**UNIVERSIDADE PAULISTA - UNIP**

GABRIEL SANABRIA FERRERI – N929JJ-6

PEDRO MIURA SUGAYAMA - D058EC9

LEONARDO CARPEJANI PRESTES – D1016H3

ERICK EIJI KANAYAMA NAGANO – N928DB0

**Aplicativo de gerência e gestão de relacionamentos humanos em ambiente Web**

SÃO PAULO – SP

2019

Índice

[1. Objetivo 3](#_Toc25188831)

[2. Introdução 4](#_Toc25188832)

[3. Fundamentos das Tecnologias e Plano de Desenvolvimento 6](#_Toc25188833)

[3.1. MVC (Model View Controller) 6](#_Toc25188834)

[3.2. API 7](#_Toc25188835)

[3.3. Base de Dados 8](#_Toc25188836)

[3.4. Divisão em 3 Camadas 10](#_Toc25188837)

[3.5. Criptografia 11](#_Toc25188838)

[4. Desenvolvimento do Projeto 13](#_Toc25188842)

[5. Funcionamento em computador 16](#_Toc25188843)

[6. Linhas de Código do fluxo 22](#_Toc25188844)

[7. Referências Bibliográficas 31](#_Toc25188845)

# Objetivo

Este trabalho tem como objetivo demonstrar o funcionamento de um sistema distribuído utilizando métodos de envio de e-mail e APIs para consulta de endereços dado um determinado CEP. Em específico, API dos Correios.

# Introdução

Com o crescimento do uso de novas tecnologias e a necessidade de portabilidade em crescente vertente dentro do mercado, a utilização de ambientes web tornou-se um recurso impreterível.

Com a portabilidade do ambiente web, é possível criar uma aplicação responsiva, que, com a mesma capacidade que é inserido em sistemas legados, também pode ser transportado para ambiente móvel (Mobile), conhecidos como PWA (Progressive Web App). O conceito de PWA foi recentemente inserido no mercado, segundo Souza, “é um termo usado para denotar aplicativos da web que usam as últimas tecnologias da web. Os aplicativos da web progressivos são páginas web (ou sites) tecnicamente regulares, mas podem aparecer ao usuário como aplicativos tradicionais ou aplicativos móveis (nativos).” (SOUZA, 2017)

Em conjunto com o desenvolvimento web, é necessária a utilização de serviços, para que seja então criado um sistema sem acoplamento desnecessário, facilitando a manutenção e desenvolvimento, uma vez que esse serviço manterá apenas uma funcionalidade do sistema conforme desejado.

Além dos fatores supracitados, para que fosse mantido a interoperabilidade e manutenibilidade dos sistemas em nível legado, é necessária e requisitada a implementação de serviços que utilizem APIs para que o desenvolvimento não seja tão árduo. É também necessário a implementação do conceito de separação de responsabilidades do sistema. Para a implementação desse conceito, é necessária a divisão do aplicativo utilizando conceitos como SOLID.

“Quando estamos aprendendo programação orientada a objetos, sem sabermos, damos a uma classe mais de uma responsabilidade e acabamos criando classes que fazem de tudo — God Class\*. Num primeiro momento isso pode parecer eficiente, mas como as responsabilidades acabam se misturando, quando há necessidade de realizar alterações nessa classe, será difícil modificar uma dessas responsabilidades sem comprometer as outras.” (ROBERTO, 2019)

Como a intenção é construir um aplicativo de sistemas distribuídos, é necessário um tratamento especial para os métodos de acesso, calculando o tempo de resposta do servidor para verificar se a conexão está instável, ou até mesmo verificar se a resposta do mesmo foi negativa, permitindo que o programa siga seu fluxo sem interrupções que derrubem o sistema de uma forma catastrófica. Para tratar esse problema, é utilizado o conceito de falha arbitrária, ou falha bizantina, para tratar todo ou qualquer erro encontrado relacionado à conexão ou à resposta do servidor, ou seja, um tratamento de erro generalizado.

# Fundamentos das Tecnologias e Plano de Desenvolvimento

# MVC (Model View Controller)

Padrão de arquitetura de software, separa as camadas de uma aplicação em diferentes níveis. Define a divisão entre três componentes: Model, View e Controller. Cada um destes componentes tem um objetivo individual e específico, estando conectados entre si. O objetivo é separar a arquitetura do software para facilitar a compreensão e a manutenção, ou seja, o objetivo em do MVC consiste em “separar as regras e lógicas do negócio da apresentação em si, permitindo um maior controle sobre a aplicação” (ZACCANINI, 2010)

A Camada em que é ocorrida a manipulação, leitura e escrita, também de validações dos dados recebidos é a Model.

É nessa camada que, por boas práticas, efetuamos a validação dos dados em propriedades do C#, além de uma predefinição de como a View deve mostrar essas informações a partir de um conceito conhecido como **Data Annotation**, que são nada mais que “classes de atributos que são usadas para definir metadados e que podemos usar para realizar a validação dos dados.” (Macoratti). O conceito de Data Annotation se estende além da model, e pode ser aplicada na camada de entidade de um projeto, como foi o caso do projeto construído e o caso mostrado abaixo

A camada de visualização do usuário, onde inserimos todas as informações de layout e demonstração do sistema, é por onde o sistema se “comunica” com o usuário, para que ele insira as informações requeridas pela página é conhecida como View.

Na camada de View, ocorrem as validações iniciais, em *Client* **¹,** se as informações foram preenchidas corretamente conforme pedido. Geralmente possui código HTML e CSS, e no Visual Studio e em projetos C#, localizam-se na pasta de Views, dentro do projeto Web. Além de HTML, podemos ter PUG, ou SASS para desenvolvimento do Front-end.

A camada de controle, responsável por receber todas as requisições do usuário é a Controller. Na camada de Controller, ocorrem chamadas às outras camadas. No caso do projeto a ser criado, essa camada ficará responsável por chamar os métodos internos para preenchimento dos dados ou inserção dos mesmos, dependendo do fluxo que o usuário se encontra. Como contexto, essa camada executaria o início da regra de negócio da aplicação, recebendo os dados diretamente da View.

Uma vez dentro da controller, e com os dados recebidos, a classe se encarrega de efetuar os métodos programados para que ocorram validações, alterações, e até mesmo novas inserções. Em C#, é a classe que contém os métodos de retorno **ActionResult**.

# API

Uma API, acrônimo para **Application Programming Interface,** é uma ferramenta utilizada para compartilhar dados e conteúdos entre softwares e aplicações. Essas APIs geralmente são usadas em uma gama de dados contextos, como dinamicamente inserindo dados dentro de uma base de dados, como selecionando os valores contidos na mesma, ou até mesmo efetuando deleção. Geralmente, para um sistema legado sofrer atualizações, é recorrida à utilização de uma API ou Middleware que traga os dados dessa API para esse sistema, já que é fácil desenvolver uma API de busca de dados por exemplo. Há 3 tipos de API, a **Privada**, **Parceiros** e **Pública**.

APIs privadas, são utilizadas apenas por empresas de forma interna

APIs de parceiros, são utilizadas para compartilhamento com companheiros e parceiros de negócio, capaz de “fornecer fluxos de receita adicionais sem comprometer a qualidade” (REDHAT)

APIs públicas, possuem sua disponibilidade generalizada, permitindo o uso de qualquer pessoa ou empresa, permitindo maior crescimento e desenvolvimento de determinado aplicativo, sendo extremamente útil para aplicações Open Source. De acordo com o tópico escrito pela empresa RedHat, a exposição de APIs a parceiros ou publico pode fazer com que ocorra uma expansão da marca de qualquer empresa, além de “Facilitar a inovação aberta ou aumentar a eficiência por meio da colaboração e de desenvolvimento externos.” (REDHAT).

Em resumo, uma API nada mais é que uma interface, que ligará o software à uma base de dados apartada na rede web. O diagrama acima demonstra o fluxo, o sistema desenvolvido executa uma requisição HTTP a um servidor que contém a API, esse servidor envia a requisição à base de dados, efetua a consulta, e a base retorna o resultado da consulta, a consulta retorna ao servidor API, que por sua vez, retorna um JSON à aplicação, e então o sistema pode trabalhar com os dados devolvidos.

A API, trabalhando principalmente por HTTPRequest possui 4 métodos de suma importância:

* HttpGet: Tem a função de ir ao servidor e executar uma função de select, sem que ocorra alteração alguma de dados, caso contrário, será barrado
* HttpPost: Tem a função de enviar ao servidor informações de um objeto ou variáveis primitivas, e então, grava na base de dados
* HttpDelete: Tem a função de executar a deleção de determinado dado a partir de algum campo único ou selecionado
* HttpPut: Tem a função de executar um update na base, efetuando alguma alteração em algum dado já existente

# Base de Dados

Banco ou base de dados, consiste em um conjunto de dados organizados e armazenados em um computador, de forma que um sistema tenha acesso às informações ali presentes por um SGBD (Sistema Gerenciador de Banco de Dados).

“[...]é uma coleção de programas que permitem aos usuários criarem e manipularem uma base de dados. Um SGBD é, assim, um sistema de software de propósito geral que facilita o processo de definir, construir e manipular bases de dados de diversas aplicações.”(TAKAI, ITALIANO, FERREIRA, 2005)

Os SGBDs(Sistema Gerenciador de Banco de Dados) mais utilizados seriam a versão relacional e não relacional, conhecida como NoSQL, “Estes últimos, cada vez mais vem conquistando espaço nas organizações” (SCUDERO,2016). Isso se deve graças à manutenção, dado o fato de que bases de dados relacionais tendem a crescer de uma forma não muito controlada, necessitando de máquinas um pouco mais robustas com o passar do tempo para conter tamanha base. Por outro lado, é deixado claro por Scudero que: “os SGBDs não relacionais, não exigem máquinas muito poderosas e, por isso, possuem uma manutenção facilitada, exigindo menos mão de obra.” (SCUDERO,2016).

