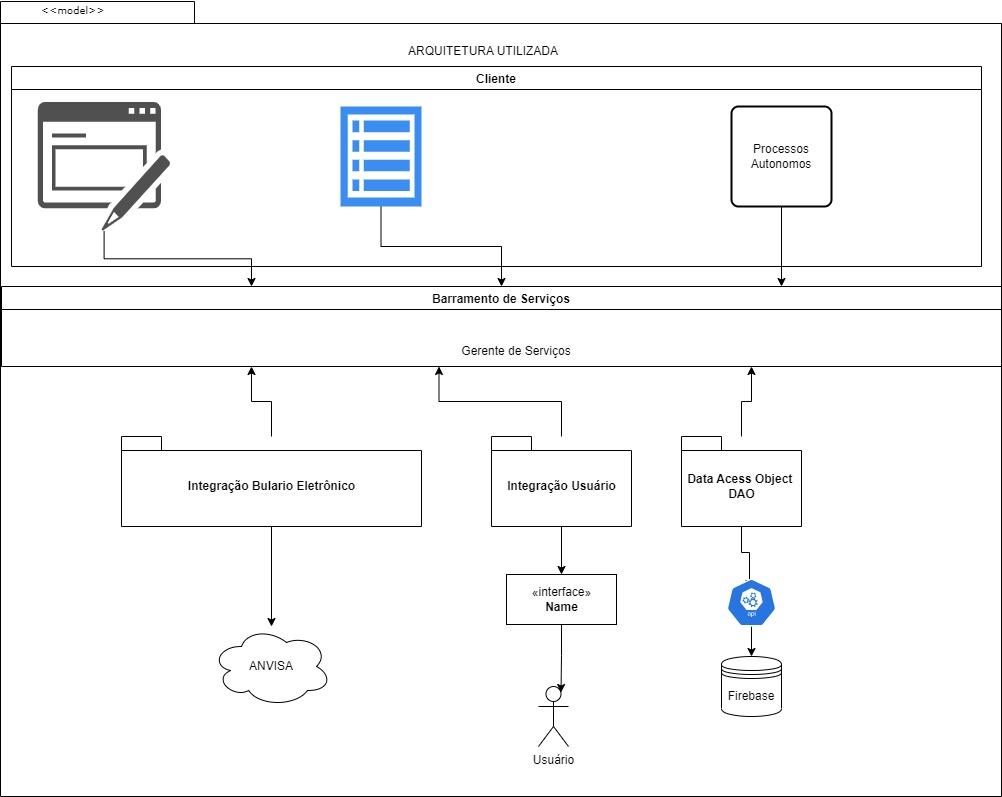
DESENVOLVIMENTO

O aplicativo AMU, produto final do trabalho corrente, destina-se a auxiliar a utilização de medicamentos por grupos de pessoas, tais como: idosos e portadores de deficiência. Para atender a esse público, a concepção do AMU deverá incorporar soluções que interajam fácil e adequadamente com esse grupo de pessoas. Para tanto, o projeto e construção do aplicativo AMU envolveu a utilização de conceitos e estratégias de engenharia de software, as quais serão detalhadas a partir da Figura 1.

Figura 1 - Visão geral da concepção do aplicativo AMU.



(Fonte: o Autor)

Através da Figura 1 pode-se observar os principais elementos do AMU, ou seja:

* Cliente
* Barramento de Serviços
* Interface de Bulário Eletrônico
* Interface de Integração de Usuário
* Interface de Acesso ao Banco de Dados

Esses elementos são descritos da forma como segue:

1. **Cliente**

Objetivamente falando é qualquer interface ou recurso utilizado pelo usuário para consumir ou solicitar um serviço do aplicativo. Pode ser um formulário ou uma lista, bem como um serviço autônomo monitorando horários. A seguir alguns tipos de tipos de clientes relacionados com o AMU.

A Figura 2 representa janelas de entrada de dados no AMU, como: cadastro de usuário/cadastro de medicamento ou cadastro de perfil.

Figura 2-Entrada de Dados



Fonte: Android Studio

As janelas com listas de dados no AMU, como: a lista de medicamentos, /lista de horários, /lista de pesquisa da ANVISA, são representadas pela Figura 3.

Figura 3-Listas de Pesquisa



Fonte: Android Studio

Na Figura 4 estão representadas as rotinas no AMU onde não existem interações do usuário, como por exemplo: a rotina que verifica um alarme.

Figura 4-Processos Autônomos



Fonte: Aplicativo diagrams.net

1. **Barramento de Serviços**

A indicação do barramento de serviços indicada na Figura 1, objetiva criar uma interface de acesso a todos os recursos do aplicativo. Para essa tarefa, utilizamos um padrão de projeto conhecido como “Facade”. O padrão “Facade” (pronuncia-se "façade” por ser uma palavra de origem francesa) propõe a criação de uma classe intermediária que serve como uma fachada para que o cliente possa acessar as funcionalidades desejadas. Essa classe encapsula a complexidade da interação entre os diversos componentes e desacopla o cliente das implementações. (1). A Figura 5 mostra um barramento de serviços padrão.

Figura 5-Representa o barramento de serviços



A representação da interface de barramento de serviços padrão entre as requisições vinda dos clientes e os serviços disponíveis é apresentada na Figura 5. As setas indicam o fluxo das mensagens de envio / resposta.

1. **Interface de Integração Bulário Eletrônico**

**Sempre referenciando-se à Figura 1, a interface de integração bulário eletrônico é o** componente que possibilita criar uma ponte entre o serviço e as consultas às fontes de dados de medicamentos. No presente trabalho, utilizamos um padrão de projeto chamado “Strategy” com a finalidade de abstrair a fonte de dados que utilizamos. A ideia básica é impedir que o aplicativo conheça os detalhes da consulta de medicamento ao ponto de, em uma situação de trocar de fonte de dados, tal mudança não gerar nenhum tipo de problema para o restante do projeto (2). A Figura 6 representa graficamente a interface bulário eletrônico.

Figura 6



Fonte:

A Figura 6 exemplifica graficamente o componente container de integração entre qualquer fonte de dados de bula e o barramento de serviços. Adicionalmente, esse componete mostra o isolamento do acesso e permite a reutilização dos diversos recursos de busca da lista de medicamentos, tais como: download de arquivo e conversão de pdf para o formato texto. Já o armazenamento de uma fonte de dados de bulas presente na Figura 1, é mostrado na Figura 7 onde, a nuvem que circunda a palavra ANVISA, indica que está acessível pela internet.

Figura 7



Fonte:

Banco de dados Firebase no formato No-Sql.

1. **Interface de Integração de Usuário**

Seguindo o mesmo padrão “Strategy”, abordado no tópico “Integração de Bulário Eletrônico", é necessário criar recursos específicos para os tipos de usuários. Por exemplo, termos uma interface para pessoas com visão reduzida ou mesmo sem visão. Dessa forma, formatamos uma lista das interfaces da seguinte forma::

1. Usuário comum
2. Usuário PCD
   1. Usuário com perda de visão ou reduzida
   2. Usuário com perda auditiva
   3. Usuário com TEA
3. Usuário Idoso

Utilizando a interface de integração de usuário podemos, por exemplo, avisar que está na hora de um medicamento abstraindo a forma que será a saída do aviso. Para alguém com perda de visão o aviso será sonoro. No entanto, para alguém com perda auditiva utilizamos a vibração do celular e notificações. Essa ação é representada pela Figura 8. Esta figura representa o componente container de Integração entre qualquer perfil de usuário e o barramento de serviços. O acesso ao container de usuário deve ser feito através de uma interface onde a implementação para cada perfil abre possibilidade da aplicação atender aos mais diversos tipos de perfis de usuários.

Figura 8



Fonte :

.

Figura 9



Fonte:

Representa Interface utilizada para acessar a integração de usuário.

Figura 1. i



Representa um ator qualquer relacionado ao perfil de usuário. Por exemplo, podemos ter usuário com visão reduzida ou um usuário comum.

1. **Interface de Acesso a Dados**

O componente de acesso dados além de abstrair o uso da API do firebase possibilita que o usuário tenha acesso em qualquer dispositivo instalado em virtude do Banco de dados estar remoto. Utilizamos o padrão de projeto “DAO”. Esse padrão possibilita a troca do banco de dados. Apenas trocando a fábrica da implementação.

Figura 1.j

Representa a camada de acesso aos dados. O AMU para cada Nó que representa uma tabela no banco de dados tem algumas classes fazendo o mapeamento ORM (Object Relational Management). Como exemplo: O nó medicamentos tem: MedicamentoDAO, IMedicamentoDAO. A primeira é a implementação e a segunda a interface.

