfica a maneira com que funcionalidades vão ser realizadas (SOUSA; JÚNIOR; FORMIGA, 2014, p. 5).

As linguagens de programação para o computador nos permitem realizar condicionais, laços de repetição e o passo a passo não estar necessariamente um abaixo do outro, a quantidade linguagens de programação existentes hoje é muito grande, porém elas são separadas em paradigmas de programação, como por

exemplo, linguagem estruturada e linguagem orientada a objeto (SOUSA; JÚNIOR; FORMIGA, 2014, p. 6 e 16).

2.5.1 O que é Programação orientada a objeto

"É um paradigma de programação baseada no conceito de classes e objeto", (MELLO; CHIARA; VILLELA, 2002, p.13), enxerga adversidade como uma coleção de objetos, que se comunicam trocando mensagens entre classes e outros objetos (SOUSA; JÚNIOR; FORMIGA, 2014, p. 16). O conceito de classe, como citado acima, é um elemento em que dados e procedimentos podem ser agrupados, as classes são codificações no formato de arquivos e sempre que é utilizada como um tipo de dado para criação de uma variável recebe o nome de objeto (MELLO; CHIARA; VILLELA, 2002, p.13).

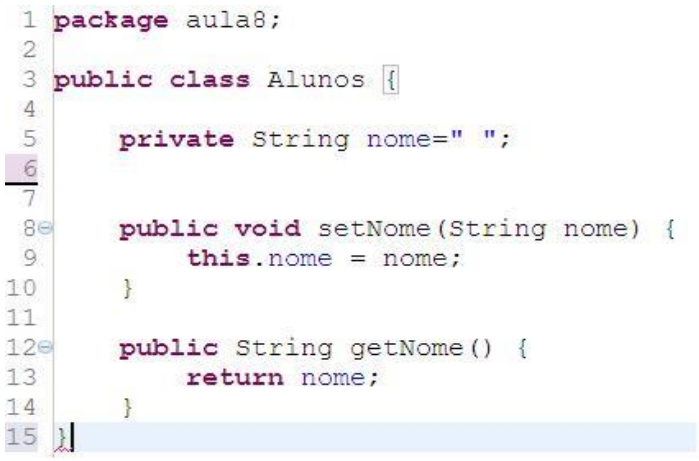
2.5.2 O que é uma classe

De acordo com Mello, Chiara e Villela o conceito de classe é:

> A classe é definida como uma estrutura de dados que contém métodos e atributos. No paradigma da orientação a objetos, os procedimentos ou funções (presentes em linguagens estruturadas, tais como C e Pascal) são chamados de métodos de uma classe. As variáveis, são declaradas dentro de uma classe, são chamadas de atributos. (MELLO; CHIARA; VILLELA, 2002, p.13).

Um exemplo de classe feita em linguagem Java, que é a base da linguagem Android, pode ser vista na figura 11.

Figura 11 - Exemplo de classe em Java



```
1  package aula8;
2
3  public class Alunos {
4
5      private String nome=" ";
6
7
8      public void setNome(String nome) {
9          this.nome = nome;
10     }
11
12     public String getNome() {
13         return nome;
14     }
15 }
```

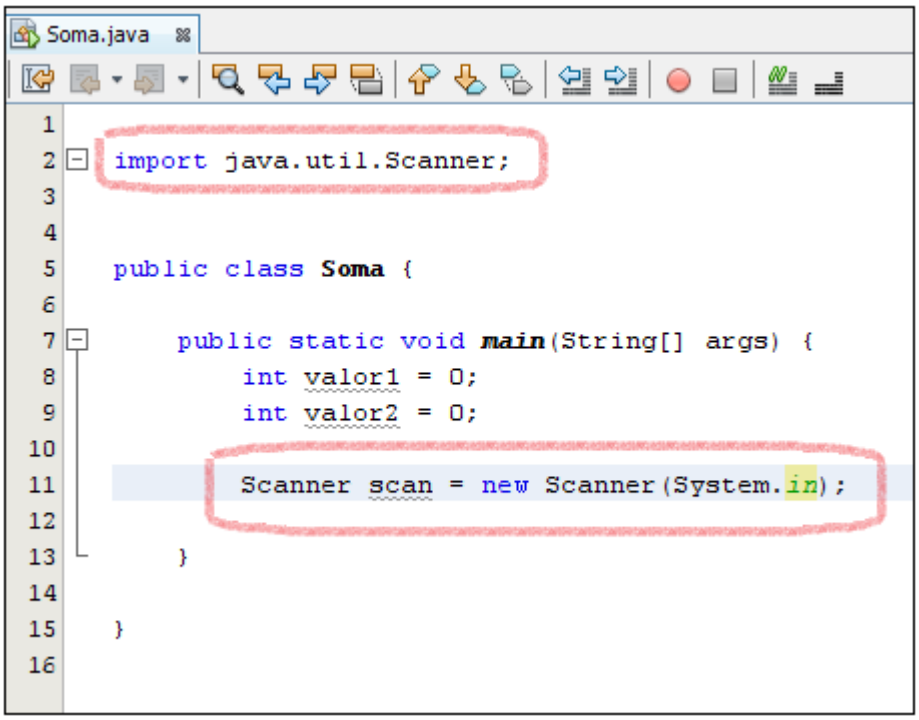
Fonte: https://goo.gl/hsDAZM (2009).

2.5.3 O que é um objeto

O objeto é a instância de uma classe, ou seja, criou-se uma ocorrência de determinada classe onde foi atribuída para uma variável do tipo da mesma, com isto, temos o conceito de objeto. A classe é o código fonte em um arquivo de texto, sem que ela seja instanciada a menos que seja uma classe abstrata, ela não serve de nada, por isso necessitamos do objeto, pois através dele podemos executar funções que foram definidos anteriormente na classe em outras partes do programa também é possível modificar e acessar os dados da classe (MELLO; CHIARA; VILLELA, 2002, p.14).

Um exemplo de instância de classe pode ser visualizado na figura 12.

Figura 12 - Exemplo de instância de classe



Fonte: https://goo.gl/WQ7Ltf (2012)

2.6 Android, Kotlin e Android Studio

Android é uma plataforma de desenvolvimento para dispositivos móveis existente desde 2003 baseada em Linux que é código aberto, portanto, qualquer pessoa que sinta vontade de modificar pode realizá-lo sem nenhum problema, a plataforma utiliza a linguagem Java para desenvolver as aplicações, consequentemente, qualquer desenvolvedor pode construir uma aplicação de forma gratuita (LECHETA, 2009, p. 19).

Apesar da existência do Android, o tempo em que se começou a investir nessa tecnologia de forma agressiva é pouco, sendo a empresa Google a pioneira a construir um SDK para o desenvolvimento de aplicativos nesta plataforma, o SDK recebeu o nome de Android Studio (LECHETA, 2009, p. 19 e 20).

A maioria esmagadora no mundo de dispositivos móveis, utilizava até o ano de 2016 a linguagem Java para o desenvolvimento de suas aplicações, porém, depois do ano de 2017, uma nova linguagem foi anunciada como oficial para o desenvolvimento de aplicações móveis, esta linguagem possui o nome de Kotlin (MITRUT, 2017).

Kotlin é uma linguagem de programação funcional, 100% interoperável com Java, isto é, é possível utilizar classes e funções de aplicações Java, pois no fundo o Kotlin é somente uma maneira mais prática e eficiente de escrever seu código porque quando interpretado, ele gera um código Java através da máquina virtual do Android Studio (MITRUT, 2017).

O Android Studio é uma IDE, ou Ambiente Integrado de Desenvolvimento em tradução livre. A IDE é um software criado com a finalidade de facilitar o desenvolvimento de aplicações, neste tipo de softwares existe todo o tipo de ferramenta necessária para o desenvolvimento como visualização de erros, compilar bibliotecas, referenciar API, buscar plugins e colocá-los em seu projeto, compilar o código, tudo através da IDE desenvolvida. Em suma, o Android Studio é primordial para o desenvolvimento de aplicativos móveis para a plataforma Android trazendo maior facilidade e conforto para o desenvolvedor (FILIPE, 2017; NOVAES, 2017).

2.7 O que é linguagem de marcação

A linguagem de marcação é um sistema para marcar um documento que indica sua estrutura lógica (como parágrafos), e fornece instruções para seu layout na página, especialmente para transmissão eletrônica e exibição (BAX, 2000).

2.7.1 O que é XML

XML é uma linguagem de marcação, ou seja, um sistema de codificação que permite qualquer tipo de informação através da World Wide Web, ela tem muito em comum com o HTML, porém ela é verdadeiramente para todos os propósitos oferecendo uma ampla variedade de aplicações, isto devido ao fato de que quem cria a estrutura do arquivo XML, é o próprio autor (LIGHT, 1999 , p. 1)

Todo o layout de uma aplicação em Android, é feita através do XML, como pode ser visualizado na figura 13.