Linguagens de programação como Node.JS, por exemplo, exibem uma afinidade com base de dados não relacional graças à sua performance com as buscas. Por isso, MongoDB vem se tornando cada vez mais popular no mercado em junção com Node.JS. O Banco de dados utilizado pelo sistema é a base SQL, que possuem sua própria linguagem. O diagrama abaixo aponta o esquema de base de dados utilizada, com suas respectivas tabelas e relacionamentos.

Foi utilizado relacionamento simples entre as tabelas. Esse relacionamento é feito com uma tabela principal, que contém referência à uma tabela secundária. A chave primária (PK) das tabelas tem como função a indicação dos dados naquela posição da tabela, a referência da segunda tabela, adotada à primeira, é conhecida como Foreign Key (FK).

Quando se trabalha com base de dados, para que o sistema possa pegar as informações, é feito um Modelo Relacionamento-Entidade entre o sistema e a base, no qual é criada uma camada de armazenamento desses dados, conhecida como Repositório.

Quanto maior é o sistema, ou mais antigo, essa responsabilidade de acesso não é passada para o sistema, e sim para uma API, que, como supracitado, efetua a conexão com o banco de dados, seja relacional ou não, e retorna seus valores para o programa, e então, esses dados passam a ser trabalhados, com validações ou regras de negócio pertinentes àquele modelo.

# Divisão em 3 Camadas

Para o processamento dos dados dentro do sistema, todo o projeto foi dividido em 3 camadas, Entidade, Camada de negócios, e camada de acesso a dados. Com o objetivo de separação das regras de negócio e de acesso a dados do sistema Web.

Entidade: têm como intuito realizar a representação de uma base ou de alguma entidade para o sistema efetuar o trabalho dos dados encima do mesmo. Dentro dessa camada, é *sine qua non* que ocorram determinadas validações relacionadas à regra de negócio.

Camada de negócios: camada que realiza as regras de negócio encima de determinada entidade. A camada de negócio possui total acesso à camada de acesso a dados, porém, não possui acesso algum às entidades, apenas efetua sua validação conforme o desejo do desenvolvedor. Por conter as regras de negócio do sistema, torna-se uma camada extremamente robusta, a qual é necessária aplicação de boas práticas de programação, conhecidas como conceitos de SOLID, para maior estabilidade do sistema e mais facilidade de implementação de novas regras de negócio ou novos componentes necessários ao sistema.

Camada de acesso a dados: camada essencial para acesso à dados contidos em base de dados, ou de qualquer outra fonte, para inserção dos mesmos no sistema, para que ocorra o trabalho dos dados nas camadas superiores. Nessa camada ficam localizadas as queries(comandos) do banco de dados, que podem ser simples comandos em string ou procedures complexas, que retornariam um objeto completo, que foi colocado em uma entidade. Para facilitar a codificação, foi utilizado a biblioteca Dapper, instalado via Nuget Package Manager.

# Criptografia



A criptografia se aprimorou com o decorrer do tempo, desde os egípcios até hoje. Em seus conceitos gerais, a criptografia é utilizada principalmente na segurança de informações que antes não estavam tão protegidas, e com o simples intuito de mascarar aquilo que a mensagem realmente significa.

Com a evolução dos meios de comunicação, a criptografia foi instaurada em grande parte desses meios, um exemplo disso é a aplicação dos recursos Web ou nos recursos empresariais que desejam ser escondidos.

**CINCO PRINCIPAIS FUNÇÕES**

• Privacidade: Que ninguém consiga ler a mensagem além da pessoa ou público alvo

• Autenticação: Processo de comprovação da identidade de outrem

• Integridade: Garante que o receptor da mensagem tenha recebido a mensagem íntegra, sem alterações.

• Não-Repúdio: Dado mecanismo que prova que quem realmente escreveu a mensagem foi o desejado autor.

• Troca de chaves: A maneira de como as chaves criptográficas são trocadas entre o mandante e o receptor.

**TIPOS DE ALGORITIMOS CRIPTOGRÁFICOS**

**CHAVE SIMÉTRICA**

É a criptografia de forma simples, já que tanto o mandante como o receptor possuem as chaves para terem acesso à informação, possui chaves que vão de 8 (mais fraca) a até 1024 bits (mais forte) e foi desenvolvido por Ron Rivest. (Exemplos: DES, IDEA, RC)

**CHAVE ASSIMÉTRICA**

É a chave criptográfica um pouco mais complexa, uma vez que são utilizadas duas chaves para o acesso à mensagem, a chave pública e a chave privada. Com a chave pública, é feita a codificação da mensagem desejada, e com a chave privada, sua decriptografia. Tanto a chave Simétrica como assimétrica são processos reversíveis de criptografia da mensagem. (Exemplos: RSA, ElGamal)

**RECURSO HASH**

A criptografia Hash, também conhecida como “one-way” é um tipo de recurso que usufrui do tipo de dado String, formado geralmente por 16 bits, 20 bits e até mesmo 512 bits. Possuí características únicas que, de acordo com Thiago Castelló: “o valor de entrada da string é indeterminado, porém a saída já tem um valor fixo, é uma função One-Way e é livre de colisão” (CASTELLÓ, 2016).

A chave Hash, que quer dizer “picar” ou “cortar” tem como objetivo computar um resumo de mensagem criando uma assinatura digital. Geralmente é utilizada com a chave assimétrica para manter a integridade de um arquivo digital. Deve ter apenas o caminho de ida, jamais o de volta, já que é One-Way, além de ser fácil e prático sua produção.

Uma vez que é necessária a segurança dos usuários, torna-se difícil a capacidade de obtenção de um item que está criptografado, com esse intuito, o sistema irá dispor de uma criptografia MD5 simples, que irá efetuar a geração de uma chave Hash para a senha do usuário.

# Desenvolvimento do Projeto

Tendo construído a base, foi necessária a comparação da base com os tipos de dados do C#, baseando-se no modelo Relacionamento-Entidade, foi construída uma classe de entidade para cada tabela dentro do sistema, fazendo com que o banco de dados então ficasse “mapeado” pelo sistema. Assim que mapeado, foi implementado o conceito de Data Annotation para futuras validações das entidades que necessitariam de preenchimento, evitando futuras exceções do sistema.

Assim como as entidades da base possuem um tipo, para que ocorra o mapeamento perfeito, é necessário que a entidade do projeto possua os mesmos tipos variáveis da base, caso contrário, não será possível armazenar o valor, e ocorrerá uma exceção.

A construção da camada de acesso a dados ocorreu com a utilização de uma herança de classe, com o intuito de economizar linhas de código e aproveitar a herança da orientação a objeto. A classe pai, possui o método de busca da string de conexão com a base de dados, permitindo que todas as filhas possam acessar a base sem problema algum.

As classes filhas, foram construídas com a utilização de um framework conhecido como Dapper¹, o mesmo executa a query a partir do C# com os parâmetros passados nos métodos, fazendo com que fique extremamente robusto e fácil a passagem do objeto para a base, e seu retorno da mesma forma. A documentação do framework Dapper afirma que “Dapper é um micro ORM ou um mapeador de objetos simples que ajuda a mapear as queries nativas do objeto C#[...]” (DESCONHECIDO, 2017, tradução minha). Os parâmetros passados no método são passados para a query string a partir de um @, durante a execução, Dapper transforma os parâmetros da string em objetos, e os envia à base para execução da query.

Após a construção da camada de acesso a dados, foi efetuada a construção da camada de negócios da aplicação, com o intuito de que ocorressem as validações necessárias conforme exigidas pelas regras de negócio do sistema, essa camada faz com que a regra seja cumprida e devolvida às camadas superiores para exposição ao usuário ou talvez para execução de outro método da camada de negócios.

Como regra básica da divisão em 3 camadas, a regra de negócio tem acesso apenas à conexão com a base de dados, jamais à entidade diretamente, uma vez que essa viria nula, assim como ele também não possui acesso às camadas superiores, apenas envia suas informações.

No conceito mais bruto, poderia ser tido como o “Middleware” de um sistema, já que trata os dados, efetua as validações, e as envia para outra camada para que o fluxo não pare.

Na continuidade da construção, após efetuar toda a regra de negócio da aplicação contida na Business, foi criada então a camada Web, que seria a conexão com o usuário propriamente dita.

A camada web iria ser construída de acordo com os padrões MVC. Após a construção da aplicação web, foi então desenvolvida a implementação daquilo que já havia sido criado (camadas inferiores supracitadas) com a camada superior de acesso do usuário, fechando assim o ciclo de desenvolvimento do aplicativo, deixando de ser back-end, para mistura com front-end ao utilizar HTML e CSS. Para que as informações fossem passadas do back-end para o front-end, foi utilizado o conceito de databinding das Models, dentro das Views, utilizando o Razor (ferramenta nativa do Visual Studio). Todas as validações utilizando Data Annotation foram então passadas para as entidades, dado seu desacoplamento do sistema, e as models apenas referenciavam um conjunto dessas classes já validadas.