Figura 1. e



1. **Problema analisado no trabalho**

No presente trabalho analisa-se o uso de medicamentos por pessoas com perfil específicos como idosos e pcds

1. **Aplicativo**

**Primeiro Acesso**

No primeiro acesso do aplicativo alguns slides de boas-vindas são apresentados. Então é apresentado ao usuário criar um cadastro novo ou efetuar o login informando e-mail e senha. Após efetuar o cadastro o aplicativo exibirá a janela principal. Se o usuário possuir um cadastro feito anteriormente escolherá a opção efetuar login.

Tanto no primeiro caso quanto no segundo caso o aplicativo exibirá a janela principal.

Quando o usuário já tiver feito login pelo menos uma vez, o aplicativo utilizará o recurso Firebase Auth do serviço de banco de dados Firebase para autenticar o usuário.

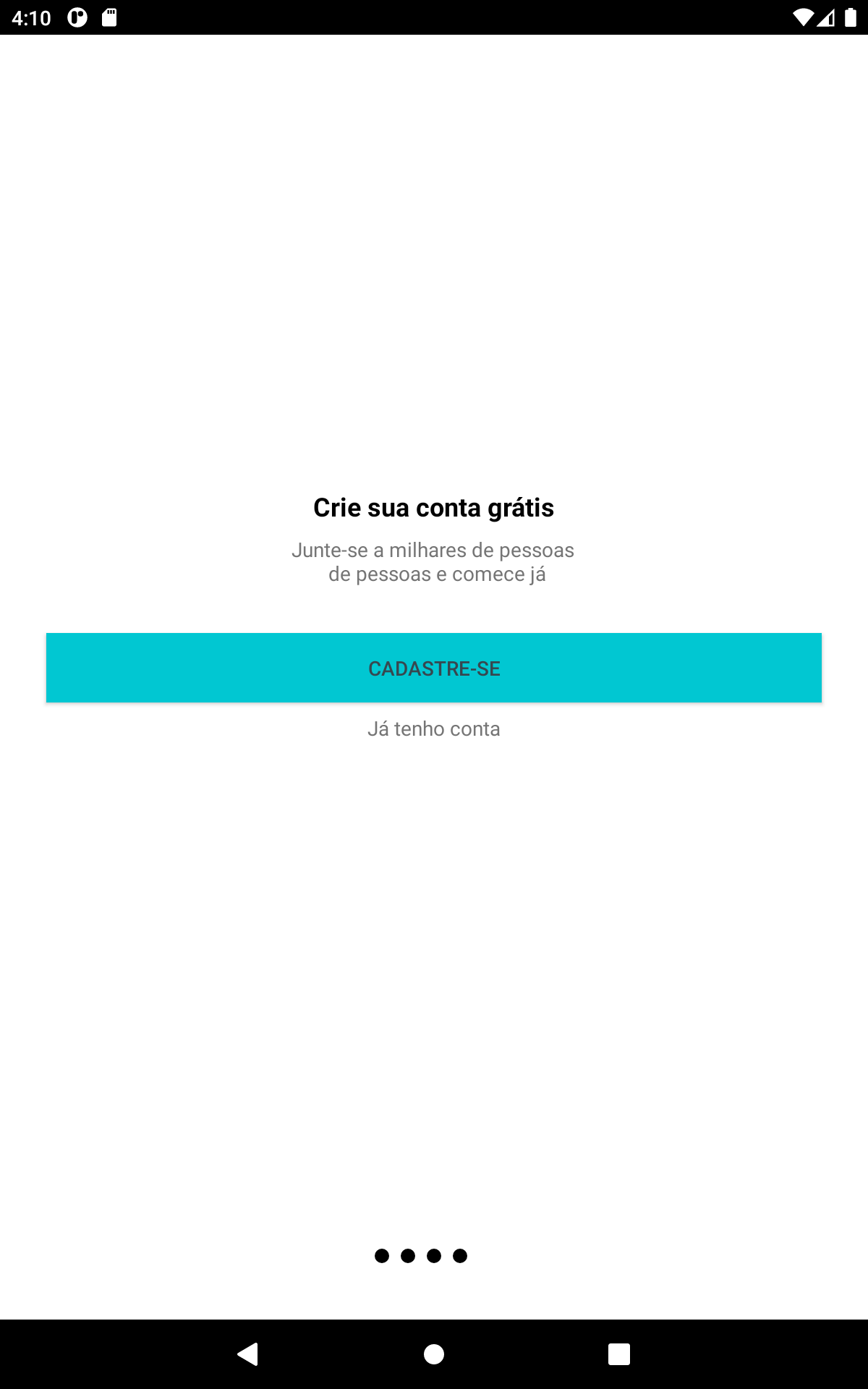
O objetivo do Firebase Authentication é facilitar o desenvolvimento de um sistema de autenticação seguro, além de melhorar a experiência de login e integração para os usuários finais. Ele oferece uma solução de identidade completa, compatível com contas de e-mail/senha, autenticação por telefone, login do Google, Twitter, Facebook, GitHub. O login por e-mail e senha foi escolhido por apresentar simplicidade e os e-mails serem únicos em qualquer servidor de e-mail. (Referência 3)

Dito isto, sempre que o usuário acessar o aplicativo fará login automático apresentando facilidade de uso desde a tela inicial.

Boas vindas

Autenticação de Usuário

Figura 2



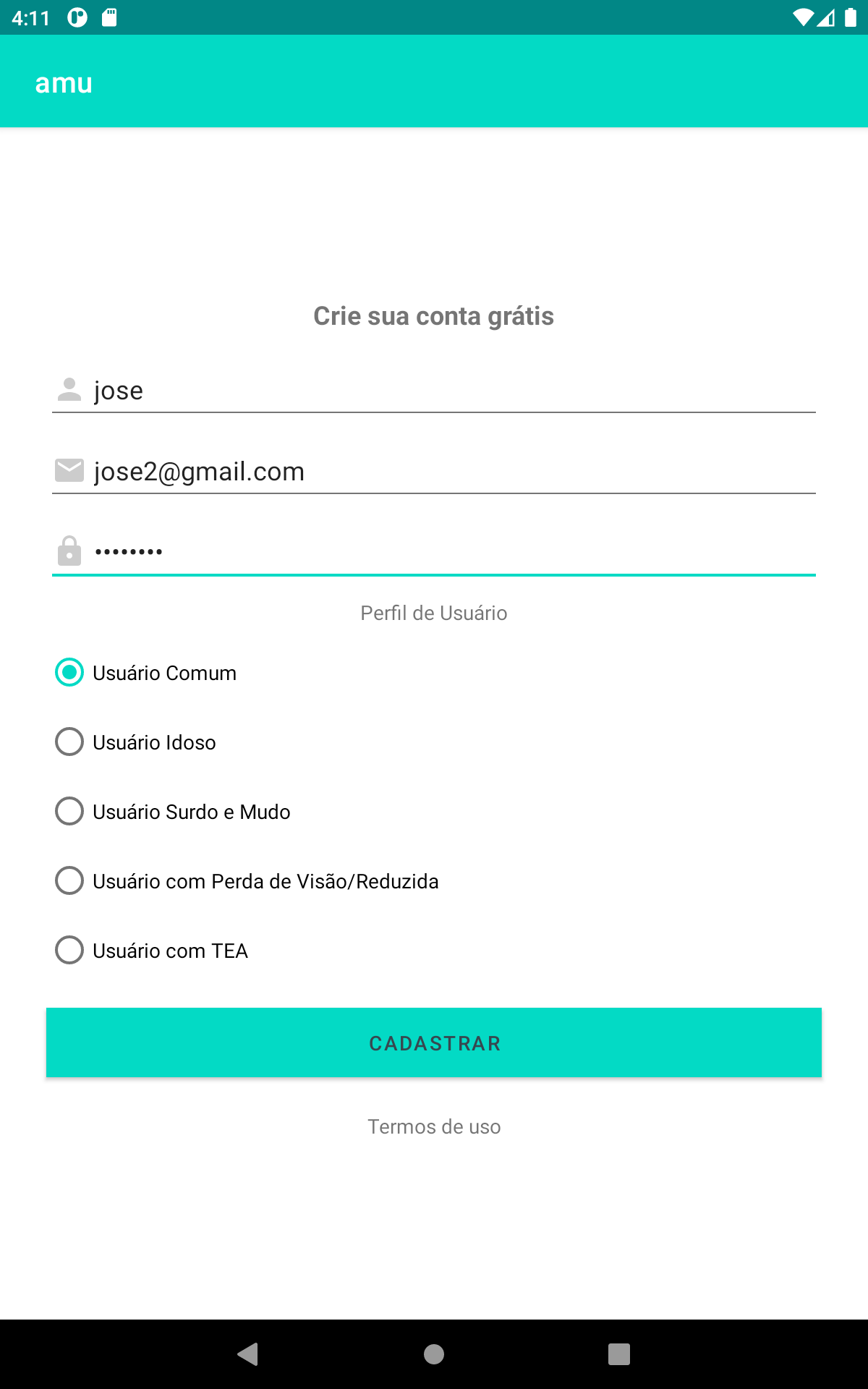
(Fonte: Autor)

Figura 3



(Fonte: Autor)

Figura 4

**Cadastro de Usuário**

(Fonte: Autor)

Nesta primeira versão do aplicativo AMU o usuário comum poderá utilizar todas as funcionalidades. O usuário com perda de visão ou reduzida terá através do uso da voz acesso a algumas funcionalidades.

