**UNIVERSIDADE PAULISTA**

GABRIEL SANABRIA FERRERI

PEDRO MIURA SUGAYAMA

LEONARDO CARPEJANI PRESTES

ERICK EIJI KANAYAMA NAGANO

**Sistema para avaliação e desempenho de funcionários**

SÃO PAULO – SP

2019

**UNIVERSIDADE PAULISTA**

GABRIEL SANABRIA FERRERI

PEDRO MIURA SUGAYAMA

LEONARDO CARPEJANI PRESTES

ERICK EIJI KANAYAMA NAGANO

**Sistema para avaliação e desempenho de funcionários**

Trabalho de conclusão de curso como parte do exigido para obtenção do título de graduação em Ciências da computação apresentado à Universidade Paulista - UNIP

Orientador: Luiz Carlos Magrini

SÃO PAULO – SP

2019

**Ficha Catalográfica**

|  |
| --- |
| Sistema para avaliação e desempenho de funcionários / Gabriel Sanabria Ferreri...[et al.]. - 2019.  50 f. : il. color  Trabalho de Conclusão de Curso (Graduação) apresentado ao Instituto de Ciência Exatas e Tecnologia da Universidade Paulista, São Paulo, 2019.  Área de Concentração: Sistemas de Computação. Orientador: Prof. Dr. Luiz Carlos Magrini.    1. Sistema para avaliação. 2. Sistema de desempenho. 3. Web application. 4. MVC. 5. Orientação a Objeto. I. Ferreri, Gabriel Sanabria. II.  Magrini, Luiz Carlos (orientador). |

CIP - Catalogação na Publicação

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da Universidade Paulista com os dados fornecidos pelo(a) autor(a).

**UNIVERSIDADE PAULISTA**

GABRIEL SANABRIA FERRERI

PEDRO MIURA SUGAYAMA

LEONARDO CARPEJANI PRESTES

ERICK EIJI KANAYAMA NAGANO

**Sistema para avaliação e desempenho de funcionários**

Trabalho de conclusão de curso como parte do exigido para obtenção do título de graduação em Ciências da computação apresentado à Universidade Paulista - UNIP

Orientador: Luiz Carlos Magrini

Aprovado em:

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_/\_\_

Prof. João Augusto Fleury Silveira

Universidade Paulista – UNIP

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_/\_\_

Prof. Luiz Roberto Forcan

Universidade Paulista – UNIP

Resumo

O uso de ferramentas web no mercado vem se tornando cada vez mais comum, qualquer empresa da atualidade possui seu próprio sistema de gestão ou contabilidade. Algumas dessas ferramentas estão desenvolvidas em desktop, fazendo com que o aplicativo careça de interoperabilidade ou manutenibilidade. Com o intuito de atingir esses aspectos de suma importância, foi desenvolvido uma ferramenta web para cadastro e avaliação de funcionários. Para alcançar o que foi proposto, foram utilizadas ferramentas para o desenvolvimento web implementando a arquitetura MVC (Model View Controller).

Palavras Chave: Interoperabilidade, manutenibilidade

Abstract

The use of web tools in the market has become more and more prevalent, any company on the actuality has its own management or accountability system. Some of these tools are being developed as desktop application, raising lack of interoperability or maintainability. With the principle of reaching such important objectives, the development of a web application was required, to sign up and make an evaluation on employees. To reach what was proposed, the use of web development was demanded, implementing MVC architecture (Model, View, Controller).

Key Words: Interoperability, maintainability

**Lista de Figuras**

[Figura 1 - Modelo MVC 15](#_Toc26619895)

[Figura 2 - Modelo de Data Annotation 16](#_Toc26619896)

[Figura 3 - Modelo de requisição de uma API 18](#_Toc26619897)

[Figura 4 - Exemplo divisão em 3 camadas 22](#_Toc26619898)

[Figura 5 - Exemplo de Query à base 22](#_Toc26619899)

[Figura 6 - Método de criptografia utilizado 25](#_Toc26619900)

[Figura 7 - Exemplificação divisão em camadas 26](#_Toc26619901)

[Figura 8 - Diagrama Base de dados do sistema 27](#_Toc26619902)

[Figura 9 - Comparação base de dados com entidade 28](#_Toc26619903)

[Figura 10 - Exemplo de uso do Dapper 29](#_Toc26619904)

[Figura 11 - Exemplo de validação no Login 30](#_Toc26619905)

[Figura 12 - Uso de data binding Model View 31](#_Toc26619906)

[Figura 13 - Tela de Login do sistema 32](#_Toc26619907)

[Figura 14 - Home page 33](#_Toc26619908)

[Figura 15 - Tela Inicial Avaliação 34](#_Toc26619909)

[Figura 16 - Tela de avaliação com campos preenchidos 34](#_Toc26619910)

[Figura 17 - Fim do fluxo avaliação 35](#_Toc26619911)

[Figura 18 - Tela de edição 35](#_Toc26619912)

[Figura 19 - Validação cadastro novo usuário 36](#_Toc26619913)

[Figura 20 - Preenchimento automático pela API 37](#_Toc26619914)

[Figura 21 - Cadastro Finalizado 37](#_Toc26619915)

[Figura 22 - Tela de relatório 38](#_Toc26619916)

[Figura 23 - Casos de uso 43](#_Toc26619917)

[Figura 24 - Diagrama UML de classes 45](#_Toc26619918)

[Figura 25 - Exemplo de implementação do Dapper 49](#_Toc26619919)

**Sumário**

[**1.** **INTRODUÇÃO** 12](#_Toc26623784)

[1.1. OBJETIVOS 13](#_Toc26623785)

[**1.1.1.** **OBJETIVOS ESPECÍFICOS** 13](#_Toc26623786)

[1.1.2. ASPECTOS TÉCNICOS PARA DESENVOLVIMENTO 13](#_Toc26623787)

[**2.** **FUNDAMENTAÇÃO TEÓRICA** 14](#_Toc26623788)

[2.1. MVC (MODEL, VIEW, CONTROLLER) 14](#_Toc26623789)

[2.1.1. MODEL 15](#_Toc26623790)

[2.1.2. VIEW 15](#_Toc26623791)

[2.1.3. CONTROLLER 16](#_Toc26623792)

[**2.2.** **API** 17](#_Toc26623793)

[**2.3.** **BASE DE DADOS** 18](#_Toc26623794)

[**2.4.** **Divisão em 3 camadas** 20](#_Toc26623795)

[**2.5.** **CRIPTOGRAFIA** 22](#_Toc26623796)

[2.5.1. HASH 22](#_Toc26623797)

[2.5.2. UTILIZAÇÂO 24](#_Toc26623798)

[**3.** **DESENVOLVIMENTO** 24](#_Toc26623799)

[**3.1.** **CRIAÇÃO DO PROJETO** 24](#_Toc26623800)

[3.1.1. PROJETO EM CAMADAS 24](#_Toc26623801)

[3.1.2. DIAGRAMA DA BASE DE DADOS 26](#_Toc26623802)

[3.1.3. CONSTRUÇÃO ENTIDADE 26](#_Toc26623803)

[3.1.4. CAMADA DE ACESSO A DADOS 27](#_Toc26623804)

[3.1.5. CAMADA DE NEGÓCIOS 28](#_Toc26623805)

[3.1.6. CONSTRUÇÃO DO WEB MVC 29](#_Toc26623806)

[**3.2.** **FLUXO WEB** 30](#_Toc26623807)

[3.2.1. FLUXO AVALIAÇÃO 32](#_Toc26623808)

[3.2.2. FLUXO DE EDIÇÃO DE COLABORADOR 33](#_Toc26623809)

[3.2.3. FLUXO DE CADASTRO 34](#_Toc26623810)

[3.2.4. FLUXO DE RELATORIO 35](#_Toc26623811)

[**4.** **ELICITAÇÃO FICTÍCIA DO SISTEMA** 35](#_Toc26623812)

[**4.1.** **ELICITAÇÃO** 35](#_Toc26623813)

[**4.2.** **ESPECIFICAÇÃO DO SOFTWARE** 37](#_Toc26623814)

[4.2.1. INTERFACE DO APLICATIVO 37](#_Toc26623815)

[4.2.2. FUNÇÕES DO SISTEMA 38](#_Toc26623816)

[4.2.3. RESTRIÇÕES DE AMBIENTE 39](#_Toc26623817)

[4.2.4. DESCRIÇÃO DE USUÁRIOS 39](#_Toc26623818)

[**4.3.** **MODELAGEM DO SISTEMA** 40](#_Toc26623819)

[4.4. PLANOS DE TESTE 42](#_Toc26623820)

[**5.** **DISCUSSÃO** 45](#_Toc26623821)

[5.1. Framework para acesso à camada de dados 45](#_Toc26623822)

[5.2. Segregação de responsabilidade das classes 45](#_Toc26623823)

[5.3. Alta robustez do programa 46](#_Toc26623824)

[6. CONCLUSÃO 46](#_Toc26623825)

[**7.** **REFERÊNCIAS BIBLIOGRÁFICAS** 47](#_Toc26623826)

[**8.** **BIBLIOGRAFIA** 49](#_Toc26623827)

# **INTRODUÇÃO**

Com o advento de novas tecnologias, tornou-se cada vez mais difícil manter sistemas de nível legado em dia. As dificuldades recorrentes de se implementar novas funcionalidades, assim como desenvolver novas características ao programa vinham se tornando uma tarefa cada vez mais árdua e refinada. Tornou-se habitual enfrentar problemas graças à implementação de um novo componente desenvolvido. Temos como ênfase o fato de que no desenvolvimento web “os funcionários pode passar menos tempo efetuando manutenções e mais tempo se dedicando a tarefas como marketing, teste[...]” (BARRON, 2018)

Para suprir essa necessidade de atualização, de interoperabilidade e manutenibilidade, sistemas Web se destoaram como a chave e resposta para diversos problemas. Com o sistema web, é possível manter constantes atualizações, em conjunto com a capacidade de ter acesso em qualquer localidade. O maior desafio então tornou-se a forma de desenvolvimento desses ambientes. Para solucionar esse desafio, foi necessária a implementação de novos padrões de arquitetura, o que transformou o desenvolvimento desses sistemas para algo mais facilitado e segmentado. Um exemplo clássico disso é o modelo **MVC** (*Model, View, Controller*), que segmenta os projetos em 3 camadas para maior facilidade. Também é tido como destaque, o modelo **MVVM**(*Model-View-ViewMode*l). Ademais, pode ser citada a utilização de outros recursos como APIs, que podem funcionar como uma camada extra de busca de dados em qualquer projeto, uma vez que as APIs consistem em efetuar consultas, atualizações, inserções e deleções dentro de um determinado sistema.

A utilização de arquiteturas, como as supracitadas, torna o sistema mais robusto, com maior capacidade de atualização e manutenção ou desenvolvimento de seus novos componentes.

Visando demonstrar da capacidade desse tipo de aplicativo e desses conceitos, foi desenvolvido um aplicativo que uniria todas as características e facilidades de um ambiente portátil. Um aplicativo cujo intuito é a resolução de problemas com a gestão de pessoas dentro de uma determinada empresa.

# OBJETIVOS

O trabalho apresentado tem como objetivo a elaboração de um software para controle de avaliações e atividades de funcionários em uma microempresa, tendo como principal usuário apenas gestores ou funcionários de cargo superior, de uma específica área. O sistema fornecerá um *overview* do desempenho dos funcionários na empresa a partir de gráficos e relatórios gerados dando maior facilidade e conforto para o usuário final dentro da plataforma. Além de permitir que o gestor efetue a avaliação com um número de atividades que ele desejar.

# **OBJETIVOS ESPECÍFICOS**

* Conseguir utilizar um framework para acesso à camada de dados
* Alcançar uma de segregação de responsabilidade de classes.
* Alcançar uma alta robustez de sistema.

# ASPECTOS TÉCNICOS PARA DESENVOLVIMENTO

Como a natureza do problema estudado é a falta de interoperabilidade e manutenibilidade, seja de sistemas legados ou sistemas desktop, foi efetuado um estudo que teve como base a leitura de código de ambas as ferramentas desenvolvidas para esse ambiente.

