SENAI – SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL
UNIDADE PINHAIS
CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

MIBI - MICROBUSINESS

PINHAIS
2020

RUAN PABLO ALVES DA SILVA

MIBI - MICROBUSINES

TCC - Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Desenvolvimento de Sistemas do SENAI – Serviço Nacional de Aprendizagem Industrial, Unidade Pinhais como requisito para a composição da segunda nota da referida disciplina, sob orientação dos professores Tiago Adelino Navarro e Rosanete Grassiani dos Santos.

PINHAIS

2020

RESUMO

Este documento compõe-se da parte explicativa do projeto MIBI – MicroBusiness, que consiste em um ambiente virtual onde micro e pequenas empresas podem divulgar seus serviços e produtos para os consumidores do Brasil inteiro, porém mais focado em incentivar a economia local das regiões onde estas empresas em potencial residem. O documento terá como objetivo especificar o tema do projeto e seus requisitos, os seus diagramas e documentações de determinados diagramas, explicação das características da programação, principalmente da Programação Orientada a Objetos e descrever os padrões de projetos GoF e Grasp de forma direta e clara. O método de pesquisa utilizado foi qualitativa, descritiva e exploratória. Ao final de tudo, foi obtido grande conhecimento e técnicas práticas de desenvolvimento de sistemas, além de comprovar a viabilidade do projeto seguir a diante.

Palavras chave: MicroBusiness; Programação Orientada a Objetos; *Padrões de Projetos;*

SUMÁRIO

DIAGRAMAS

1 INTRODUÇÃO

Dentre o avanço tecnológico, muitas empresas novas elaboram seus produtos e serviços de forma criativa e inovadora para satisfazer o gosto atual do consumidor. Mas mesmo assim, muitas delas não ganham tanta visibilidade quanto deveriam, mesmo tendo capacidades para se tornarem grandes empresas futuramente.

Até as ideias mais brilhantes podem ser ofuscadas pela falta de interesse que o Brasil tem de aprender ou desfrutar de coisas novas. E isso ocorre desde no ambiente familiar até no mercado de trabalho, contando também para pequenas e microempresas que não são tão procuradas pelos consumidores que só cobiçam dos maiores, sem pensar no comércio do seu bairro que tem produtos ou serviços iguais ou melhores, mesmo com menos capacidade de investimento.

Este trabalho, terá como foco inicial uma solução para este problema de visibilidade que as empresas ainda pequenas sofrem, e seguidamente com as metodologias estudadas para adquirir capacidade de fazê-lo viável.

2 MIBI – MICROBUSINESS

De acordo com a notícia publicada em Suno Notícias, o Instituto Brasileiro de Geografia e Estatística (IBGE) realizou um estudo em 2017 denominado Demografia das Empresas e Estatísticas de Empreendedorismo, foi ressaltado que o Brasil fecha mais empresas do que abrem a cada ano. Isto acontecia em anos anteriores, porém o número de empresas que se tornaram inativas — que no caso, fala de 699 mil — é 0,5% menor que o ano anterior, assim destacando uma queda de 23 mil empresas líquidas. Além disso, IBGE também aponta que metade das empresas param suas atividades antes de completar quatro anos de portas abertas.

Há vários fatores que acabam acarretando ao aumento destes números, e um deles é a falta de visibilidades de pequenas e microempresas com grande potencial de crescimento, ainda mais em tempos onde pessoas devem evitar sair de suas residências e no aumento de desemprego por conta da crise que vivemos. Como seria então o futuro dos donos destas empresas? A frustração de um sonho não ser realizado? O desespero de não talvez não ter seu ganha pão amanhã?

Por isso a ideia do projeto MIBI apareceu. Seu objetivo é justamente dar em um ambiente virtual aos cidadãos de todo o país a oportunidade de consumirem produtos e serviços excelentes de empreendedores que ainda não tiveram a chance de crescer. As micro e pequenas empresas — e somente elas — poderão divulgar seus produtos e serviços na plataforma, assim as pessoas podem visualizá-las e entrar em contato com a própria empresa se tiverem interesse em seus produtos e/ou serviços. Assim, tanto a economia nacional e principalmente a local seria incentivada, estimulando micro e pequenas empresas a se tornarem grandes um dia e os seus clientes receberem produtos e serviços de excelente qualidade. Ou seja, todos saem ganhando.

2.1 REQUISITOS DO PROJETO

Existem três tipos de requisitos: os requisitos funcionais, não funcionais e as regras de negócio. Resumidamente, elas estarão levantadas a seguir.

2.1.1 Requisitos Funcionais

Requisitos Funcionais são todas as funcionalidades que o sistema deverá ter, e os da MIBI serão estes:

- Deve ter como pesquisar empresas, produtos e serviços;
- Deve ser possível realizar cadastro de usuário;
- Deve ser possível realizar a entrada de um usuário cadastrado no sistema web;
- Deve ser possível o usuário alterar suas informações cadastradas;

- Deve ser possível o usuário cadastrar sua empresa no sistema, se ainda não tenha feito;
- Deve ser possível o usuário alterar as informações da empresa que cadastrou;
- Deve ser possível o usuário deletar a sua empresa cadastrada;
- Deve ser possível o usuário deletar sua conta;
- Se ele tiver com uma empresa cadastrada e deletar os seu cadastro de usuário, o cadastro da empresa também será deletada do sistema;

2.1.2 Requisitos não funcionais e regras de negócio

Requisitos não funcionais são aqueles que se referem às características do sistema e não necessariamente as coisas que ele irá realizar, como restrições, validações, segurança e entre outros.