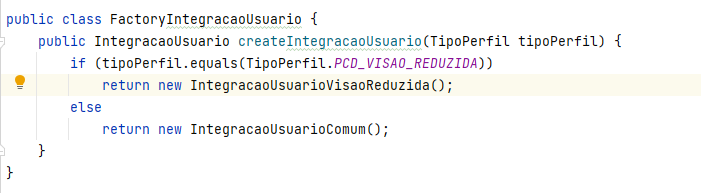
Para usuários idosos, surdos/mudos e TEA não tem funcionalidades adaptadas.

O perfil é o elemento chave na comunicação entre as camadas do aplicativo. O Aplicativo determina através do perfil a forma como os comandos devem ser respondidos, para sustentar esse recurso, utilizamos padrões de Projetos. O principal padrão de projeto adotado foi o Strategy.

Quando o aplicativo inicia pelo cadastro inicial sabe qual o perfil do usuário. Uma variável global com o nome "integração usuário" então é inicializada por uma fábrica de objetos. Para exemplificar o uso desse recurso, quando o aplicativo dispara um alarme indicando que um medicamento deve ser tomado é feito em duas etapas: através de notificação e por áudio. Isto é necessário para atender quem não pode ler uma notificação. Então o aplicativo fará uma chamada para "integração usuário" que dispara um alarme em áudio. A mesma chamada existe para o usuário comum, mas como não tem utilidade para ele não fará nada. Então cada perfil tem uma série de funcionalidades que serão chamadas de acordo com o perfil de usuário, com isso, teremos Integração Usuário Comum ou Integração Visual Reduzida e etc.

O fragmento de código abaixo demonstra o uso de uma fábrica de objeto para ser utilizado conforme o perfil.

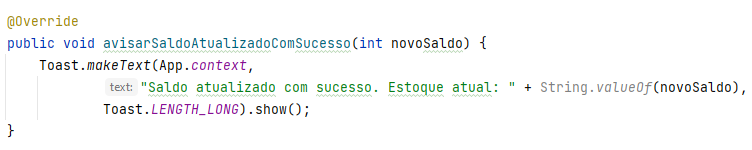
Figura 5



Quanto ao uso do Strategy abaixo temos dois fragmentos de código.

O primeiro fragmento dá suporte ao perfil de usuário comum.

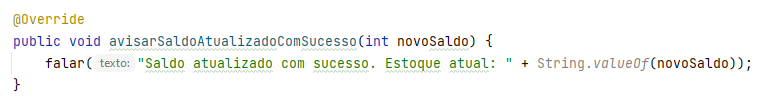
Figura 6



Na imagem acima, ao atualizar o saldo de medicamentos uma mensagem é exibida na tela.

O segundo fragmento dá suporte ao usuário com visão reduzida/perda

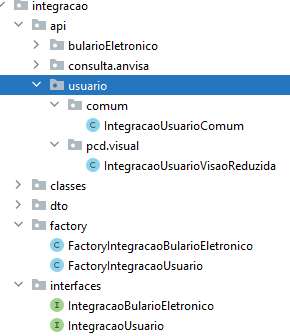
Figura 7



Na imagem acima, ao atualizar o saldo de medicamentos um áudio é reproduzido. Assim podemos constatar o uso do padrão Strategy.

Para dar suporte a estrutura mencionada acima foi necessário dividir o projeto em camadas. Abaixo temos uma imagem que exibe a estrutura da camada de Api-integração

Figura 8



### Janela Principal

Como o objetivo do presente trabalho é ajudar pessoas com alguma dificuldade em fazer uso de medicamentos e nesta primeira versão serão atendidos um público com dificuldade visual reduzida/perda foram desenvolvidos alguns comandos de voz. Partimos da premissa que tal perfil precisa de um cuidador para fazer algumas etapas preliminares as quais iremos abordar em tópico anexo. Tais comandos estão disponíveis desde a tela principal exibida abaixo:



Comandos de voz disponíveis após o usuário tocar em qualquer área da tela principal. O aplicativo informa que o usuário poderá falar o comando reproduzindo um pequeno bip. Após escutar o comando falado o aplicativo informa que a captura foi encerrada e então processa a voz:

Figura 8



(Fonte: Android Studio)

**Comandos Disponíveis**

O AMU falará os comandos de voz disponíveis

**Lista de Medicamentos**

O AMU lerá as informações principais de cada medicamento previamente cadastrado

**Detalhe do Medicamento “apelido”**

O AMU localizará na lista o medicamento que corresponde ao apelido cadastrado. Após localizar o medicamento exibirá o detalhe lendo as principais informações do medicamento e informando que está na janela de detalhe do medicamento.

**Administrar**

O AMU trocará o perfil do usuário temporariamente possibilitando que o cuidando faça a administração dos dados cadastrados. Entre esses itens está: Cadastrar medicamento, cadastrar horários, cadastrar compra ou informar uma redução na quantidade de medicamento disponível. Poderá alterar o apelido do medicamento entre outros dados. Após efetuar a mudança basta clicar no botão “Restaurar perfil”.

**Sair**

O AMU será fechado.

Para termos acesso a todos os recursos de voz foi necessário utilizar alguns recursos importantes relativos à plataforma Android, bem como fundamentos do desenvolvimento móvel. O recurso de reconhecimento de voz, reprodução de voz e mesmo o uso do firebase são normalmente assíncronos.

Devido a essa característica precisamos “assinar” métodos para receber as respostas ao final da execução. Para isso foi necessário o uso do recurso de CallBacks.

Trata-se de uma função que é executada quando algum evento acontece ou depois que algum código chega ao estado desejado. Também conhecido como função de retorno, o Callback cria regras dentro de outras funções para que sejam utilizadas no futuro. Normalmente, ele age de forma assíncrona, ou seja, não é executado imediatamente. A aplicação continuará rodando enquanto espera o momento certo da sua execução.

O Callback é muito comum na linguagem Java, por exemplo, durante a busca de dados ou reprodução de mídia. Isso porque permite ao programador especificar o que deve ocorrer quando a execução acabar.

As notificações (ou “**callbacks**”) permitem que o aplicativo seja informado quando uma transação tiver seu status alterado. Dessa forma, podemos identificar quando um dado solicitado já foi devolvido pelo banco de dados firebase ou quando uma captura de voz já foi obtida.

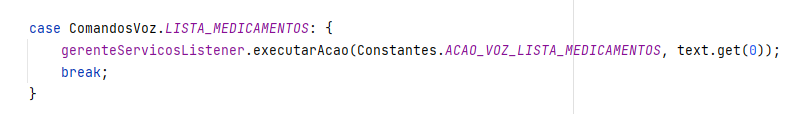
Para fazer uso deste importante recurso na programação orientada a objeto foi necessário fazer uso massivo de interfaces.

Alguns fragmentos de código abaixo mostram o uso de um desses esquemas.

Por exemplo para obter a lista de medicamentos a partir do uso da voz:

Fragmento 1

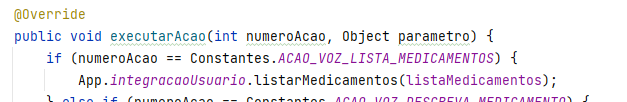
Figura 9



Fragmento 2

Fragmento 3

Figura 10



Nos fragmentos acima precisamos destacar os elementos chaves:

Gerente Serviço Listener e Constantes que determinam a entrada do comando e ação resultado. Vamos detalhar o fluxo:

Primeiramente vemos no fragmento 1 que durante a definição do formulário Principal Activity implementados a interface Gerente Servico Listener. Essa classe fará o papel do call-back.

