DESENVOLVIMENTO

O aplicativo AMU produto final do trabalho corrente destina-se a auxiliar a utilização dos medicamentos por grupos de pessoas com alguma comorbidade. Logo, para isso é necessário apresentar algumas bases de que propiciem a adequação de forma mais simples. A figura 200 demostra alguns componentes:

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

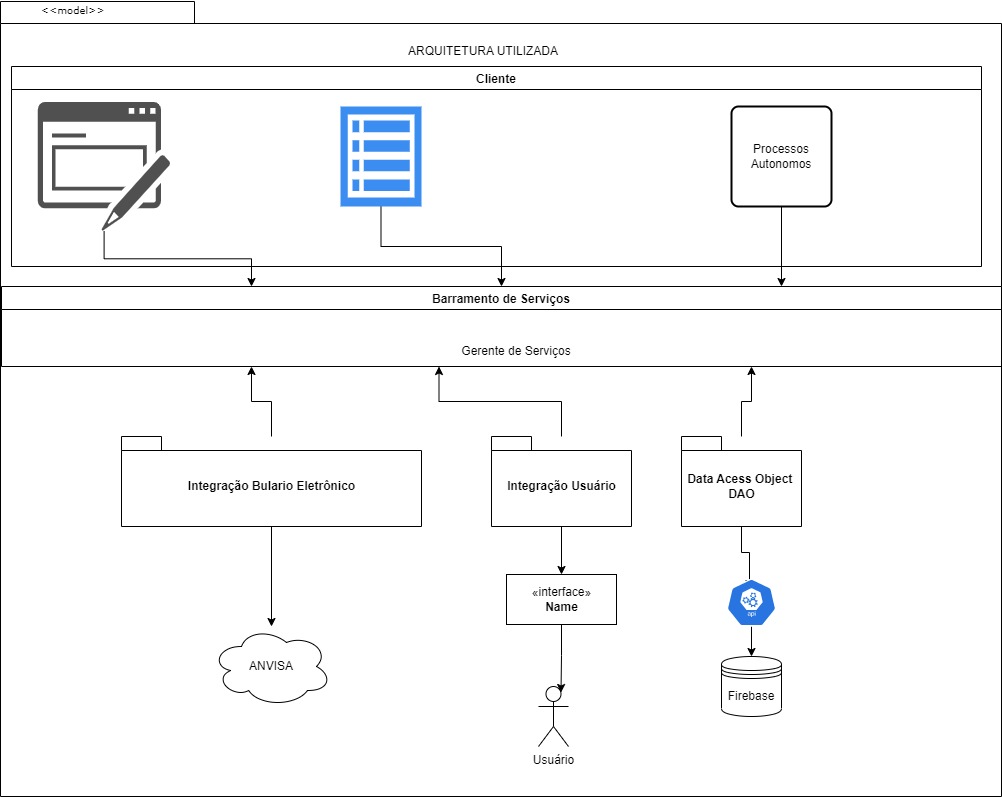


Figura 2000

Explicando a arquitetura de alto nível

1. **Cliente**

Objetivamente falando seria qualquer interface ou recurso utilizado pelo usuário para consumir ou solicitar um serviço do aplicativo. Pode ser um formulário ou uma lista. Assim como pode ser um serviço autônomo monitorando horários por exemplo.

1. **Barramento de Serviços**

Objetiva criar uma interface de acesso a todos os recursos do aplicativo. Para realizar 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)

1. **Interface de Integração Bulário Eletrônico**

Esse componente possibilita criar uma ponte entre o serviço e a consulta a fontes de dados de medicamentos. Aqui 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 chegando ao ponto de trocar a fonte de dado e a mudança não gerar nenhum tipo de problema no restante do projeto. (2)

1. **Interface de Integração de Usuário**

Seguindo o mesmo padrão “Strategy” abordado no tópico “Integração de Bulário Eletronico” é necessário criar recursos específicos para os tipos de comorbidades apresentadas. Por exemplo temos uma interface para pessoas com visão reduzida ou mesmo sem visão. Segue a lista das interfaces:

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

Utilizando a interface podemos por exemplo avisar que é horário de 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.

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 está remoto. Utilizamos o padrão de projeto “DAO”. Esse padrão possibilita a troca do banco de dados. Apenas trocando a fábrica da implementação.

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.

