

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DE SÃO PAULO**

Campus São João da Boa Vista

Trabalho Final de Curso

4º ano – Curso Técnico em Informática

Prof. Breno Lisi Romano e Prof. Luiz Angelo Valota Francisco

**O PROCESSO DE DESENVOLVIMENTO DAS
FUNCIONALIDADES DO MÓDULO DE ACOMPANHAMENTO
DE FAMILIARES DO PROJETO GERAÇÕES**

Aluno: Davi Martins Sassarão

Prontuário: 1620231

São João da Boa Vista – SP

2019

Resumo

O papel deste documento é mostrar o modo como foi desenvolvido o módulo de acompanhamento de familiares do sistema Gerações. Os requisitos levantados para o módulo, o diagrama de casos de uso que fora feito, as páginas protótipo e o desenvolvimento em si. Introduzindo os conceitos básicos acerca do projeto, suas motivações e contextos, as ferramentas e as linguagens de programação utilizadas ao longo do processo.

Sumário

1	Introdução	6
1.1	Contextualização e Motivação	6
1.2	Objetivo Geral	9
1.3	Objetivos Específicos	9
2	Desenvolvimento	10
2.1	Levantamento Bibliográfico	10
2.1.1	Introdução às linguagens de programação	10
2.1.2	Programação orientada a objetos	11
2.1.3	Linguagem PHP	14
2.1.4	<i>MySQL</i>	16
2.2	Desenvolvimento da Pesquisa	18
2.2.1	Requisitos do Módulo 03	18
2.2.2	Diagrama de Casos de Uso e suas iterações	21
2.2.3	Refatoração dos Protótipos para as Funcionalidades do Módulo 03	28
2.2.4	Exemplificação do desenvolvimento das funcionalidades das iterações a partir das páginas-protótipo	30
3	Conclusões e Recomendações	44
3.1	Recapitulação	44
3.2	Etapas desenvolvidas ao longo do trabalho	44
3.3	Pontos positivos e negativos	45
3.4	Recomendações de trabalhos futuros	45
	Referências Bibliográficas	46

Índice de Ilustrações

Figura 1 - Hierarquia dos subsistemas	7
Figura 2 – Hierarquia e funcionalidades do subsistema 01	7
Figura 3 - Hierarquia e funcionalidades do subsistema 02	8
Figura 4 - Hierarquia e funcionalidades do subsistema 03	8
Figura 5 - Perspectiva do módulo 3	9
Figura 6 - Diagrama de Casos de Uso do módulo 3.....	21
Figura 7 - Página protótipo inicial do módulo 3	22
Figura 8 - Página protótipo "Finanças"	22
Figura 9 - Categorias de supervisão mostradas ao clicar o item "Meus idosos"	23
Figura 10 - Página protótipo "Cuidados Diários"	23
Figura 11 - Tela protótipo "Sinais Vitais"	24
Figura 12 - Tela protótipo "Higiene"	24
Figura 13 - Tela protótipo "Evolução Diária"	25
Figura 14 - Tela protótipo "Eliminação"	25
Figura 15 - Tela protótipo "Padrão Alimentar"	26
Figura 16 - Tela protótipo "Patologias e Prescrições"	27
Figura 17 - Tela protótipo "Atividades recreativas"	27
Figura 18 - <i>Dashboard</i> com todas as categorias de todos os módulos.	28
Figura 19 - Página "Editar Informações" dentro de uma <i>section</i> identificada como "md3"..	29
Figura 20 - Rotas do módulo 3, situadas no arquivo " <i>Route</i> ".	29
Figura 21 - Parte do arquivo Controller do módulo 3, onde cada função é uma página que o sistema consegue renderizar a partir das rotas criadas.	30
Figura 22 - Todas as pastas pertencentes à pasta <i>App</i> , além do arquivo " <i>Route</i> " e da classe abstrata DAO.....	31
Figura 23 - Classe 'EditarInfo'.....	31
Figura 24 - Método "alterar"	32
Figura 25 - Método "buscarPorId"	32
Figura 26 - Método 'listar'	33
Figura 27 - Código da página "Editar Informações"	33
Figura 28 - Método controlador de rotas da página "Editar Informações"	34
Figura 29 - Classe "FinanceirosResp"	34
Figura 30 - método "listar" da página "Finanças"	35
Figura 31 - Código da página "Finanças"	35

Figura 32 - método controlador de rotas da página "Finanças"	36
Figura 33 - Classe 1 da página "Cuidados Diários"	36
Figura 34 - Classe 2 da página "Cuidados Diários"	37
Figura 35 - Classe 3 da página "Cuidados Diários"	37
Figura 36 - Classe 4 da página "Cuidados Diários"	38
Figura 37 - Método "listar" da tabela "Hidratação"	39
Figura 38 - Método "listar" da tabela "Banho de Sol"	39
Figura 39 - Método "listar" da tabela "Repelente"	40
Figura 40 - Método "listar" da tabela "Padrão de Sono"	40
Figura 41 - Código da tabela "Hidratação"	41
Figura 42 - Código da tabela "Banho de sol"	41
Figura 43 - Código da tabela "Repelente"	42
Figura 44 - Código da tabela "Padrão de Sono"	42
Figura 45 - Método controlador da página "Cuidados Diários"	43

1 Introdução

1.1 Contextualização e Motivação

O Município de São João da Boa Vista localiza-se no interior do estado de São Paulo e estima-se que sua população atualmente é de 90.637 (2018) habitantes e seu Índice de Desenvolvimento Humano é de 0,797 (2010). Além disso, sua área territorial é de 516,399 km² (2018) e com uma densidade demográfica de 161,96 hab/km² (2010) [1].

São João da Boa Vista também é a melhor cidade do Brasil para se viver após os 60 anos, entre os municípios com até 100 mil habitantes segundo o índice de Desenvolvimento Urbano para a Longevidade (IDL). Segundo o índice de Desenvolvimento Humano Municipal (IDHM), a expectativa de vida dos cidadãos sanjoanenses aumentou de 73 para 77 anos, consequentemente o município começou a investir mais no bem-estar dos idosos. Um exemplo de tais investimentos são os centros de convivência do idoso (CCIs), atualmente o município possui três deles, em bairros específicos da cidade [2].

É também nesse município que se localiza o campus São João da Boa Vista do Instituto Federal de Ciência e Tecnologia de São Paulo, que está em funcionamento desde o ano de 2007, primeiramente como Unidade de Ensino Descentralizada (UNED) do até então Centro Federal de Educação Tecnológica de São Paulo (CEFET). Atualmente o campus conta com cursos técnicos integrados ao ensino médio, técnicos concomitantes ou subsequente, bacharelados, engenharias, licenciaturas, pós-graduações e tecnologias [3].

Dentre os cursos técnicos integrados ao ensino médio há o técnico em informática, que visa ensinar competências como análise de dados; colaboração no levantamento de informações junto aos usuários para a elaboração de anteprojetos de sistemas; desenvolvimento e manutenção de programas; implantação e manutenção de sistemas; desenvolvimento de testes e simulações, gerando os arquivos necessários; análise dos resultados dos programas, identificando desvios e realizando correções [4].

No quarto ano desse curso, é proposto aos alunos que analisem, desenvolvam e implantem um sistema durante as aulas de Prática de Desenvolvimento de Sistemas, ministrada pelo professor Breno Lisi Romano. O mesmo decide qual será o propósito do sistema, bem como seu público alvo e realiza a análise dos macro-requisitos. No ano de 2019 foi proposto um portal para cuidado especializado de idosos, tendo como público-alvo instituições de longa permanência e como motivação os dados estatísticos sobre a terceira idade citados acima. Para o desenvolvimento, o

sistema foi dividido em 3 subsistemas: o acesso externo, o gerenciamento de idosos e o controle gerencial, todos com 3 módulos (funcionalidades) cada, totalizando 9 módulos.

Figura 1 - Hierarquia dos subsistemas

O subsistema 01 deve gerenciar, manter e garantir o acesso dos usuários em geral, dos enfermeiros e dos responsáveis pelos idosos:

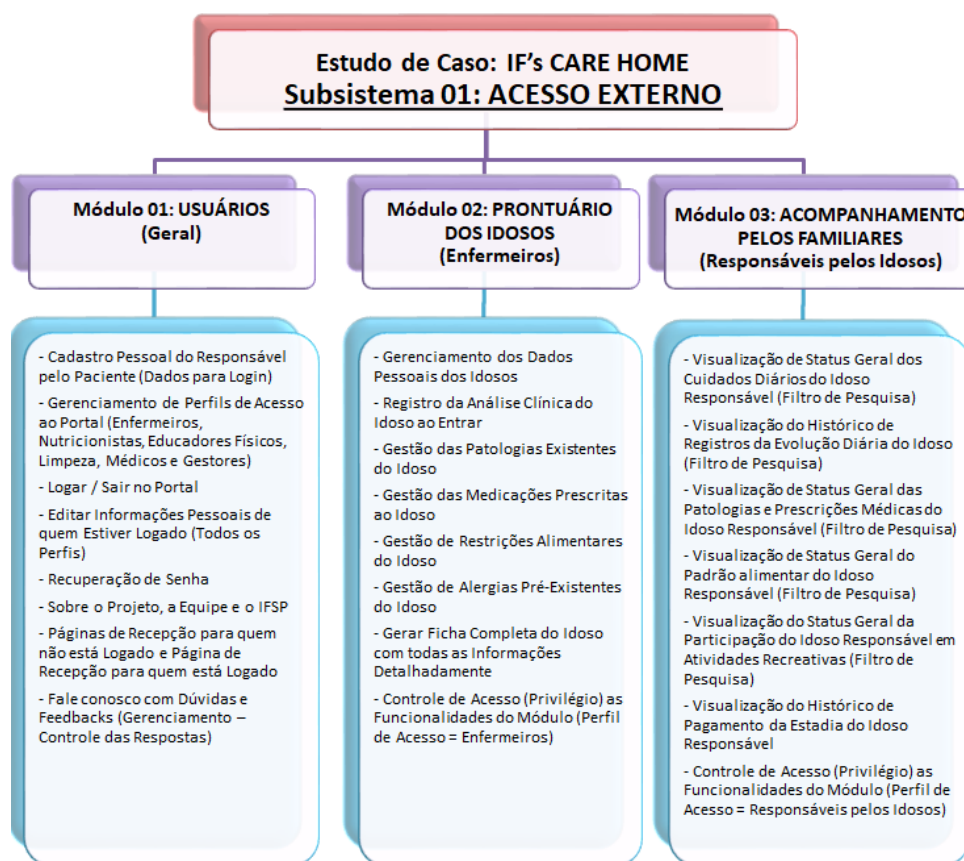


Figura 2 – Hierarquia e funcionalidades do subsistema 01

O subsistema 02 deve gerenciar, controlar e cadastrar as atividades relacionadas aos idosos que frequentam casas de longa permanência, bem como o estoque de alimentos e produtos farmacêuticos:

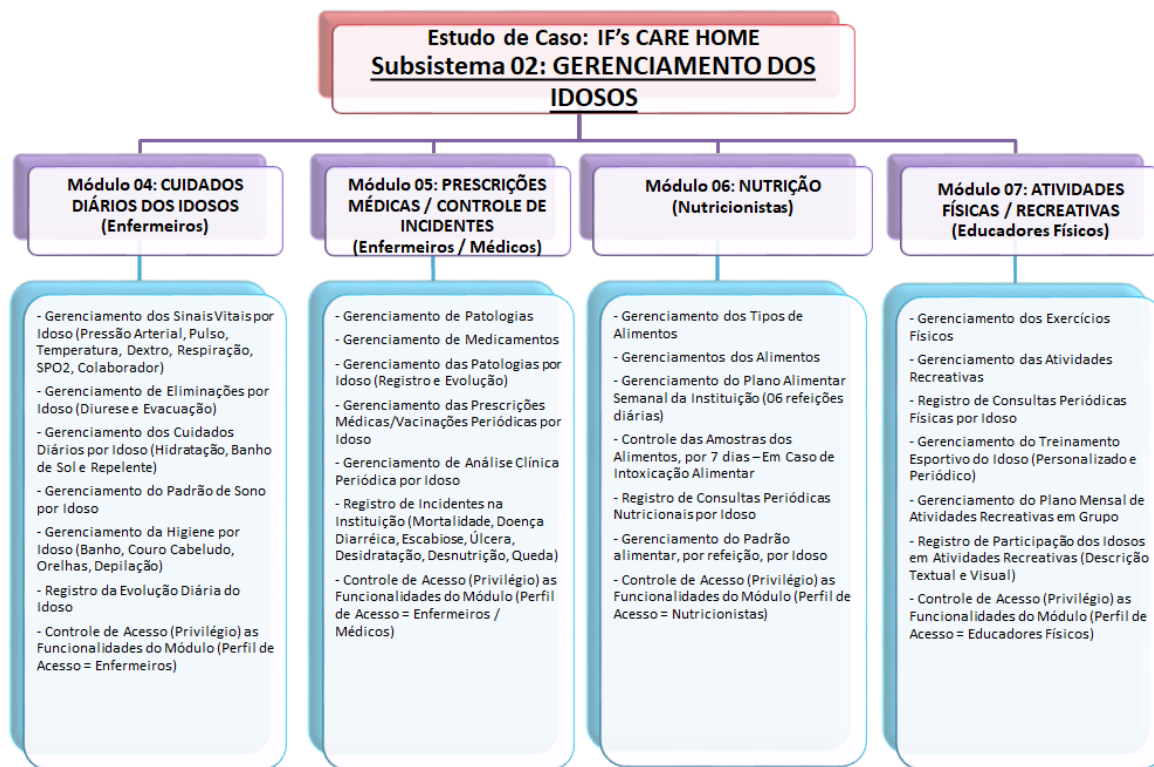


Figura 3 - Hierarquia e funcionalidades do subsistema 02

O subsistema 03 deve registrar, controlar e gerenciar as atividades do setor administrativo de instituições de longa permanência, bem como a geração de relatórios especializados para o mesmo setor:

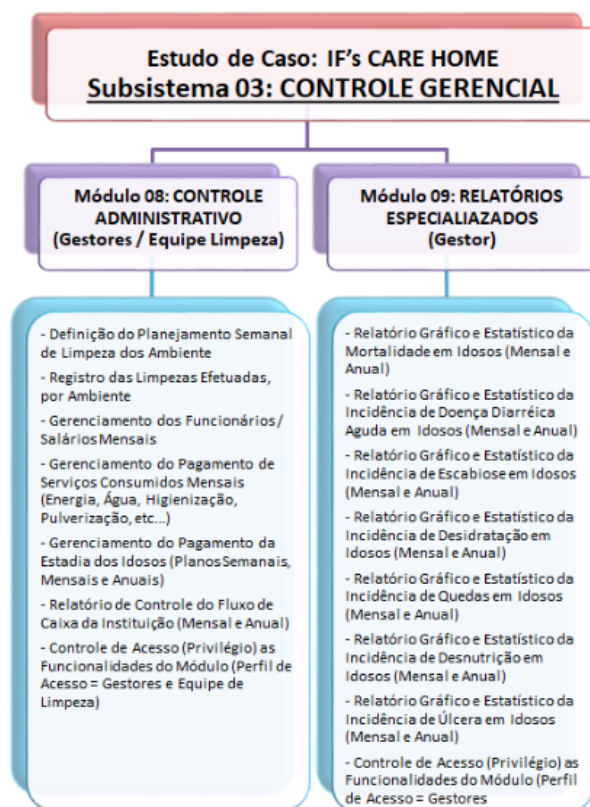


Figura 4 - Hierarquia e funcionalidades do subsistema 03

O módulo de Acompanhamento dos Familiares, que será abordado ao longo deste documento, tem por finalidade englobar diferentes funcionalidades do sistema, como visualizar registros da evolução diária, das patologias, prescrições médicas e padrão alimentar do idoso, além do histórico de participação do idoso nas atividades e do histórico de pagamento de estadia. Todas essas funcionalidades deverão ser visualizadas pelos familiares, para que eles realizem um acompanhamento detalhado dos seus idosos que estão ingressados nas instituições de longa permanência.