Para início, foi pensado utilizar o paradigma da linguagem de programação orientada à objetos, com o intuito de manter a manutenibilidade extremamente fácil, quando aplicado corretamente. Uma vez que, a partir disso, será possível segregar a responsabilidade das classes e de seus métodos, abrindo então a chance da utilização de uma interface de desenvolvimento.

Em conjunto com a orientação a objeto, foi utilizado também o conceito de MVC, que, como foi supracitado, tem o intuito de manter a manutenibilidade do sistema e a segregação, em adendo que a partir do uso do MVC, é efetuada uma adaptação, e então, o conceito de ViewModel padrão passa a ser assumido pela Model, e o conceito de Model é então assumido pela camada de entidades, tornando necessária a utilização de camadas, e evitando assim, maiores dependências de objetos contidos em um mesmo projeto.

Como foi necessário dinamismo dos dados, foi estudada e planejada a construção de uma base de dados relacional SQL, e para seu manuseio foi utilizado o Microsoft Sql Server Management Studio. As consultas em base não foram feitas de forma completamente segura para demonstração apenas

Enfim, todas as devidas partes foram agrupadas para desenvolvimento do protótipo do sistema e durante seu desenvolvimento, a separação de camadas foi efetuada, tendo como camadas: DAO (Objeto de acesso a dados), Business (Regras de negócio), Entity (Representando as entidades da base), Util (Utilitários para o programa, contendo códigos para uso genérico, sendo criptografia ou qualquer outro algoritmo que fosse utilizado por todo o projeto), e o próprio projeto Web com template MVC.

# **FUNDAMENTAÇÃO TEÓRICA**

Neste capítulo serão demonstrados todos os fundamentos no desenvolvimento do projeto, relacionados a Model, View, Controller, API, e conexão com base de dados, criptografia para os dados salvos na base de dados e divisão em 3 camadas.

# MVC (MODEL, VIEW, CONTROLLER)

Padrão de arquitetura de software, que 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 da arquitetura MVC consiste em, segundo Zaccanini (2010) , separação das camadas de regras e das camadas de lógica do negócio da apresentação, permitindo que seja exercido um maior controle sobre o desenvolvimento e sobre a aplicação em si.

A lógica que manipula os dados está na model, e contida apenas na model; a lógica que mostra os dados fica na view, e o código que manuseia as requisições do usuário e seus valores de entrada é contido apenas na Controller. Com essa clara divisão entre cada pedaço, a aplicação será mais facilmente mantida e estendida conforme seu tempo de vida, não importando quão grande fique. (FREEMAN, Adam. 2013 p.52)

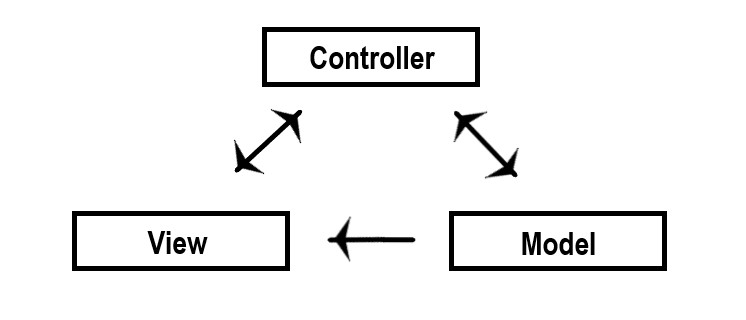


Figura 1 - Modelo MVC

* + 1. MODEL

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

É nessa camada que, por boas práticas, é efetuada a validação dos dados em propriedades do C#, além de uma predefinição de como a *View* deve renderizar essas informações a partir de um conceito conhecido como **Data Annotation**, que são nada mais que formas de efetuar determinada validação dentro do escopo das entidades.

O conceito de Data Annotation se estende além da *model*, e pode ser aplicada na camada de entidade de um projeto. Segundo Burbeck (1992) “[...] A *model* controla o comportamento e os dados da aplicação dominante, respondendo a requisições sobre seu status, e respondendo a instruções de mudança de estado.”

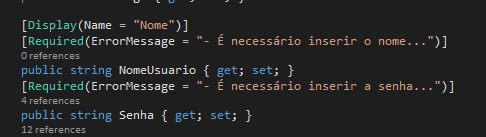


Figura 2 - Modelo de Data Annotation

* + 1. VIEW

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.

Na camada de View, ocorrem as validações iniciais, em *Client,*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. Majeed (2018, p.1) afirma que: “[…] Views são utilizadas para preparar a interface da aplicação. Utilizando essa interface, o usuário consegue interagir com a aplicação.”

* + 1. CONTROLLER

“*Controllers*, que processam as requisições que chegam, performam operações na *Model*, e seleciona quais *Views* irá renderizar ao usuário” (FREEMAN, Adam. 2013. p.51).

Ou seja, uma camada de controle é responsável por manusear todas as requisições do usuário e efetuar validações mais profundas dentro dos dados. Na camada de controle é efetuada a chamadas às outras camadas contidas no sistema. No caso do projeto que será 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,** que irão retornar uma ViewResult contida, que, com os dados inseridos, irá então mostrar ao usuário o que aconteceu com os valores inseridos, ou mostrar novos dados a partir dos valores inseridos.

# **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 necessária a implementação de uma API ou Middleware que traga os dados dessa base de dados externa para o sistema. O que torna o trabalho menos árduo é a facilidade com a qual é possível implementar uma API no sistema.

APIs 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 ou para aplicações de nível legado no ramo empresarial. APIs geralmente são desenvolvidas seguindo o conceito de REST, acrônimo para **Representational State Transfer,** sendo um estilo de arquitetura, que em seu conceito, responde conforme estados. Fielding (2000, p.18) afirma em seu artigo científico “[...]REST enfatiza a escalabilidade das interações de um componente, generalidade das interfaces e implantação dos componentes independentes[...]reduzindo a latência[...]”

Os principais métodos com o qual uma API interage com o servidor e recebe requisições HTTP são: Get, Put, Post, Delete.

* 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

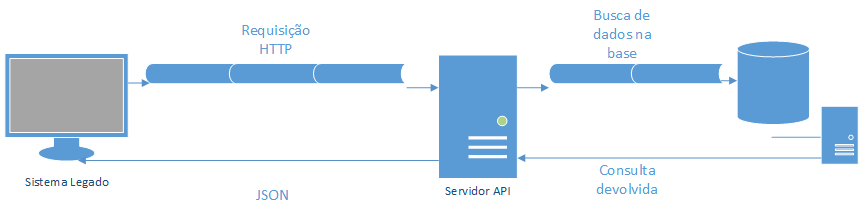


Figura 3 - Modelo de requisição de uma API

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 ou XML à aplicação, e então o sistema pode trabalhar com os dados devolvidos.

Uma biblioteca com base em API providencia um conjunto de pontos de entrada de códigos e símbolos/parâmetros associados permitindo que um programador pode usar o código de alguém para fazer o trabalho sujo de manter a interface atual entre sistemas parecidos, considerando que o programador obedeça às restrições de arquitetura e linguagem que venham com esse código. (FIELDING, Roy Thomas. 2000. p.138)

# **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, Scudero (2016) afirma em seu artigo que “Estes últimos, cada vez mais vem conquistando espaço nas organizações”. 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.

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.

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 Chave Estrangeira (FK).

Quando se trabalha com base de dados, para que o sistema possa pegar as informações, é feito um Modelo Entidade-Relacionamento entre o sistema e a base, no qual é criada uma camada de armazenamento desses dados, conhecida como Repositório, ou seja, esse modelo trata de como os dados serão mostrados na base de dados, porém, não mostra quantidade tampouco seus valores.

[...]é uma descrição do banco de dados de forma independente de implementação em um SGBD. O modelo conceitual registra que dados podem aparecer no banco de dados, mas não registra como estes dados estão armazenados a nível de SGBD (HEUSER,Carlos Alberto. 2008 p.5)

O Modelo Entidade-Relacionamento deve conter uma propriedade chamada Cardinalidade, como Heuser (2008) afirma “[...]uma propriedade importante de um relacionamento é a de quantas ocorrências de uma entidade podem estar associadas a uma determinada ocorrência através do relacionamento.” Ou seja, o uso da cardinalidade passa a ser necessário para o desenvolvimento de um sistema robusto, dado que a demonstração dessa informação para o programador, permite verificar qual a quantidade de tabelas e informações que serão alteradas quando ocorrer uma inserção, deleção, ou até mesmo uma atualização dos dados ali contidos.

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, para que o mesmo, trabalhe sobre esses dados recebidos, 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.

A divisão em 3 camadas tem como objetivo gerar desacoplamento de regras contidas dentro dos projetos que formam o sistema, mantendo então sua robustez, flexibilidade e possível resistência a quebra com algumas mudanças que forem implementadas no sistema. Isso acontece porque cada camada manterá suas próprias regras de acesso e de validação dos dados, e devolverá para a camada superior a informação requerida.

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.

S.O.L.I.D. é uma coleção de melhores práticas, para princípios de design orientado a objeto que podem ser aplicados ao seu design, permitindo que se cumpram vários objetivos desejados, como menor responsabilidade, maior manutenibilidade, locação intuitiva de códigos[...] (OCAMPO, Joe, 2011, p.4)

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. Chen (2003, p.3) afirma ainda sobre a camada de dados que “[...] uma vez que os dados são o aspecto mais complexos de muitas informações do sistema, ela é essencial na estruturação do sistema[...]”

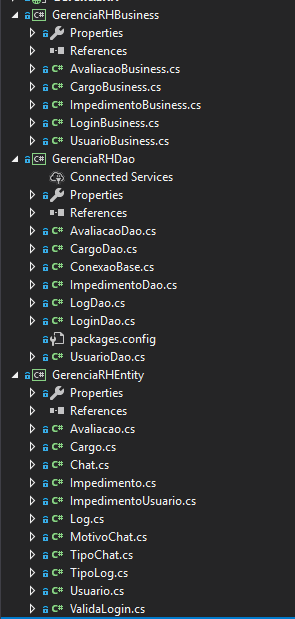


Figura 4 - Exemplo divisão em 3 camadas

A imagem acima demonstra a divisão prática do projeto em 3 camadas, Business (Camada de negócios), DAO (Camada de acesso a dados), e Entity (Camada de entidades).

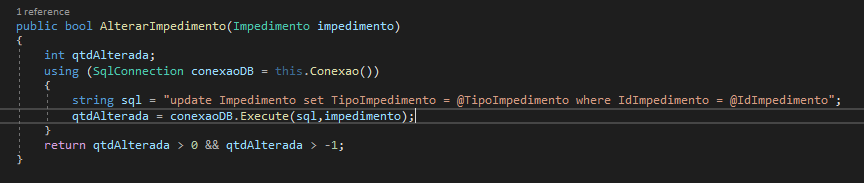


Figura 5 - Exemplo de Query à base

# **CRIPTOGRAFIA**

# HASH

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.

Encriptar é a principal aplicação da criptografia; faz com que os dados se tornem incompreensíveis com o objetivo de garantir a confidenciabilidade. A encriptação usa um algoritmo chamado cifra e um valor secreto chamada chave; se o valor da chave é desconhecido, não é possível descriptografar, tampouco saber sobre qualquer tipo de informação na mensagem encriptada – E o atacante tampouco[...] (AUMASSON, Jean-Phillipe, 2018, p.23)

**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)

Segundo Moreno (2005, p.30), “a chave que é utilizada para criptografar os dados é a mesma chave utilizada para decifrá-los”, ou seja, não há alterações de chaves, por esse motivo, tanto a pessoa que envia quanto a pessoa que recebe possuem a mesma chave, Moreno (2005, p.30) também ressalta que “Na criptografia assimétrica, usa-se outra chave que possui um valor relacionado com essa primeira chave”. Ou seja, a principal diferença entre a chave simétrica e a chave seguinte é a utilização e o meio de origem de ambas.