O fluxo web será como o usuário manterá seu fluxo dentro do sistema, a partir da conexão com a internet, é possível acessar a o aplicativo publicado em um determinado servidor web, uma vez que o backend e o front serão mantidos na mesma localidade. O Fluxo web do sistema consiste em:

* Login do usuário ao sistema
* Desligamento de um funcionário de sua área
* Avaliação de um funcionário de sua área
* Obtenção de relatórios atuais, mensais e anuais
* Edição de usuário
* Envio de notificação por mensagem ao ser desligado
* Implementações futuras como chat para discussão de avaliação ou até mesmo resposta ao e-mail recebido do sistema

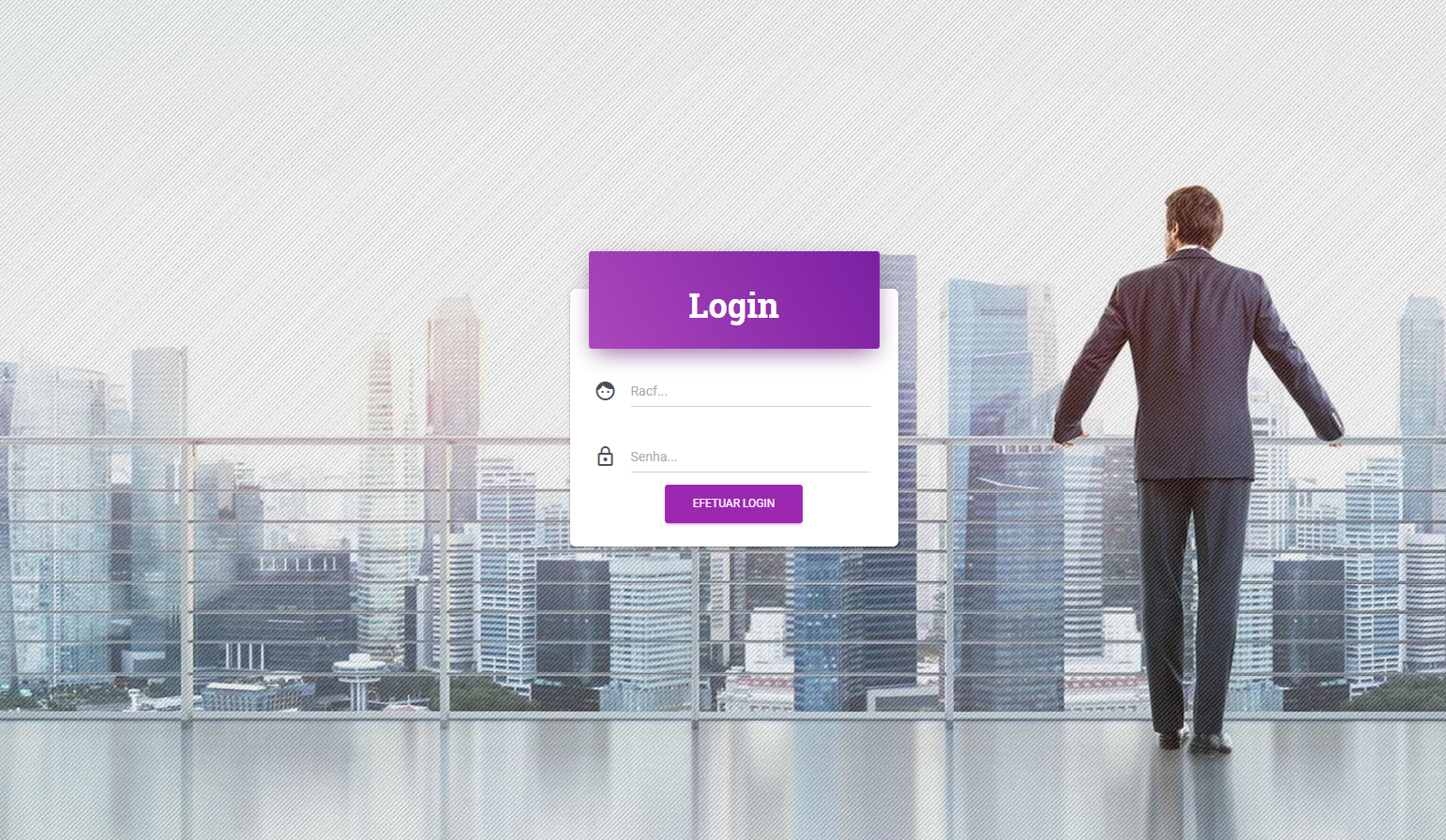
Assim que ocorrer a abertura do sistema no ambiente web, será obtido a tela de login diretamente, e é então pedido um usuário e senha para acesso ao sistema

Assim que inserir seu login e senha, caso correto, o usuário então ganhará acesso ao sistema, entrando na home

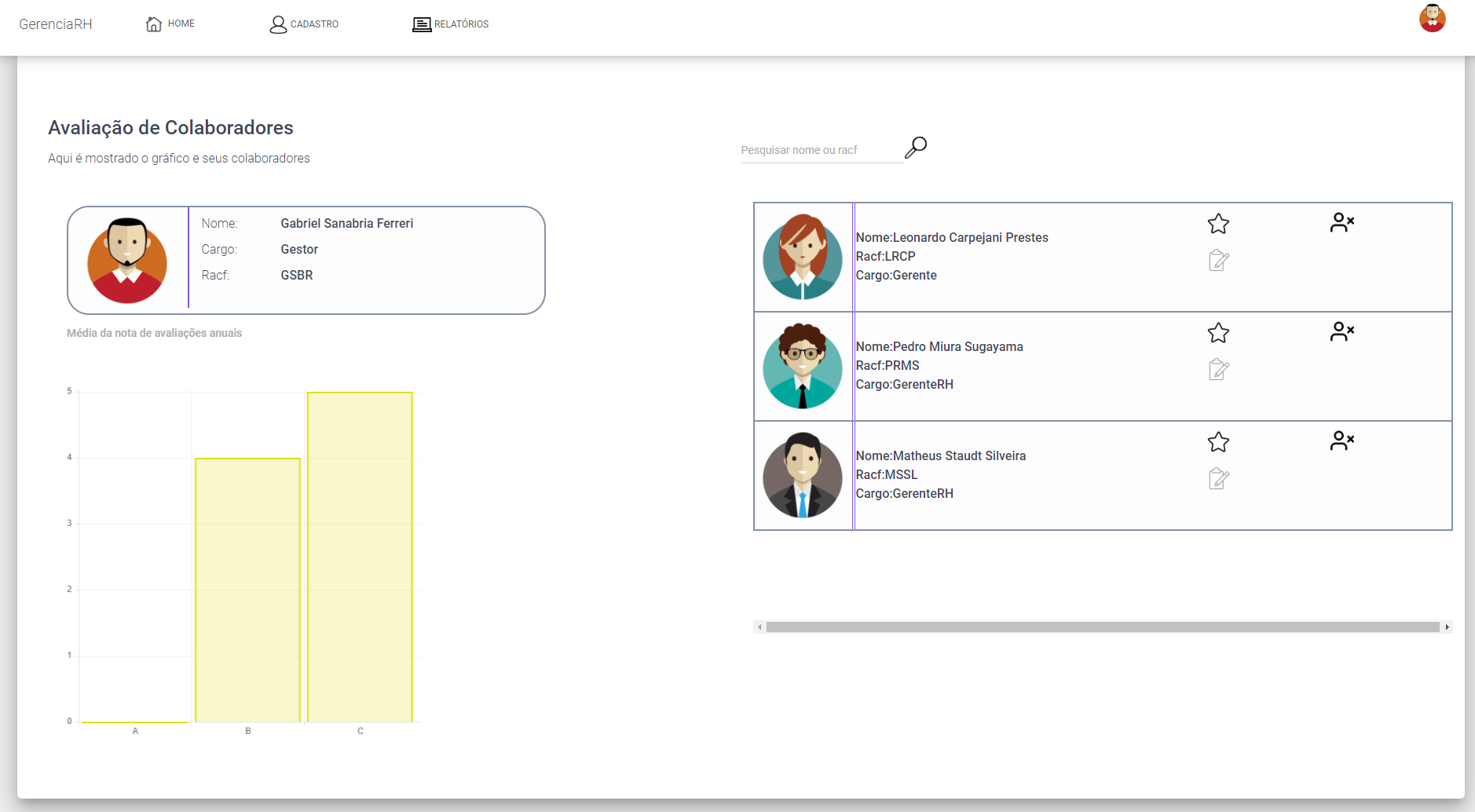
Na página de home, o usuário terá então uma visão geral de seu time, tendo a capacidade de efetuar uma as seguintes ações:

* Efetuar Avaliação
* Efetuar Desligamento
* Efetuar Edição do usuário subordinado
* Efetuar um cadastro no sistema
* Gerar relatórios
* Verificar gráfico com quantidade de médias das avaliações atuais geradas

