Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Apucarana

Coordenação de Engenharia de Computação- COENC
Engenharia de Computação

João Pedro Cavani Meireles, RA: 2321424
Filipe Augusto Parreira Almeida, RA: 2320622

RELATÓRIO PROJETO FINAL – PROGRAMAÇÃO ORIENTADA À OBJETO

Apucarana
29/06/2023

1. DESCRIÇÃO DO PROJETO

Para o projeto final foi definido pelo professor um tema, para que fosse implementando um software que abordasse este tema. O tema escolhido pelo professo foi “prefeitura”, sendo assim, foi implementado um software para o setor de saneamento de água para uma cidade, ou seja, o setor responsável por fornecer e regulamentar o fornecimento de água.

O programa então tem como principal intuito facilitar o acesso aos serviços para o cidadão, que será o principal usuário do sistema, fornecendo serviços de auto atendimento, de certa forma que agilize o tempo do cidadão (chamado no software de Cliente), onde não será mais necessário ir até uma agência física deste setor da prefeitura pra resolver seus eventuais problemas com o serviço disponibilizado por este setor.

2. MODELAGEM UML

A seguir vão ser apresentados então os diagramas (e tabelas), que foram cruciais para modelar a lógica do sistema, em alguns aspectos o sistema pode não seguir exatamente a modelagem.

- Diagrama de casos de uso:

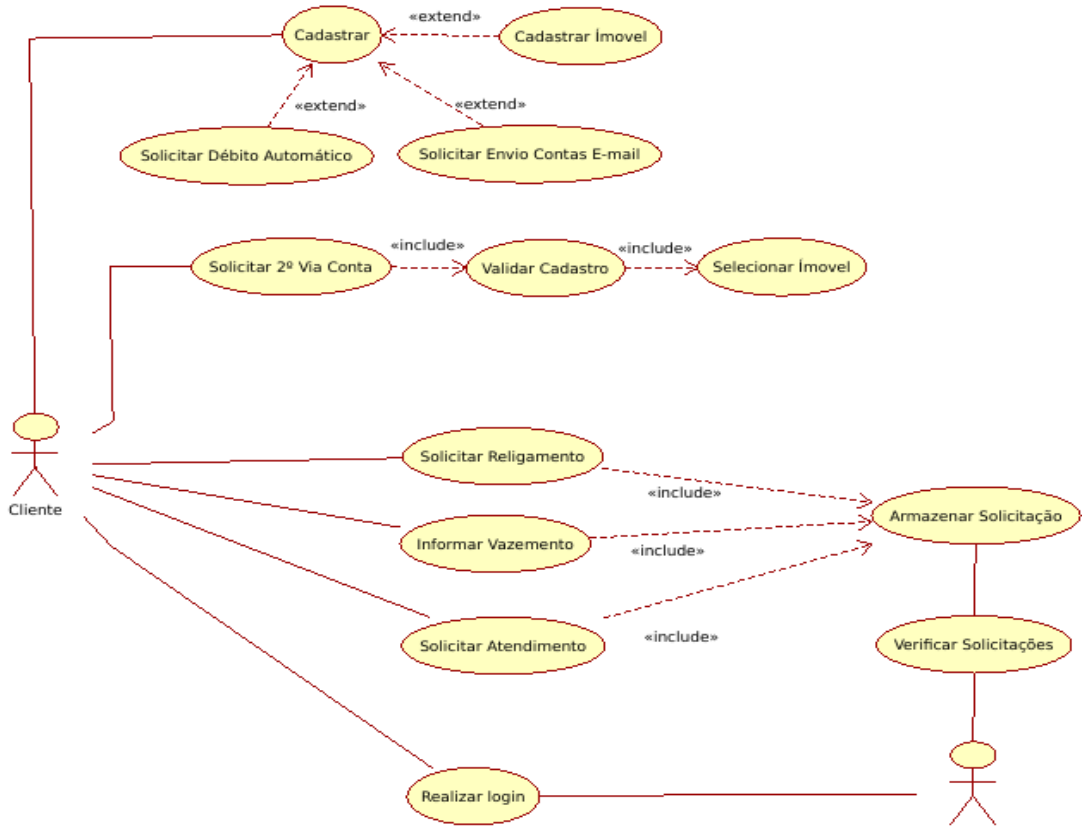


Figura 1. Diagrama de Casos de Uso

- Tabelas de casos de uso:

Caso de Uso	Valor
Nome do Caso de Uso	Cadastrar
Atores Envolvidos	Cliente
Finalidade	O cliente realiza o cadastro de seus dados no sistema.
Descrição detalhada	O cliente insere seus dados na interface gráfica, e então é cadastrado no banco de dados.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Cadastrar Imóvel
Atores Envolvidos	Cliente
Finalidade	O cliente cadastra um imóvel no banco de dados
Descrição detalhada	O cliente pode ou não cadastrar um imóvel juntamente com o cadastro, porém esse recurso também é disponibilizado posteriormente. Os imóveis cadastrados ficam vinculados ao cadastro do cliente.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Solicitar envio de contas e-mail
Atores Envolvidos	Cliente
Finalidade	O cliente solicita o envio de suas faturas por e-mail
Descrição detalhada	É uma opção que o cliente realiza na hora do cadastro, onde ele solicita que suas faturas mensais sejam enviadas por e-mail automaticamente.

Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Solicitar Débito Automático
Atores Envolvidos	Cliente
Finalidade	Cliente solicita que suas faturas sejam debitadas automaticamente em sua conta bancária.
Descrição detalhada	É uma opção onde o cliente a seleciona e insere os dados da sua conta bancária, de forma que suas faturas sejam debitadas automaticamente na conta bancária cadastrada.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Solicitar Segunda Via de Conta
Atores Envolvidos	Cliente
Finalidade	Cliente solicita a segunda via da sua fatura mensal.
Descrição detalhada	Caso haja algum problema e o cliente não tenha recebido sua fatura, o mesmo, pode então solicitar a segunda via.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Validar Cadastro
Atores Envolvidos	Cliente
Finalidade	O sistema solicita a valida o login do usuário.
Descrição detalhada	O sistema solicita as informações de login do usuário para que o usuário

	acesse sua conta e posteriormente selecione o imóvel que ele deseja solicitar a fatura.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Selecionar Imóvel
Atores Envolvidos	Cliente
Finalidade	Permite o cliente solicitar um imóvel.
Descrição detalhada	O cliente pode solicitar um imóvel para visualizar as faturas relacionadas exclusivamente com o imóvel selecionado.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Solicitar Religamento
Atores Envolvidos	Cliente, Funcionário
Finalidade	O cliente solicita o religamento do seu abastecimento de água.
Descrição detalhada	É enviado uma solicitação de religamento para que um funcionário possa aprovar e religar o abastecimento de água quando desligado.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Informar Vazamento
Atores Envolvidos	Cliente, Funcionário
Finalidade	O cliente informa um vazamento para a empresa.
Descrição detalhada	O cliente envia uma solicitação informando um vazamento e

	especificando o local, para que o funcionário possa tomar as devidas providencias.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Solicitar Atendimento
Atores Envolvidos	Cliente, Funcionário
Finalidade	O cliente solicita um eventual atendimento presencial.
Descrição detalhada	É realizado uma solicitação onde o cliente solicita um atendimento presencial, agendando um horário. Dessa forma o funcionário processa essa solicitação.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Realizar Login
Atores Envolvidos	Cliente, Funcionário
Finalidade	Tanto o funcionário quanto o cliente podem realizar o login com as suas informações.
Descrição detalhada	É solicitado então o login do usuário (id, senha), para que seja então carregado as informações do usuário que o realiza.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Armazenar Solicitação
Atores Envolvidos	Cliente, Funcionário
Finalidade	O sistema armazena a solicitação

	feita pelo cliente.
Descrição detalhada	A solicitação feita pelo cliente é armazenada no banco de dados possibilitando que o funcionário tenha acesso a elas.
Tabelas manipuladas	

Caso de Uso	Valor
Nome do Caso de Uso	Verificar Solicitações
Atores Envolvidos	Funcionário, Cliente
Finalidade	Permite que o funcionário verifique as solicitações feitas pelos clientes.
Descrição detalhada	O funcionário pode então ter acesso as solicitações armazenadas no banco de dados, aprovando-as ou não.
Tabelas manipuladas	