Figura 13 - Exemplo de layout construído em Android através do XML

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:layout_height="wrap_content"
        android:id="@+id/linearLayout1"
        android:layout_width="wrap_content"
        android:layout_x="20dip"
        android:layout_y="16dip">
        <EditText
            android:text="15"
            android:id="@+id/etSecond"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:inputType="phone" />
        <Button
            android:id="@+id/btnStart"
            android:text="START"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
```

Fonte: https://i.stack.imgur.com/LZNi5.png (2011)

2.8 Oque é uma API

API, em inglês, Application Programming Interface, ou Interface de programação de aplicações em português, é uma ponte, que conecta aplicações, onde podem se interagir mesmo que tenham linguagens de programação distintas (FERNANDES, 2018).

Se uma aplicação usa a API do Google Maps por exemplo, esta aplicação pode requisitar alguns dados que a API oferece, sendo que esses dados são somente permitidos para compartilhar após a criação de uma conta e validação dela pela própria criadora da API, resumidamente, temos acessos a alguns dados do banco de dados da empresa que oferece a API (FERNANDES, 2018).

Já no caso do Facebook, quando fazemos uso da API dele precisamos de permissões mais específicas, depois do vazamento de dados que ocorreu em 2017, o Facebook atualizou ainda mais suas políticas de segurança quanto ao uso da API, pois como supracitado, o uso de informações de usuários é liberado, porém, antecede a uma análise friamente feita pela empresa (FERNANDES, 2018).

A API é invisível para o usuário final, já que não passa de uma ponte interna feita no código, um exemplo prático de API que acontece frequentemente é o famoso “fazer login com o Facebook”, o site em si não possui as informações de login do seu Facebook, mas faz a consulta dos dados dentro do banco de dados do Facebook, através da API (FERNANDES, 2018).

2.9 Fase de testes de um software

A fase de teste dentro do desenvolvimento software é de vital importância e necessidade, evitando o aumento de custo com correção de falhas conforme o desenvolvimento do software ocorre, entretanto algumas empresas ainda negligenciam isso, outras empresas são ainda mais ousadas e sequer tem uma equipe para realizar o teste de software, quando isso acontece, colocam os próprios desenvolvedores para testa-lo e isso é um erro fatal, pois quando se desenvolve algo e a mesma pessoa a testa é muito difícil perceber erros (SOUZA; GASPAROTTO, 2013, p. 1).

2.9.1 A importância do teste de software

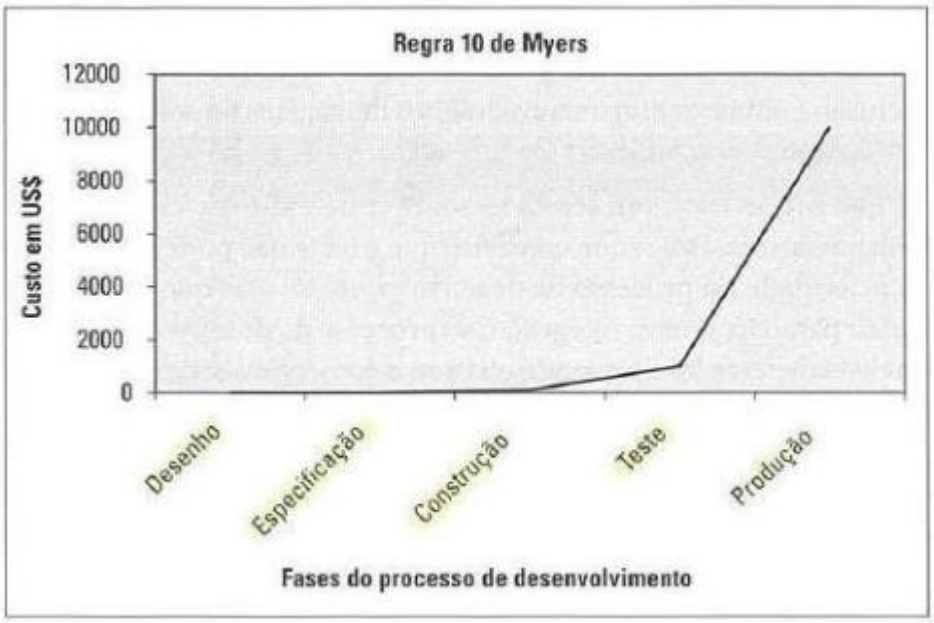
Como citado anteriormente, esta etapa é de suma importância e totalmente significativa, pois conseguimos identificar erros inseridos no projeto ao decorrer do mesmo e verificar se todos os requisitos do cliente foram atendidos antes do software entrar em ambiente de produção, o que pode dificultar na correção de erros (SOUZA; GASPAROTTO, 2013, p. 2, apud MOLINARI, 2012).

Jamais será lançado no mercado um software que não há erros, os ambientes de utilização do software, apesar de estarem na mesma plataforma, são totalmente diferentes, isto é, quem testa o software não tem todas as possibilidades de recriar todos os ambientes possíveis para testar o software, até porque a quantidade de número de testes seria infinita (SOUZA; GASPAROTTO, 2013, p. 3).

Quanto mais cedo o erro no software for descoberto e corrigido, menor é seu custo no projeto. De acordo com Souza e Gasparotto (2013, p. 3 apud MYERS, 1979) “o custo em correção cresce exponencialmente 10 vezes para cada estágio que o projeto avança”.

Podemos ver um exemplo desse crescimento exponencial na figura 14.

Figura 14 - Custo dos efeitos com o passar das fases do projeto de desenvolvimento de software



Fonte: (SOUZA; GASPAROTTO, 2013).

2.9.2 Técnicas para o teste de software

Existem técnicas de teste de softwares para um melhor feedback, que são classificados com base na origem das informações utilizadas para teste, abordando perspectivas diferentes do software e podem ser classificadas como: Funcional e estrutural (SOUZA; GASPAROTTO, 2013, p. 7 apud NETO, 2013).

2.9.2.1 Técnica estrutural ou Teste de caixa branca

Esta técnica avalia o comportamento interno do sistema, o código fonte. É recomendada nos níveis de Teste de Unidade e Teste de integração que normalmente ficam a cargo dos desenvolvedores do código fonte, por terem uma maior familiaridade do código. O objetivo é auxiliar na redução de problemas que podem ocorrer nas funções ou unidades que compõem um software (SOUZA; GASPAROTTO, 2013, p. 7).

2.9.2.2 Técnica funcional ou Teste de caixa preta

Apesar de o nome ser família, o teste de caixa preto possui este nome pois a estrutura interna do sistema não é considerada enquanto a execução destes testes são realizados, sendo assim, esta técnica pode ser realizada em qualquer nível de teste.

> Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado previamente conhecido. Haverá sucesso no teste se o resultado obtido for igual ao resultado esperado. O componente de software a ser testado pode ser um método, uma função interna, um programa, um componente, um conjunto de programas e/ou componentes ou mesmo uma funcionalidade (SOUZA; GASPAROTTO, 2013, p.7 apud NETO, 2013).

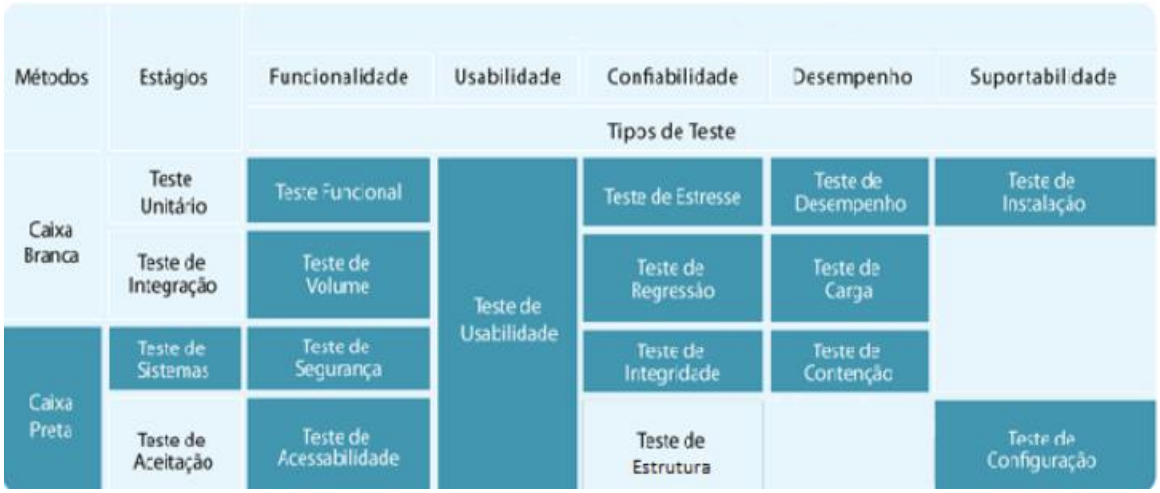
Existem outras técnicas além destas duas, porém não é do objetivo deste projeto que sejam abordadas.

2.9.3 Tipos de testes