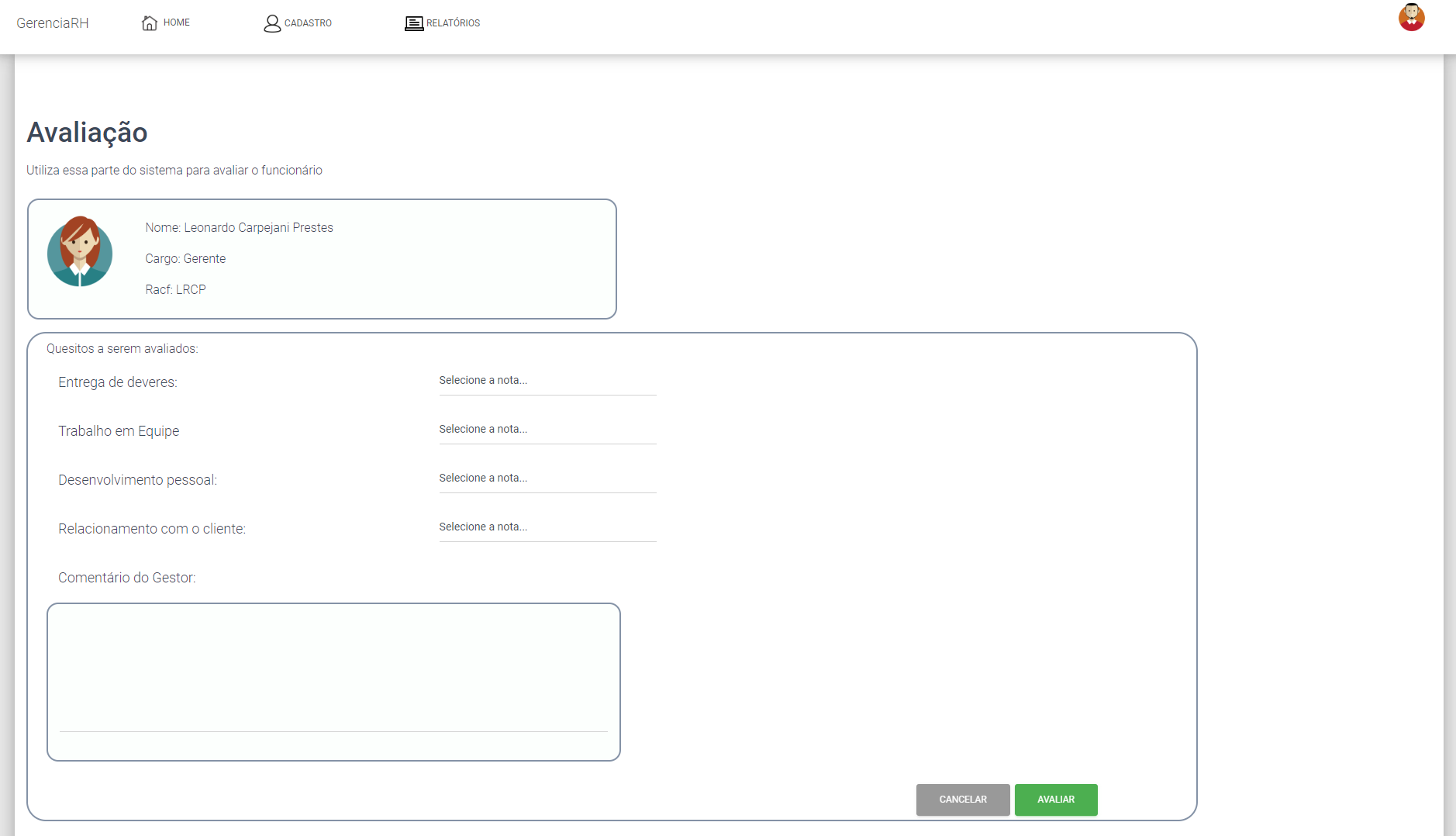
# Funcionamento em computador



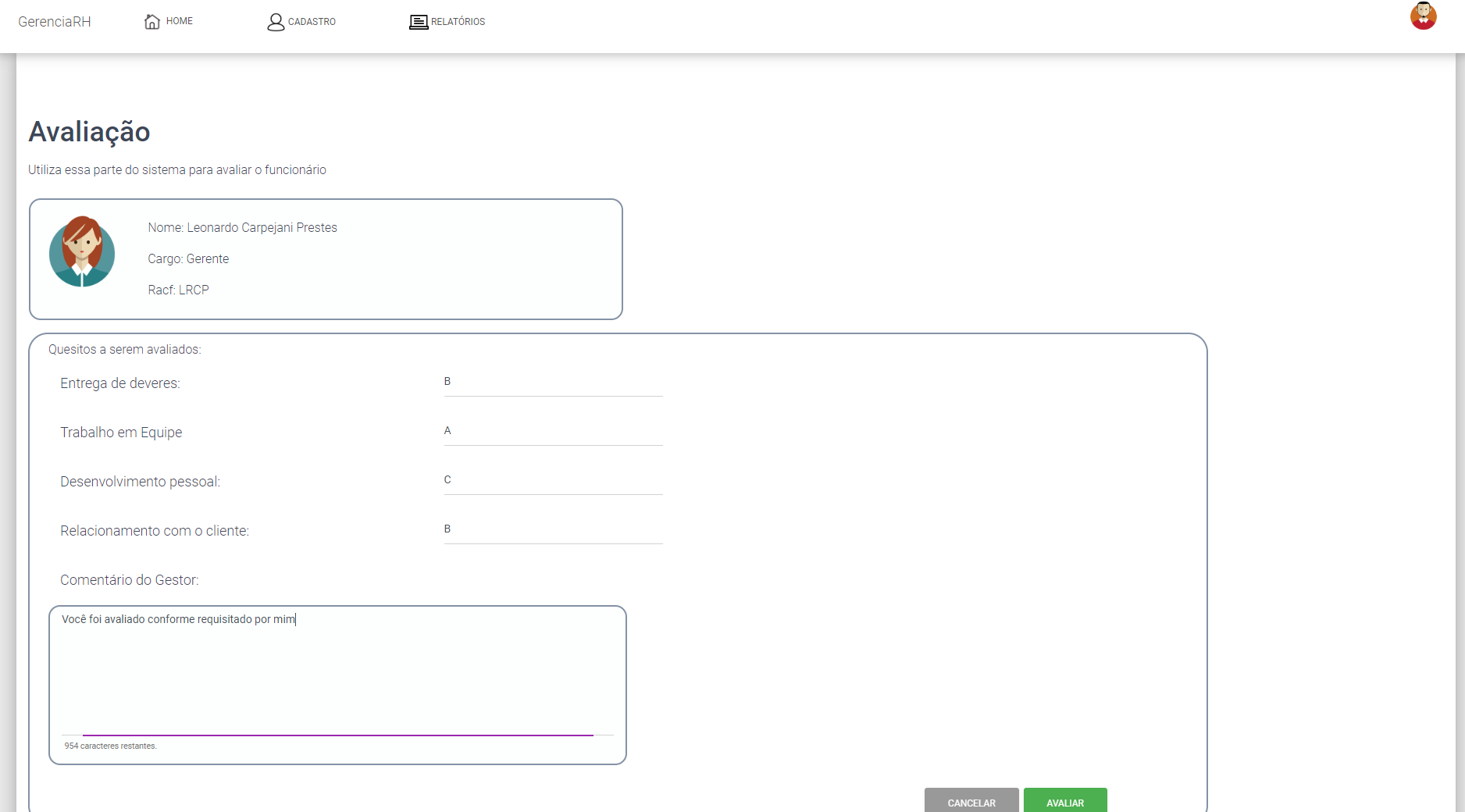
- Tela de Login Inicial



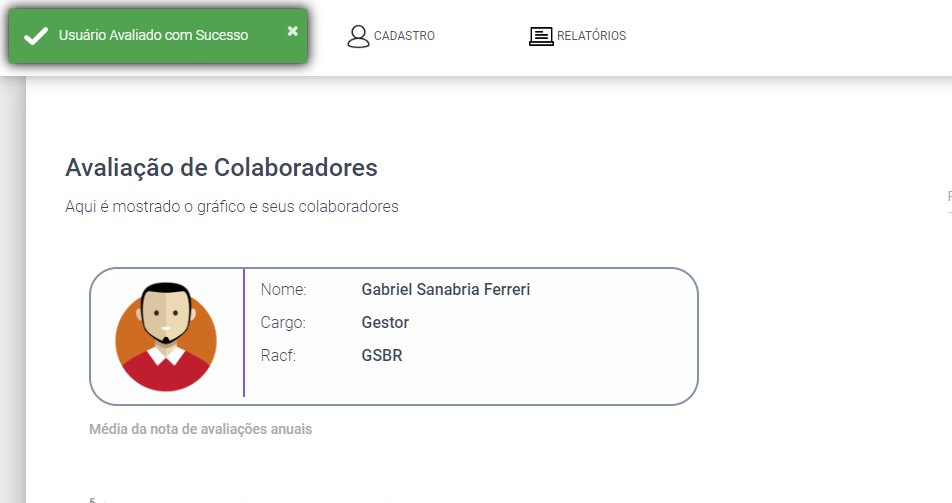
Assim que inserir seu login e senha, caso correto, o usuário então ganhará acesso ao sistema, entrando na home



Ao clicar em avaliação, o usuário poderá efetuar a avaliação de seu colaborador escolhido.

