

Lojas Happy

Grupo 01

Criação de plataforma única para controle geral
dos funcionários com os clientes e estoque

Fernando Martiniano Nascimento

Gustavo Nascimento

Erick Michael Porto

Vitor Carlos Mendes Lucats

Daniel Moraes Barbosa

Danilo Melo da Silva

Documentação

Nome fictício. Loja Happy

Endereço. Av. Eng. Eusébio Stevaux, 823 - Santo Amaro, São Paulo - SP

Objetivo. Facilitar a usabilidade dos caixas, gerentes e administradores, possibilitando, por meio de uma plataforma única, controle total dos produtos e seus fornecedores, assim como de clientes.

Logo.



Problema conceitual. A loja de brinquedos XPTO atualmente efetua todas suas vendas de forma manual, sendo elas recibos, notas manuais de compra e controla todo seu caixa e faturamento via caderno e após isso lança em uma planilha de Excel, recentemente a loja decidiu melhorar seu atendimento ao cliente, onde eles querem ter controle melhor de suas compras, vendas, faturamentos e gastos com algum sistema, também pensam em expandir o sistema para controle de seus fornecedores podendo criar agendamento de entrega de produtos, e controle de estoque e por último ofertar para o cliente uma fidelidade ou vínculo onde gera descontos a cada X compras. Desta forma será construído para o cliente um sistema de CRM com controles empresariais para uso local.

Código-fonte.

Linguagem. Java

Versionador de projeto.

<https://github.com/Caous/LojaDeBrinquedo/blob/main/README.md>

Requisitos Funcionais

- Cadastrar Clientes
- Cadastrar Produtos
- Cadastrar fornecedores
- Fornecer os dados dos clientes cadastrados
- Fornecer os dados dos produtos em estoque
- Fornecer os dados dos fornecedores
- Realizar a venda de produtos
- Relatar as vendas organizadas por cliente
- Relatar o histórico de vendas organizado por data
- Realizar o login do funcionário

Requisitos Não-Funcionais

- Ter usabilidade simples para os funcionários de qualquer faixa etária
- Validar as informações de todos os cadastros tanto na tela quanto dentro das classes
- Não disponibilizar os dados dos clientes para todos os tipos de funcionários
- Fornecer as informações conforme o tipo de usuário utilizando o sistema
- Disponibilizar a opção de excluir fornecedores e clientes apenas para administradores e gerentes
- Diferenciar os tipos de clientes