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 Castelló (2016) “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”

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.

# UTILIZAÇÂO

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.

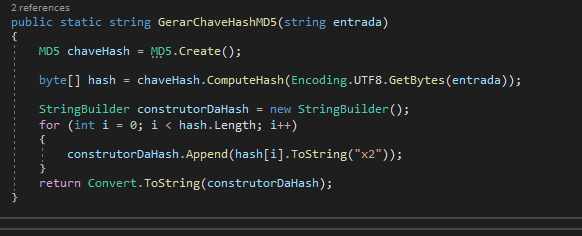


Figura 6 - Método de criptografia utilizado

# **DESENVOLVIMENTO**

Neste capítulo será descrito como foi sucedido todo o desenvolvimento do trabalho, tendo como base o capítulo 2

# **CRIAÇÃO DO PROJETO**

De início, o projeto como um todo teria o foco em um ambiente Web com acesso à um servidor que conteria a base de dados das informações necessárias, portanto, o primeiro passo tomado foi a criação do template da solução na ferramenta Visual Studio, a criação será descrita em partes.

# PROJETO EM CAMADAS

Durante a criação e divisão do projeto, já foi pensado o como seria feita a construção do mesmo em um ambiente de fácil desenvolvimento, portanto, o projeto foi criado com o seguinte template:

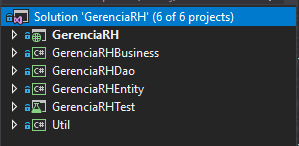


Figura 7 - Exemplificação divisão em camadas

* GerenciaRH: Parte WEB
* GerenciaRHBusiness: Regras de negócio de acesso à base
* GerenciaRHDao: Regras de acesso à base (comandos)
* GerenciaRHEntity: Camada que possui a Entidade representante
* GerenciaRHTest: Camada simples de teste unitário para teste da aplicação
* Util: Camada de utilitários, que conterá os algoritmos extras

# DIAGRAMA DA BASE DE DADOS

Para o início da construção do projeto, foi utilizado um diagrama de construção de base de dados

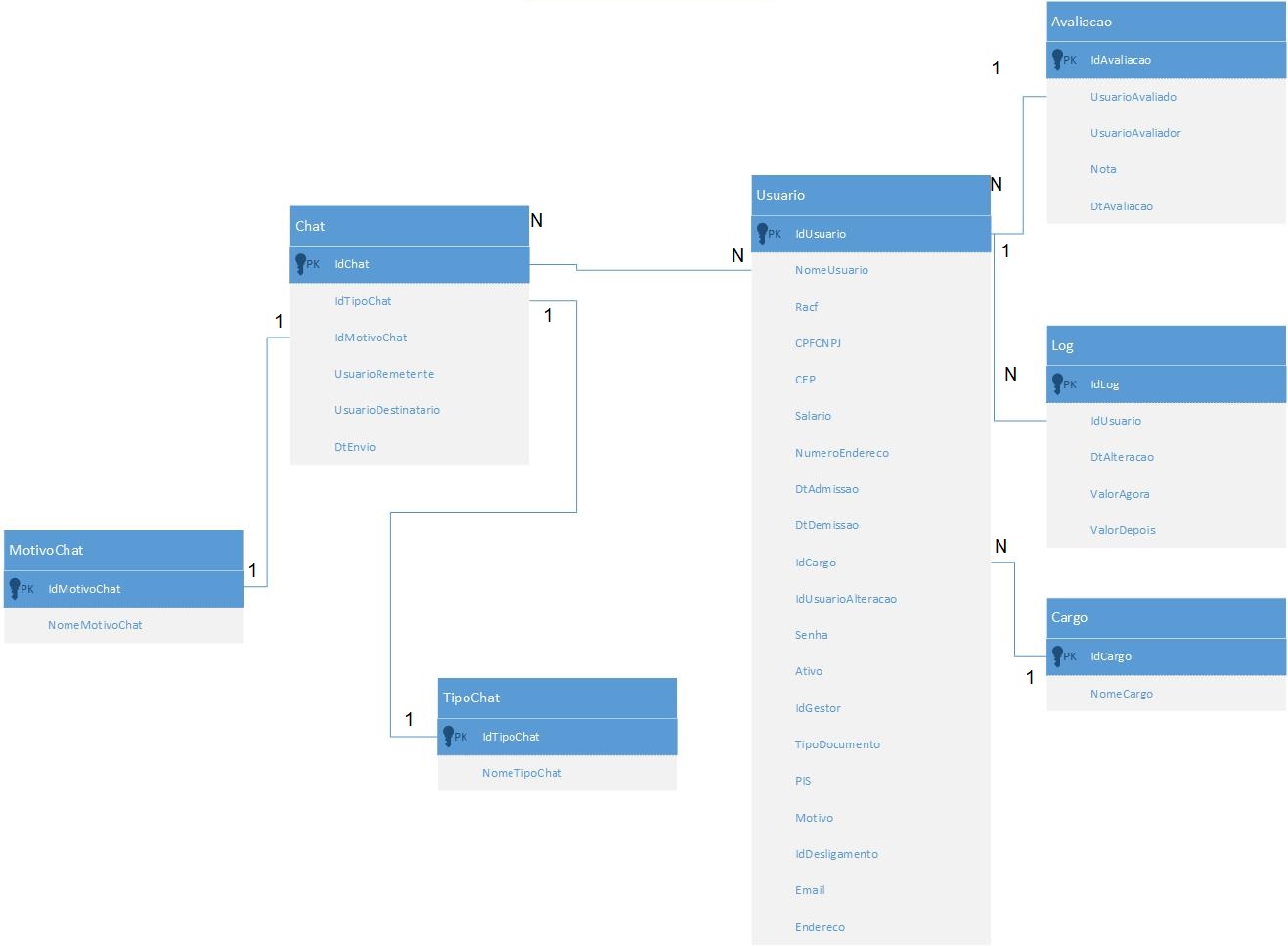


Figura 8 - Diagrama Base de dados do sistema

Baseando-se nesse diagrama, foi instalado e construída uma base dentro do protótipo. E então efetuado um cadastro simples de cada entidade dentro das tabelas, para facilidade durante o desenvolvimento do projeto.

# CONSTRUÇÃO ENTIDADE

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. O exemplo abaixo representa o modelo da classe Usuário, com a entidade do banco de dados Usuário

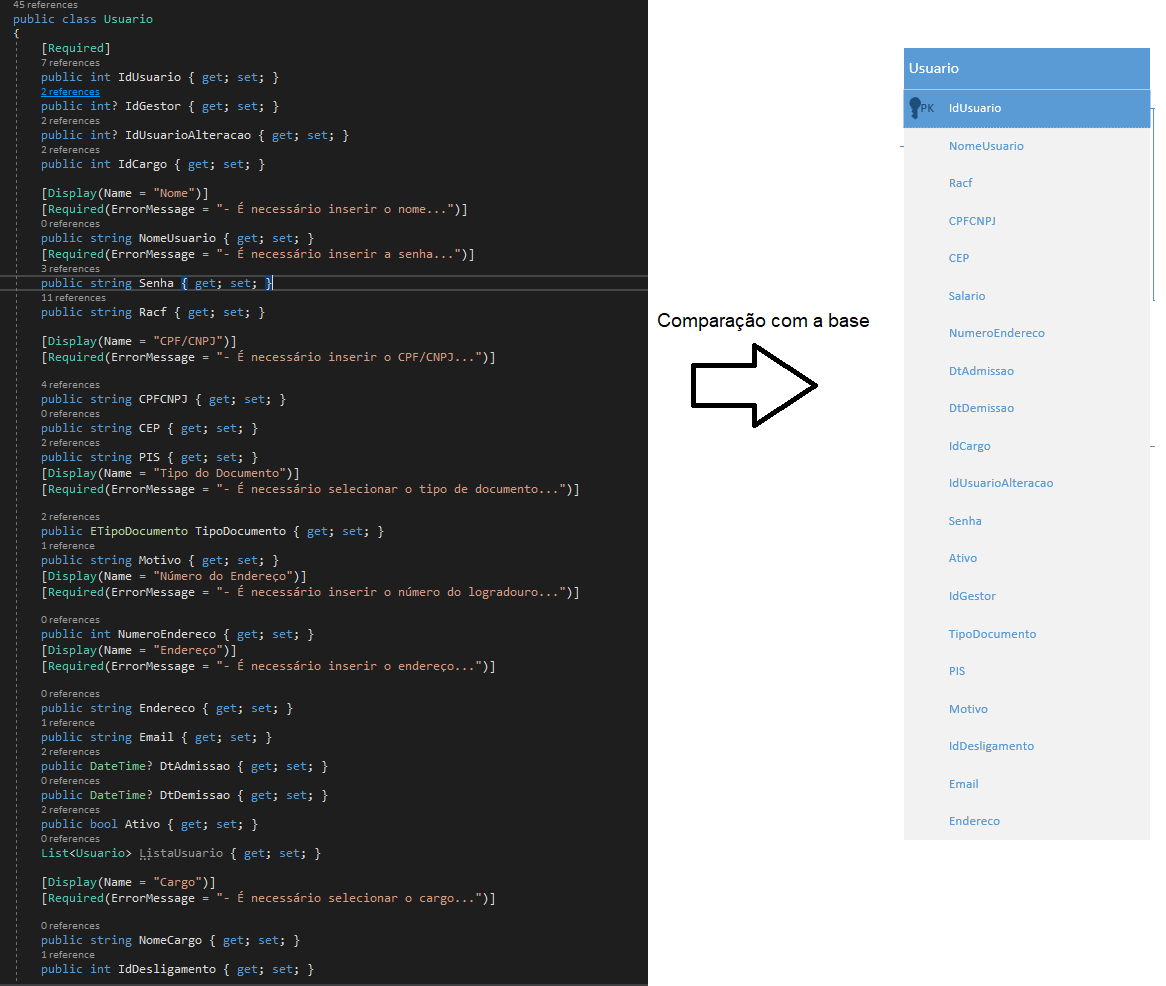


Figura 9 - Comparação base de dados com entidade

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.

# CAMADA DE ACESSO A DADOS

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)

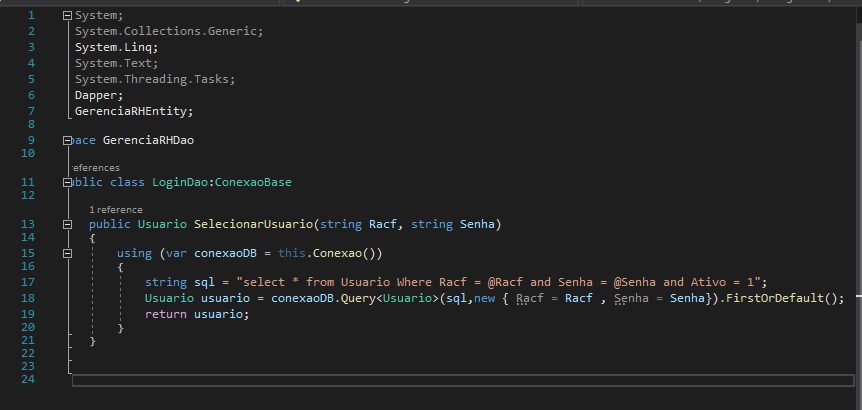


Figura 10 - Exemplo de uso do Dapper

É possível ver na imagem que 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.

# CAMADA DE NEGÓCIOS

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.