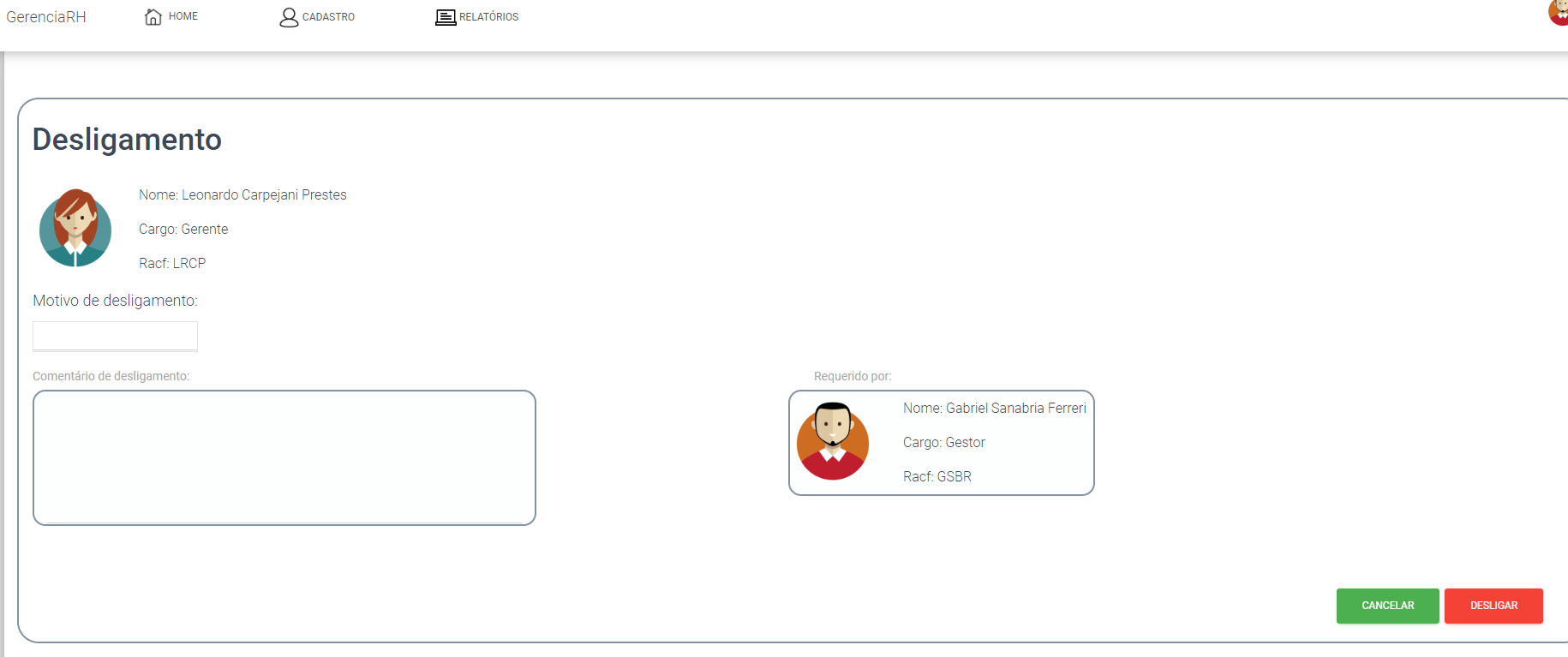


Assim que efetuar a avaliação, é enviado novamente à home, porém com um aviso e seu gráfico com valores atualizados e terminando assim o fluxo de avaliação de seu funcionário.

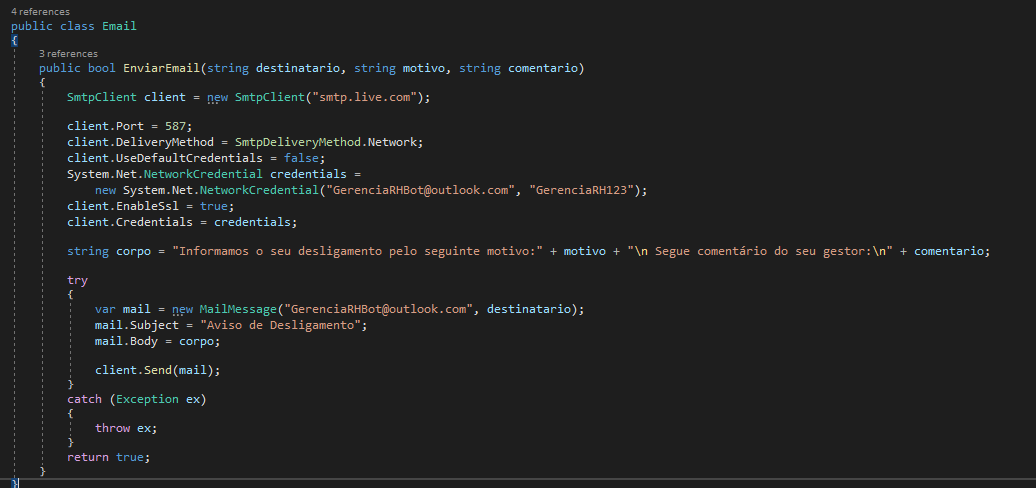


- Fim do fluxo de avaliação do funcionário

Assim que o usuário efetuar o clique no ícone do desligamento, será disponibilizada para ele uma visualização da tela, com o funcionário que assim foi clicado para desligamento

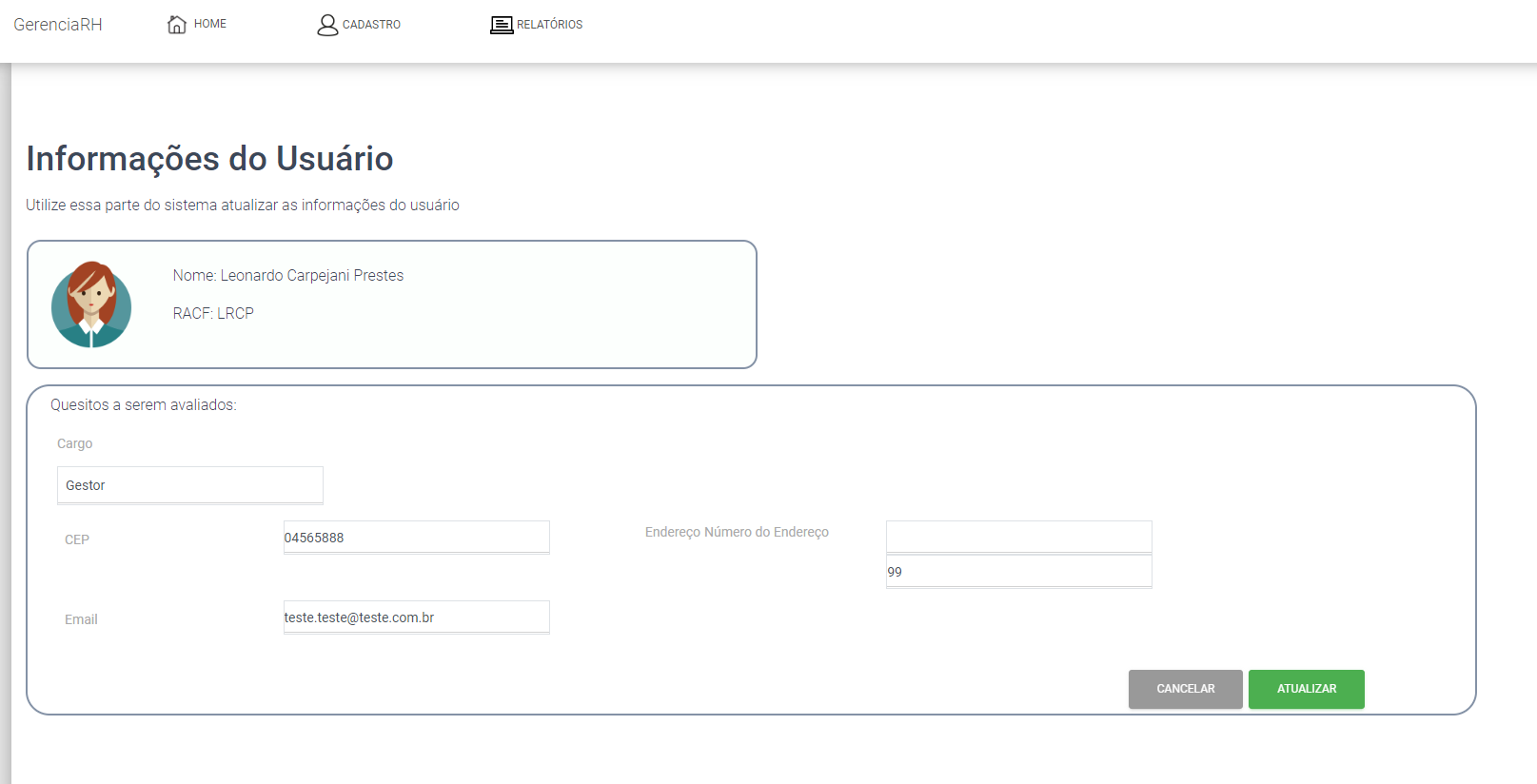


Assim que o usuário efetuar o clique em Desligar, o mesmo não aparecerá mais na sua tela de home, e o seu colaborador terá recebido um e-mail formalmente comunicando o desligamento e comparecimento ao RH para esclarecer assuntos pendentes ou o como ocorreu o desligamento.