Diagrama Caso de Uso

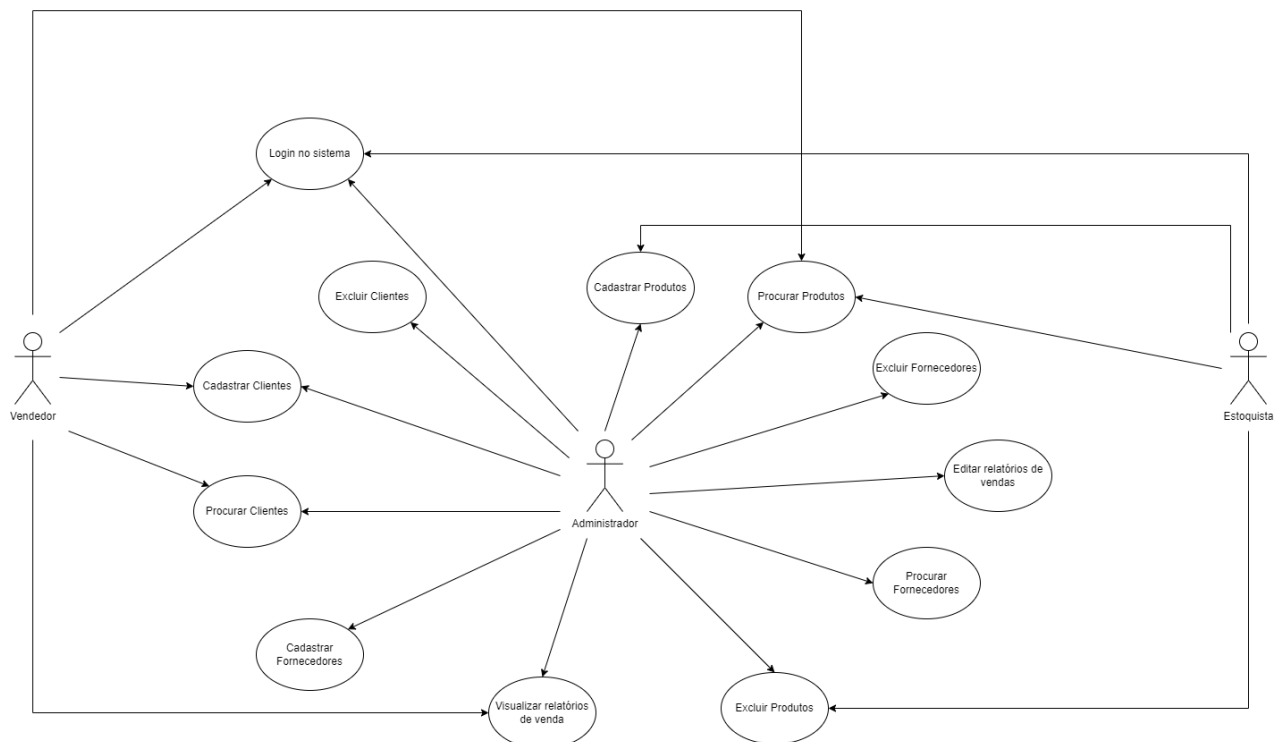
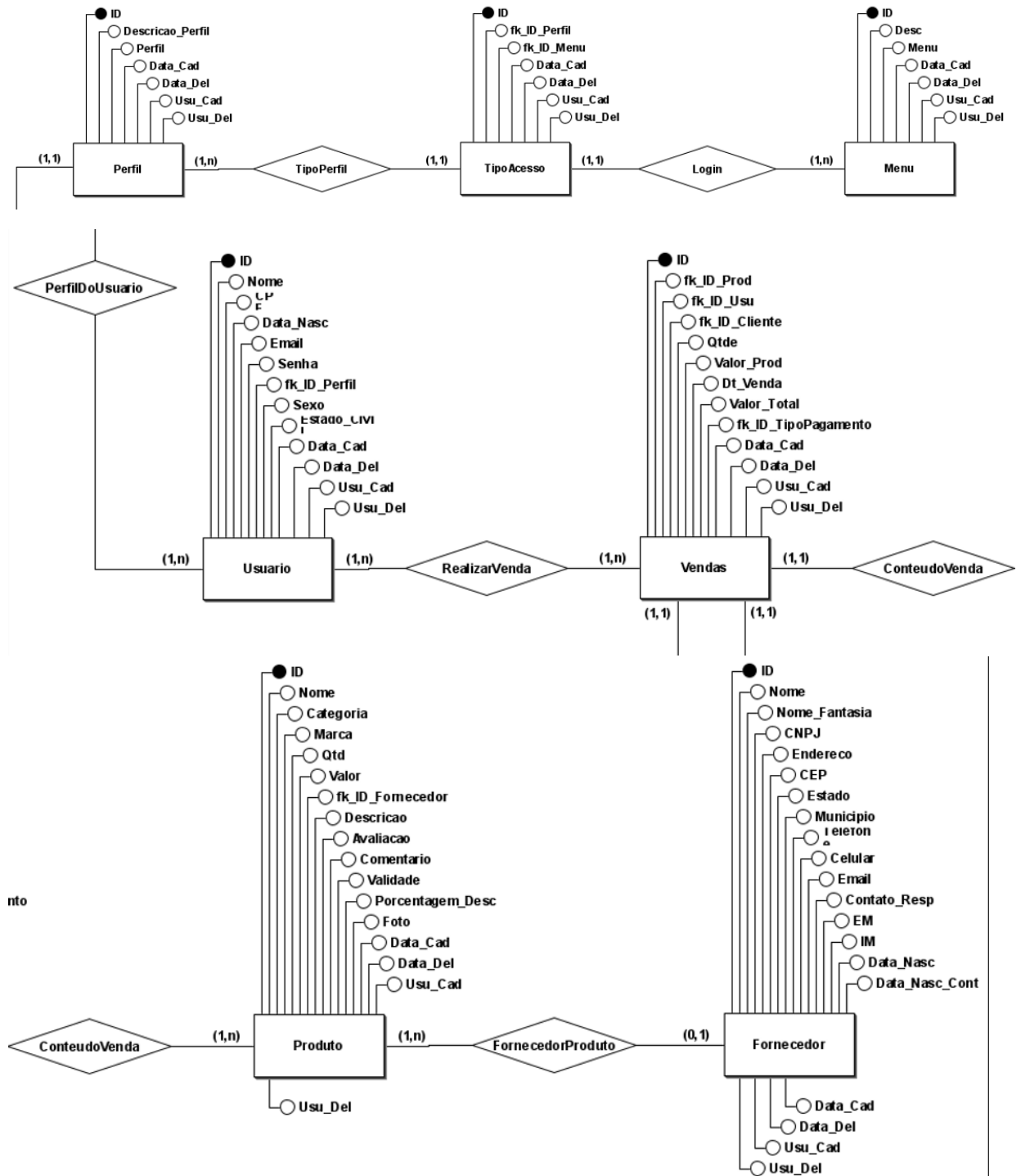
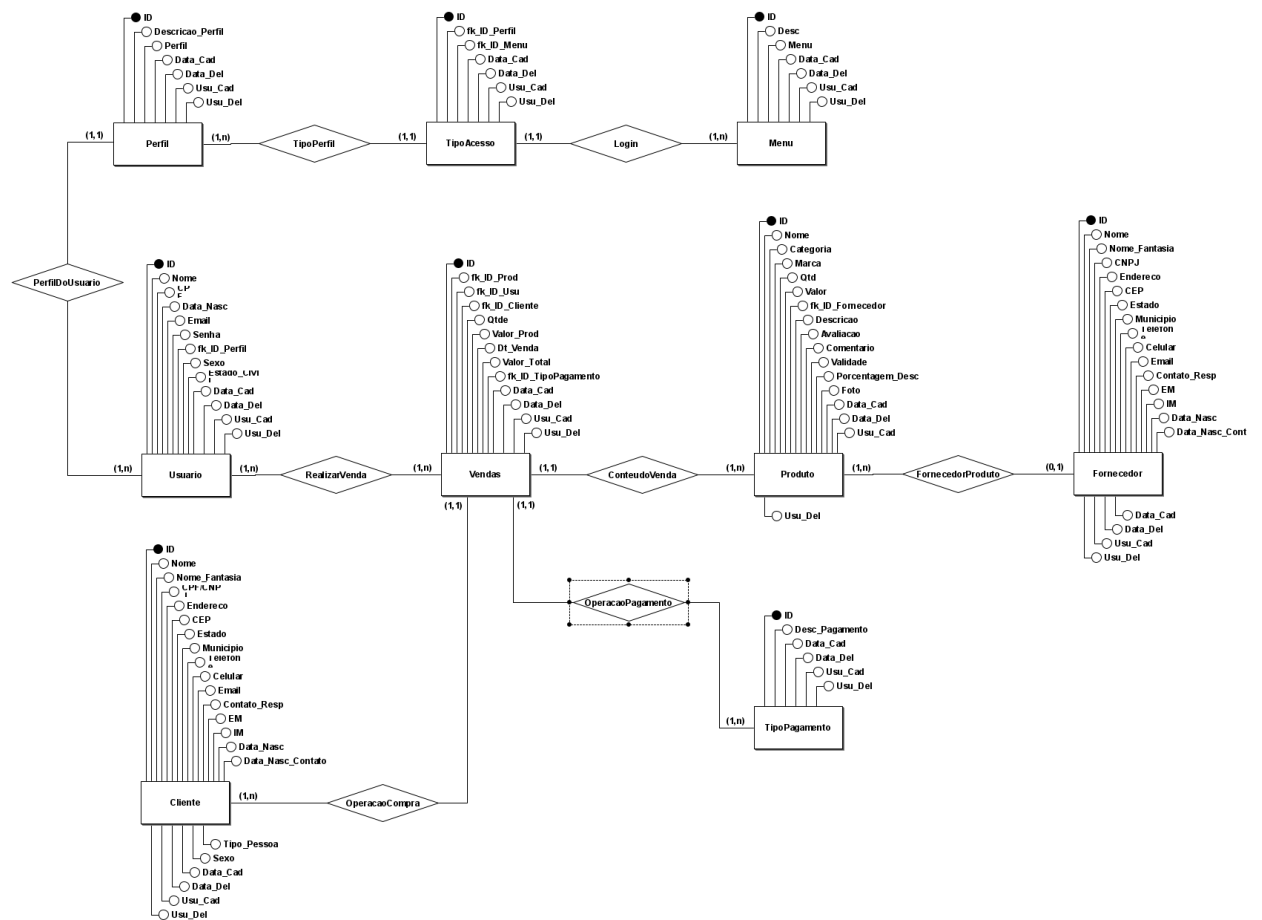
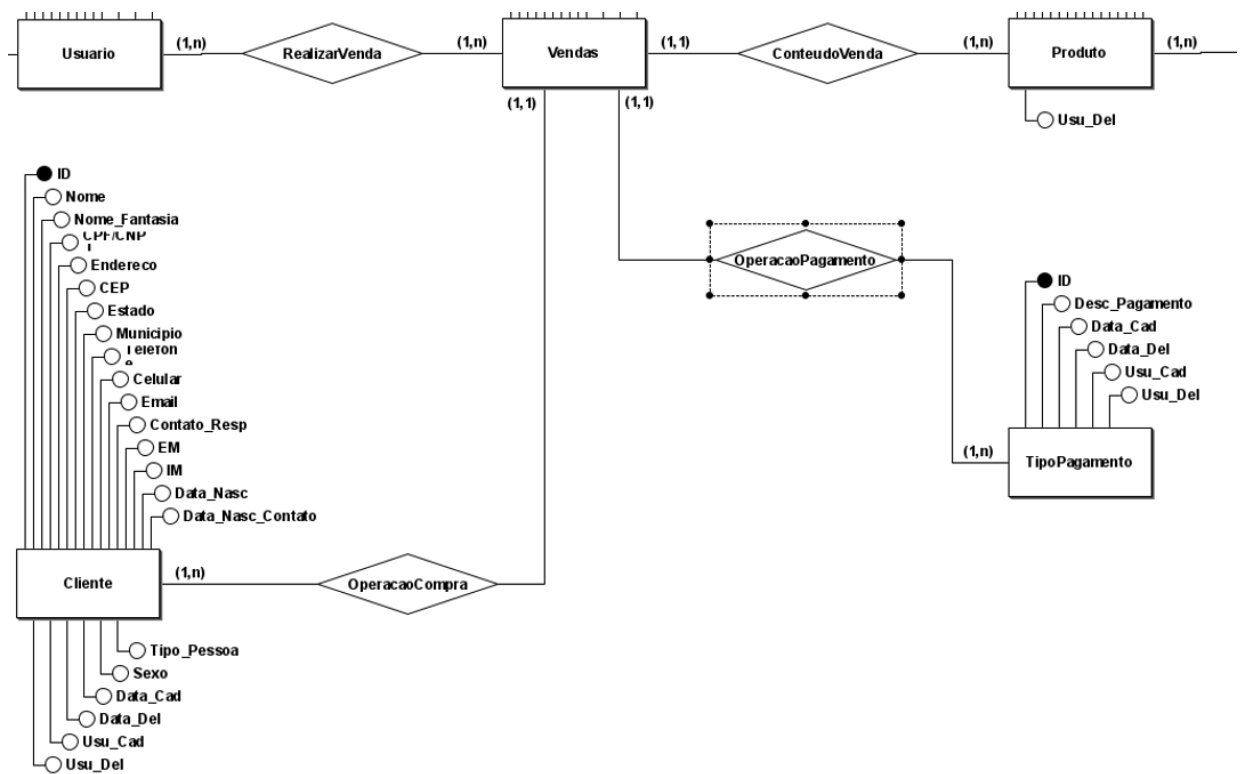


