

**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA  
PAULA SOUZA**

**Faculdade de Tecnologia Baixada Santista  
Rubens Lara**

**Curso Superior de Tecnologia em  
Sistemas para Internet**

**JULIA GATO HEITOR  
MARJORYE CIARDULLO RAMIREZ**

**CARETAKER**

**Santos, SP**

**2023**

**JULIA GATO HEITOR**  
**MARJORYE CIARDULLO RAMIREZ**

**CARETAKER**

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia Rubens Lara, como exigência para a obtenção do Título de Tecnólogo em Sistemas para Internet.

**Orientadores: Prof. Felipe Cannarozzo  
Lourenço**

**Prof.<sup>a</sup> Me. Rosemeire Cardozo Vidal**

**Santos, SP**

**2023**

## RESUMO

Com o avanço tecnológico e o uso de aplicativos para tarefas diárias, as gerações com costumes analógicos são muitas vezes desconsideradas como audiência alvo de novos sistemas. Entretanto, com o envelhecimento, a necessidade e o desejo por independência não se esvaem, resultando em uma era tecnológica que avança cada vez mais rápido, porém tendo apenas jovens em mente. Com o intuito de desenvolver um sistema de auxílio a organização de um público mais idoso, foi realizado uma aplicação de introdução, controle e organização de informações médicas com ênfase em simplicidade, acessibilidade e intuitividade, levando em conta um público menos acostumado com aplicativos e tecnologia. Durante o desenvolvimento, foram realizadas pesquisas e construção teórica de requisitos e diagramas em que serviram de base para a composição prática da aplicação. Para averiguação de resultados, vistos os objetivos definidos de início, diversos testes de sistema foram realizados para a apreciação de erros, bem como testes junto à possíveis usuários da faixa etária almejada a fim de examinar o cumprimento desses requisitos, baseando-se nesses resultados majoritariamente positivos para concluir a utilidade real do aplicativo, bem como utilizando as críticas para efetuar melhorias para o seu uso.

**Palavras-chaves:** Idosos. Tratamento. Agenda. Organização.

## **ABSTRACT**

With technological advancements and the reliance on apps for daily tasks, generations with analog habits are often disregarded as the target audience for new systems. However, as people age, the need and desire for independence does not fade away, resulting in a rapidly advancing technological era that predominantly caters to the younger population. In order to develop a system to assist the organization of an older audience, an application was created for the introduction, control, and organization of medical information, with an emphasis on simplicity, accessibility, and intuitiveness, taking into account a user base that may be less accustomed to apps and technology. During the development process, research and theoretical construction of requirements and diagrams were conducted to serve as a foundation for the practical composition of the application. To evaluate the results and achieve the defined objectives, various system tests were carried out to identify any errors, along with tests involving potential users within the targeted age range to assess the fulfillment of these requirements. Based on predominantly positive results, the actual usefulness of the application was concluded, and feedback was utilized to make improvements for its usage.

**Keywords:** Elderly. Treatment. Schedule. Organization.

## LISTA DE ABREVIATURAS E SIGLAS

JS – JAVASCRIPT	10
SPA – SINGLE PAGE APPLICATION	10
SQL – STRUCTURED QUERY LANGUAGE	10
RN – REACT NATIVE	10
API – APPLICATION PROGRAMMING LANGUAGE	11
REST – REPRESENTATIONAL STATE TRANSFER	11
JWT – JSON WEB TOKEN	11
HTML – HYPERTEXT MARKUP LANGUAGE	17
CSS – CASCADING STYLE SHEETS	17
AJAX – ASYNCHRONOUS JAVASCRIPT AND XML	18
OOAD – OBJECT ORIENTED ANALYSIS AND DESIGN	19
HTTP – HYPERTEXT TRANSFER PROTOCOL	19
JSON – JAVASCRIPT OBJECT NOTATION	24
MER – MODELO ENTIDADE RELACIONAMENTO	28

## LISTA DE ILUSTRAÇÕES

ILUSTRAÇÃO 01 – FUNCIONALIDADES DENTRO DO <i>APP MYTHERAPY</i>	12
ILUSTRAÇÃO 02 – FUNCIONALIDADES DENTRO DO <i>APP MEDICATION REMINDER</i>	13
ILUSTRAÇÃO 03 – FUNCIONALIDADES DENTRO DO <i>APP PILL REMINDER &amp; MEDICINE APP</i>	14
ILUSTRAÇÃO 04 – ROTA DO TIPO <i>GET</i> PARA A PÁGINA DE AGENDA	20
ILUSTRAÇÃO 05 – ROTA DO TIPO <i>POST</i> PARA O CADASTRO DE USUÁRIO	21
ILUSTRAÇÃO 06 – ROTA DO TIPO <i>PUT</i> PARA ATUALIZAÇÃO DE USUÁRIO	22
ILUSTRAÇÃO 07 – ROTA DO TIPO <i>DELETE</i> PARA A REMOÇÃO DE MEDICAMENTO	23
ILUSTRAÇÃO 08 – TABELAS DA APLICAÇÃO <i>CARETAKER</i>	28
ILUSTRAÇÃO 09 – CÓDIGO DO CONSTRUTOR DO FORMULÁRIO DE MEDICAMENTO	30
ILUSTRAÇÃO 10 – REQUISIÇÃO DE DADOS NA PÁGINA DE AGENDA	31
ILUSTRAÇÃO 11 – FUNÇÃO DE ORGANIZAÇÃO DE INFORMAÇÕES DA AGENDA	32
ILUSTRAÇÃO 12 – TELA DE CADASTRO DA APLICAÇÃO	33
ILUSTRAÇÃO 13 – TELA DA “ÁREA DO PACIENTE”	34
ILUSTRAÇÃO 14 – FORMULÁRIO DE MEDICAMENTO PREENCHIDO	35
ILUSTRAÇÃO 15 – TELA DA PÁGINA DE AGENDA	36
ILUSTRAÇÃO 16 – CONFIRMAÇÃO DE AÇÃO	37
ILUSTRAÇÃO 17 – ALTERAÇÕES APÓS <i>FEEDBACK</i>	40

## **LISTA DE TABELAS**

TABELA 1 – REQUISITOS FUNCIONAIS DO APLICATIVO CARETAKER	25
TABELA 2 – REQUISITOS NÃO FUNCIONAIS DO APLICATIVO CARETAKER	26

# SUMÁRIO

1 INTRODUÇÃO.....	9
1.1 OBJETIVO .....	9
1.1.1 OBJETIVO GERAL .....	10
1.1.2 OBJETIVOS ESPECÍFICOS .....	10
1.2 ESTADO DA ARTE .....	11
2 DESENVOLVIMENTO .....	15
2.1 ANÁLISE DO SISTEMA.....	15
2.1.1 ANÁLISE DE REQUISITOS .....	24
REQUISITOS FUNCIONAIS .....	24
2.1.2 DIAGRAMA DE CASO DE USO .....	26
2.1.3 FLUXO DE EVENTOS .....	27
2.2 BANCO DE DADOS.....	27
2.3 CAMADA DE NEGÓCIO .....	29
2.4 CAMADA DE APRESENTAÇÃO.....	32
3 RESULTADO .....	38



# 1 INTRODUÇÃO

A facilidade proporcionada pelo uso de meios digitais permite sua aplicação na organização cotidiana de maneira rápida e eficiente. Nesse sentido, o aplicativo *Caretaker* tem o propósito de aproveitar essa facilidade para fornecer a organização necessária de consultas médicas, medicamentos e exames para aqueles que necessitam de um sistema próprio que atenda às suas necessidades.

“Os aplicativos voltados para a saúde e o cuidado de idosos são recursos importantes, visto que essas informações obtidas por meio da internet e outras mídias podem influenciar o estilo de vida, propiciar a detecção precoce de eventuais problemas de saúde e promover o envelhecimento ativo e saudável. Além disso, despertam o interesse e a curiosidade da população idosa, tornando-se um recurso de entretenimento que contribui também para a sua inclusão digital”. (AMORIM, Diane Nogueira Paranhos. et. Al, 2018)

Portanto, uma aplicação voltada para a organização da agenda médica dos usuários se apresenta como uma solução prática e eficaz para promover a saúde, prevenir riscos e garantir um acompanhamento terapêutico adequado. Além disso, essa ferramenta pode ser útil para proporcionar independência aos usuários idosos, facilitando sua organização com o registro atualizado das suas informações clínicas.

“Os aplicativos objetivam (...) o auxílio ao cuidado de idosos e a divulgação de informações sobre saúde e sobre doenças e tratamentos. Esses recursos podem trazer grande contribuição à saúde e ao aperfeiçoamento do cuidado ao idoso, sendo um instrumento de monitoramento, informação e promoção de hábitos saudáveis”. (AMORIM, Diane Nogueira Paranhos. et. Al, 2018)

## 1.1 OBJETIVO

O objetivo deste trabalho é desenvolver um aplicativo móvel direcionado a pessoas idosas, com o intuito de auxiliar na gestão de medicamentos, consultas e exames médicos.

### 1.1.1 OBJETIVO GERAL

O objetivo geral do *Caretaker* ("cuidador" em português) é servir como uma ferramenta assistiva para a organização da agenda médica, direcionado para o uso pessoal dos usuários a fim de manter suas informações médicas centralizado em um só lugar, de maneira a traduzir instrumentos analógicos, como uma agenda de mão ou um calendário, à um meio digital de fácil acesso e usabilidade.