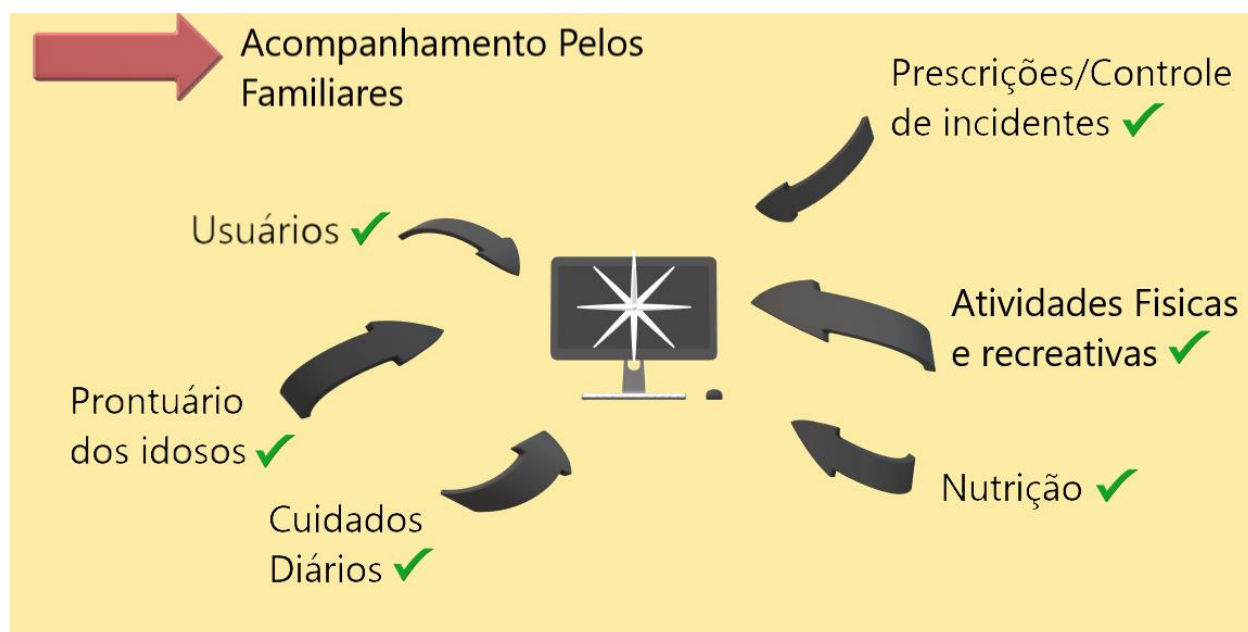


Figura 5 - Perspectiva do módulo 3

1.2 Objetivo Geral

O objetivo geral deste documento compreende a que se diz respeito ao desenvolvimento das funcionalidades do módulo 3, seus protótipos, suas iterações e correções, apresentando suas etapas e o modo como foram divididas, bem como as dificuldades que foram surgindo ao longo delas.

1.3 Objetivos Específicos

Os objetivos específicos presentes neste documento são:

- O processo de levantamento de requisitos;
- A divisão das funcionalidades em casos de uso e o desenvolvimento das páginas protótipo a partir dos casos de uso;
- A migração das páginas protótipo para o sistema parcialmente integrado (refatoração);
- O desenvolvimento das funcionalidades das iterações a partir das páginas-protótipo;

2 Desenvolvimento

2.1 Levantamento Bibliográfico

2.1.1 Introdução às linguagens de programação

Computadores são utilizados para realizarem todos os tipos de tarefas, desde administrar usinas nucleares até jogos eletrônicos em *smartphones*. Para que se realizem tais tarefas, foram criadas linguagens de programação específicas para cada tarefa. Atualmente, existem quatro tipos específicos de aplicações de computadores e as linguagens associadas a elas:

1. Aplicações Científicas: os primeiros computadores criados na década de 1940 eram para uso restrito de aplicações científicas e exigiam cálculos aritméticos de ponto flutuante e por isso utilizavam uma estrutura de dados relativamente simples. Para suprir a necessidade de uma estrutura mais complexa, as primeiras linguagens de programação foram criadas. *Assembly* foi a primeira delas, e logo depois surgiu *Fortran*, a primeira linguagem para aplicações científicas. ALGOL 60 e outras demais linguagens posteriores também tinham a mesma função. Porém como eficiência é a principal preocupação quando o assunto é aplicações científicas, nenhuma dessas linguagens posteriores é significativamente melhor que a *Fortran* e isso explica o seu uso até os dias de hoje;
2. Aplicações Empresariais: O uso de computadores para fins empresariais começou na década de 1950, porém foi apenas na década de 1960 que foi desenvolvida uma linguagem de alto nível para aplicações empresarias, chamada COBOL. Esta provavelmente é a linguagem mais usada para tais aplicações devido sua facilidade em gerar relatórios elaborados, suas maneiras precisas em descrever e armazenar números decimais e sua habilidade em realizar cálculos aritméticos com números decimais;
3. Inteligência Artificial: Esta é uma ampla área de aplicações caracterizadas pelo uso de computações simbólicas em vez de numéricas. Consiste em manipular nomes (símbolos) em vez de números e é feita de modo mais fácil por meio de listas ligadas de dados em vez de vetores. A primeira linguagem desenvolvida para ser utilizada para o desenvolvimento de aplicações de IA foi a linguagem funcional *Lisp*, em 1959. No início dos anos 1970 surgiu uma alternativa a essa linguagem funcional, a programação lógica utilizando a linguagem *Prolog* e mais recentemente, algumas aplicações de IA são escritas em sistemas de linguagens como *C. Scheme*.

4. Software para a Web: esta área possui uma ampla coleção de linguagens destinadas ao desenvolvimento de páginas da *World Wide Web*. Tal coleção vai desde linguagens de marcação (não de programação) como HTML até linguagens de programação de propósito geral, como *Java*. Como é necessário que haja um conteúdo dinâmico na Web, alguma capacidade de computação é incluída na tecnologia de apresentação deste conteúdo. Esta funcionalidade é escrita por uma linguagem de programação embarcado em um documento HTML. O mesmo normalmente é escrito com uma linguagem scripting, com JavaScript ou PHP. [5]

2.1.2 Programação orientada a objetos

A POO (Programação Orientada a Objetos) estrutura um programa dividindo-o em vários objetos de alto nível. Cada objeto representa e modela algum aspecto dos requisitos do programa que você está desenvolvendo. Tais objetos se interagem entre si, para orientar o fluxo global do programa. Com essa técnica inovadora de estruturar um programa, ele se torna uma simulação viva do requisito que você está sintetizando.

Resumindo, **objeto** em POO é uma construção de programa que encapsula estado ou comportamento. Uma vez criados, os objetos permitem que você modele seu programa em termos reais e abstrações.

Classe é outro conceito importante em POO, pois é ela que define todas as características comuns a um objeto, define todos os **atributos** e **comportamentos** expostos pelo objeto, além de definir quais mensagens seus objetos respondem.

Atributos são as características de uma classe visíveis externamente. Cor dos olhos, altura e peso são exemplos de atributos.

Comportamento é qualquer ação executada por um objeto quando passada uma mensagem ou em resposta a uma mudança de estado.

Exemplificando tudo em um código, temos um **objeto** carro. Um carro tem modelo e marca (**atributos**). Um carro saberá calcular seu preço descontado. Todos os objetos carro são instâncias da **classe** carro. Uma classe carro poderia ser como segue:

```

Public class Carro
{
    Public String marca;
    Public String modelo;

    Public double calculaPreco(int qtde)
    {
        double desconto = 0;
        double precoUnitario = 25000;
        double precoFinal;

        if (qtde >= 1 && qtde <=3)
        {
            desconto = 2000;
        }
        else if (qtde > 3)
        {
            desconto = 4000;
        }

        precoFinal = ((qtde * precoUnitario) - desconto);
        return precoFinal;
    }
}

```

Public double calculaPreco(int qtde) é um **método**, ou seja, um comportamento exercido pelo objeto. Existem três tipos de métodos que são fundamentais para a construção de uma classe: o método construtor, o *get* e o *set*.

O método **construtor** é usado para inicializar objetos durante sua instanciação. A criação de objetos é chamada instanciação porque ela cria uma instância do objeto da classe.

Get e **set** são métodos configuradores de atributos. Métodos **get** são chamados de assessores, pois eles permitem que você acesse dados internos do objeto, enquanto os métodos **set** são chamados de mutantes, pois eles permitem que você altere o estado interno do objeto.

Além destes conceitos básicos como objeto, classe, atributo e método, há mais quatro conceitos fundamentais da POO, e por isso são chamados de pilares da programação orientada a objetos:

1. **Encapsulamento:** Em vez de ver um programa como único e monolítico, o encapsulamento permite que ele seja dividido em várias partes menores e independentes. Cada parte possui sua implementação e realiza seu trabalho independentemente. O encapsulamento mantém essa independência, ocultando detalhes da implementação de cada parte. Resumindo, o encapsulamento permite que você construa partes ocultas da implementação do programa, criando uma espécie de caixa preta. Existem três tipos de encapsulamento: **Público** (garante acesso a todos os objetos), **protegido** (garante acesso à instância, ou seja, para aquele objeto, e para todas as subclasses) e **privado** (garante o acesso apenas para a instância).
2. **Abstração:** é o processo de simplificar um problema difícil. Transformar o requisito real em algo abstrato, não se preocupando com cada detalhe real, mas tratando apenas dos detalhes pertinentes ao programa. A abstração tem duas vantagens. A primeira é que ela permite que você sintetize um requisito facilmente, a outra é ela ajuda a obter reutilização. Muitas vezes os programas são complexos e especializados. Esta especialização combinada com dependência desnecessária entre os componentes torna difícil a reutilização de códigos existentes. Sempre que possível, é de suma importância criar objetos que resolvam um domínio inteiro de requisitos. Resumindo, a abstração permite que você abstrata um requisito uma vez e depois use essa abstração por todo o domínio desse requisito.
3. **Herança:** A reutilização de código citada em abstração é feita muitas vezes pela característica de herança. Tal característica otimiza a produção do programa em tempo e linhas de código. Basicamente, um objeto herda as características (atributos e métodos) de todos os objetos acima dele, seus “ancestrais”. A herança a partir das características do objeto mais acima é considerada herança direta, enquanto as demais são consideradas heranças indiretas. Exemplificando, na família a criança herda diretamente do pai e indiretamente do avô e do bisavô, e o mesmo acontece com os objetos em uma POO.
4. **Polimorfismo:** Como já foi dito, os objetos filhos herdam as características e ações de seus “ancestrais”. Porém em alguns casos é necessário que as ações para um mesmo método seja diferente. E essa é a função da característica polimorfismo, que consiste na alteração do funcionamento interno de um método herdado de um objeto

pai. Exemplificando, temos um objeto pai “Eletrodoméstico”. Esse objeto possui o método “Ligar()”. E temos dois objetos filhos, “Televisão” e “Geladeira”, que não irão ser ligados da mesma forma. Assim, precisamos para reescrever cada uma das classes filhas o método “Ligar()”.

Esses quatro pilares são essenciais no entendimento de qualquer linguagem orientada a objetos. Cada linguagem irá implementar esses pilares de uma forma, mas essencialmente suas implementações são iguais. No próximo capítulo veremos como funciona a POO na linguagem *php*, bem como os quatro pilares e os conceitos básicos dessa forma de estruturar programas. [6]

2.1.3 Linguagem PHP

A linguagem PHP (*Personal Home Page*) foi criada em 1995 por Rasmus Lerdorf. Originalmente como um *script Perl/CGI* para visualizar quantos visitantes estavam lendo seu currículo online. A partir daí a linguagem sofreu muitas mudanças e hoje está em sua sexta versão e é uma das linguagens mais usadas para o desenvolvimento web.

Todo script de *php* é escrito dentro das chaves `<?php?>`, começando com `<?php` e terminando com `?>`, e pode ser colocado dentro de linhas de marcação HTML. Veja um exemplo:

```
<h3>Bem-vindo!</h3>
<?php
    Echo "<p>Algum conteúdo dinâmico aqui</p>";
?>
<p>Algum conteúdo estático aqui</p>
```

Chaves como `<h3>` e `<p>` são próprias da linguagem de marcação HTML, e tudo dentro das chaves `<?php?>` será rodado como um script de PHP.

O PHP suporta quatro tipos de dados, que são os mais comuns: *Boolean* (verdadeiro/falso), *integer* (números inteiros) *float* (números decimais), *String* (sequência de caracteres) e *array* (vetor).

É a partir destes tipos de dados que se cria variáveis em PHP. Variáveis são identificadas com o cifrão (\$) no começo. Nomes de variáveis podem começar com uma letra ou um *underscore*, e pode consistir em letras, números ou outros caracteres ASCII, variando entre 127 a 255.

As variáveis também são divididas em escopos. A localização da declaração influencia o domínio em que a variável pode ser acessada, e existem 4 tipos de escopo: as variáveis locais, parâmetros de função, variáveis globais e variáveis estáticas.

Variáveis locais são aquelas declaradas dentro de uma função e só podem ser referenciadas dentro daquela função em que foi declarada. **Parâmetros de função** são nada mais que os argumentos do cabeçalho de uma função, como em muitas linguagens de programação, qualquer função que aceita argumentos deve declarar estes em seu cabeçalho. **Variáveis globais**, em contraste com as variáveis locais, podem ser acessadas em qualquer parte do programa. E as **Variáveis estáticas** são aquelas que não perdem seu valor quando uma função é fechada, diferente dos parâmetros de função.

Além destes quatro escopos, o PHP possui um escopo maior que todos eles, que são as **variáveis superglobais**, que nada mais são que um conjunto de variáveis pré-definidas que são acessíveis a partir de qualquer lugar dentro do script executado e oferece uma quantidade substancial de informação sobre o ambiente específico. Neste documento, focaremos em apenas duas delas, **\$_GET** e **\$_POST**.