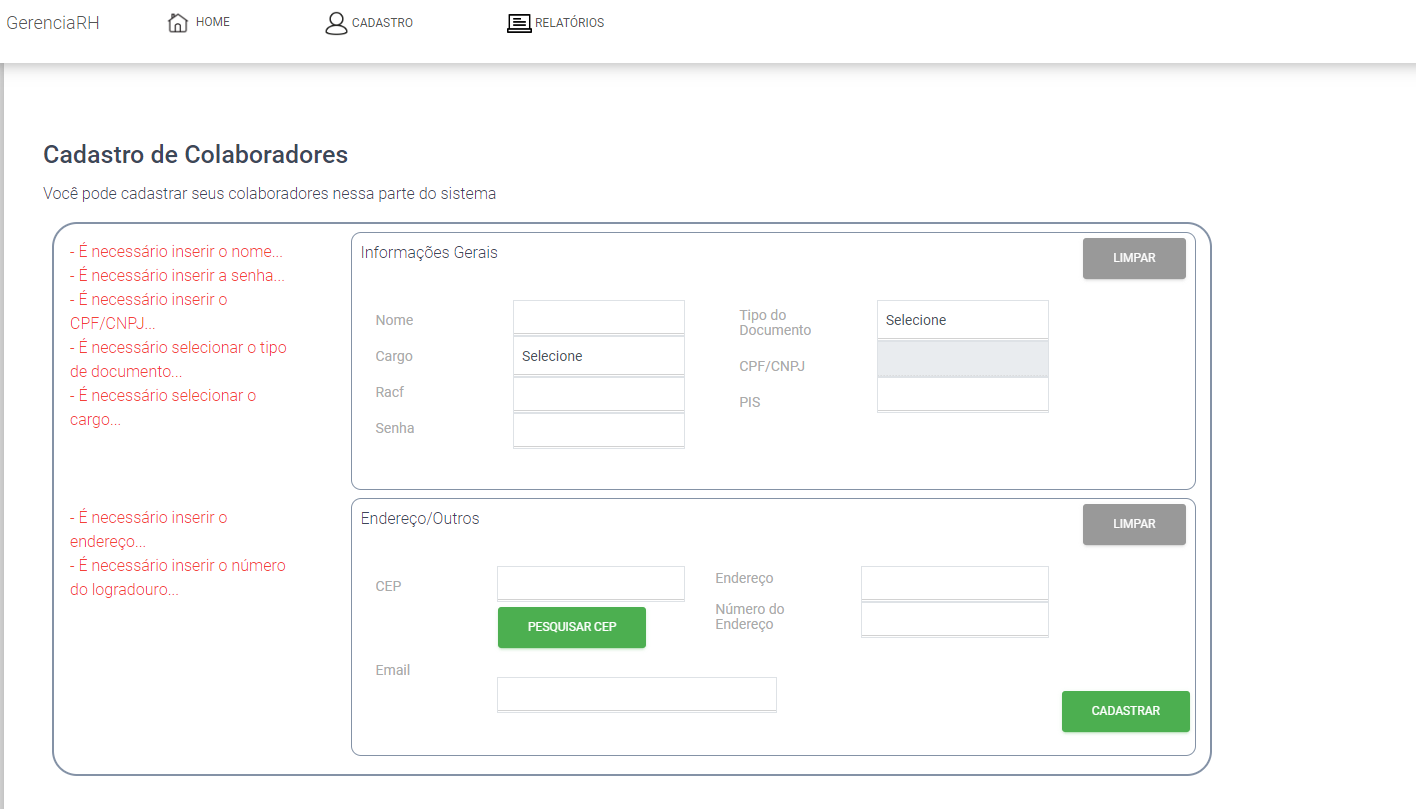
- Código do envio de e-mail

Assim que o usuário na home, efetuar o clique no ícone de edição, o mesmo será movido para uma pagina web de edição dos dados daquele usuário dentro do sistema

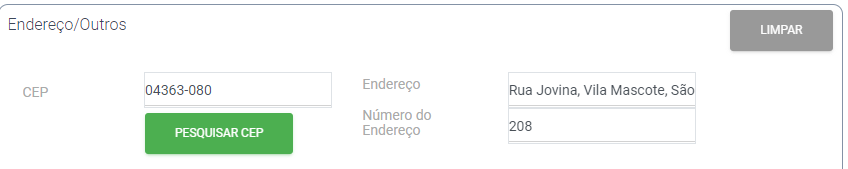


Assim que clicar em atualizar, retornará à home com um aviso de que seu colaborador foi atualizado com sucesso.

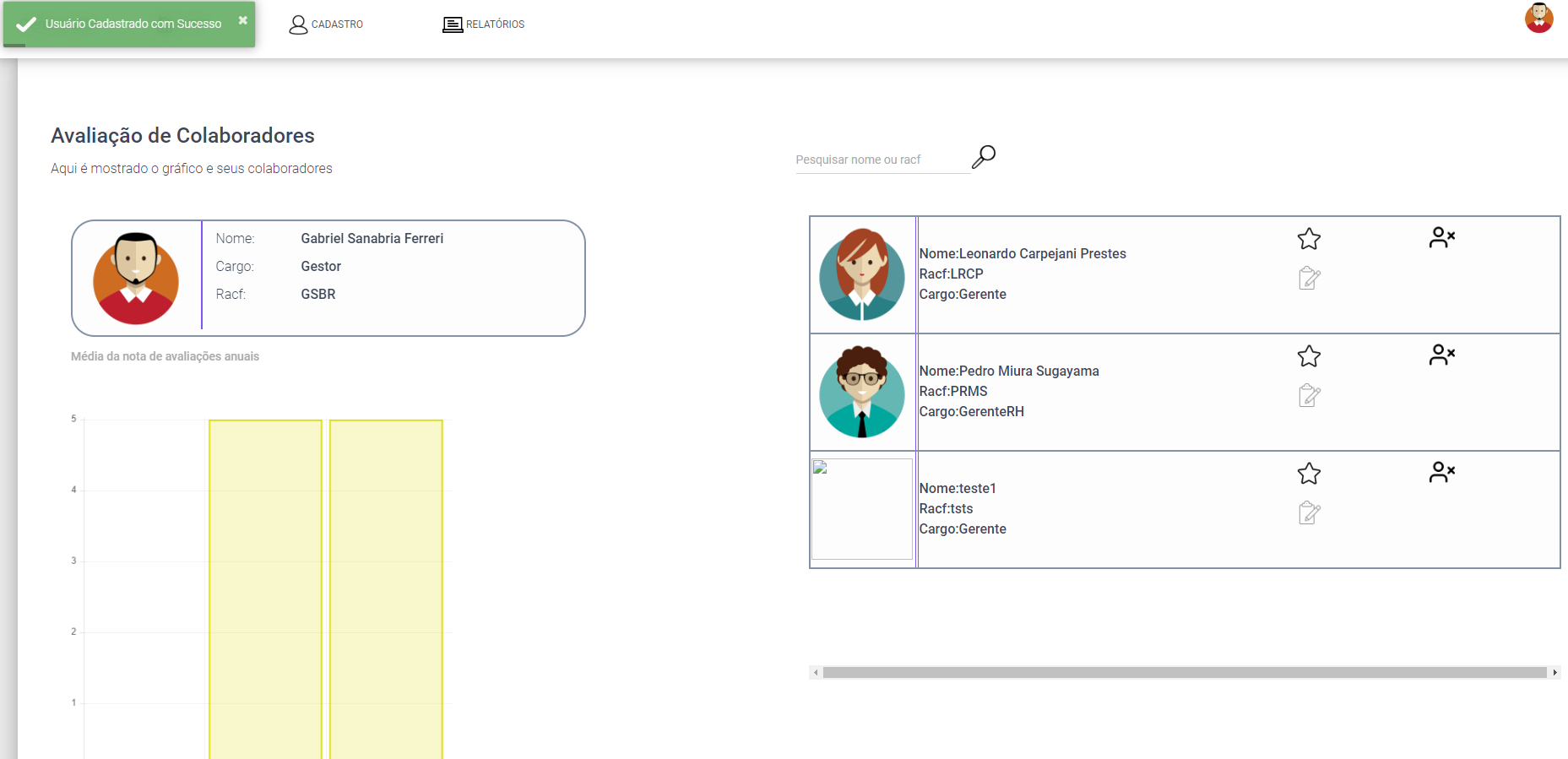
Ao clicar na aba de cadastro, na sua NavBar, o usuário será realocado para a View de Cadastro de um novo usuário no seu sistema, por consequência, para sua área em específico, no qual, ocorrerá validação dos campos e então, caso preenchido, será devidamente inserido no sistema para acesso posterior



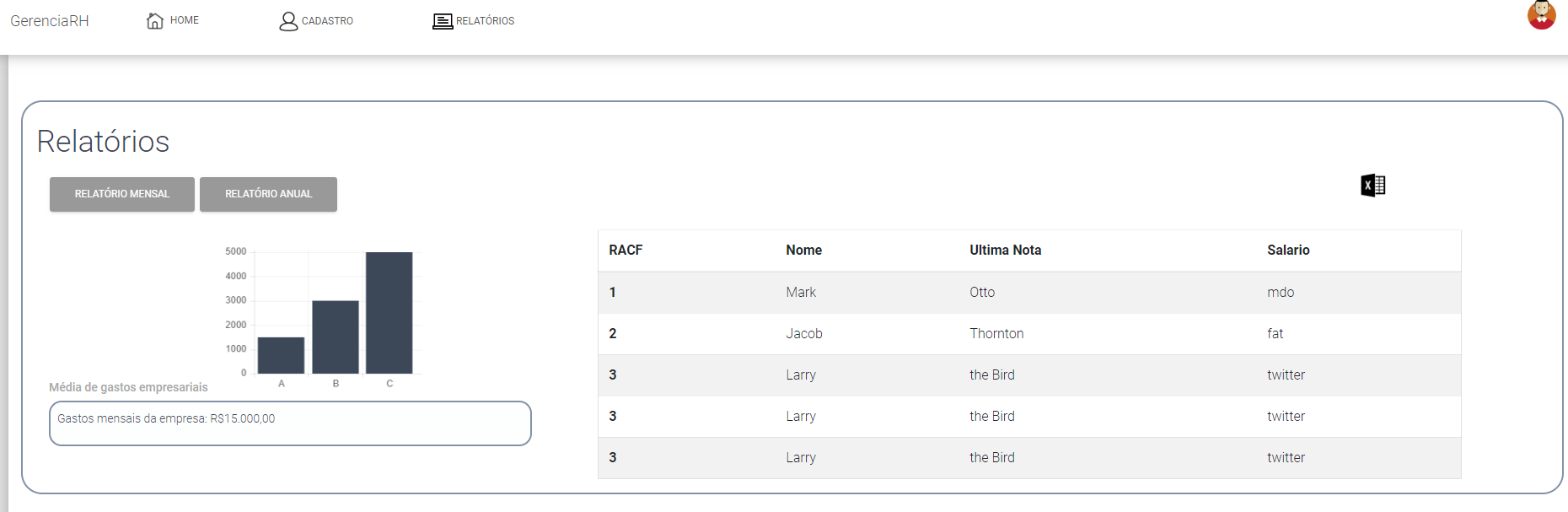
A tela possui também uma implementação única de um componente que acessa à uma API cedida pelos correios, para busca de endereço via CEP. Ao inserir os valores no campo CEP e pressionar o botão Pesquisar CEP, o endereço é automaticamente preenchido



Após cadastro, usuário retorna para home com seu novo colaborador em sua lista e uma mensagem de sucesso



Ao clicar em relatório, o usuário irá visualizar uma tela com relatórios, aonde será possível visualizar o gasto mensal de todos os funcionários contidos sob sua gestão. O gráfico funcionará igual ao da Home, atualizado dinamicamente conforme os valores contidos em base de dados



# Linhas de Código do fluxo

public class AtualizarUsuarioController : Controller

    {

        UsuarioBusiness \_usrBusiness = new UsuarioBusiness();

        // GET: Desligamento

        public ActionResult AtualizarIndex(string racfUsuario)