- Diagrama de Classes

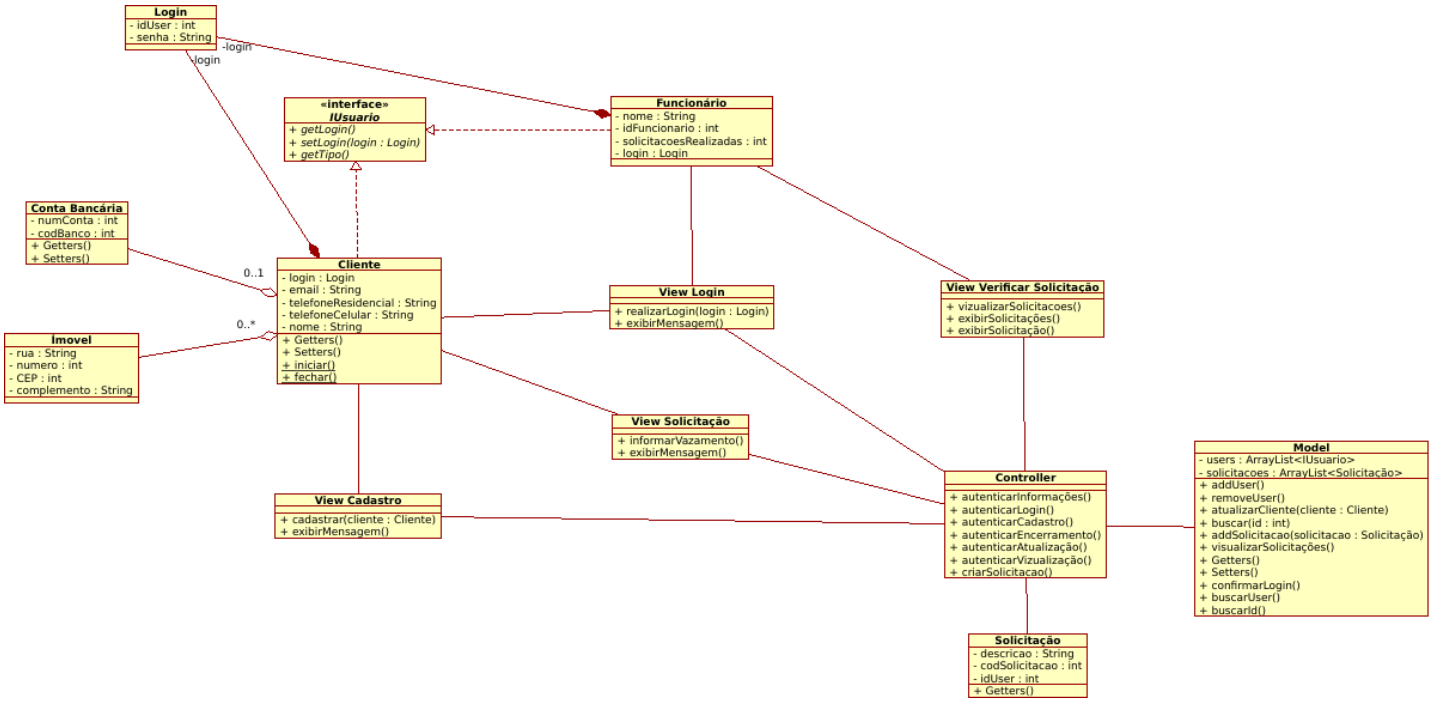


Figura 2 - Diagrama de Classes

- Diagramas de Sequência
    - Cadastrar

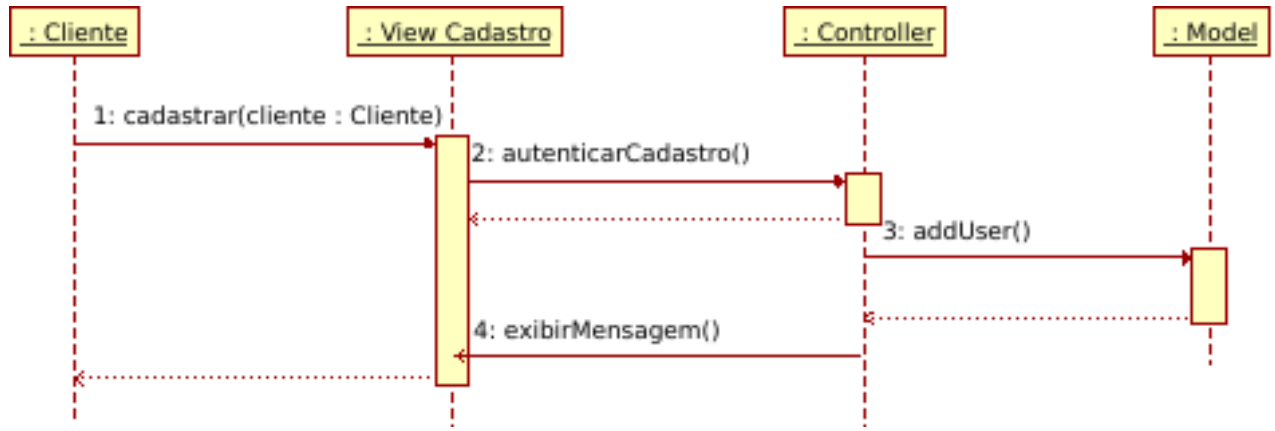


Figura 3. Diagrama de Sequência - Cadastrar