Primeiro, é necessário ver como o PHP trabalha na criação de formulários Web e como essa linguagem é tão eficaz quando o assunto é recolher e processar dados valiosos do usuário.

Como as variáveis superglobais, há dois métodos para passar dados de um *script* para outro: o **GET** e o **POST**. Apesar do GET ser o padrão, é recomendado usar o POST por ele ser capaz de gerenciar mais dados, o que é um ponto considerável quando se trata de formulários que inserem e modificam grandes blocos de texto. Se você optar por usar o método POST, qualquer dado enviado para outro script PHP deve ser referenciado com a superglobal **\$_POST**, e o mesmo vale para o método GET, que deve ser referenciado pela superglobal **\$_GET**. Abaixo temos um exemplo de formulário onde seus dados são passados pelo método POST:

```
<?php
    //Se o campo name (nome) está preenchido
    If (isset($_POST['name']))
    {
        $name = htmlentities($_POST['name']);
        $email = htmlentities($_POST['email']);
        printf ("Oi %s! <br/>", $name);
        printf("O endereço %s em breve será um imã de spam! <br/>", $email);
    }
?>
```

```

<form action="subscribe.php" method="post">
    <p> Name:<br/> <input type="text" id="name" name="name" /> </p>
    <p> Email Address:<br/> <input type="text" id="email" name="email" /> </p>
    <input type="submit" id="submit" name="submit" value="Ir!" /> </p>
</form>

```

Superficialmente, é desta forma que a linguagem PHP trabalha, desenvolvendo *scripts* para páginas Web a partir de linhas de marcação de HTML.

2.1.4 MySQL

MySQL é um servidor de banco de dados relacional baseado na linguagem de programação SQL (*Structured Query Language*), criado em 1996 por iniciativa de funcionários da empresa sueca TcX DataKonsult. O software provou ser tão popular que, em 2001, eles fundaram uma empresa totalmente específica na oferta de produtos relacionados ao MySQL, chamando-a de MySQLAB. Atualmente está em sua oitava versão (8.0.16) e é desenvolvido pela *Oracle Corporation*.

Uma tabela de banco de dados relacional é uma estrutura usada para armazenar e organizar todo o tipo de informações. Tais tabelas são usadas de diversas maneiras, desde a mais simples, como para armazenar informações pessoais de usuários, quanto para armazenar informações transacionais.

O MySQL suporta vários tipos de tabelas, sejam elas criadas para armazenar dados de longo prazo ou não, e cada uma delas com seus propósitos específicos. Dessa forma o MySQL oferece vários mecanismos de armazenamento diferentes, da maneira mais adequada para os requisitos. Este documento aborda especificamente o InnoDB, que foi usado para o desenvolvimento da pesquisa.

InnoDB é um mecanismo robusto de armazenamento transacional e um banco de dados backend por si mesmo. Suas tabelas são ideais para os seguintes cenários, entre outros: atualizações intensivas, transações e recuperação automática de crash. Desde a sua versão 3.23.44, o mecanismo suporta chaves estrangeiras, um recurso considerado não disponível em outros mecanismos de armazenamento.

O MySQL suporta uma infinidade de dados e todos eles estão divididos em três tipos: data e hora, numérico e *string*. Aqui listaremos os mais usados:

- DATETIME: dado do tipo data e hora, formato AAAA-MM-DD HH:MM:SS e com um intervalo de 1000-01-01 00:00:00 até 9999-12-31 23:59:59;

- BIGINT: dado do tipo numérico, oferece o maior intervalo para números inteiros do MySQL;
- TINYINT: dado do tipo numérico, corresponde ao menor intervalo de números inteiros do MySQL;
- FLOAT: dado do tipo numérico, corresponde a números decimais flutuantes com precisão simples do MySQL;
- CHAR: dado do tipo string, oferece a representação de caracteres de tamanho fixo do MySQL, suportando um tamanho máximo de 255 caracteres.

Além dos dados e seus tipos, o MySQL também possui atributos para serem usados com eles, são estes os frequentemente usados:

- AUTO_INCREMENT: como o próprio nome já diz, é um atributo que automaticamente incrementa dados inteiros a um identificador inteiro único a linhas recém inseridas;
- NULL: de modo simples, significa que nenhum valor pode existir para um dado campo;
- NOT NULL: definir uma coluna como NOT NULL irá proibir qualquer tentativa de inserir um valor nulo nessa coluna.
- PRIMARY KEY: é o atributo mais importante numa tabela, pois a coluna que for atribuída com esse atributo não poderá ter seus valores como nulos ou repetidos. Este atributo é comumente utilizado para valores identificadores, como número de RG ou CPF, prontuário, número de série ou qualquer outra identificação.

Apresentados os dados e seu tipos, bem como seus atributos, apresentaremos os comandos básicos usados em MySQL como criar e deletar bancos de dados MySQL, criar e deletar tabelas, inserir, alterar e selecionar dados de uma tabela:

- CREATE DATABASE databasename: cria um banco de dados com o nome databasename;
- DROP DATABASE databasename: deleta o banco de dados com o nome databasename;
- CREATE TABLE tablename: cria uma tabela com o nome tablename;
- DROP TABLE tablename: deleta a tabela com o nome tablename;
- ALTER TABLE tablename ADD COLUMN name CHAR: altera a estrutura da tabela tablename, adicionando uma coluna chamada name com dados CHAR;
- INSERT INTO tablename: insere dados dentro da tabela tablename;

- UPDATE tablename SET: altera dados dentro da tabela tablename;
- SELECT name FROM tablename: seleciona a coluna name da tabela tablename;
- DELETE FROM tablename WHERE name="John": apaga os dados da linha da tabela tablename onde na coluna name possui o dado "John". [7]

2.2 Desenvolvimento da Pesquisa

2.2.1 Requisitos do Módulo 03

Os macrореquisitos do módulo 3 mostrados no diagrama do sistema foram filtrados e detalhados em requisitos funcionais e não-funcionais, sendo funcionais os requisitos que garantem o bom funcionamento do sistema, enquanto os não-funcionais são aqueles que garantem a qualidade do sistema.

Com base nesses conhecimentos, foram pensados treze requisitos funcionais e oito requisitos não funcionais para serem desenvolvidos no módulo 3.

2.2.1.1 Requisitos Funcionais (resumidos em telas)

- Ao realizar o login no sistema, o usuário será redirecionado para a home page do sistema, que possui um menu lateral com as opções "Finanças", "Sinais Vitais", "Cuidados Diários", "Higiene", "Eliminação", "Evolução Diária", "Padrão Alimentar", "Patologias e Prescrições", "Atividades Recreativas" e "Editar Informações";
- Ao clicar na opção "Finanças", o usuário será redirecionando a uma tela contendo informações referentes ao pagamento da estadia do idoso com quem está vinculado, com uma tabela com as seguintes colunas: "Data de vencimento", "Forma de pagamento", "Valor pagamento" e "Tipo de plano". Todos os valores mostrados nessa tabela devem estar relacionados ao usuário que está logado;
- Ao clicar na opção "Sinais Vitais", o usuário será redirecionando a uma tela contendo informações referentes aos sinais vitais do idoso com quem está vinculado, com seis tabelas, uma para cada item: "Pressão Arterial" (colunas: "Data/Hora", "Enfermeiro Responsável" e "Pressão Arterial"), "Temperatura" (colunas: "Data/Hora", "Enfermeiro Responsável" e "Temperatura"), "Dextro" (colunas: "Data/Hora", "Enfermeiro Responsável" e "Quantidade de glicose no sangue (mg/dl)"), "Pulso" (colunas: "Data/Hora", "Enfermeiro Responsável" e "Batimentos por minuto"), "Respiração" (colunas: "Data/Hora", "Enfermeiro Responsável" e

“Respiração”) e “Oximetria de Pulso” (colunas: “Data/Hora”, “Enfermeiro Responsável” e “Nível de oxigênio sanguíneo”);

- Ao clicar na opção “Cuidados Diários”, o usuário será redirecionando a uma tela contendo informações de checagem dos cuidados diários do idoso com quem está vinculado, com quatro tabelas, uma para cada item: “Hidratação”, “Banho de Sol”, “Repelente” e “Padrão de Sono”, todas elas possuem as colunas “Data”, “Hora” e “Enfermeiro (a) Supervisor (a) ”;
- Ao clicar na opção “Higiene”, o usuário será redirecionando a uma tela contendo informações de checagem de higiene em geral do idoso com quem está vinculado, com quatro tabelas, uma para cada item: “Banho”, “Couro Cabeludo”, “Orelha” e “Depilação”, todas elas possuem as colunas “Data”, “Hora” e “Enfermeiro (a) Supervisor (a) ”;
- Ao clicar na opção “Eliminação”, o usuário será redirecionando a uma tela contendo informações de checagem das eliminações do idoso com quem está vinculado, com duas tabelas, uma para cada item: “Evacuação” e “Diurese”, ambas possuem as colunas “Data”, “Hora” e “Enfermeiro (a) Supervisor (a) ”;
- Ao clicar na opção “Evolução Diária”, o usuário será redirecionando a uma tela contendo informações de melhorias físicas, psicológicas e comportamentais do idoso com quem está vinculado, com uma tabela com as colunas “Melhoria do idoso”, “Observação” e “Enfermeiro (a) Supervisor (a) ”;
- Ao clicar na opção “Padrão Alimentar”, o usuário será redirecionando a uma tela contendo informações referentes a dieta do idoso com quem está vinculado, com uma tabela com as colunas “Plano de Refeição”, “Status da Alimentação” e “Enfermeiro (a) Supervisor (a) ”;
- Ao clicar na opção “Patologias e Prescrições”, o usuário será redirecionando a uma tela contendo informações referentes as doenças, medicamentos e tratamentos do idoso com quem está vinculado, com três tabelas: “Patologias” (colunas: “Nome da Patologia”, “Data de incidência”, “Grau” e “Enfermeiro (a) Supervisor (a) ”), “Prescrições” (colunas: “Nome do medicamento”, “Dosagem”, “Validade da prescrição” e “Enfermeiro (a) Supervisor (a) ”), e “Vacinações” (colunas: “Dosagem (ml)”, “Data da Vacina” e “Enfermeiro (a) Supervisor (a) ”).
- Ao clicar na opção “Atividades Recreativas”, o usuário será redirecionando a uma tela contendo informações referentes as atividades físicas e recreativas que o idoso

com quem está vinculado realizou, com uma tabela com as colunas “Data”, “Atividade realizada” e “Enfermeiro (a) Supervisor (a) ”;

- Ao clicar na opção “Editar Informações”, o usuário será redirecionado a uma tela contendo um formulário mostrando suas seguintes informações pessoais: nome, e-mail, telefone, número de RG, número de CPF, data de nascimento e endereço. Todas elas serão informações alteráveis, uma vez que o usuário edite o campo desejado e clique no botão “alterar” esse campo terá seu valor alterado.

2.2.1.2 Requisitos Não-funcionais

- Processo de Software: É necessário que o Processo de Software a ser utilizado seja o Redmine + Kanban;
- É necessário que o sistema seja desenvolvido utilizando a linguagem PHP;
- É necessário a utilização do *design pattern* MVC (*Model View Controller*), não será utilizada nenhuma ferramenta de geração de código;
- É necessário disponibilizar, através da Web, a documentação do desenvolvimento do Protótipo do Projeto de SGI;
- Este sistema deverá ser capaz de propiciar uma interface agradável ao usuário, com opções simples, e autoexplicativas;
- O sistema deve garantir que o único que terá permissão para acessar essa página é o responsável pelo idoso;
- O menu de opções (“Home”, “Finanças”, “Meu idoso”, “Sair”) deve ser horizontal e estar posicionado na parte superior da tela;
- O sistema deverá ser executado nos navegadores: Google Chrome, Mozilla Firefox, e Internet Explorer.

2.2.2 Diagrama de Casos de Uso e suas iterações

Para conceituar tais requisitos, foi utilizado A UML (Linguagem de Modelagem Unificada) de diagrama de casos de uso. Os requisitos funcionais foram relacionados a partir de suas prioridades e o diagrama desenvolvido foi este:

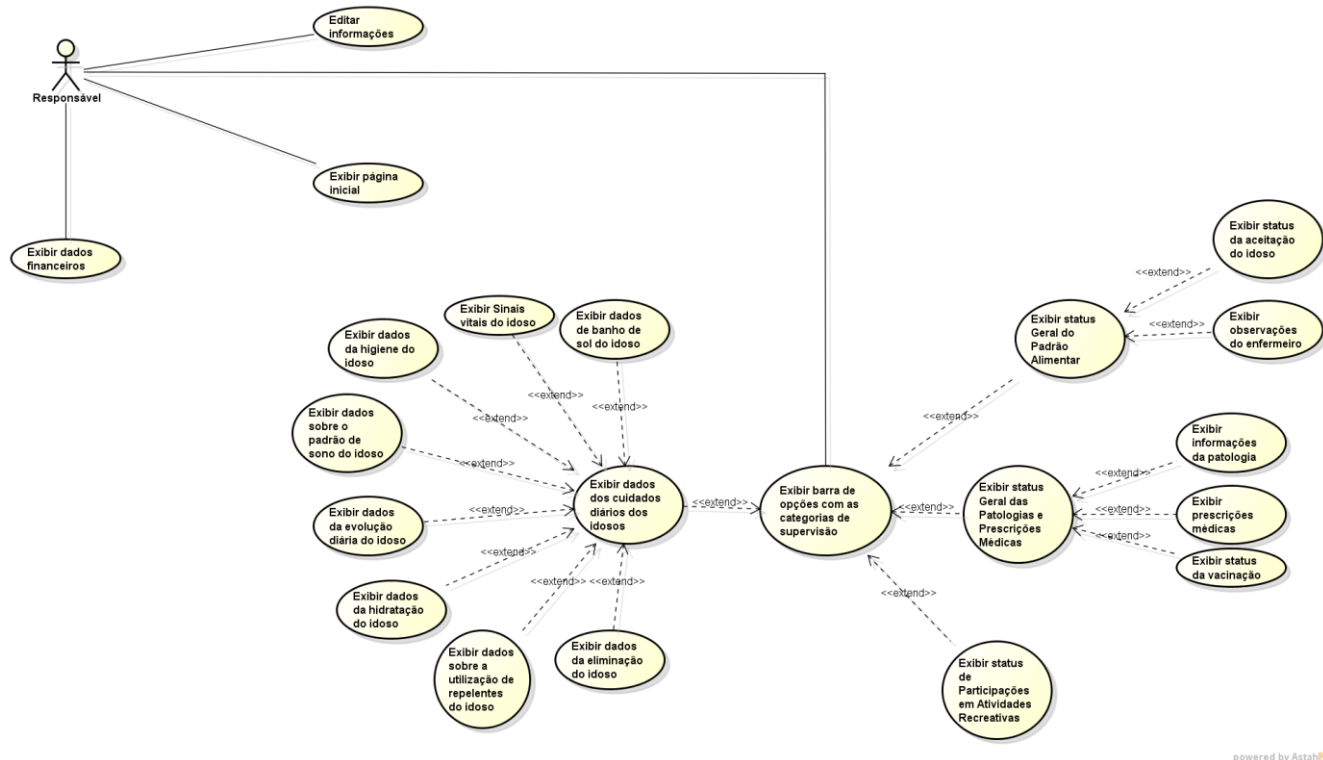


Figura 6 - Diagrama de Casos de Uso do módulo 3

Após os requisitos levantados e o diagrama criado, o próximo passo foi criar as telas protótipo com base no diagrama, a maioria dos casos de uso foram transformados em tabelas, e essas agrupadas em 10 telas.