Através do controle de calendário virtual e do uso de lembretes, o aplicativo é capaz de oferecer um local concentrado de informações médicas relevantes do usuário, incluindo datas, horários, nomes de seus profissionais médicos específicos, entre outros, para garantir um acompanhamento adequado dos tratamentos e manutenção de seu histórico médico.

Para implementação do aplicativo, foram utilizadas as linguagens *JavaScript* (JS) por meio do framework de *Single Page Application* (SPA), *React Native* (RN), para o *front-end*, e *Python*, com o *microframework Flask* para o *back-end*. O banco de dados seguirá o sistema de *Structured Query Language* (SQL) chamado *MySQL*.

### 1.1.2 OBJETIVOS ESPECÍFICOS

A fim de cumprir as funcionalidades essenciais da aplicação desenvolvida em RN, foi possível:

- o registro, alteração e remoção de dados de perfil, consultas, exames, medicamentos e lembretes, através de componentes do RN;
- visualização em agenda dos registros de consultas, exames e medicamentos cadastrados, por meio de Componentes de RN;

- uso da SPA para renderização de diferentes páginas desde formulários de cadastro às telas de organização de dados (eg. página de perfil e agenda).

Além do *microframework Flask* para a criação da *Application Programming Interface Representational State Transfer (API REST)*, possibilitando:

- conexão ao banco de dados, para permitir registro, alteração ou remoção de dados, realizados por usuários durante o cadastro das informações e lembretes na aplicação;
- rotas específicas para serem utilizadas pelo *front-end* para manipulação e persistência dos dados;
- realização da autenticação com o uso de *JSON Web Token (JWT)*.

## 1.2 ESTADO DA ARTE

É pertinente a análise de outras aplicações com mesmo ou similar propósito ao *Caretaker*. Em razão disso, alguns aplicativos que apresentam similaridades o suficiente ao projeto foram analisados:

- **Aplicativo 1:** *MyTherapy* (Alarme de Medicamentos)

O aplicativo "*MyTherapy* - Alarme de Medicamentos" tem como objetivo principal fornecer uma agenda para lembretes de medicamentos cadastrados pelo usuário. Por meio dele, é possível manter um controle abrangente de tratamentos específicos, utilizando recursos de registro, frequência e modos de administração de doses ao longo do tratamento cadastrado.

No que diz respeito às funcionalidades secundárias, mencionadas na Ilustração 1, o aplicativo oferece diversas opções de registro de sintomas, humor e bem-estar, auxiliando no tratamento do paciente e permitindo a inclusão de informações adicionais e relevantes ao quadro clínico cadastrado (ex. medicamentos utilizados pelo usuário). Além dessas funcionalidades, há

a opção de cadastro de uma equipe, que poderá acompanhar todo o tratamento listado dentro do aplicativo.

**Ilustração 1** - Funcionalidades dentro do *app MyTherapy*



**Fonte:** Google Store.

- **Aplicativo 2:** *Medication Reminder* (Hora do Remédio)

O aplicativo "Hora do Remédio" tem como objetivo auxiliar no acompanhamento de tratamentos com medicamentos e vitaminas, criando lembretes para o usuário nas datas e horários adequados.

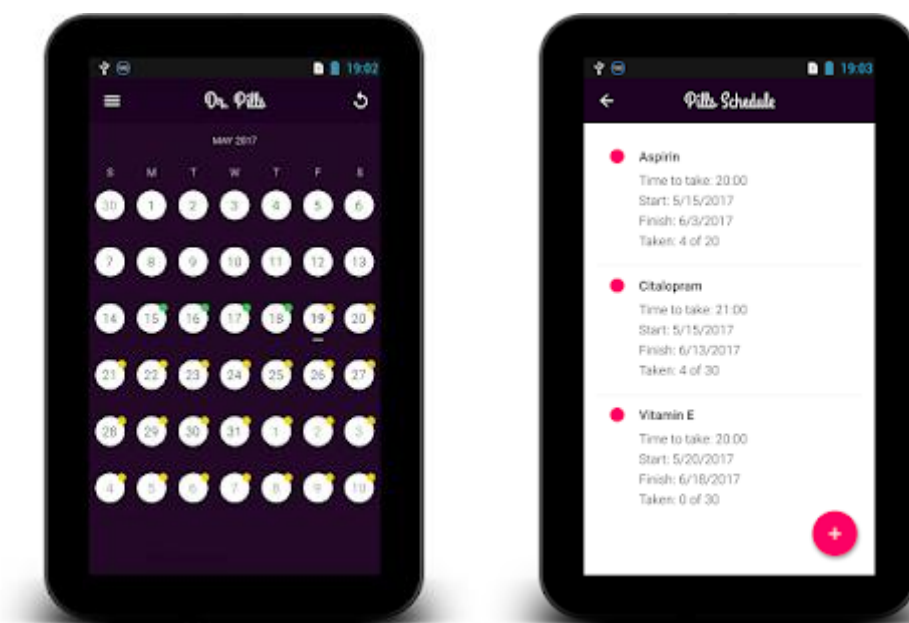
Projetado para lembrar o usuário em momentos específicos, o aplicativo registra lembretes com a hora de início e término do tempo especificado para tomar um determinado medicamento, persistindo se necessário, até que o usuário tenha tomado seu remédio.

Entre suas funcionalidades, demonstradas na Ilustração 2, destaca-se um cronograma de medicamentos integrado a um calendário que permite uma

visualização abrangente do período de tratamento e dos remédios cadastrados, proporcionando maior controle e organização das informações necessárias para cada tratamento adicionado no aplicativo.

Além da opção de calendário e cadastro de informações, há também a possibilidade de edição, adição e remoção de outras prescrições, lembretes de recarga de medicamentos, diário de medicação e, em um plano pago, lembretes para medicação de familiares e animais de estimação.

**Ilustração 2** – Funcionalidades dentro do *app Medication Reminder*.



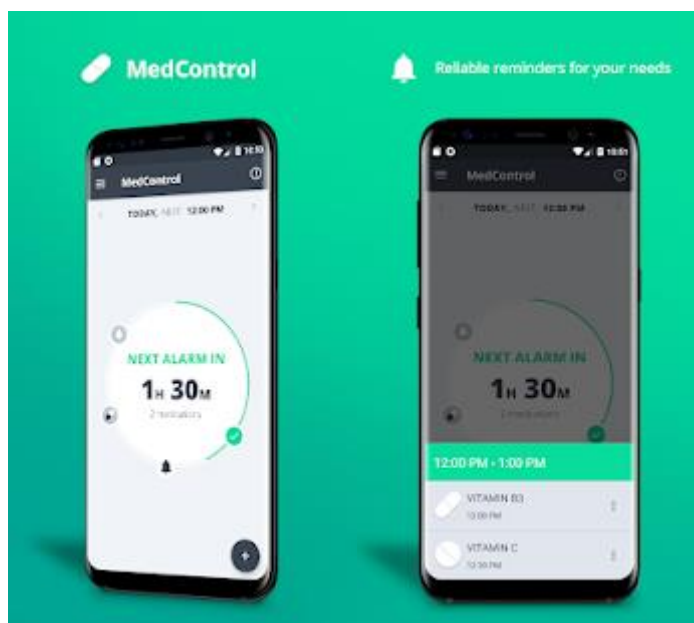
**Fonte:** Google Store.

- **Aplicativo 3:** *Pill Reminder & Medication Tracker*

O aplicativo "*Pill Reminder & Medication Tracker*" (em português: "Lembrete de Pílulas e Acompanhamento de Medicamentos") é uma aplicação estrangeira com o propósito de auxiliar no acompanhamento do uso de medicamentos. Seu objetivo é evitar que o usuário se esqueça de tomar um remédio ou tome uma dose incorreta, o que poderia colocar sua saúde em risco.

Entre as principais funcionalidades, conforme mostra a Ilustração 3, destaca-se o uso de alarmes como lembretes, fornecendo informações ao usuário sobre a quantidade de medicamentos a serem tomados. Além disso, o aplicativo possui um cronômetro para calcular o tempo restante até a próxima dose do medicamento cadastrado e, também, um organizador de medicamentos, que funciona como um histórico dos remédios tomados, permitindo o acompanhamento do usuário.

**Ilustração 3** – Funcionalidades dentro do app *Pill Reminder & Medicine App*



**Fonte:** Google Store.

## **2 DESENVOLVIMENTO**

Este capítulo apresenta o processo de desenvolvimento do aplicativo para cadastro de medicamentos, exames e consultas médicas para pessoas idosas, bem como uma análise do sistema e seus requisitos.

### **2.1 ANÁLISE DO SISTEMA**

O primeiro passo do desenvolvimento do aplicativo foi a definição de suas funcionalidades necessárias. A mais notável entre elas sendo a implementação de um calendário, dividido por mês, que permite que o usuário denote dias específicos em que um determinado registro irá ocorrer, listando na mesma página todas as marcações para aquele mês.

#### **FUNCIONALIDADES**

Um registro pode ser criado para uma data escolhida, devendo ser de um dos tipos oferecidos pela aplicação: uma consulta, exame, medicamento ou lembrete genérico — possibilitando o usuário a descrever livremente o que necessita lembrar.

Uma vez criado esse registro, ele é disposto em formato de um "cartão", referenciando o dia daquele mês, e apresentando as demais informações mais relevantes, dependendo do tipo. Por exemplo, um registro de consulta irá informar o dia da consulta, horário, o nome do médico e sua especialização, enquanto um registro de medicamento tem um ícone indicador de medicamento, bem como sua frequência diária em horas (e.g. de 3 em 3 horas).