Regras de negócio também podem ser consideradas um tipo de requisitos não funcionais, pois se refere a normas/diretrizes que o sistema deve seguir. Os requisitos não funcionais e as regras de negócio do projeto estarão levantadas a seguir:

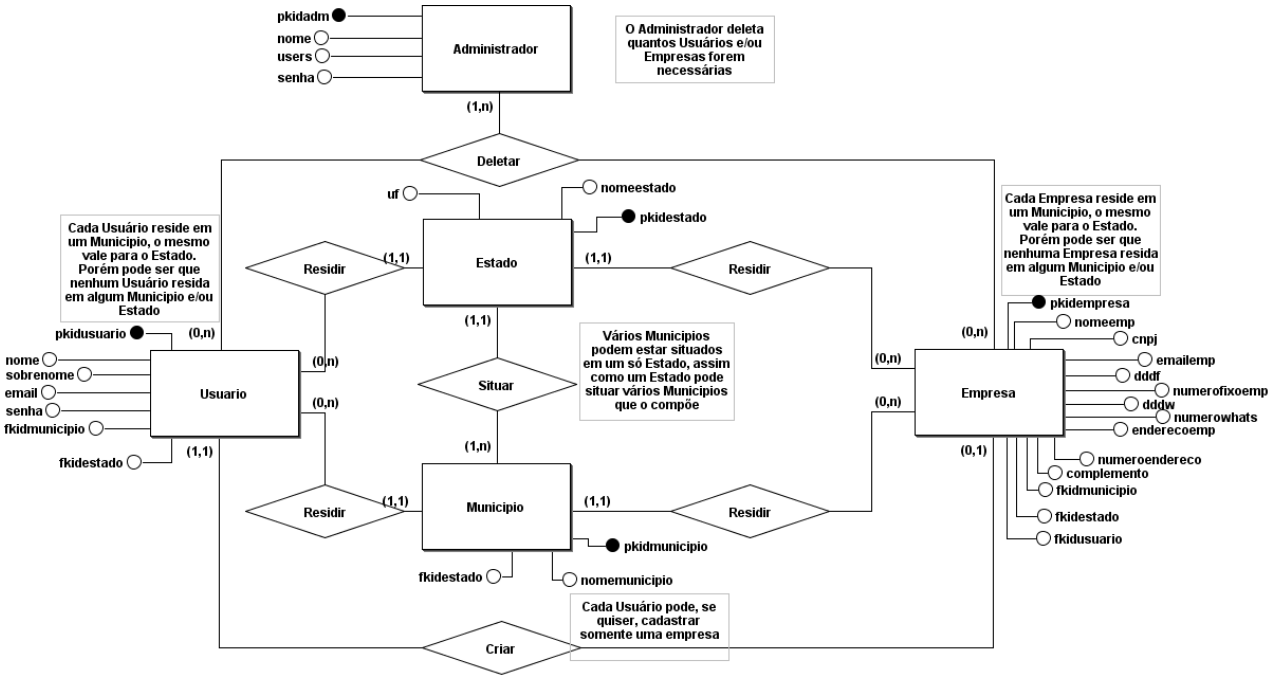
- Os botões, em geral, devem ter a cor puxada para turquesa e letras maiúsculas em negrito de cor branca;
- Qualquer pessoa que acessar o site podem pesquisar empresas, produtos e serviços;
- Somente usuários que efetuaram o Login poderão entrar em contato com as empresas;
- Os botões onde redireciona a página para a tela de cadastro de usuário e para efetuar o Login devem estar no canto superior direito da tela principal;
- Após o Login, o botão que redireciona a página para a tela do usuário e o que faz o logout, ficarão no canto superior direito da tela principal;
- Na Tela do usuário, deverá conter as opções de editar e deletar os dados cadastrados (tanto do usuário quanto da empresa, e caso a empresa não tenha sido criada, deve haver a opção de cadastrá-la);
- Somente empresas com CNPJ válido poderão ser cadastradas;
- Somente pequenas e microempresas poderão ser cadastradas e se manterem no cadastro;
- Deverá ser deletado aquela empresa que não cumprir os termos de uso, ou dependendo da gravidade até mesmo a conta de usuário.