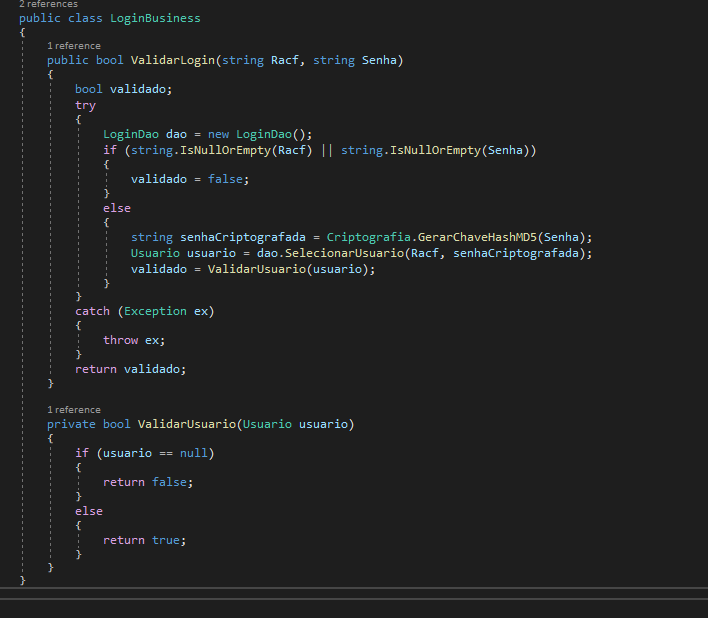


Figura 11 - Exemplo de validação no Login

# CONSTRUÇÃO DO WEB MVC

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.

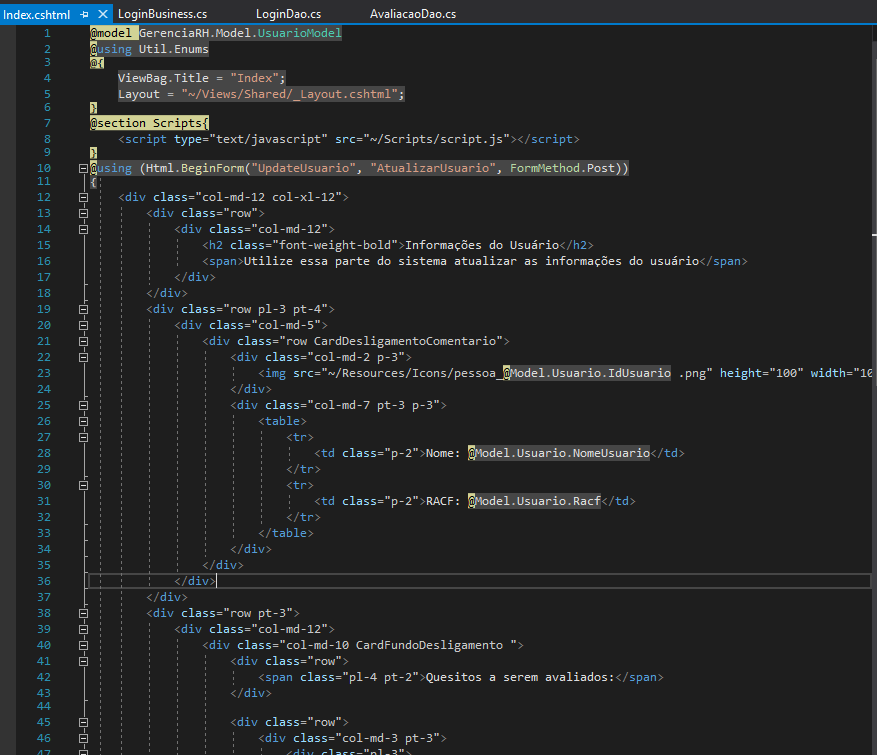


Figura 12 - Uso de data binding Model View

# **FLUXO WEB**

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
* Avaliação de um funcionário de sua área
* Obtenção de relatórios atuais, mensais e anuais
* Edição de usuário
* 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