        {

            return Json(new { redirect = "/AtualizarUsuario/Index?racfUsuario=" + racfUsuario + "" }, JsonRequestBehavior.AllowGet);

        }

        public ActionResult Index(string racfUsuario)

        {

            Usuario usuario = \_usrBusiness.BuscarUsuario(racfUsuario);

            UsuarioModel model = new UsuarioModel();

            model.Usuario = usuario;

            var tipoCargo = (ETipoCargo)usuario.IdCargo;

            Cargo cargo = new Cargo() { TipoCargo = tipoCargo };

            model.Cargo = cargo;

            return View(model);

        }

        public ActionResult UpdateUsuario(UsuarioModel model)

        {

            Usuario usuario = new Usuario();

            var idCargo = (int)model.Cargo.TipoCargo;

            usuario = \_usrBusiness.BuscarUsuario(model.Usuario.Racf);

            usuario.Email = model.Usuario.Email;

            usuario.CEP = model.Usuario.CEP;

            usuario.Endereco = model.Usuario.Endereco;

            usuario.NumeroEndereco = model.Usuario.NumeroEndereco;

            usuario.IdCargo = idCargo;

            bool sucesso = \_usrBusiness.UpdateUsuario(usuario);

            if (sucesso == true)

            {

                TempData["Sucesso"] = "U";

                return RedirectToAction("Index", "Home");

            }

            else

            {

                string mensagem = Resources.Mensagens.mensagemErroAvaliacao;

                TempData["Erro"] = mensagem;

                return RedirectToAction("Index", "Home");

            }

            return View();

        }

    }

public class AvaliacaoController : Controller

    {

        UsuarioBusiness \_usrBusiness = new UsuarioBusiness();

        AvaliacaoBusiness avaliacaoBusiness = new AvaliacaoBusiness();

        string mensagem = "";

        // GET: Avaliacao

        public ActionResult AvaliacaoIndex(string racfUsuario)

        {

            return Json(new { redirect = "/Avaliacao/Index?racfUsuario=" + racfUsuario +"" }, JsonRequestBehavior.AllowGet);

        }

        public ActionResult Index(string racfUsuario)

        {

            try

            {

                Usuario usuarioASerAvaliado = \_usrBusiness.BuscarUsuario(racfUsuario);

                UsuarioModel model = new UsuarioModel

                {

                    UsuarioAvaliacao = usuarioASerAvaliado

                };

                return View(model);

            }

            catch (Exception)

            {

                mensagem = Resources.Mensagens.mensagemErroAcessoAvaliacao;

                ViewData["Erro"] = mensagem;

                return RedirectToAction("Index", "Home");

            }

        }

        [HttpPost]

        public ActionResult EfetuarAvaliacao(UsuarioModel model, UsuarioLogado usuarioLogado,Usuario usuario)

        {

            try

            {

                model.Avaliacao.UsuarioAvaliador = usuarioLogado.usuarioLogado.IdUsuario;

                model.Avaliacao.UsuarioAvaliado = model.UsuarioAvaliacao.IdUsuario;

                bool avaliado = avaliacaoBusiness.InserirAvaliacao(model.Avaliacao);

                if (avaliado == true)

                {

                    TempData["Sucesso"] = "A";

                    return RedirectToAction("Index", "Home");

                }

                else

                {

                    mensagem = Resources.Mensagens.mensagemErroAvaliacao;

                    TempData["Erro"] = mensagem;

                    return RedirectToAction("Index", "Home");

                }

            }

            catch (Exception)

            {

                mensagem = Resources.Mensagens.mensagemErroBase;

                TempData["Erro"] = mensagem;

                return RedirectToAction("Index", "Home");

            }

        }

    }

public class CadastroController : Controller

    {

        // GET: Cadastro

        public ActionResult Index()

        {

            return View();

        }

        public ActionResult EfetuarCadastro(UsuarioModel model, UsuarioLogado usuarioLogado)

        {

            UsuarioBusiness usuarioBusiness = new UsuarioBusiness();

            ValidaLogin validaLogin = new ValidaLogin();

            var idCargo = (int)model.Cargo.TipoCargo;

            model.Usuario.IdCargo = idCargo;

            validaLogin = usuarioBusiness.CadastrarUsuario(model.Usuario, usuarioLogado.usuarioLogado.IdUsuario);

            if (validaLogin.Sucesso == true)

            {

                TempData["Sucesso"] = "S";

                return RedirectToAction("Index", "Home");

            }

            else

            {

                ViewBag.Sucesso = "N";

                ViewBag.Mensagem = validaLogin.Mensagem;

                return View("Index");

            }

        }

        public JsonResult ConsultaCEP(string CEP)

        {

            CepBusiness cepBusiness = new CepBusiness();

            var dados = cepBusiness.ConsultarEndereco(CEP);

            if(dados.Logradouro == null)

            {

                return Json("error");

            }

            return Json(dados);

        }

    }

public class DesligamentoController : Controller

    {

        UsuarioBusiness \_usrBusiness = new UsuarioBusiness();

        string mensagem = "";

        // GET: Desligamento

        public ActionResult DesligamentoIndex(string racfUsuario, string racfGestor)

        {

            return Json(new { redirect = "/Desligamento/Index?racfUsuario=" + racfUsuario + "&racfGestor=" + racfGestor + "" }, JsonRequestBehavior.AllowGet);

        }

        public ActionResult Index (string racfUsuario, string racfGestor)

        {

            try

            {

                Usuario usuarioDesligado = \_usrBusiness.BuscarUsuario(racfUsuario);

                Usuario gestor = \_usrBusiness.BuscarUsuario(racfGestor);

                UsuarioModel model = new UsuarioModel

                {

                    Usuario = gestor,

                    UsuarioDesligamento = usuarioDesligado

                };

                return View(model);

            }

            catch (Exception)

            {

                mensagem = Resources.Mensagens.mensagemErroBase;

                TempData["Erro"] = mensagem;

                return RedirectToAction("Home", "Index");

            }

        }

        //TODO - SALVAR NA BASE QUE USUARIO FOI DESLIGADO, RETORNANDO AVISO DE SUCESSO PRA VIEW ANTERIOR

        public ActionResult SalvarDesligamento(UsuarioModel model,UsuarioLogado usuarioLogado)

        {

            try

            {

                model.Usuario.Racf = model.UsuarioDesligamento.Racf;

                model.Usuario.IdDesligamento = usuarioLogado.usuarioLogado.IdUsuario;

                if (\_usrBusiness.DeletarUsuario(model.Usuario) == false)

                {

                    mensagem = Resources.Mensagens.mensagemErroDesligamento;

                    TempData["Erro"] = mensagem;

                    return RedirectToAction("Index","Home");

                }

                else

                {

                    Email email = new Email();

                    email.EnviarEmail(model.UsuarioDesligamento.Email, model.Usuario.Motivo, model.ComentarioDesligamento);

                    TempData["Sucesso"] = "D";

                    return RedirectToAction("Index", "Home");

                }

            }

            catch (Exception)

            {

                mensagem = Resources.Mensagens.mensagemErroBase;

                TempData["Erro"] = mensagem;

                return RedirectToAction("Index", "Home");

            }

        }

    }

public class HomeController : Controller

    {

        public ActionResult Index(string racf)

        {

            UsuarioModel modelUsuario = new UsuarioModel();

            AvaliacaoBusiness avaliacaoBusiness = new AvaliacaoBusiness();

            UsuarioBusiness business = new UsuarioBusiness();

            List<Avaliacao> listAvaliacao = new List<Avaliacao>();

            if (string.IsNullOrEmpty(racf) && Session["usuarioLogado"] != null)

            {

                modelUsuario.Usuario = (Usuario)Session["usuarioLogado"];

            }

            else

            {

                modelUsuario.Usuario = business.BuscarUsuario(racf);

            }

            if (modelUsuario.Usuario == null)

            {

                return RedirectToAction("Index", "Login");

            }

            modelUsuario.listUsuario = business.ListarUsuarios(modelUsuario.Usuario);

            listAvaliacao = avaliacaoBusiness.ListarAvaliacoes(modelUsuario.Usuario.IdUsuario);

            modelUsuario.QuantidadeA = listAvaliacao.Where(x => x.Nota == "A").Count();

            modelUsuario.QuantidadeB = listAvaliacao.Where(x => x.Nota == "B").Count();

            modelUsuario.QuantidadeC = listAvaliacao.Where(x => x.Nota == "C").Count();

            return View(modelUsuario);

        }

        public ActionResult PesquisarUsuario(string chave)

        {

            return PartialView();

        }

    }

public class LoginController : Controller

    {

        string mensagem = "";

        // GET: Login

        public ActionResult Index()

        {

            LoginModel model = new LoginModel();

            return View(model);

        }

        public ActionResult EfetuarLogin(LoginModel model)

        {

            try

            {

                LoginBusiness Loginbusiness = new LoginBusiness();

                bool loginEfetuado = Loginbusiness.ValidarLogin(model.Usuario.Racf, model.Usuario.Senha);

                if (loginEfetuado)

                {

                    UsuarioBusiness usuarioBusiness = new UsuarioBusiness();

                    Usuario userLogado = usuarioBusiness.BuscarUsuario(model.Usuario.Racf);

                    Session["usuarioLogado"] = userLogado;

                    return RedirectToAction("Index", "Home", new { racf = model.Usuario.Racf });

                }

                else

                {

                    mensagem = Resources.Mensagens.mensagemErroLogin;

                    TempData["Erro"] = mensagem;

                    return View("Index");

                }

            }

            catch (Exception)

            {

                mensagem = Resources.Mensagens.mensagemErroBase;

                TempData["Erro"] = mensagem;

                return View("Index");

            }

        }

    }

public class RelatorioController : Controller

    {

        // GET: Relatorio

        public ActionResult Index()