3 DIAGRAMAS

3.1 MODELAGEM DO BANCO DE DADOS

3.1.1 Modelo Conceitual

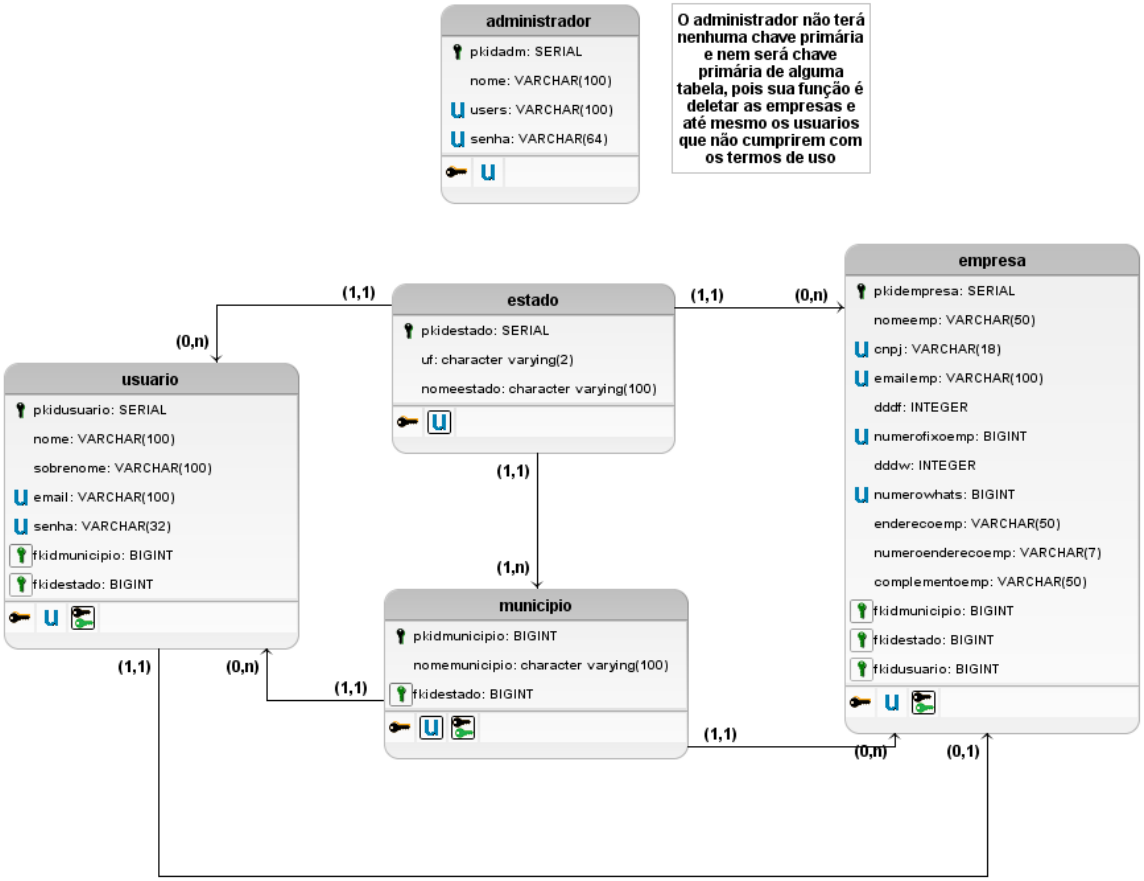
O modelo conceitual mostra as entidades, os atributos de cada entidade, e também o relacionamento entre elas:



Modelo Conceitual 1

3.1.2 Modelo Lógico

O modelo lógico é criado utilizando como referência o modelo conceitual. Ele define os atributos mais precisamente, colocando os tipos dos atributos (se receberá números, datas, todos os tipos de caracteres, entre outros), quais são as chaves primárias (o identificador de cada linha de dados inserida na tabela) e estrangeiras (o identificador de uma tabela inserida em outra, assim realizando uma relação entre elas) e entre outras limitações que devem se levar em conta:



Modelo Lógico 1

3.1.3 Modelo Físico

O modelo físico já é a criação das tabelas, adaptano e respeitando as limitações do SGBD que foi escolhido (neste caso, foi escolhido o PostgreSQL) e deve ser produzido de acordo com o modelo lógico.

Para as tabelas estado e municipio, foi utilizado scripts gerados diretamente da API do IBGE. Disponível em: https://ibge-sql.herokuapp.com. Por fim, ficou desta forma:

```
1
2   /*FOI UTILIZADO A API DO IBGE PARA A CRIAÇÃO E INSERÇÃO
3     DOS DADOS DAS TABELAS ESTADO E MUNICIPIO*/
4
5   --Criando tabela estado
6   CREATE TABLE IF NOT EXISTS estado (
7                   pkidestado BIGINT  PRIMARY KEY,
8                   uf VARCHAR(2) NOT NULL,
9                   nomeestado VARCHAR(100) NOT NULL
10              );
11  INSERT INTO Estado VALUES (11, 'RO', 'Rondônia');
12  INSERT INTO Estado VALUES (12, 'AC', 'Acre');
13  INSERT INTO Estado VALUES (13, 'AM', 'Amazonas');
14
15  --              ...Inserindo todos os Estados do Brasil...
16
17  --Criando tabela municipio
18  CREATE TABLE IF NOT EXISTS municipio (
19                  pkidmunicipio BIGINT  PRIMARY KEY,
20                  nomemunicipio VARCHAR(100) NOT NULL,
21                  fkidestado BIGINT NOT NULL,
22                    CONSTRAINT fkidestado
23                      FOREIGN KEY (fkidestado) REFERENCES Estado(pkidestado)
24              );
25  INSERT INTO Municipio VALUES (1100015, 'Alta Floresta D''Oeste', 11);
26  INSERT INTO Municipio VALUES (1100023, 'Ariquemes', 11);
27  INSERT INTO Municipio VALUES (1100031, 'Cabixi', 11);
28
29  --              ...Inserindo todos os Municipios de cada Estado...
30
```

