

# DeskConnect<sup>+</sup>

Amanda Alves (aag)

Anna Luiza Caraciolo (alcaf)

Dara Vasconcelos (dcsv)

Douglas Ventura (dvsf)



# **Parte 1**

## **Correções do**

### **DeskConnect**

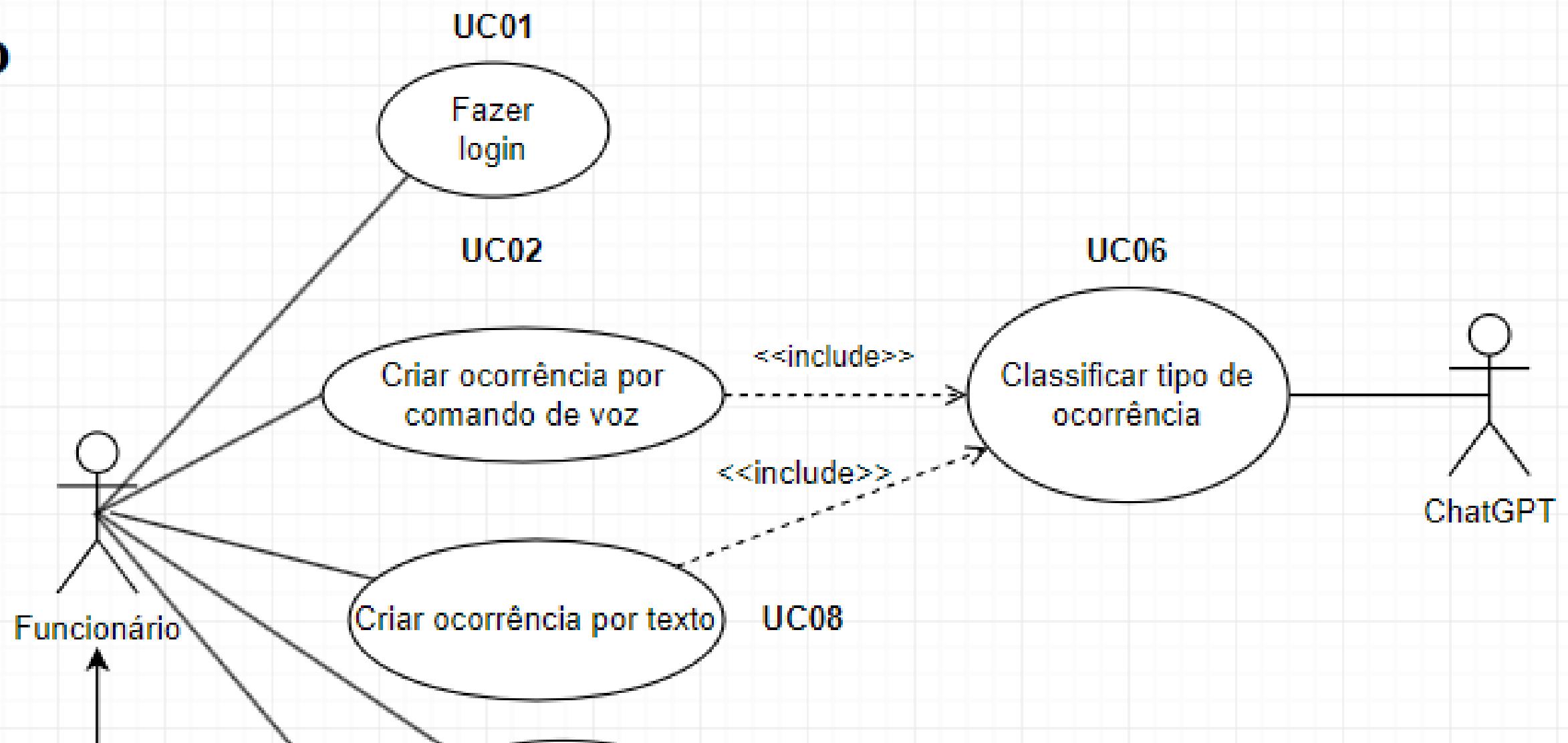
#### **monolito**

- Diagramas
- Código

# Diagrams

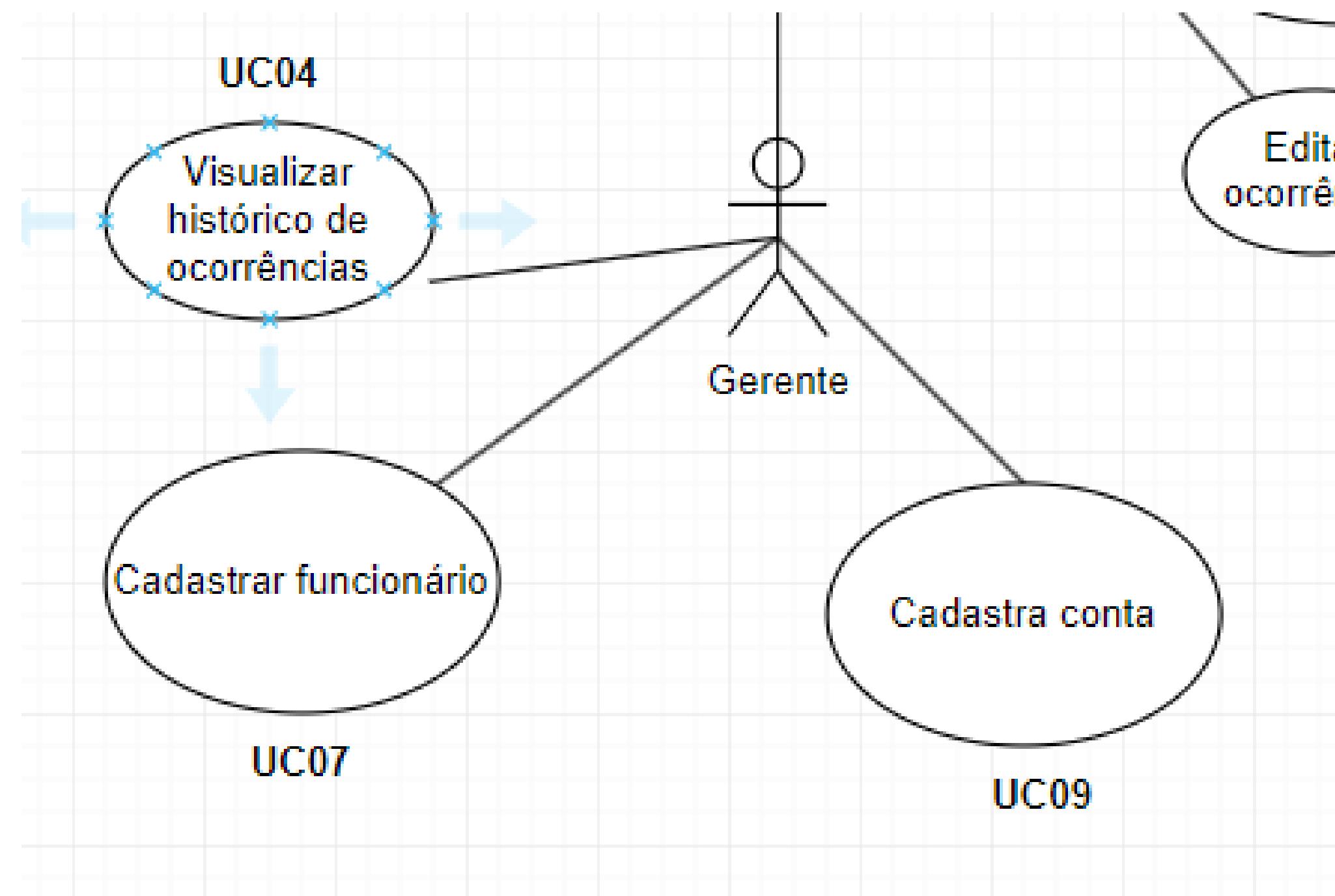
- ✓ novo caso de uso: Criar Cartão por Texto
- ✓ O caso de uso Classificar Ocorrência deve estar ligado ao ChatGPT