Os tipos de testes que serão necessários para o projeto que está sendo desenvolvido será especificado na fase de planejamento do projeto. Vários fatores devem ser levados em consideração para decidir os testes que serão realizados, como por exemplo: o público alvo, se haverá disponibilidade na internet, se tem integração com outros sistemas, se terá acessibilidade para pessoas com necessidades especiais, e entre outros fatores, sendo assim, devem-se identificar os riscos do projeto e seu nível de tolerância a erros (SOUZA; GASPAROTTO, 2013, p. 8 apud BASTOS et al., 2007).

Pode-se visualizar um exemplo de distribuição de tipo de testes por categorias na figura 15, e na figura 16 pode-se visualizar alguns dos tipos de teste existentes juntamente com sua descrição.

Figura 15 - Distribuição de tipos de testes por categorias



Fonte: Adaptado de (SOUZA; GASPAROTTO, 2013).

Figura 16 - Tipos de teste e suas descrições

Tipos de teste	Descrição
Teste de Estresse	Avalia o desempenho do sistema com volume de acesso/transações acima da média esperada e em condições extremas de uso.
Teste de Performance ou Desempenho	Avalia se o sistema atende os requisitos de performance (proficiência) com volume de acesso/transações dentro do esperado. Verifica se o tempo de resposta da aplicação está conforme o esperado.
Teste de Contenção	Avalia se o sistema retorna a um status operacional após uma falha.
Teste de Segurança	Avalia o nível de segurança do sistema quanto a perfis de acesso, permissões e uso de tags html nos campos.
Teste de Regressão	Avalia se funcionalidade que estava funcionando ainda funciona após uma modificação no sistema. Todo o sistema é verificado neste teste.
Teste de Integração	Avalia se a interconexão entre aplicações ou funcionalidades funciona corretamente.
Teste de Funcionalidade	Avalia se o sistema funciona conforme descrito nos requisitos.
Teste de Usabilidade	Verifica se a navegabilidade, objetivos e padrões de tela estão conforme especificado e se atendem da melhor forma ao usuário.
Teste de Volume	Avalia o comportamento do sistema com grande quantidade de dados.
Teste de Acessibilidade	Avalia se a aplicação está navegável para um usuário com algum tipo de deficiência visual.
Teste de Estrutura	Avalia se a codificação foi feita corretamente e se todos os métodos são utilizados.
Teste de Carga	Avalia o funcionamento da aplicação com a utilização de uma grande quantidade de usuários simultâneos.
Teste de Instalação	Testar se a instalação da aplicação obteve sucesso.
Teste de Configuração	Avalia se a aplicação funciona corretamente em ambientes diferentes de hardware ou software (Ex: Navegadores diferentes).

Fonte: Adaptado de (SOUZA; GASPAROTTO, 2013)

3. Desenvolvimento do aplicativo

Este capítulo irá abordar o desenvolvimento do aplicativo, tendo em conta que o capítulo anterior já explicou tudo o que será usado para o desenvolvimento do mesmo, diagramas, banco de dados e todas as outras técnicas supracitadas.

Para o desenvolvimento dos diagramas foi utilizado a aplicação Astah Community e para o desenvolvimento do DER e MER foi utilizado o BrModelo versão 3.2.

3.1 Requisitos funcionais e não funcionais

Todo aplicativo possui um propósito, como apresentado na introdução deste trabalho, sendo assim, o desenvolvimento de software deve começar pelos requisitos funcionais e não funcionais, pois somente assim é possível visualizar o que deve ser desenvolvido para conseguir atingir o propósito necessário.

Abaixo podemos visualizar a tabela de requisitos funcionais do aplicativo me lembre.

Tabela 1: Requisitos funcionais

Identificação	Requisito funcional	Categoria	Prioridade
RF001	O sistema deve criar uma pasta chamada “Sem pasta” quando iniciar pela primeira vez	Programação	Muito alta
RF002	Quando o usuário excluir alguma pasta que já contém compromissos associados, o sistema deve passar os compromissos desta pasta para a pasta “sem pasta”	Programação	Muito alta
RF003	O sistema deve disparar um alerta quando o horário e data do compromisso forem iguais a data e horário atual	Programação	Alta

RF004	O sistema deve alertar o usuário sobre erros e perguntar se realmente deseja excluir algo quando solicitado	Programação	Alta
RF005	Toda vez que o horário atual atingir 23:59 o sistema deve atualizar a lista de notificações a serem disparadas	Programação	Muito alta
RF006	Toda vez que inserido um novo compromisso, o sistema deve atualizar a lista de notificações a serem disparadas	Programação	Muito alta
RF007	Toda vez que excluído um compromisso, o sistema deve atualizar a lista de notificações a serem disparadas	Programação	Muito alta

Fonte: Elaborado pelo autor.

A tabela dos requisitos funcionais deixa o entendimento muito mais claro sobre o que o sistema deve realizar, entretanto, também é necessário saber quais os requisitos não funcionais do sistema. Segue abaixo uma tabela apresentando os mesmos.

Tabela 2 : Requisitos não funcionais

Identificação	Requisito não funcional	Categoria	Prioridade
RNF001	O sistema deve conter uma interface limpa e de fácil entendimento	Design	Alta

RNF002	O aplicativo deve executar em celulares a partir da versão Android 6.0.	Programação	Muito alta
RNF003	O aplicativo deve conter uma filtragem de compromissos por dia do mês e pasta	Programação	Alta

Fonte: Elaborado pelo autor.

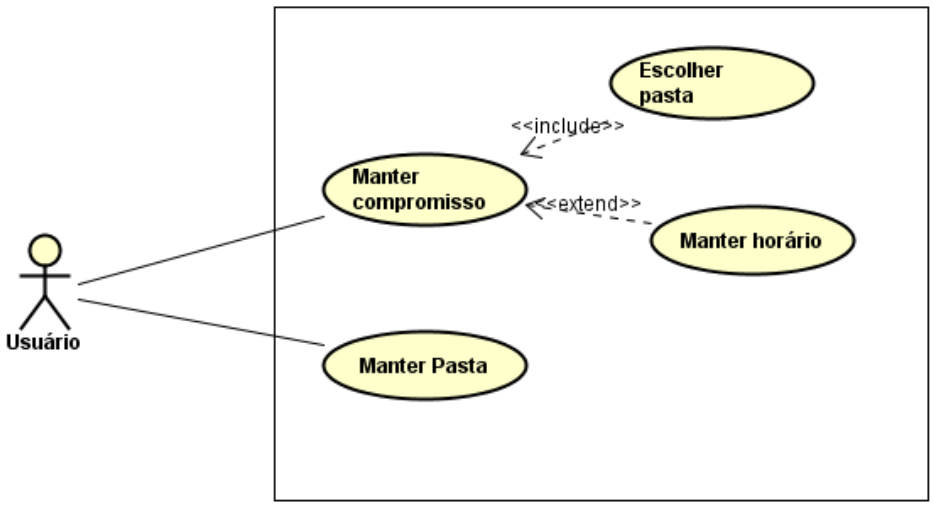
Após definido os requisitos funcionais e não funcionais, podemos dar início a outros diagramas para melhor esclarecimento de desenvolvimento do aplicativo Me Lembre.

3.2 Diagrama de caso de uso

O diagrama de caso de uso é capaz de nos mostrar todos os atores e ações que o usuário pode realizar dentro do sistema.

Na figura 17 podemos visualizar o diagrama de caso de uso do aplicativo Me Lembre em uma versão macro.

Figura 17 : Diagrama de caso de uso