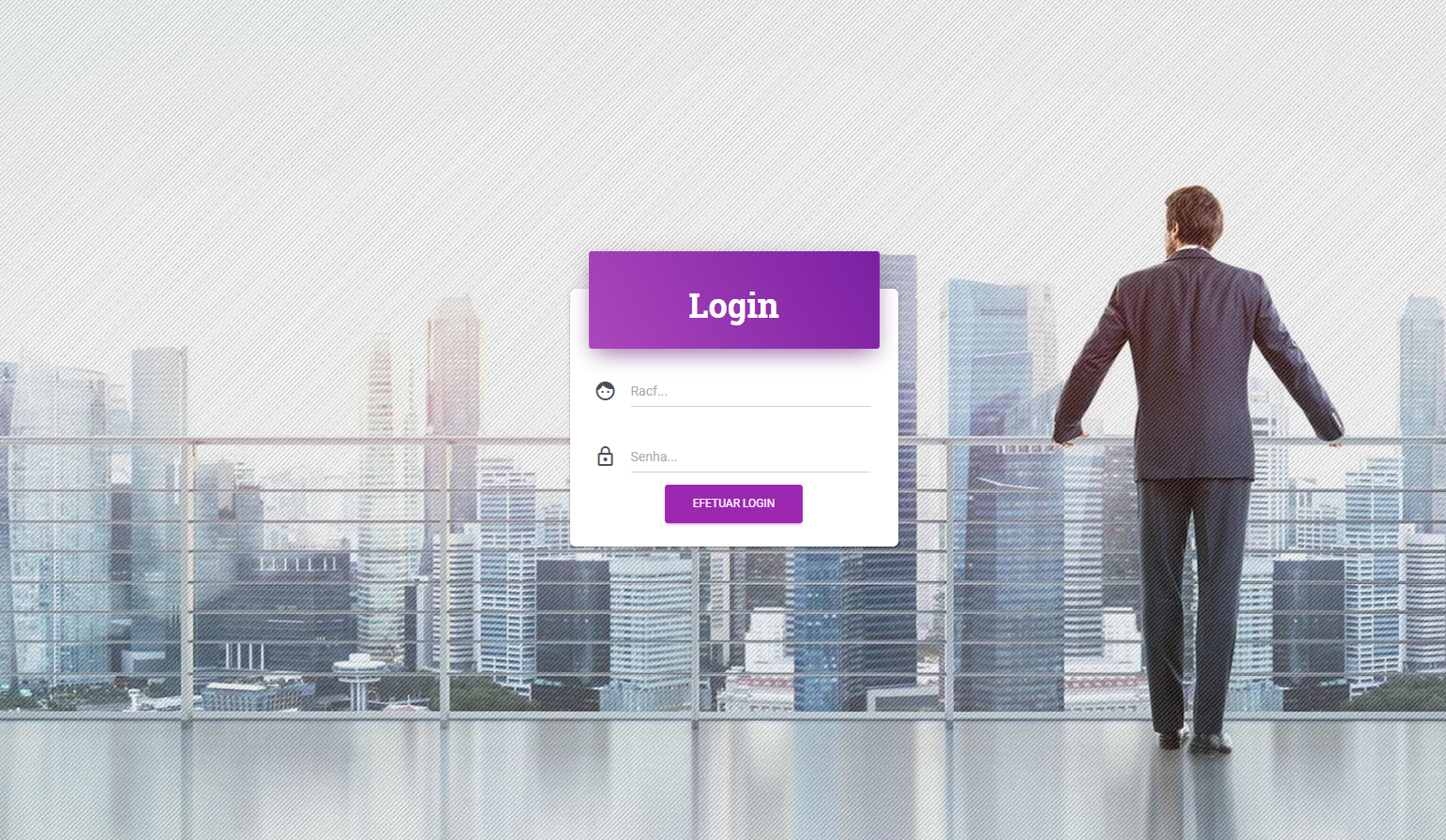


Figura 13 - Tela de Login do sistema

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

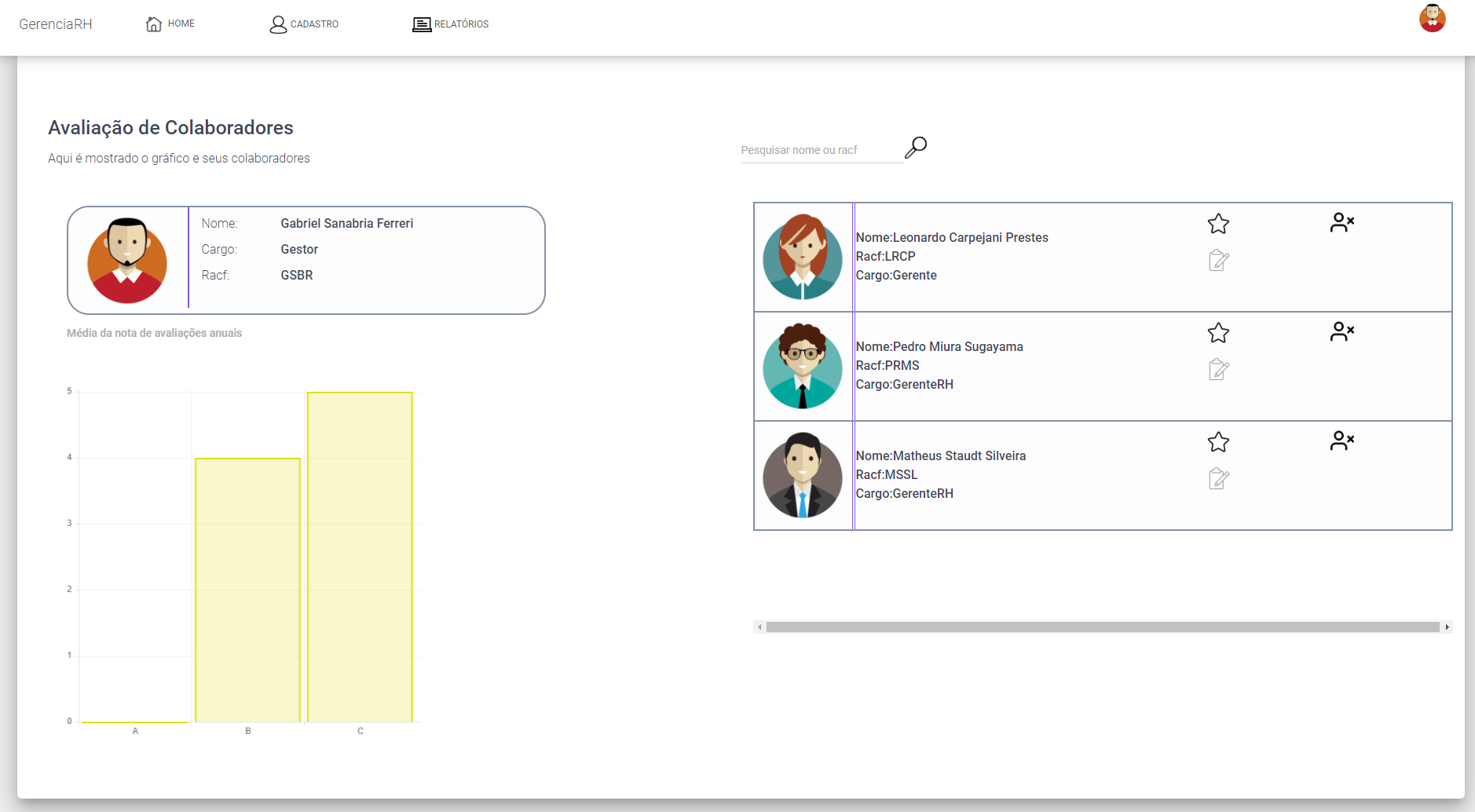


Figura 14 - Home page

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

# FLUXO AVALIAÇÃO

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

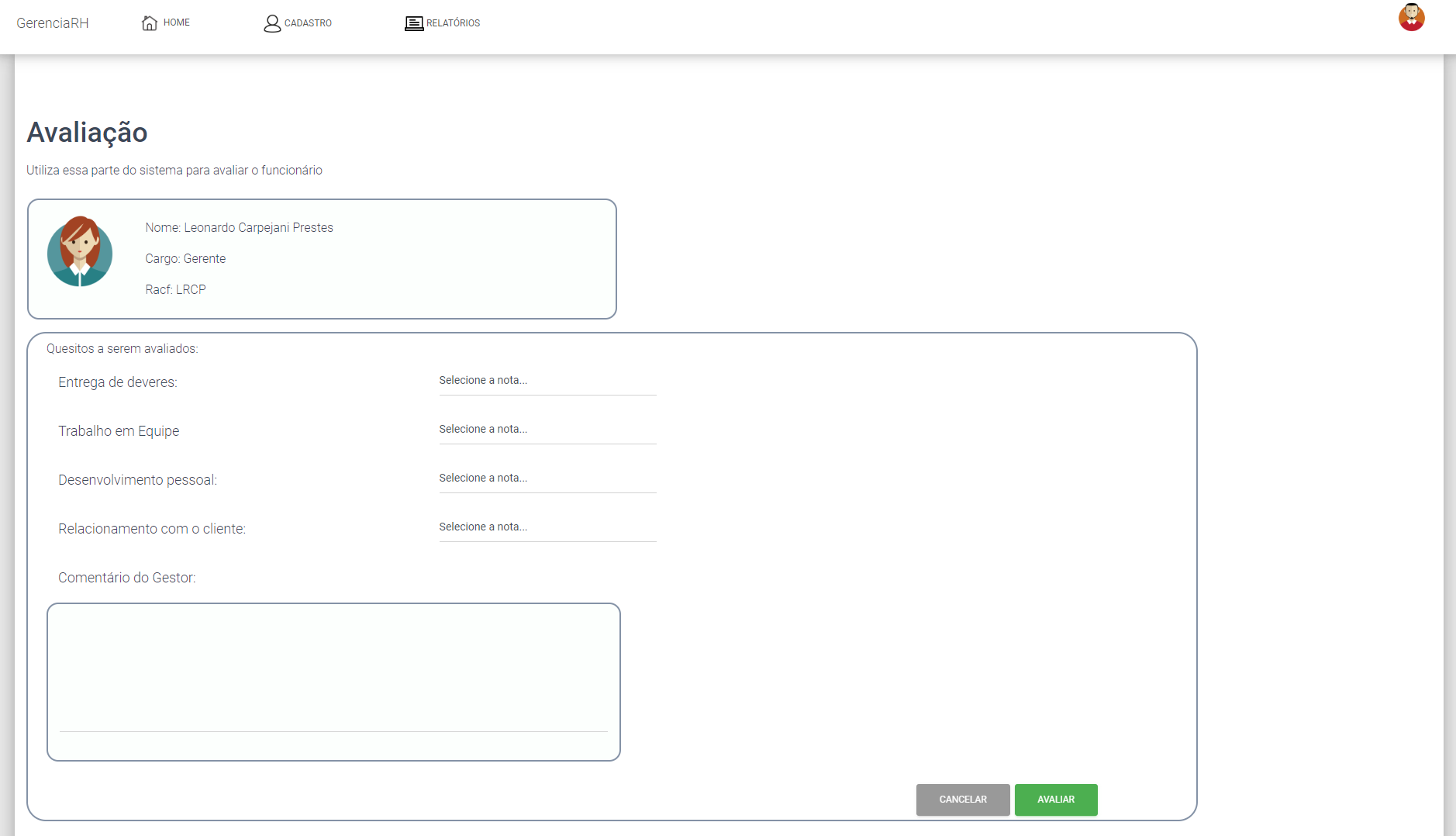


Figura 15 - Tela Inicial Avaliação

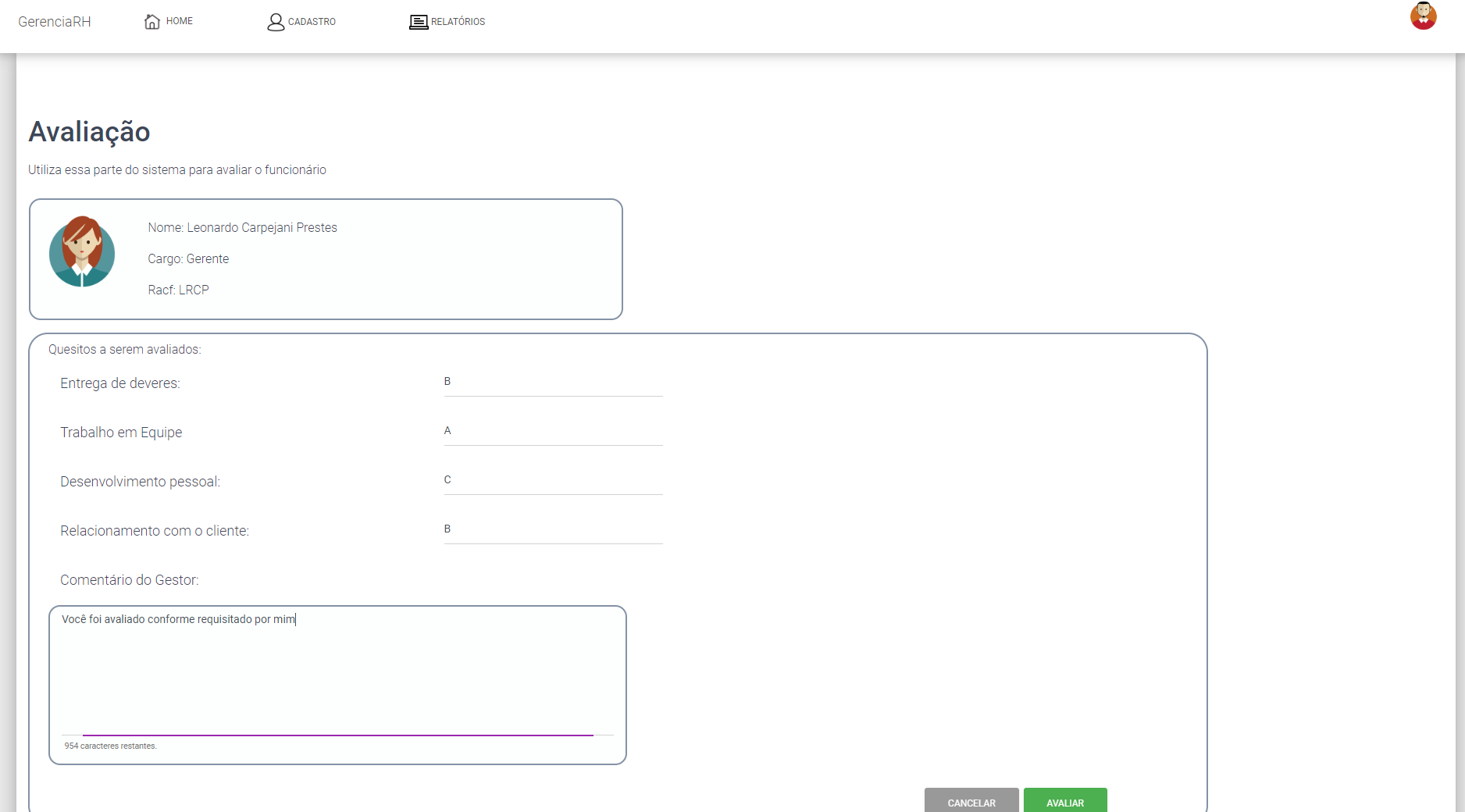


Figura 16 - Tela de avaliação com campos preenchidos

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.

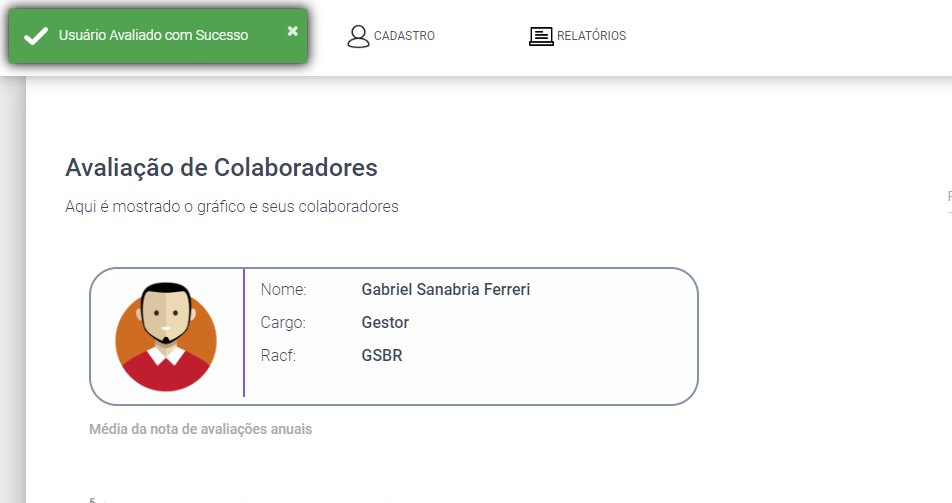


Figura 17 - Fim do fluxo avaliação

# FLUXO DE EDIÇÃO DE COLABORADOR

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

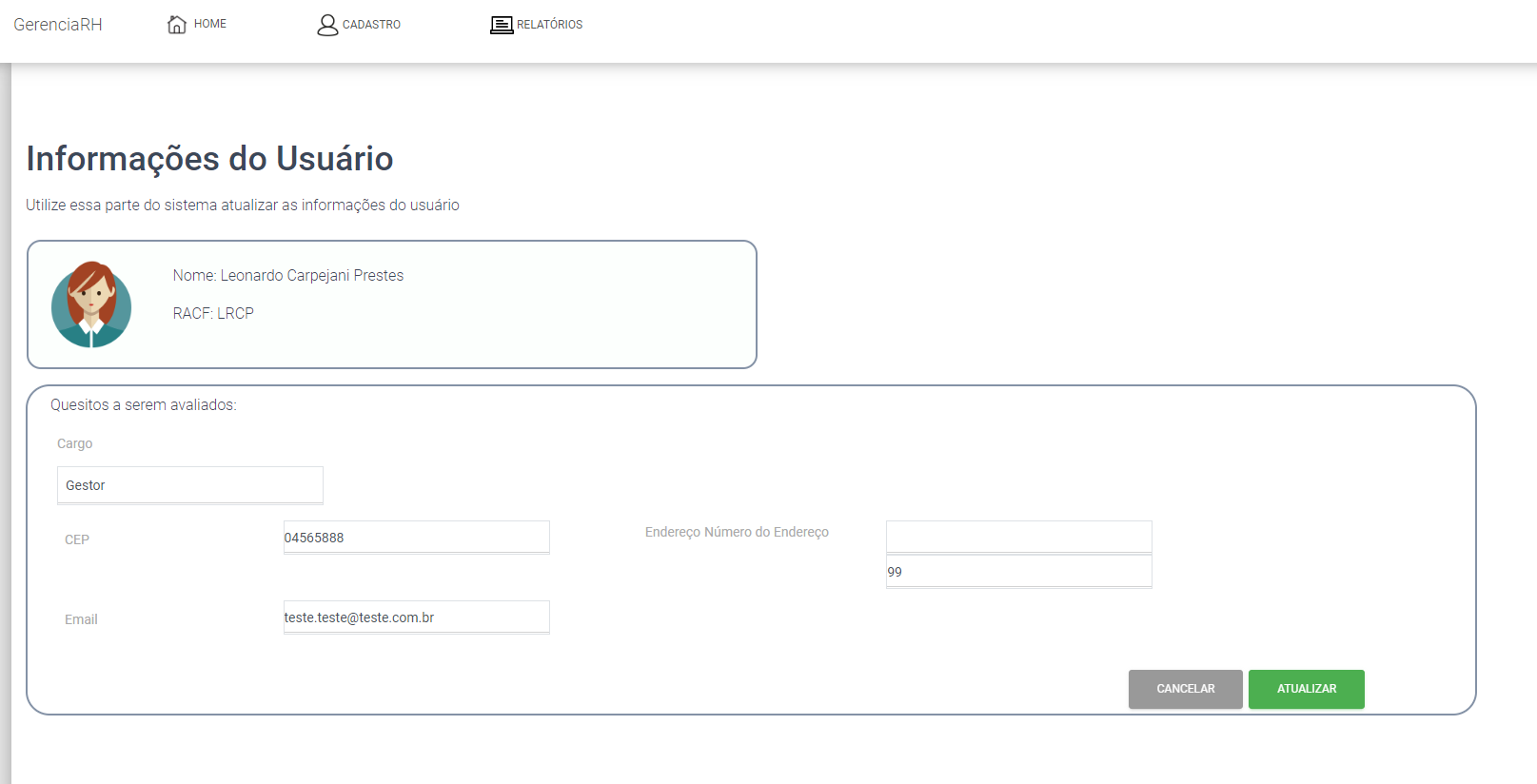


Figura 18 - Tela de edição

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

# FLUXO DE CADASTRO

Ao clicar na aba de cadastro, no 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 validação utilizada, foi feita pelo Data Annotation.