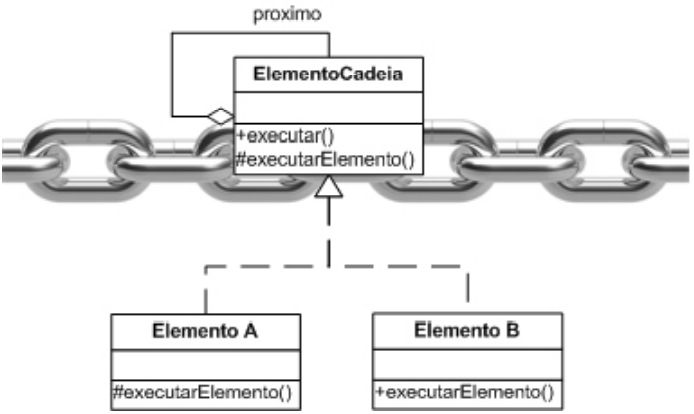
O fragmento 2 ocorre após a classe responsável pela escuta dos comandos de voz definirem que o comando é “Lista de Medicamentos”. Cada comando de voz tem uma ação de voz conectada. No caso atual é reproduzir em áudio a lista de medicamentos. Nesse momento uma chamada Callback ocorre para executar o próximo fragmento.

O fragmento 3 então aciona a variável atual que representa o perfil do usuário. Então o recurso de áudio lerá os itens que constam na lista de medicamentos exibida.

Figura 11

### 

O Aplicativo fará uma verificação de minuto a minuto sobre horários ativos de medicamentos. Como o aplicativo realiza chamadas assíncronas do banco de dados firebase será necessário adotar um outro padrão de projeto o Chain of Responsibility é um padrão de projeto que cria uma cadeia de execução na qual cada elemento processa as informações e em seguida delega a execução ao próximo da sequência. Em sua implementação tradicional, os elementos são percorridos até que um deles faça o tratamento da requisição, encerrando a execução depois disso. Como alternativa, também é possível criar uma cadeia de execução onde cada um executa sua funcionalidade até que a cadeia termine ou ela seja explicitamente finalizada por um dos elementos. Será necessário aninhar as chamadas de bancos de dados pois para geração de alarme precisaremos de: Horários Ativos, Utilizações realizadas e Registro de Alarmes. A imagem abaixo demonstra:

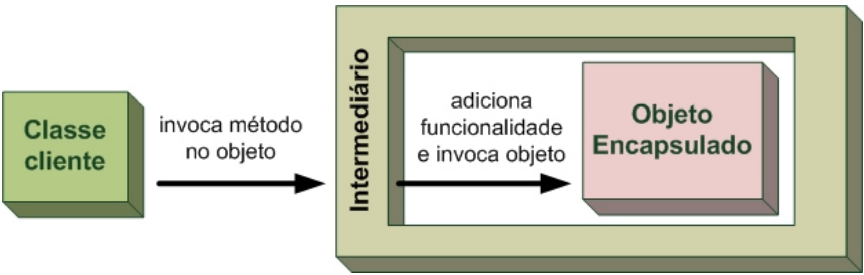
Figura 12

(Fonte: Livro Design Patterns em Java: Projeto orientado a objetos guiado por padrões - Eduardo Guerra)

Para diminuir o acoplamento, combinamos esse padrão com o padrão de projeto Proxy. O Proxy é um padrão de projeto estrutural que permite que você forneça um substituto ou um espaço reservado para outro objeto. Um proxy controla o acesso ao objeto original, permitindo que você faça algo ou antes ou depois do pedido chegar ao objeto original.

A ideia básica desse padrão é criar uma classe que envolve uma outra do mesmo tipo. Dessa forma, ela pode ser passada de forma transparente como se fosse a classe original para quem a irá utilizar. A imagem abaixo demonstra:

Figura 13



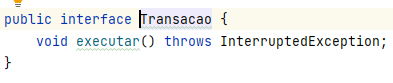
(Fonte: Livro Design Patterns em Java: Projeto orientado a objetos guiado por padrões - Eduardo Guerra)

No nosso caso utilizaremos os dois padrões basicamente para desacoplar a interface das chamadas de banco de dados.

Agora vamos colocar fragmentos de Código Java demonstrando o uso do padrão.

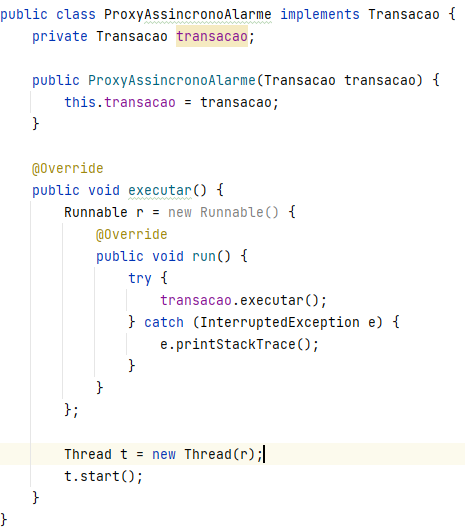
O primeiro fragmento apresenta a interface transações responsável por representar o componente intermediário

Figura 14



O segundo fragmento apresenta a classe proxy que implementa a interface transação. Ela adiciona uma funcionalidade à classe cliente do mesmo tipo.

Figura 15



No fragmento vemos que quando o método executar é invocado ele cria uma Thread e executa dentro dela o método da classe concreta que foi passada no construtor do proxy como transação cliente.

Dessa forma a classe proxy abstrai e desacopla a classe cliente. Então o método executa que deve ser executado é chamado.

Veja o terceiro fragmento.

Na parte final do fragmento percebemos o encadeamento das classes responsáveis por buscar os dados e disparar o alarme.

Figura 16

### 

### Detalhes do Medicamento

Após o usuário selecionar um medicamento da lista será exibida a tela de detalhes dos medicamentos. Nela encontram-se as principais funcionalidades que atendem o uso do medicamento: Registro de Utilização do Medicamento, compra de medicamento e horário de medicamentos.

Figura 17

### 

(Fonte: Autor)

Primeiramente iremos fragmentar a tela em diversos pedaços e abordar o que cada funcionalidade e após isso abordar como a camada de serviço interage com as funcionalidades.

Figura 17.a



Funcionalidade disponível somente se o apelido ou a cor ou a quantidade de medicamentos por embalagem for alterada.

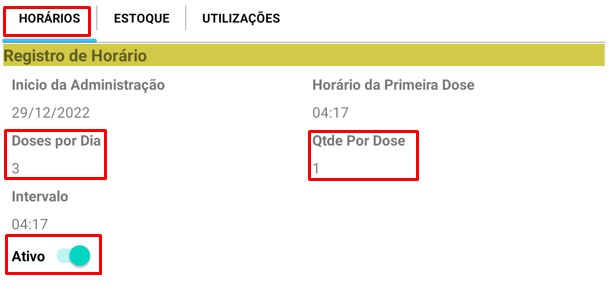
Figura 17.b



A funcionalidade sinaliza ao aplicativo que uma dose foi tomada pelo usuário do medicamento atual. São feitas algumas críticas para tornar o processo válido:

* O horário precisa estar cadastrado e ativo. O fragmento de tela abaixo demonstra o cadastro ativo de um medicamento.

Figura 17.c

* 
* (Fonte: Autor)
* O saldo do medicamento precisa ser maior ou igual a quantidade de medicamento por dose.

Após a validação, o aplicativo obterá o último saldo e criará um novo registro de saldo com o abatimento do saldo atual pela quantidade utilizada.

Também fará um registro com o horário da utilização. Os dois fragmentos da tela demonstram o que foi falado no texto acima.

O primeiro fragmento exibe o controle de saldo de estoque de um medicamento:

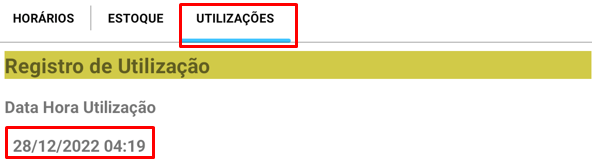
Figura 17.d



(Fonte: Autor)

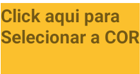
O segundo fragmento exibe o registro de uma utilização do medicamento:

Figura 17.e



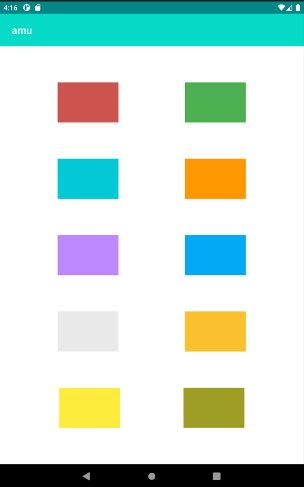
(Fonte: Autor)

Figura 17.f



Funcionalidade importante para usuários idosos já que a cor do medicamento pode ajudar a selecionar facilmente qual o medicamento. Ao clicar no botão uma paleta simples de cores será exibida. A lista de cores é reduzida para tornar mais simples o processo.

Figura 17.g

O usuário poderá apenas clicar em uma das cores e assim mudar a forma como é apresentada a linha na lista de medicamentos.

(Fonte: Autor)

Figura 17.h



Funcionalidade que tem o objetivo de dar entrada no estoque de medicamentos. Com base no que foi informado na caixa de texto “estoque” será adicionado no saldo atual.