Modelo Físico 1

```
27  INSERT INTO Municipio VALUES (1100031, 'Cabixi', 11);
28
29  --              ...Inserindo todos os Municipios de cada Estado...
30
31  --Criando tabela usuario
32  CREATE TABLE usuario (
33      pkidusuario SERIAL PRIMARY KEY,
34      nome VARCHAR(100),
35      sobrenome VARCHAR(100),
36      email VARCHAR(100),
37      senha VARCHAR(32),
38      fkidmunicipio BIGINT REFERENCES municipio(pkidmunicipio),
39      fkidestado BIGINT REFERENCES estado(pkidestado),
40      UNIQUE (pkidusuario, email, senha)
41  );
42
43  --Criando tabela empresa
44  CREATE TABLE empresa (
45      pkidempresa SERIAL PRIMARY KEY,
46      nomeemp VARCHAR(50),
47      cnpj VARCHAR(18),
48      emailemp VARCHAR(100),
49      dddf INTEGER,
50      numerofixoemp BIGINT,
51      dddw INTEGER,
52      numerowhats BIGINT,
53      enderecoemp VARCHAR(50),
54      numeroenderecoemp VARCHAR(7),
55      complementoemp VARCHAR(50),
56      fkidmunicipio BIGINT REFERENCES municipio(pkidmunicipio),
57      fkidestado BIGINT REFERENCES estado(pkidestado),
58      fkidusuario BIGINT REFERENCES (pkidusuario),
59      UNIQUE (pkidempresa, cnpj, numerowhats, emailemp, numerofixoemp)
60  );
61
```

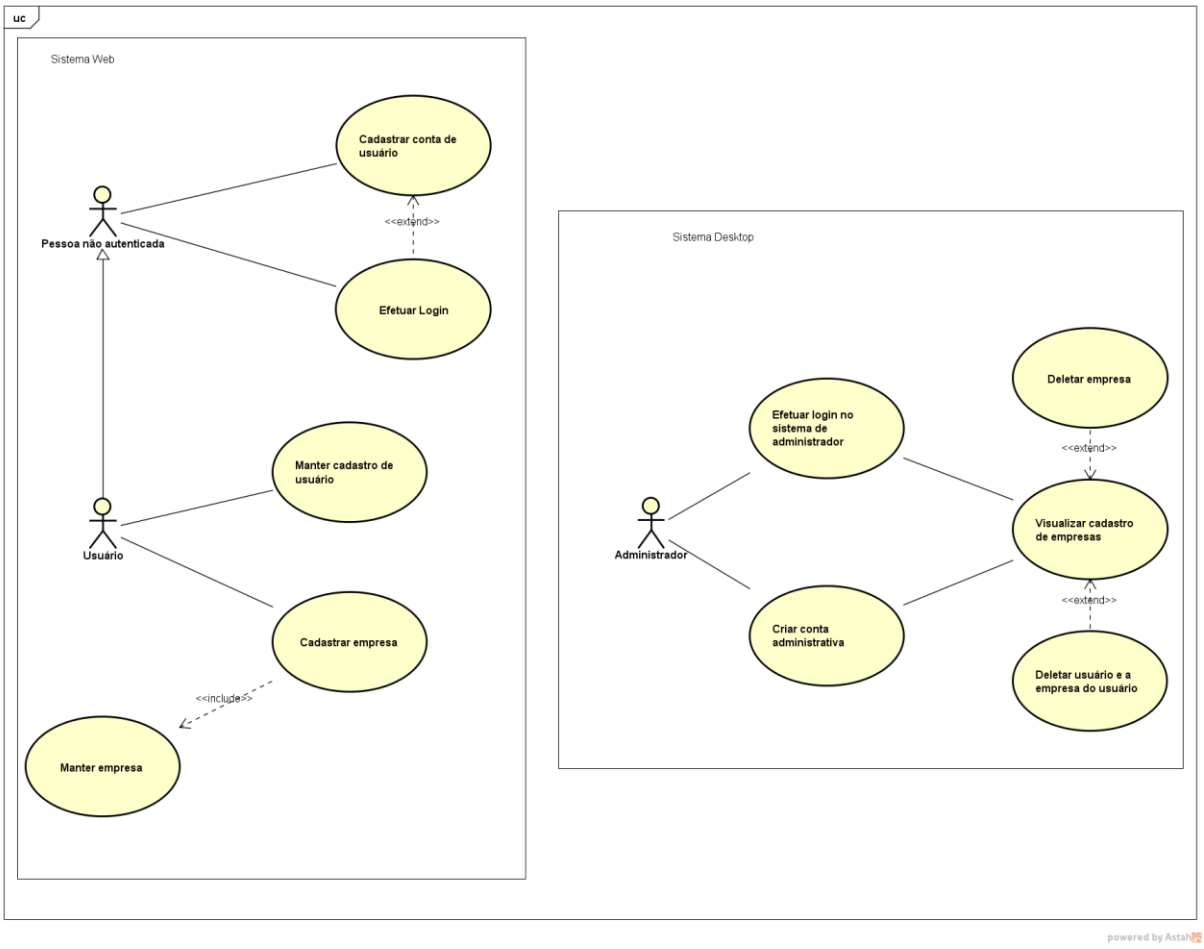
Modelo Físico 3

```
59      UNIQUE (pkidempresa, cnpj, numerowhats, emailemp, numerofixoemp)
60  );
61
62  --Criando tabela administrador
63  CREATE TABLE administradora (
64      pkidadm SERIAL PRIMARY KEY,
65      nome VARCHAR(100),
66      users VARCHAR(100),
67      senha VARCHAR(64),
68      UNIQUE (pkidadm, users, senha)
69  );
70
71
72
73
```