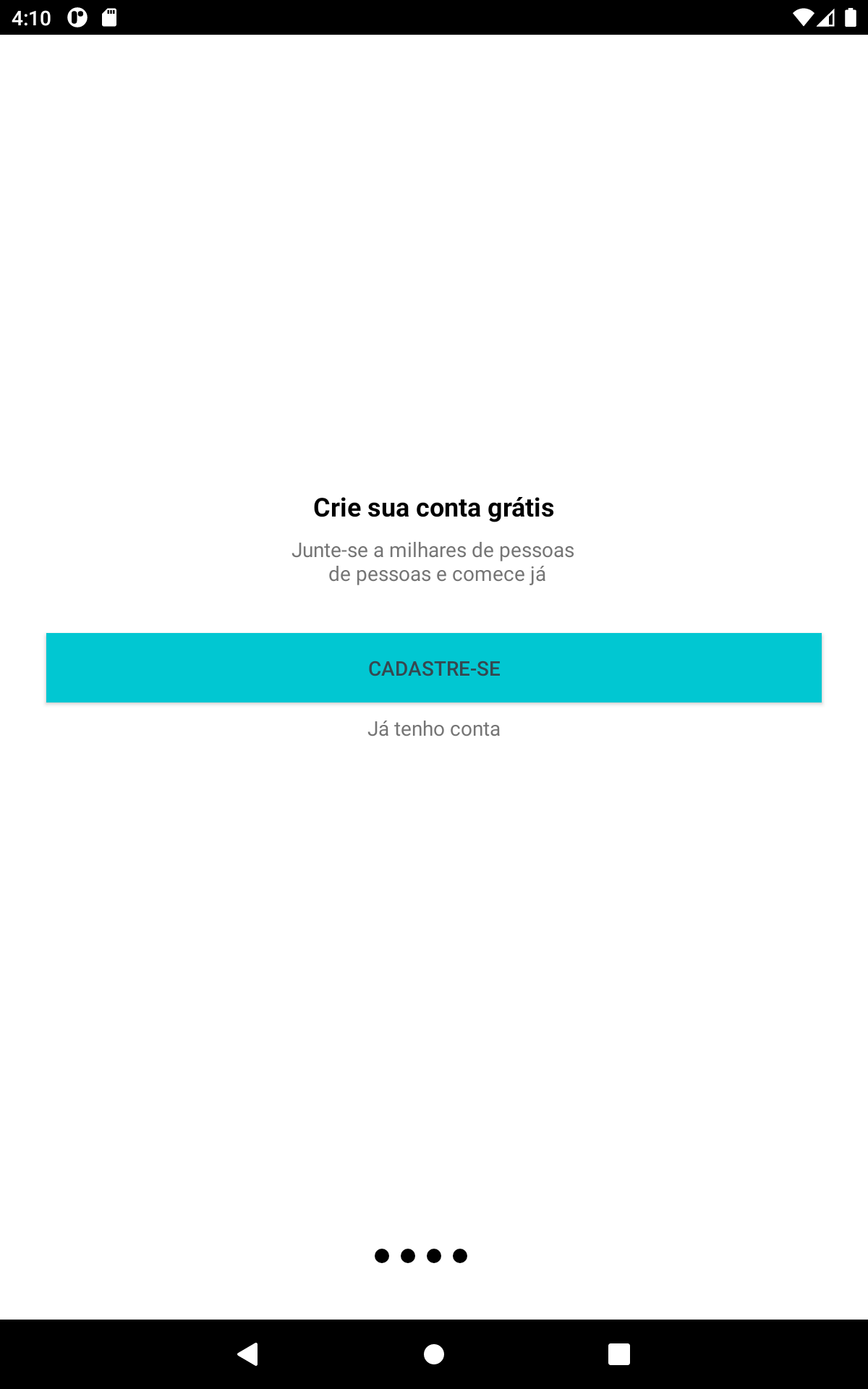
Tanto no primeiro caso quanto no segundo caso o aplicativo exibirá a janela principal.

Quando o usuário já tiver feito login pelo menos uma vez o aplicativo utilizará o recurso FirebaseAuth 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. (Referencia 3)

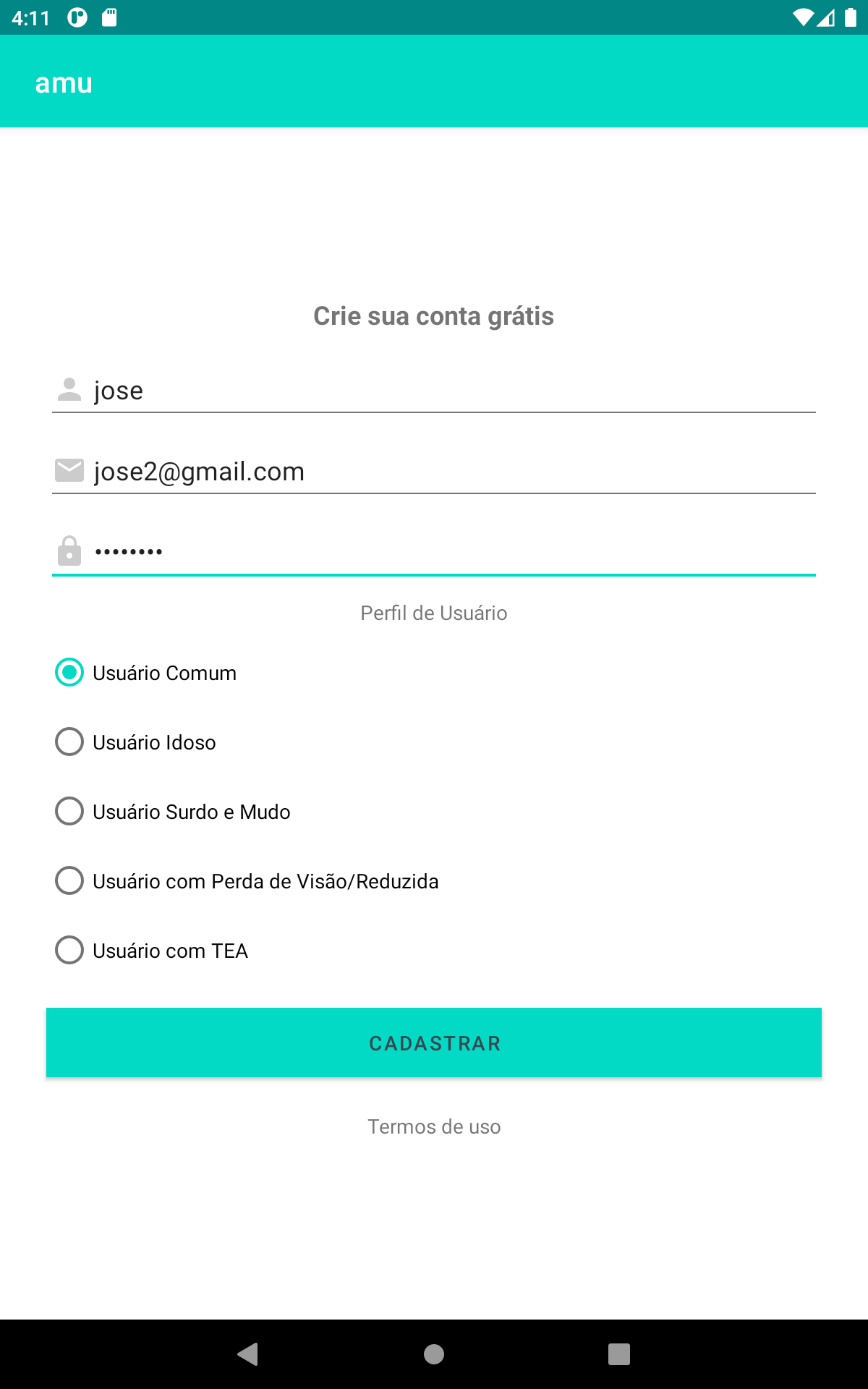
Dito isto, sempre que o usuário acessar o aplicativo fará login automático apresentando facilidade de uso desde a tela inicial.