## Diagrama de caso de uso



# Diagrams

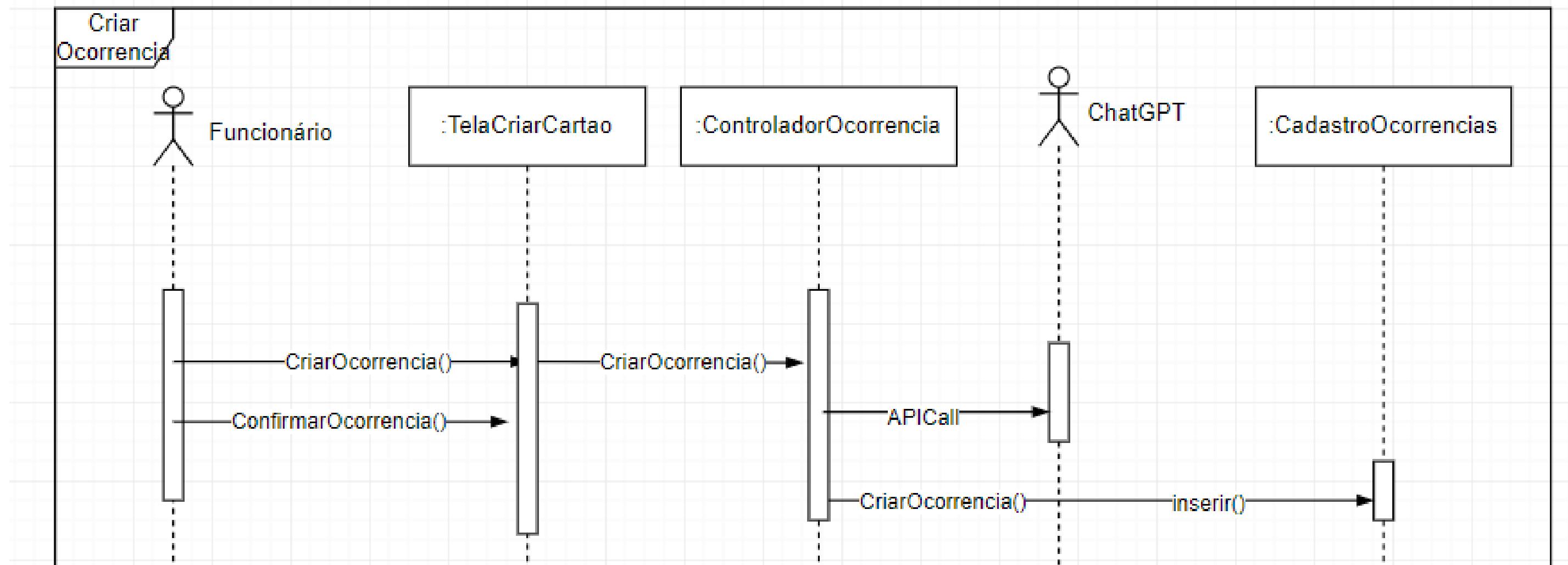
✓ Adicionei o caso de uso 09 para criação de conta