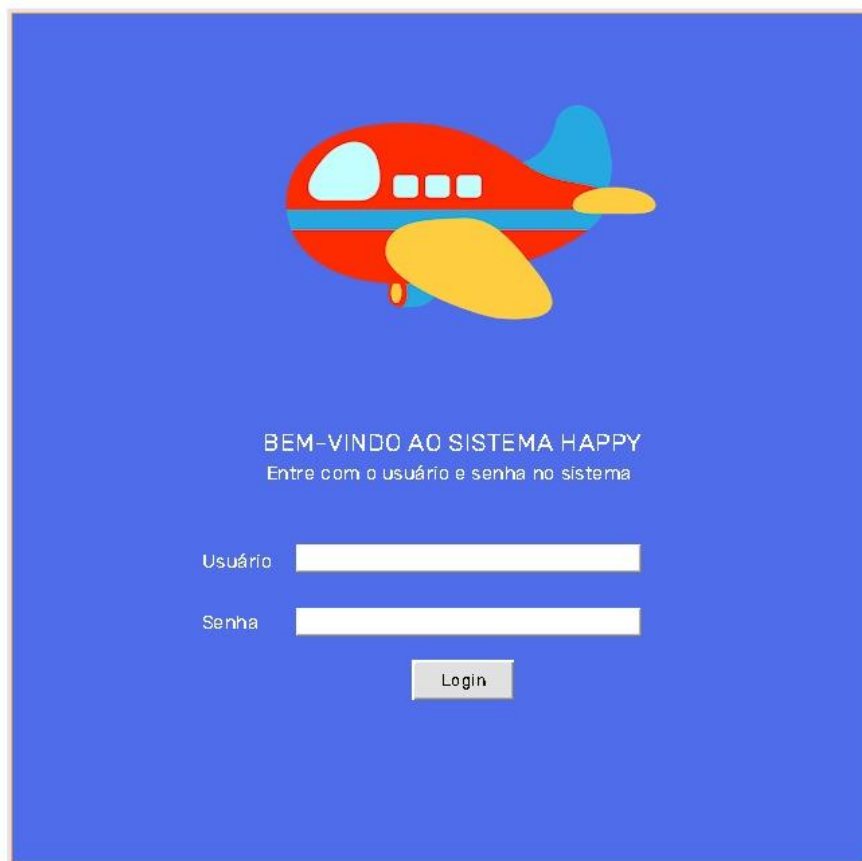
Diagrama ER





Apresentação de Telas

1. Tela de Login:



BEM-VINDO AO SISTEMA HAPPY
Entre com o usuário e senha no sistema


Usuário

Senha

Login

2. Tela do Sistema Geral:

Produtos | Cliente | Pagamento

 Controle de clientes

Posui Cadastro
☒ Sim ☐ Não

CPF
 000.000.00-00 Pesquisar

☒ Empresa

Nome Senac E-mail administrador@email.com IM 000.000.000-00 IE 000.000.000-00

Nome Fantasia Senac Santo Maro Cont. Responsável 01/01/1999 Telefone 000.000.000-00 Celular 000.000.000-00


Dt. Nascimento 000.000.000-00 ☒ Masculino ☐ Feminino

Endereço 000.000.000-00 CEP 000.000.000-00 Estado 000.000.000-00 Município 000.000.000-00

Salvar

5. Tela de Pagamento:

Produtos | Cliente | Pagamento

 Pagamento Finalizar Compra Cancelar

Nome Senac CPF 000.000.00-00 E-mail administrador@email.com


Endereço 000.000.000-00 Estado 000.000.000-00 Município 000.000.000-00 CEP 000.000.000-00

Desconto 2 Tipo Pagamento Cartão de crédito

Produtos Vendidos 0 Total de Vendas R\$ 0 Descontos Aplicados R\$ 0

Nome	Qtd	Valor	Marca	Fornecedor	Desconto
Bola de futebol	10	69,90	Nike	Nike	5
Boneca	5	120,00	Barbie	Happy	0
Nerf	2	50,00	Nerf	Disneylandia	2