Boas vindas



Autenticação de Usuário



**Cadastro de Usuário**

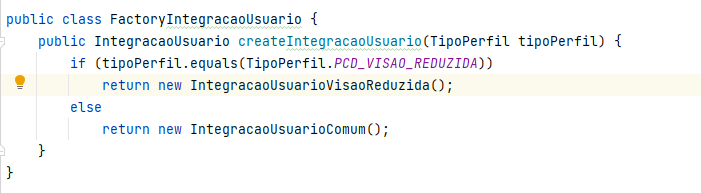
Nesta primeira versão do aplicativo AMU o usuário comum poderá utilizar todas as funcionalidades. O usuário com perda de visão ou reduzida terá através do uso da voz acesso a algumas funcionalidades.

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.

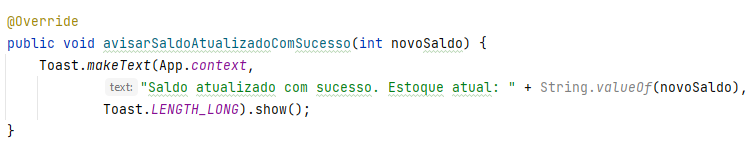
Quando o aplicativo inicia pelo cadastro inicial sabe qual o perfil do usuário. Uma variável global com o nome “integracaoUsuario” então é inicializada por uma fábrica de objetos. Para exemplificar o uso desse recurso, quando o aplicativo dispara um alarme indicando que um medicamento deve ser tomado 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 “integracaoUsuario” que disparará um alarme em audio. 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 IntegracaoUsuarioComum ou IntegracaoUsuarioVisaoReduzida e etc.

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



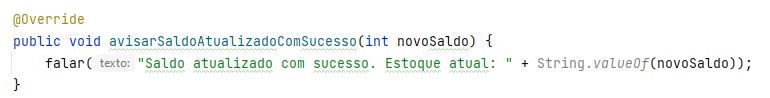
Quanto ao uso do Strategy abaixo temos dois fragmentos de código.

O primeiro fragmento dá suporte ao perfil de usuário comum.



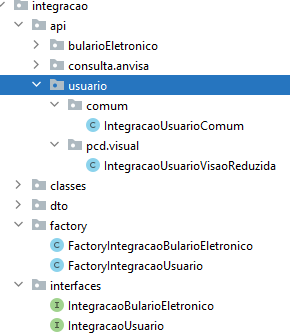
Na imagem acima ao atualizar o saldo de medicamentos uma mensagem é exibida na tela.

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



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



### 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 comados 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:

 **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 a 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 assincronos.

Devido a essa caracteristica precisamos “assinar” metodos para receber as respostas ao final da execução. Para issofoi necessário o uso do recuso 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 ja 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.

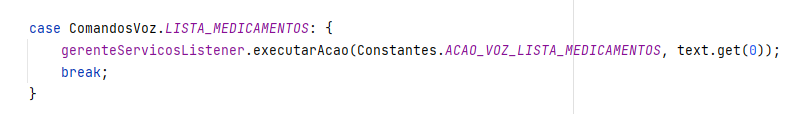
Alguns fragmentos de código abaixo mostram o uso de um desses esquemas.