A primeira delas é a **página inicial**, que conterá as informações pessoais do usuário, ao lado está o menu lateral com links para as categorias de supervisão do idoso e para os dados financeiros:

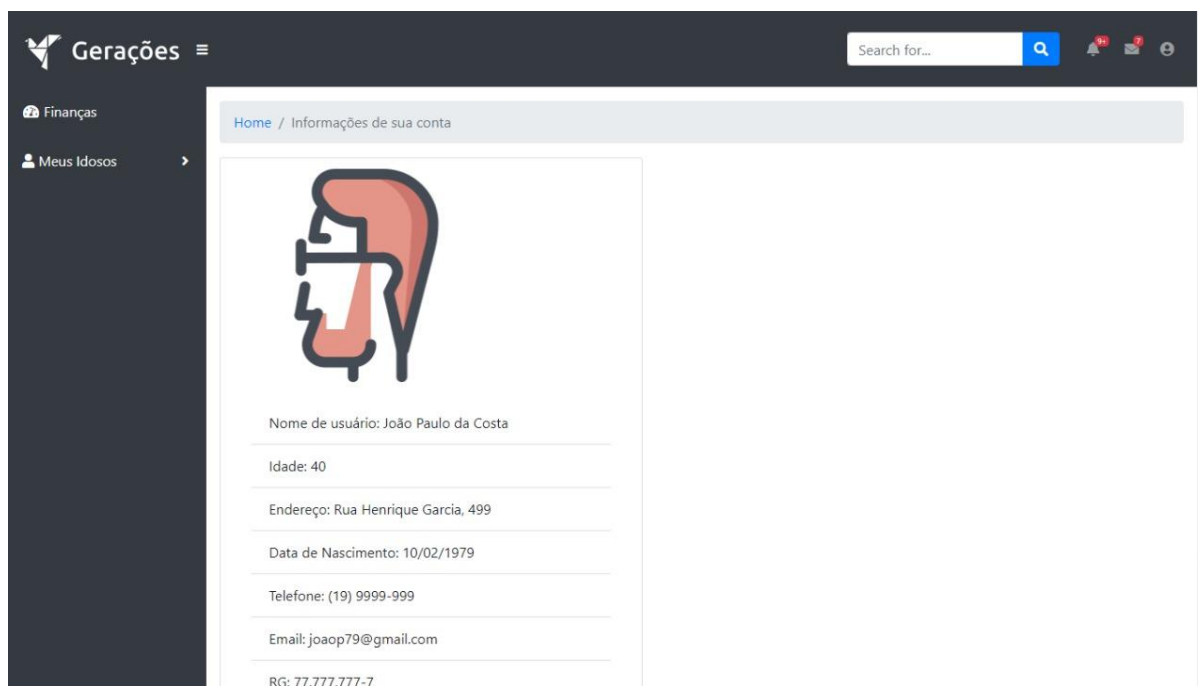


Figura 7 - Página protótipo inicial do módulo 3

Ao clicar na opção “**Finanças**” o usuário seria redirecionado a uma tela contendo uma tabela com dados referentes ao pagamento da estadia:

Relatórios Finanças

Tabela de Dados

Show 10 entries

Search:

Data de pagamento	Período referente ao pagamento	Valor pagamento	Forma de pagamento	Próximo pagamento
Dia 01	01/04/2019-01/05/2019	800,00	Dinheiro	10/05/2019
Dia 03	03/04/2019-03/05/2019	800,00	Dinheiro	10/05/2019
Dia 04	04/04/2019-04/05/2019	800,00	Dinheiro	10/05/2019
Dia 05	05/04/2019-05/05/2019	800,00	Dinheiro	10/05/2019
Dia 10	10/04/2019-10/05/2019	800,00	Dinheiro	10/05/2019
Dia 10	10/04/2019-10/05/2019	800,00	Dinheiro	10/05/2019
Dia 15	15/04/2019-15/05/2019	800,00	Dinheiro	10/05/2019
Dia 20	20/04/2019-20/05/2019	800,00	Dinheiro	10/05/2019

Figura 8 - Página protótipo "Finanças"

Ao clicar na opção “**Meus idosos**” é mostrado um menu com todas as categorias de supervisão disponibilizadas para os responsáveis:

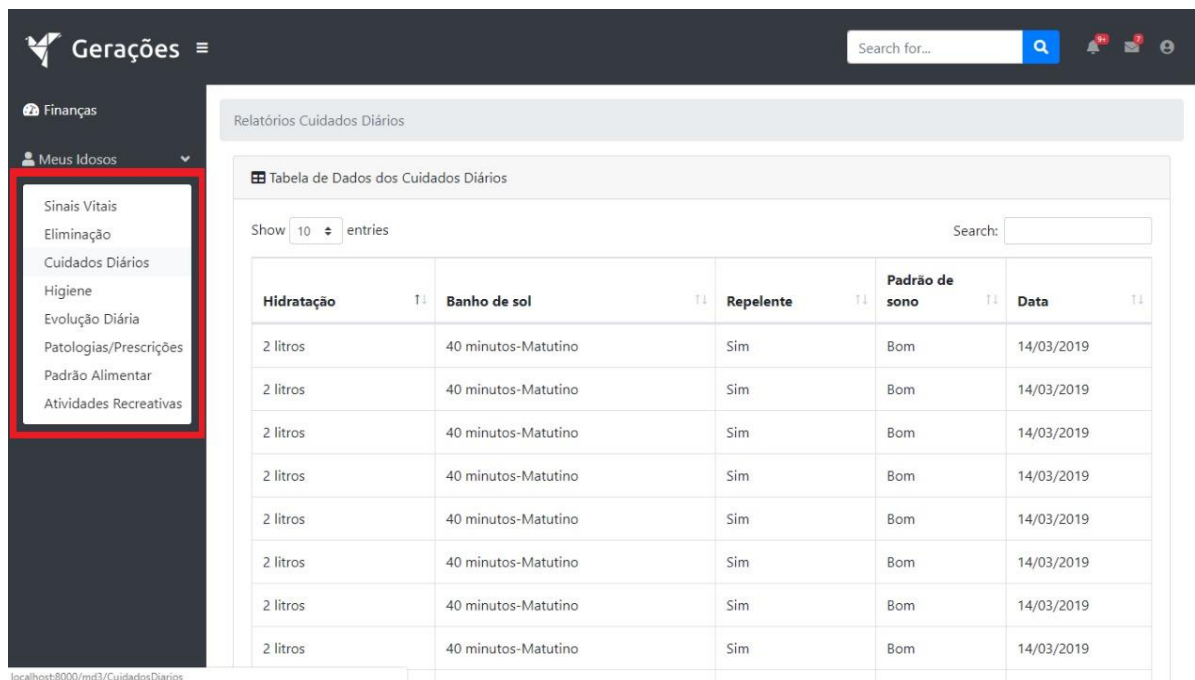


Figura 9 - Categorias de supervisão mostradas ao clicar o item "Meus idosos"

Ao clicar na opção “**Cuidados Diários**” o usuário seria redirecionado para uma página com uma tabela contendo dados sobre hidratação, banho de sol, repelente, padrão de sono e a data do registro:

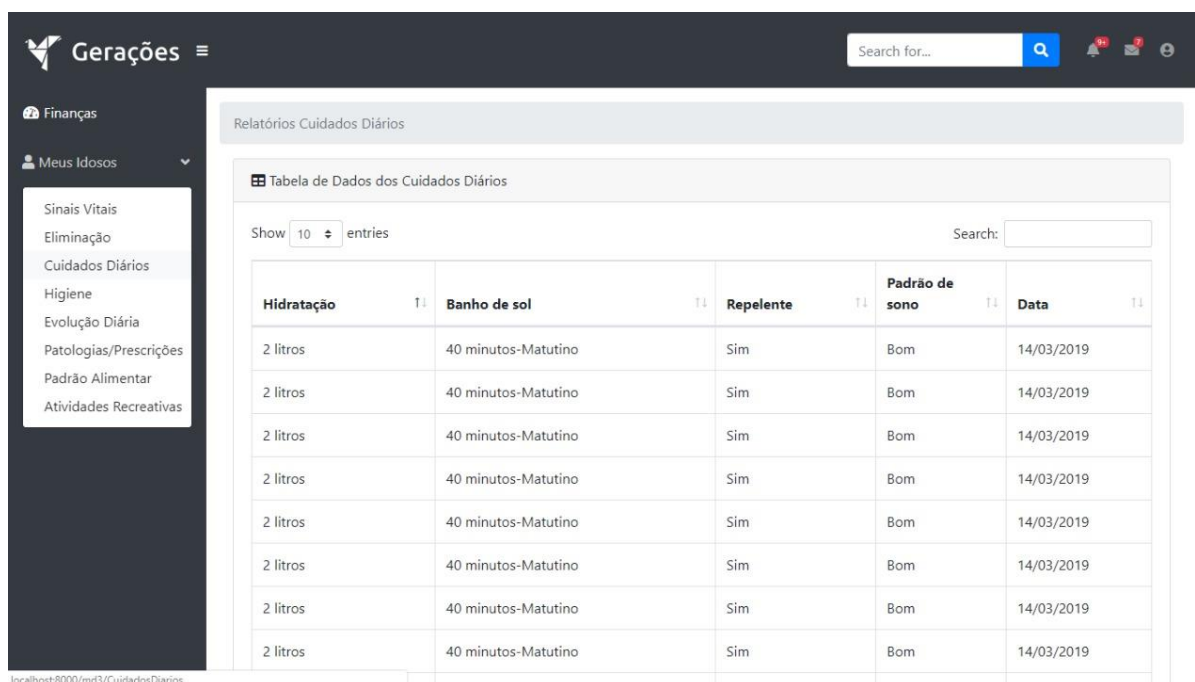
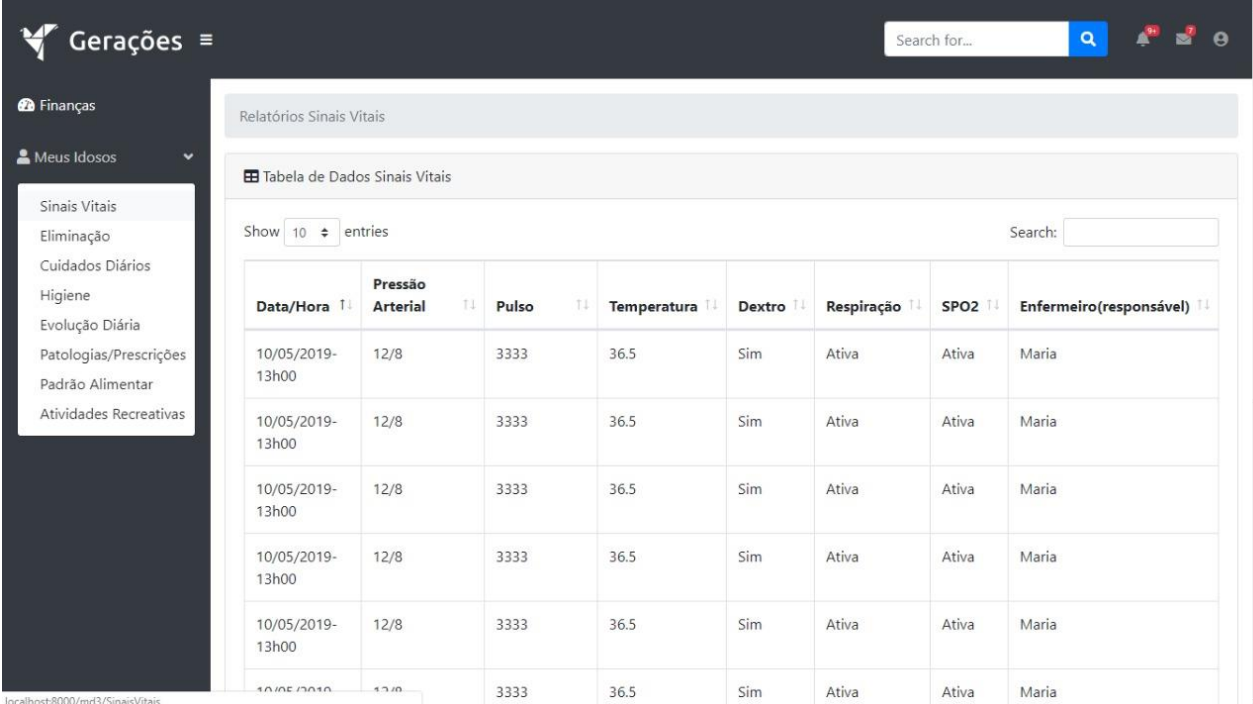


Figura 10 - Página protótipo "Cuidados Diários"

Ao clicar na opção “**Sinais Vitais**” o usuário seria redirecionado para uma página com uma tabela contendo valores sobre pressão arterial, pulso, temperatura, dextro, respiração e SPO2, além da data e hora do registro e o enfermeiro responsável pelo registro:

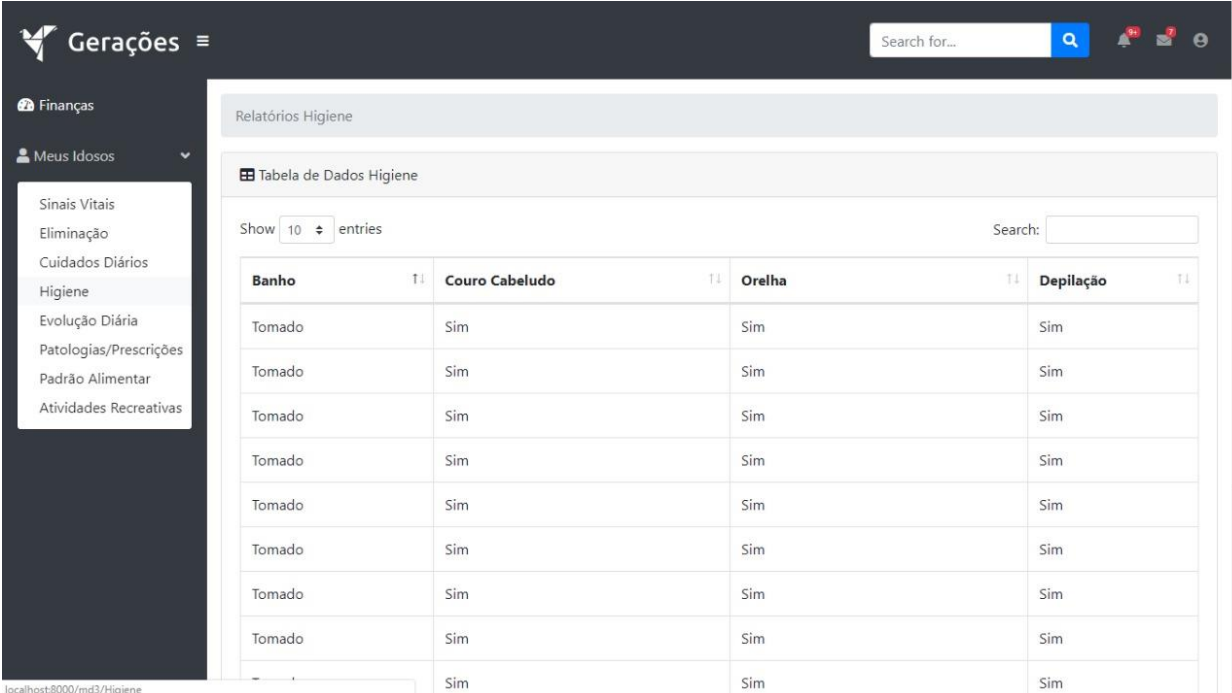


localhost:8000/md3/SinaisVitalis

Data/Hora	Pressão Arterial	Pulso	Temperatura	Dextro	Respiração	SPO2	Enfermeiro(responsável)
10/05/2019-13h00	12/8	3333	36.5	Sim	Ativa	Ativa	Maria
10/05/2019-13h00	12/8	3333	36.5	Sim	Ativa	Ativa	Maria
10/05/2019-13h00	12/8	3333	36.5	Sim	Ativa	Ativa	Maria
10/05/2019-13h00	12/8	3333	36.5	Sim	Ativa	Ativa	Maria
10/05/2019-13h00	12/8	3333	36.5	Sim	Ativa	Ativa	Maria

Figura 11 - Tela protótipo "Sinais Vitais"

Ao clicar na opção “**Higiene**” o usuário seria redirecionado para uma página com uma tabela contendo informações se o idoso vinculado a ele recebeu os devidos tratamentos sobre banho, couro cabeludo, orelha e depilação:

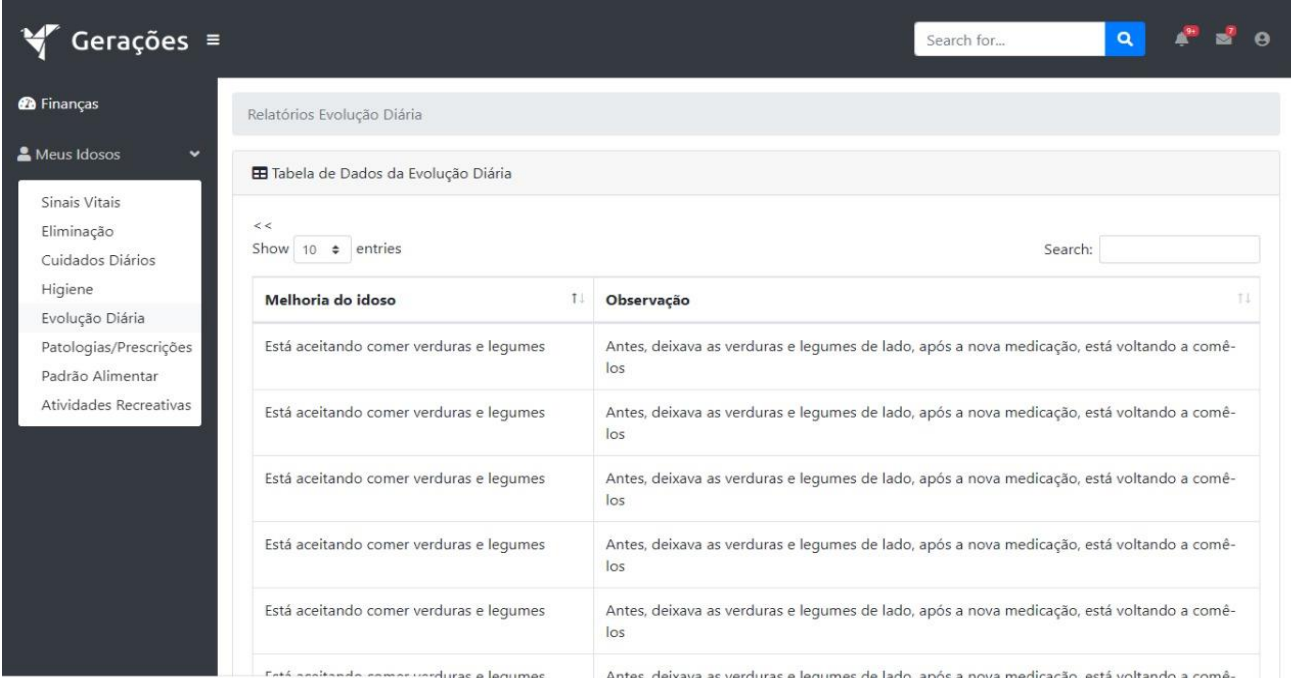


localhost:8000/md3/Higiene

Banho	Couro Cabeludo	Orelha	Depilação
Tomado	Sim	Sim	Sim
Tomado	Sim	Sim	Sim
Tomado	Sim	Sim	Sim
Tomado	Sim	Sim	Sim
Tomado	Sim	Sim	Sim
Tomado	Sim	Sim	Sim
Tomado	Sim	Sim	Sim
Tomado	Sim	Sim	Sim

Figura 12 - Tela protótipo "Higiene"

Ao clicar na opção “**Evolução diária**” o usuário seria redirecionado para uma página com uma tabela contendo informações sobre melhorias notadas no comportamento e no físico do idoso, além de um campo de observações:

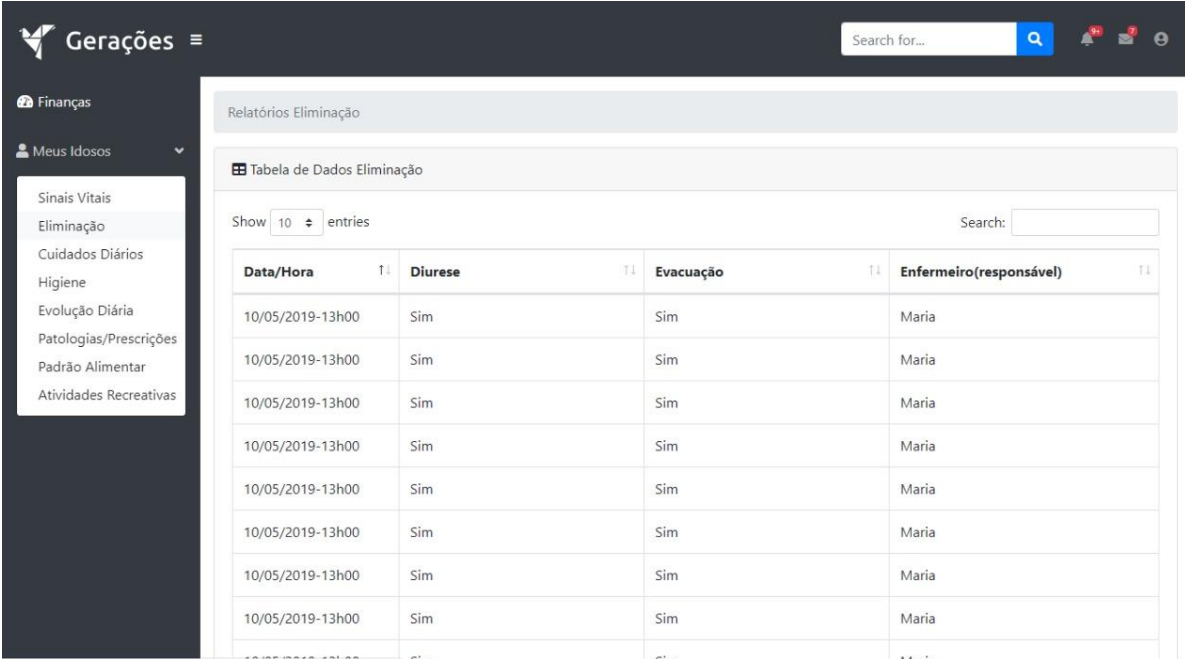


localhost:8000/md3/EvolucaoDiaria

Melhoria do idoso	Observação
Está aceitando comer verduras e legumes	Antes, deixava as verduras e legumes de lado, após a nova medicação, está voltando a comê-los
Está aceitando comer verduras e legumes	Antes, deixava as verduras e legumes de lado, após a nova medicação, está voltando a comê-los
Está aceitando comer verduras e legumes	Antes, deixava as verduras e legumes de lado, após a nova medicação, está voltando a comê-los
Está aceitando comer verduras e legumes	Antes, deixava as verduras e legumes de lado, após a nova medicação, está voltando a comê-los
Está aceitando comer verduras e legumes	Antes, deixava as verduras e legumes de lado, após a nova medicação, está voltando a comê-los
Está aceitando comer verduras e legumes	Antes, deixava as verduras e legumes de lado, após a nova medicação, está voltando a comê-los

Figura 13 - Tela protótipo "Evolução Diária"

Ao clicar na opção “**Eliminação**” o usuário seria redirecionado para uma página com uma tabela contendo informações sobre a frequência que o idoso vinculado ao usuário está realizando as eliminações de diurese e evacuação, além da data e a hora do registro e o enfermeiro responsável pelo registro:



localhost:8000/md3/Eliminacao

Data/Hora	Diurese	Evacuação	Enfermeiro(responsável)
10/05/2019-13h00	Sim	Sim	Maria
10/05/2019-13h00	Sim	Sim	Maria
10/05/2019-13h00	Sim	Sim	Maria
10/05/2019-13h00	Sim	Sim	Maria
10/05/2019-13h00	Sim	Sim	Maria
10/05/2019-13h00	Sim	Sim	Maria
10/05/2019-13h00	Sim	Sim	Maria
10/05/2019-13h00	Sim	Sim	Maria

Figura 14 - Tela protótipo "Eliminação"

Ao clicar na opção “**Padrão Alimentar**” o usuário seria redirecionado para uma página com uma tabela contendo informações sobre as refeições consumidas pelo idoso que o usuário está vinculado, além de seu grau de aceitação e a data do registro:

localhost:8000/md3/PadraoAlimentar

©2019 Todos os Direitos Reservados: PDS 2019

Figura 15 - Tela protótipo "Padrão Alimentar"

Ao clicar na opção “**Patologias e Prescrições**” o usuário seria redirecionado para uma página com três tabelas: uma contendo dados sobre as patologias do idoso, contendo os nomes das patologias, data de incidência e seu grau. Outra contendo dados sobre os medicamentos consumidos pelo idoso: o nome, a dosagem e a validade da prescrição dos medicamentos. E uma última contendo dados sobre as vacinações tomadas pelo idoso: para qual vírus a vacina foi destinada, a dosagem, data da vacinação e observações:

Relatórios Patologias e Prescrições

Tabela de Dados Patologias

Nome da Patologia	Data de incidência	Grau	Observações iniciais
Gripe Influenza	14/05/2019	Avançado	Sim
Gripe Influenza	14/05/2019	Avançado	Sim
Gripe Influenza	14/05/2019	Avançado	Sim
Gripe Influenza	14/05/2019	Avançado	Sim

Showing 1 to 4 of 4 entries

Showing 1 to 10 of 57 entries

Tabela de Dados Prescrições

Nome do Medicamento	Dosagem	Validade da prescrição	Observações
Prednisolona	1 comprimido 100mg pela manhã	O idoso deve estar alimentado antes de tomar a dose	
Prednisolona	1 comprimido 100mg pela manhã	O idoso deve estar alimentado antes de tomar a dose	
Prednisolona	1 comprimido 100mg pela manhã	O idoso deve estar alimentado antes de tomar a dose	
Prednisolona	1 comprimido 100mg pela manhã	O idoso deve estar alimentado antes de tomar a dose	

Showing 1 to 10 of 57 entries

Tabela de Dados Vacinações

Vacinação	Dosagem (ml)	Data da Vacina	Observações
Febre Amarela	15	23/05/2019	Devido a contração muscular, a área onde a vacina foi aplicada sofreu lesão
Febre Amarela	15	23/05/2019	Devido a contração muscular, a área onde a vacina foi aplicada sofreu lesão
Febre Amarela	15	23/05/2019	Devido a contração muscular, a área onde a vacina foi aplicada sofreu lesão

Figura 16 - Tela protótipo "Patologias e Prescrições"

Por último, ao clicar na opção “**Atividades Recreativas**” o usuário seria redirecionado para uma página contendo uma tabela com informações sobre as atividades físicas e recreativas realizadas pelo idoso no asilo que frequenta. A data e observações sobre a atividade realizada também são mostradas:

Relatórios Atividades Recreativas

Tabela de Dados das Atividades Recreativas Realizadas pelo idoso

Data	Atividade Realizada	Observações
14/05/2019	Sessão de Alongamento	O idoso conseguiu exercer com sucesso todos os exercícios de alongamento!
14/05/2019	Sessão de Alongamento	O idoso conseguiu exercer com sucesso todos os exercícios de alongamento!
14/05/2019	Sessão de Alongamento	O idoso conseguiu exercer com sucesso todos os exercícios de alongamento!
14/05/2019	Sessão de Alongamento	O idoso conseguiu exercer com sucesso todos os exercícios de alongamento!