    - Informar Vazamento

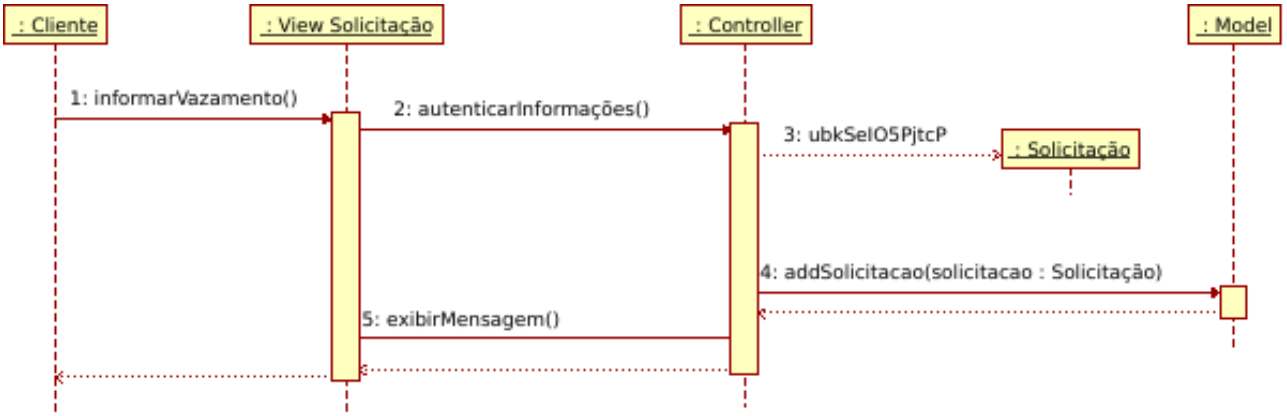


Figura 4. Diagrama de Sequência - Informar Vazamento

    - Realizar Login

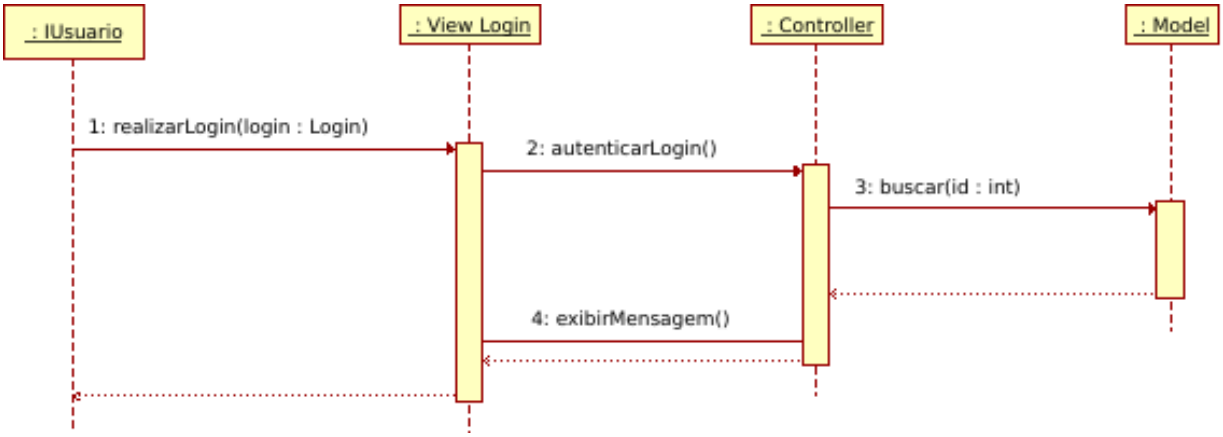


Figura 5. Diagrama de Sequência - Realizar Login