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

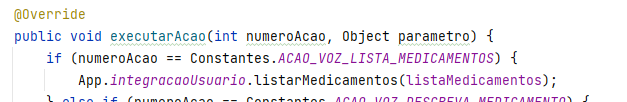
Fragmento 1



Fragmento 2



Fragmento 3



Nos fragmentos acima precisamos destacar os elementos chaves:

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

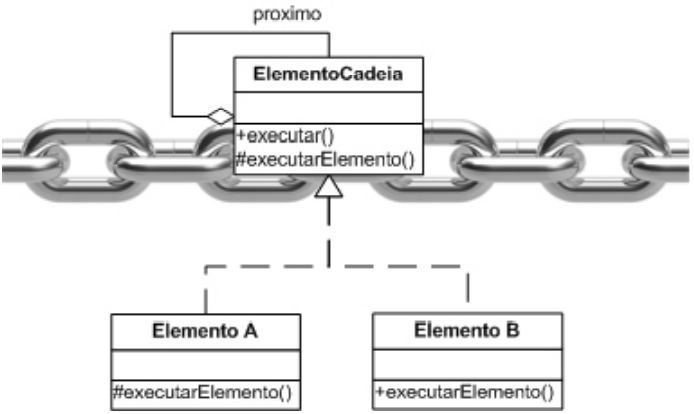
Primeiramente vemos no fragmento 1 que durante a definição do formulário PrincipalActivity implementados a interface GerenteServicoListener. Essa classe fará o papel do call-back.

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.

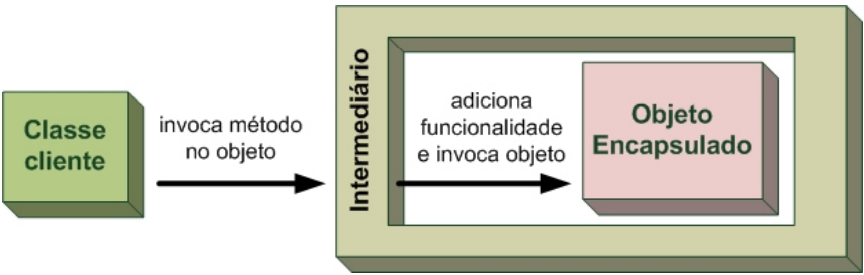
### 

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 encadear 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:



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.

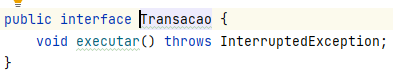
A ideia básica desse padrão é criar uma classe que envolve uma outra do mesmo tipo. Dessa forma, ela pode ser passada de forma transparente como se fosse a classe original para quem a irá utilizar. A imagem abaixo demonstra:



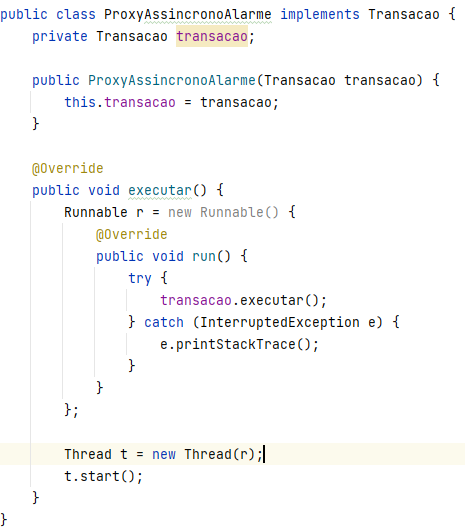
No nosso caso utilizaremos os dois padrões basicamente para desacoplar a interface das chamadas de banco de dados.

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



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



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.

### 

### Detalhes do Medicamento

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

### 

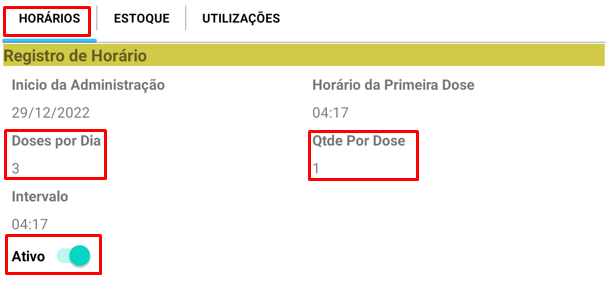
Primeiramente iremos fragmentar a tela em diversos pedaços e abordar o que cada funcionalidade e após isso abordar como a camada de serviço interage com as funcionalidades.



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



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

* Horário precisa está cadastrado e ativo. O fragmento de tela abaixo demonstra o cadastro ativo de um medicamento.
* 
* O saldo do medicamento precisa ser maior ou igual a quantidade de medicamento por dose.

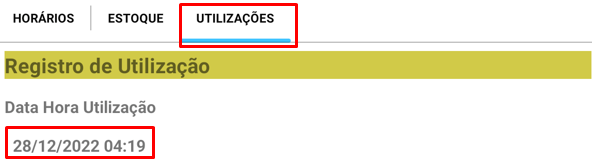
Após a validação o aplicativo obterá o ultimo saldo e criará um novo registro de saldo com o abatimento do saldo atual pela quantidade utilizada.

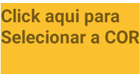
Também fará um registro com o horário da utilização. Os dois fragmentos da tela demonstram o que foi falado no texto acima.