Showing 1 to 4 of 4 entries

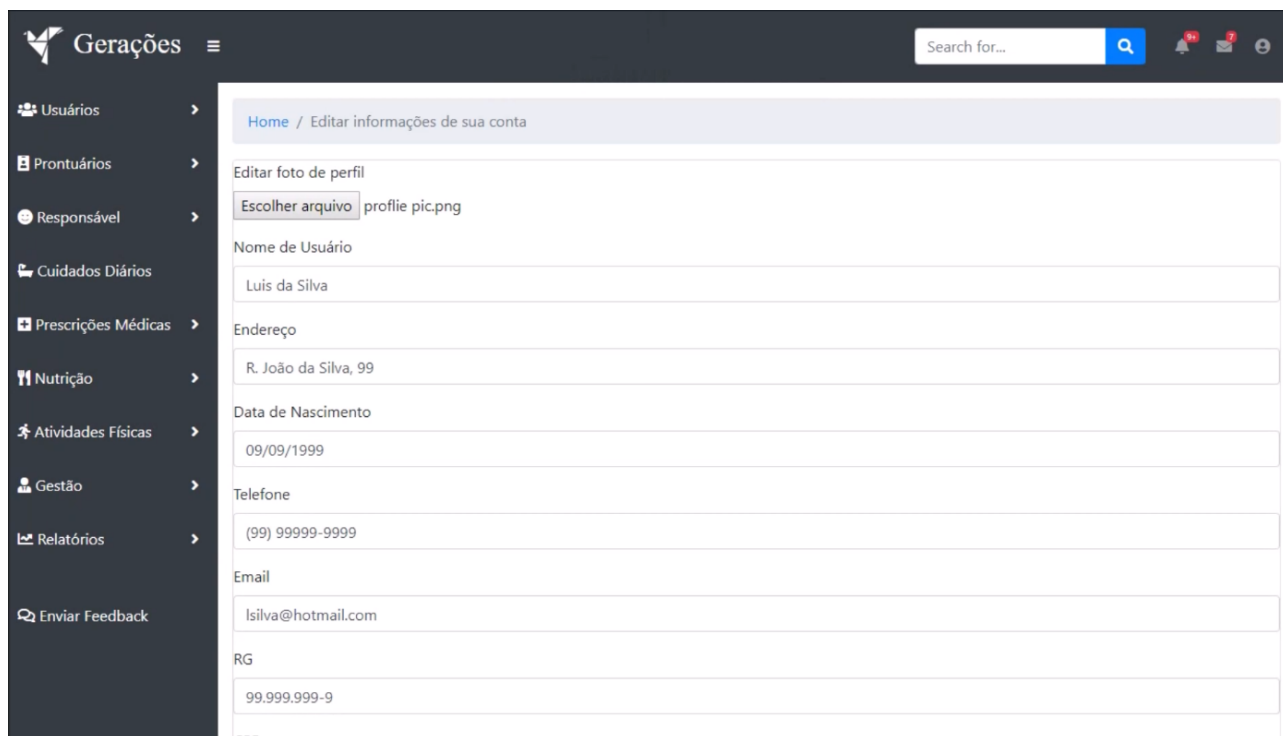
Showing 1 to 10 of 57 entries

Figura 17 - Tela protótipo "Atividades recreativas"

2.2.3 Refatoração dos Protótipos para as Funcionalidades do Módulo 03

O processo de refatoração dos códigos do sistema consistiu em coletar todas as páginas protótipo de todos os módulos e unificar em um único código. Para isso, foi utilizada a arquitetura MVC, possibilitando a criação de rotas para as diversas páginas do sistema. Aqui focaremos apenas na refatoração dos códigos do módulo 3.

Primeiramente, o menu lateral foi unificado em um único menu de categorias, e o mesmo anexado no menu principal do sistema, chamado *dashboard*:



The screenshot displays the 'Gerações' dashboard interface. On the left is a dark sidebar menu with categories: 'Usuários', 'Prontuários', 'Responsável', 'Cuidados Diários', 'Prescrições Médicas', 'Nutrição', 'Atividades Físicas', 'Gestão', 'Relatórios', and 'Enviar Feedback'. The main content area is titled 'Home / Editar informações de sua conta'. It contains a form for editing a user profile with the following fields: 'Editar foto de perfil' (with a file upload button 'Escolher arquivo' and a preview of 'profile pic.png'), 'Nome de Usuário' (filled with 'Luis da Silva'), 'Endereço' (filled with 'R. João da Silva, 99'), 'Data de Nascimento' (filled with '09/09/1999'), 'Telefone' (filled with '(99) 99999-9999'), 'Email' (filled with 'lsilva@hotmail.com'), and 'RG' (filled with '99.999.999-9'). The top of the dashboard features a search bar, a notification bell with a red badge, and a user profile icon.

Figura 18 - Dashboard com todas as categorias de todos os módulos.

Para que todas as páginas pudessem ser acessadas livremente, foi criado um sistema de rotas e um arquivo *controller* para todos os módulos. Além disso, foi preciso que todas as páginas do sistema fossem distinguidas em qual módulo elas pertencem a partir de uma *tag* `<section>`:

```

<section id="md3">
  <div class="container-fluid">
    <div id="content-wrapper">
      <ol class="breadcrumb">
        <li class="breadcrumb-item">
          <a href="#">Home</a>
        </li>
        <li class="breadcrumb-item active">Responsável</li>
        <li class="breadcrumb-item active">Editar informações de sua conta</li>
      </ol>
      <div class="card mb-3" id="cardprincipal">
        <form method="post" action="/md3/edicaoInfo?id=?php echo $this->getView()>
          <div class="form-group">
            <label>Nome de Usuário</label>
            <input name="nome" id="nome" type="text" class="form-control" plac
          </div>
          <...4 linhas />
          <...4 linhas />
          <...4 linhas />
          <div class="form-group">
            <label>Email</label>
            <input name="email" id="email" type="text" class="form-control" pl
          </div>
          <...4 linhas />
          <...4 linhas />
          <button class="btn btn-outline-secondary" type="submit" id="button-add
        </form>
      </div>
    </div>
  </div>
</section>

```

Figura 19 - Página "Editar Informações" dentro de uma *section* identificada como "md3".

```

//MOD03
new RouteUnique("md3-index",           "/md3/",           "Modulo03Controller", "index"),
new RouteUnique("md3-financas",        "/md3/financas",   "Modulo03Controller", "financas"),
new RouteUnique("md3-CuidadosDiarios", "/md3/CuidadosDiarios", "Modulo03Controller", "CuidadosDiarios"),
new RouteUnique("md3-SinaisVitalis",   "/md3/SinaisVitalis", "Modulo03Controller", "SinaisVitalis"),
new RouteUnique("md3-Eliminacao",      "/md3/Eliminacao",  "Modulo03Controller", "Eliminacao"),
new RouteUnique("md3-Higiene",         "/md3/Higiene",     "Modulo03Controller", "Higiene"),
new RouteUnique("md3-EvolucaoDiaria",  "/md3/EvolucaoDiaria", "Modulo03Controller", "EvolucaoDiaria"),
new RouteUnique("md3-PatologiasPrescricoes", "/md3/PatologiasPrescricoes", "Modulo03Controller", "PatologiasPrescricoes"),
new RouteUnique("md3-PadraoAlimentar", "/md3/PadraoAlimentar", "Modulo03Controller", "PadraoAlimentar"),
new RouteUnique("md3-AtividadesRecreativas", "/md3/AtividadesRecreativas", "Modulo03Controller", "AtividadesRecreativas"),
new RouteUnique("md3-EditarInfo",      "/md3/EditarInfo",  "Modulo03Controller", "EditarInfo"),
new RouteUnique("md3-edicaoInfo",       "/md3/edicaoInfo",   "InfoController", "edicaoInfo"),

```

Figura 20 - Rotas do módulo 3, situadas no arquivo "Route".

```

class Modulo03Controller extends Action{

    public function index(){
        $this->render('index','dashboard' , '..../');
    }

    public function finanzas(){

        $FinanceirosRespDAO = new \App\DAO\FinanceirosRespDAO();
        $listfinanzas = $FinanceirosRespDAO->listar();
        $this->getView()->dados = $listfinanzas;
        $this->render('finanzas','dashboard','../');
    }

    public function CuidadosDiarios(){
        $CuidadosDiariosRespDAO = new \App\DAO\CuidadosDiariosRespDAO();
        $cuidados_diarios = $CuidadosDiariosRespDAO->listarHidratacao();
        $this->getView()->dadosHidratacao = $cuidados_diarios;
        $cuidados_diarios = $CuidadosDiariosRespDAO->listarBanhoSol();
        $this->getView()->dadosBanhoSol = $cuidados_diarios;
        $cuidados_diarios = $CuidadosDiariosRespDAO->listarRepelente();
        $this->getView()->dadosRepelente = $cuidados_diarios;
        $cuidados_diarios = $CuidadosDiariosRespDAO->listarPadraoSono();
        $this->getView()->dadosPadraoSono = $cuidados_diarios;

        $this->render('CuidadosDiarios','dashboard','../');
    }
}

```

Figura 21 - Parte do arquivo Controller do módulo 3, onde cada função é uma página que o sistema consegue renderizar a partir das rotas criadas.

O processo de refatoração consistia também em criar uma pasta única para as modificações feitas no *layout* do sistema a partir de linguagens CSS e *JavaScript*, como no módulo 3 não houveram modificações usando tais linguagens, não será necessário aprofundar nesse tema.

2.2.4 Exemplificação do desenvolvimento das funcionalidades das iterações a partir das páginas-protótipo

Primeiramente, todos os casos de uso semelhantes foram agrupados em iterações e dado um prazo para entrega. A primeira iteração possuía os casos de uso mais simples, como exibir o menu de categorias e a página “Editar Informações”. Na segunda fora agrupada casos de uso que envolviam registros diários sobre o idoso, como “Cuidados Diários” e “Finanças”. Na terceira fora agrupada os casos de uso referentes ao padrão alimentar do idoso e suas eliminações. No quarto e último fora agrupado os casos de uso referentes às telas “Patologias e Prescrições” e “Atividades Recreativas”:

Para o desenvolvimento do *back-end* das páginas do módulo 3, foi preciso criar arquivos *model* (classes) e DAO (*Data Access Object*) para cada página (*view*). Desse modo, todos os arquivos para o funcionamento do sistema já estariam devidamente separados:

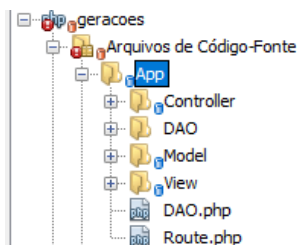


Figura 22 - Todas as pastas pertencentes à pasta *App*, além do arquivo "*Route*" e da classe abstrata *DAO*

Dentro da pasta *controller* ficavam todos os arquivos referentes a controle de rotas, na pasta *DAO* arquivos responsáveis pela conexão e interação com o banco de dados, na pasta *model* todos os arquivos são classes, onde cada uma delas representa uma página do sistema, e por último a pasta *view*, ficam nada mais que os arquivos *front-end* do sistema, ou seja, as páginas-protótipo. Como esta é uma seção de exemplificação, aqui será mostrado apenas as 3 principais funcionalidades do módulo 3: editar informações, listar com uma tabela e listar com duas ou mais tabelas.

2.2.4.1 Editar Informações

Primeiramente fora criado uma classe 'EditarInfo' com as variáveis apontadas como necessários no levantamento de requisitos e os métodos *get* e *set*:

```
<?php

namespace App\Model;

class EditarInfo {
    private $id;
    private $nome;
    private $endereco;
    private $data_nascimento;
    private $telefone;
    private $email;
    private $rg;
    private $cpf;

    public function __get($atributo) {
        return $this->$atributo;
    }

    public function __set($atributo, $valor) {
        $this->$atributo = $valor;
    }
}

?>
```

Figura 23 - Classe 'EditarInfo'

Após isso, um arquivo DAO que conectaria com o banco de dados para:

- 1) Atualizar os valores através do comando UPDATE, inserido em um método de alteração;
- 2) Buscar as informações do usuário a partir de seu ID, através dos comandos SELECT e WHERE dentro do método 'BuscarPorId';
- 3) Listar as informações do usuário através do comando SELECT inserido em um método de listagem.

```
public function alterar($editarInfo){
    $uDAO = new UsuarioDAO();
    $uDAO->verify();
    $sql = "UPDATE usuarios SET USU_NOME = ?, USU_EMAIL = ?, USU_CPF = ?, USU_RG = ?, USU_DATA_NASCIMENTO = ?, USU_TELEFONE = ?, USU_ENDERECO = ? WHERE USU_ID = ?";

    $stmt = $this->getConnection()->prepare($sql);

    $id = $editarInfo->__get('id'); echo 'id:'. $id;
    $nome = $editarInfo->__get('nome');
    $endereco = $editarInfo->__get('endereco');
    $data_nascimento = $editarInfo->__get('data_nascimento');
    $telefone = $editarInfo->__get('telefone');
    $email = $editarInfo->__get('email');
    $rg = $editarInfo->__get('rg');
    $cpf = $editarInfo->__get('cpf');

    $stmt->bindParam(1, $nome);
    $stmt->bindParam(2, $email);
    $stmt->bindParam(3, $cpf);
    $stmt->bindParam(4, $rg);
    $stmt->bindParam(5, $data_nascimento);
    $stmt->bindParam(6, $telefone);
    $stmt->bindParam(7, $endereco);
    $stmt->bindParam(8, $id);

    $stmt->execute();
}
```

Figura 24 - Método "alterar"

```
public function buscarPorId($id) {
    $uDAO = new UsuarioDAO();
    $uDAO->verify();

    $sql = "SELECT USU_ID, USU_NOME, USU_ENDERECO, USU_DATA_NASCIMENTO, USU_TELEFONE, USU_EMAIL, USU_RG, USU_CPF FROM usuarios WHERE USU_ID = :ID";
    $stmt = $this->getConnection()->prepare($sql);
    $stmt->bindParam(":ID", $id);
    $stmt->execute();
    $result = $stmt->fetch();

    if ($result > 0){
        $editarInfo = new EditarInfo();
        $editarInfo->__set('id', $result['USU_ID']);
        $editarInfo->__set('nome', $result['USU_NOME']);
        $editarInfo->__set('endereco', $result['USU_ENDERECO']);
        $editarInfo->__set('data_nascimento', $result['USU_DATA_NASCIMENTO']);
        $editarInfo->__set('telefone', $result['USU_TELEFONE']);
        $editarInfo->__set('email', $result['USU_EMAIL']);
        $editarInfo->__set('rg', $result['USU_RG']);
        $editarInfo->__set('cpf', $result['USU_CPF']);
        return $editarInfo;
    }
    else{
        $editarInfo = new EditarInfo();
        $editarInfo->__set('id', '');
        return $editarInfo;
    }
}
```

Figura 25 - Método "buscarPorId"


```

public function listar() {
    $uDAO = new UsuarioDAO();
    $uDAO->verify();

    $infoalterada = array();
    $sql = "SELECT USU_ID, USU_NOME, USU_ENDereco, USU_DATA_NASCIMENTO, USU_TELEFONE, USU_EMAIL, USU_RG, USU_CPF FROM usuarios";
    $stmt = $this->getConnection()->prepare($sql);
    $stmt->execute();
    $results = $stmt->fetchAll(PDO::FETCH_ASSOC);

    foreach ($results as $row) {
        $editarInfo = new EditarInfo();
        $editarInfo->__set('id', $row['USU_ID']);
        $editarInfo->__set('nome', $row['USU_NOME']);
        $editarInfo->__set('endereco', $row['USU_ENDereco']);
        $editarInfo->__set('data_nascimento', $row['USU_DATA_NASCIMENTO']);
        $editarInfo->__set('telefone', $row['USU_TELEFONE']);
        $editarInfo->__set('email', $row['USU_EMAIL']);
        $editarInfo->__set('rg', $row['USU_RG']);
        $editarInfo->__set('cpf', $row['USU_CPF']);
        array_push($infoalterada, $editarInfo);
    }

    return $infoalterada;
}
}

```

Figura 26 - Método 'listar'