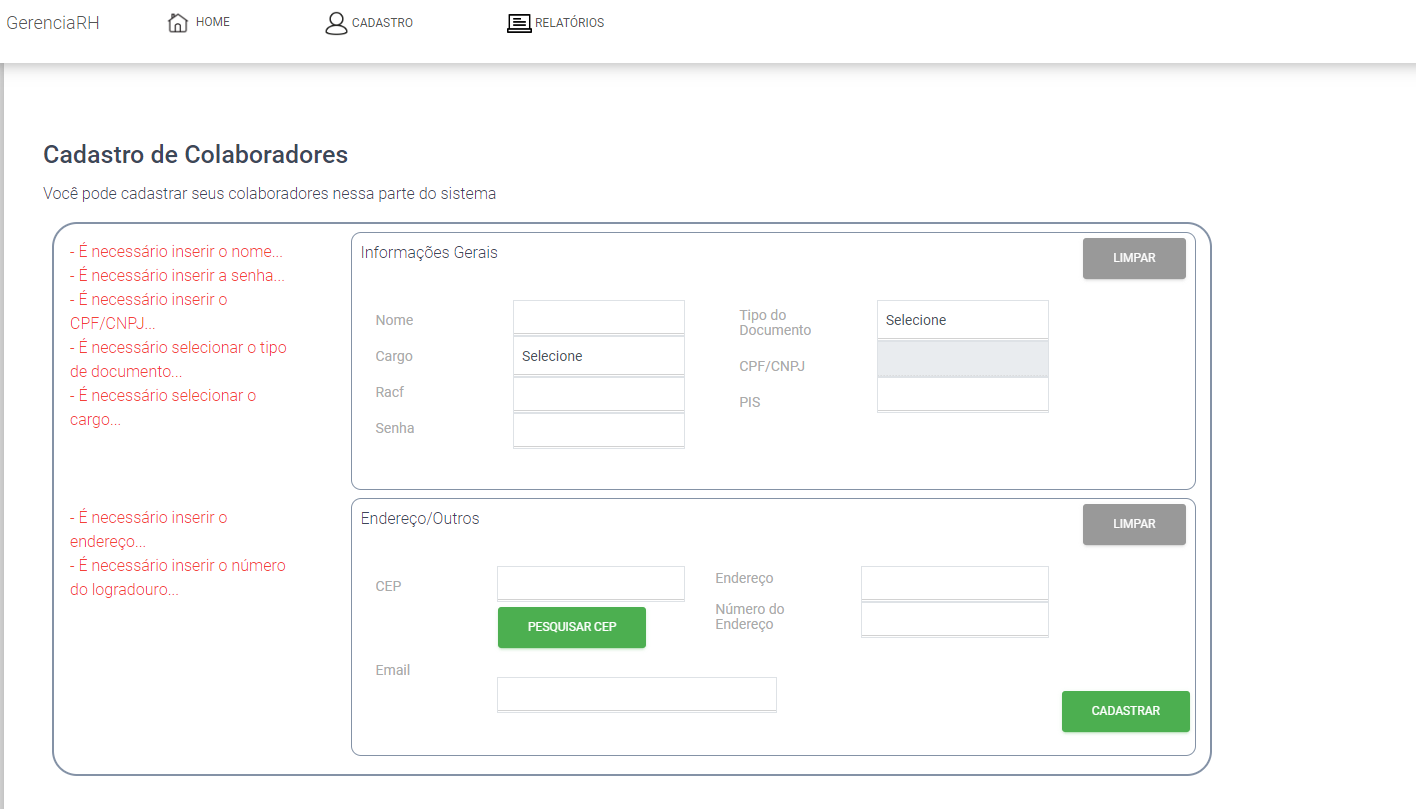


Figura 19 - Validação cadastro novo usuário

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

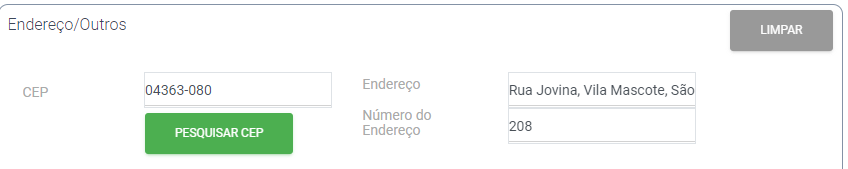


Figura 20 - Preenchimento automático pela API

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

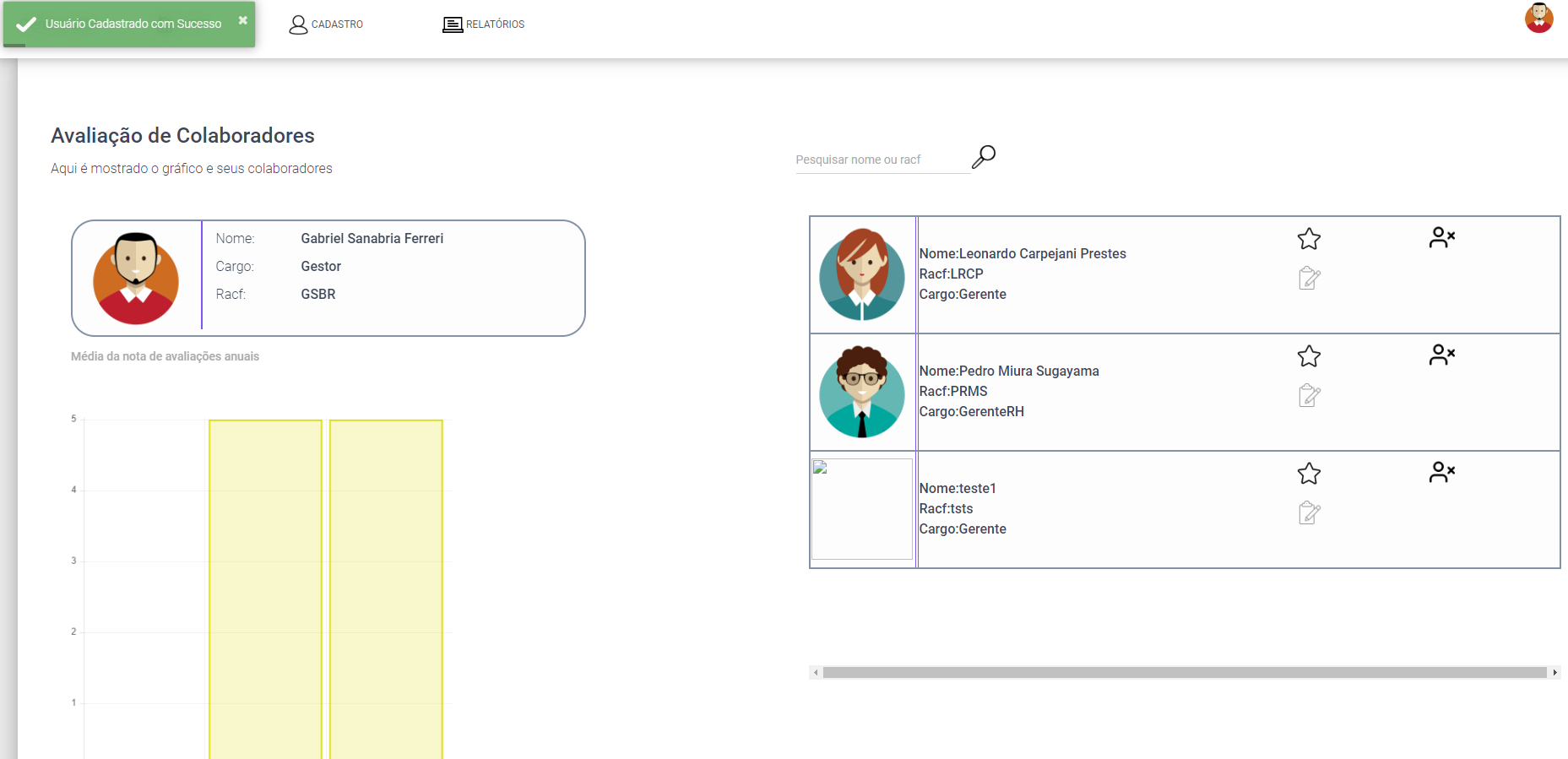


Figura 21 - Cadastro Finalizado

# FLUXO DE RELATORIO

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, sendo possível a extração dos dados cadastrados pelo sistema para uma planilha apartada.

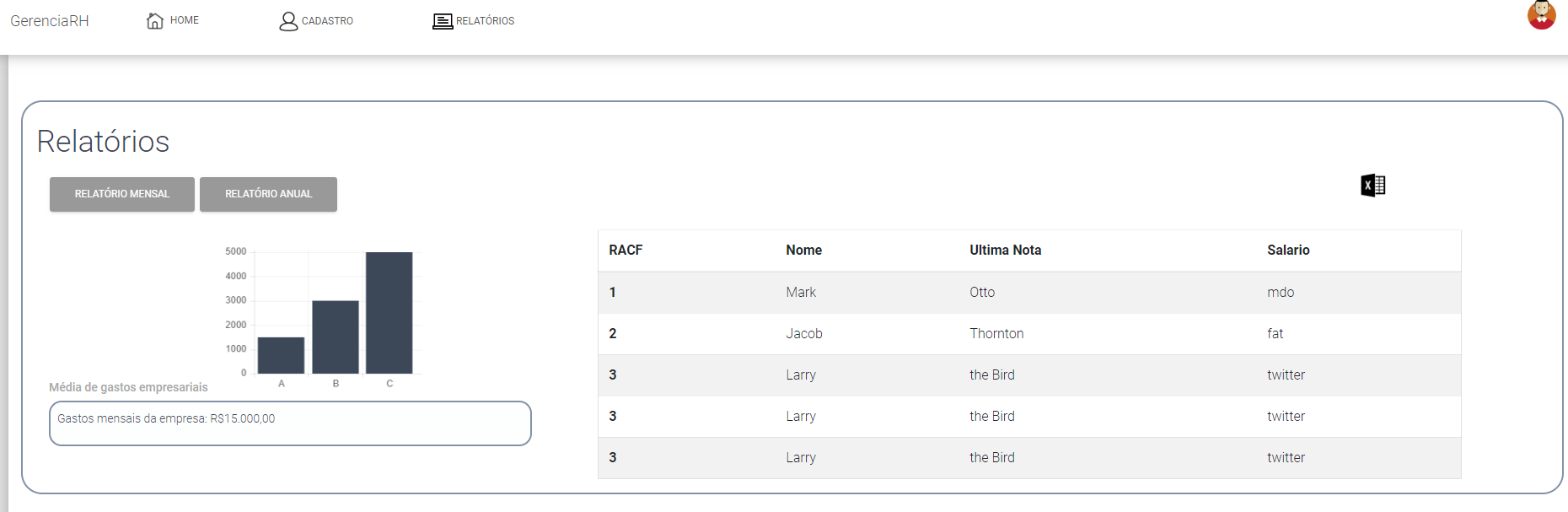


Figura 22 - Tela de relatório

# **ELICITAÇÃO FICTÍCIA DO SISTEMA**

# **ELICITAÇÃO**

CONTEXTUALIZAÇÃO

**O que é pedido**: É pedida a criação de uma ferramenta que efetue a gestão de colaboradores de áreas apartadas em particular, no qual apenas o gestor terá acesso ao sistema

**Problema**: RH Ineficiente para processos de demissão ou relatório.

**Afeta**: Colaboradores das áreas requeridas e gestor

**Impacto**: Médio

**Solução**: Criação de uma plataforma de Gestão e RH que tenha capacidade de efetuar relatórios e início do processo de demissão

**Posição Geral**: O aplicativo será criado para o cliente juntamente de um analista de requisitos entregue pela própria empresa junto com um analista de negócios para avaliação de que a ferramenta, que cumpra o que é pedido.