## **ACESSIBILIDADE E *USER EXPERIENCE* (UX)**

Devido ao público-alvo do aplicativo ser de uma faixa etária mais alta e de esperada menor experiência tecnológica, também foi dada ênfase à usabilidade e intuitividade do aplicativo. A usabilidade de um *software* é definida como:

“Um conjunto de atributos de *software* relacionado ao esforço necessário para seu uso e para o julgamento individual de tal uso por determinado conjunto de usuários.” (ISO/IEC 9126-1991)

Para o desenvolvimento de um sistema acessível, é necessário levar em consideração vários fatores, dentre eles “o grau de familiaridade do usuário médio do site com o uso da internet e suas ferramentas” (BRITO, Keila. 2011). Observação relevante ao *Caretaker*, cujo público de idade avançada necessita que o sistema acomode usuários com baixa experiência no uso de *softwares* e aplicativos.

Dentre os atributos, foram implementados elementos textuais grandes e componentes com cores de alto contraste para garantir legibilidade, elementos maiores para garantir facilidade do manuseio e seleção, e o uso de ícones junto à descrições explicativas, detalhando as utilidades referenciadas.

## ***REACT NATIVE***

A utilização de JS para o desenvolvimento da parte visual da aplicação se dá devido à sua alta disponibilidade, posto que “*JavaScript* está presente em praticamente todos os navegadores e, por isso, está disponível em quase todos os aparelhos” (HAVERBEKE, 2018).



Entre os *frameworks* que utilizam JS, o RN foi escolhido devido a ser voltado justamente à criação de aplicações em sistemas móveis, bem como à popularidade do *NodeJS*, utilizado para seus ambientes de desenvolvimento. Nesse *framework*, os componentes são divididos em arquivos apenas do tipo JS, cujo código é então “traduzido” em *Hypertext Markup Language (HTML)* para visualização no aparelho.

Neles, são elaborados os elementos e a lógica, permitindo a implementação de funções e variáveis para manipulação de dados e controle da renderização na página, além de estilização baseada na linguagem *Cascading Style Sheets (CSS)*, uma vez que o RN permite a criação de estilos por meio de objetos JS.

*“For the render stage, the developer returns HTML markup from a React component’s render method, which React then renders directly into the page as necessary.” (EISENMAN, 2015).*

A facilidade oferecida pelo RN, através do *Node*, para o desenvolvimento de aplicações *mobile* é demonstrada pela possibilidade de atualização “em tempo real” do aplicativo durante o desenvolvimento, com as mudanças do código sendo refletidas em um servidor local de modo automático.

*“(...) because React Native is “just” JavaScript, you don’t need to rebuild your application in order to see your changes reflected; instead, you can hit Command+R to refresh your application just as you would any other web page.” (EISENMAN, 2015).*

Um dos componentes principais da aplicação é o de calendário, elaborado para listar todas as consultas e exames marcados para determinado mês, indicando em cada registro o dia, o tipo (consulta, exame, medicamento ou lembrete), o nome do médico (caso consulta) e o horário marcado, uma vez conectado à *API* para obter as informações registradas e apresentá-las na página para o usuário.

Essa conexão é feita de forma assíncrona, por meios de *requests* utilizando biblioteca baseada em AJAX (*Asynchronous JavaScript And XML*), permitindo a alteração das informações após o carregamento da página através de funções de *callback* para essas requisições, bem como o seu *update* uma vez recebida a resposta do servidor, sem a necessidade de um novo carregamento, tudo isso ocorrendo “por trás dos panos” (W3Schools).

“Mas para um sistema baseado em JavaScript, eu poderia afirmar que esse estilo de assincronia com *callback* é uma escolha sensata. Uma das forças do JavaScript é sua simplicidade, (...) Embora os *callbacks* não tendem a ser códigos simples, como conceito, eles são agradavelmente simples e ainda assim poderosos o suficiente para escrever servidores web de alta performance.” (HAVEBERKE, 2018)

No caso do uso de *React Native*, a página é renderizada novamente de modo automático, após qualquer alteração (*update*) é realizada. Isso segue o “ciclo de vida do componente”, utilizado tanto pelo *ReactJS* quanto RN, que ocorre dependendo do uso de *states* (estados) e *props* (propriedades), elementos típicos *React*, responsáveis pela manipulação de variáveis e que permitem a comunicação entre eles. Esse ciclo de vida segue a construção, renderização, alteração e destruição de um determinado componente.

“There are two types of data that control a component: *props* and *state*. *props* are set by the parent and they are fixed throughout the lifetime of a component. For data that is going to change, we have to use *state*.” (React Native Docs)

Além disso, é possível “reciclar” componentes, usando-os como “modelos”, para definição de classes frequentemente utilizadas, que podem ser importadas para uso dentro de qualquer outro componente, apenas determinando algumas propriedades específicas para cada novo uso (e.g. um botão pode ter apenas um ícone, ou ícone e título, ou ícone, título e subtítulo, dependendo da página e seu

propósito, sendo isso determinado pelas *props* passadas à esse componente modelo).

*“Your own components can also use props. This lets you make a single component that is used in many different places in your app, with slightly different properties in each place by referring to props in your render function.”*  
(React Native Docs)

## **FLASK**

Por meio do *SQLAlchemy*, a comunicação da *API* com o banco de dados se mostrou mais fácil com o uso de *OOAD (Object Oriented Analysis and Design)*, a modelagem orientada à objetos, que permite facilidade na manipulação de dados.

*“(...) motivaram a busca por um novo paradigma cujos benefícios se aliassem às conquistas realizadas, até então, no intuito de transformarem o desenvolvimento de sistemas em uma atividade de engenharia. (...) OO [Análise e Modelagem Orientadas a Objetos] permite, portanto, reunir, em um mesmo conceito (...), dados e operações.”* (PEREIRA, Luiz Antônio de Moraes, 2011).

A *API* foi elaborada para criar e conectar-se com o banco de dados, feito em *MySQL*, permitindo o uso dos seguintes tipos de requisições *HTTP (Hypertext Transfer Protocol)*:

- **GET:** utilizado para a recuperação dos dados cadastrados pelo usuário dentro do aplicativo. Demonstrado na Ilustração 4, a rota obtém todos os cadastros relacionados ao usuário e os devolve em uma lista de objetos.

**Ilustração 4** – Rota do tipo *GET* para a página de Agenda

```

@app.route("/calendario/<user_id>", methods=["GET"])
@jwt_required()
def get_data(user_id):
    user = User.find_user_by_id(user_id)

    if not user:
        return {"message": "Usuário não encontrado!"}, 404

    fetched_data = []
    medications = Medication.find_all(user_id)
    for m in medications:
        fetched_data.append({
            'usuario_id': m.usuario_id,
            'item_id': m.id,
            'titulo': m.nome,
            'frequencia': m.frequencia_horas,
            'descricao': m.obs Medicamento,
            'dose': m.dosagem,
            'qt medicamento': m.qt medicamento,
            'local': '',
            'data': '',
            'horario': '',
            'tipo': 'medicamento'
        })
    exams = Exam.find_all(user_id)
    for e in exams:
        fetched_data.append({
            'usuario_id': e.usuario_id,
            'item_id': e.id,
            'titulo': e.exame,
            'frequencia': '',
            'descricao': e.medico,
            'local': e.local,
            'data': e.data,
            'horario': e.horario,
            'tipo': 'exame'
        })
    appointments = Appointment.find_all(user_id)
    for a in appointments:
        fetched_data.append({
            'usuario_id': a.usuario_id,
            'item_id': a.id,
            'titulo': a.nome,
            'frequencia': '',
            'descricao': a.descricao,
            'local': '',
            'data': a.data,
            'horario': a.horario,
            'tipo': 'consulta'
        })
    reminders = Reminder.find_all(user_id)
    for r in reminders:
        fetched_data.append({
            'usuario_id': r.usuario_id,
            'item_id': r.id,
            'titulo': 'Lembrar',
            'frequencia': '',
            'descricao': r.descricao,
            'local': '',
            'data': a.data,
            'horario': a.horario,
            'tipo': 'lembrete'
        })

    return {"message": "success", "fetched_data": fetched_data}, 200

```

*Fonte: Autoras, 2023*

- **POST:** utilizado para o cadastro de informações do usuário e de dados relevantes para o seu monitoramento através da interface do aplicativo, ao final

já retornando as informações necessárias para a realização do *login* do usuário, vide Ilustração 5.

**Ilustração 5** – Rota do tipo *POST* para o cadastro de usuário

```
@app.route("/usuario/registrar", methods=["POST"])
def register():

    user_email = User.find_user_by_email(email=request.json["email"])
    user_username = User.find_user_by_username(username=request.json["username"])

    if user_email:
        return {"message": "Esse email já está sendo utilizado. Se você já tiver uma conta, use a opção de 'Entrar'."}, 404

    if user_username:
        return {"message": "Esse nome de usuário já está sendo utilizado. Escolha outro nome de usuário (você pode colocar números ou símbolos (@#$%*) para deixá-lo mais único."}, 404

    new_user = User(
        request.json["username"],
        '', # nome
        request.json["email"],
        request.json["senha"],
        '', # nascimento
    )

    try:
        new_user.save_to_db()
    except Exception as err:
        return {"message": f"Error while register new user: {err}"}, 404

    access_token = create_access_token(identity=new_user.email)
    response = {}
    response["access token"] = access_token
    response["usuário"] = {
        "id": new_user.id,
        "username": new_user.username,
        "nome": new_user.nome,
        "email": new_user.email,
        "nascimento": new_user.nascimento,
    }
    return response, 200
```

*Fonte: Autoras, 2023*

- **PUT:** rota utilizada para atualização de dados já cadastrados no sistema, como por exemplo a alterações de informações do usuário, demonstradas na página de perfil, conforme exibido na Ilustração 6.