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

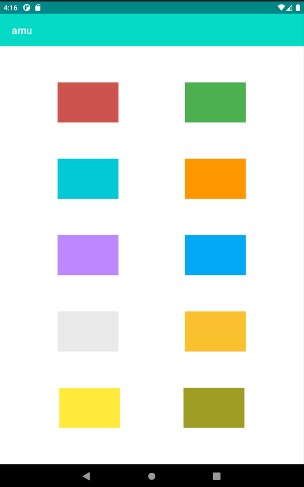


O segundo fragmento exibe o registro de uma utilização do medicamento:





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.

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



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.



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



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:

 **Comandos Disponíveis**

O AMU falará os comandos de voz disponíveis

 **Horário do Medicamento**

O Amu 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. Porem 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 ainda existe na embalagem ou no estoque.

 **Entrada “quantidade”**

O AMU executará a mesma funcionalidade do botão +. Porem 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.

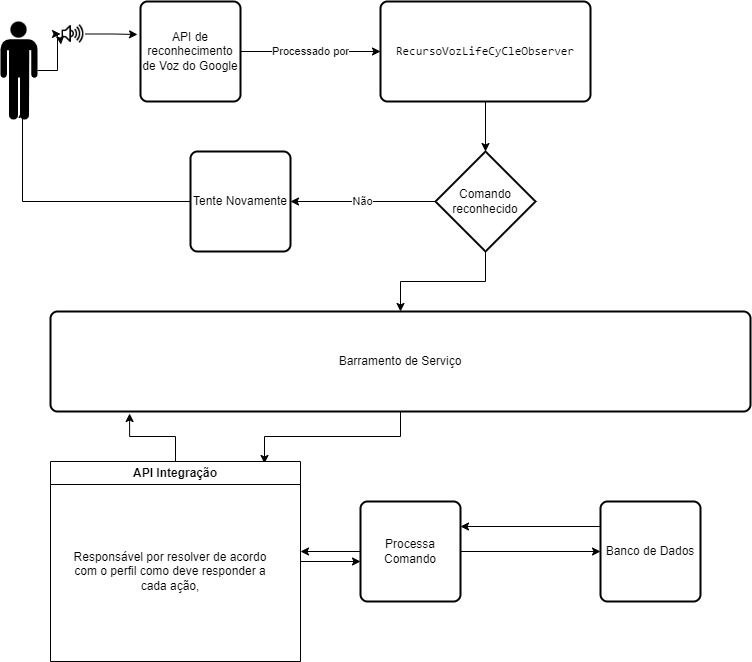
 **Saída “quantidade”**

O AMU executará a mesma funcionalidade do botão +. Porem 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.



A classe RecursoVozLifeCyCleObserver 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.

Alem de utilizar o processo de call-back foi necessário fazer uso do reconhecimento de voz do google identificado através da intente RecognizerIntent.

O uso da intent 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.

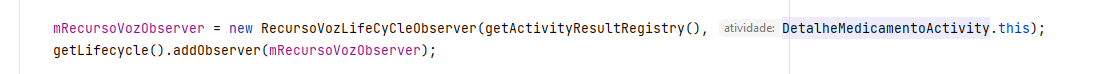
Tal recurso é útil para pessoas com deficiência que usam um teclado ou simplesmente para quem está tentando encontrar uma maneira de aumentar a produtividade e melhorar seu fluxo de trabalho.

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 as classes.

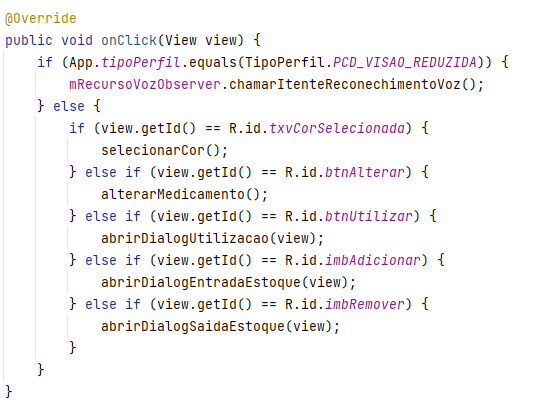
Conforme o fragmento 1 de código java implementamos a interface GerenteServicosListener que terá o papel de dar suporte de call-back.



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



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



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.

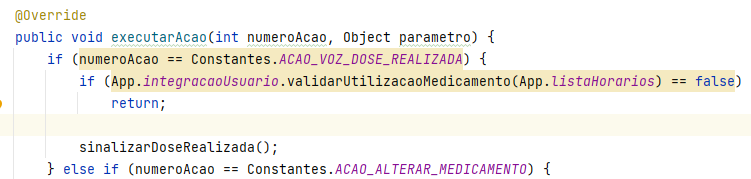


Embora extenso utilizamos uma estrutura swich. Claro que poderíamos extender o serviço do padrão de projeto Strategy para eliminar a estrutura. Porem isto demandaria mais tempo e tornando o projeto ainda mais complexo. Para diminuir o tempo de desenvolvimento optamos por não utilizar.