Figura 17.i



Funcionalidade que tem o objetivo de dar saída no estoque de medicamentos. Com base no que foi informado na caixa de texto “estoque” será subtraído do saldo atual.

Figura 17.j



Funcionalidade que tem a finalidade de abrir o cadastro de horário.

A funcionalidade relacionada acima em sua maioria tem um correspondente para dar suporte através dos comandos de voz.

Vamos listar

Comandos de voz disponíveis após o usuário tocar em qualquer área da tela detalhe do medicamento. O aplicativo informa que o usuário poderá falar o comando reproduzindo um pequeno bip. Após escutar o comando falado o aplicativo informa que a captura foi encerrada e então processa a voz:

Figura 8



(Fonte: Android Studio)

**Comandos Disponíveis**

O AMU falará os comandos de voz disponíveis

**Horário do Medicamento**

O SAMU fará a leitura dos dados principais do último horário cadastrado se houver. Caso não exista, o usuário receberá a informação também por voz.

**Remédio Utilizado**

O AMU executará a mesma funcionalidade que o botão Utilização. Porém informará o resultado da ação por voz para o usuário ter certeza que utilizou o remédio certo.

**Estoque Atual**

O AMU informará a quantidade do medicamento que ainda existe na embalagem ou no estoque.

**Entrada “quantidade”**

O AMU executará a mesma funcionalidade do botão +. Porém informará o resultado da ação por voz para o usuário ter certeza que deu entrada no remédio certo. A palavra quantidade é a máscara para representar o número informado de entrada.

**Saída “quantidade”**

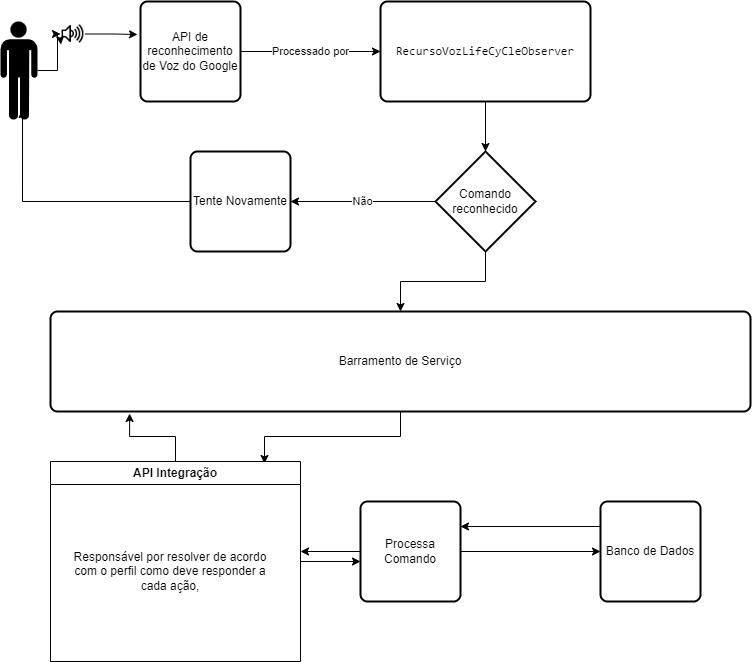
O AMU executará a mesma funcionalidade do botão +. Porém informará o resultado da ação por voz para o usuário ter certeza que deu saída no remédio certo. A palavra quantidade é a máscara para representar o número informado de saída.

**Voltar**

O AMU voltará para a tela principal.

Similarmente ao comando de voz “LISTA DE MEDICAMENTOS” o AMU precisará recorrer ao uso de algumas API para por exemplo executar a ação “UTILIZAR MEDICAMENTO”. Vamos descrever agora o processo.

Figura 18



(Fonte: Autor)

A classe Recurso Voz LifeCyCle Observer é fundamental no processamento de voz. Para facilitar o desenvolvimento unificamos todas as chamadas de voz para ela. Novamente para conseguir esse objetivo utilizamos o call-back.

Além de utilizar o processo de callback foi necessário fazer uso do reconhecimento de voz do google identificado através da intenção RecognizerIntent.

O uso da intenção tem por base Android Speech API que fornece controle de reconhecimento, serviços em segundo plano, intenções e suporte para vários idiomas. A simples adição à entrada do usuário para o aplicativo é um recurso muito poderoso.

Tal recurso é útil para pessoas com deficiência que usam um teclado ou simplesmente para quem está tentando encontrar uma maneira de aumentar a produtividade e melhorar seu fluxo de trabalho.

A imagem acima descreve o processamento de qualquer comando de voz realizado pelo usuário. Vamos descrever agora o comando de voz “REMÉDIO UTILIZADO”. Vamos tentar relacionar cada etapa das classes.

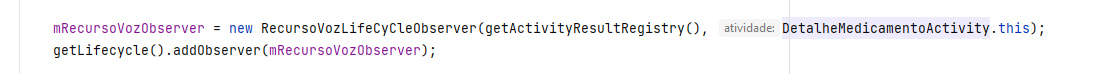
Conforme o fragmento 1 de código java implementamos a interface Gerente Serviços Listener que terá o papel de dar suporte de call-back.

Figura 19



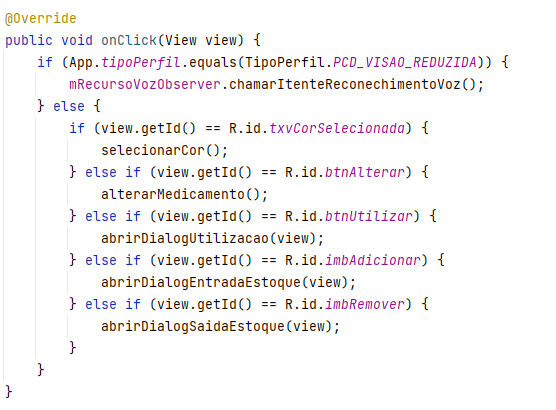
No segundo fragmento já assinamos para o gerenciador de voz qualquer interação por voz do usuário com a tela com o simples toque na tela

Figura 20



No terceiro fragmento vemos que qualquer clique na tela e se o perfil for para o reconhecimento de voz acionará a interface de voz

Figura 21



Logo após invocar o serviço de reconhecimento de voz o assistente do google aparece com o símbolo característico da imagem abaixo

Imagem

O serviço do google então percebe que o usuário parou de falar e devolve para o AMU o controle passando o conteúdo da voz em formato de texto. O fragmento abaixo de código java demonstra esse tratamento.

Figura 22



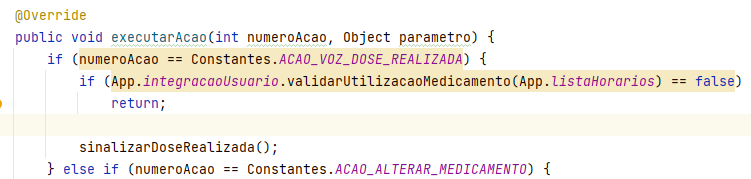
Embora extenso, utilizamos uma estrutura switch. Claro que poderíamos estender o serviço do padrão de projeto Strategy para eliminar a estrutura. Porém isso demandaria mais tempo e tornando o projeto ainda mais complexo. Para diminuir o tempo de desenvolvimento optamos por não utilizar.

Novamente vemos no trecho acima de código o uso da interface Gerente Serviços Listener. Assim que o comando de voz é processado e reconhecido a ação é realizada através do tipo de ação “ACAO\_VOZ\_DOSE\_REALIZADA”.

Aqui é importante destacar que a tela do Android é chamada para resolver essa funcionalidade.

O fragmento abaixo exibe como é feito o processo. Um método para executar Ação da interface do Gerente Serviços Listener implementada pela tela do Android é acionada. O fragmento então demonstra a chamada seguinte onde irá percorrer pelas camadas de serviços e no final registrar a utilização do medicamento.