Modelo Físico 2

3.2 CASOS DE USO

O diagrama de casos de uso descreve todas as funcionalidades que os usuários irão utilizar e facilita o levantamento de requisitos funcionais do sistema.



Casos de uso 1

3.3 DIAGRAMA DE CLASSES

O Diagrama de Classes é a essência da UML, é como se fosse um modelo lógico das classes criadas no projeto. De forma resumida, foi feito o Diagrama de Classes referindo-se à ação que o Administrador tem com as classes Usuario e Empresa:

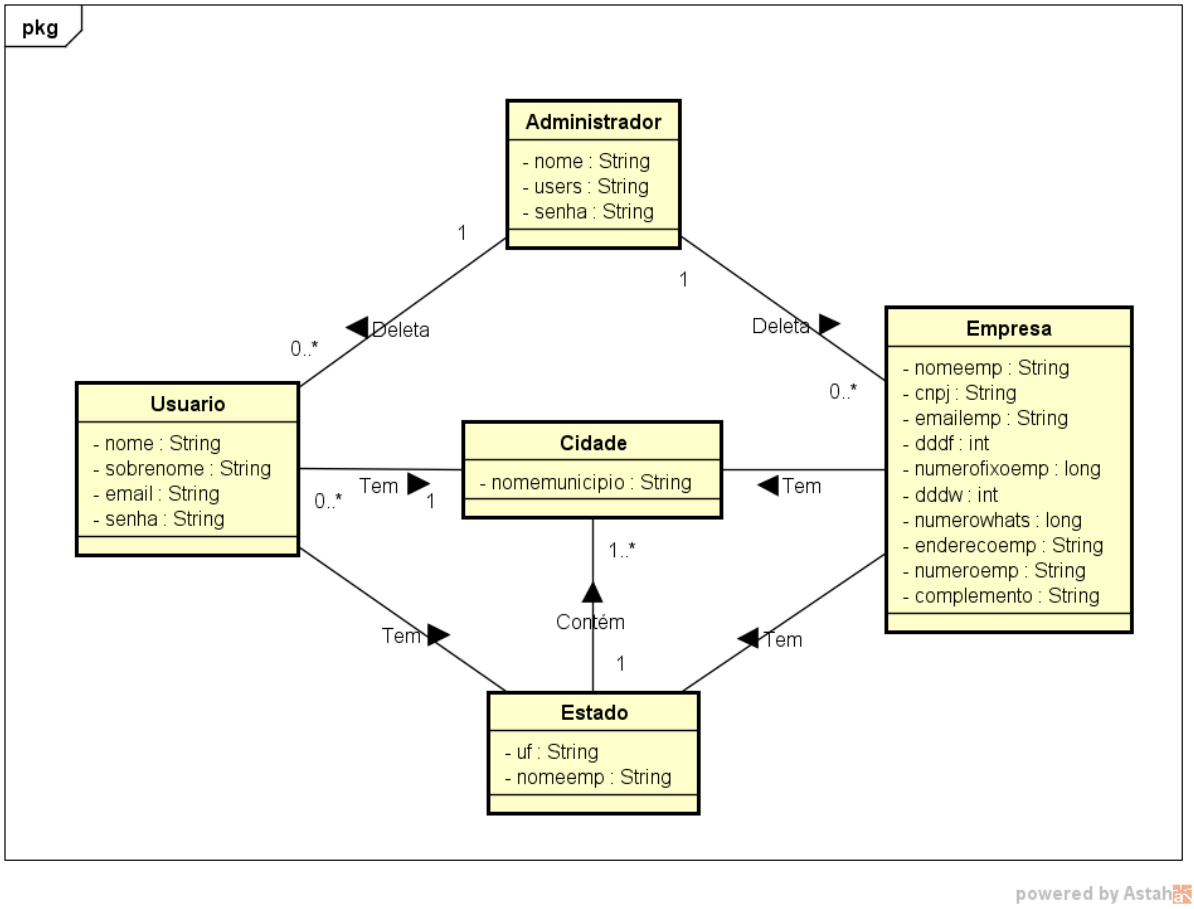


Diagrama de Classes 1

3.4 DIAGRAMA DE SEQUÊNCIA

Este é um exemplo de diagrama de sequência feito com o login do administrador do projeto. Todos os diagramas estarão em anexo para melhor visualização.