# Diagrams

- ✓ Estão faltando as fronteiras em vários diagramas de sequência

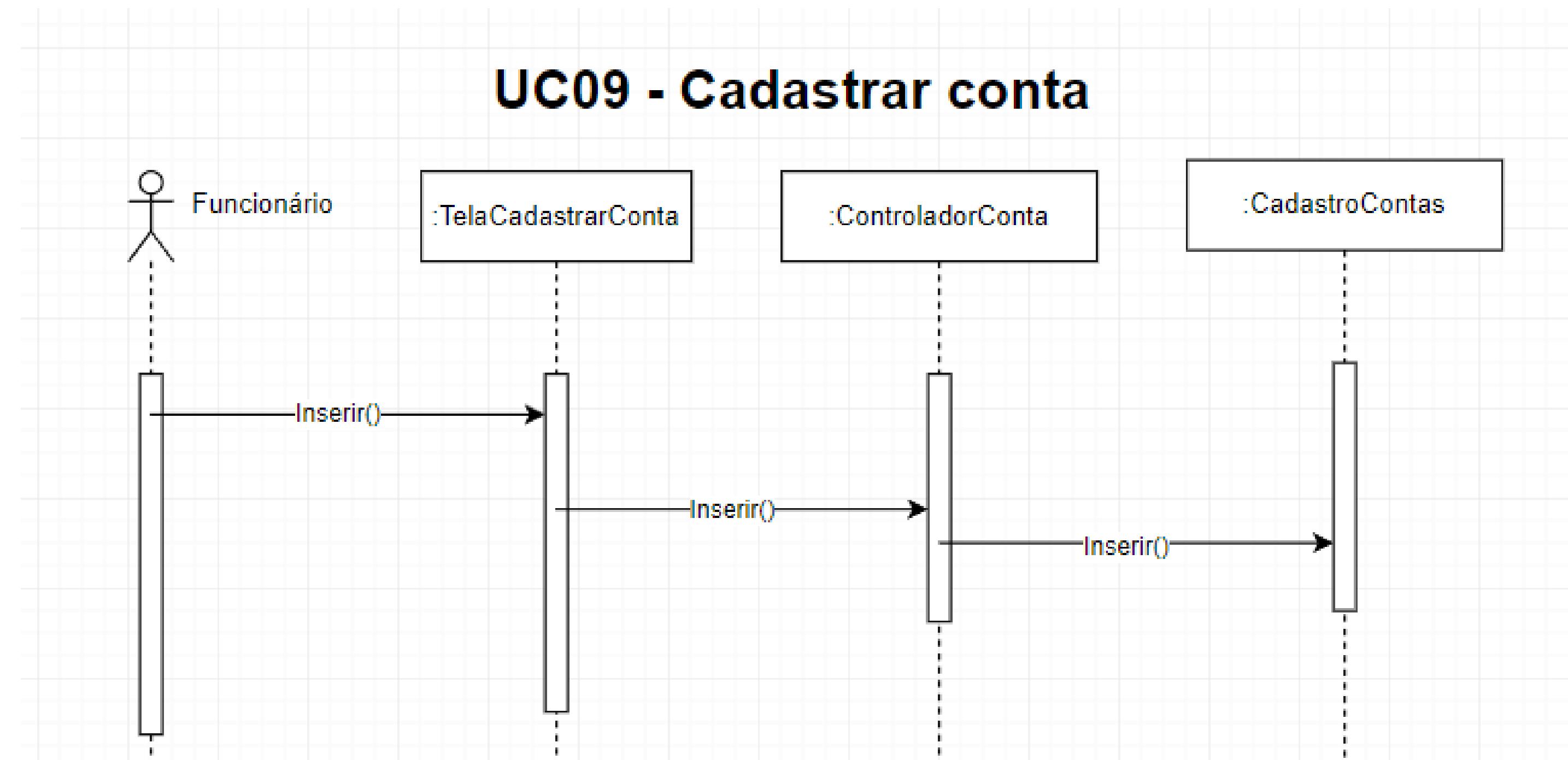
## UC07 - Criar cartão por texto



# Diagrams

- ✓ Estão faltando as fronteiras em vários diagramas de sequência

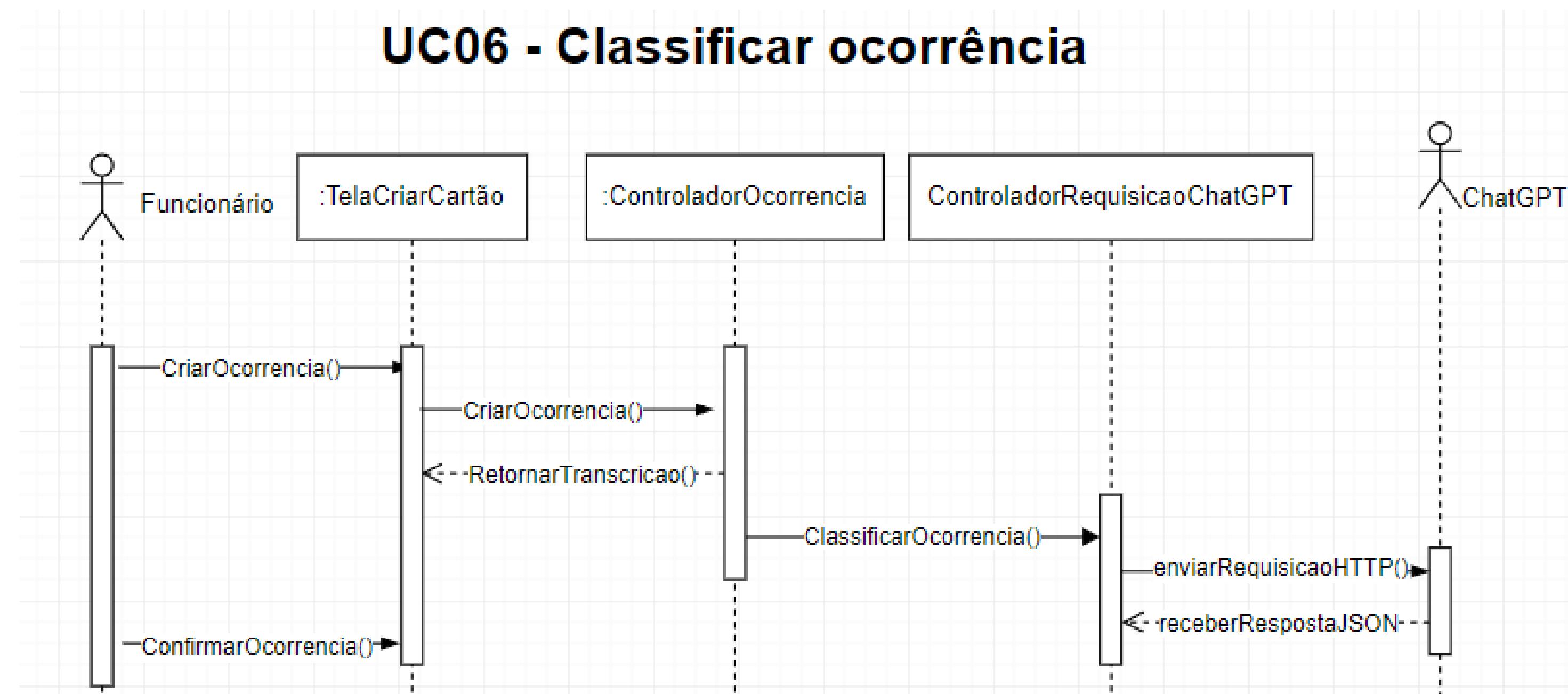
## UC09 - Cadastrar conta



# Diagrams

- ✓ Estão faltando as fronteiras em vários diagramas de sequência

## UC06 - Classificar ocorrência



# Persistência

← → C ⓘ localhost:8080/nova\_ocorrencia

## Criar uma nova ocorrência

Descrição:

Data de Criação:

Data de Início:

Data de Fim:

Setor:

Responsável:



← → C ⓘ localhost:8080/ocorrencia

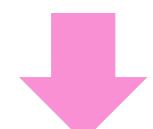
Descrição	Data de criação	Data de finalização	Setor
Teste de criação de ocorrencia	18/12/2023		Manutençao

← → C ⓘ localhost:8080/nova\_conta

## Cadastro de novo usuário

Login

Senha



← → C ⓘ localhost:8080/conta

## Contas cadastradas

Login	Senha
12345	senha

# Implementação

## Padrão observer

```
Observer.java ×  
code > deskconnect > src > main > java > com > grupo11 > aps > deskconnect  
1 package com.grupo11.aps.deskconnect.negocio;  
2  
3 public interface Observer {  
4     void addOccurrence(Ocorrencia ocorrencia);  
5  
6     void removeOccurrence(Ocorrencia ocorrencia);  
7 }  
8
```

```
@Entity  
public class Funcionario implements Observer {  
    private String nome;  
    @Id  
    private String cpf;  
    private String funcao;  
    private Double salario;  
  
    private String setor;  
    @ManyToMany  
    private List<Ocorrencia> subjects;  
  
    @Override  
    public void addOccurrence(Ocorrencia ocorrencia) {  
        if (subjects == null) {  
            subjects = new ArrayList<>();  
        }  
        subjects.add(ocorrencia);  
    }  
  
    @Override  
    public void removeOccurrence(Ocorrencia ocorrencia) {  
        if (subjects != null) {  
            subjects.remove(ocorrencia);  
        }  
    }  
}
```

# Implementação

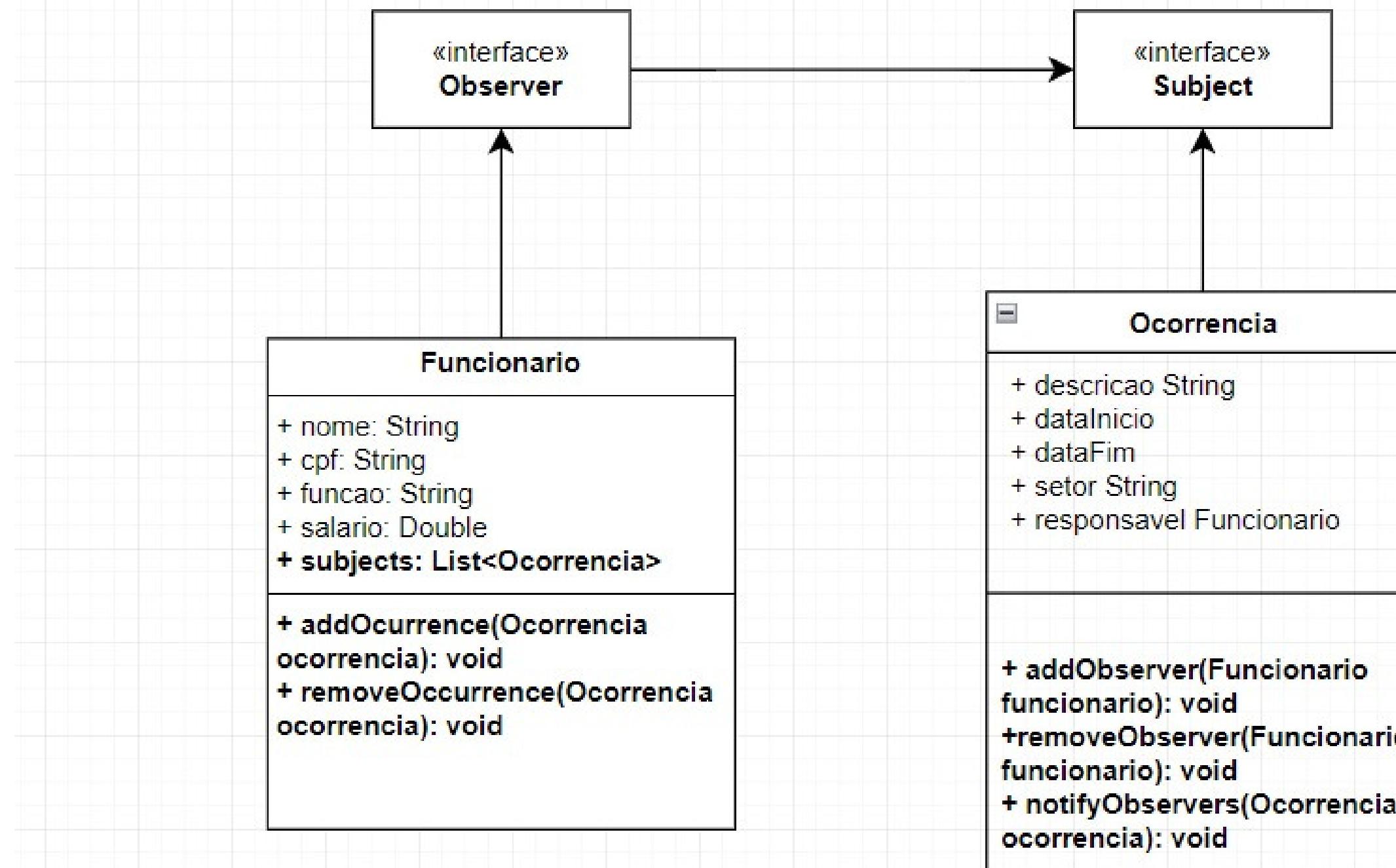
## Padrão observer

```
public interface Subject {  
    void addObserver(Funcionario funcionario);  
    void removeObserver(Funcionario funcionario);  
    void notifyObservers(Ocorrencia ocorrencia);  
}
```

```
@Entity  
public class Ocorrencia implements Subject {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    private String descricao;  
    private String dataInicio;  
    private String dataFim;  
    private String dataCriacao;  
    private String setor;  
  
    @Override  
    public void addObserver(Funcionario funcionario) {  
        observers.add(funcionario);  
        funcionario.addOccurrence(this);  
    }  
  
    @Override  
    public void removeObserver(Funcionario funcionario) {  
        observers.remove(funcionario);  
        funcionario.removeOccurrence(this);  
    }  
  
    @Override  
    public void notifyObservers(Ocorrencia ocorrencia) {  
        // chamar na criação da ocorrência  
        for (Funcionario observer : observers) {  
            observer.addOccurrence(ocorrencia);  
        }  
    }  
}
```

# Padrões de projeto

Observer



# Implementação

## Padrão factory



The screenshot shows a Java code editor with two tabs: `FuncionarioFactory.java` and `FuncionarioController.java`. The `FuncionarioFactory.java` tab is active, displaying the following code:

```
1 package com.grupo11.aps.deskconnect.negocio;
2
3 public class FuncionarioFactory {
4     public Funcionario createFuncionario(String tipo, String nome, String cpf, String funcao, Double salario, String setor) {
5         if ("Gerente".equals(tipo)) {
6             return new Gerente(nome, cpf, funcao, salario, setor);
7         } else if ("FuncionarioComum".equals(tipo)) {
8             return new FuncionarioComum(nome, cpf, funcao, salario, setor);
9         } else {
10             throw new IllegalArgumentException("Invalid employee type");
11         }
12     }
13 }
```

The code implements a factory method `createFuncionario` that takes parameters for employee type, name, CPF, function, salary, and department. It uses an if-else block to create either a `Gerente` or a `FuncionarioComum` object based on the type. If an invalid type is provided, it throws an `IllegalArgumentException`.