**Ilustração 6** – Rota do tipo *PUT* para atualização de um usuário

```
@app.route("/medicamento/<med_id>", methods=["PUT"])
@jwt_required()
def update_medication(med_id):
    token_email = get_jwt_identity()
    user = User.find_user_by_email(email=token_email)
    if not user:
        return {"message": "Usuário não encontrado!"}, 404

    medication = Medication.find_medication_by_id(med_id)

    if not medication:
        return {"message": "Medicamento não encontrado!"}, 404

    if not medication.usuario_id == user.id:
        return {"message": "Você não pode alterar essa informação!"}, 403

    medication.nome = request.json["nome"]
    medication.dosagem = request.json["dosagem"]
    medication.qt_medicamento = request.json["qt_medicamento"]
    medication.obs_medicamento = request.json["obs_medicamento"]
    medication.frequencia_horas = request.json["frequencia_horas"]

    try:
        medication.save_to_db()
    except Exception as err:
        return {"mensagem": f"Erro ao atualizar medicamento: {err}"}, 400

    return {"message": "medication updated succesfully!"}, 200
```

*Fonte: Autoras, 2023*

- **DELETE:** rota utilizada para a remoção de dados cadastrados pelo usuário dentro da aplicação, uma vez encontrados, deletados permanentemente do banco de dados, bem como mostra a Ilustração 7.

**Ilustração 7** – Rota do tipo *DELETE* para a remoção de medicamento

```
@app.route("/exame/<exame_id>", methods=["DELETE"])
@jwt_required()
def delete_exam(exame_id):
    token_email = get_jwt_identity()
    user = User.find_user_by_email(email=token_email)
    if not user:
        return {"message": "Usuário não encontrado!"}, 404

    exam = Exam.find_exam_by_id(exame_id)

    if not exam.usuario_id == user.id:
        return {"message": "Você não pode deletar essa informação!"}, 403

    if exam:
        exam.delete_from_db()
        return {"message": "Exame deletado com sucesso!"}, 200

    return {"message": "Error while delete exam"}, 400
```

*Fonte: Autoras, 2023*

## **JSON WEB TOKEN**

Além das rotas listadas, todos os acessos às funcionalidades da *API* são realizados através do uso de *JWT* para método de segurança e de autenticação de usuários, que funciona de maneira a retornar um *token* assim que o usuário realiza o *login* no aplicativo. Com esse token retornado, ele é sempre enviado com as requisições à *API*, sendo a autenticação dos dados inseridos realizada e a permissão concedida.

“Com a utilização do *JSON Web Token* no controle de usuários em *APIs REST* é possível garantir que o usuário que faz a solicitação de dados possui os níveis de acesso necessário para a leitura daquela informação, e sem a necessidade de consultar as informações e credenciais de acessos do usuário no banco de dados em toda requisição, uma vez que o próprio token é capaz de identificar o usuário e seu nível de acesso, reduzindo assim o processamento do servidor”. (MONTANHEIRO, Lucas Souza. et. al, 2017)

Essa restrição é implementada para proteger os dados pessoais e sensíveis dos usuários, e também impedindo que outrem consigam visualizar, alterar ou eliminar dados que não lhes pertencem.

Para a manipulação dos dados por parte da *API*, o *back-end* utiliza modelos das tabelas do banco de dados, garantindo assim fácil acesso nas *views* (funções presentes no *framework* criadas para responder às requisições feitas à *API*, em formato *JavaScript Object Notation*, ou, “*JSON*”) para a manipulação dos dados.

### **2.1.1 ANÁLISE DE REQUISITOS**

Adiante, são apresentados os requisitos identificados para o desenvolvimento do aplicativo *Caretaker*. O processo de análise compreendeu a identificação dos requisitos funcionais e não funcionais, bem como sua priorização de acordo com a relevância para os usuários do aplicativo.

#### **REQUISITOS FUNCIONAIS**

Requisitos funcionais são as especificações que descrevem as funções ou tarefas que o sistema deve ser capaz de executar, a fim de atender as necessidades dos usuários. Esses requisitos descrevem o comportamento esperado do sistema em resposta a entradas específicas, bem como as saídas resultantes dessas entradas, e “dependem do tipo de software a ser desenvolvido, de quem são seus possíveis usuários e da abordagem geral adotada pela organização ao escrever os requisitos” (SOMMERVILLE, 2011). Em outras palavras, os requisitos funcionais definem o que o sistema deve fazer para cumprir as expectativas do usuário.

Para o *Caretaker*, os requisitos funcionais são:



**Tabela 1** – Requisitos Funcionais do aplicativo *Caretaker*

[RF01] O sistema deve permitir organização através de calendário.
[RF02] O sistema deve permitir que o usuário cadastre medicações.
[RF03] O sistema deve permitir que o usuário insira informações sobre consultas médicas.
[RF04] O sistema deve permitir que o usuário monitore as informações adicionadas.
[RF05] O sistema deve permitir a configuração de lembretes para informações adicionais.
[RF06] O sistema deve permitir que o usuário insira exames realizados.
[RF07] O sistema deve permitir controle de rotina.
[RF08] O sistema deve dividir registros em categorias (exames, consultas etc).
[RF09] O sistema deve organizar cadastros com datas específicas para o mês e ano relevantes.
[RF10] O sistema deve mostrar as informações de medicamentos cadastrados com sua frequência e dosagem.

*Fonte: Autoras, 2023*

## REQUISITOS NÃO FUNCIONAIS

Requisitos não funcionais “não estão diretamente relacionados com os serviços específicos oferecidos pelo sistema a seus usuários” (SOMMERVILLE, 2011). Estes são as especificações que descrevem os critérios que o sistema deve atender em termos de desempenho, segurança, usabilidade, entre outros aspectos que não sejam necessariamente relacionados às funções do sistema. Esses requisitos estabelecem as expectativas de qualidade do sistema em relação à sua eficiência, capacidade de suportar cargas de trabalho, segurança de dados, facilidade de uso, entre outras características relevantes para o usuário.

Para o *Caretaker*, os requisitos não funcionais são:

**Tabela 2** – Requisitos Não Funcionais do aplicativo *CareTaker*

[RNF01] O sistema deve ser desenvolvido utilizando o framework <i>React Native</i> .
[RNF02] O sistema deve ter design <i>clean</i> e de fácil visibilidade para acessibilidade.
[RNF03] O sistema deve ter navegabilidade simples e intuitiva.
[RNF04] O sistema deve utilizar de senha para proteger informações sensíveis.
[RNF05] O sistema acessar salvar algumas informações localmente para controle de login.

*Fonte: Autoras, 2023*

### 2.1.2 DIAGRAMA DE CASO DE USO

Com a confecção de um diagrama de caso de uso, foi criado de forma gráfica e semântica toda a parte de interação do usuário para o sistema desenvolvido, gerando possíveis rotas iniciais e finais, e apresentando as funcionalidades principais da aplicação.

Isso tem por objetivo representar os comportamentos viáveis, delineando o funcionamento do sistema durante todo o processo desde a provação inicial do usuário ao sistema até o fim esperado da ação.

“Outro aspecto relacionado é que um modelo apresenta apenas uma visão ou cenário de um fragmento do mundo real. É por conseguinte necessário a produção de vários modelos de forma a melhor representar e compreender o sistema correspondente. Por exemplo, a construção de uma obra de engenharia civil apresenta inúmeros modelos, cada qual correspondendo a uma interpretação ou visão da mesma “realidade”. (SILVA, Alberto. et. Al, 2001)

No caso do *Caretaker*, as interações performadas pelo usuário com o sistema se concentram, principalmente, na parte de registro, edição e remoção de informações, sendo que os caminhos referentes a essas interações se conectam com

a parte de agenda, responsável pela listagem e organização das informações adicionadas.

O Diagrama de Caso de Uso pode ser consultado no Apêndice A.

### 2.1.3 FLUXO DE EVENTOS

No âmbito do desenvolvimento do aplicativo, foram criados fluxos de eventos para guiar o usuário desde o início até o final do processo, contemplando tanto o fluxo principal quanto fluxos alternativos. Essa estruturação considerou o ponto de vista do usuário e previu as diversas possibilidades de comportamento.

No *Caretaker*, o fluxo de eventos está fortemente relacionado ao registro, alteração e remoção de diferentes tipos de informações, que são acessíveis somente mediante um *login*, obtido após o cadastro prévio. Ademais, as informações registradas são exibidas na área de agenda e podem ser manipuladas conforme a necessidade do usuário.

A descrição completa dos fluxos de eventos encontra-se detalhada no Apêndice B.

## 2.2 BANCO DE DADOS

A concepção do Banco de Dados envolve cinco tabelas: **Usuário**, **Medicamento**, **Consulta**, **Exame** e **Lembrete**. Esse modelo simplificado foi elaborado uma vez que a interação do usuário ocorre somente entre ele e o sistema, jamais necessitando da intervenção ou mistura de informações de terceiros.