O próximo passo foi declarar o caminho da rota e que o método a ser usado para coletar os dados do formulário seria o método GET, e além disso foi colocado em cada campo do formulário um código de listagem, para que assim que a página carregasse, os campos estariam preenchidos com os dados salvos no banco de dados, porém editáveis. Tais dados seriam encontrados através do método 'BuscarPorId':

```

<div class="card mb-3" id="cardprincipal">
    <form method="post" action="/md3/edicaoInfo?id=<?php echo $this->getView()->dados->__get('id') ?>" class="form">
        <div class="form-group">
            <label>Nome de Usuário</label>
            <input name="nome" id="nome" type="text" class="form-control" placeholder="Coloque seu nome de usuário" value="<?php echo $this->getView()->dados->__get('nome');?>" />
        </div>
        <div class="form-group">
            <label>Endereço</label>
            <input name="endereco" id="endereco" type="text" class="form-control" placeholder="Coloque seu endereço" value="<?php echo $this->getView()->dados->__get('endereco');?>" />
        </div>
        <div class="form-group">
            <label>Data de Nascimento</label>
            <input name="data_nascimento" id="data_nascimento" type="text" class="form-control" placeholder="dd/mm/aaaa" value="<?php echo $this->getView()->dados->__get('data_nascimento');?>" />
        </div>
        <div class="form-group">
            <label>Telefone</label>
            <input name="telefone" id="telefone" type="text" class="form-control" placeholder="(99) 99999-9999" value="<?php echo $this->getView()->dados->__get('telefone');?>" />
        </div>
        <div class="form-group">
            <label>Email</label>
            <input name="email" id="email" type="text" class="form-control" placeholder="nome@exemplo.com" value="<?php echo $this->getView()->dados->__get('email');?>" />
        </div>
        <div class="form-group">
            <label>RG</label>
            <input name="rg" id="rg" type="text" class="form-control" placeholder="99.999.999-9" value="<?php echo $this->getView()->dados->__get('rg');?>" />
        </div>
        <div class="form-group">
            <label>CPF</label>
            <input name="cpf" id="cpf" type="text" class="form-control" placeholder="999.999.999-99" value="<?php echo $this->getView()->dados->__get('cpf');?>" />
        </div>
        <button class="btn btn-outline-secondary" type="submit" id="button-addon2">Alterar</button>
    </form>
</div>

```

Figura 27 - Código da página "Editar Informações"

Por último foi preciso organizar as rotas no arquivo *controller*, para que assim pudesse ser conectado todos os métodos para suas respectivas funcionalidades:

```

public function EditarInfo() {
    $EditarInfoDAO = new \App\DAO\EditarInfoDAO();

    $editarinfo = $EditarInfoDAO->buscarPorId($_SESSION['id']);

    $this->getView()->dados = $editarinfo;
    $this->render('EditarInfo', 'dashboard', '../');
}
}

```

Figura 28 - Método controlador de rotas da página "Editar Informações"

2.2.4.2 Listar informações em uma tabela

Para esta seção utilizaremos a página “Finanças”, pois possui apenas uma tabela que listará valores referentes ao pagamento da estadia do idoso. Os primeiros passos foram semelhantes ao processo citado acima: foi criada uma classe com as variáveis listadas necessários e os métodos *get* e *set*. Note que as variáveis estão diferentes das criadas nas páginas protótipo, pois eram estes os campos existentes no banco de dados:

```

<?php
namespace App\Model;

class FinanceirosResp {
    private $DataVencimento;
    private $TipoPagamento;
    private $valor;
    private $TipoPlano;

    public function __get($atributo) {
        return $this->$atributo;
    }

    public function __set($atributo, $valor) {
        $this->$atributo = $valor;
    }
}

```

Figura 29 - Classe "FinanceirosResp"

O arquivo DAO seria mais simples também, afinal esta é uma página que necessita apenas do recurso de listagem. Foi usado os comandos SELECT e WHERE para identificar e listar apenas os valores pertencentes ao usuário que está logado:

```

public function listar() {

    $uDAO = new UsuarioDAO();
    $uDAO->verify();

    $listfinancas = array();

    $sql = "SELECT * FROM pagamentos_idosos";
    $stmt = $this->getConn()->prepare($sql);
    $stmt->execute();
    $results = $stmt->fetchAll(\PDO::FETCH_ASSOC);

    foreach ($results as $row) {
        $financas = new FinanceirosResp();
        $financas->__set('DataVencimento',$row['PAG_DATA_VENCIMENTO']);
        $financas->__set('TipoPagamento',$row['PAG_TIPO_PAGAMENTO']);
        $financas->__set('valor',$row['PAG_VALOR']);
        $financas->__set('TipoPlano',$row['PAG_PLANO_TIPO']);
        array_push($listfinancas, $financas);
    }

    return $listfinancas;
}

```

Figura 30 - método "listar" da página "Finanças"

Em seguida, foi necessário criar as linhas de código que listariam os dados na tabela a partir do método *get*:

```

<ol class="breadcrumb">
    <li class="breadcrumb-item">
        <a href="#">Home</a>
    </li>
    <li class="breadcrumb-item active">Responsável</li>
    <li class="breadcrumb-item active">Finanças</li>
</ol>

<!-- DataTables Example -->
<div class="card mb-3">
    <div class="card-header">
        <i class="fas fa-table"></i>
        Tabela de Dados</div>
    <div class="card-body">
        <div class="table-responsive">
            <div id="dataTable_wrapper" class="dataTables_wrapper dt-bootstrap4">

                <div class="row"><div class="col-sm-12">
                    <table class="table table-bordered dataTable" id="dataTable" width="100%" cellspacing="0" role="grid" aria-describedby="dataTable_info">
                        <thead>
                            <tr role="row"><th class="sorting_asc" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-sort="ascending" aria-label="Data de pagamento:">
                                Data de vencimento</th>
                            <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="Período referente ao pagamento">
                                Forma de pagamento </th>
                            <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="Valor do pagamento">
                                Valor pagamento</th>
                            <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="Forma de pagamento">
                                Tipo de plano</th>
                        </thead>
                        <tfoot>
                        </tfoot>
                        <tbody>

                            <?php foreach ($this->getView()->dados as $financas) { ?>
                                <tr>
                                    <td><?php echo $financas->__get('DataVencimento'); ?></td>
                                    <td><?php echo $financas->__get('TipoPagamento'); ?></td>
                                    <td><?php echo $financas->__get('valor'); ?></td>
                                    <td><?php echo $financas->__get('TipoPlano'); ?></td>
                                </tr>
                            <?php } ?>

                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</div>

```

Figura 31 - Código da página "Finanças"

Por último foi criado um método controlador de rotas para a página, assim o método “listar” poderia encontrar a página certa a ser usado:

```
public function finanzas() {  
  
    $FinanceirosRespDAO = new \App\DAO\FinanceirosRespDAO();  
    $listfinancas = $FinanceirosRespDAO->listar();  
    $this->getView()->dados = $listfinancas;  
    $this->render('financas','dashboard','../');  
}
```

Figura 32 - método controlador de rotas da página "Finanças"

2.2.4.3 Listar informações em duas tabelas ou mais

Esta foi a funcionalidade mais complexa de se desenvolver do módulo 3, pois foi preciso criar uma classe para cada tabela, porque caso contrário no momento de se obter os dados do banco de dados as tabelas existentes na página tentariam obtê-los de uma vez. Para exemplificar essa funcionalidade, usaremos a página “Cuidados Diários” pois ela possui um total de quatro tabelas, então como foi explicado antes, foi preciso criar quatro classes distintas para tal página, as variáveis possuem o mesmo nome que as variáveis do banco de dados para melhor interpretação do código, e para diferenciar as classes e suas variáveis foram colocados identificadores de 1 a 4:

```
<?php  
  
namespace App\Model;  
  
class CuidadosDiariosRespl {  
    private $CUD_ID1;  
    private $CUD_DATA1;  
    private $CUD_HORAI;  
    private $FK TIPOS CUIDADOS DIARIOS CDT ID1;  
    private $FK ENFERMEIROS USU ID1;  
    private $CDT_CUIDADO1;  
    private $USU_NOME1;  
  
    public function __get($atributo) {  
        return $this->$atributo;  
    }  
  
    public function __set($atributo, $valor) {  
        $this->$atributo = $valor;  
    }  
}
```

Figura 33 - Classe 1 da página "Cuidados Diários"

```

<?php

namespace App\Model;

class CuidadosDiariosResp2 {
    private $CUD_ID2;
    private $CUD_DATA2;
    private $CUD_HORA2;
    private $FK TIPOS CUIDADOS DIARIOS CDT ID2;
    private $FK ENFERMEIROS USU ID2;
    private $CDT_CUIDADO2;
    private $USU_NOME2;
    private $PERIODO;

    public function __get($atributo) {
        return $this->$atributo;
    }

    public function __set($atributo, $valor) {
        $this->$atributo = $valor;
    }
}

```

Figura 34 - Classe 2 da página "Cuidados Diários"

```

<?php

namespace App\Model;

class CuidadosDiariosResp3 {
    private $CUD_ID3;
    private $CUD_DATA3;
    private $CUD_HORA3;
    private $FK TIPOS CUIDADOS DIARIOS CDT ID3;
    private $FK ENFERMEIROS USU ID3;
    private $CDT_CUIDADO3;
    private $USU_NOME3;

    public function __get($atributo) {
        return $this->$atributo;
    }

    public function __set($atributo, $valor) {
        $this->$atributo = $valor;
    }
}

```

Figura 35 - Classe 3 da página "Cuidados Diários"

```

<?php

namespace App\Model;

class CuidadosDiariosResp3 {
    private $CUD_ID3;
    private $CUD_DATA3;
    private $CUD_HORA3;
    private $FK TIPOS CUIDADOS DIARIOS CDT ID3;
    private $FK ENFERMEIROS USU ID3;
    private $CDT CUIDADO3;
    private $USU_NOME3;

    public function __get($atributo) {
        return $this->$atributo;
    }

    public function __set($atributo, $valor) {
        $this->$atributo = $valor;
    }
}

```

Figura 36 - Classe 4 da página "Cuidados Diários"

O arquivo DAO também foi diferente, para cada tabela foi preciso criar um método de listagem, usando os comandos SELECT e WHERE, esse último usado para identificar o ID do cuidado diário (repelente, padrão de sono, banho de sol ou hidratação). Além desses dois comandos SQL foi preciso o uso do comando INNER JOIN pois haviam dados que precisavam ser listados mais que se encontravam em tabelas separadas. Cada classe seria destinada para um cuidado diário específico, ou seja, para uma tabela específica:

```

public function listarHidratacao() {
    $uDAO = new UsuarioDAO();
    $uDAO->verify();

    $cuidados_diarios = array();
    $sql = "SELECT cd.`CUD_ID`, cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`, cd.`CUD_DATA`, cd.`CUD_HORA`, tcd.CDT_CUIDADO, usu.USU_NOME FROM `cuidados_diarios` as cd
    Inner join tipos_cuidados_diarios as tcd on (tcd.CDT_ID = cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`)
    INNER JOIN usuarios as usu on (usu.USU_ID = `FK_ENFERMEIROS_USU_ID`)
    INNER JOIN idosos as ido on (ido.IDS_PRONTUARIO = `FK_IDOSOS_IDS_PRONTUARIO`)
    WHERE cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID` = 2
    ORDER BY `cd`.`CUD_ID` ASC";

    $stmt = $this->getConn()->prepare($sql);
    $stmt->execute();
    $results = $stmt->fetchAll(\PDO::FETCH_ASSOC);

    foreach ($results as $row) {
        $CuidadosDiarios = new CuidadosDiariosResp1();
        $CuidadosDiarios->__set('$CUD_ID1', $row['CUD_ID']);
        $CuidadosDiarios->__set('$CUD_DATA1', $row['CUD_DATA']);
        $CuidadosDiarios->__set('$CUD_HORA1', $row['CUD_HORA']);
        $CuidadosDiarios->__set('$CDT_CUIDADO1', $row['CDT_CUIDADO']);
        $CuidadosDiarios->__set('$USU_NOME1', $row['USU_NOME']);

        array_push($cuidados_diarios, $CuidadosDiarios);
    }

    return $cuidados_diarios;
}

```

Figura 37 - Método “listar” da tabela "Hidratação"

```

public function listarBanhoSol() {
    $uDAO = new UsuarioDAO();
    $uDAO->verify();

    $cuidados_diarios = array();
    $sql = "SELECT cd.`CUD_ID`, cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`, cd.`CUD_DATA`, cd.`CUD_HORA`, tcd.CDT_CUIDADO, usu.USU_NOME FROM `cuidados_diarios` as cd
    Inner join tipos_cuidados_diarios as tcd on (tcd.CDT_ID = cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`)
    INNER JOIN usuarios as usu on (usu.USU_ID = `FK_ENFERMEIROS_USU_ID`)
    INNER JOIN idosos as ido on (ido.IDS_PRONTUARIO = `FK_IDOSOS_IDS_PRONTUARIO`)
    WHERE cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID` = 2
    ORDER BY `cd`.`CUD_ID` ASC";

    $stmt = $this->getConn()->prepare($sql);
    $stmt->execute();
    $results = $stmt->fetchAll(\PDO::FETCH_ASSOC);

    foreach ($results as $row) {
        $CuidadosDiarios = new CuidadosDiariosResp2();
        $CuidadosDiarios->__set('$CUD_ID2', $row['CUD_ID']);
        $CuidadosDiarios->__set('$CUD_DATA2', $row['CUD_DATA']);
        $CuidadosDiarios->__set('$CUD_HORA2', $row['CUD_HORA']);
        $CuidadosDiarios->__set('$FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID2', $row['FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID']);
        $CuidadosDiarios->__set('$USU_NOME2', $row['USU_NOME']);

        array_push($cuidados_diarios, $CuidadosDiarios);
    }

    return $cuidados_diarios;
}

```

Figura 38 - Método “listar” da tabela "Banho de Sol"

```

public function listarRepelente() {
    $uDAO = new UsuarioDAO();
    $uDAO->verify();

    $cuidados_diarios = array();
    $sql = "SELECT cd.`CUD_ID`, cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`, cd.`CUD_DATA`, cd.`CUD_HORA`, tcd.CDT_CUIDADO, usu.USU_NOME FROM `cuidados_diarios` as cd
    Inner join tipos_cuidados_diarios as tcd on (tcd.CDT_ID = cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`)
    INNER JOIN usuarios as usu on (usu.USU_ID = `FK_ENFERMEIROS_USU_ID`)
    INNER JOIN idosos as ido on (ido.IDS_PRONTUARIO = `FK_IDOSOS_IDS_PRONTUARIO`)
    WHERE cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID` = 2
    ORDER BY `cd`.`CUD_ID` ASC";

    //echo $sql;
    //exit();

    $stmt = $this->getConn()->prepare($sql);
    $stmt->execute();
    $results = $stmt->fetchAll(\PDO::FETCH_ASSOC);

    foreach ($results as $row) {
        $CuidadosDiarios = new CuidadosDiariosResp3();
        $CuidadosDiarios->__set('$CUD_ID3', $row['CUD_ID']);
        $CuidadosDiarios->__set('$CUD_DATA3', $row['CUD_DATA']);
        $CuidadosDiarios->__set('$CUD_HORA3', $row['CUD_HORA']);
        $CuidadosDiarios->__set('$CDT_CUIDADO3', $row['CDT_CUIDADO']);
        $CuidadosDiarios->__set('$USU_NOME3', $row['USU_NOME']);

        array_push($cuidados_diarios, $CuidadosDiarios);
    }

    return $cuidados_diarios;
}