# Implementação

## Padrão factory



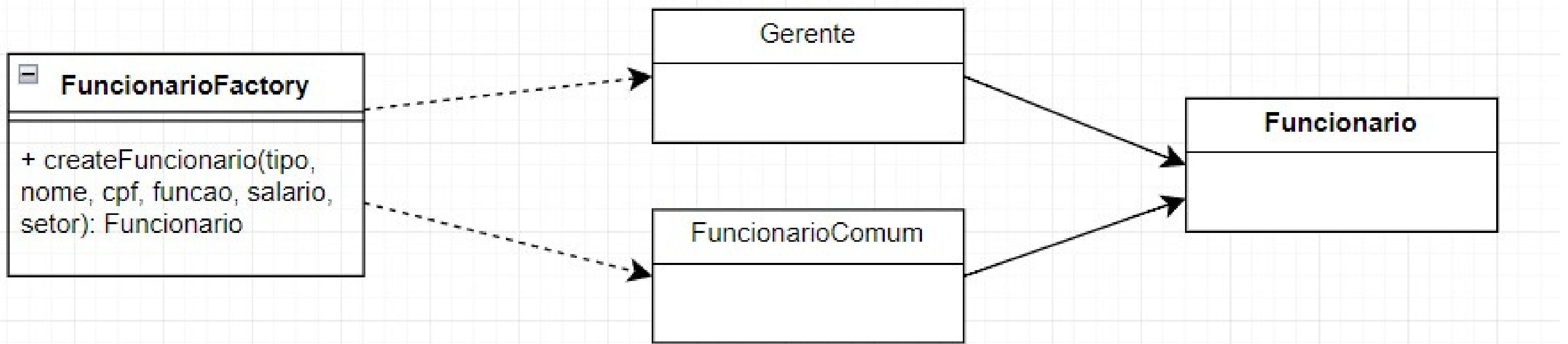
The screenshot shows a Java code editor with two tabs: `FuncionarioController.java` and `FuncionarioFactory.java`. The `FuncionarioController.java` tab is active, displaying the following code:

```
code > deskconnect > src > main > java > com > grupo11 > aps > deskconnect > comunicacao > FuncionarioController.java > {} com.grupo11.aps.deskconnect.comunicacao
```

```
15  @Controller
16  public class FuncionarioController {
17
18      @Autowired
19      Fachada fachada;
20
21      @RequestMapping("/novo_funcionario")
22      public String mostrarFormulaoInserirFuncionario(Model model) {
23          model.addAttribute("funcionario", new Funcionario());
24          return "createFuncionario";
25      }
26      @PostMapping("/inserir_funcionario")
27      public String inserirFuncionario(@ModelAttribute("funcionario") Funcionario funcionario, @RequestParam("employeeType") String employeeType) {
28          FuncionarioFactory factory = new FuncionarioFactory();
29          Funcionario createdFuncionario = factory.createFuncionario(employeeType,
30              funcionario.getNome(),
31              funcionario.getCpf(),
32              funcionario.getFuncao(),
33              funcionario.getSalario(),
34              funcionario.getSetor()
35          );
36          fachada.inserirFuncionario(createdFuncionario);
37          return "successPage";
38      }
39  }
```

The `FuncionarioFactory` class is highlighted in orange, indicating it is being used or defined in the current file.

## Factory method



# Parte 2

## DeskConnect SOA

- **Especificação do modelo de negócios**
  - Modelo de informação de negócio
  - Modelo navegacional
  - Protótipo de interface gráfica
- **Análise de serviços**
  - Arquitetura de serviços
  - Modelo de interação dos serviços
  - Modelo de informação refinado
  - Diagrama de componentes de serviços (nível de análise)
- **Projetar serviços**
  - Diagrama de componentes do sistema (com padrões arquiteturais)

# Linha do Tempo

[Voltar ao índice](#)



# Protótipo de interface gráfica

 Figma

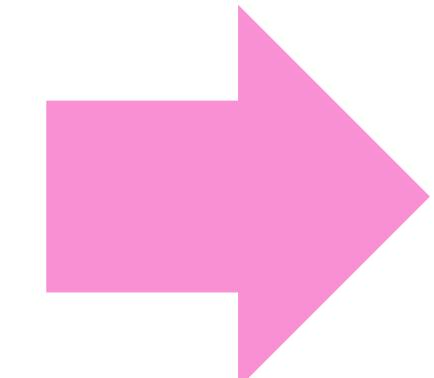
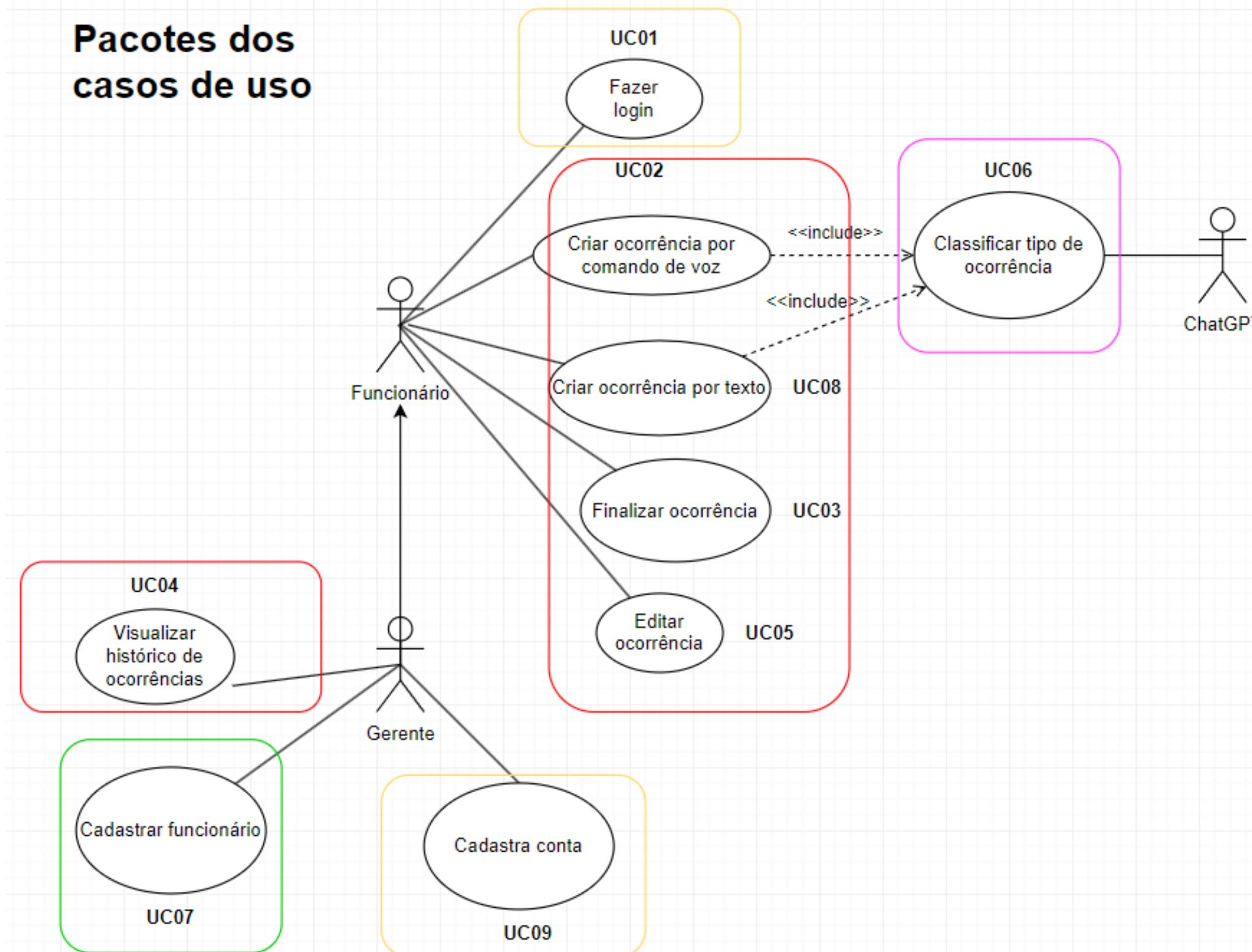
# Linha do Tempo

[Voltar ao índice](#)