Desse modo, as tabelas se relacionam através do uso de chaves estrangeiras, com os *ids* de um registro estabelecendo sua conexão a outro, necessitando apenas da chave primária para de imediato obter as informações de uma outra tabela. A Ilustração 8 assim demonstra:

**Ilustração 8** – Tabelas da Aplicação *Caretaker*

Usuário	Exame	Medicamento	Consulta	Lembrete
ID_Usuario (PK)	ID_Exame (PK)	ID_Medicamento (PK)	ID_Consulta (PK)	ID_Lembrete (PK)
nm_Username	ID_Usuario (FK)	ID_Usuario (FK)	ID_Usuario (FK)	ID_Usuario (FK)
nm_Nome	nm_Medico	ds_Nome_Medicamento	nm_Nome_Medico	ds_Descricao
ds_Email	ds_Exame	ds_Dosagem	ds_Descricao	dt_Data
ds_Senha	ds_Local	qt_Qt_Medicamento	dt_Data	hr_Horario
dt_Nascimento	dt_Data	ds_Obs_Medicamento	hr_Horario	
	hr_Horario	qt_Frequencia_Horas		

**Fonte:** Autoras, 2023

Por exemplo, cada usuário possui um *id* que é sua chave primária. Dessa maneira, essa será a chave estrangeira em todas as outras tabelas, relacionando medicamentos, consultas, exames e lembretes à um determinado usuário.

*“When converting an ER model to a database schema, we work through each entity and then through each relationship according to the following rules to end up with a set of database tables. (...) For each strong entity, create a table comprising its attributes and designate the primary key. The parts of any composite attributes are also included here. For each weak entity, create a table comprising its attributes and including the primary key of its owning entity. The primary key of the owning entity is known as a foreign key here, because it’s a key not of this table, but of another table.”*  
(TAHAGHOGLI. et. al, 2006)

A partir disso foi criado o Modelo Entidade Relacionamento (MER) que encontra-se no Apêndice C.

## 2.3 CAMADA DE NEGÓCIO

Para o desenvolvimento de uma das principais funcionalidades da aplicação, a de disposição de informações cadastradas no calendário (referido como a página “Agenda”), onde o usuário pode guardar todos os seus lembretes, faz-se uso de um componente específico, que em uma camada interna utiliza de *states* para manipulação de seus dados, que poderão também ser passados como *props* uma vez que o calendário é importado na Agenda, a fim de realizar a comunicação direta com o componente específico e persistir os estados dos dados obtidos.

Essa situação é mais bem exemplificada na opção de “editar”, onde as informações de um determinado lembrete são passadas como *props* e transformadas em *states*, para que o seu acesso seja realizado sem a necessidade de uma nova chamada à *API*. Ao clicar na opção de editar, as informações daquele elemento são passadas ao componente principal do aplicativo e em seguida salvas em um *state*. Sendo o componente hierarquicamente superior, ele então pode passar esse objeto como uma *prop* para qualquer outro componente da aplicação.

Como pode ser observado na Ilustração 9, o construtor do componente da tela de formulário de medicamentos apresenta seu estado inicial, com os valores vazios no qual a página deve começar e, abaixo da declaração de funções, uma condição em que uma *prop* é passada com as informações do objeto escolhido para edição. As informações dessa *prop* são, em seguida, integradas ao *state*, para que sejam dispostas ao usuário.

**Ilustração 9** – Código do construtor do formulário de medicamento

```
class MedForm extends Component{
  constructor(props){
    super(props);
    this.state = {
      medicamento: '',
      dosagem: '',
      qt: 0,
      freq: 0,
      tratamento: '',
    };

    this.handleNum = this.handleNum.bind(this);
    this.validate = this.validate.bind(this);
    this.saveChanges = this.saveChanges.bind(this);
    this.create = this.create.bind(this);
    this.resetState = this.resetState.bind(this);

    if (this.props.editObj) {
      const obj = this.props.editObj;

      this.state.medicamento = obj.titulo;
      this.state.dosagem = obj.dose;
      this.state.qt = obj.qt_medicamento;
      this.state.freq = obj.frequencia;
      this.state.tratamento = obj.descricao;
      this.state.usuario_id = obj.usuario_id;
      this.state.item_id = obj.item_id;
    }
  }
}
```

**Fonte:** Autoras, 2023

Além disso, no componente referente à Agenda, foram implementados métodos para a obtenção de informações que se comunicam com a *API*. O método inicial é o *getData*, demonstrado na Ilustração 10, que através de requisição do tipo *GET*, junto ao *token* de acesso do usuário, realiza a aquisição de dados e o preenchimento das listas de lembretes auxiliando na listagem das informações na tela específica.

**Ilustração 10** – Requisição de dados na página de Agenda

```

getData() {
  const access_token = localStorage.getItem("access_token");
  let usuario = localStorage.getItem("usuario");
  if (!usuario || !access_token) {
    alert('Ocorreu um erro, por favor faça o login novamente!');
    this.props.logout();
    return;
  }

  usuario = JSON.parse(usuario);
  $.ajax({
    type: 'GET',
    url: 'http://127.0.0.1:5000/calendario/' + usuario.id,
    headers: {
      "Content-Type": "application/json",
      "Authorization": `Bearer ${access_token}`,
    },
    data: {},
    success: function(data){
      this.setState({data: data.fetched_data});
      this.setDates(data.fetched_data);
    }.bind(this),
    error: function(data){
      if (data.status == 401) {
        alert('Sua sessão expirou, por favor faça o login novamente.')
        this.props.logout();
      }
      if (data.status == 404) {
        alert(data.responseJSON.message)
      }
    }.bind(this),
  });
}

```

**Fonte:** Autoras, 2023

Após resposta do servidor, o método *displayDateInfo*, vide Ilustração 11, funciona como um ordenador de informações de acordo com as suas respectivas datas. Nele, os dados recebidos do usuário são percorridos e é realizada separação por mês e atualização das listas pré-definidas, prontas para serem dispostas na tela da aplicação.

**Ilustração 11** – Função de organização de informações da Agenda

```
displayDateInfo(monthNumber) {
  let { data } = this.state;
  let dateInfo = [];
  for (let d of data) {
    let month key = Number(d.data.split('/')[1]);
    let current_month = monthNumber ? monthNumber : new Date().getMonth() + 1;

    if (month key === current_month) {
      dateInfo.push({
        date: d.data,
        id: d.item id,
        title: d.titulo,
        time: d.horario.slice(0,-3),
        desc: d.descricao,
        type: d.tipo,
        form: d
      })
    }

    if (d.tipo === 'medicamento') {
      dateInfo.push({
        date: d.data,
        id: d.item id,
        title: d.titulo,
        time: d.time,
        desc: d.descricao,
        dose: d.dose,
        qt: d.qt_medicamento,
        freq: d.frequencia,
        type: d.tipo,
        form: d
      })
    }
  }
  this.setState({dateInfo});
}
```

*Fonte: Autoras, 2023*

## 2.4 CAMADA DE APRESENTAÇÃO

Na primeira tela, estão dispostas opções distintas com botões para cadastro e *login* de usuário, uma permitindo que usuários que já possuem conta possam inserir suas informações para conseguir acesso (pelo tempo que sua sessão JWT for válida) e, em outra, um formulário de cadastro na plataforma, com instruções adicionais sobre os campos a serem preenchidos, conforme demonstrado na Ilustração 12.



**Ilustração 12** – Tela de cadastro da aplicação

**CARETAKER**  
cuide da sua saúde

← VOLTAR

**Nome de usuário:**  
O nome de usuário é um apelido utilizado para entrar na sua conta, você pode usar números ou símbolos (@#&\*) para deixá-lo mais único.

**E-mail:**

**Senha:**  
Cuidado, apenas informe sua senha para pessoas de total confiança. Você pode usar números ou símbolos para deixá-la mais segura.

**Confirmar Senha:**

**CADASTRAR**

INÍCIO AGENDA PERFIL OPÇÕES

**Fonte:** Autoras, 2023

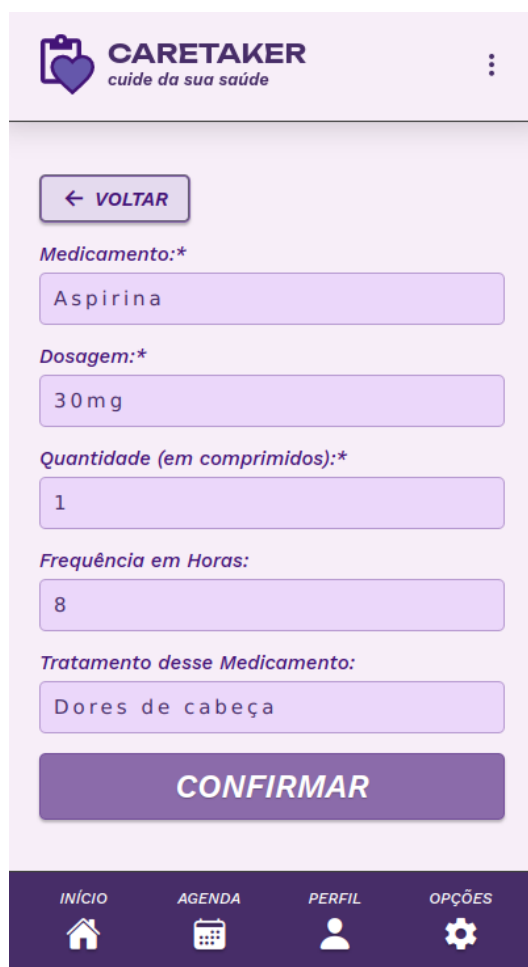
Para o *login*, foi implementado um formulário que deve ser preenchido com as informações já cadastradas anteriormente pelo usuário. Nela, há autenticação do *login* inserido, fazendo com que a aplicação realize uma requisição do tipo *GET* para validar o usuário e verificar se as informações são compatíveis, retornando um *token* de acesso.