6. Tela de Controle (Clientes):

 Controle de clientes

☒ Empresa ☐ Excluir

Nome Nome Fantasia E-mail Cont. Responsável

CPF/CNPJ IM IE Telefone

Celular Dt. Nascimento ☒ Masculino ☐ Feminino

Endereço CEP Estado Município

Nome	E-mail	CPF/CNPJ	Cont. Resp.
Gustavo	gustavo@happy.com.br	000.000.000-00	Dan
Erick	erick@happy.com.br	000.000.000-00	Vitor
Fernando	fernando@happy.com.br	000.000.000-00	N/A

7. Tela de Controle (Produtos):

 Controle de Produtos

Nome Categoria Marca Fornecedor


Quantidade R\$ Avaliação Validade

Porcentagem % Foto ☐ Excluir

Descrição Comentário

Nome	Qtd	Valor	Marca	Fornecedor	Desconto
Bola de futebol	10	69,90	Nike	Nike	5
Boneca	5	120,00	Barbie	Happy	0
Nerf	2	50,00	Nerf	Disneylandia	2

8. Tela de Controle (Fornecedor):

 Controle de Fornecedor ☐ Excluir

Nome Nome Fantasia E-mail Cont. Responsável


CNPJ IM IE Telefone

Celular Dt. Nascimento Endereço Município

CEP Estado

Nome	E-mail	CPF/CNPJ	Cont. Resp.
Gustavo	gustavo@happy.com.br	000.000.000-00	Dan
Erick	erick@happy.com.br	000.000.000-00	Vitor
Fernando	fernando@happy.com.br	000.000.000-00	N/A

9. Tela de Relatórios:

 Relatório

Nome Vendedor Categoria Marca Fornecedor

Quantidade Dt. Venda Porcentagem Cliente

Nome	Qtd	Valor	Marca	Fornecedor	Desconto	Valor Total
Bola de futebol	10	69,90	Nike	Nike	5	R\$ 1.526,00
Boneca	5	120,00	Barbie	Happy	0	R\$ 50,00
Nerf	2	50,00	Nerf	Disneylandia	2	R\$ 5,00

9. Telas de Permissão de Acesso:

Perfil

Acesso

Menu

Tipo Pagamento

Menu

ADM

Descrição

heats containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Salvar

Nome	Descrição	Ativo
ADM	Administrador do sistem	True
Gerente	Gerente geral	False
Caixa	Registrador de caixa	True

Cancelar

Perfil

Acesso

Menu

Tipo Pagamento

Tipo Pagamento

Descrição

heats containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Salvar

Nome	Descrição	Ativo
ADM	Administrador do sistem	True
Gerente	Gerente geral	False
Caixa	Registrador de caixa	True

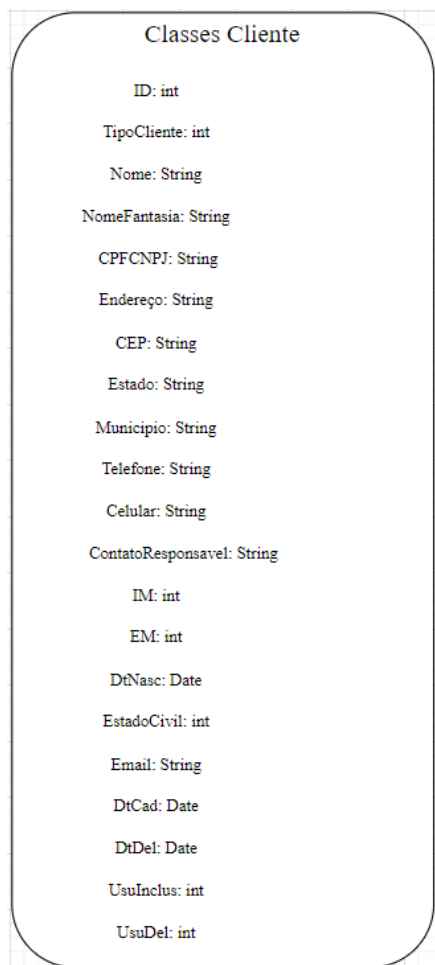
Cancelar

Definição de classes do sistema:

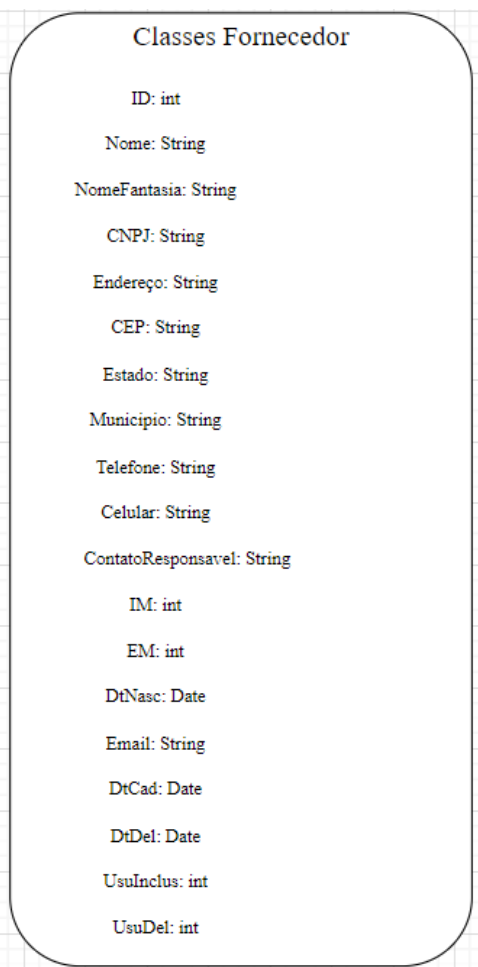
Utilização do Designer Domain-Driven Design (DDD) para o projeto juntamente com arquitetura de generic repository.

Classes de consumidores no banco de dados iram utilizar interfaces de herança, para consumidores. Com injeção de dependência.

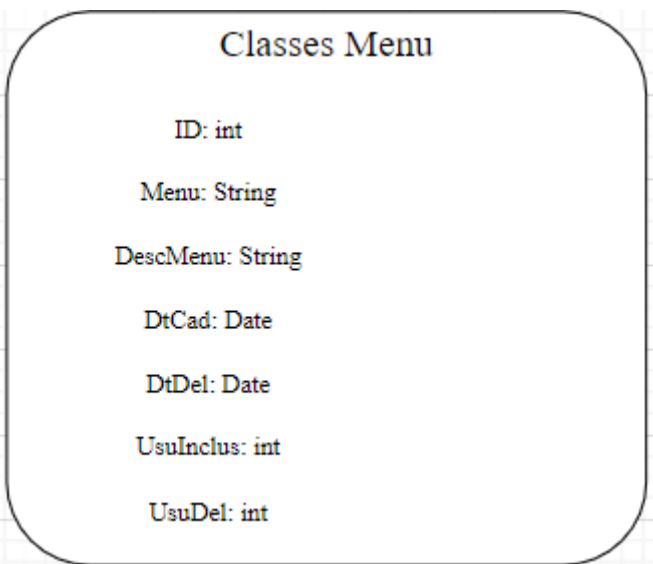
1. Classes Cliente:



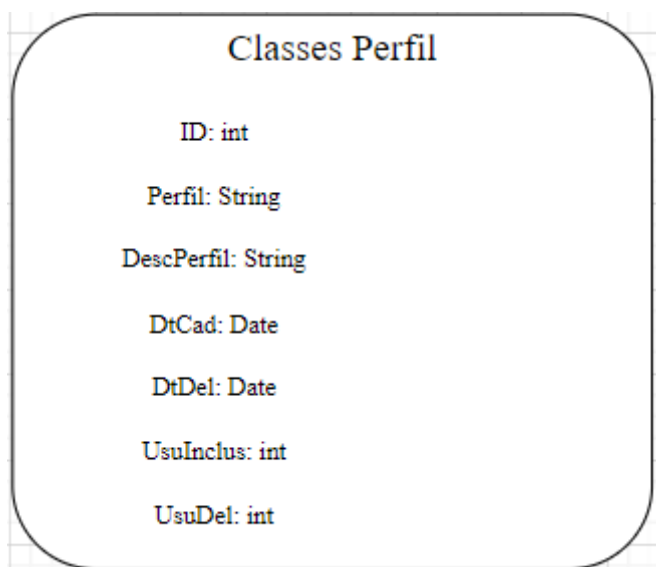
2. Classes Fornecedor:



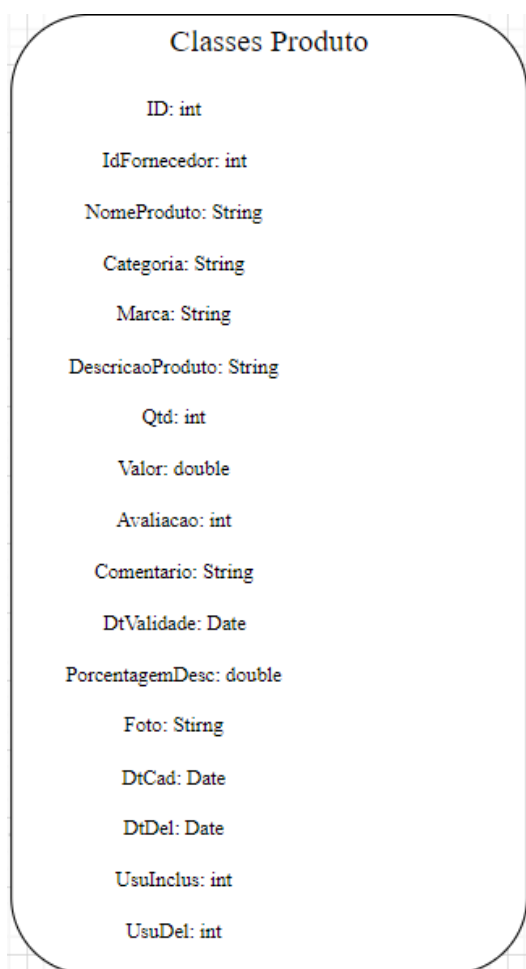
3. Classes Menu:



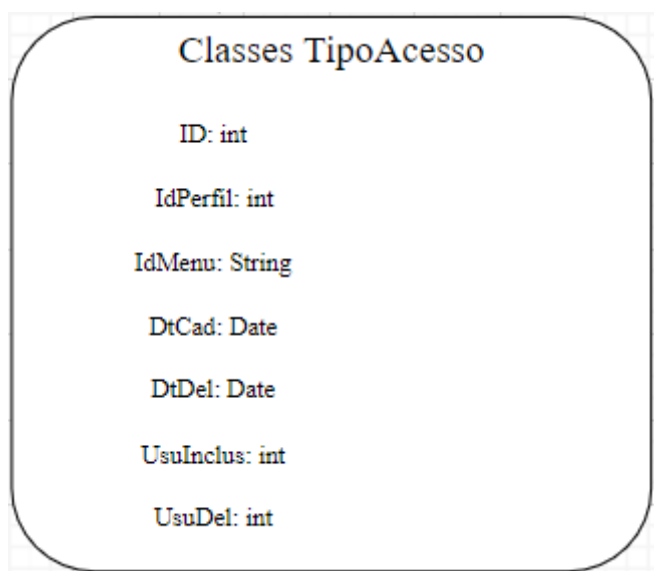
4. Classes Perfil:



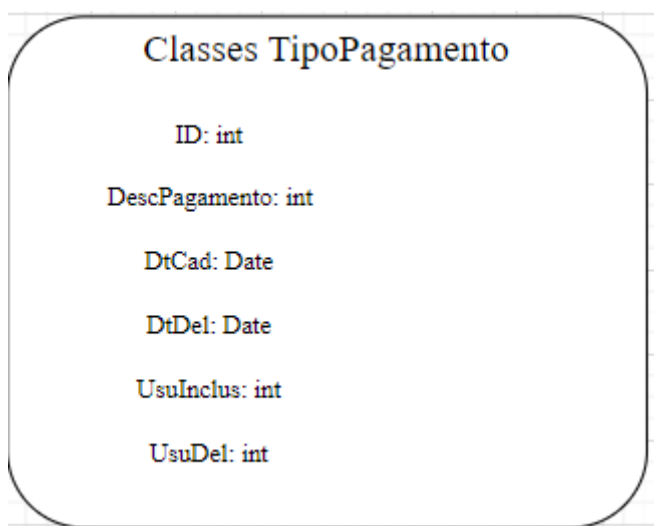
5. Classes Produto:



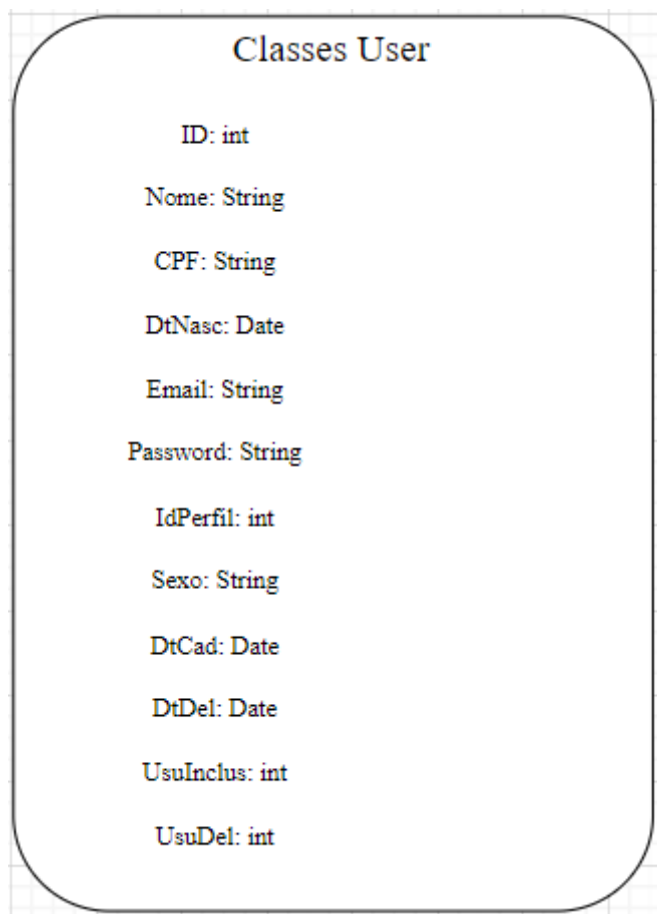
6. Classes TipoAcesso:



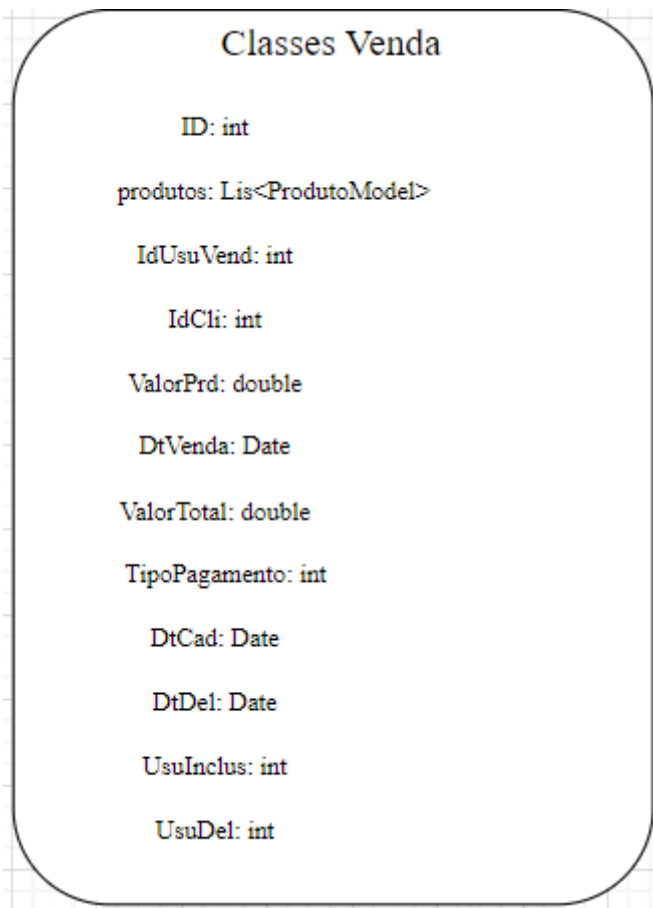
7. Classes TipoPagamento:



8. Classes User:



9.Classes Venda:



Validações

Arquivos com tratativas de erros comuns e consumidores.

- Banco de dados (Erro conexão, sqlStatement, sqlFormat)
- Conversion (parse – int – string – decimal – double)
- Regra de negócio

As validações a seguir estão localizadas dentro das classes anteriormente apresentadas.

1. Validação Cliente:

```

232  @Override
233  public void validObject(ClienteModel objeto) throws PropertiesValidator {
234
235      if (this.TipoCliente == 1) {
236
237          if (!validString(this.ContatoResponsavel)) {
238              throw new PropertiesValidator("Por favor preencher o Responsável");
239          }
240          if (!validString(this.CpfCnpj)) {
241              throw new PropertiesValidator("Por favor preencher o CNPJ");
242          }
243          if (!validString(this.Nome)) {
244              throw new PropertiesValidator("Por favor preencher o Nome da empresa");
245          }
246          if (!validString(this.NomeFantasia)) {
247              throw new PropertiesValidator("Por favor preencher o Nome Fantasia da empresa");
248          }
249          if (!validString(this.CEP)) {
250              throw new PropertiesValidator("Por favor preencher o CEP da empresa");
251          }
252          if (!validString(this.Celular)) {
253              throw new PropertiesValidator("Por favor preencher o Celular da empresa");
254          }
255          if (!validString(this.Email)) {
256              throw new PropertiesValidator("Por favor preencher o Email da empresa");
257          }
258          if (!validString(this.Endereco)) {
259              throw new PropertiesValidator("Por favor preencher o Endereco da empresa");
260          }
261          if (!validString(this.Estado)) {
262              throw new PropertiesValidator("Por favor preencher o Estado da empresa");
263          }
264          if (!validString(this.Municipio)) {
265              throw new PropertiesValidator("Por favor preencher o Municipio da empresa");
266          }
267          if (!validString(this.Telefone)) {
268              throw new PropertiesValidator("Por favor preencher o Telefone da empresa");
269          }
270          if (!validDate(this.DtCad)) {
271              throw new PropertiesValidator("Por favor preencher a Data de cadastro da empresa (TI)");
272          }
273          if (!validInt(this.UsuInclus)) {
274              throw new PropertiesValidator("Por favor preencher o usuário de inclusão da empresa (TI)");
275          }
276
277      } else {
278

```

```

277         } else {
278
279             if (!validString(this.Nome)) {
280                 throw new PropertiesValidator("Por favor preencher o Nome do cliente");
281             }
282             if (!validString(this.CEP)) {
283                 throw new PropertiesValidator("Por favor preencher o CEP do cliente");
284             }
285             if (!validString(this.Celular)) {
286                 throw new PropertiesValidator("Por favor preencher o Celular do cliente");
287             }
288             if (!validString(this.CpfCnpj)) {
289                 throw new PropertiesValidator("Por favor preencher o Cpf do cliente");
290             }
291             if (!validString(this.Email)) {
292                 throw new PropertiesValidator("Por favor preencher o Email do cliente");
293             }
294             if (!validString(this.Endereco)) {
295                 throw new PropertiesValidator("Por favor preencher o Endereco do cliente");
296             }
297             if (!validString(this.Estado)) {
298                 throw new PropertiesValidator("Por favor preencher o Estado do cliente");
299             }
300             if (!validString(this.Municipio)) {
301                 throw new PropertiesValidator("Por favor preencher o Municipio do cliente");
302             }
303             if (!validString(this.Sexo)) {
304                 throw new PropertiesValidator("Por favor preencher o Sexo do cliente");
305             }
306             if (!validString(this.Telefone)) {
307                 throw new PropertiesValidator("Por favor preencher o Telefone do cliente");
308             }
309             if (!validDate(this.DtCad)) {
310                 throw new PropertiesValidator("Por favor preencher a Data de cadastro do cliente (TI)");
311             }
312             if (!validDate(this.DtNasc)) {
313                 throw new PropertiesValidator("Por favor preencher a Data de nascimento do cliente (TI)");
314             }
315             if (!validInt(this.UsuInclus)) {
316                 throw new PropertiesValidator("Por favor preencher o usuário de inclusão do cliente (TI)");
317             }
318         }
319     }
320
321
322
323     @Override
324     public boolean validString(String value) throws PropertiesValidator {
325
326         if (value.isBlank() || value.isEmpty()) {
327             return false;
328         }
329
330         return true;
331     }
332
333     @Override
334     public boolean validDate(Date value) throws PropertiesValidator {
335         if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
336             return false;
337         }
338         return true;
339     }
340
341     @Override
342     public boolean validInt(int value) throws PropertiesValidator {
343         if (value >= 0) {
344             return true;
345         }
346         return false;
347     }
348
349     @Override
350     public boolean validDouble(double value) throws PropertiesValidator {
351         if (value >= 0) {
352             return true;
353         }
354         return false;
355     }
356 }