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

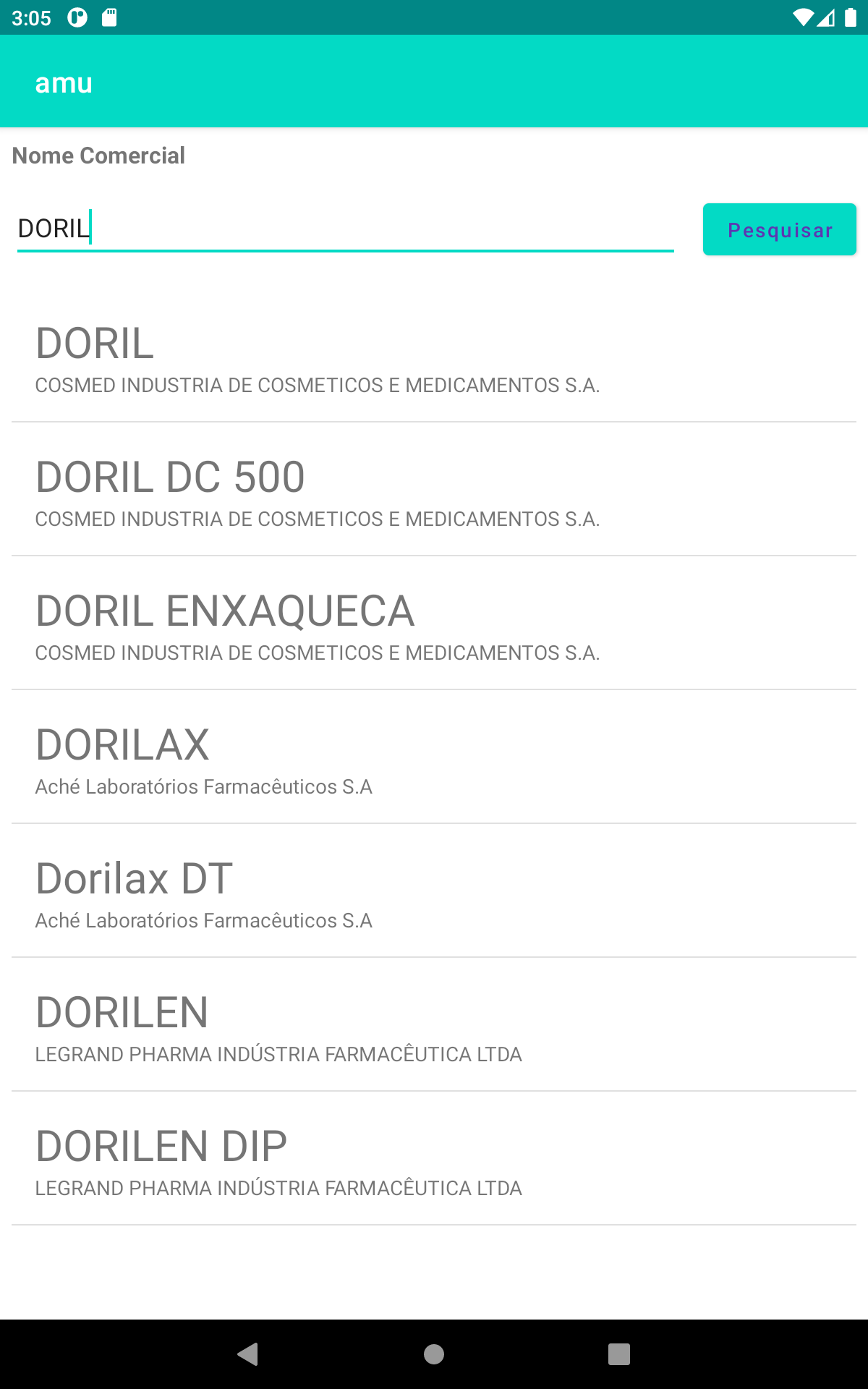
Aqui é importante destacar que a tela do Android é chamda para resolver essa funcionalidade.

O fragmento abaixo exibe como é feito o processo. Um método executarAcao da interface do GerenteServicosListener implementa pela tela do Android é acionada. O fragmento então demontra a chamada seguinte onde irá percorrer pelas camadas de serviços e no final registrar a utilização do medicamento.