Fonte: Elaborado pelo autor.

As tabelas a seguir apresentam a explicação dos casos de uso, mas como explicação prévia, a palavra “manter” no diagrama refere-se à sigla CRUD que traduzida do inglês, significa: criar, listar, atualizar e excluir.

Tabela 3: Caso de uso - Manter Compromisso

Nome	Manter compromisso
Descrição	O usuário pode realizar as quatro operações do CRUD

Fonte: Elaborado pelo autor.

Tabela 4: Caso de uso - Manter horário

Nome	Manter horário
Descrição	Neste caso de uso o usuário ainda pode realizar o CRUD dos horários dentro de cada compromisso, sendo opcional.

Fonte: Elaborado pelo autor.

Tabela 5: Caso de uso - Selecionar pasta

Nome	Selecionar pasta
Descrição	Neste caso de uso o usuário deverá selecionar em que pasta aquele compromisso irá se encontrar.

Fonte: Elaborado pelo autor.

Tabela 6: Caso de uso - Manter Pasta

Nome	Manter pasta
Descrição	Neste caso de uso o usuário pode realizar todas as operações do CRUD,

	porém, quando se excluir uma pasta, todos os compromissos desta vão ser atribuídas a pasta “Sem pasta”.

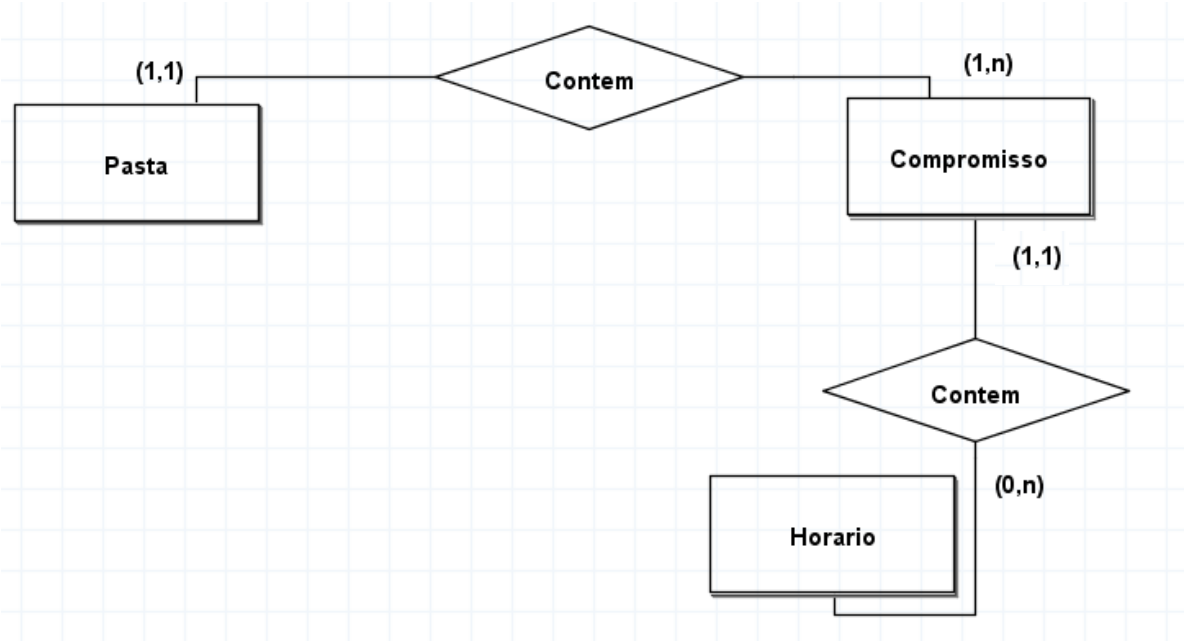
Fonte: Elaborado pelo autor.

Agora que foi estabelecido o diagrama de caso de uso podemos dar continuidade com a documentação do projeto, depois de pensado o que o usuário poderá realizar, é de extrema importância mentalizar e construir o banco de dados em que os dados serão guardados.

3.3 DER e MER do banco de dados do aplicativo

Neste capítulo será apresentado a estrutura de banco de dados construída necessária para a elaboração do projeto, começando pelo MER ou Modelo de Entidade Relacionamento, onde é demonstrado somente as entidades e o relacionamentos entre elas, podemos visualizar o MER na figura 18.

Figura 18: Modelo de Entidade Relacionamento



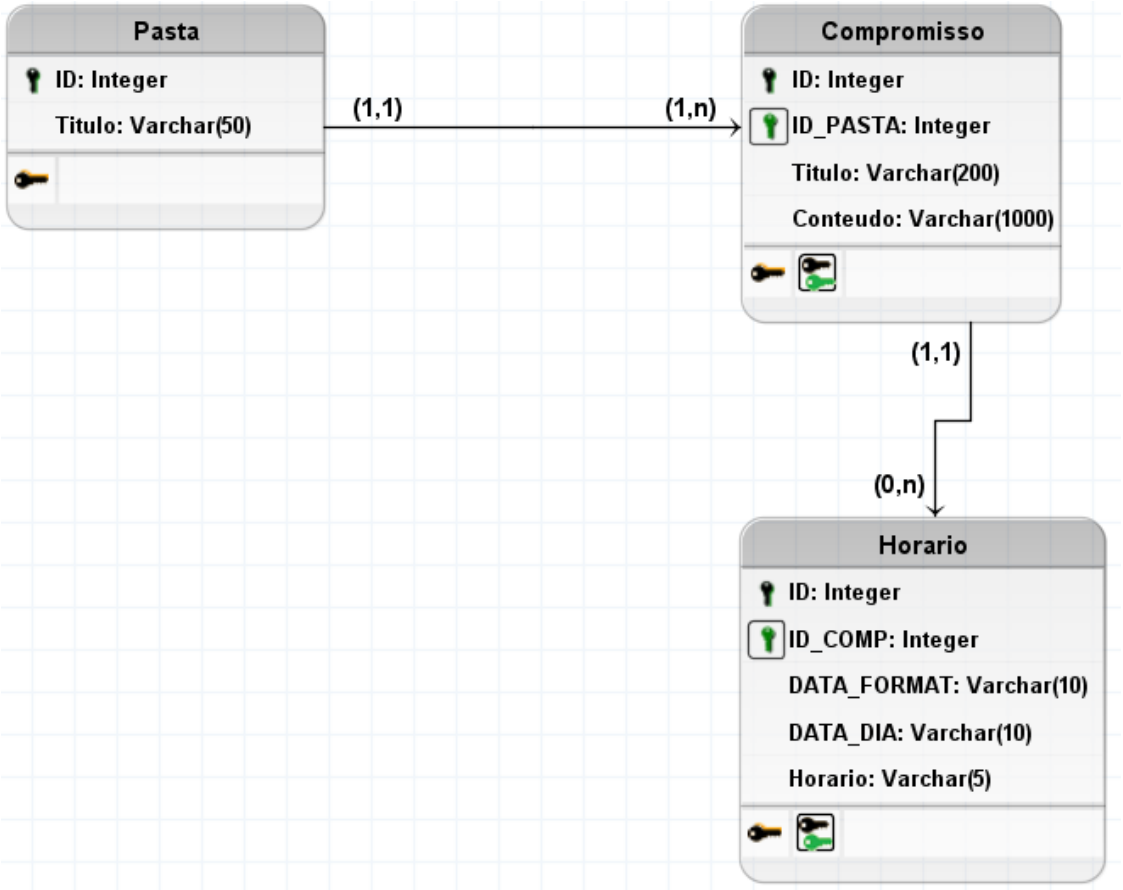
Fonte: Elaborado pelo autor.

O diagrama acima nos demonstra que, um compromisso pode estar em somente uma pasta, porém, uma pasta pode conter vários compromissos, a ligação

entre compromisso e horário tem um funcionamento diferente, onde um compromisso pode ter nenhum ou vários horários.

No DER ou Diagrama de Entidade Relacionamento, podemos visualizar os atributos de cada entidade e o tipo deles como podemos ver na figura 19.

Figura 19 : Diagrama de entidade relacionamento



Fonte: Elaborado pelo autor.

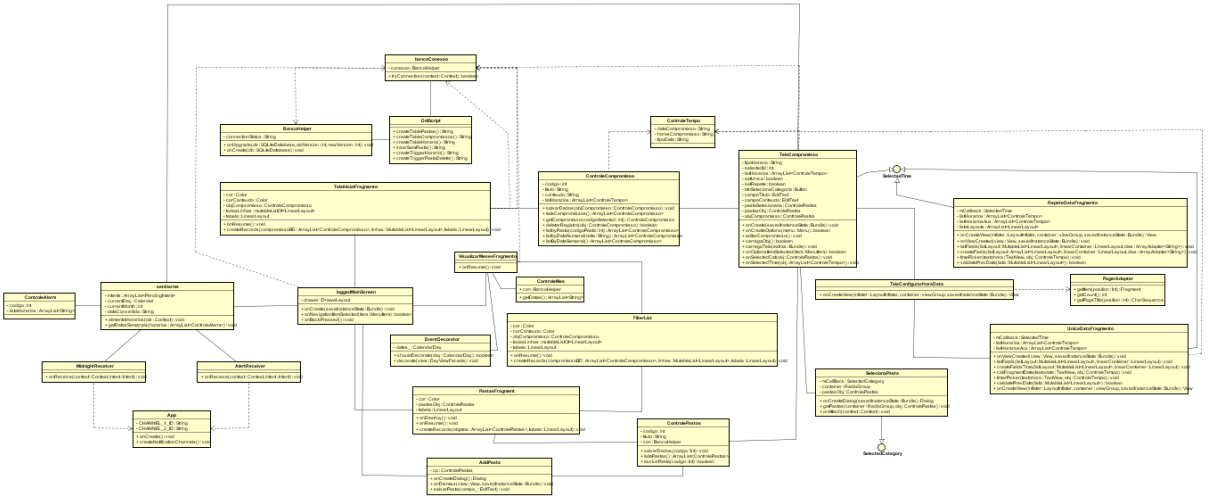
No diagrama de entidade relacionamento, podemos ver os campos e o tipo deles, também existe o relacionamento entre as entidades, porém, este diagrama tem o intuito de mostrar quais dados serão armazenados e como eles serão armazenados.

Agora que a estrutura de banco de dados foi apresentada, podemos partir para o diagrama de classes do programa, onde mostra todas as classes necessárias para o programa funcionar.

3.4 Diagrama de classes

O diagrama de classes tem como objetivo demonstrar como são as classes do programa, mostrando seus atributos, o tipo deles, e seus métodos, também contendo a visibilidade dos métodos e dos atributos. O diagrama de classes desta aplicação é extenso, portanto, será mostrado uma miniatura na figura 20 e nas figuras que procedem é possível encontrar essa miniatura com um zoom para uma melhor visualização de grupos de classe.

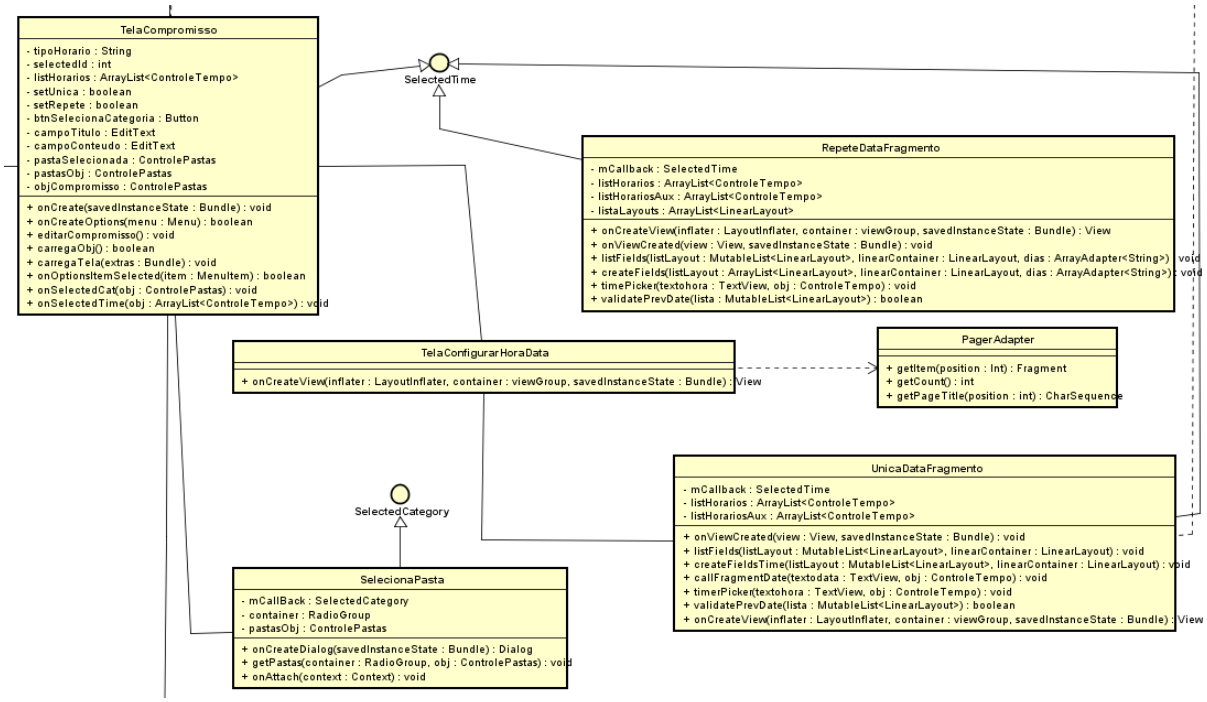
Figura 20: Diagrama de classes completo



Fonte: Elaborado pelo autor.

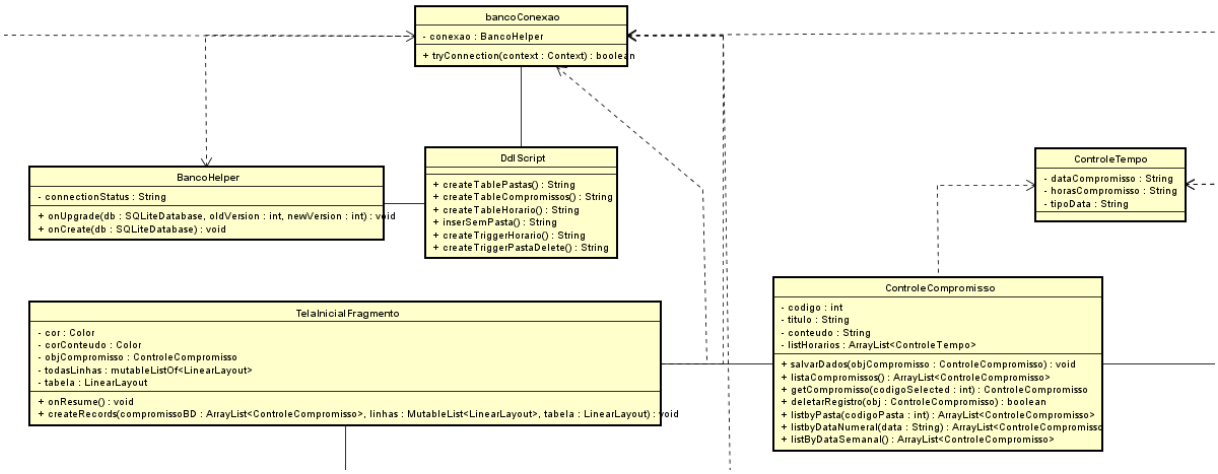
Como supracitado, a figura 20 nos entrega uma visão muito ampla do diagrama, impedindo a visualização clara dos atributos e métodos que cada classe possui, sendo assim, as próximas figuras fazem parte desta figura acima, porém, ampliadas da direita para a esquerda para melhor visualização.

Figura 21: Diagrama de classe - parte 1



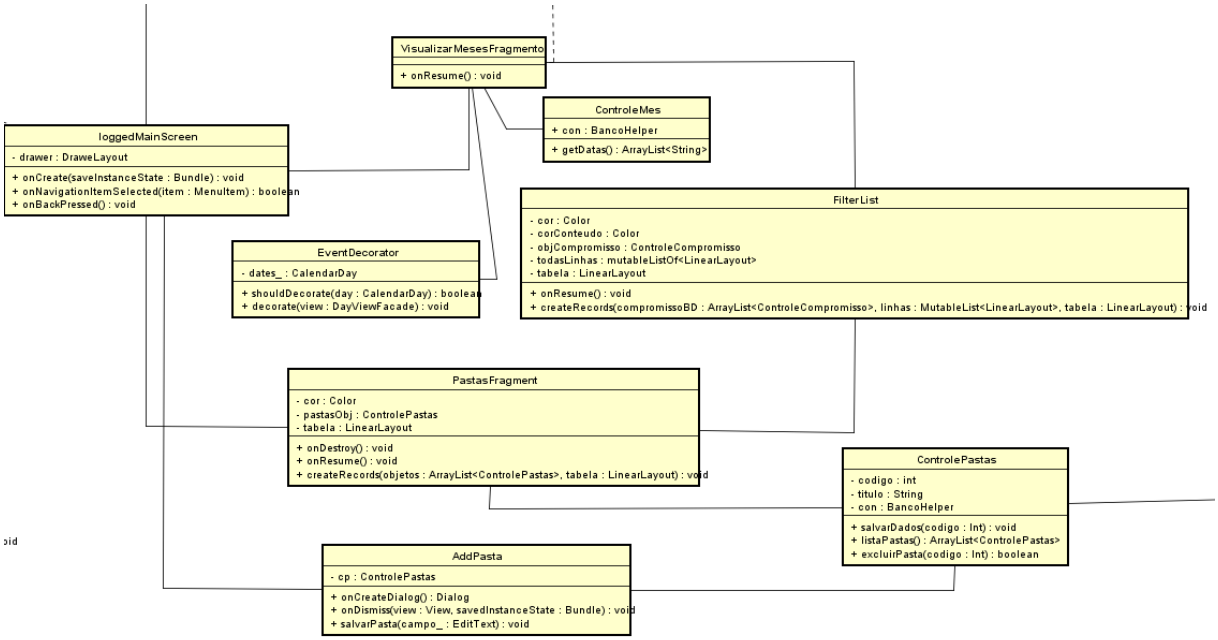
Fonte: Elaborado pelo autor.

Figura 22: Diagrama de classe parte 2



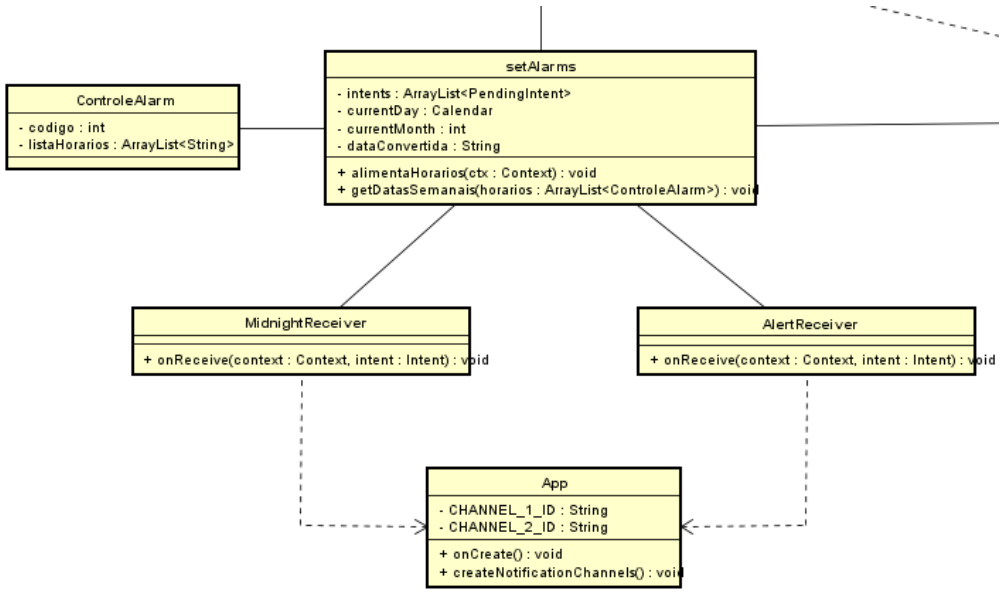
Fonte: Elaborado pelo autor.

Figura 23: Diagrama de classe parte 3



Fonte: Elaborado pelo autor.

Figura 24: Diagrama de classe parte 4



Fonte: Elaborado pelo autor.

Com o diagrama de classes apresentado, as tabelas seguintes vão descrever classe por classe, para um maior entendimento do leitor.

Tabela 7: Classe loggedMainScreen

Nome da classe	loggedMainScreen
Resumo	Tem a função de instanciar partes do programa como, menu lateral e atribuir interações.
Método	**Descrição**
onCreate	Instancia o menu e atribui o fragmento selecionado na classe TelaInicialFragmento, também alimenta a lista de notificações.
onNavigationItemSelected	Filtra o item selecionado no menu lateral e permite ir até o fragmento.
onBackPressed	Trata a ação do voltar no android.

Fonte: Elaborado pelo autor.

Tabela 8: Classe TelaInicialFragmento

Nome da classe	TelaInicialFragmento
Resumo	Lista todos os compromissos existentes sem nenhum tipo de filtro
Método	**Descrição**
onResume	Recupera todos os compromissos que existem no banco de dados
createRecords	Cria todos os compromissos na tela

Fonte: Elaborado pelo autor.

Tabela 9: Classe VisualizarMesesFragmento

Nome da classe	VisualizarMesesFragmento
Resumo	Recupera todos os compromissos do mês e marca o dia quando houver um compromisso naquele dia
Método	**Descrição**
onResume	Recupera os compromissos do banco de dados, marca os dias no calendário e modifica o evento setOnMonthChangedListener para a cada mês que mudar, a classe listar as datas do novo mês.

Fonte: Elaborado pelo autor.

Tabela 10: Classe ControleMes

Nome da classe	ControleMes
Resumo	Recupera todas as datas
Método	**Descrição**
getDatas	Recupera todas as datas diferentes do banco de dados.

Fonte: Elaborado pelo autor.

Tabela 11: Classe EventDecorator