```

2. Validação Fornecedor:

```
202  @Override
203  public void validObject(FornecedorModel objeto) throws PropertiesValidator {
204      if (!validString(this.ContatoResponsavel)) {
205          throw new PropertiesValidator("Por favor preencher o Responsável");
206      }
207      if (!validString(this.Cnpj)) {
208          throw new PropertiesValidator("Por favor preencher o CNPJ");
209      }
210      if (!validString(this.Nome)) {
211          throw new PropertiesValidator("Por favor preencher o Nome da empresa");
212      }
213      if (!validString(this.NomeFantasia)) {
214          throw new PropertiesValidator("Por favor preencher o Nome Fantasia da empresa");
215      }
216      if (!validString(this.CEP)) {
217          throw new PropertiesValidator("Por favor preencher o CEP da empresa");
218      }
219      if (!validString(this.Celular)) {
220          throw new PropertiesValidator("Por favor preencher o Celular da empresa");
221      }
222      if (!validString(this.Email)) {
223          throw new PropertiesValidator("Por favor preencher o Email da empresa");
224      }
225      if (!validString(this.Endereco)) {
226          throw new PropertiesValidator("Por favor preencher o Endereco da empresa");
227      }
228      if (!validString(this.Estado)) {
229          throw new PropertiesValidator("Por favor preencher o Estado da empresa");
230      }
231      if (!validString(this.Municipio)) {
232          throw new PropertiesValidator("Por favor preencher o Municipio da empresa");
233      }
234      if (!validString(this.Telefone)) {
235          throw new PropertiesValidator("Por favor preencher o Telefone da empresa");
236      }
237      if (!validDate(this.DtCad)) {
238          throw new PropertiesValidator("Por favor preencher a Data de cadastro da empresa (TI)");
239      }
240      if (!validInt(this.UsuInclus)) {
241          throw new PropertiesValidator("Por favor preencher o usuário de inclusão da empresa (TI)");
242      }
243  }
244
245  @Override
246  public boolean validDate(Date value) throws PropertiesValidator {
247      if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
248          return false;
249      }
250      return true;
251  }
252
253  @Override
254  public boolean validInt(int value) throws PropertiesValidator {
255      if (value >= 0) {
256          return true;
257      }
258      return false;
259  }
260
261  @Override
262  public boolean validDouble(double value) throws PropertiesValidator {
263      if (value >= 0) {
264          return true;
265      }
266      return false;
267  }
268
269  }
```

3. Validação Menu:

```

81     @Override
82     public void validObject(MenuModel objeto) throws PropertiesValidator {
83         if (validString(this.Menu)) {
84             throw new PropertiesValidator("Por favor preencher o nome do menu");
85         }
86         if (validString(this.DescMenu)) {
87             throw new PropertiesValidator("Por favor preencher a descrição do menu");
88         }
89         if (!validDate(this.DtCad)) {
90             throw new PropertiesValidator("Por favor preencher a Data de cadastro do menu(TI)");
91         }
92         if (!validInt(this.UsuInclus)) {
93             throw new PropertiesValidator("Por favor preencher o usuário de inclusão do menu (TI)");
94         }
95     }
96
97     @Override
98     public boolean validString(String value) throws PropertiesValidator {
99
100         if (value.isBlank() || value.isEmpty()) {
101             return false;
102         }
103
104         return true;
105     }
106
107     @Override
108     public boolean validDate(Date value) throws PropertiesValidator {
109         if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
110             return false;
111         }
112         return true;
113     }
114
115     @Override
116     public boolean validInt(int value) throws PropertiesValidator {
117         if (value >= 0) {
118             return true;
119         }
120         return false;
121     }
122
123     @Override
124     public boolean validDouble(double value) throws PropertiesValidator {
125         if (value >= 0) {
126             return true;
127         }
128         return false;
129     }
130 }

```

4. Validação Perfil:

```

82     @Override
83     public void validObject(PerfilModel objeto) throws PropertiesValidator {
84         if (validString(this.Perfil)) {
85             throw new PropertiesValidator("Por favor preencher o nome do perfil");
86         }
87         if (validString(this.DescPerfil)) {
88             throw new PropertiesValidator("Por favor preencher a descrição do perfil");
89         }
90         if (!validDate(this.DtCad)) {
91             throw new PropertiesValidator("Por favor preencher a Data de cadastro do menu(TI)");
92         }
93         if (!validInt(this.UsuInclus)) {
94             throw new PropertiesValidator("Por favor preencher o usuário de inclusão do menu (TI)");
95         }
96     }
97
98     @Override
99     public boolean validString(String value) throws PropertiesValidator {
100
101         if (value.isBlank() || value.isEmpty()) {
102             return false;
103         }
104
105         return true;
106     }
107
108     @Override
109     public boolean validDate(Date value) throws PropertiesValidator {
110         if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
111             return false;
112         }
113         return true;
114     }
115
116     @Override
117     public boolean validInt(int value) throws PropertiesValidator {
118         if (value >= 0) {
119             return true;
120         }
121         return false;
122     }
123
124     @Override
125     public boolean validDouble(double value) throws PropertiesValidator {
126         if (value >= 0) {
127             return true;
128         }
129         return false;
130     }
131 }