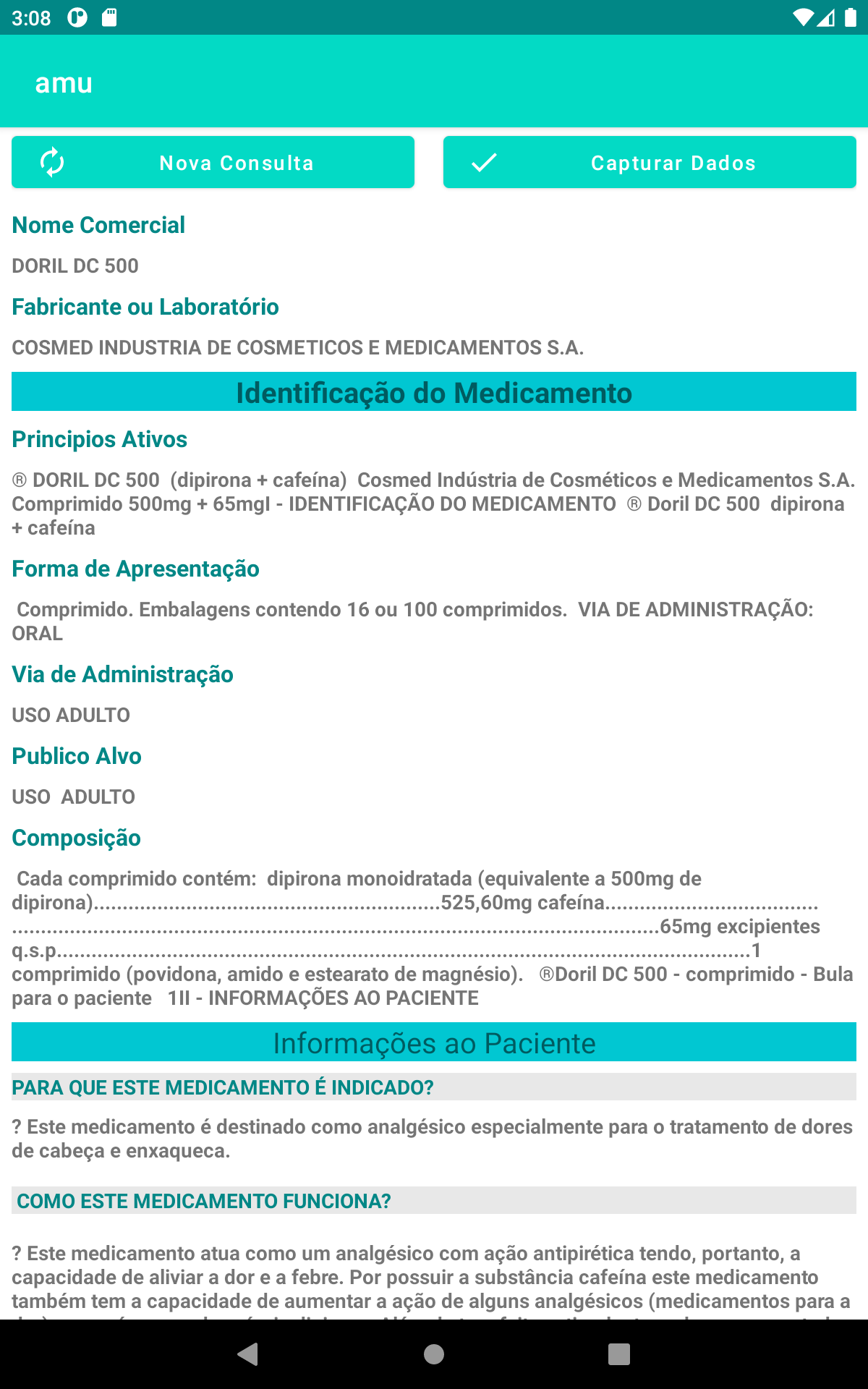


O tópico final que será abordado será a pesquisa de medicamentos na base de dados da ANVISA. Nesta versão essa parte não terá suporte ao recuso de voz, porém utilizaremos da mesma forma algumas apis. Entre elas API Retrofit assim como o recurso de call-back já que a busca na ANVISA tem como característica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

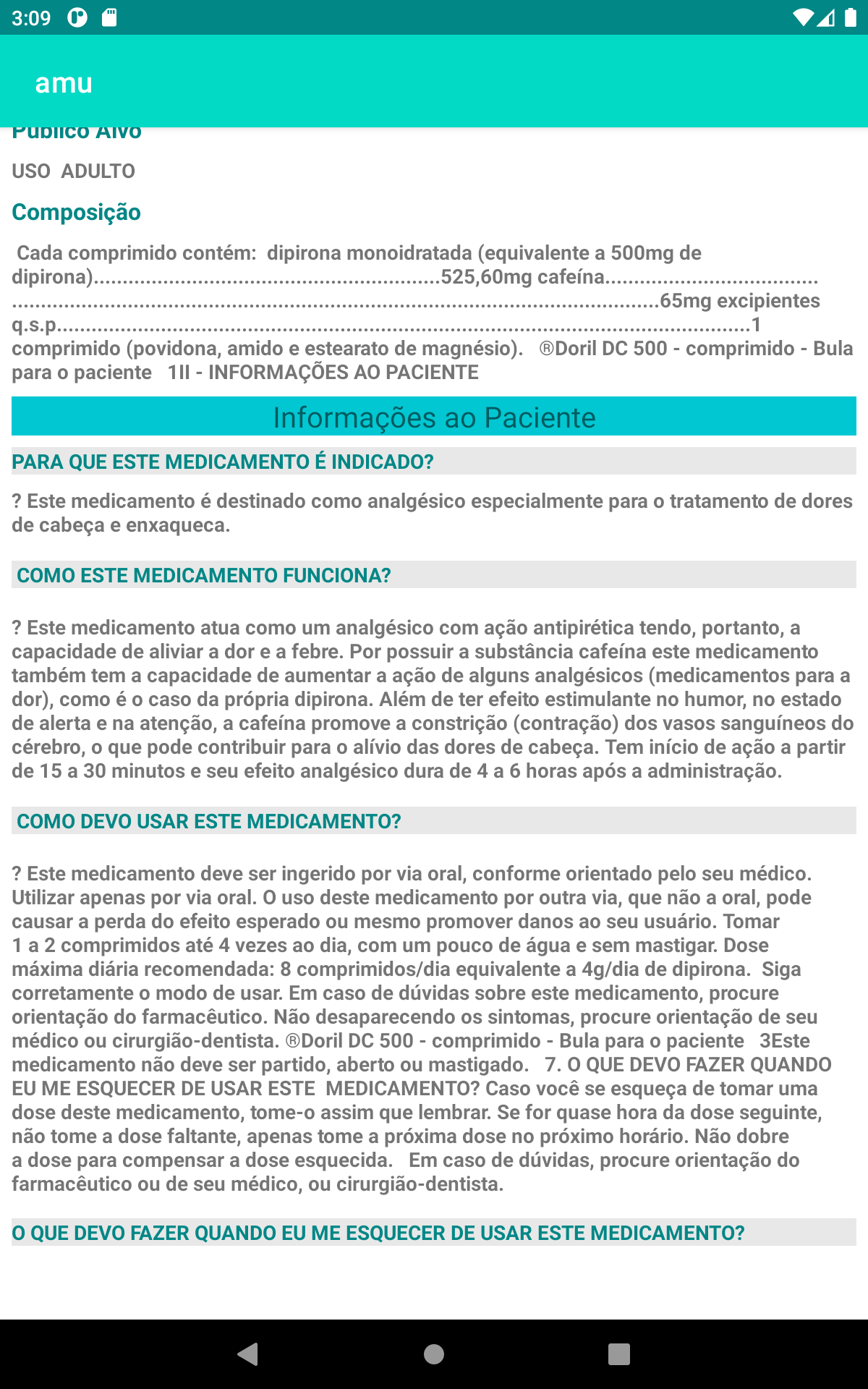
Abaixo vemos a tela que deverá ser exibida após o usuário clicar no + da tela principal.