Nome da classe	EventDecorator
Resumo	Indica o dia em que existe um compromisso
Método	**Descrição**
shouldDecorate	Define se o dia deve ou não ser pintado
Decorate	Adiciona a cor da pintura e o tamanho do indicador

Fonte: Elaborado pelo autor.

Tabela 12: Classe PastasFragment

Nome da classe	PastasFragment
Resumo	Recupera todas as pastas do banco de dados
Método	**Descrição**
onDestroy	Fecha a conexão com o banco de dados
onResume	Recupera todas as pastas do banco de dados
createRecord	Criar os registros das pastas na tela.

Fonte: Elaborado pelo autor.

Tabela 13: Classe ControlePastas

Nome da classe	ControlePastas
Resumo	Faz o controle de todas as operações do banco de dados das classes PastasFragment, AddPasta e SelecionaPasta
Método	**Descrição**
salvarDados	Salva ou altera os dados do banco de dados, de acordo com o Código recebido.
listaPastas	Recupera todas as pastas do banco de dados
excluirPasta	Exclui a pasta do banco de dados de acordo com o código passado.

Fonte: Elaborado pelo autor.

Tabela 14: Classe AddPasta

Nome da classe	AddPasta
Resumo	Uma tela flutuante para editar, criar ou excluir uma pasta.
Método	**Descrição**
salvarPasta	Passa como parâmetros os valores necessários para a criação ou alteração de uma pasta para a classe ControlePasta
onCreateDialog	Recupera todas as pastas do banco de dados
onDismiss	Verifica se a classe PastasFragment está visível, se estiver visível o aplicativo reabri o fragmento para atualizar a lista.

Fonte: Elaborado pelo autor.

Tabela 15: Classe FilterList

Nome da classe	FilterList
Resumo	Esta tela tem a função de receber parâmetros de outras classes para listar compromissos de acordo com os parâmetros passados, seja pode data, ou nome de pasta
Método	**Descrição**
onResume	Le os parâmetros recebidos e recupera os dados do banco de dados de acordo com o parâmetro recebido.
createRecords	Cria a listagem os dados na tela

Fonte: Elaborado pelo autor.

Tabela 16: Classe TelaCompromisso

Nome da classe	TelaCompromisso
Resumo	Esta classe representa o compromisso, portanto, é onde o usuário pode criar, excluir ou alterar o compromisso.
Método	**Descrição**
onCreate	Carrega todos os dados necessários para a tela e atribui valores aos atributos da classe
onCreateOptions	Este método cuida de inflar o menu que existe no topo da tela, colocando um layout nele.
editarCompromisso	Este método realiza a chamada de outros métodos para validar os dados que o usuário digitou e salvar os dados digitados.
carregaObj	Este método pega todas as entradas do usuário e carrega um objeto do tipo ControleCompromisso para a classe ControleCompromisso realizar operações com esse objeto.
carregaTela	Pega os dados de um objeto do tipo ControleCompromisso e coloca os dados do objeto na tela para o usuário.
onOptionsItemSelected	Visualiza a opção selecionada pelo usuário no menu do topo e faz ações diferentes de acordo com a opção selecionada
onSelectedCat	Este método é um método sobrescrito da interface SelectedCategory que é responsável de transmitir os dados de uma classe para outra

onSelectedTime	Este método é um método sobrescrito da interface SelectedTime que é responsável de transmitir os dados de uma classe para outra

Fonte: Elaborado pelo autor.

Tabela 17: Classe ControleCompromisso

Nome da classe	ControleCompromisso
Resumo	Esta classe é responsável pelas ações do banco de dados em tudo que se diz respeito a compromisso.
Método	**Descrição**
salvarDados	Salva ou altera os dados daquele compromisso e os horários do mesmo.
listaCompromissos	Busca todo os compromissos no banco de dados sem os horários.
getCompromisso	Responsável por buscar somente um compromisso e seus respectivos horários de acordo com o código solicitado.
deletarRegistro	Exclui o compromisso de banco de dados e todos os seus horários de acordo com o código solicitado.
listByPasta	Lista todo os compromissos de acordo com o código da pasta solicitado.
listByDataNumeral	Lista todos os compromissos de acordo com a data numeral, isto é, quando se refere a data “Única”, exemplo: 28/10/2018.
listByDataSemanal	Lista todos os compromissos de acordo

	com o dia da semana, isto é, quando se refere a data “Repetição”, exemplo: Quarta.

Fonte: Elaborado pelo autor.

Tabela 18: Classe ControleTempo

Nome da classe	ControleTempo
Resumo	Esta classe tem a função de ser instanciada como objeto para facilitar a comunicação de dados, portanto não existem métodos nela.

Fonte: Elaborado pelo autor.

Tabela 19: SelecionaPasta

Nome da classe	SelecionaPasta
Resumo	Esta classe serve como uma tela flutuante na classe TelaCompromisso, com essa tela o usuário consegue selecionar em que pasta o compromisso se encontrará.
Método	**Descrição**
onCreateDialog	Cria a tela e mostra para o usuário.
getPastas	Lista as opções de pasta existentes no banco de dados.
onAttatch	Este método serve de conversão para o atributo mCallBack.

Fonte: Elaborado pelo autor.

Tabela 20: Classe TelaConfigurarHoraData

Nome da classe	TelaConfigurarHoraData

Resumo	Esta classe serve como base para os fragmentos UnicaDataFragmento e RepeteDataFragmento
Método	**Descrição**
onCreateView	Cria um caminho de fragmento para a classe MyPagerAdapter e atribui o UnicaDataFragmento como fragmento inicial.

Fonte: Elaborado pelo autor.

Tabela 21: Classe PagerAdapter

Nome da classe	PagerAdapter
Resumo	Serve como base para a classe TelaConfigurarHoraData para a criação de tabs
Método	**Descrição**
getItem	Atribui o código da posição do item a uma classe.
getCount	Pega todas as tabs disponíveis.
getPageTitle	Atribui o nome das tabs.

Fonte: Elaborado pelo autor.

Tabela 22: Classe RepeteDataFragmento

Nome da classe	RepeteDataFragmento