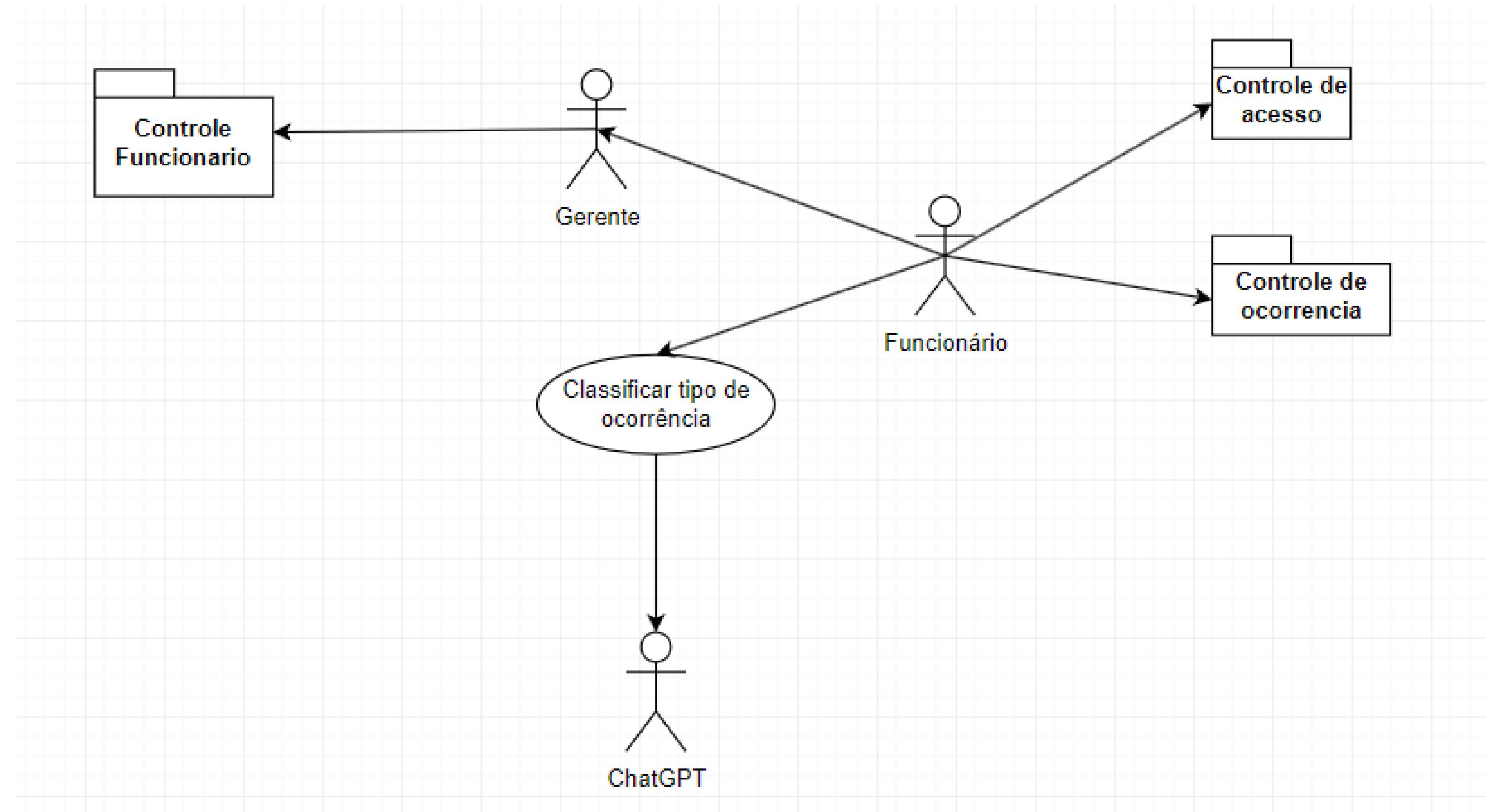
# Empacotar casos de uso

## Pacotes dos casos de uso



# Empacotar casos de uso

[Voltar ao índice](#)

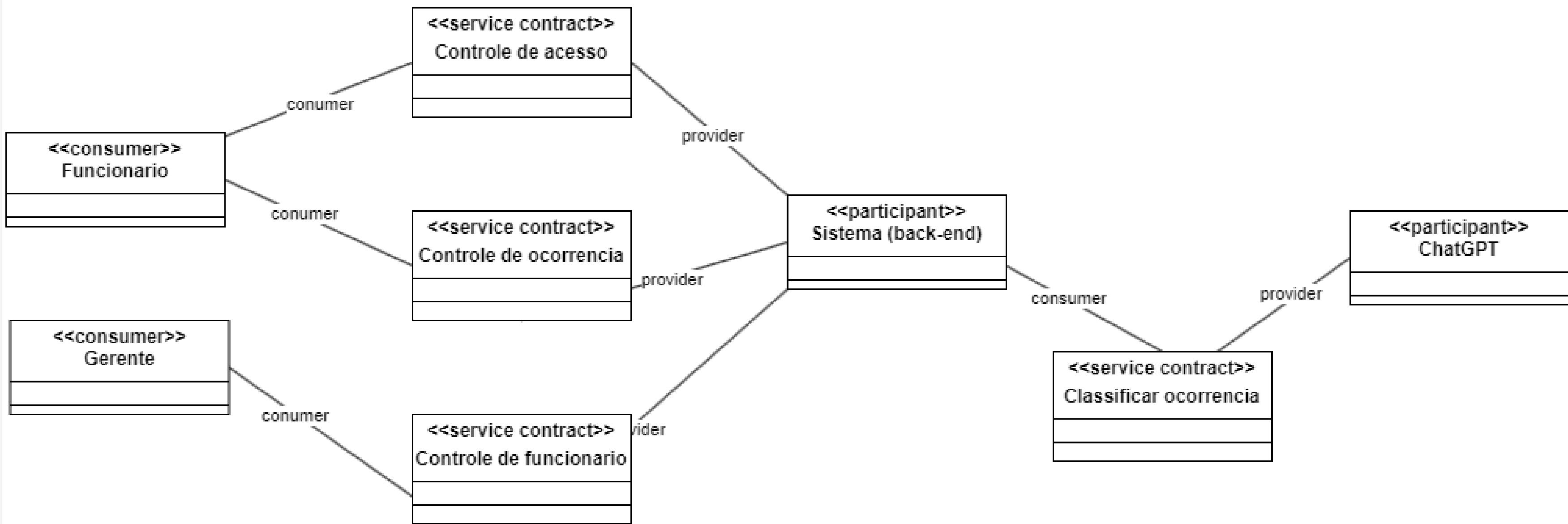


# Arquitetura de serviços

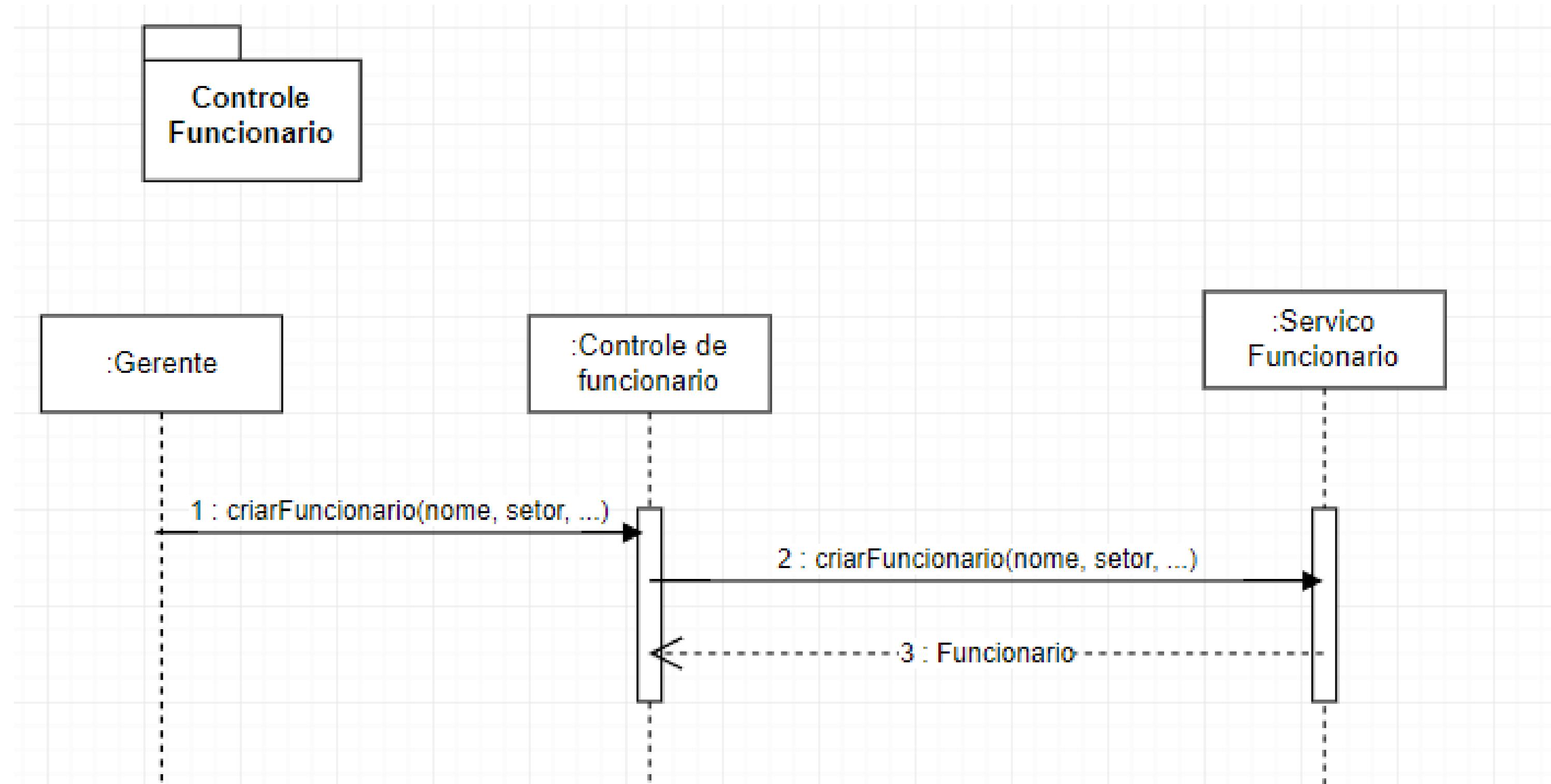
[Voltar ao índice](#)