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



A segunda parte da tela continua exibindo os detalhes



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 inicializar o estoque do medicamento através da quantidade por embalagem, o usuário poderá cadastrar o horário.

A próxima imagem demonstra esse recurso.



Como já explicamos a tela de detalhe de medicamento anteriormente não será necessário alongarmos 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 uma interface BularioEletronicoClient. A interface é sustentada com o uso da tecnologia REST. Esta tecnologia tem como base o uso de arquivo do tipo JSON. A grande vantagem dessa tecnologia é facilidade em implementar qualquer api.

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.

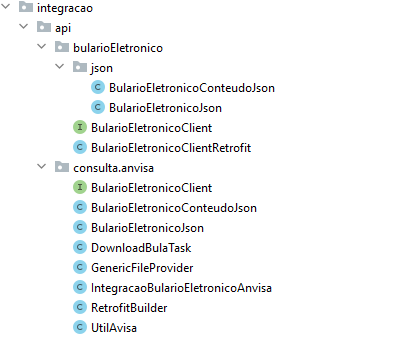
Uma das regras do REST é que você deve obter um dado (um recurso) ao vincular a um determinado URL. As APIs REST permitem que uma solicitação de um recurso vá do cliente para o servidor e, em seguida, que as informações relevantes sejam enviadas de volta como resposta.

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.



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.



### Banco de Dados Firebase

O AMU utiliza para armazenar os dados o 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.

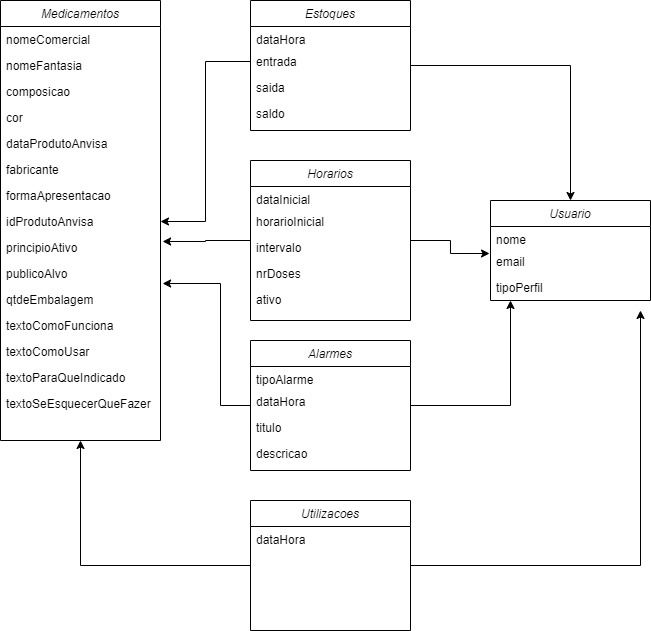
O Realtime Database fornece uma linguagem de regras flexível e baseada em expressões, chamada Firebase Realtime Database Security Rules, para definir como seus dados devem ser estruturados e quando os dados podem ser lidos ou gravados. Quando integrado ao Firebase Authentication, os desenvolvedores podem 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**

****

### Dicionário de Dados

**Tabela de Usuários**

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

**Campos**

|  |  |  |
| --- | --- | --- |
| **Campo** | **Descrição** |  |
| idUsuario | Código do Usuário |  |
| Nome | Nome do Usuario |  |
| 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 Médicamento |  |
| 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 integrações 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 projetados 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
   * Pagina 233
   * Facade ou Bus de Serviço
2. Design Patterns com Java – Projeto orientado a objetos guiado por padrões.
   * Pagina 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. Definção 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
   * Pagina 87
   * Chain of Responsibility
10. Design Patterns com Java – Projeto orientado a objetos guiado por padrões
    * Pagina 100
    * Proxies
11. Definição Banco de RealTime
    * Acessado em 31/12/2022
    * <https://firebase.google.com/docs/database/android/structure-data>