Resumo	Esta classe é onde os horários de repetição, ou seja, que se repete todo dia específico da semana é atribuído.
Método	**Descrição**
onCreateView	Infla a tela para o sistema.
onViewCreated	Faz as conversões necessárias para o

	sistema, verifica se já existem valores no banco de dados deste tipo de horário.
listFields	Lista os horários já existentes.
createFields	Permite a criação de novos horários.
timerPicker	Faz a chamada do relógio para o usuário escolher um horário.
validatePrevDate	Valida todos os campos necessários e dispara mensagens se necessário.

Fonte: Elaborado pelo autor.

Tabela 23: Classe UnicaDataFragmento

Nome da classe	UnicaDataFragmento
Resumo	Esta classe é onde os horários de único, ou seja, que acontece somente em uma data específica.
Método	**Descrição**
onCreateView	Infla a tela para o sistema.
onViewCreated	Faz as conversões necessárias para o sistema, verifica se já existem valores no banco de dados deste tipo de horário.
listFields	Lista os horários já existentes.
createFields	Permite a criação de novos horários.
timerPicker	Faz a chamada do relógio para o usuário escolher um horário.
validatePrevDate	Valida todos os campos necessários e dispara mensagens se necessário.
callFragmentDate	Faz a chamada do calendário para o usuário escolher uma data.

Fonte: Elaborado pelo autor.

Tabela 24: Classe BancoHelper

Nome da classe	BancoHelper
Resumo	Classe onde se baseia toda conexão com o banco de dados.
Método	**Descrição**
onUpgrade	Este método é chamado quando se muda a versão do banco de dados e limpa a estrutura anterior.
onCreate	Cria a estrutura do banco de dados se necessário

Fonte: Elaborado pelo autor.

Tabela 25: Classe DdlScript

Nome da classe	DdlScript
Resumo	Classe abstrata onde contém toda a estrutura do banco de dados
Método	**Descrição**
createTablePastas	Cria a estrutura da tabela pasta.
createTableCompromisso	Criar a estrutura da tabela compromisso.
createTableHorario	Cria a estrutura da tabela horário.
insertSemPasta	Cria uma inserção na tabela pasta para criar o registro “sem pasta”.
createTriggerHorario	Cria trigger para excluir todos os horários do compromisso excluído.
createTriggerPastaDelete	Cria trigger para atribuir os compromissos para a pasta “sem pasta” quando uma pasta é excluída.

Fonte: Elaborado pelo autor.

Tabela 26: Classe bancoConexao

Nome da classe	bancoConexao
Resumo	Classe abstrata para obter a conexão do

	banco de dados.
Método	**Descrição**
tryConnection	Tenta abrir uma conexão com o banco de dados.

Fonte: Elaborado pelo autor.

Tabela 27: Classe setAlarms

Nome da classe	setAlarms
Resumo	Tem a função de criar as notificações dos compromissos em seus devidos horários e datas.
Método	**Descrição**
alimentaHorarios	Recupera todos os horários únicos do banco de dados e cria as notificações.
getDatasSemanais	Recupera todos os horários repetição do banco de dados e devolve para a função alimentaHorarios.

Fonte: Elaborado pelo autor.

Tabela 28: Classe ControleAlarm

Nome da classe	ControleAlarm
Resumo	Tem a função de funcionar como instancia de objeto, não contendo métodos.

Fonte: Elaborado pelo autor.

Tabela 29: Classe MidnightReceiver

Nome da classe	MidnightReceiver
Resumo	Esta classe é responsável por receber os alertas do sistema referente a classe setAlarms, neste caso o horário é atribuido as 23:59.

Método	Descrição
onReceive	Dispara a notificação de acordo com o horário especificado e atribui o conteúdo a notificação.

Fonte: Elaborado pelo autor.

Tabela 30: Classe AlertReceiver

Nome da classe	AlertReceiver
Resumo	Esta classe é responsável por receber os alertas do sistema referente a classe setAlarms.
Método	**Descrição**
onReceive	Dispara a notificação de acordo com o horário especificado e atribui o conteúdo a notificação.

Fonte: Elaborado pelo autor.

Tabela 31: Classe App

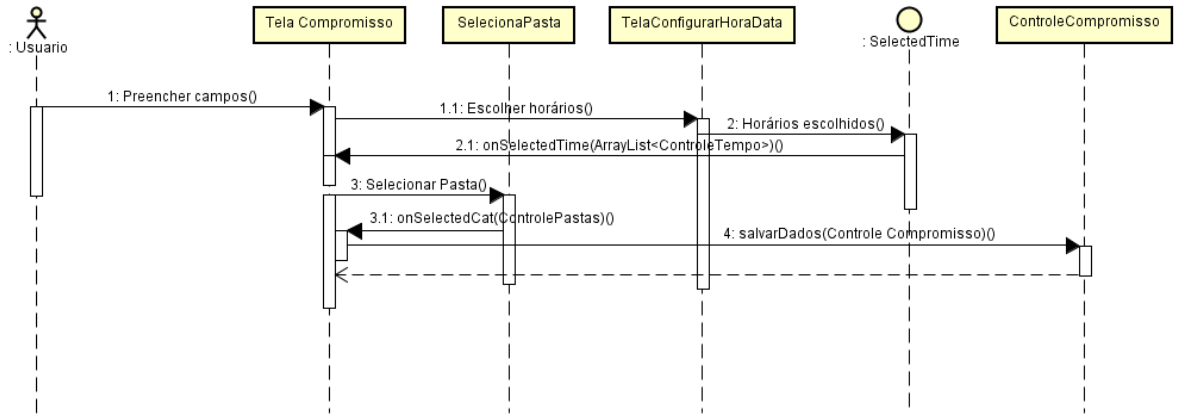
Nome da classe	App
Resumo	Esta classe é responsável criar um canal de comunicação para a notificação ser disparada.
Método	**Descrição**
onCreate	Realiza a chamada da função createNotificationsChannel.
createNotificationsChannel	Cria os canais de notificação.

Fonte: Elaborado pelo autor.

3.4 Diagrama de sequência

O diagrama de sequência tem como objetivo demonstrar o funcionamento do sistema seguindo passos, mostrando como as classes interagem e como o usuário interage com elas. Na figura 25 podemos ver como as classes se comportam quando o usuário deseja criar ou alterar um novo compromisso.

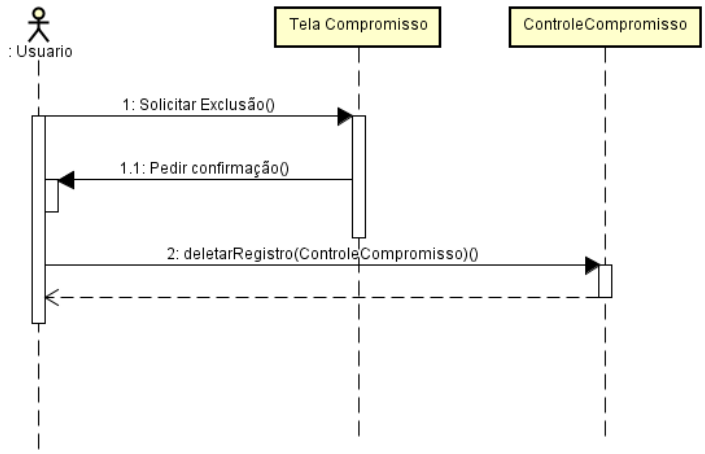
Figura 25: Diagrama de sequência - criar ou alterar compromisso



Fonte: Elaborado pelo autor.

O usuário preenche todos os campos necessários, escolhe os horários para o lembrete se necessário, seleciona uma pasta para aquele compromisso e salva. O mesmo processo funciona para a operação “alterar”.

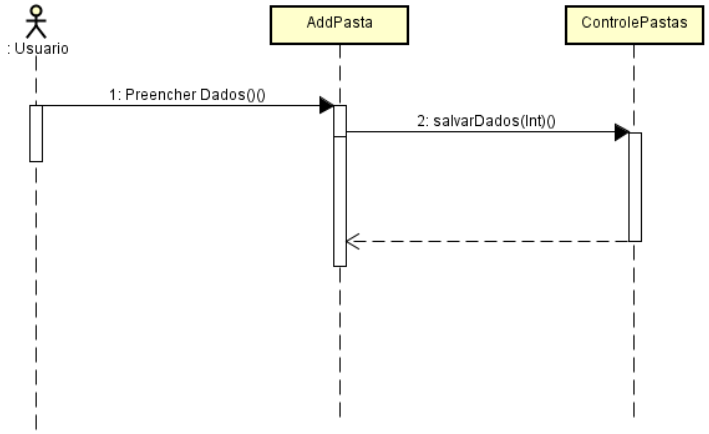
Figura 26: Diagrama de sequência - excluir compromisso



Fonte: Elaborado pelo autor.

No diagrama acima podemos ver como as classes se comportam na exclusão de um compromisso. O usuário solicita a exclusão e então a classe retorna uma mensagem de confirmação, se a mensagem retornar positiva, a exclusão é feita, caso contrário nada acontece.

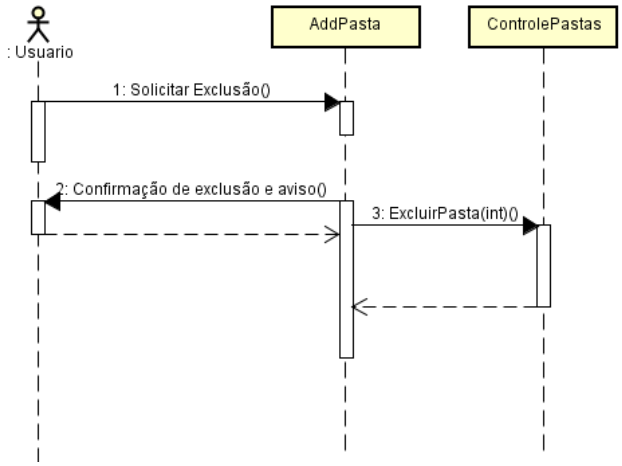
Figura 27: Diagrama de sequência - criar ou alterar pasta



Fonte: Elaborado pelo autor.

O processo de criação e alteração de uma pasta é muito simples como podemos visualizar no diagrama acima, nos próximos capítulos a apresentação das telas deixará ainda mais claro como o processo de excluir, alterar ou criar uma nova pasta é simples.

Figura 28: Diagrama de sequência - exclusão de pasta



Fonte: Elaborado pelo autor.

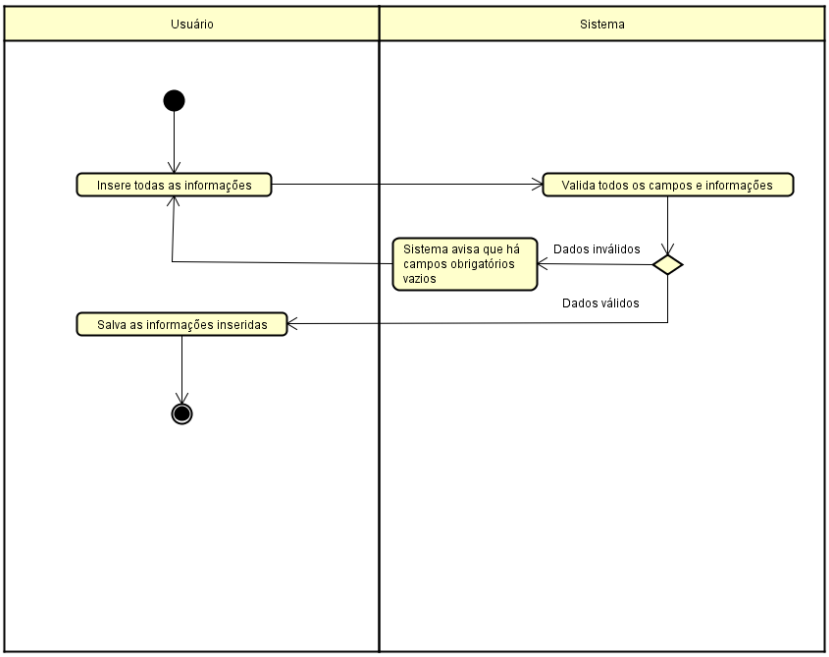