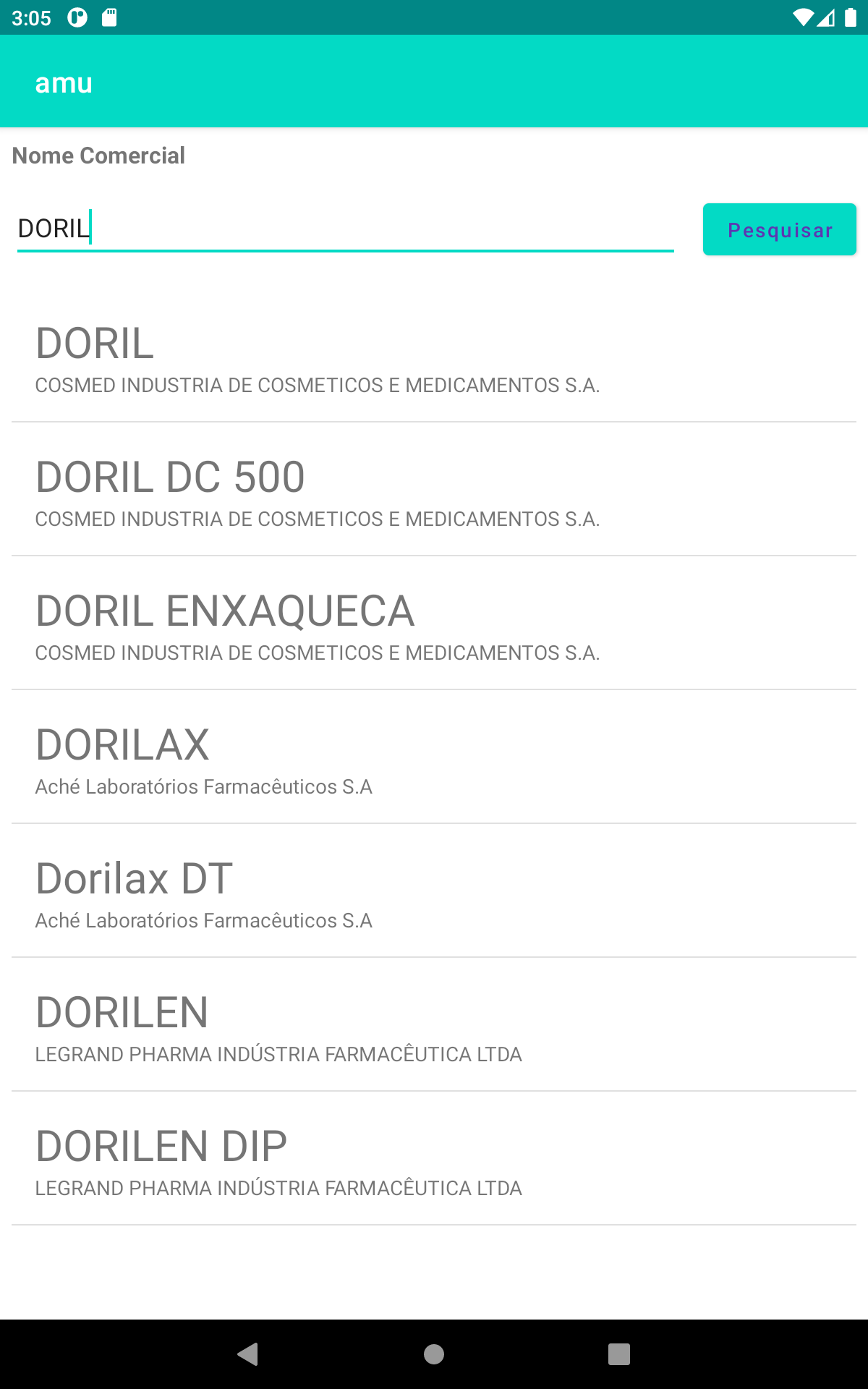
Figura 23



O tópico final que será abordado será a pesquisa de medicamentos na base de dados da ANVISA. Nesta versão essa parte não terá suporte ao recurso de voz, porém utilizaremos da mesma forma algumas chamadas de Apis. Entre elas API Retrofit assim como o recurso de callback já que a busca na ANVISA tem como característica ser um serviço assíncrono por escolha de implementação. Naturalmente vamos precisar abordar nessa seção a conversão de arquivo pdf para arquivo texto.

Abaixo vemos a tela que deverá ser exibida após o usuário clicar no + da tela principal.

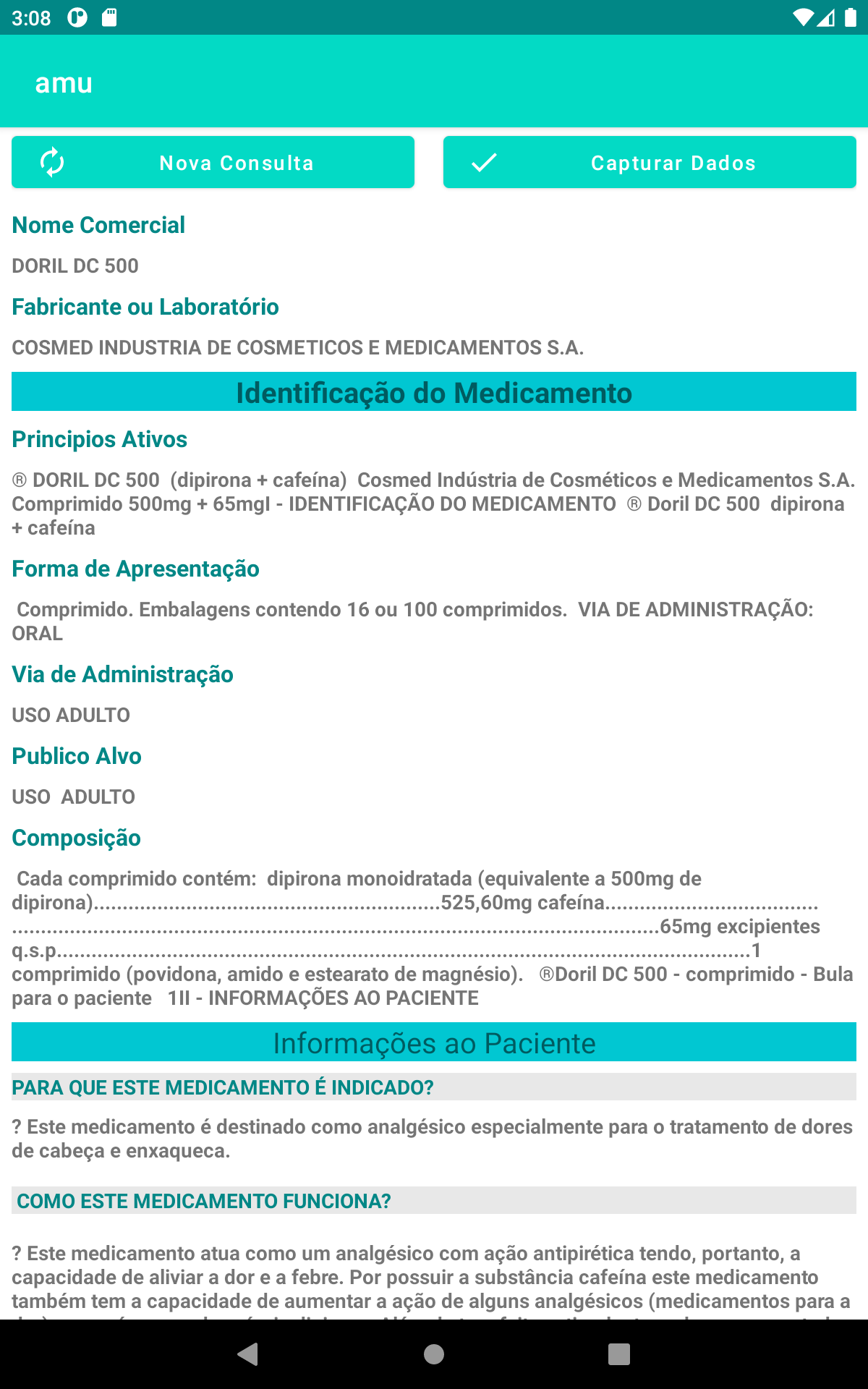
Figura 24



(Fonte: Autor)

Aqui vemos uma lista preenchida com medicamento chamado “doril”. Ao clicar em um dos itens da lista pesquisada na ANVISA através da API retrofit podemos visualizar os detalhes obtidos.