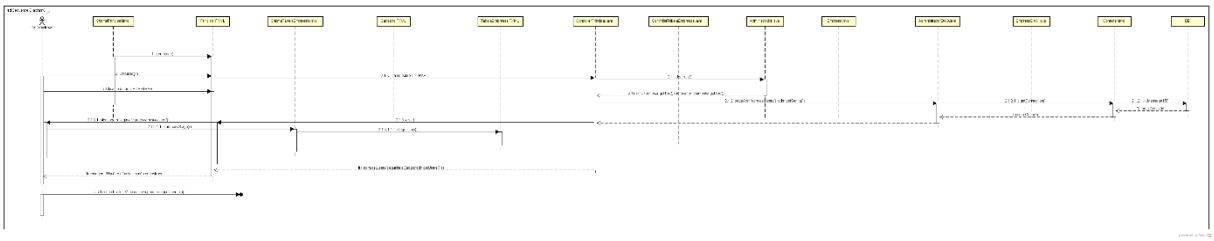


Diagrama de Sequência 1

4 CONCEITOS DA PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

4.1 MÉTODO

Método é como se fosse uma função e que é associada a um objeto. Será mostrado um logo a seguir.

4.2 CLASSES E OBJETOS

Vamos dizer que você comprou uma xícara azul pastel com uma estampa bacana de rosas. Este é um objeto seu agora, correto? Mas onde você comprou esta

xícara tinha vários outros tipos de xícaras, haviam térmicas, pretas com uma estampa de caveira e etc. Então, resumidamente, você obteve um uma xícara, um *objeto*, que estava à venda a uma loja de xícaras, que mesmo sendo diferentes continuavam sendo *classificadas* como xícaras. É correto dizer então que seu objeto pode e deve ser classificado como uma xícara, então, ela pertence a uma classe, e que a sua xícara nada mais é do que uma instância dessa classe chamada “xícara”.

Agora, partindo para a programação, imagine que essa loja tenha um sistema de gerenciamento das vendas, e que nesse sistema ela tem a classe xícara:

```
public class Xicara{
 String cor;
 String estampa;
 Double preco;
}
```

Então, quando ela estava cadastrando que xícaras novas, criou objetos assim:

```
public class executa{
//Isto abaixo é um método!
 public static void main(String[ ] args){
   Xicara xicara = new Xicara( );
   xicara.cor = "azul pastel";
   xicara.estampa = "rosas";
   xicara.preco = 35.50;

   Xicara outraXicara = new Xicara( );
   outraXicara.cor = "preto";
   outraXicara.estampa = "caveira";
   outraXicara.preco = 24.99;
 }
}
```

4.3 COMPOSIÇÃO

É quando a estrutura de uma classe é composta por outras classes, por exemplo, existe a classe Poupanca que tem a variável Double saldo, e também tem a ContaCorrente com variável Double saldo. E a classe Banco é composta por essas duas classes:

```
public class Poupanca{
 Double saldo;
}
public class ContaCorrente{
 Double saldo;
}


public class Banco{
  Poupanca[ ] p;
 ContaCorrente[ ] cc;
 int numCorrente, numPoupanca;
 void inicia(){
   p = new Poupanca[100];
   cc = new ContaCorrente[100];
   numCorrente = 1;
   numPoupanca = 1;
 }
 void abreCorrente(){
   c[numCorrente] = new ContaCorrente( );
 }
 void abrePoupanca(){
   c[numPoupanca] = new Poupanca( );
 }
}
```

4.4 AGREGAÇÃO

Bem semelhante à composição, a agregação é quando uma classe utiliza outras classes em seu código, mas não sendo uma parte dela mesma como a composição:

```
public class Banco{
  Poupanca p;
  ContaCorrente cc;
  Double cofre;
 void abreCofre(){
    cofre = p.saldo;
    cofre += cc.saldo;
  }
}
```

4.5 ENCAPSULAMENTO

Encapsulamento é utilizado em classes para conservar as variáveis essenciais para o correto funcionamento da classe, não deixando com que qualquer usuário que esteja usando o sistema acabe manipulando estas variáveis de uma forma que até mesmo todo o software tenha o risco de parar de funcionar. Nisto então cria-se para cada variável dois métodos diferentes para ela, denominado getters e setters. Os setters recebem valores novos para a variável, enquanto os getters dão o retorno:

```
public class Banco{
  Poupanca p;
  ContaCorrente cc;
  Double cofre;

  public Double getCofre(){
    return cofre;
  }

  public void setCofre(Double cofre){
    this.cofre = cofre;
  }
```

```
 void abreCofre(){
    setCofre(p.saldo);
    setCofre(getCofre( ) + cc.saldo);
    System.out.println(Dinheiro no cofre: + getCofre( ));
//retorna o valor do p e cc somado
  }
}
```

4.6 HERANÇA (EXTENDS E INTERFACE)

Herança com extends consiste em uma classe “mãe”, que a classe “filho” recebe tudo que a classe “mãe” tem em sua estrutura, além da própria estrutura. Aqui vai um exemplo literal:

```
public class Filho extends Mae {
 String cabelo;
   public Filho(OlhosAzuis olhosAzuis) {
      cabelo = "ondulado";
      // chama o construtor da classe mãe, ou seja, da classe "Mae"
      super(cabelo, olhosAzuis);
   }
}
```