Como podemos perceber, todo processo de exclusão exige uma confirmação. Isto acontece pois, às vezes, o usuário pode ter cometido um erro, sendo assim, a confirmação sempre é necessária.

3.5 Diagrama de atividade

Depois de estabelecido os diagramas de sequência, classe, MER, DER e o diagrama de uso, temos o diagrama de atividades, que por último, mas não menos importante, demonstra a atividade realizada pelo usuário ao efetuar alguma operação no aplicativo.

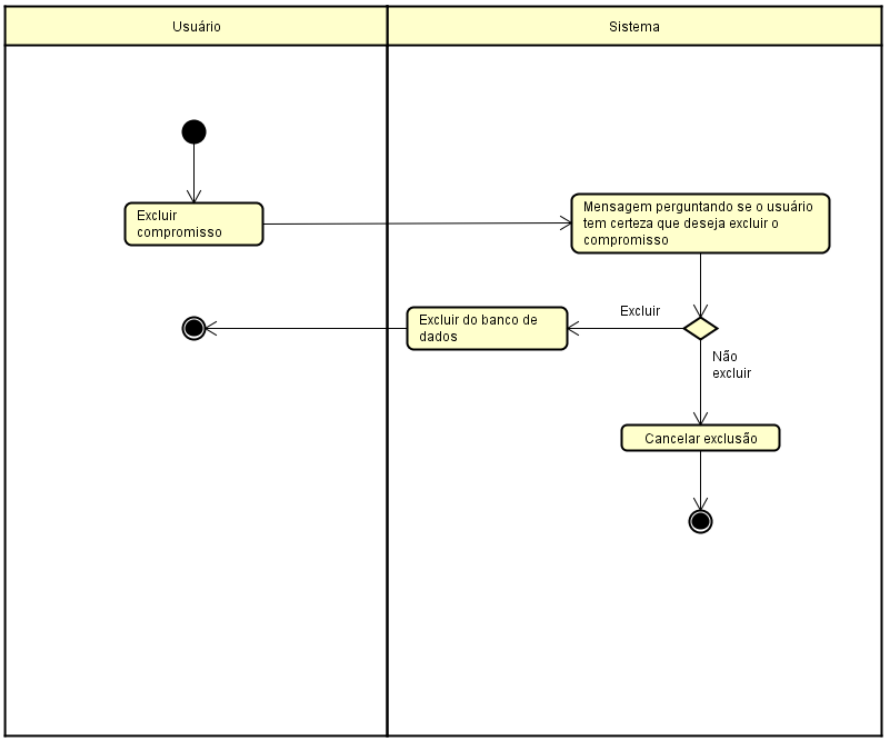
A figura 29 demonstra o que acontece quando o usuário quer criar ou alterar um compromisso e como o que o sistema faz para que isso possa acontecer.

Figura 29: Diagrama de atividade - criar ou alterar compromisso



Fonte: Elaborado pelo autor.

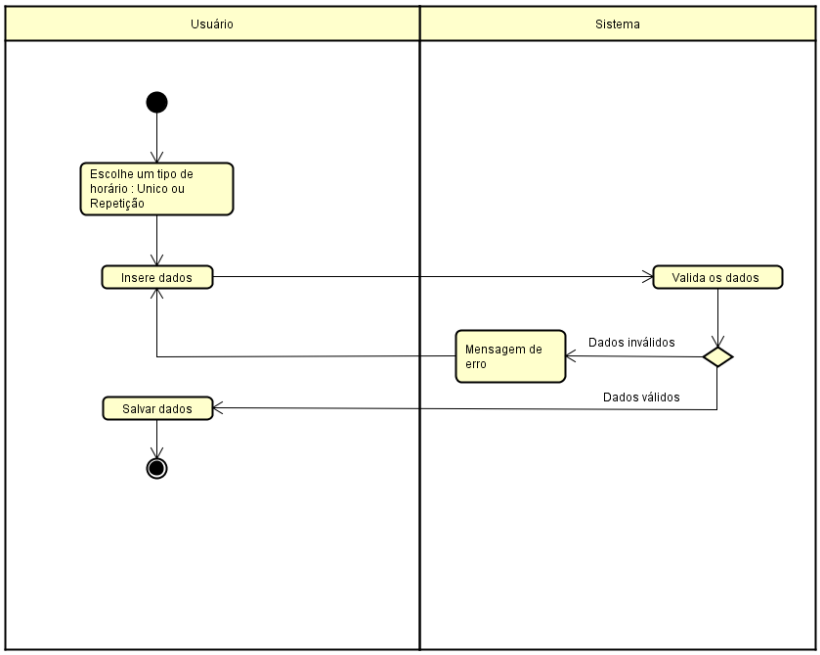
Figura 30: Diagrama de atividade - excluir compromisso



Fonte: Elaborado pelo autor.

Nas figuras abaixo veremos como funciona a inserção, alteração e exclusão de horários.

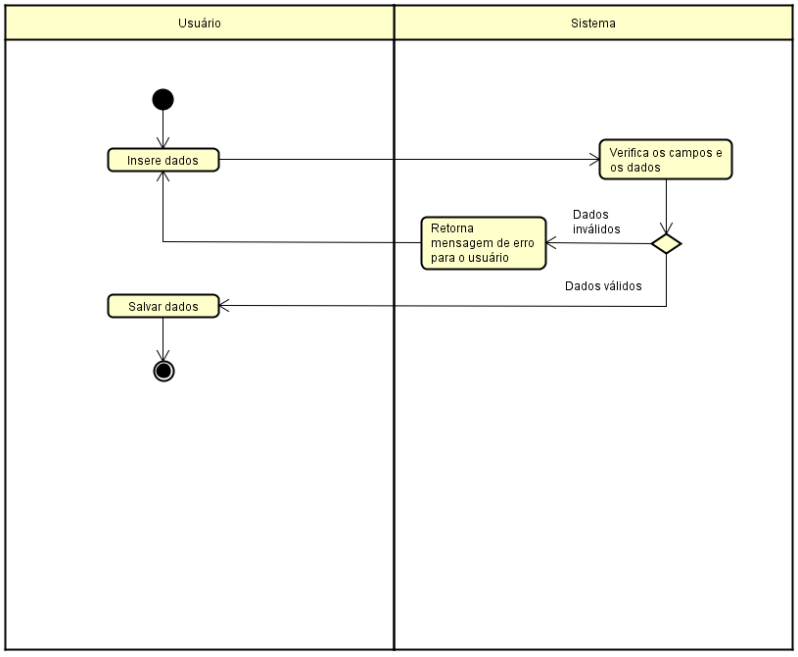
Figura 31: Diagrama de atividade - criar ou alterar horário



Fonte: Elaborado pelo autor.

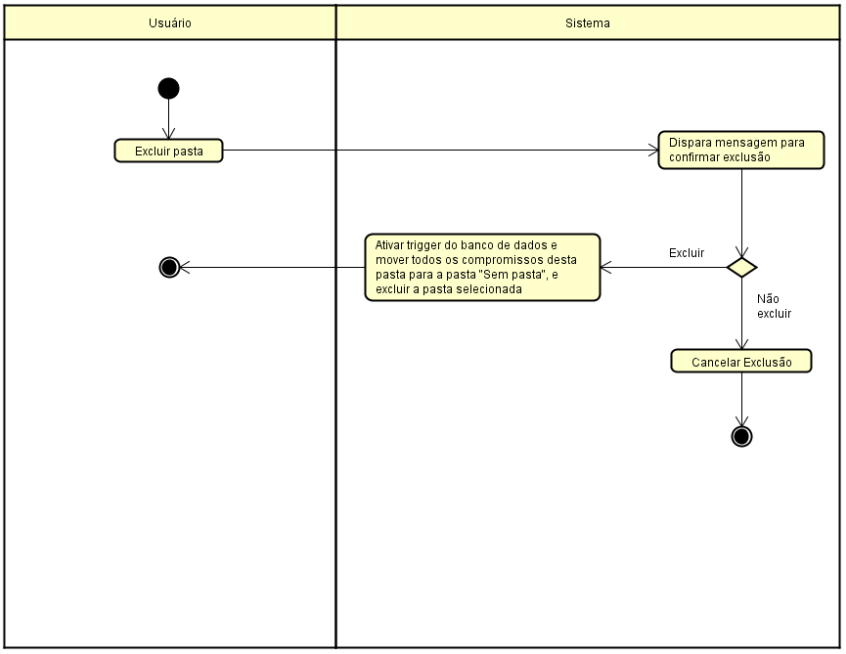
O controle de horário não foi citado em outros momentos pois ele não é salvo no banco de dados, mas sim, controlado por vetores e objetos, sendo assim os valores são transmitidos por objetos através das classes.

Figura 32: Diagrama de atividade - criar ou alterar pasta



Fonte: Elaborado pelo autor.

Figura 33: Diagrama de atividade - excluir pasta

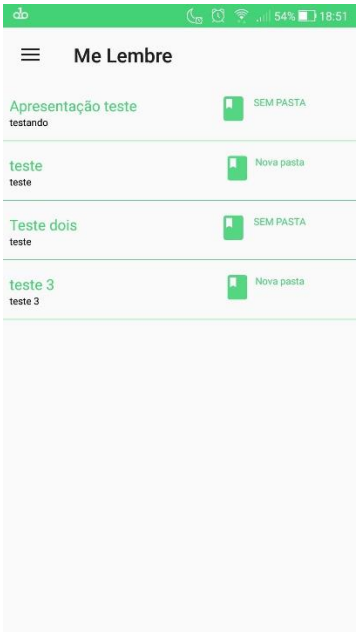


Fonte: Elaborado pelo autor.

3.6 Apresentação das telas do aplicativo

Neste capítulo será abordado todas as telas referentes ao aplicativo e, logo abaixo, uma breve explicação sobre quando necessário.

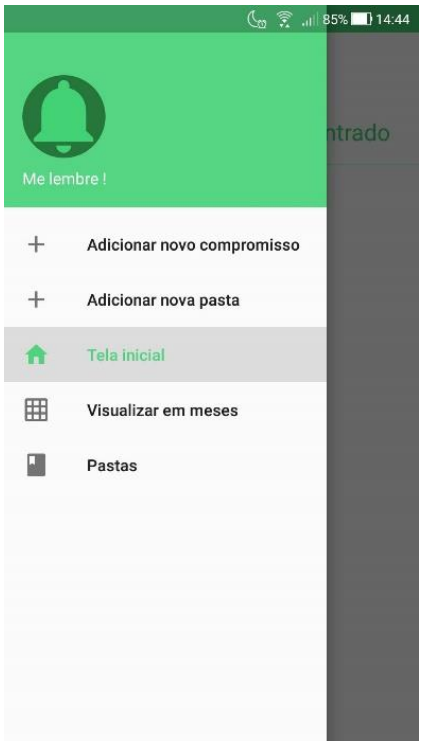
Figura 34: Tela inicial



Fonte: Elaborado pelo autor.

Na Tela Inicial, figura 34, são listados todos os compromissos já criados pelo usuário.

Figura 35: Menu inicial



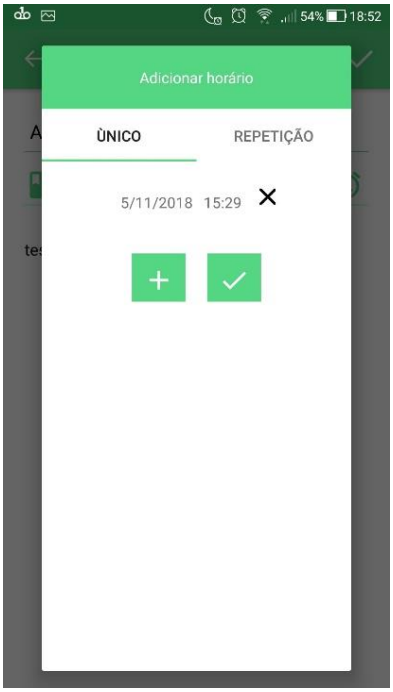
Fonte: Elaborado pelo autor.

No menu lateral, podemos ver a opção “Adicionar novo compromisso”, quando o usuário escolhe essa opção, ele consegue criar um novo compromisso, como podemos ver na figura 36.

Figura 36: Criar compromisso

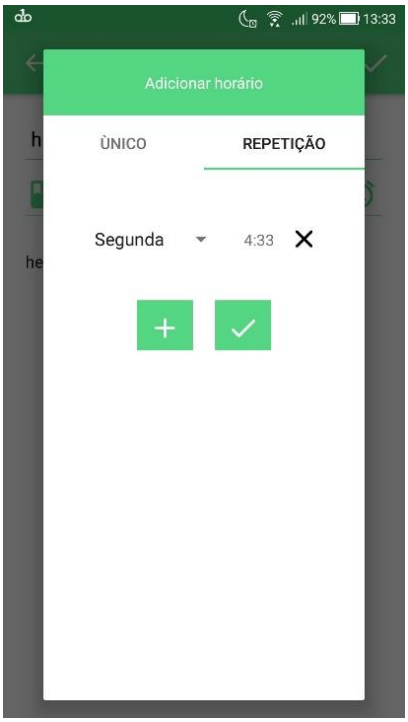
Fonte: Elaborado pelo autor.

Figura 37: Compromisso - selecionar data e horário



Fonte: Elaborado pelo autor.

Figura 38: Compromisso - selecionar data e horário 2



Fonte: Elaborado pelo autor.

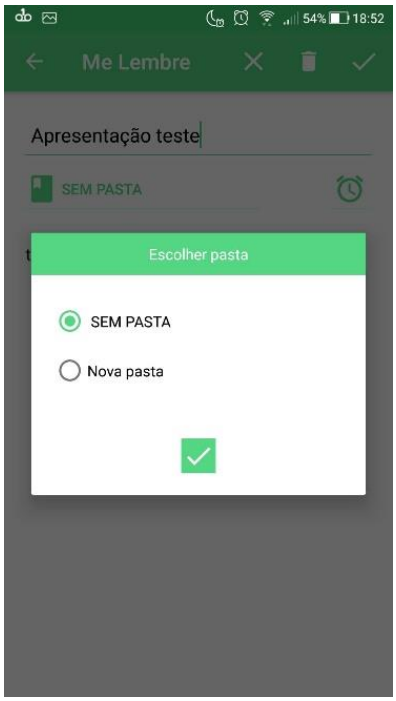
Figura 39: Erro ao selecionar dois horários

Fonte: Elaborado pelo autor.

Nas figuras acima, visualiza-se a criação de um novo compromisso. Na figura 39, ocorre um erro que é disparado quando o usuário cria uma data e um horário em um tipo, e tenta criar uma data e horário em outro tipo diferente.

Na figura 39, havia uma data e horário criada no tipo “Único”, quando o usuário tenta criar uma data e horário no tipo de horário “Repetição” o sistema dispara essas mensagem, o mesmo funciona para a situação inversa.

Figura 40: Selecionar pasta



Fonte: Elaborado pelo autor.

Para salvar um compromisso, basta que o usuário clique no ícone do lado superior direito da tela compromisso.

Na tela inicial podemos visualizar todos os compromissos, para o usuário ter maiores detalhes, alterar ou excluir o compromisso basta ele clicar em um compromisso e a aplicação o levará para uma tela como podemos ver na figura 41.

Figura 41: Alterar ou excluir compromisso

Fonte: Elaborado pelo autor.

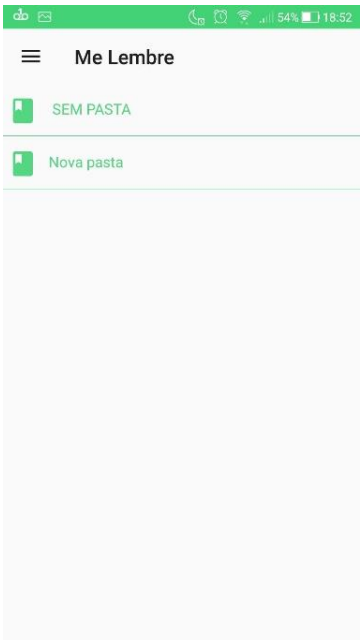
O processo de alteração é muito simples, quando o usuário clica em um compromisso ele é levado à esta tela com todas as informações daquele compromisso solicitado, sendo assim ele pode alterar o que desejar e clicar em salvar novamente.

Se o usuário deseja excluir o compromisso, basta ele clicar no ícone de lixeira na barra superior e confirmar a exclusão como mostra a figura 42.

Figura 42: Excluir compromisso

Fonte: Elaborado pelo autor.

Figura 43: Listar pastas



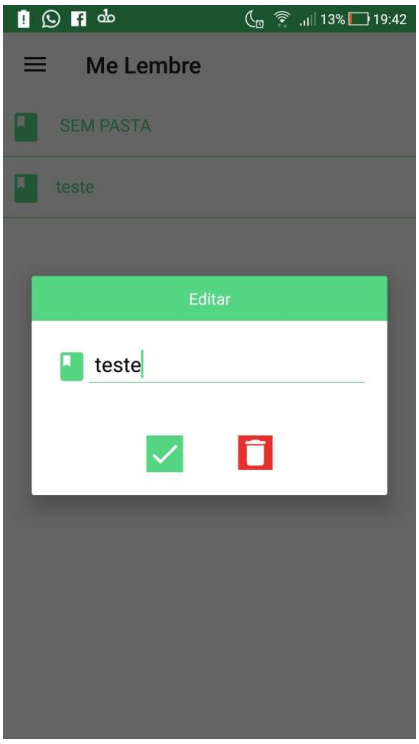
Fonte: Elabora pelo autor.

Na figura 43 podemos visualizar a listagem de pastas, no menu lateral, quando o usuário seleciona a opção “Adicionar nova pasta” podemos criar uma nova pasta como na figura 44.

Figura 44: Criar nova pasta

Fonte: Elaborado pelo autor.

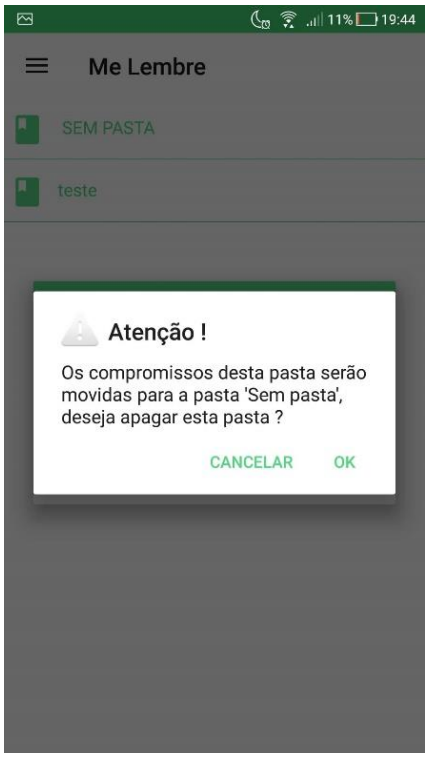
Figura 45: Editar ou excluir pasta



Fonte: Elaborado pelo autor.

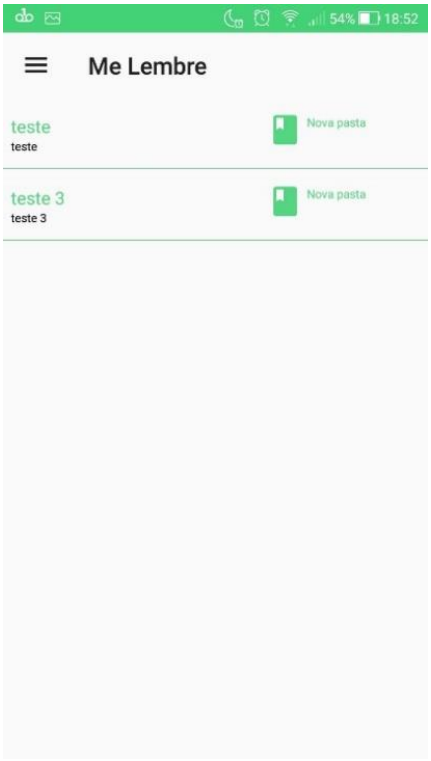
Na figura 45, ao contrário do compromisso, quando o usuário dá somente um clique na pasta, o usuário é levado a uma tela onde são mostrados todos os compromissos daquela pasta, como podemos ver na figura 47, porém, quando ele clica e segura por alguns segundos, como na figura 45, aparece um tela flutuante que permite o usuário alterar ou excluir a pasta.

Figura 46: Excluir pastas



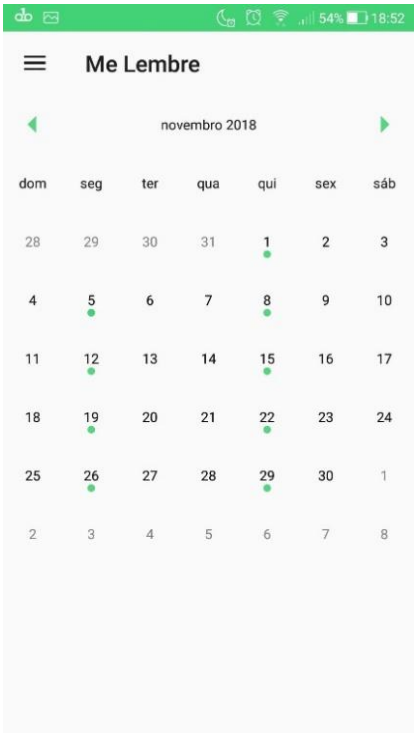
Fonte: Elaborado pelo autor.

Figura 47: Filtro por pasta



Fonte: Elaborado pelo autor.

Figura 48: Visualizar em meses

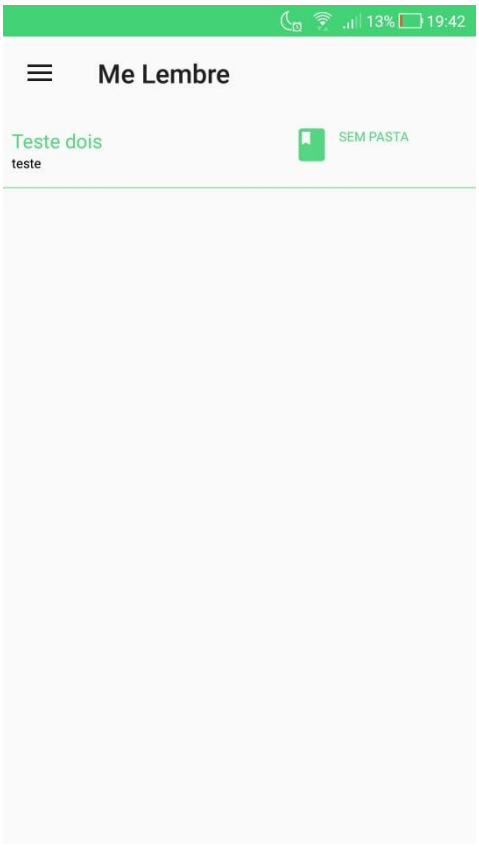


Fonte: Elaborado pelo autor.

Quando o usuário seleciona a opção “Visualizar em meses” o aplicativo abre uma tela com um calendário, onde são marcados todos os dias em que o usuário tem um compromisso que irá acontecer no mês em que ele está visualizando.

Para saber quais compromissos irão acontecer naquele dia, basta o usuário dar um clique simples no dia e o aplicativo irá abrir uma tela com os compromissos daquele dia para o usuário, como na figura 49.

Figura 49: Filtro por dia



Fonte: Elaborado pelo autor.

4. Considerações finais