```

Figura 39 - Método "listar" da tabela "Repelente"

```

public function listarPadraoSono() {
    $uDAO = new UsuarioDAO();
    $uDAO->verify();

    $cuidados_diarios = array();
    $sql = "SELECT cd.`CUD_ID`, cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`, cd.`CUD_DATA`, cd.`CUD_HORA`, tcd.CDT_CUIDADO, usu.USU_NOME FROM `cuidados_diarios` as cd
    Inner join tipos_cuidados_diarios as tcd on (tcd.CDT_ID = cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID`)
    INNER JOIN usuarios as usu on (usu.USU_ID = `FK_ENFERMEIROS_USU_ID`)
    INNER JOIN idosos as ido on (ido.IDS_PRONTUARIO = `FK_IDOSOS_IDS_PRONTUARIO`)
    WHERE cd.`FK_TIPOS_CUIDADOS_DIARIOS_CDT_ID` = 2
    ORDER BY `cd`.`CUD_ID` ASC";

    // echo $sql;
    //exit();

    $stmt = $this->getConn()->prepare($sql);
    $stmt->execute();
    $results = $stmt->fetchAll(\PDO::FETCH_ASSOC);

    foreach ($results as $row) {
        $CuidadosDiarios = new CuidadosDiariosResp4();
        $CuidadosDiarios->__set('$CUD_ID4', $row['CUD_ID']);
        $CuidadosDiarios->__set('$CUD_DATA4', $row['CUD_DATA']);
        $CuidadosDiarios->__set('$CUD_HORA4', $row['CUD_HORA']);
        $CuidadosDiarios->__set('$CDT_CUIDADO4', $row['CDT_CUIDADO']);
        $CuidadosDiarios->__set('$USU_NOME4', $row['USU_NOME']);

        array_push($cuidados_diarios, $CuidadosDiarios);
    }

    return $cuidados_diarios;
}

```

Figura 40 - Método "listar" da tabela "Padrão de Sono"

O processo de listagem foi semelhante ao usado na listagem para uma tabela. A diferença é que haviam quatro tabelas para inserir os códigos de listagem e cada tabela possui seu próprio conjunto de variáveis:


```

<div class="card mb-3">
  <div class="card-header">
    <i class="fas fa-table"></i>
    Tabela de Dados Hidratação
  </div>
  <div class="card-body">
    <div class="table-responsive">
      <div id="dataTable_wrapper" class="dataTables_wrapper dt-bootstrap4">
        <div class="row">
          <div class="col-sm-12">
            <table class="table table-bordered dataTable" id="dataTable" width="100%" cellspacing="0" role="grid" aria-describedby="dataTable_info">
              <thead>
                <tr role="row">
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">Data</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">Hora</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">Enfermeiro(a) Supervisor(a)</th>
                </tr>
              </thead>
              <tfoot>
            </tfoot>
            <tbody>
              <?php foreach ($this->getView()->dadosHidracao as $CuidadosDiarios) { ?>
                <tr>
                  <td><?php echo $CuidadosDiarios->__get('$CUD_DATA1'); ?></td>
                  <td><?php echo $CuidadosDiarios->__get('$CUD_HORA1'); ?></td>
                  <td><?php echo $CuidadosDiarios->__get('$USU_NOME1'); ?></td>
                </tr>
              <?php } ?>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figura 41 - Código da tabela "Hidratação"

```

<div class="card mb-3">
  <div class="card-header">
    <i class="fas fa-table"></i>
    Tabela de Dados Banho de Sol
  </div>
  <div class="card-body">
    <div class="table-responsive">
      <div id="dataTable_wrapper" class="dataTables_wrapper dt-bootstrap4">
        <div class="row">
          <div class="col-sm-12">
            <table class="table table-bordered dataTable" id="dataTable" width="100%" cellspacing="0" role="grid" aria-describedby="dataTable_info">
              <thead>
                <tr role="row">
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">Data</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">Hora</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">Enfermeiro(a) Supervisor(a)</th>
                </tr>
              </thead>
              <tfoot>
            </tfoot>
            <tbody>
              <?php foreach ($this->getView()->dadosBanhoSol as $CuidadosDiarios) { ?>
                <tr>
                  <td><?php echo $CuidadosDiarios->__get('$CUD_DATA2'); ?></td>
                  <td><?php echo $CuidadosDiarios->__get('$CUD_HORA2'); ?></td>
                  <td><?php echo $CuidadosDiarios->__get('$USU_NOME2'); ?></td>
                </tr>
              <?php } ?>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figura 42 - Código da tabela "Banho de sol"

```

<div class="card mb-3">
  <div class="card-header">
    <i class="fas fa-table"></i>
    Tabela de Dados Repelente
  </div>
  <div class="card-body">
    <div class="table-responsive">
      <div id="dataTable_wrapper" class="dataTables_wrapper dt-bootstrap4">
        <div class="row">
          <div class="col-sm-12">
            <table class="table table-bordered dataTable" id="dataTable" width="100%" cellspacing="0" role="grid" aria-describedby="dataTable_info">
              <thead>
                <tr role="row">
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">
                    Data</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">
                    Hora</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">
                    Enfermeiro(a) Supervisor(a)</th>
                </tr>
              </thead>
              <tfoot>
                <tfoot>
              </tfoot>
              <tbody>
                <?php foreach ($this->getView()->dadosRepelente as $CuidadosDiarios) { ?>
                  <tr>
                    <td><?php echo $CuidadosDiarios->__get('$CUD_DATA3');?></td>
                    <td><?php echo $CuidadosDiarios->__get('$CUD_HORA3');?></td>
                    <td><?php echo $CuidadosDiarios->__get('$USU_NOME3');?></td>
                  </tr>
                <?php } ?>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figura 43 - Código da tabela "Repelente"

```

<div class="card mb-3">
  <div class="card-header">
    <i class="fas fa-table"></i>
    Tabela de Dados Padrão de Sono
  </div>
  <div class="card-body">
    <div class="table-responsive">
      <div id="dataTable_wrapper" class="dataTables_wrapper dt-bootstrap4">
        <div class="row">
          <div class="col-sm-12">
            <table class="table table-bordered dataTable" id="dataTable" width="100%" cellspacing="0" role="grid" aria-describedby="dataTable_info">
              <thead>
                <tr role="row">
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">
                    Data</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">
                    Hora</th>
                  <th class="sorting" tabindex="0" aria-controls="dataTable" rowspan="1" colspan="1" aria-label="">
                    Enfermeiro(a) Supervisor(a)</th>
                </tr>
              </thead>
              <tfoot>
                <tfoot>
              </tfoot>
              <tbody>
                <?php foreach ($this->getView()->dadosPadraoSono as $CuidadosDiarios) { ?>
                  <tr>
                    <td><?php echo $CuidadosDiarios->__get('$CUD_DATA4');?></td>
                    <td><?php echo $CuidadosDiarios->__get('$CUD_HORA4');?></td>
                    <td><?php echo $CuidadosDiarios->__get('$USU_NOME4');?></td>
                  </tr>
                <?php } ?>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figura 44 - Código da tabela "Padrão de Sono"

E finalmente temos o método controlador de rotas. Para esse, foi preciso criar uma variável de listagem para cada tabela, exceto isso ele é semelhante a todas as outras funcionalidades citadas acima:

```
public function CuidadosDiarios() {  
    $CuidadosDiariosRespDAO = new \App\DAO\CuidadosDiariosRespDAO();  
    $cuidados_diarios = $CuidadosDiariosRespDAO->listarHidratacao();  
    $this->getView()->dadosHidratacao = $cuidados_diarios;  
    $cuidados_diarios = $CuidadosDiariosRespDAO->listarBanhoSol();  
    $this->getView()->dadosBanhoSol = $cuidados_diarios;  
    $cuidados_diarios = $CuidadosDiariosRespDAO->listarRepelente();  
    $this->getView()->dadosRepelente = $cuidados_diarios;  
    $cuidados_diarios = $CuidadosDiariosRespDAO->listarPadraoSono();  
    $this->getView()->dadosPadraoSono = $cuidados_diarios;  
  
    $this->render('CuidadosDiarios','dashboard','../');  
}
```

Figura 45 - Método controlador da página "Cuidados Diários"

A única página a usar a funcionalidade de alterar informações fora apenas a página usada como exemplo (Editar Informações), as páginas que possuíam listagem em uma tabela foram poucas, como “Finanças”, “Padrão Alimentar” e “Evolução Diária”, já as páginas com duas ou mais tabelas foram “Cuidados Diários”, “Sinais Vitais”, “Higiene”, “Eliminação”, “Patologias e Prescrições” e “Atividades Recreativas”. Muitas páginas tiveram tabelas adicionadas devido a necessidade de adaptação as tabelas do banco de dados.

3 Conclusões e Recomendações

3.1 Recapitulação

Foi proposto aos alunos do quarto ano de informática de 2019, do campus São João da Boa Vista do Instituto Federal de Ciência e Tecnologia de São Paulo, analisar, desenvolver e implantar um portal especializado para cuidado de idosos, especialmente voltado para instituições de longa permanência. Tendo como motivação o fato de que a expectativa de vida na cidade de São João da Boa Vista aumentou de 73 para 77 anos e também foi eleita a melhor cidade para se viver após os 60 anos. Este projeto foi dividido em 3 subsistemas com 3 módulos cada e cada módulo com uma funcionalidade específica. O módulo de acompanhamento dos familiares, que fora abordado neste documento, teve como principal finalidade englobar diversas funcionalidades para serem direcionadas exclusivamente para o uso dos familiares dos idosos. O objetivo geral deste documento fora mostrar o processo de desenvolvimento desse módulo, como fora dividido suas iterações e exemplificando suas principais funcionalidades.

3.2 Etapas desenvolvidas ao longo do trabalho

Primeiramente, fora mostrado o processo de levantamento de requisitos (tendo como base os macrerequisitos) e estes separados em requisitos funcionais (garantem o funcionamento do sistema) e não-funcionais (garantem a qualidade do sistema). Ao final do processo, foram levantados um total de treze requisitos funcionais e oito requisitos não funcionais.

Com os requisitos levantados, o próximo passo foi o desenvolvimento do diagrama de casos de uso, e com base nesse o desenvolvimento das páginas protótipo: páginas com apenas *front-end* pré-determinado. Nessa etapa foram desenvolvidas um total de dez páginas protótipo.

A refatoração dos códigos para o projeto parcialmente integrado foi o passo seguinte. Nessa etapa todos os códigos de todos os módulos foram unificados em um único código refatorado, e usando a arquitetura MVC, foram criadas rotas para todas as páginas de todos os módulos para que assim todas as páginas pudessem ser acessadas livremente.

Com os códigos refatorados, era a vez de desenvolver o *back-end* das páginas protótipo, e para isso os casos de uso foram divididos em iterações (grupos de casos de uso com funcionalidades semelhantes). Algumas páginas protótipo tiveram de ser adaptadas pois suas tabelas não condiziam com as tabelas do banco de dados. Nesta etapa também fora exemplificado o processo de desenvolvimento das três principais funcionalidades do módulo 3: editar informações, listar dados em uma tabela e listar dados em duas tabelas ou mais, usando como exemplo três das dez páginas existentes no módulo.

Portanto, considerando as etapas do desenvolvimento de um *software*, levando em conta a estrutura levantamento de requisitos/criação de diagramas/desenvolvimento de protótipos/desenvolvimento do sistema, é evidente que o objetivo deste trabalho foi cumprido.

3.3 Pontos positivos e negativos

Escrevendo este trabalho, foi gratificante poder perceber a evolução do módulo de acompanhamento dos familiares, a dedicação e esforço que todos os integrantes do módulo apostaram para que essa parte do sistema se edificasse.

Acredito que este trabalho também servirá de *feedback* do processo de desenvolvimento do módulo 3, pois grande parte dele está muito bem detalhado.

Como ponto negativo gostaria de destacar o fato de que não foi possível detalhar ainda mais tal processo, explicando como fora desenvolvido todas as funcionalidades do módulo, tela por tela, pois o trabalho ficaria muito extenso.

3.4 Recomendações de trabalhos futuros

Uma boa proposta de projeto seria a de criação de um sistema voltado para o empreendedorismo regional, onde os usuários pudessem alimentá-lo com informações estratégicas sobre a região. Acredito que seria bem acolhido pela Associação Comercial e Empresarial (ACE) e pelo SEBRAE.

Referências Bibliográficas

[1] IBGE. **IBGE | Brasil em Síntese | São Paulo | Panorama**. 201X. Disponível em: <<https://cidades.ibge.gov.br/brasil/sp/sao-joao-da-boa-vista/panorama>>. Acesso em 16 de agosto de 2019.

[2] Prefeitura de São João da Boa Vista. **Portal da Prefeitura de São João da Boa Vista | São João em Notícia | São João é também a melhor cidade do país para idosos**. 2017. Disponível em: <http://saojoao.sp.gov.br/home/ler_noticia.php?id=2312>. Acesso em 23 de agosto de 2019.

[3] Ministério da Educação | **Instituto Federal de São Paulo | Sobre o Campus**. 20XX. Disponível em: <<https://www.sbv.ifsp.edu.br/sobre-campus>>. Acesso em 06 de setembro de 2019.

[4] Ministério da Educação | **Instituto Federal de São Paulo | Cursos | Técnicos | Técnico Integrado em Informática**. 20XX. Disponível em: <<https://www.sbv.ifsp.edu.br/index.php/component/content/article/64-ensino/cursos/168-tecnico-integrado-informatica>>. Acesso em 06 de setembro de 2019.

[5] SEBESTA, Robert W. **Conceitos de Linguagem de Programação**. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=J3RZDwAAQBAJ&oi=fnd&pg=PR1&dq=linguagens+de+programa%C3%A7%C3%A3o+hist%C3%B3ria&ots=IK3yi_kRaf&sig=i9ByuTw7MaQmTKziKdKp-LgblAs#v=onepage&q&f=false>. Acesso em 09 de setembro de 2019.

[6] SINTES, Antony. **Aprenda Programação Orientada a Objetos em 21 dias**. 1ª Edição. São Paulo: Perason Education do Brasil, 2010.

[7] GILMORE, Jason W. **Dominando PHP e MySQL**. 1ª Edição. Rio de Janeiro: Editora Alta Books, 2011.