- Verificar Solicitação

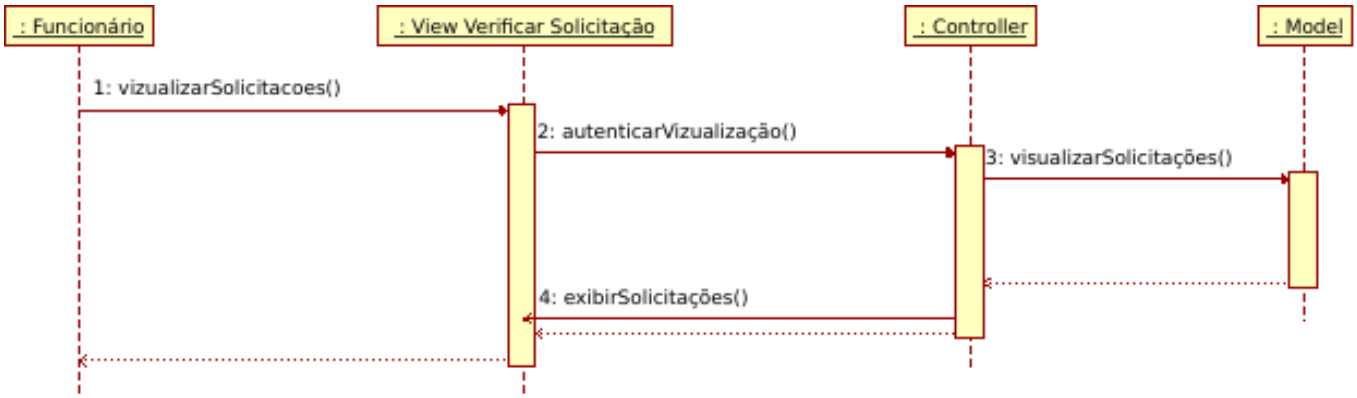


Figura 6. Diagrama de Sequência - Verificar Solicitação

- Diagrama de Estados
  - Cadastro

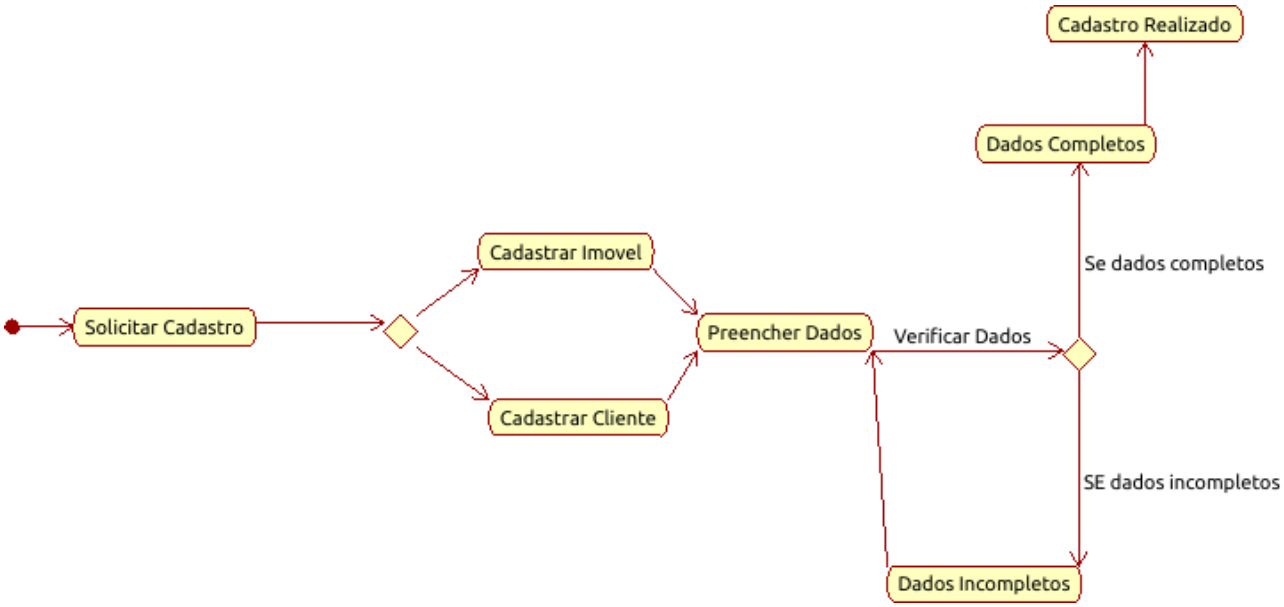
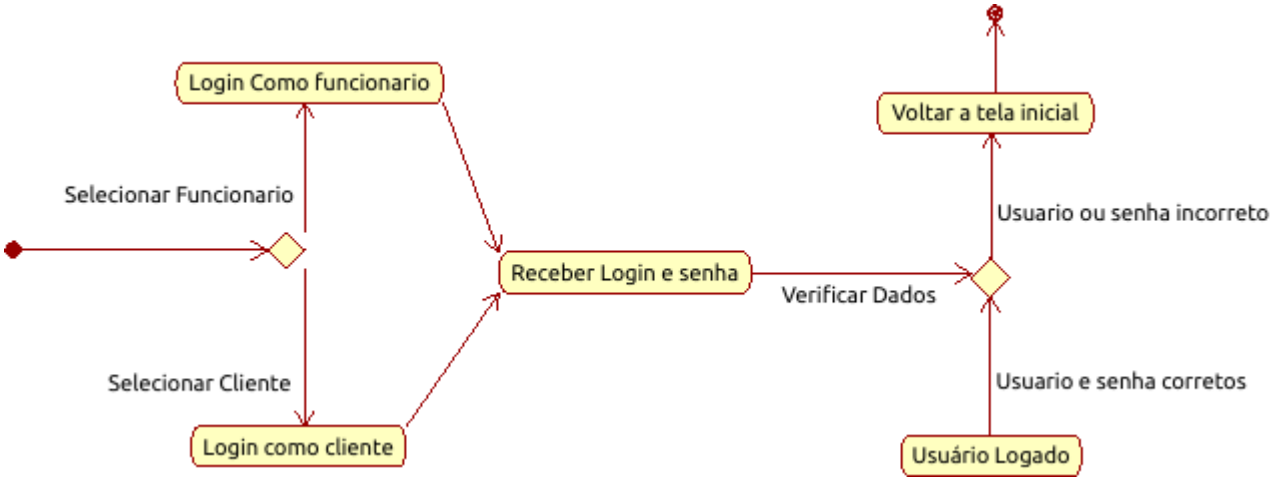