**Sentença** de Posição

**Para**: Empresa fictícia

**Quer**: Um aplicativo que efetue o auxílio do gestor dentro da empresa, permitindo processos iniciais de demissão e relatórios

**O que faz o aplicativo**: Permite que seja feito o cadastro de um gestor no sistema, logo depois, o gestor irá efetuar o cadastro de seus funcionários para gestão de salários totais da área

**DESCRIÇÃO DOS ENVOLVIDOS**

Os envolvidos com o sistema serão, a empresa, que comprou a ferramenta, e seus clientes a partir da comunicação efetuada pelo sistema. Todo ou qualquer colaborador de cargo de gerente ou superior, poderá efetuar o cadastro de seus colaboradores no sistema para controle e gestão dos mesmos.

**AMBIENTE DO USUÁRIO**

O usuário irá efetuar o login, dado seu acesso. Após entrar no sistema, cadastrará cada um dos funcionários, e conforme regras da empresa, efetuará avaliações dentro da aplicação de tempos em tempos. Serão disponibilizados Dashboards de avaliações e dashboards mensais do desempenho e gasto de sua respectiva área.

**VISÃO GERAL DO QUE FOI CONTRATADO (PRODUTO)**

O produto será baseado em ambiente web. Para toda ou qualquer entrada no sistema, deverá ser efetuado um login conforme cadastrado na base de dados para que então, ocorra a validação, por meio de Hash, da senha cadastrada com o usuário, e o mesmo consiga entrar no sistema, tendo então o completo acesso ao sistema e suas funcionalidades

**ANÁLISE E NEGOCIAÇÃO**

O desenvolvimento do aplicativo tem como objetivo facilitar a gestão dos gerentes e superiores, com o intuito de aliviar a demanda do departamento de RH da empresa, ao solicitarem planilhas e relatórios, ou pedidos de demissão, farão isso pelo software

**APLICATIVO**

a. Deve ser efetuado um login inicial, de um superior, para que então possam ser cadastrados os colaboradores sob sua gestão da área.

b. Deve ocorrer o envio de e-mail automático a partir de uma caixa anônima, o pedido de desligamento de qualquer funcionário

c. Deve ser colocada a opção de gerar relatório.

d. Não deve ser possível responder à caixa de e-mail, ou seja, o mesmo tem que efetuar a deleção automática de qualquer e-mail recebido

e. Deve ocorrer o armazenamento em um arquivo de log, na base de dados, de toda ou qualquer ação do usuário

**CLIENTE**

a. Os colaboradores devem ser cadastrados no sistema de forma completa, com todas as informações pedidas no formulário

# **ESPECIFICAÇÃO DO SOFTWARE**

# INTERFACE DO APLICATIVO

|  |  |  |
| --- | --- | --- |
| Número | Interfaces | Descrição |
| 1 | Tela de Login | Usuário efetuará acesso ao sistema por essa tela |
| 2 | Tela de início | Usuário terá acesso a dashboard de notas e a todas as outras telas |
| 3 | Tela de edição | Usuário poderá editar dados contidos de determinado colaborador |
| 4 | Tela de Cadastro | Usuário irá efetuar a inserção das informações de determinado usuário |
| 5 | Tela de relatório | Usuário conseguirá ter acesso à tela de dashboard para controle de gastos |
| 6 | Tela de Desligamento | Usuário conseguirá efetuar desligamento de qualquer colaborador sob a gestão de sua gerência |
| 7 | Tela de Avaliação | Usuário conseguirá efetuar avaliação de determinado colaborador |

# FUNÇÕES DO SISTEMA

|  |  |  |
| --- | --- | --- |
| Número | Funções | Descrição |
| 1 | Cadastro de colaborador | O sistema deverá permitir o cadastro de colaborador dentro do sistema logado |
| 2 | Desligamento de colaborador | O sistema deverá permitir o desligamento e implementar conforme regulamentos da empresa |
| 3 | Avaliação de Colaborador | O sistema deverá permitir avaliações constantes a princípio, porém, assim que possível, normalizar às regras da empresa |
| 4 | Edição de colaborador | O sistema deve permitir edição de informações cruciais ao gestor da área |
| 5 | Emissão de Relatórios | O sistema deve permitir a emissão de relatórios para facilitação do trabalho do gestor |

# RESTRIÇÕES DE AMBIENTE

|  |  |  |
| --- | --- | --- |
| Número | Restrição | Descrição |
| 1 | Sistema Operacional | SO mínimo: Windows 7 |
| 2 | Conexão à internet | Conexão a todo momento é necessária |
| 3 | Navegador | Navegador obrigatório é Google Chrome |

# DESCRIÇÃO DE USUÁRIOS

|  |  |  |
| --- | --- | --- |
| Número | Usuários | Descrição |
| 1 | Administrador | Acesso total ao sistema para cadastro inicial ou resolução de problemas |
| 2 | Gestor | Usuário do sistema que irá efetuar cadastro de usuários de sua respectiva área |

# **MODELAGEM DO SISTEMA**

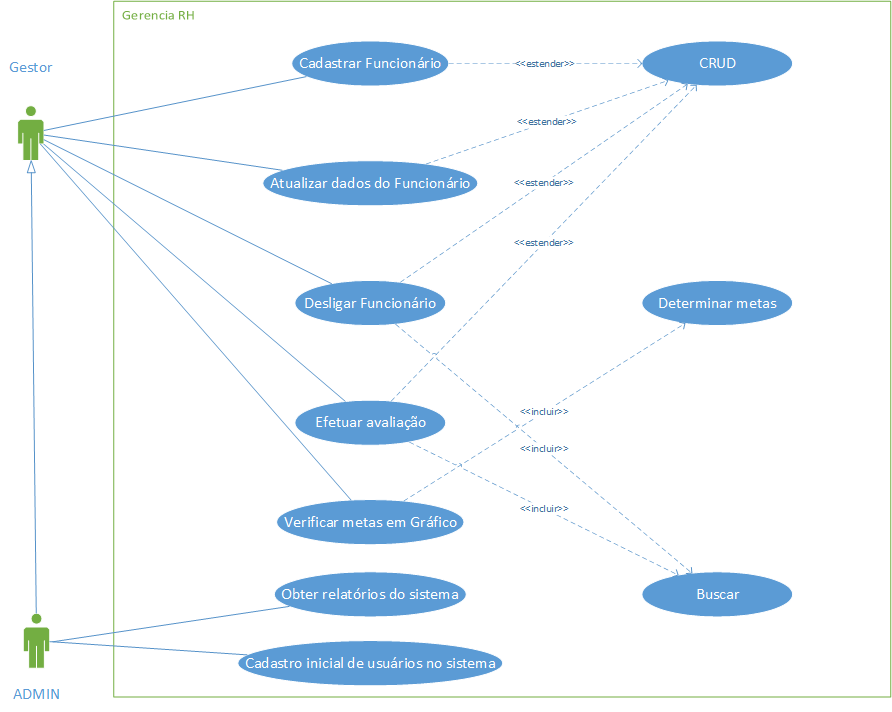


Figura 23 - Casos de uso

O fluxo perfeito do sistema será baseado da seguinte forma: Usuário gestor efetua login, cadastra seus funcionários conforme dados obtidos de algum ambiente da empresa, e ao acessar a tela home, terá acesso a qualquer fluxo desejado. Se desejar alterar informações, basta clicar no ícone. Se desejar desligar algum funcionário, será executado da mesma forma. E assim por diante. Metas serão feitas futuramente no software a partir dos dados contidos nos gráficos (somatórias das notas A, B, C) por time

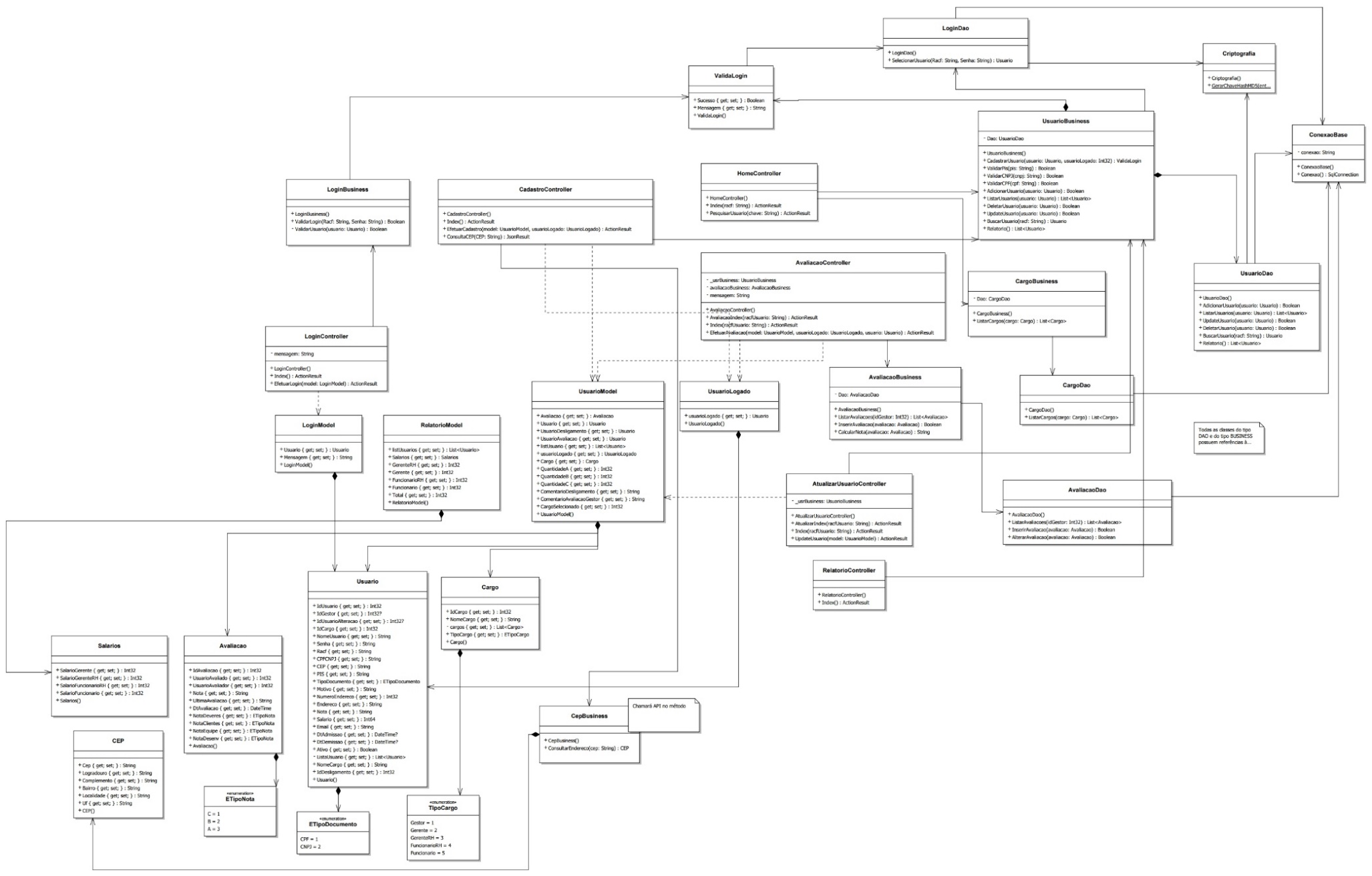


Figura 24 - Diagrama UML de classes

# PLANOS DE TESTE

**Caso de uso:** Teste de cadastro de funcionário

**Situação:** Inclusão do usuário

**Teste 1:** Todos os campos devidamente inseridos

**Teste bem-sucedido:** Guardar na base as informações do usuário, retornar usuário para homepage com aviso: “Usuário cadastrado com sucesso”

**Teste malsucedido:** Retornará aviso ao usuário, informando com mensagem: “Erro ao gravar na base, entre em contato com um administrador”.