A tela principal, ou a “Área do Paciente”, demonstrada na Ilustração 13, lista as opções de registro de informações do usuário. Ao clicar em uma delas, o usuário é levado ao formulário específico daquela informação, no qual ele deve preencher para que seja salvo no banco de dados. A exceção é a opção de acesso à Agenda, que leva o usuário a uma página que visualiza todos os lembretes, por mês e dia, oferecendo em cada um a possibilidade de editar e deletá-los.

**Ilustração 13** – Tela da “Área do Paciente”

**Fonte:** Autoras, 2023

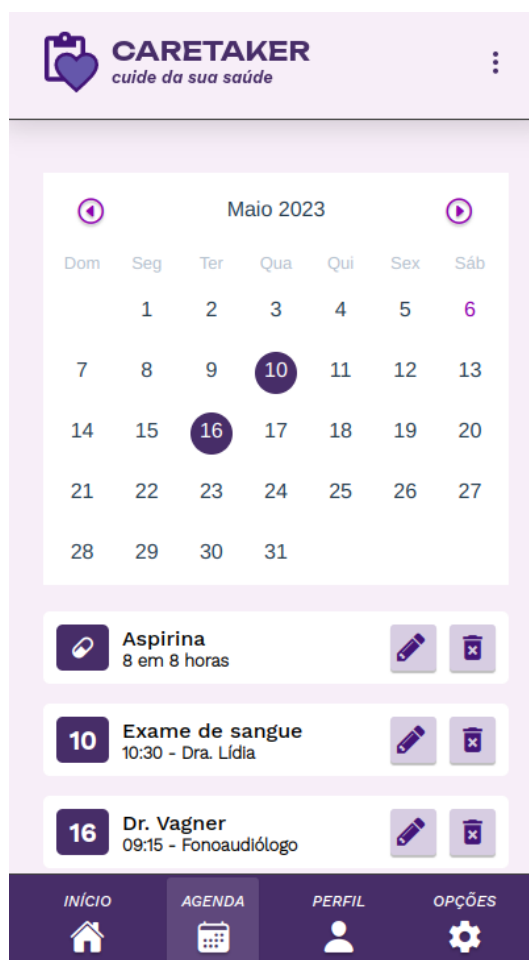
Para as telas de cadastro de medicamento, exame, consulta e lembretes, foram implementados formulários que possuem campos específicos, dependendo do tipo de cadastro que o usuário realizar, conforme exemplificado na Ilustração 14. Todos os cadastros são enviados ao banco de dados através de uma requisição *POST* e armazenados para serem listados como forma de lembrete na tela de calendário, de acordo com as suas respectivas datas.

**Ilustração 14** – Formulário de medicamento preenchido

The screenshot displays the 'CARETAKER' mobile application interface. At the top, the logo features a heart with a white pulse line, followed by the text 'CARETAKER' and the tagline 'cuide da sua saúde'. A menu icon is visible in the top right corner. Below the header, a '← VOLTAR' button is positioned at the top left. The form contains several input fields: 'Medicamento:\*' with 'Aspirina', 'Dosagem:\*' with '30mg', 'Quantidade (em comprimidos):\*' with '1', 'Frequência em Horas:' with '8', and 'Tratamento desse Medicamento:' with 'Dores de cabeça'. A large 'CONFIRMAR' button is centered at the bottom of the form. The bottom navigation bar includes four icons labeled 'INÍCIO', 'AGENDA', 'PERFIL', and 'OPÇÕES'.

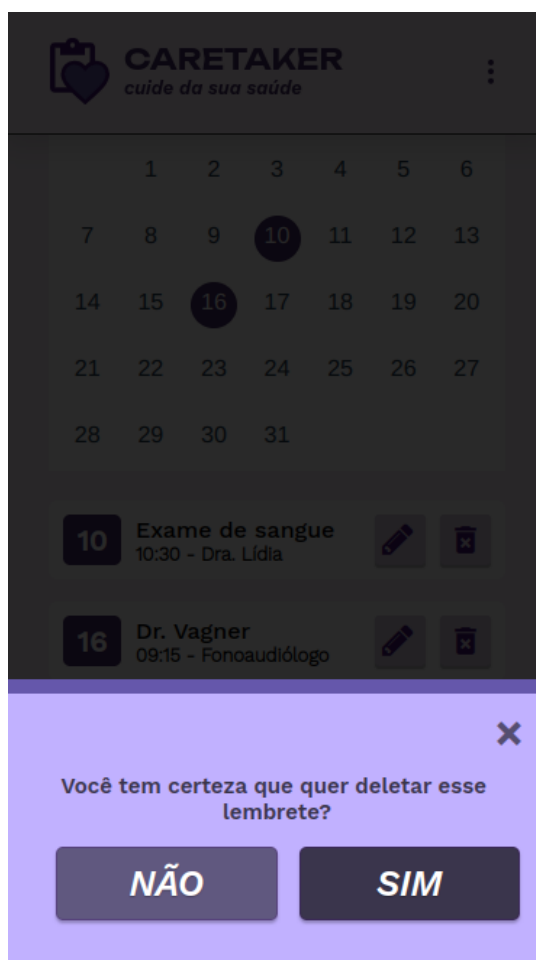
**Fonte:** Autoras, 2023

Já na tela de Agenda, vide Ilustração 15, foram implementados: o componente de calendário, através do qual estão marcadas as datas com algum lembrete, e listagem de todos os registros de informações oferecidas pelo usuário relevantes àquele mês. Cada lembrete possuindo opções de edição e remoção e, assim, podendo já performar requisições específicas para alteração no banco de dados.

**Ilustração 15** – Tela da página de Agenda

**Fonte:** Autoras, 2023

Ademais, como medida de precaução, assim demonstrado na Ilustração 16, uma mensagem é mostrada ao usuário ao tentar remover um lembrete, pedindo sua confirmação e assim garantindo total certeza da ação.

**Ilustração 16** – Confirmação de Ação

**Fonte:** Autoras, 2023

## 3 RESULTADO

Este capítulo apresenta os resultados dos testes realizados, evidenciando a usabilidade e a receptividade do aplicativo pelos usuários, além de identificar possíveis melhorias para aprimorar a experiência do público-alvo.

### 3.1 TESTES DE DESENVOLVIMENTO

Os testes de desenvolvimento são uma etapa crítica do processo de desenvolvimento de *software*, sendo particularmente importantes para garantir que o aplicativo seja confiável, seguro e capaz de atender às necessidades dos usuários. Com isso, para garantir a qualidade e o bom funcionamento do aplicativo, foram performados os seguintes tipos de testes ao longo da construção do *Caretaker*: testes de unidade, funcionais e de integração.

Em relação aos testes de unidade, diversas análises foram aplicadas em partes específicas do código, como funções e métodos individuais. Esses testes garantem que cada parte do aplicativo esteja funcionando corretamente antes que seja integrada com outras partes.

Os testes funcionais, por sua vez, foram focados em testar as funcionalidades do aplicativo, como o cadastro de medicamentos, consultas médicas, exames e lembretes. Eles ajudam a garantir que o aplicativo atenda às expectativas do usuário e funcione conforme o esperado.

### 3.2 TESTES DE INTEGRAÇÃO

Além dos testes principais, foram realizados testes de integração de ambas as partes que compõem o aplicativo, que ajudam a identificar problemas de comunicação e integração que podem surgir entre diferentes componentes do aplicativo.

Os testes realizados por usuários são essenciais para avaliar a usabilidade e eficiência de um aplicativo. No caso do *Caretaker*, foram selecionados seis usuários — na faixa etária de 65-75 anos e sem contato prévio com a aplicação — para testá-lo, com a finalidade de avaliar sua funcionalidade e identificar possíveis problemas e dificuldades de uso.

As tarefas designadas para os usuários foram: realizar cadastro, realizar *login*, cadastrar uma consulta, acessar a agenda e deletar um agendamento. Os resultados dessas tarefas foram examinados para determinar a facilidade de uso do aplicativo e identificar possíveis problemas de usabilidade.

Entre os usuários testados, um deles afirmou ter dificuldade no uso de tecnologia e preferir meios analógicos, enquanto os demais relataram utilizar aplicativos móveis com certa frequência, variando de uso diário a ocasional. Apesar dessas diferenças, todos os usuários relataram que o aplicativo era intuitivo e fácil de navegar. Em relação à realização de cadastros, a resposta foi mista, com metade dos usuários realizando a atividade sem problemas e a outra metade enfrentando algumas dúvidas.

Os usuários testados não tiveram dificuldades em enxergar todos os elementos, nem em ler os textos ou utilizar os botões do aplicativo. Ademais, todos consideraram o aplicativo útil e afirmaram que o utilizariam em seu dia a dia, caso lhes fosse oferecido como opção. O consenso entre os usuários foi de que o aplicativo

é bom em manter todas as informações de tratamento em um só lugar, sendo o calendário uma maneira fácil de acessá-las.