## Arquitetura de serviço

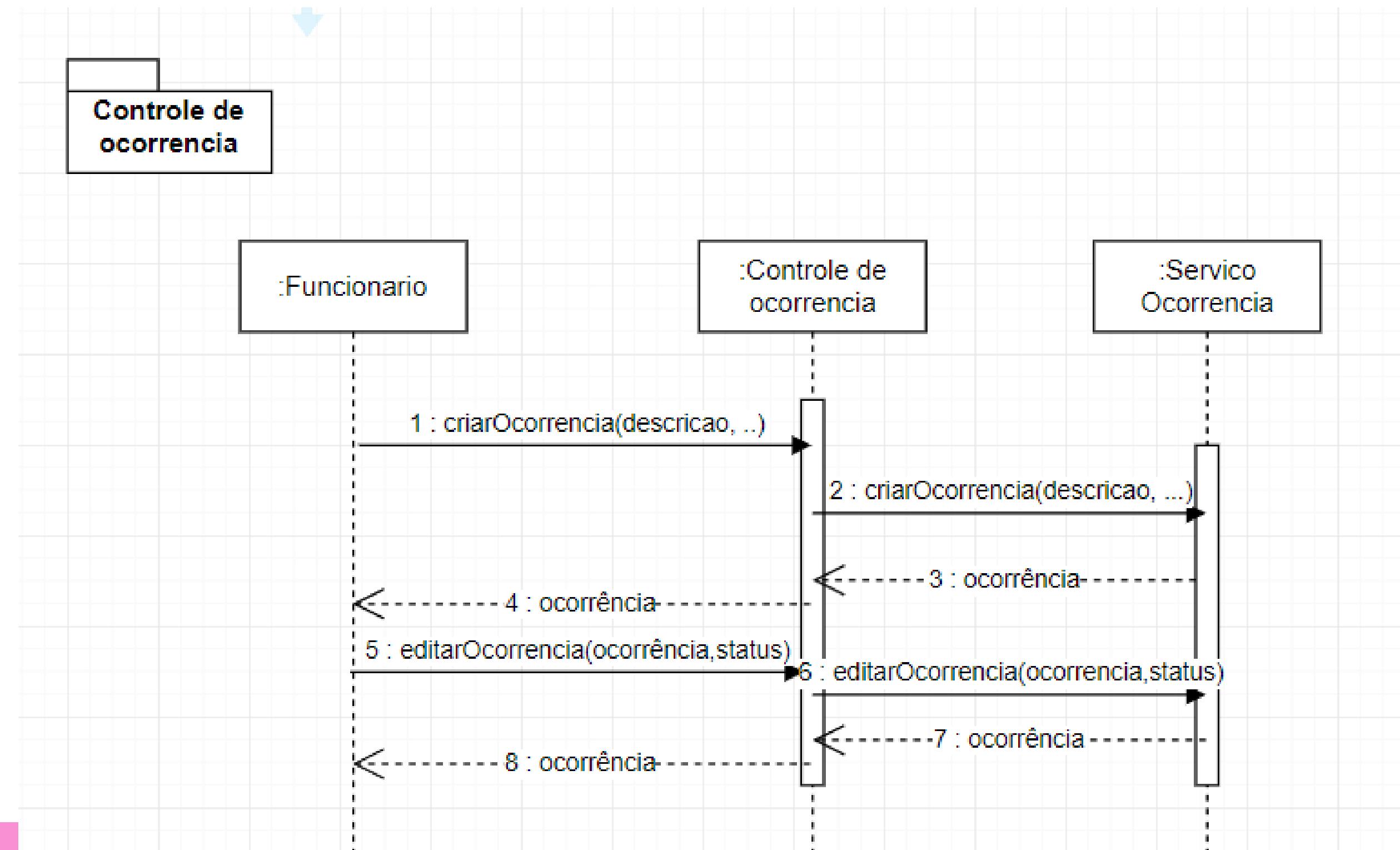
Subsistemas dão origem a componentes que implementam serviços de controle  
Controladores dão origem a componentes que implementam serviços de controle