Este trabalho teve como objetivo final atender a uma necessidade pessoal de uma maneira simples e eficaz, desenvolvendo uma aplicação para a plataforma Android, que atinge uma parcela maior da população brasileira. O objetivo do aplicativo desde o planejamento foi possuir uma interface muito limpa e de fácil uso. Após a conclusão do desenvolvimento pode-se notar que esses objetivos foram alcançados, já que a interface é muito comum para os usuários de celulares e o aplicativo é mais engessado em questões de configurações, permitindo que o usuário se preocupe apenas com os compromissos que ele deve realizar.

O desenvolvimento foi a parte mais crucial deste projeto, tendo em vista que foi utilizado uma linguagem diferente das habituais, o Kotlin, que por trás, quando compilado, ele gera um código em Java para o interpretador conseguir realizar as operações necessárias, ele tem o objetivo de facilitar a escrita do Java, já que o mesmo é muito verboso. Olhando essa janela de oportunidade de aprender algo novo foi escolhido esta linguagem para ser trabalhada nesse projeto, buscando também se atualizar e se preparar melhor para o mercado de trabalho quando o assunto é programação para dispositivos móveis, o desafio foi grande porem os resultados foram os esperados.

Como possíveis trabalhos futuros, melhorias devem ser feitas tanto na documentação quanto na parte prática do projeto, sendo sugerido:

- Criar um lembrete para todos os dias em horários diferentes;
- Implementar sistema de login;
- Compartilhar compromissos com outras pessoas;
- Deixar a listagem de compromissos mais clara;
- Desenvolver sistema online utilizando a mesma base de dados para as duas aplicações;

REFERÊNCIAS

BOOCH G.; RUMBAUGH J.; JACOBSON I; **UML**: Guia do usuário. 2ed. Ver. E ampli.; Tradução de Fábio Freitas da Silva e Cristina de Amorim Machado; Rio de Janeiro: Elsevier, 2006, p. 2–16.

CORDEIRO, Filipe. **Android SDK: O que é? Para que serve? Como usar?,** 2018. Disponível em: <https://www.androidpro.com.br/blog/android-studio/android-sdk/>. Acesso em: 17 de Nov de 2018.

DEVMEDIA. **Série: eu sobrevivo sem UML?** Disponível em: <https://www.devmedia.com.br/uml/ >. Acesso em: 05 de Jun de 2018.

FERNANDES, André. **Oque é API? Entenda de uma maneira Simples,** 2018. Disponível em:<http://blog.vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples/>. Acesso em: 13 de Jun de 2018.

GASPAROTTO, Machado. **SQL: Aprenda a utilizar a chave primária e a chave estrangeira,** 2017. Disponível em: <https://www.devmedia.com.br/sql-aprenda-a-utilizar-a-chave-primaria-e-a-chave-estrangeira/37636>. Acesso em: 10 de Jun de 2018.

GUEDES, Araújo. **UML2**: Uma abordagem prática. São Paulo: Novatec, 2010, p.19 - 24.

LIGHT, Richard. **Iniciando em XML**. Tradução de Neilande de Moraes; MAKRON Books, 1997, p. 1–3.

MEDEIROS, Higor. **Introdução a requisitos de software,** 2013. Disponível em: <https://www.devmedia.com.br/introducao-a-requisitos-de-software/29580>. Acesso em: 08 de Jun de 2018.

MELLO R.; CHIARA R.; VILLELA R.; **Aprendendo Java 2**. São Paulo: Novatec, 2002, p.14-15.

MELLO, Silva. **Levantamento de requisitos.** 2010, p. 1-3 Disponível em: <http://www.ice.edu.br/TNX/encontrocomputacao/artigos-internos/aluno_leandro_cicero_levantamento_de_requisitos.pdf >. Acesso em: 07 de Jun de 2018.

MITRUT, Wellington. **Como kotlin se tornou a nossa linguagem principal para Android,** 2017. Disponível em: <https://medium.com/blog-do-mitrut/como-kotlin-se-tornou-a-nossa-linguagem-principal-para-android-24c9492fa273>. Acesso em: 17 de Nov de 2018.

NOVAES, Rafael. **O que é e para que serve IDE?,** 2014. Disponível em: < https://www.psafe.com/blog/o-que-serve-ide/>. Acesso em: 17 de Nov de 2018.

PANDEY, Manoj. **What are the differences between DDL and DML,** 2017. Disponível em: <https://www.quora.com/What-are-the-differences-between-DDL-and-DML/>. Acesso em: 08 de Jun de 2018.

RIBEIRO, Leandro. **O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML, 2012.** Disponível em: <https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>. Acesso em: 05 de Jun de 2018.

RICARTE, Marques. **Oque é uma classe,** 2000. Disponível em: <http://www.dca.fee.unicamp.br/cursos/PooJava/classes/conceito.html>. Acesso em: 08 de Jun de 2018.

RODRIGUES, Joel. **Modelo Entidade Relacionamento (MER) e Diagrama Entidade-Relacionamento (DER),** 2014. Disponível em: <https://www.devmedia.com.br/modelo-entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332>. Acesso em: 09 de Jun de 2018.

SOUZA, Pires; GASPAROTTO, Segoria. **A importância da atividade de teste no desenvolvimento de software,** in.: XXXIII ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 11 out 2013, Salvador/ BA, p.1–7, disponível em: <

http://www.abepro.org.br/biblioteca/enegep2013_TN_STO_177_007_23030.pdf >. Acesso em: 13 de jun de 2018.

TONSIG, Luiz. **MYSQL:** Aprendendo na prática; Rio de Janeiro: Ciência Moderna, 2006, p. 18.

TYBEL, Douglas. **Orientações básicas na orientação de um diagrama de classes,** 2016. Disponível em: <https://www.devmedia.com.br/orientacoes-basicas-na-elaboracao-de-um-diagrama-de-classes/37224>. Acesso em: 06 de Jun de 2018.

VENTURA, Plínio. **Entendendo o diagrama de atividades da UML,** 2016. Disponível em: <http://www.ateomomento.com.br/uml-diagrama-de-atividades/>. Acesso em: 08 de Jun de 2018.

VIEGAS, Gustavo. **Introdução a UML,** 2009. Disponível em: <https://www.devmedia.com.br/introducao-a-uml/6928>. Acesso em: 08 de Jun de 2018.

WIKIBOOKS. **Programação orientada a objetos/Classes e objetos,** 2017. Disponível em: <https://pt.wikibooks.org/wiki/Programa%C3%A7%C3%A3o_Orientada_a_Objetos/Classes_e_Objetos>. Acesso em: 08 de Jun de 2018.