**Teste 2:** Campos parcialmente inseridos

**Teste bem-sucedido:** Retornará ao usuário a mensagem de validação: “Nem todos os campos foram preenchidos corretamente”, não permitindo armazenamento na base

**Teste malsucedido:** Retornará ao usuário uma mensagem especificando o erro.

**Caso de uso:** Atualizar dados do funcionário

**Situação:** Alteração do usuário

**Teste 1:** Campos devidamente preenchidos com alterações

**Teste bem-sucedido:** Guardará informações na base, avisará usuário com mensagem: “Usuário atualizado com sucesso”

**Teste malsucedido:** Manterá os campos preenchidos, e avisará o usuário com a mensagem: “Erro ao atualizar usuário na base”

**Teste 2:** Esvaziar um campo

**Teste bem-sucedido:** Informará ao usuário que há falta de preenchimento de campos

**Teste malsucedido:** Retornará ao usuário uma mensagem especificando o erro.

**Caso de uso:** Desligar funcionário

**Situação:** Desligando funcionário no sistema

**Teste 1:** Selecionando e desligando usuário com campos preenchidos

**Teste bem-sucedido:** Guardará a exclusão de forma lógica com um campo booleano na tabela, informará o usuário com a mensagem de exclusão bem-sucedida

**Teste malsucedido:** Informará o usuário que não pôde ser feito o desligamento

**Teste 2:** Desligando funcionário com motivo vazio

**Teste bem-sucedido:** Informará ao usuário para preencher o campo de motivo com um aviso

**Teste malsucedido:** Informará ao usuário para preencher o campo de motivo com um aviso

**Caso de uso:** Efetuar Avaliação

**Situação:** Avaliação do usuário no sistema

**Teste 1:** Inserindo avaliação com todos os campos devidamente preenchidos

**Teste bem-sucedido:** Guardará a informação informará o usuário com a mensagem de avaliação bem-sucedida

**Teste malsucedido:** Informará o usuário que não pôde ser feito a avaliação dentro do sistema

**Teste 2:** Avaliando funcionário com campos vazios

**Teste bem-sucedido:** Informará ao usuário para preencher os campos

**Teste malsucedido:** Informará que ocorreu algum erro durante a avaliação do colaborador e armazenará em log

**Caso de uso:** Verificar metas em gráfico

**Situação:** Conseguir verificar a somatória das notas na homepage

**Teste 1:** Entrando na homepage para total de notas

**Teste bem-sucedido:** Exibirá para o usuário um dashboard completo contendo as informações dos últimos conjuntos de notas concedidas

**Teste malsucedido:** Informará ao usuário que não pôde ser feito a busca das informações na base de dados.

**Teste 2:** Conseguir pegar as informações de salários na tela de relatório

**Teste bem-sucedido:** Exibirá para o usuário um gráfico completo, informando a junção dos salários dentro da base

**Teste malsucedido:** Informará ao usuário que ocorreu um erro ao buscar as informações

**Caso de uso:** Obter relatório do sistema

**Situação:** Obter relatório do sistema

**Teste 1:** Verificar relatório de sucessão de eventos dentro do sistema

**Teste bem-sucedido:** Mostrará para o admin a visualização das ações efetuadas pela quantidade X de usuários do sistema

**Teste malsucedido:** Informará ao admin que não foi possível efetuar a busca dos últimos dados

**Caso de uso:** Cadastro inicial de usuário no sistema

**Situação:** Cadastrar primeiro usuário

**Teste 1:** Cadastro inicial de usuário pelo sistema

**Teste bem-sucedido:** Informará ao admin que o cadastro foi efetuado com sucesso dentro do sistema

**Teste malsucedido:** Informará ao admin que não foi possível efetuar o cadastro

**Teste 2:** Cadastro inicial de usuário pelo sistema com campos vazios

**Teste bem-sucedido:** Informará ao admin que o cadastro não pode ser feito com campos obrigatórios vazios

**Teste malsucedido:** Informará ao admin que não foi possível efetuar o cadastro devido a algum erro na base

# **DISCUSSÃO**

Neste capítulo será apresentada a discussão sobre o que foi apresentado durante o projeto e se os objetivos específicos foram atingidos conforme o planejamento.

# Framework para acesso à camada de dados

O fluxo geral do sistema em questões de camadas é View – Controller – Business – Data Access Object – Captura de dados. Conforme foi demonstrado no projeto, ao utilizar o Dapper e atribuir os conceitos do mesmo ao projeto, foi possível efetuar uma fácil execução de qualquer query que o programa pedisse, fosse em forma de string ou em forma de Procedure, para manter a segurança da base caso necessário, sendo suficiente abrir uma conexão com a base e utilizar SqlConnection.Execute ou SqlConnection.Query, alcançando o objetivo de utilizar um framework para acesso à camada de dados, aumentando a segurança.

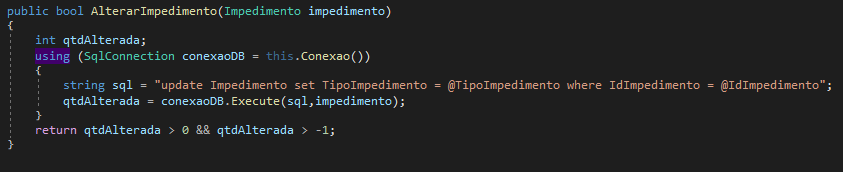


Figura 25 - Exemplo de implementação do Dapper

# Segregação de responsabilidade das classes

Com a separação do projeto em camadas, foi possível manter cada classe das respectivas camadas responsável por uma etapa de manuseio de dados. Por exemplo, as classes contidas em Business, mantiveram dentro de si apenas regras de negócios pertinentes àquele fluxo. Já as classes contidas dentro de Data Access Layer, mantiveram seu propósito único de efetuar a conexão com a base de dados e executar o que fosse demandado pelo código. Ao manter essa arquitetura, o objetivo de segregar a responsabilidade foi alcançado, mantendo a facilidade de implementar futuras alterações pertinentes à nova função.

# Alta robustez do programa

Como o sistema é web, é acessível por qualquer sistema operacional ou navegador que tenha acesso à rede, tornando sua interoperabilidade alta. Como o sistema possui aplicação de orientação a objeto e segregação de responsabilidade das classes e dos projetos, qualquer outra manutenção que tenha de ser feita será efetuada de forma rápida e eficaz, tornando também sua manutenibilidade alta, gerando, por consequência, um programa robusto, alcançando o objetivo proposto inicialmente

# CONCLUSÃO

O trabalho acima efetuou a descrição, passo a passo, de como ocorreu o desenvolvimento de uma ferramenta web baseada em ASP.NET MVC.

Ao desenvolver um sistema com separação de camadas, é possível afirmar que novas ferramentas e novos algoritmos podem ser adicionados ao sistema ou melhorar aquilo que já é possuído. O objetivo específico era alcançar uma ferramenta robusta, que usufruísse de frameworks para acesso à camada de dados, permitindo maior escalabilidade e segurança, e fornecendo uma segregação de responsabilidade de classes para maior manutenibilidade e interoperabilidade.

Dadas as implementações do sistema e de como foi criado, o objetivo proposto foi alcançado. Além do sistema possuir todos os objetivos propostos em específico, o mesmo gera, no âmbito de regras de negócio, capacidade para o gestor efetuar avaliações conforme as atividades propostas a seus colaboradores, permitindo que exerça controle sobre a equipe, de uma forma que a junção das avaliações demonstrará ao gestor como a equipe em questão tem performado, dando um overview geral da equipe.

# **REFERÊNCIAS BIBLIOGRÁFICAS**

Heuser (2008) - HEUSER, Carlos Alberto. **Projeto de Banco de Dados**. 4. ed. atual. [*S. l.*]: Bookman, 2008. 282 p. v. 4. ISBN 978-8577803828.

(AUMASSON, Jean-Philippe, 2018) - AUMASSON, Jean-Philippe. **SERIOUS CRYPTOGRAPHY**: A Practical Introduction to Modern Encryption. [*S. l.*]: No Starch Pres, 2018. 434 p. ISBN 978-1-59327-826-7.

(BARRON,2018) - 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

Burbeck (1992) - BURBECK, Steve. **Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC)**. 1992. Disponível em: https://web.archive.org/web/20120729161926/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html. Acesso em: 07 nov 2019

Castelló (2016) - CASTELLÓ, Thiago; VAZ, Verônica. **Tipos De Criptografia**. 2016 Disponível em: http://www.gta.ufrj.br/grad/07\_1/ass-dig/index.html. Acesso em: 02 nov. 2019.

Chen (2003) - CHEN, Shu-Ching *et al*. **A Three-Tier System Architecture Design and Development for Hurricane Occurrence Simulation**, [*s. l.*], v. 1, ed. 1, p. 1-7, 2003. Disponível em: https://pdfs.semanticscholar.org/5a99/d4cdc10988ba2c7a920d1ac2d06b9cdc3eef.pdf. Acesso em: 7 dez. 2019.

(DESCONHECIDO, 2017) - DESCONHECIDO. **What is Dapper? How to use Dapper in Asp.NET MVC?** 2017. Disponível em: http://www.codedigest.com/quick-start/17/what-is-dapper-how-to-use-dapper-in-aspnet-mvc. Acesso em: 07 dez. 2019

Fielding (2000, p.18) - FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. 180 p. Dissertação (Doutorado de Filosofia em informações e ciências da computação) - Universidade da Califórnia, Irvine, 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\_dissertation.pdf. Acesso em: 7 dez. 2019.

(FREEMAN, Adam. 2013) - FREEMAN, Adam. **Pro ASP.NET MVC 5**. 5. ed. atual. [*S. l.*]: Apress, 2013. 832 p. ISBN 978-1-4302-6530-6.

Majeed (2018, p.1) - MAJEED, Abdul; RAUL, Ibtisam. **MVC Architecture: A Detailed Insight to the Modern Web Applications Development.** t. Peer Rev J Sol Photoen Sys, 2018. Disponível em: https://crimsonpublishers.com/prsp/pdf/PRSP.000505.pdf. Acesso em: 06 nov 2019.

Moreno (2005) - MORENO, Edward David; PEREIRA, Fábio Dacêncio; CHIARAMONTE, Rodolfo Barros. **Criptografia em Software e Hardware**. 1. ed. [*S. l.*]: Novatec, 2005. 23 p. v. 1.

(OCAMPO, Joe, 2011) - OCAMPO, Joe *et al*. **Pablo‘s SOLID Software Development**. [*S. l.*: *s. n.*], 2009. 80 p. v. 1. 80 p.

Scudero (2016) - SCUDERO, Erick. **TOP 10 Principais SGBDs do mercado global.** Disponível em https://becode.com.br/principais-sgbds/. Acesso em: 09 nov 2019

(TAKAI, ITALIANO, FERREIRA, 2005) - TAKAI, Osvaldo; ITALIANO, Isabel; FERREIRA, Eduardo. **Introdução a banco de dados.** 2005. Disponível em: https://www.ime.usp.br/~jef/apostila.pdf. Acesso em: 09 nov. 2019

(Zaccanini, 2010) - 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

# **BIBLIOGRAFIA**

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 nov. 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

DESCONHECIDO. **Criptografia**. 2012.Disponível em: http://cartilha.cert.br/criptografia/. Acesso em: 01 nov. 2019.

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

FOWLER, Martin. **Presentation Domain Data Layering,** 2015. Disponível em: https://martinfowler.com/bliki/PresentationDomainDataLayering.html. Acesso em: 7 dez. 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.

KESSLER, Gary C. **An Overview of Cryptography**. 1999. Disponível em: http://www.garykessler.net/library/crypto.html#password. Acesso em: 04 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

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

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

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. atual. [*S. l.*]: Apress, 2010. 790 p. ISBN 978-0-13-703515-1.

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.