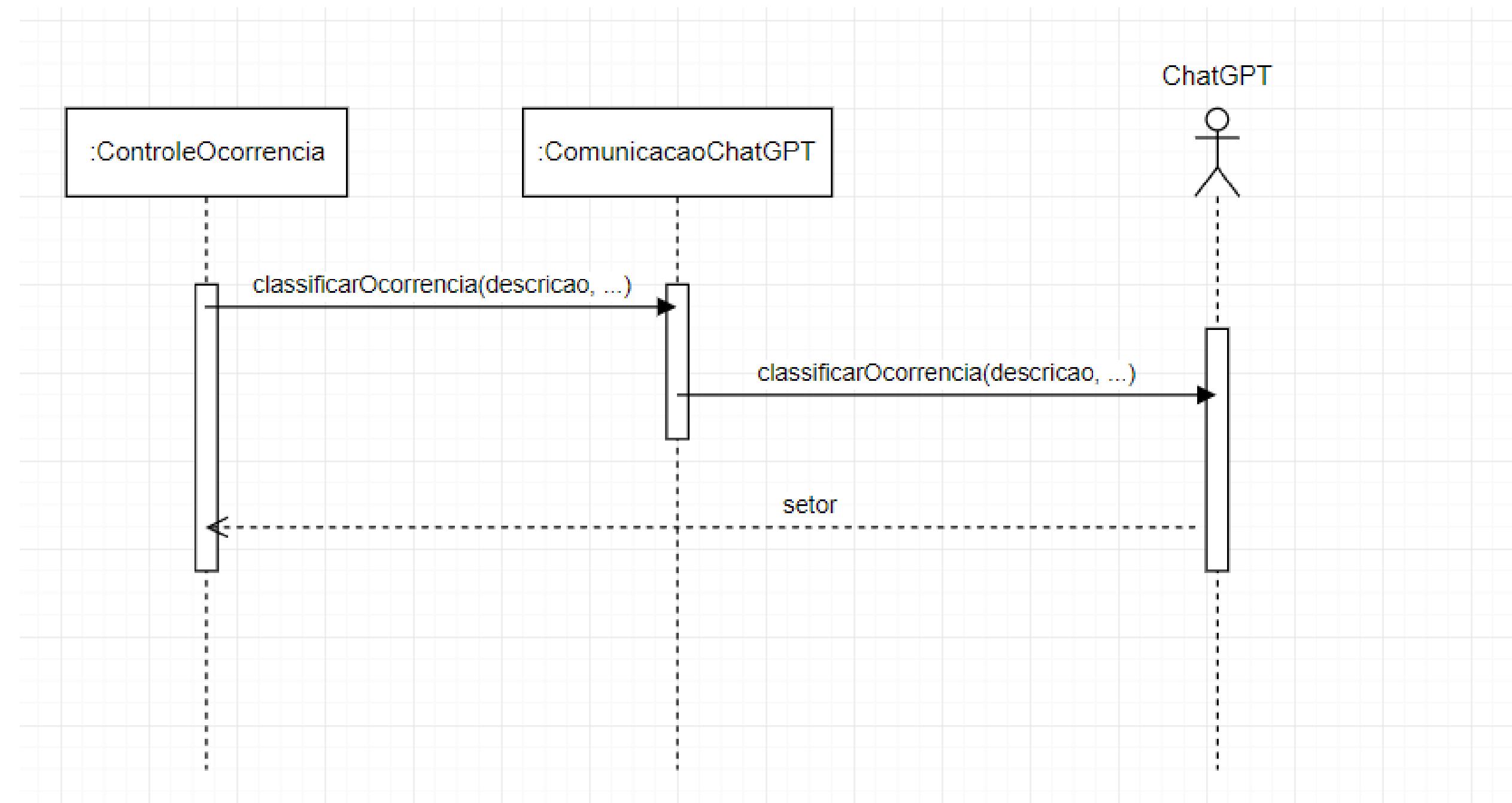
# Modelo de interação de serviços



# Modelo de interação de serviços

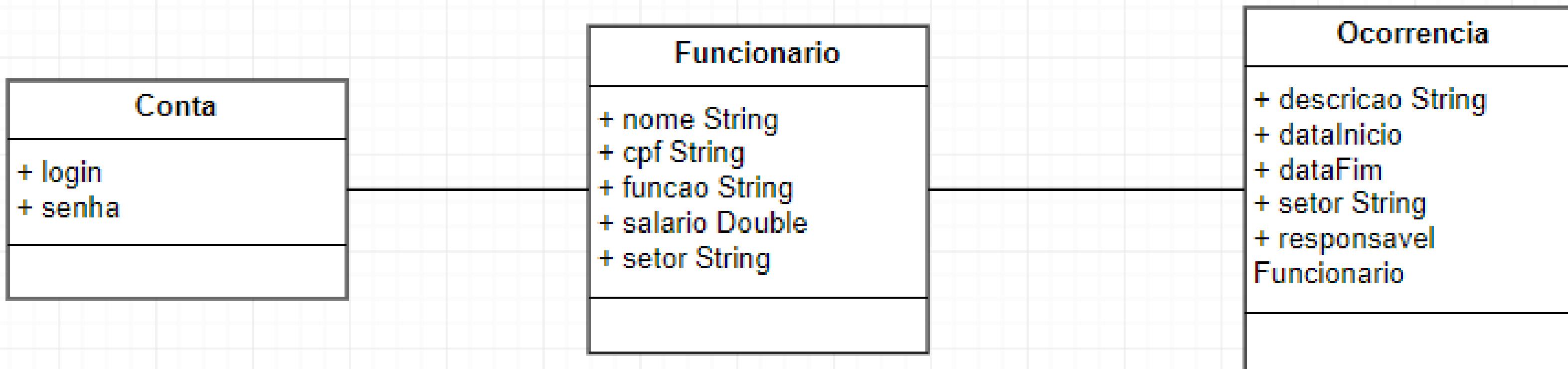


# Modelo de interação de serviços

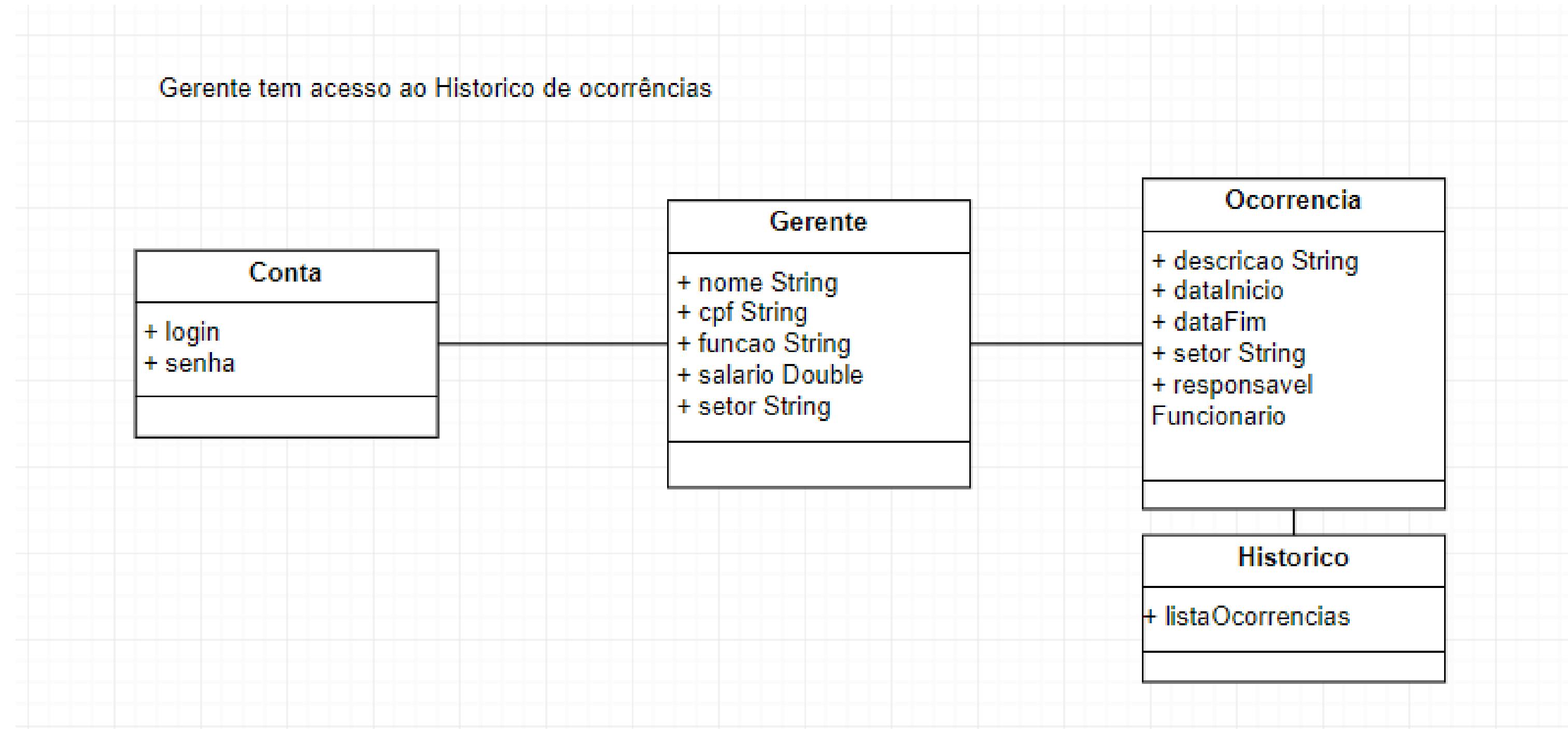


# Modelo de informação refinado

Funcionario tem acesso ao controle de ocorrencia

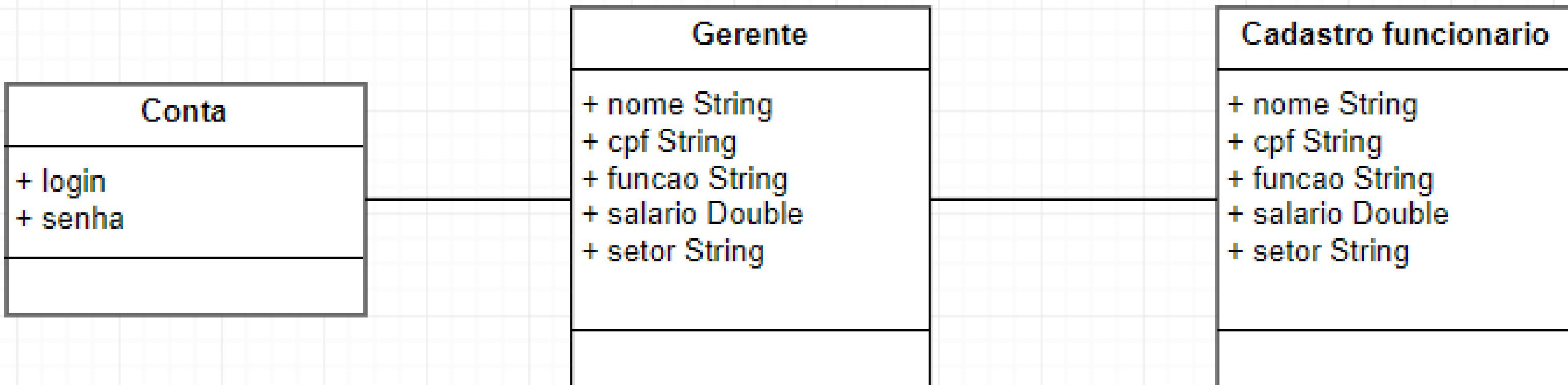


# Modelo de informação refinado

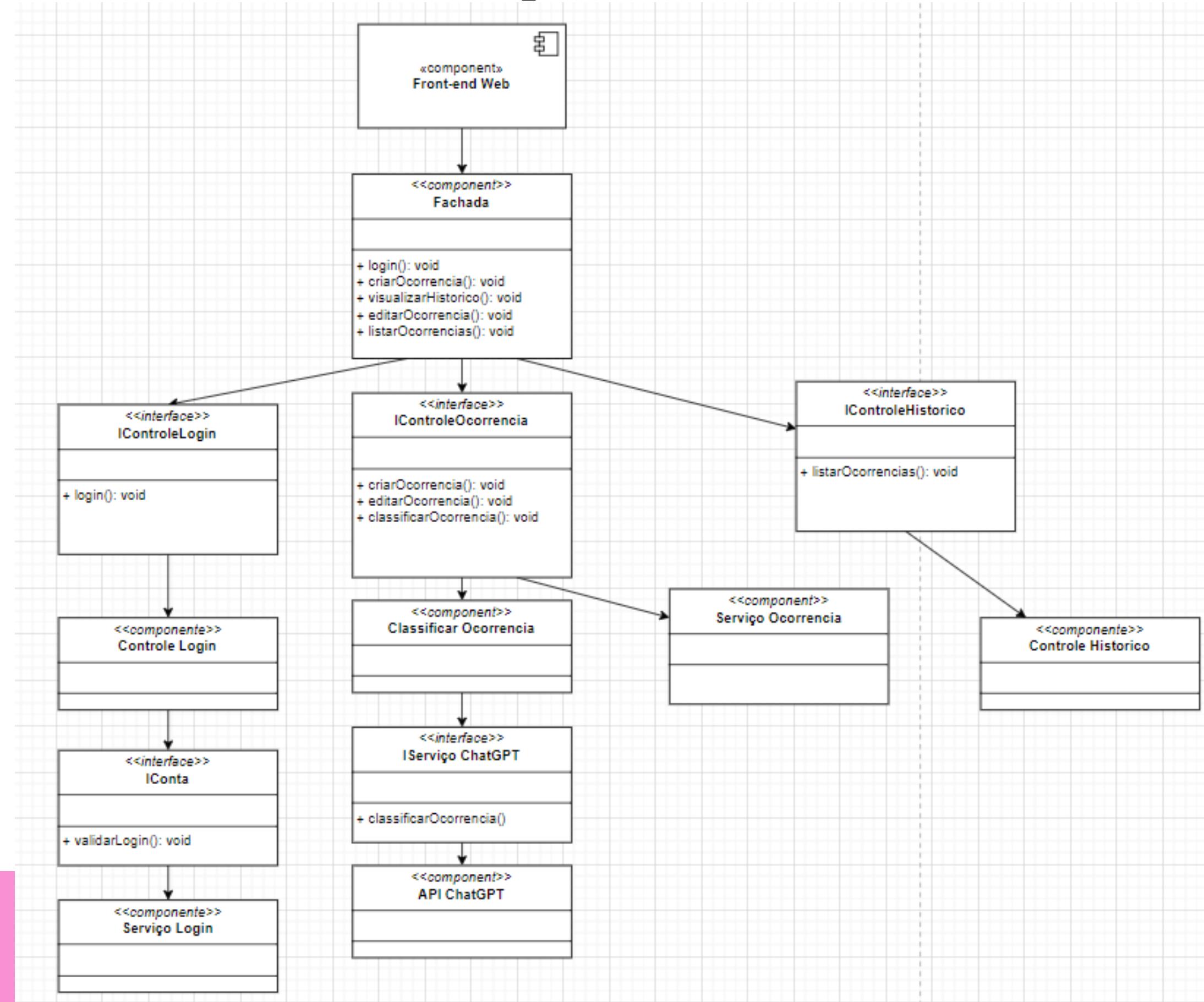