Figura 7. Diagrama de Estado - Cadastro

  - Login



- Solicitar Religamento

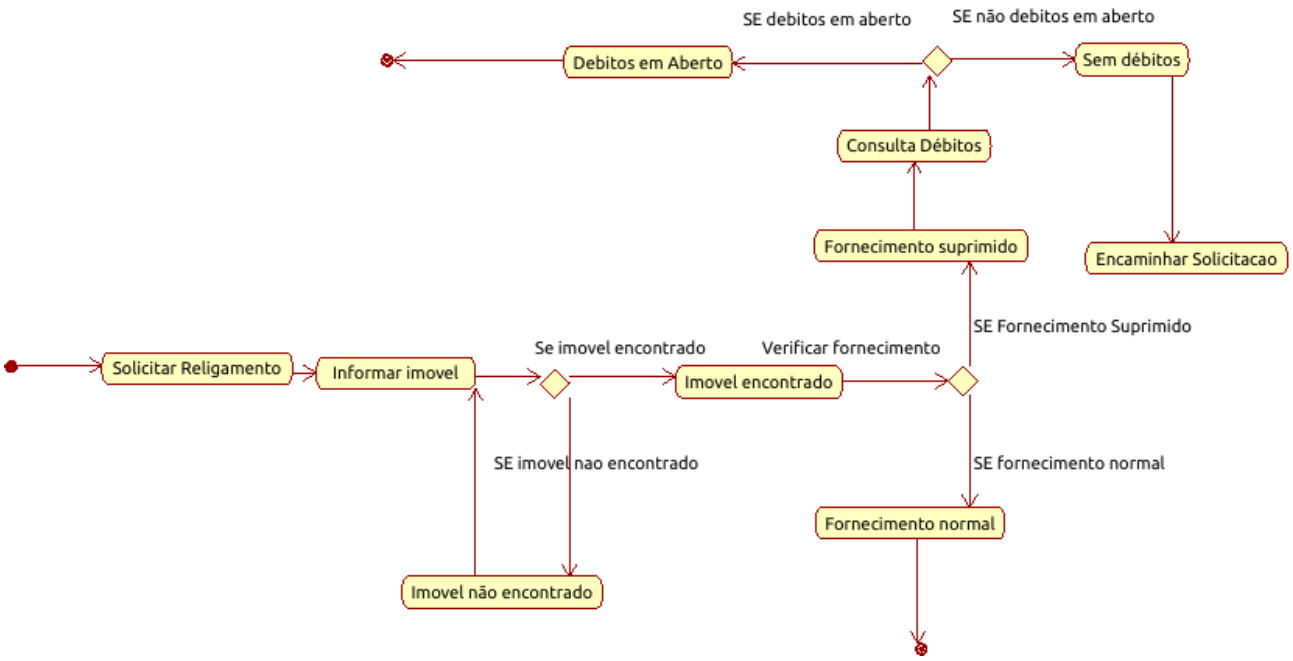


Figura 8. Diagrama de Estado - Solicitar Religamento

- Solicitar Segunda via de conta

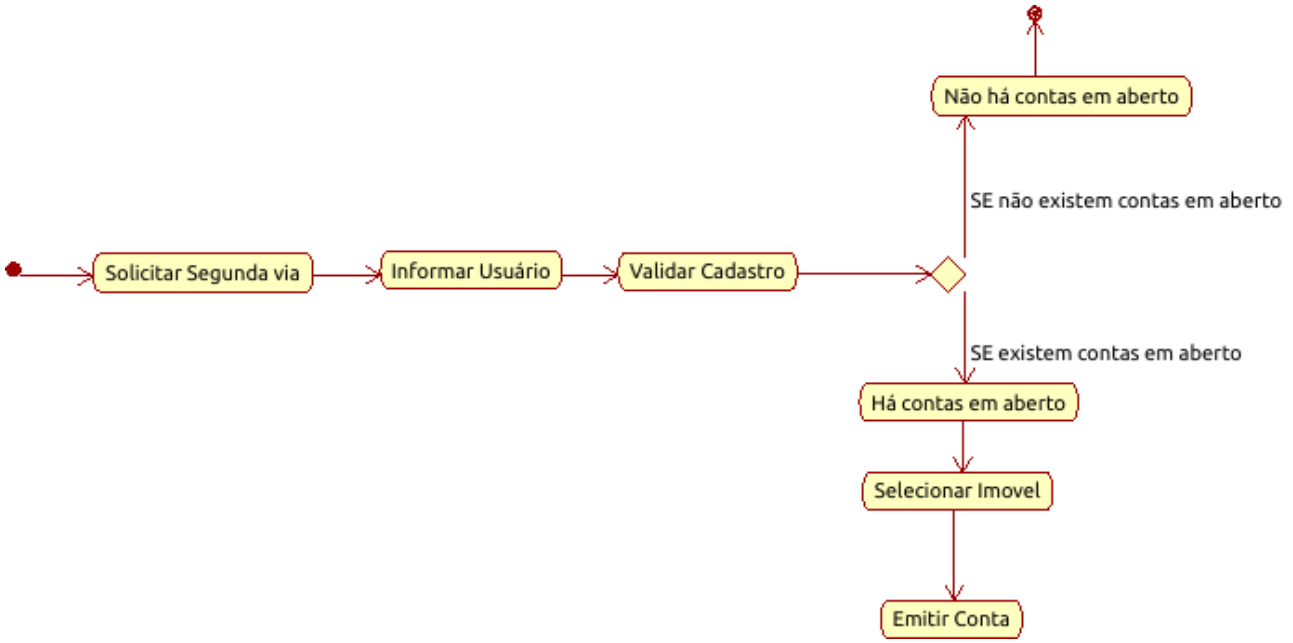


Figura 9. Diagrama de Estados Solicitar Segunda Via de Conta

- Informar Vazamento

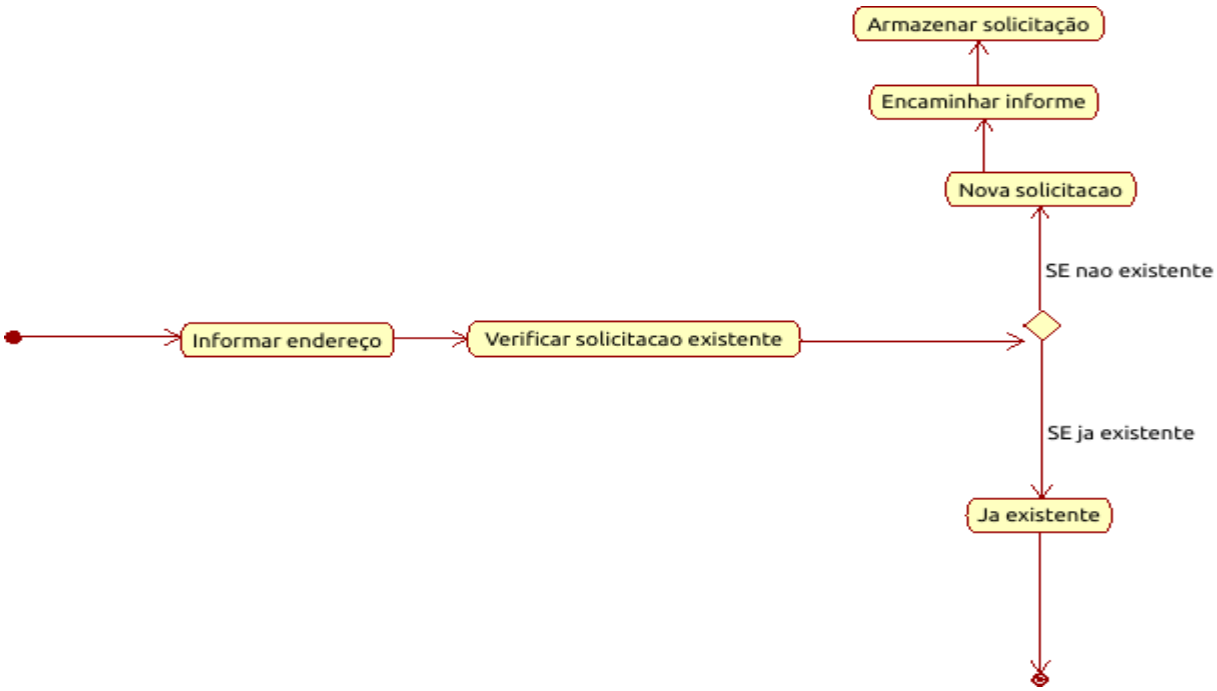


Figura 10. Diagrama de Estados - Informar Vazamento

- Solicitar Atendimento

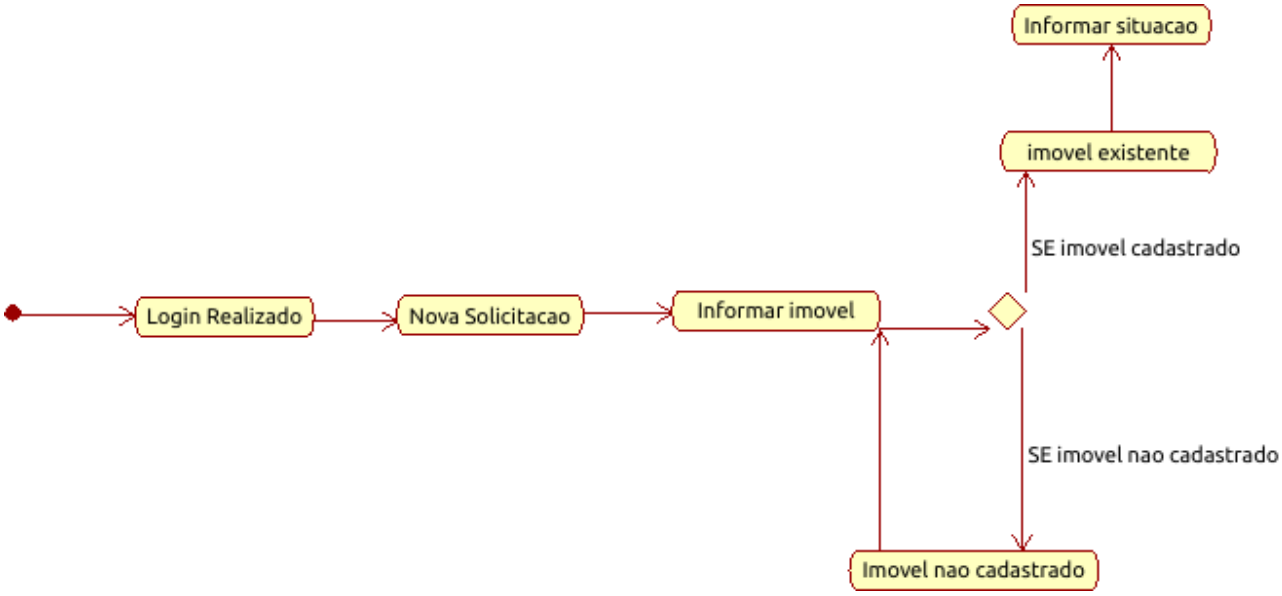


Figura 11. Diagrama de Estado - Solicitar Atendimento

3. IMPLEMENTAÇÕES – POLIMORFISMO, DESIGN PATTERNS, CONTROLE DE ERROS E INTERFACE PARA INTEGRAÇÃO

Para a implementação do código foi implementado, um design pattern, controle de erros e o polimorfismo, como solicitado pelo professor. O design pattern escolhido foi o **Singleton**, pois, após uma análise, ele foi essencial para a implementação do projeto. Ele foi implementado nas classes **Cliente** e **Controller**, fazendo com que durante todo o processo de execução do software o objeto das duas classes fossem o mesmo, exceto pela classe cliente, onde, caso o cliente escolhesse "sair" da sua conta, o objeto deixava de existir (voltando a assumir o valor **nulo**). Segue o trecho de código tanto da classe **Cliente** quanto da classe **Controller**.

```
5   public class Cliente implements IUsuario {
6
7       private static Cliente cliente;
8       private String nome;
9       private ArrayList<Imovel> imoveis = new ArrayList<>();
10      private Login login;
11      private String email;
12      private int telefoneResidencial;
13      private int telefoneCelular;
14      private ContaBancaria contaBancaria = new ContaBancaria();
15
16      public static Cliente iniciar(){
17          if (cliente == null)
18              cliente = new Cliente();
19          return cliente;
20      }