Agora, o tipo de herança que se realiza com interface é utilizado quando duas ou mais classes utilizam do mesmo método. Um exemplo encontrado no site https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos:

```
public interface Automovel {
   void acelerar();
   void frear();
   void acenderFarol();
}
```

```
public class Carro implements Automovel {

  /*...*/

  @Override
  public void acelerar() {
    this.mecanismoAceleracao.acelerar();
  }

  @Override
  public void frear() {
    /* código do carro para frear */
  }

  @Override
  public void acenderFarol() {
    /* código do carro para acender o farol */
  }

  /*...*/
}

public class Moto implements Automovel {

  /*...*/

  @Override
  public void acelerar() {
    /* código específico da moto para acelerar */
  }
```

```
  @Override
  public void frear() {
    /* código específico da moto para frear */
  }

  @Override
  public void acenderFarol() {
    /* código específico da moto para acender o farol */
  }

  /*...*/
}
```

4.7 POLIMORFISMO

A palavra "polimorfismo" vem do grego *poli* = muitas, *morphos* = forma. Na programação, significa que um mesmo método usado por dois objetos diferentes, uma de cada classe diferente, é implementado de formas diferentes. Outro exemplo retirado do mesmo link:

```
public class Main {
  public static void main(String[] args) {
    Automovel moto = new Moto("Yamaha XPTO"
     +" - 100", new MecanismoDeAceleracaoDeMotos());
    Automovel carro = new Carro("Honda Fit",
        new MecanismoDeAceleracaoDeCarros());
    List < Automovel > listaAutomoveis = Arrays.asList(moto, carro);
    for (Automovel automovel : listaAutomoveis) {
      automovel.acelerar();
      automovel.acenderFarol();
    }
  }
}
```

5 PADRÕES DE PROJETOS (GOF E GRASP)

5.1 GOF

Os Padõres GoF são os mais conhecidos e praticados no mercado, ele tem como o objetivo principal o reaproveitamento de código nos projetos e agrupa seus

Padrões de Projetos em três tipos diferentes: Creational, Structural e Behavioral (Criação, Estrutura e Comportamental).

5.1.1 Criacional Design Patterns

- Factory mode (modelo de fábrica): consiste em fornecer uma interface para criar objetos em uma superclasse, mas permite que as subclasses alterem o tipo de objeto que será instanciado;
- Abstract Factory (fábrica abstrata): consiste em produzir famílias de objetos relacionados sem especificar suas classes concretas;
- Builder (construtor): consiste em construir objetos complexos passo a passo, para que daí o objeto seja representado de outras formas utilizando o mesmo método;
- Prototype: consiste em permitir copiar objetos existentes sem fazer com que seu código dependa de suas classes.

5.1.2 Structural Design Patterns

- Bridge (ponte): consiste em dividir uma classe muito extensa em várias minunciosamente menores relacionadas em duas hierarquias separadas, sendo uma delas abstração e outra implementação, que podem ser desenvolvidas independentemente;
- Composite (composto): consiste em permitir a composição de objetos em estruturas de árvores e trabalhando com elas como se fossem objetos individuais;
- Decorator (decorador): consiste em permitir anexos novos de comportamentos aos objetos, fazendo um envoltório que contêm esses comportamentos, criando objetos especiais desse envoltório e adicionando os objetos comum dentro desses especiais;
- Facade (fachada): consiste em fornecer uma interface simplificada para utilizar em qualquer tipo de conjunto complexo de classes, como bibliotecas;
- Flyweight: consiste em ajustar mais objetos à quantidade disponível de RAM, compartilhando partes comuns dos objetos entre vários objetos, ao invés de manter todos os dados em cada objeto;
- Proxy: consiste em fornecer um substituto ou espaço reservado para outro objeto, por conta de que o proxy controla o acesso ao objeto original, assim tendo controle no que fazer antes ou depois que a solicitação chegue ao principal objetivo, que no caso é o objeto.

5.1.3 Behavioral Design Patterns

- Chain of Responsibility (Cadeia de responsabilidade): consiste em passar solicitações ao longo de uma cadeia de manipuladores. Funcionaria quase como uma corrida de passar bastão, pois ao receber uma solicitação cada manipulador decide processar a solicitação ou passa-la para frente;
- Command (comando): consiste em transformar uma solicitação em um objeto independente, sendo ele o portador de todas as informações da sobre a solicitação. Essa transformação permite parametrizar métodos com diferentes solicitações, atrasar ou enfileirar a execução de uma solicitação e oferecer suporte a operações que podem ser desfeitas;
- Iterator (iterador): consiste em percorrer elementos de uma coleção sem expor sua representação subjacente (lista, pilha, árvore etc.);
- Mediator (mediador): consiste em dar a permissão de amenizar as dependências hostis para o sistema entre os projetos, ou seja, a comunicação entre os objetos é forçada a ser somente por meio de um mediador;
- Memento (lembrança): consiste em salvar um tipo de “ponto de restauração” e resetar o estado anterior de um objeto sem revelar os detalhes de sua implementação;
- State (estado): consiste em permitir que um objeto altere seu comportamento quando seu estado interno for alterado, como se o objeto mudasse de classe.
- Observer (observador): consiste em definir um mecanismo de assinatura para modificar vários objetos sobre qualquer tipo de evento que ocorra no objeto que estão observando;
- Strategy (estratégia): consiste em definir uma família de algoritmos, colocar cada um eles em uma classe separada e tornar seus objetos intercambiáveis;
- Template Method (método do modelo): consiste em definir a estrutura de um algoritmo da superclasse, mas permite que as subclasses substituam etapas específicas do algoritmo sem alterar sua real estrutura;
- Visitor (visitante): consiste em separar algoritmos dos objetos nos quais eles operam.