No entanto, um dos usuários relatou ter dificuldade em entender como preencher os formulários de cadastro de informações, citando ter dificuldade no geral na utilização de tecnologia. Relevante menção, visto que, durante o estudo, metade dos usuários relataram ter dúvidas quanto ao preenchimento desses formulários. Essa informação se prova valiosa para o desenvolvimento da aplicação, permitindo que sejam identificadas áreas de melhoria no aplicativo, a fim de torná-lo mais acessível para usuários que enfrentam desafios semelhantes.

Em resumo, os testes realizados pelos usuários indicam que o aplicativo em questão se mostra intuitivo, eficiente e útil para a maioria dos usuários. No entanto, sendo identificadas algumas áreas que carecem de melhorias no intuito de torná-lo mais acessível e fácil de usar para todos.

**Ilustração 17** – Alterações após *feedback*

**Dosagem:\***  
Dose do medicamento a ser tomada (em mg ou ml).

  
**Quantidade (em comprimidos):\***  
Somente número, que será equivalente à quantidade de comprimidos (ex. 2).  
**Frequência em Horas:**  
Somente número, que será equivalente à frequência em horas de tomar o medicamento (ex. 6).  
**Tratamento desse Medicamento:**  
O que esse medicamento está tratando (ex. dor de cabeça).

**Fonte:** Autoras, 2023



Utilizando de elemento já existente em um dos componentes da aplicação, foram adicionadas descrições instrucionais nos formulários de cadastro, indicando ao que se referem e como devem ser preenchidos os campos do aplicativo, como exemplo, a Ilustração 17 mostra as alterações realizadas especificamente no formulário de medicamentos, porém, alterações similares também foram feitas nas páginas de cadastro de exames e consultas.

### **3.3 CONCLUSÃO**

O objetivo desta tese foi desenvolver uma aplicação capaz de atender as necessidades de um público idoso, menos familiarizado com tecnologia e aplicações móveis. A proposta era criar um aplicativo acessível e intuitivo, de forma que aqueles que ainda utilizavam meios analógicos pudessem adotá-lo como uma alternativa viável.

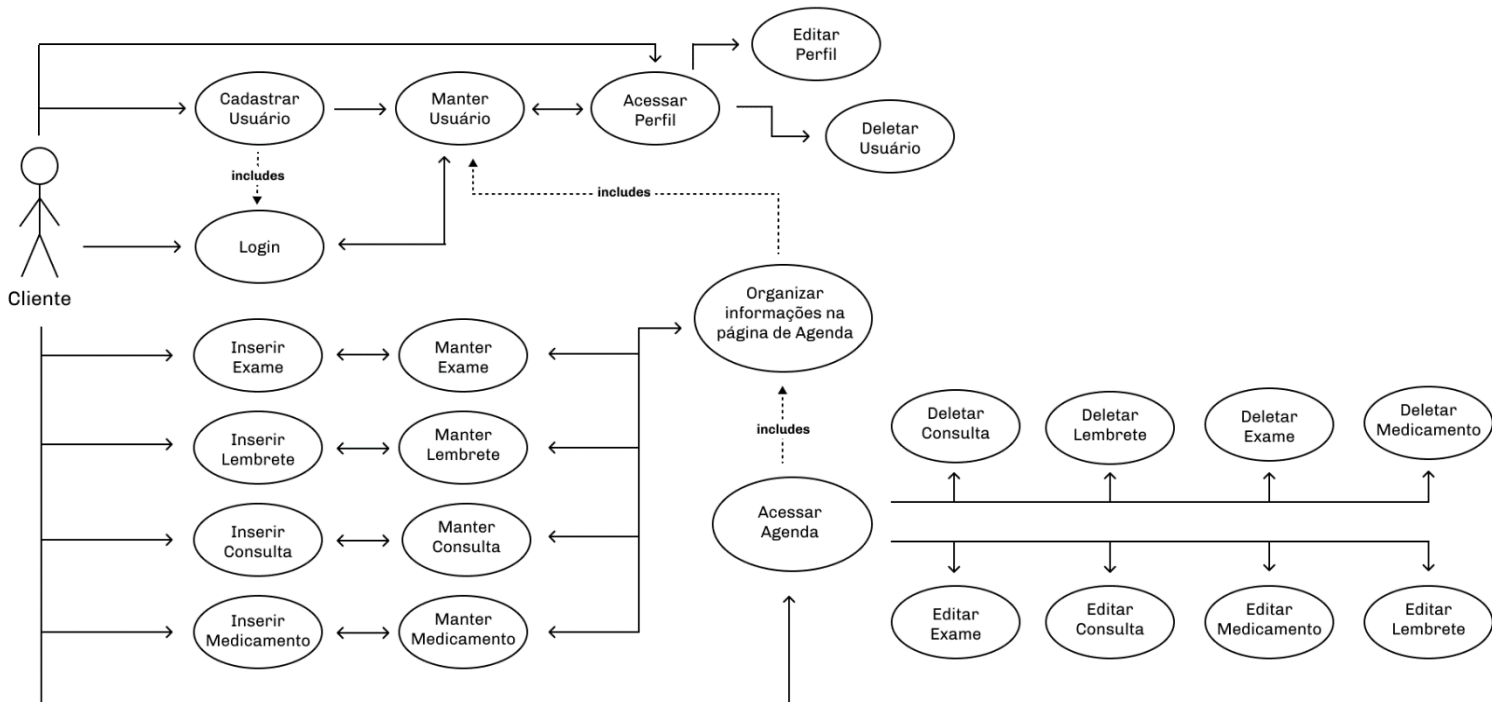
Os testes realizados com esse público confirmaram que o objetivo proposto foi alcançado, com alguns ajustes identificados para melhorar a facilidade de uso. O aplicativo foi bem recebido pelos usuários, demonstrando sua acessibilidade e intuitividade. No entanto, pequenas modificações tiveram de ser realizadas para aprimorar ainda mais a experiência do usuário e garantir que o aplicativo seja uma verdadeira alternativa digital aos usuários alvo.

## REFERÊNCIAS

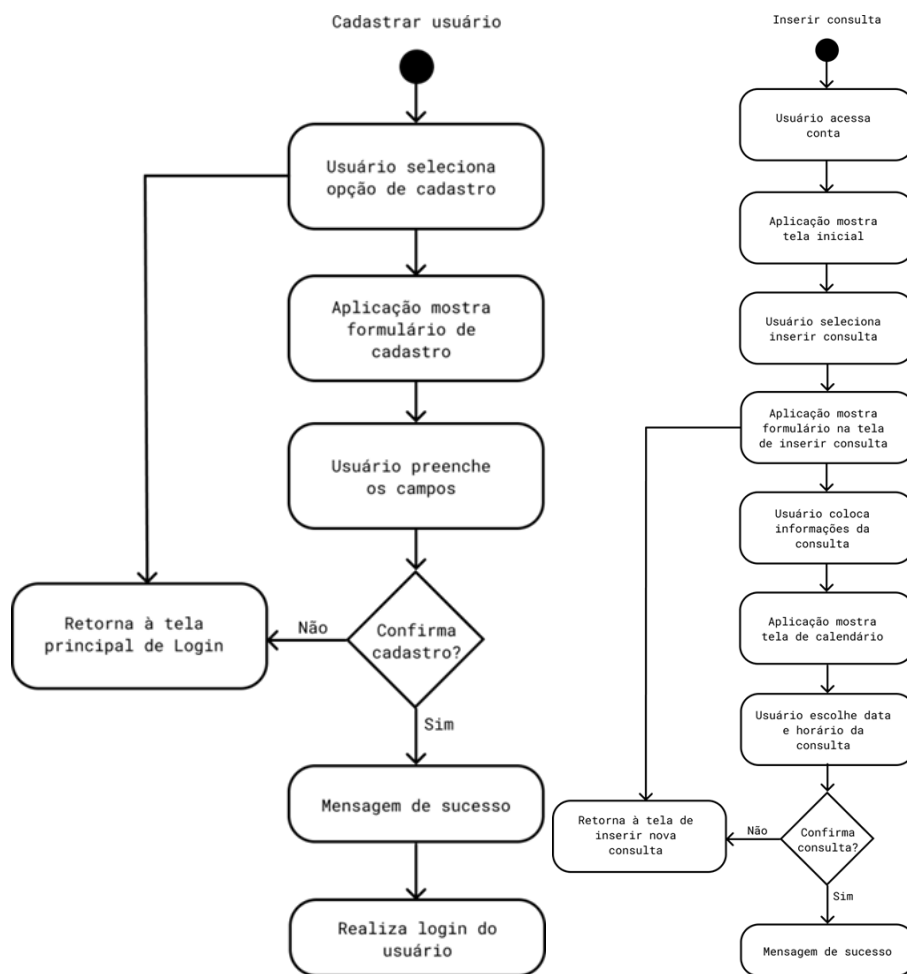
- **AJAX Introduction.** W3Schools. Disponível em: [https://www.w3schools.com/js/js\\_ajax\\_intro.asp](https://www.w3schools.com/js/js_ajax_intro.asp). Acesso em 28 maio 2023.
- AMORIM, Diane Nogueira Paranhos; SAMPAIO, Luisa Veríssimo Pereira; CARVALHO, Gustavo de Azevedo; VILAÇA, Karla Helena Coelho. **Aplicativos móveis para a saúde e o cuidado de idosos.** 1ª Edição, Brasília, 2018. Disponível em: <https://www.arca.fiocruz.br/bitstream/handle/iciict/25776/7.pdf?sequence=2&isAllowed=y>. Acesso em 18 de maio de 2023.
- BRITO, Keila. **Fundamentos do Desenvolvimento Web: Curso Técnico em Informática.** Colatina: CEAD / Ifes, 2011. Disponível em: [http://redeetec.mec.gov.br/images/stories/pdf/eixo\\_infor\\_comun/tec\\_inf/081112\\_fund\\_desenv.pdf](http://redeetec.mec.gov.br/images/stories/pdf/eixo_infor_comun/tec_inf/081112_fund_desenv.pdf). Acesso em: 18 maio 2023.
- EISENMAN, Bonnie. **Learning React Native: Building Native Mobile Apps with JavaScript.** 1ª Edição, Sebastopol/EUA, 2015. Disponível em: <https://pepa.holla.cz/wp-content/uploads/2016/12/Learning-React-Native.pdf> Acesso em: 30 maio 2023.
- Envelhecimento ativo: uma política de saúde / World Health Organization; tradução Suzana Gontijo. **Biblioteca Virtual em Saúde.** 1ª Ed. Brasília/DF, 2005. Disponível em: [https://bvsmis.saude.gov.br/bvs/publicacoes/envelhecimento\\_ativo.pdf](https://bvsmis.saude.gov.br/bvs/publicacoes/envelhecimento_ativo.pdf). Acesso em: 30 abr. 2022.
- HAVERBEKE, Manjin. **JavaScript Eloquente.** 2ª Edição, 2022. Disponível em: <https://github.com/braziljs/eloquente-javascript>. Acesso em: 14 maio 2023.
- **“Medication Reminder”.** Google Play Store: Enjoy millions of the latest Android apps, games, music, movies, TV, books, magazines & more. Disponível em: <https://play.google.com/store/apps/details?id=com.devsoldiers.calendar.pills.limit>. Acesso em: 30 abr. 2022.
- MONTANHEIRO, Lucas Souza; CARVALHO, Ana Maria Martins; RODRIGUES, Jackson Alves. **Utilização de JSON Web Token na Autenticação de Usuários em APIs REST.** 1ª Edição, Morrinhos, 2017. Disponível em: [https://www.enacomp.com.br/2017/docs/json\\_web\\_token\\_api\\_rest.pdf](https://www.enacomp.com.br/2017/docs/json_web_token_api_rest.pdf). Acesso em 28 de maio de 2023.
- **“MyTherapy”.** Google Play Store: Enjoy millions of the latest Android apps, games, music, movies, TV, books, magazines & more. Disponível em: <https://play.google.com/store/apps/details?id=eu.smartpatient.mytherapy> Acesso em: 30 abr. 2022.