# Modelo de informação refinado

Gerente tem permissão de cadastrar funcionários

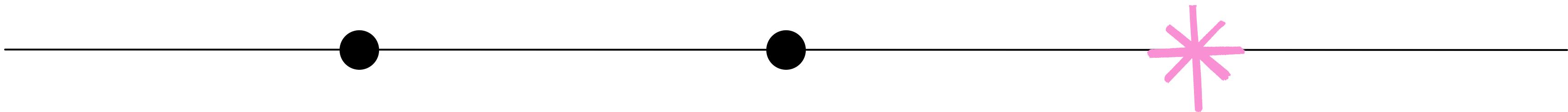


# Diagrama de componentes do sistema



# Linha do Tempo

[Voltar ao índice](#)

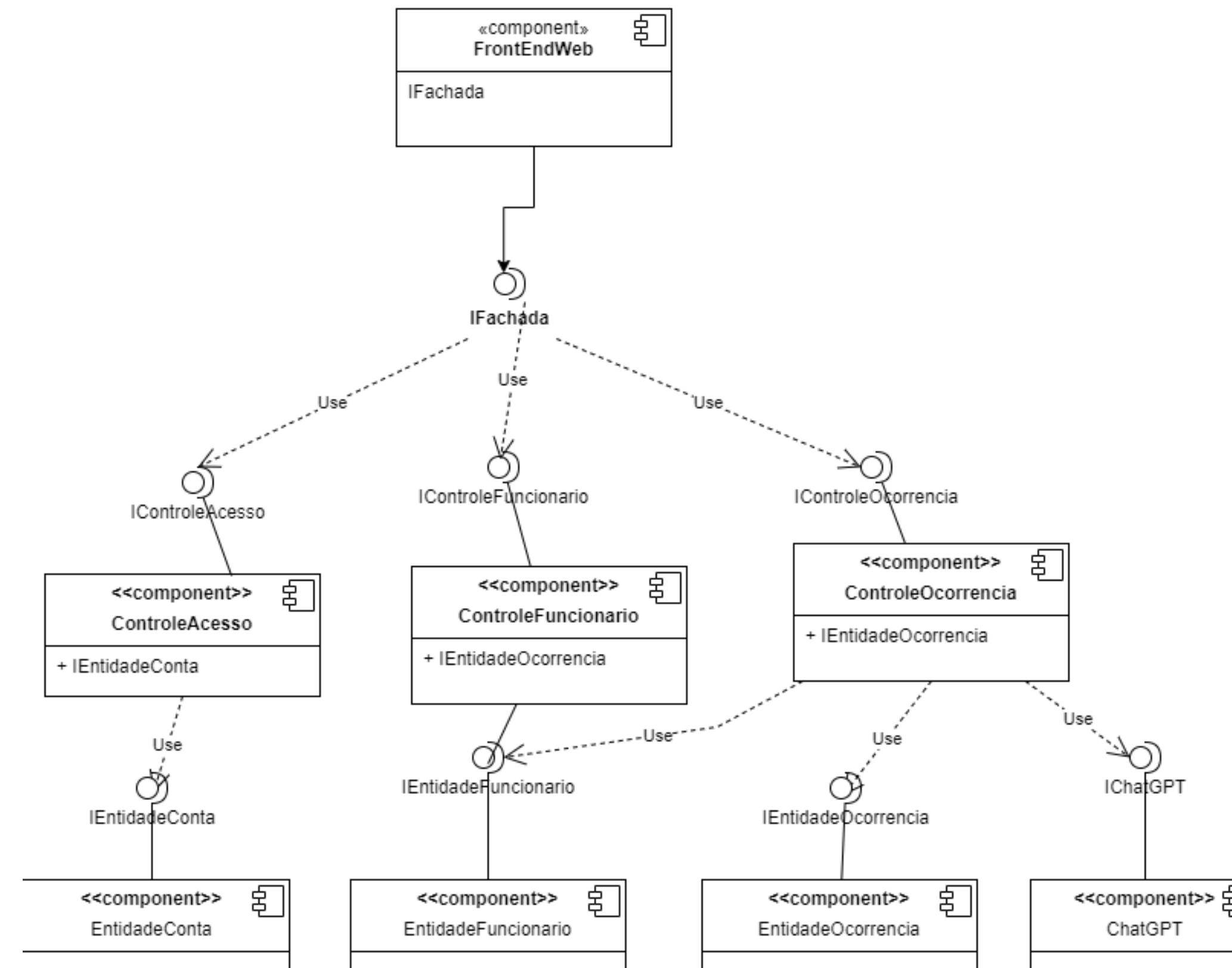


**ESPECIFICAÇÃO  
DO MODELO DE  
NEGÓCIOS**

**ANÁLISE DE  
SERVIÇOS**

**PROJETAR  
SERVIÇOS**

# Componentes de serviço



[Voltar ao índice](#)