5.2 GRASP

Os Padrões de Projeto GRASP são consideravelmente diferentes que os demais conhecidos, eles não devem ser encarados como soluções pré-definidas para problemas específicos, mas sim compreendidos como princípios que ajudam os

desenvolvedores a projetar de uma forma bem estruturada aplicações orientadas a objetos.

Existem ao todo nove padrões GRASP, onde cinco são dos Princípios Fundamentais e quatro dos Princípios avançados.

5.2.1 Princípios Fundamentais

- Controller (controlador): consiste em atribuir a responsabilidade de lidar com os eventos do sistema de forma global, representando totalmente os casos de uso do sistema. Praticamente todos os eventos funcionais estão situados nele;
- Creator (criador): consiste em determinar qual classe deve ser responsável pela criação de certos objetos;
- Information Expert (especialista na informação): consiste em determinar quando se deve delegar a responsabilidade para um projeto que seja especialista naquela competência;
- Low Coupling (baixo acoplamento): consiste em determinar que as classes não devem depender de objetos concretos, mas sim de abstrações para permitir que ocorra mudanças sem impacto, evitando complicações no momento em que o usuário utilizar do software;
- High Cohesion (Alta Coesão): consiste em determinar que as classes devem se focar apenas na sua responsabilidade e nada mais que isso.

5.2.2 Princípios Avançados

- Indirection (indireção): consiste em ajudar a manter o baixo acoplamento, através da cedência de obrigações através de uma classe mediadora;
- Protected Variations (proteção contra variações): consiste em proteger o sistema contra a variação de componentes, encapsulando o comportamento que de fato é importante.
- Polymorphism (polimorfismo): consiste em atribuir responsabilidades à abstrações e não objetos concretos, permitindo que eles possam sofrer variação conforme foi necessitado;
- Pure Fabrication (pura fabricação): consiste em uma classe que não representa nenhum conceito no domínio do problema, ela somente opera como uma classe prestadora de serviços, ou seja, somente executa o que se pede. É projetada para que possamos ter um baixo acoplamento e alta coesão no sistema;

6 CONCLUSÃO

Ao apresentar a solução para este problema e a importância do assunto, e seguidamente com tópicos de conteúdos aprendidos em sala e por pesquisas, concluo que é viável e útil um ambiente para que empresas tenham mais visibilidade e demanda, e que consumidores consigam ter um contato mais ágil com a empresa em que gostaria de desfrutar de seus produtos ou serviços, pois além da economia local girar, empresas com potencial poderão investir mais em si mesmas e os consumidores terão cada vez mais variedades de produtos e serviços de qualidade.

REFERÊNCIAS

BOAS, Leandro Vilas. Padrões GRASP — Padrões de Atribuir Responsabilidades. Disponível em: https://medium.com/@leandrovboas/padrões-grasp-padrões-de-atribuir-responsabilidades-1ae4351eb204. Acessado em: 10/07/2020.

SOUSA, Julio Martins de. Levantamento de Requisitos – O ponto de partida do projeto de software. Disponível em: https://blog.cedrotech.com/levantamento-de-requisitos-o-ponto-de-partida-do-projeto-de-software/#:~:text=Mas%20o%20que%20é%20o,que%20o%20software%20vai%20fazer. Acessado em: 09/07/2020.

LAZARINI, Jader. IBGE: 21% das empresas quebram após o primeiro ano em atividade. Disponível em: https://www.sunoresearch.com.br/noticias/ibge-empresas-quebram-apos-um-ano/. Acessado em: 09/07/2020.

Professor Digital. Modelagem de dados: modelo conceitual, modelo lógico e físico. Disponível em: https://www.luis.blog.br/modelagem-de-dados-modelo-conceitual-modelo-logico-e-fisico.html. Acessado em: 09/07/2020.

VIEIRA, Rodrigo. UML — Diagrama de Casos de Uso. Disponível em: https://medium.com/operacionalti/uml-diagrama-de-casos-de-uso-29f4358ce4d5. Acessado em: 09/10/2020.

NETO, Antonio Rodrigues Carvalho. Java orientação a objetos (associacao, composicao, agregacao). Disponível em: https://pt.slideshare.net/armandodaniel777/java-orientao-a-objetos-associacao-composicao-agregacao. Acessado em: 10/07/2020.

ROBERTO, Jones. Design Patterns — Parte 2 — Os Padrões do GOF. Disponível em: https://medium.com/xp-inc/desing-patterns-parte-2-2a61878846d. Acessado em:10/07/2020.