21      public static void fechar(){
22          cliente = null;
23      }
24      public static void setCliente(Cliente user){
25          cliente = user;
26      }
27      public static Cliente getCliente(){
28          return cliente;
29      }
```

Figura 12. Singleton – Cliente

```
5   public class Controller {
6       private Model model;
7       private Solicitacao solicitacao;
8       private static Controller controller;
9
10      public static Controller iniciar(){
11          if (controller == null)
12              controller = new Controller();
13          return controller;
14      }
15
16      private Controller(){
17          IUsuario funcionario = new Funcionario(nome:"Administrador");
18          Login loginFuncionario = new Login(idUser: "admin", senha: "admin");
19          funcionario.setLogin(login:loginFuncionario);
20          this.model = new Model();
21          this.model.addUsuario(usuario: funcionario);
22      }
```

Figura 13. Singleton – Controller

Vale ressaltar que estes são os principais usos de Singleton dentro do código do software, porém há outros casos de implementação de Singleton em janelas.

O controle de erros implementado foi para identificar letras em campos onde é necessário inserir somente números, prevenindo assim eventuais erros em tempo de execução, o método que implementa este controle de erros foi implementado na classe **Controller**, onde é recebido como parâmetro a **String** a ser "convertida" para inteiro, e caso ocorresse a exceção seria retornado o valor "-999". Segue o trecho de código:

```
23      public int validaNumeros(String numero){
24          int num;
25          try{
26              num = Integer.parseInt(s:numero);
27          }catch(NumberFormatException e){
28              num = -999;
29              System.out.println(x:e.getMessage());
30          }
31          return num;
32      }
```