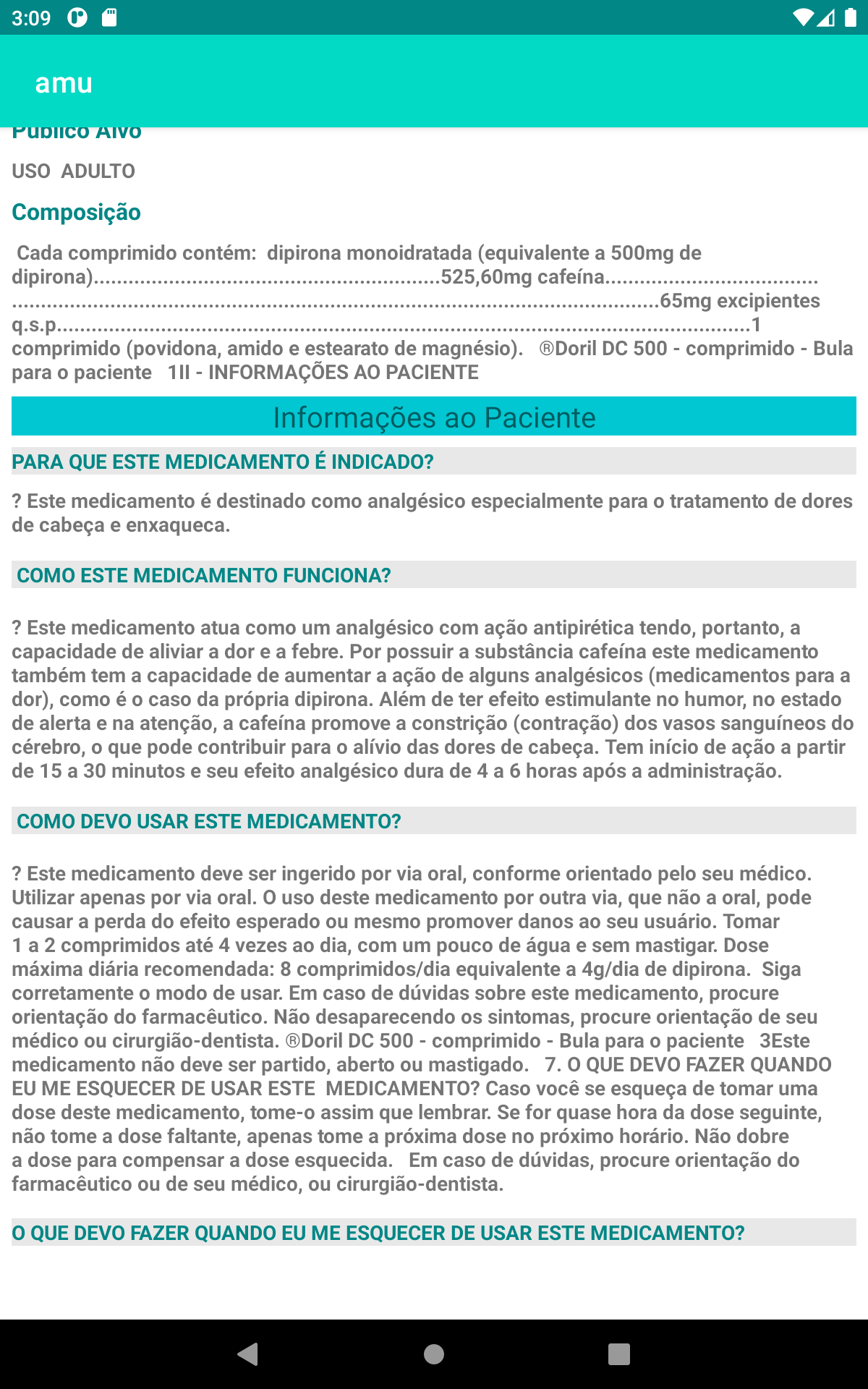
Figura 25



(Fonte: Autor)

A segunda parte da tela continua exibindo os detalhes

Figura 26



(Fonte: Autor)

O primeiro fragmento da tela possui um botão para capturar os dados. Ao clicar nesse botão o usuário entrará no cadastro de medicamento onde poderá complementar os dados como apelido, cor e quantidade de medicamento por embalagem. Além de já iniciar o estoque do medicamento através da quantidade por embalagem, o usuário poderá cadastrar o horário.

A próxima imagem demonstra esse recurso.

Figura 27



(Fonte: Autor)

Como já explicamos a tela de detalhe do medicamento anteriormente não será necessário o detalhamento da tela. O usuário deverá apenas complementar as informações de apelido, cor, quantidade por embalagem e horário. Após isso clique no botão confirmar.

Uma informação importante para destacar nesta funcionalidade é que a fonte de pesquisa não precisa necessariamente ser a ANVISA. Isso é possível devido ao uso do Padrão Strategy. Então podemos ter fontes de Dados diferentes para a busca dos dados de medicamentos. Para dar suporte ao recurso utilizamos a interface Bulário Eletrônico Cliente. A interface é sustentada com o uso da tecnologia REST. Esta tecnologia tem como base o uso de arquivos do tipo JSON. A grande vantagem dessa tecnologia é a facilidade em conectar aplicações que rodam em ambientes heterogêneos.

REST é uma coleção de regras que os desenvolvedores seguem ao criar APIs; um conjunto de princípios que regem como diferentes programas se comunicam. Portanto, uma API REST é simplesmente uma API que aplica esses princípios.

Quando um cliente, um programa que solicita a conexão com uma API, solicita um recurso (informações que podem ser comunicadas e compartilhadas usando uma API), o estado existente do recurso é transferido de volta pelo servidor em uma representação padronizada.

Uma das regras do REST é que você deve obter um dado (um recurso) ao vincular a um determinado URL. As APIs REST permitem que uma solicitação de um recurso vá do cliente para o servidor e, em seguida, que as informações relevantes sejam enviadas de volta como resposta.

Uma solicitação consiste em um endpoint (a URL que você solicita), um método, que define o tipo de solicitação enviada ao servidor; cabeçalhos, que representam os metadados; e dados.

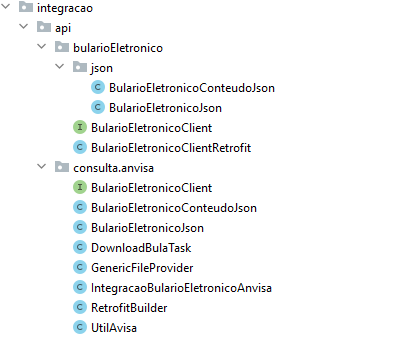
O fragmento abaixo demonstra isso, com as definições.

Figura 28



Para dar suporte a estrutura mencionada acima foi necessário dividir o projeto em camadas. Abaixo temos uma imagem que exibe a estrutura da camada de Api-integração.

Figura 29



### Banco de Dados Firebase

O AMU utiliza para armazenar os dados do banco de dados Firebase.

O Firebase Realtime Database é um banco de dados hospedado na nuvem. Os dados são armazenados como JSON e sincronizados em tempo real para cada cliente conectado. Todos os usuários compartilham uma instância do Realtime Database e recebem automaticamente atualizações com os dados mais recentes.

O Firebase Realtime Database permite criar aplicativos avançados e colaborativos, permitindo acesso seguro ao banco de dados diretamente do código do lado do cliente. Os dados são mantidos localmente e, mesmo offline, os eventos em tempo real continuam a ser acionados, proporcionando ao usuário final uma experiência responsiva. Quando o dispositivo recupera a conexão, o Realtime Database sincroniza as alterações de dados locais com as atualizações remotas que ocorreram enquanto o cliente estava offline, mesclando quaisquer conflitos automaticamente.

O Realtime Database fornece uma linguagem de regras flexível e baseada em expressões, chamada Firebase Realtime Database Security Rules, para definir como seus dados devem ser estruturados e quando os dados podem ser lidos ou gravados. Quando integrado ao Firebase Authentication, os desenvolvedores podem definir quem tem acesso a quais dados e como eles podem acessá-los.

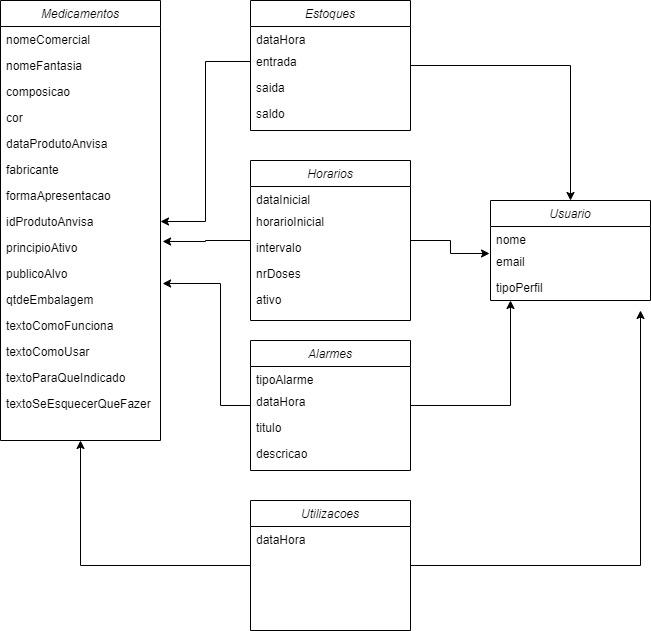
O Realtime Database é um banco de dados NoSQL e, como tal, possui otimizações e funcionalidades diferentes em comparação com um banco de dados relacional. A API do Realtime Database foi projetada para permitir apenas operações que podem ser executadas rapidamente. Isso permite que você crie uma ótima experiência em tempo real que pode atender a milhões de usuários sem comprometer a capacidade de resposta.

Conforme a estrutura mostrada abaixo não temos encadeamento de tabelas. Essa decisão foi tomada para melhorar a performance conforme recomendado na página do Firebase. Por isso o banco do Aplicativo AMU é plano.