        {

            UsuarioBusiness usuarioBusiness = new UsuarioBusiness();

            var usuario = (Usuario)Session["usuarioLogado"];

            List<Usuario> usuarios = new List<Usuario>();

            RelatorioModel relatorioModel = new RelatorioModel();

            relatorioModel.listUsuarios = usuarioBusiness.Relatorio();

            usuarios = usuarioBusiness.ListarUsuariosSalarios(usuario);

            relatorioModel.Gerente = usuarios.Where(x => x.IdCargo == 2).ToList().Count;

            relatorioModel.GerenteRH = usuarios.Where(x => x.IdCargo == 3).ToList().Count;

            relatorioModel.FuncionarioRH = usuarios.Where(x => x.IdCargo == 4).ToList().Count;

            relatorioModel.Funcionario = usuarios.Where(x => x.IdCargo == 5).ToList().Count;

            Salarios salarios = new Salarios();

            salarios.SalarioFuncionario = (int)usuarios.Where(x => x.IdCargo == 5).Sum(x => x.Salario);

            salarios.SalarioFuncionarioRH = (int)usuarios.Where(x => x.IdCargo == 4).Sum(x => x.Salario);

            salarios.SalarioGerente = (int)usuarios.Where(x => x.IdCargo == 2).Sum(x => x.Salario);

            salarios.SalarioGerenteRH = (int)usuarios.Where(x => x.IdCargo == 3).Sum(x => x.Salario);

            relatorioModel.Salarios = salarios;

            relatorioModel.Total = relatorioModel.Salarios.SalarioFuncionario + relatorioModel.Salarios.SalarioFuncionarioRH + relatorioModel.Salarios.SalarioGerente + relatorioModel.Salarios.SalarioGerenteRH;

            return View(relatorioModel);

        }

    }

public class Email

    {

        public bool EnviarEmail(string destinatario, string motivo, string comentario)

        {

            SmtpClient client = new SmtpClient("smtp.live.com");

            client.Port = 587;

            client.DeliveryMethod = SmtpDeliveryMethod.Network;

            client.UseDefaultCredentials = false;

            System.Net.NetworkCredential credentials =

                new System.Net.NetworkCredential("GerenciaRHBot@outlook.com", "GerenciaRH123");

            client.EnableSsl = true;

            client.Credentials = credentials;

            string corpo = "Informamos o seu desligamento pelo seguinte motivo:" + motivo + "\n Segue comentário do seu gestor:\n" + comentario;

            try

            {

                var mail = new MailMessage("GerenciaRHBot@outlook.com", destinatario);

                mail.Subject = "Aviso de Desligamento";

                mail.Body = corpo;

                client.Send(mail);

            }

            catch (Exception ex)

            {

                throw ex;

            }

            return true;

        }

    }

public class CepBusiness

    {

        public CEP ConsultarEndereco(string cep)

        {

            CEP OCep = new CEP();

            try

            {

                WebClient client = new WebClient();

                var result = client.DownloadString("http://viacep.com.br/ws/" + cep + "/json/");

                result = result.Replace("£", "").Replace("Ã", "ã");

                if (result.Contains("erro"))

                {

                    return OCep;

                }

                else

                {

                    OCep = JsonConvert.DeserializeObject<CEP>(result);

                }

            }

            catch (Exception ex)

            {

            }

            return OCep;

        }

    }

public class ImpedimentoBusiness

    {

        private readonly ImpedimentoDao Dao = new ImpedimentoDao();

        public List<Impedimento> listarImpedimentos(Impedimento impedimento)

        {

            List<Impedimento> impedimentos = Dao.ListarImpedimentos(impedimento);

            return impedimentos.Count > 0 ? impedimentos : new List<Impedimento>();

        }

        public bool InserirImpedimento (Impedimento impedimento)

        {

            bool inserido = Dao.InserirImpedimento(impedimento);

            return inserido;

        }

        public bool UpdateImpedimento (Impedimento impedimento)

        {

            bool alterado = Dao.AlterarImpedimento(impedimento);

            return alterado;

        }

    }

public class LoginBusiness

    {

        public bool ValidarLogin(string Racf, string Senha)

        {

            bool validado;

            try

            {

                LoginDao dao = new LoginDao();

                if (string.IsNullOrEmpty(Racf) || string.IsNullOrEmpty(Senha))

                {

                    validado = false;

                }

                else

                {

                    string senhaCriptografada = Criptografia.GerarChaveHashMD5(Senha);

                    Usuario usuario = dao.SelecionarUsuario(Racf, senhaCriptografada);

                    validado = ValidarUsuario(usuario);

                }

            }

            catch (Exception ex)

            {

                throw ex;

            }

            return validado;

        }

        private bool ValidarUsuario(Usuario usuario)

        {

            if (usuario == null)

            {

                return false;

            }

            else

            {

                return true;

            }

        }

    }

# Referências Bibliográficas

MELOTO, Paulo. **Sistema Web vs Sistema Local**. 21/09/2016. Disponível em: <https://sistemasoma.com.br/blog/tecnologia/sistema-web-vs-sistema-local>Acesso em: 09 de Novembro de 2019.

DESCONHECIDO. **Conceitos de Software e Engenharia de Software.** Disponível em:

<https://www.devmedia.com.br/conceitos-de-software-e-engenharia-desoftware/15730>. Acesso em 02 de Novembro.2019

DESCONHECIDO. **Conceitos de Engenharia de Software**. Disponível em:

<https://www.academia.edu/6899720/Conceitos\_de\_Engenharia\_de\_Software>. Acesso em 01 de Novembro de 2019

CASTELLÓ, Thiago; VAZ, Verônica. **Tipos De Criptografia**. Disponível em:

<http://www.gta.ufrj.br/grad/07\_1/ass-dig/index.html>. Acesso em: 02 Novembro 2019.

ALECRIM, Emerson. **Criptografia**. 2005. Disponível em:

<http://www.infowester.com/criptografia.php>. Acesso em: 01 nov. 2019.

GARRET, Filipe. **O que é Criptografia?** 2014. Disponível em:

<http://www.techtudo.com.br/artigos/noticia/2012/06/o-que-e-criptografia.html>. Acesso em: 03 nov. 2019.

DESCONHECIDO. **Criptografia**. 2012. Disponível em:

<http://cartilha.cert.br/criptografia/>. Acesso em: 01 nov. 2019.

KESSLER, Gary C.. **An Overview Of Cryptography**. 1999. Disponível em:

<http://www.garykessler.net/library/crypto.html#password>. Acesso em: 04 nov. 2019.

VIANA, Ericksen. **Criptografia: Conceitos e Aplicações.** Disponível em: 30

<http://www.devmedia.com.br/criptografia-conceito-e-aplicacoes-revista-easy-netmagazine-27/26761>. Acesso em: 01 nov. 2019.

CONDE, A.R.; SOBRADO, P.P. **Vulnerabilidades Criptográficas**. Disponível em:<https://eseida.wikispaces.com/file/view/Vulnerabilidades+Criptográficas.pdf>. Acesso em: 01 nov. 2019.

SOARES, Salomão. **Criptografia**. 2012. Disponível em: <http://www.ceap.br/material/MAT15032012194445.pdf>. Acesso em: 01 nov. 2019.

BARRON, Brenda. **12 Important Advantages of Web Responsive Design.** 2018. Disponível em < https://business.tutsplus.com/articles/advantages-of-responsive-web-design--cms-30703>. Acesso em: 03 nov.2019

ZACCANINI, Rafael. **O que é e por que utilizar o ASP.NET MVC**. 2010. Disponível em < https://www.devmedia.com.br/o-que-e-e-por-que-utilizar-o-asp-net-mvc/18544>. Acesso em: 05 nov 2019

MACORATTI, José. **C# Validando dados com Data Annotations com Windows Forms.** Disponível em <http://www.macoratti.net/13/12/c\_vdda.htm>.Acesso em: 09 nov 2019

MEDEIROS, Higor. **Introdução ao padrão MVC.** 2013. Disponível em < https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>. Acesso em: 08 nov 2019

DESCONHECIDO. **O que é uma API?**, Disponível em < https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em: 08 nov 2019

DESCONHECIDO. **Scholarly Publishing – MIT Libraries.** Disponível em < https://libraries.mit.edu/scholarly/publishing/apis-for-scholarly-resources/>. Acesso em: 05 nov. 2019

TAKAI, Osvaldo; ITALIANO, Isabel; FERREIRA, Eduardo. **Introdução a banco de dados.** Disponível em < https://www.ime.usp.br/~jef/apostila.pdf>. Acesso em: 09 nov. 2019

DESCONHECIDO. **Banco de dados relacional e não relacional: Quando utilizar?.** Disponível em < https://mercadoemfoco.unisul.br/banco-de-dados-relacional-e-nao-relacional-quando-utilizar/>. Acesso em 09 nov. 2019

SCUDERO, Erick. **TOP 10 Principais SGBDs do mercado global.** Disponível em < https://becode.com.br/principais-sgbds/>. Acesso em: 09 nov 2019

DESCONHECIDO. **Material Kit**. Disponível em < https://www.creative-tim.com/product/material-kit>. Acesso em: 02 jun 2019

SOUZA, Antonio Anderson De. **O que é PWA (Progressive Web App) e porque isso pode aumentar seus resultados mobile.** Disponível em < https://vizir.com.br/2017/08/o-que-e-pwa-progressive-web-app-porque-isso-pode-aumentar-seus-resultados-mobile/>. Acesso em: 21 nov 2019

ROBERTO, João. **O que é SOLID : O guia completo para você entender os 5 princípios da POO.** Disponível em < https://medium.com/joaorobertopb/o-que-%C3%A9-solid-o-guia-completo-para-voc%C3%AA-entender-os-5-princ%C3%ADpios-da-poo-2b937b3fc530>. Acesso em 21 nov 2019