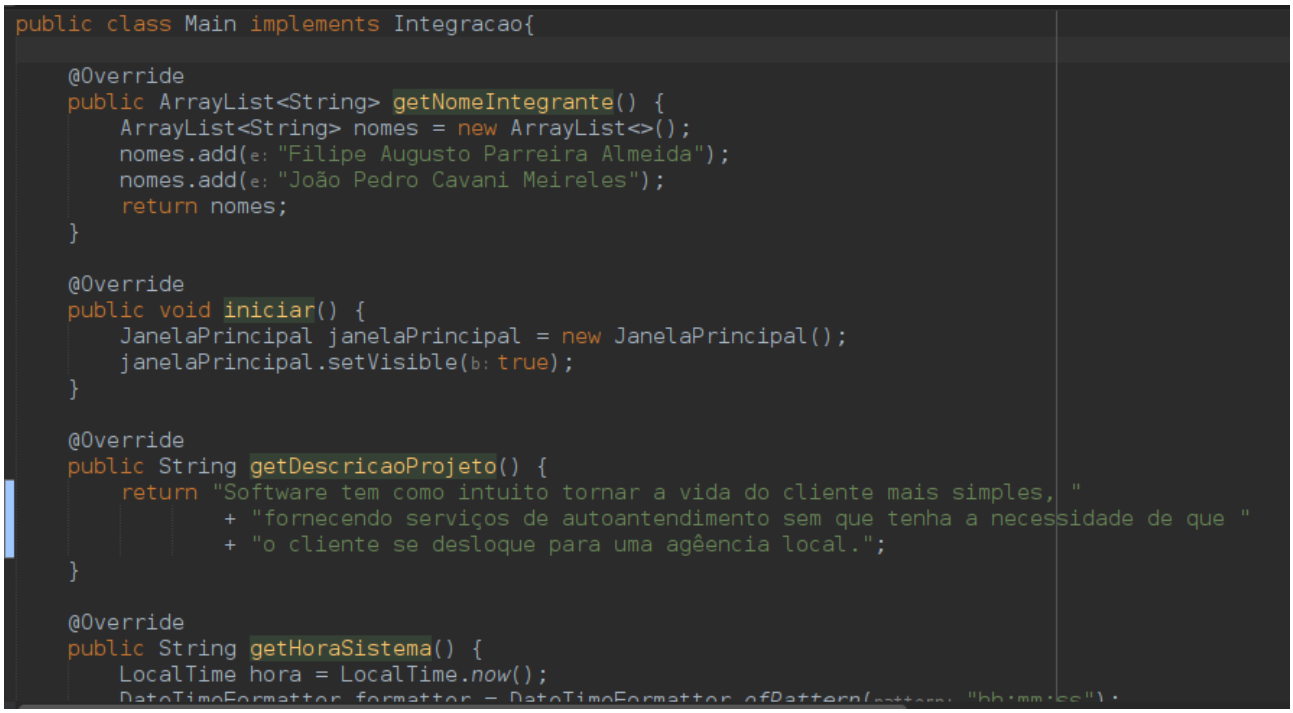
Figura 14. Controle de Erros

O polimorfismo foi implementado na **Janela Principal,** onde, ao usuário realizar o login é detectado se ele é um **Cliente** ou **Funcionário**, dependendo do tipo, é aberto uma janela diferente para cada um. Segue o trecho de código da implementação:

```
155            if (this.controller.validaLogin(login)){
156                int posiUser = this.controller.buscarUser(login);
157                IUsuario user = this.controller.getUsuario(indice: posiUser);
158
159                if (user.getTipo().equalsIgnoreCase(anotherString: "Funcionario")){
160                    paginas.show(parent: this.painelPrincipal, name: "telaServFuncionario");
161                } else if (user.getTipo().equalsIgnoreCase(anotherString: "Cliente")){
162                    cliente = (Cliente) user;
163                    Cliente.setCliente(user: cliente);
164                    this.painelPrincipal.add(comp: TelaInfos.iniciar(layout: paginas), constraints: "telaInfos");
165                    this.painelPrincipal.add(new TelaCadastroImovel(paginas), constraints: "telaCadastroImovel");
166                    this.painelPrincipal.add(new TelaVazamento(layout: paginas), constraints: "telaVazamento");
167                    paginas.show(parent: this.painelPrincipal, name: "telaServicos");
168                }
169            }else {
170                JOptionPane.showMessageDialog(parentComponent: this, message: "Usuário e/ou senha incorretos", title: "Logi
171            }
```

Figura 15. Polimorfismo

Após uma reunião com o grupo de integração, foi decidido acrescentar uma classe **Main** que implementa a interface passada por eles, sendo assim foi criado esta classe, em que todos os métodos pedidos são implementados, e em específico o método iniciar, é chamado a classe **Janela Principal** para inicializar o programa. Segue grande parte da classe **Main** para demostrar a implementação:



```
public class Main implements Integracao{

    @Override
    public ArrayList<String> getNomeIntegrante() {
        ArrayList<String> nomes = new ArrayList<>();
        nomes.add(e: "Filipe Augusto Parreira Almeida");
        nomes.add(e: "João Pedro Cavani Meireles");
        return nomes;
    }

    @Override
    public void iniciar() {
        JanelaPrincipal janelaPrincipal = new JanelaPrincipal();
        janelaPrincipal.setVisible(b: true);
    }

    @Override
    public String getDescricaoProjeto() {
        return "Software tem como intuito tornar a vida do cliente mais simples, "
                + "fornecendo serviços de autoantendimento sem que tenha a necessidade de que "
                + "o cliente se desloque para uma agêencia local.";
    }

    @Override
    public String getHoraSistema() {
        LocalTime hora = LocalTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern: "hh:mm:ss");
```

Figura 16. Classe Main - Integração

Segue, o link para acesso ao projeto completo, via GitHub:

filipeoioi/ProjetoFinalPOO: Projeto final da disciplina de Programação Orientada à Objeto, na qual é projetado um software de um Sistema de Gerenciamento de Água e Esgoto, para uma prefeitura. (github.com)

4. REFERÊNCIAS BIBLIOGRÁFICAS

G. Booch, J. Rumbaugh, I. Jacobson. UML, Guia do Usuário. Editora Campus, 2000.

H. M. Deitel, P. J. Deitel. Java: Como Programar, 8a. Edição. Pearson, 2010