```

5. Validação Produto:

```

182     @Override
183     public void validObject(ProdutoModel objeto) throws PropertiesValidator {
184         if (!validString(this.NomeProduto)) {
185             throw new PropertiesValidator("Por favor preencher o nome do Produto");
186         }
187         if (!validString(this.Categoria)) {
188             throw new PropertiesValidator("Por favor preencher o Categoria do Produto");
189         }
190         if (!validString(this.DescricaoProduto)) {
191             throw new PropertiesValidator("Por favor preencher o Descrição do Produto");
192         }
193         if (!validString(this.Marca)) {
194             throw new PropertiesValidator("Por favor preencher o Marca do Produto");
195         }
196         if (!validInt(this.IdFornecedor)) {
197             throw new PropertiesValidator("Por favor preencher o fornecedor");
198         }
199         if (!validInt(this.Qtd)) {
200             throw new PropertiesValidator("Por favor preencher a quantidade de produto");
201         }
202         if (!validDate(this.DtCad)) {
203             throw new PropertiesValidator("Por favor preencher a Data de cadastro da empresa (TI)");
204         }
205         if (!validInt(this.UsuInclus)) {
206             throw new PropertiesValidator("Por favor preencher o usuário de inclusão da empresa (TI)");
207         }
208     }
209
210     @Override
211     public boolean validString(String value) throws PropertiesValidator {
212
213         if (value.isBlank() || value.isEmpty()) {
214             return false;
215         }
216
217         return true;
218     }
219
220     @Override
221     public boolean validDate(Date value) throws PropertiesValidator {
222         if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
223             return false;
224         }
225         return true;
226     }
227
228     @Override
229     public boolean validInt(int value) throws PropertiesValidator {
230         if (value >= 0) {
231             return true;
232         }
233         return false;
234     }
235
236     @Override
237     public boolean validDouble(double value) throws PropertiesValidator {
238         if (value >= 0) {
239             return true;
240         }
241         return false;
242     }
243 }

```

6. Validação TipoAcesso:


```

81     @Override
82     public void validObject(TipoAcessoModel objeto) throws PropertiesValidator {
83         if (!validInt(this.IdMenu)) {
84             throw new PropertiesValidator("Por favor selecionar o menu");
85         }
86         if (!validInt(this.IdPerfil)) {
87             throw new PropertiesValidator("Por favor selecionar o perfil");
88         }
89         if (!validDate(this.DtCad)) {
90             throw new PropertiesValidator("Por favor preencher a Data de cadastro do pagamento (TI)");
91         }
92         if (!validInt(this.UsuInclus)) {
93             throw new PropertiesValidator("Por favor preencher o usuário de inclusão do pagamento (TI)");
94         }
95     }
96
97     @Override
98     public boolean validString(String value) throws PropertiesValidator {
99
100         if (value.isBlank() || value.isEmpty()) {
101             return false;
102         }
103
104         return true;
105     }
106
107     @Override
108     public boolean validDate(Date value) throws PropertiesValidator {
109         if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
110             return false;
111         }
112         return true;
113     }
114
115     @Override
116     public boolean validInt(int value) throws PropertiesValidator {
117         if (value >= 0) {
118             return true;
119         }
120         return false;
121     }
122
123     @Override
124     public boolean validDouble(double value) throws PropertiesValidator {
125         if (value >= 0) {
126             return true;
127         }
128         return false;
129     }
130 }

```

7. Validação TipoPagamento:

```

72     @Override
73     public void validObject(TipoPagamentoModel objeto) throws PropertiesValidator {
74         if (validString(this.DescPagamento)) {
75             throw new PropertiesValidator("Por favor preencher a descrição do pagamento");
76         }
77         if (!validDate(this.DtCad)) {
78             throw new PropertiesValidator("Por favor preencher a Data de cadastro do pagamento (TI)");
79         }
80         if (!validInt(this.UsuInclus)) {
81             throw new PropertiesValidator("Por favor preencher o usuário de inclusão do pagamento (TI)");
82         }
83     }
84
85     @Override
86     public boolean validString(String value) throws PropertiesValidator {
87
88         if (value.isBlank() || value.isEmpty()) {
89             return false;
90         }
91
92         return true;
93     }
94
95     @Override
96     public boolean validDate(Date value) throws PropertiesValidator {
97         if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
98             return false;
99         }
100         return true;
101     }
102
103     @Override
104     public boolean validInt(int value) throws PropertiesValidator {
105         if (value >= 0) {
106             return true;
107         }
108         return false;
109     }
110
111     @Override
112     public boolean validDouble(double value) throws PropertiesValidator {
113         if (value >= 0) {
114             return true;
115         }
116         return false;
117     }
118 }

```

8. Validação User:

```

143     @Override
144     public void validObject(UserModel objeto) throws PropertiesValidator {
145
146         if (!validString(this.Nome))
147             throw new PropertiesValidator("Por favor preencher o nome");
148
149         if (!validString(this.CPF))
150             throw new PropertiesValidator("Por favor preencher o CPF");
151
152         if (!validString(this.Email))
153             throw new PropertiesValidator("Por favor preencher o Email");
154
155         if (!validString(this.Password))
156             throw new PropertiesValidator("Por favor preencher o Password");
157
158         if (!validString(this.Sexo))
159             throw new PropertiesValidator("Por favor preencher o Sexo");
160
161         if (!validDate(this.DtNasc))
162             throw new PropertiesValidator("Por favor preencher a Data de Nascimento");
163     }
164
165     @Override
166     public boolean validString(String value) throws PropertiesValidator {
167
168         if (value.isBlank() || value.isEmpty())
169             return false;
170
171         return true;
172     }
173
174     @Override
175     public boolean validDate(Date value) throws PropertiesValidator {
176         if (value == null || value.toString().isBlank() || value.toString().isEmpty())
177             return false;
178         return true;
179     }
180
181     @Override
182     public boolean validInt(int value) throws PropertiesValidator {
183         throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem
184     }
185
186     @Override
187     public boolean validDouble(double value) throws PropertiesValidator {
188         throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem
189     }
190
191 }

```

9. Validação Venda:

```

138     @Override
139     public void validObject(VendaModel objeto) throws PropertiesValidator {
140         if (!validInt(this.IdCli)) {
141             throw new PropertiesValidator("Por favor preencher o cliente");
142         }
143         if (!validInt(this.produtos.size())) {
144             throw new PropertiesValidator("Por favor preencher o produto");
145         }
146         if (!validInt(this.IdUsuVend)) {
147             throw new PropertiesValidator("Por favor preencher o usuário da venda (TI)");
148         }
149         if (!validInt(this.TipoPagamento)) {
150             throw new PropertiesValidator("Por favor preencher o tipo pagamento");
151         }
152         if (!validDate(this.DtCad)) {
153             throw new PropertiesValidator("Por favor preencher a Data de cadastro da empresa (TI)");
154         }
155         if (!validInt(this.UsuInclus)) {
156             throw new PropertiesValidator("Por favor preencher o usuário de inclusão da empresa (TI)");
157         }
158     }
159
160     @Override
161     public boolean validString(String value) throws PropertiesValidator {
162
163         if (value.isBlank() || value.isEmpty()) {
164             return false;
165         }
166
167         return true;
168     }
169
170     @Override
171     public boolean validDate(Date value) throws PropertiesValidator {
172         if (value == null || value.toString().isBlank() || value.toString().isEmpty()) {
173             return false;
174         }
175         return true;
176     }
177
178     @Override
179     public boolean validInt(int value) throws PropertiesValidator {
180         if (value >= 0) {
181             return true;
182         }
183         return false;
184     }
185
186     @Override
187     public boolean validDouble(double value) throws PropertiesValidator {
188         if (value >= 0) {
189             return true;
190         }
191         return false;
192     }
193 }

```