Como o Firebase Realtime Database permite aninhar dados em até 32 níveis de profundidade, você pode ficar tentado a pensar que essa deve ser a estrutura padrão. No entanto, quando você busca dados em um local em seu banco de dados, também recupera todos os seus nós filhos. Além disso, quando você concede a alguém acesso de leitura ou gravação em um nó em seu banco de dados, você também concede acesso a todos os dados desse nó. Portanto, na prática, é melhor manter sua estrutura de dados o mais plana possível.

**Estrutura de Banco de Dados AMU**

Figura 30

****

(Fonte: Autor)

### Dicionário de Dados

Todos os dados do Firebase Realtime Database são armazenados como objetos JSON. Você pode pensar no banco de dados como uma árvore JSON hospedada na nuvem. Ao contrário de um banco de dados SQL, não há tabelas ou registros. Quando você adiciona dados à árvore JSON, eles se tornam um nó na estrutura JSON existente com uma chave associada. Você pode fornecer suas próprias chaves, como IDs de usuário ou nomes semânticos, ou elas podem ser fornecidas para você usando [push()](https://firebase.google.com/docs/reference/android/com/google/firebase/database/DatabaseReference#push()) .

Utilizo o nome “tabela” apenas para simplificação, embora conforme explicado acima existem nós para representar os dados.

**Tabela de Usuários**

**Objetivo:** registrar os usuários que utilizam o aplicativo.

**Campos**

| **Campo** | **Descrição** |  |
| --- | --- | --- |
| idUsuario | Código do Usuário |  |
| Nome | Nome do Usuário |  |
| Email | Email do usuário | Relacionado a Autenticação do Firebase |
| tipoPerfil | Tipo do Perfil | Campo que determina o comportamento do AMU |

**Tabela Medicamentos**

**Objetivo:** Registrar os medicamentos com dados capturados da Anvisa

**Campos**

| **Campo** | **Descrição** |  |
| --- | --- | --- |
| idMedicamento | Código Automático |  |
| idUsuario | Código do Usuário |  |
| nomeComercial | Nome Comercial do Medicamento |  |
| nomeFantasia | Apelido do Medicamento |  |
| qtdeEmbalagem | Quantidade do Medicamento na Embalagem |  |
| Cor | Cor que identifica o medicamento na lista de medicamentos |  |
| composicao | Composição do Medicamento | Importado da bula |
| dataProdutoAnvisa | Data do Produto Anvisa | Importado da Bula. Também utilizado para determinar se a bula foi alterada. |
| idProdutoAnvisa | Chave Identificadora da Bula na Anvisa | Importado da Bula. Também utilizado para determinar se a bula foi alterada. |
| fabricante | Nome do Fabricante da Bula da Anvisa. | Importado da Bula. |
| formaApresentacao | Forma da Apresentação do Remédio | Importado da Bula. |
| principioAtivo | Princípio Ativo | Importado da Bula. |
| publicoAlvo | Público Alvo | Importado da Bula. |
| textoComoFunciona | Como funciona o medicamento | Importado da Bula. |
| textoComoUsar | Como usar o medicamento | Importado da Bula. |
| textoParaQueIndicado | Para que o medicamento é indicado | Importado da Bula. |
| textoSeEsquecerQueFazer | O Que fazer se esquecer de tomar o medicamento | Importado da Bula. |

**Tabela de Estoque**

**Objetivo**: Registrar a quantidade disponível do medicamento para utilização do usuário. O usuário poderá adicionar saldo ou subtrair o saldo.

**Campos**

| **Campo** | **Descrição** |  |
| --- | --- | --- |
| idEstoque | Código Automático |  |
| dataHora | Data e Hora do Registro |  |
| Entrada | Quantidade de Entrada no Estoque. |  |
| Saída | Quantidade de Saída do Estoque. |  |
| Saldo | Saldo considerando saldo anterior mais entrada menos saída |  |

**Tabela de Alarmes**

**Objetivo**: Registrar os alarmes produzidos com base no horário e utilizações.

**Campos**

| **Campo** | **Descrição** |  |
| --- | --- | --- |
| idAlarme | Código Automático |  |
| tipoAlarme | Tipo de Alarme |  |
| dataHora | Data e Hora do Alarme |  |
| Titulo | Título do Alarme |  |
| Descricao | Descrição do Alarme |  |

**Tabela de Horários**

**Objetivo**: Registrar os horários que os medicamentos devem ser tomados

**Campos**

| **Campo** | **Descrição** |  |
| --- | --- | --- |
| idHorario | Código Automático |  |
| idMedicamento | Código do medicamento |  |
| idUsuario | Código do Usuário |  |
| dataInicial | Data Inicial que o medicamento deve ser administrado |  |
| horaInicial | Horário da Primeira dose do dia |  |
| Intervalo | Intervalo entre as doses |  |
| nrDoses | Número de Doses por dia |  |
| QtdePorDose | Quantidade que deve ser administrada durante cada dose |  |
| Ativo | Indica se horário está ativo ou não. |  |

**Tabela de Utilização**

**Objetivo**: Registrar quando o usuário utiliza o medicamento.

**Campos**

| **Campo** | **Descrição** |  |
| --- | --- | --- |
| idUtilizacao | Código Automático |  |
| idMedicamento | Código do Medicamento |  |
| idUsuario | Código do Usuário |  |
| DataHora | Data e Hora da Utilização |  |

**Dicionário**

**ESB**

Um ESB, ou barramento de serviço corporativo, é um padrão pelo qual um componente de software centralizado realiza integração a sistemas back-end (e conversões de modelos de dados, conectividade profunda, roteamento e solicitações) e disponibiliza essas integrações e conversões como interfaces de serviço para reutilização por novos aplicativos. O padrão ESB é geralmente implementado usando um tempo de execução de integração e um conjunto de ferramentas especialmente projetadas que garantem a melhor produtividade possível.

**Strategy**

O Strategy é um padrão que deve ser utilizado quando uma classe possuir diversos algoritmos que possam ser utilizados de forma intercambiável. A solução proposta pelo padrão consiste em delegar a execução do algoritmo para uma instância que compõe a classe principal. Dessa forma, quando a funcionalidade for invocada, no momento de execução do algoritmo, será invocado um método da instância que a compõe. A Figura 1.4 apresenta um diagrama que mostra a estrutura básica do padrão.

**Referências**

1. Design Patterns com Java – Projeto orientado a objetos guiado por padrões
   * Página 233
   * Facade ou Bus de Serviço
2. Design Patterns com Java – Projeto orientado a objetos guiado por padrões.
   * Página 200
3. Autenticação no Utilizando Firebase
   * Acessado em 28/12/2022
   * <https://firebase.google.com/products/auth?gclid=CjwKCAiA76-dBhByEiwAA0_s9To21pNpQy3x9y79Uyl8YgNx3iwk0U0Ox8dx45IfKmPpNjXxG22eVhoCC60QAvD_BwE&gclsrc=aw.ds>
4. Uso de Polimorfismo em Java
   * Acessado em 28/12/2022
   * <https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140>
5. Definição de Barramento de Serviços
   * Acessado em 07/12/2022
   * <https://www.ibm.com/br-pt/cloud/learn/esb#:~:text=Um%20ESB%2C%20ou%20barramento%20de,como%20interfaces%20de%20servi%C3%A7o%20para>
6. Definição de CallBack
   * Acessado em 30/12/2022
   * <https://www.jivochat.com.br/blog/ferramentas/o-que-e-callback.html>
   * <https://pt.stackoverflow.com/questions/27177/o-que-%C3%A9-callback>
7. Uso da API de reconhecimento de voz
   * Acessado em 30/12/2022
   * <https://www.developer.com/guides/exploring-the-android-speech-api-for-voice-recognition/>
8. Conceituação do Uso de REST API
   * Acessado em 31/12/2022
   * <https://www.algolia.com/blog/product/what-is-a-rest-api/?utm_source=google&utm_medium=paid_search&utm_campaign=rl_amer_search_nb_dynamic&utm_content=blog_product_dynamic&utm_term=&utm_region=amer&utm_model=nonbrand&utm_ag=rl&utm_persona=dev&_bt=566443924550&_bm=&_bn=g&gclid=Cj0KCQiAtbqdBhDvARIsAGYnXBO_ScMhgHXIq95FqMQ6ZJhiYsC02C-lE6gK8hT_ObqR9EHBSbNqr3caAvvBEALw_wcB>
9. Design Patterns com Java – Projeto orientado a objetos guiado por padrões
   * Página 87
   * Chain of Responsibility
10. Design Patterns com Java – Projeto orientado a objetos guiado por padrões
    * Página 100
    * Proxies
11. Definição Banco de RealTime
    * Acessado em 31/12/2022
    * <https://firebase.google.com/docs/database/android/structure-data>