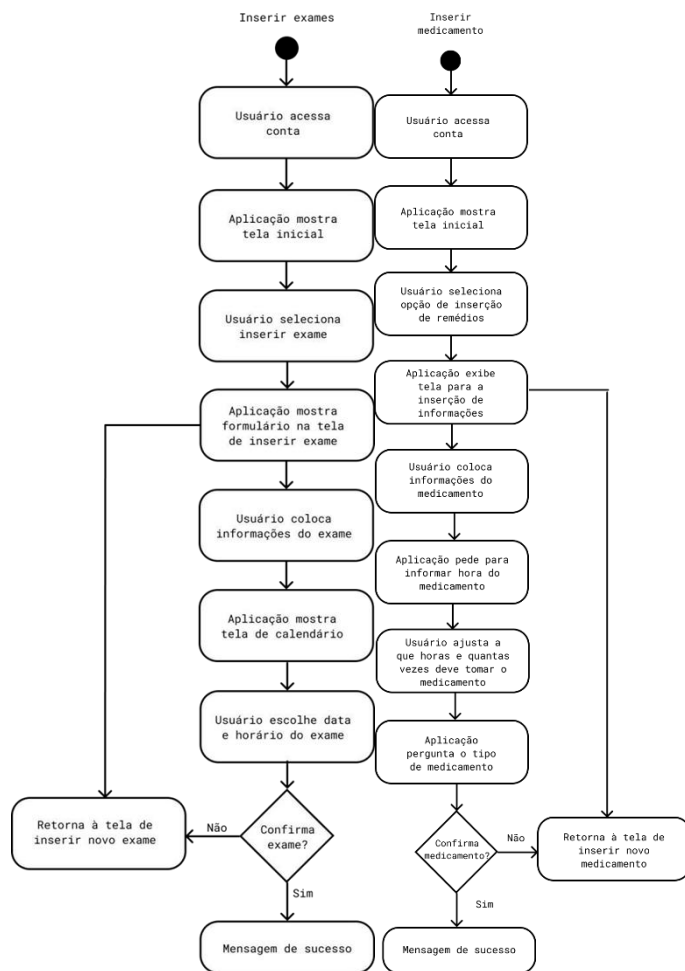
- PEREIRA, Luiz Antônio de Moares. **Análise e Modelagem de Sistemas com UML**. 1ª Edição, Rio de Janeiro, 2011. Disponível em: <https://luizantoniopereira.com.br/downloads/publicacoes/AnaliseEModelagemComUML.pdf>. Acesso em: 18 maio 2023.
- “**Pill Reminder & Medication Tracker**”. Google Play Store: Enjoy millions of the latest Android apps, games, music, movies, TV, books, magazines & more. Disponível em: <https://play.google.com/store/apps/details?id=app.medcontrol.alarm.pillreminder>. Acesso em: 30 abr. 2022.
- **REACT NATIVE DOCS**. Disponível em: <https://reactnative.dev/docs/> Acesso em 19 maio 2023.
- SILVA, Alberto Manuel Rodrigues da; VIDEIRA, Carlos Alberto Escalera. **UML, Metodologias e Ferramentas CASE**. 1ª Edição, Lisboa, 2001. Disponível em: [http://www.cesarkallas.net/arquivos/livros/informatica/UML\\_Metodologias\\_e\\_Ferramentas\\_CASE\\_portugues\\_.pdf](http://www.cesarkallas.net/arquivos/livros/informatica/UML_Metodologias_e_Ferramentas_CASE_portugues_.pdf). Acesso em 18 maio 2023.
- SOMMERVILLE, Ian. **Engenharia de Software**. Tradução da 9ª Edição, São Paulo, 2011. Disponível em: <https://www.facom.ufu.br/~william/Disciplinas%202018-2/BSI-GSI030-EngenhariaSoftware/Livro/engenhariaSoftwareSommerville.pdf>. Acesso em: 30 maio 2023.
- TAHAGHOGHI, Seyed M. M. “Saied”; WILLIAMS, Hugh E. **Learning MySQL**. 1ª Edição, Sebastopol/EUA, 2006. Disponível em: <http://160592857366.free.fr/joe/ebooks/ShareData/Learning%20MySQL.pdf>. Acesso em: 30 maio 2023.

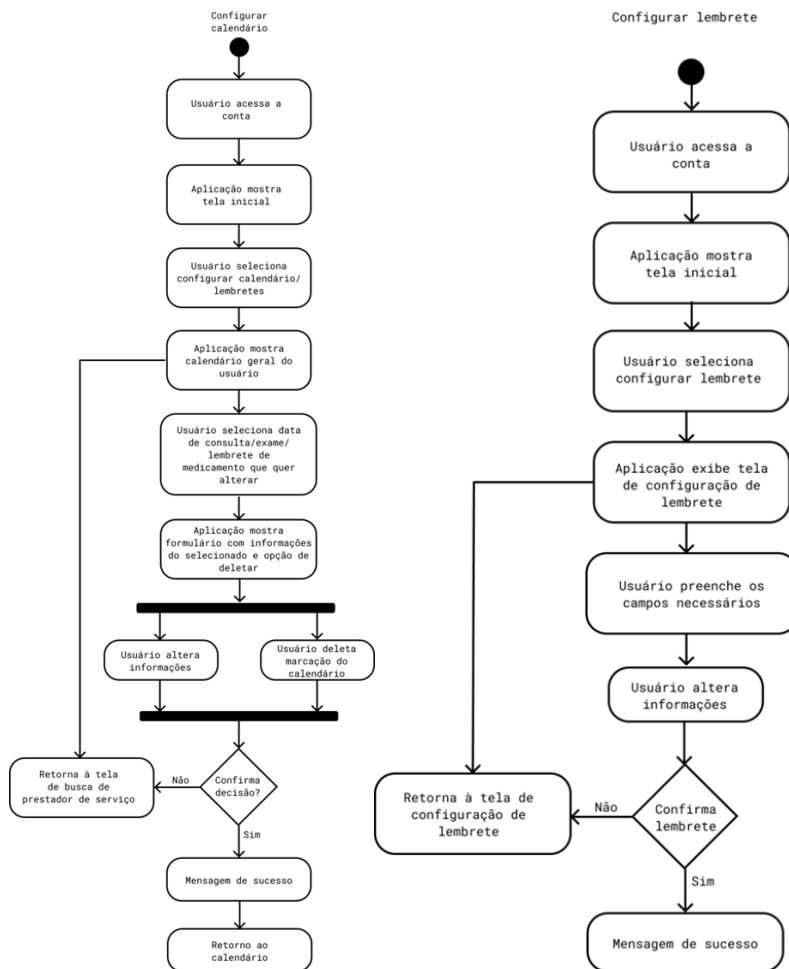
# APÊNDICE A



## APÊNDICE B







## APÊNDICE C

