Trabalho Integrado dos componentes de Banco de Dados II, Engenharia de Software I e Programação II

Autores: Davi Ribeiro, Eduardo Ghiel, Eduardo Tessaro e Maick Tonet
Instituição: Unoesc São Miguel do Oeste

Introdução

O gerenciamento de chamados representa uma atividade central para a eficácia operacional de qualquer empresa ou organização que ofereça produtos ou serviços. Este processo abrange o registro, acompanhamento e resolução de problemas e solicitações provenientes de clientes, fornecedores e colaboradores.

A implementação de um sistema de gerenciamento de chamados eficiente é crucial para aprimorar a satisfação do cliente, otimizar o tempo de resolução de problemas e aumentar a produtividade dos agentes de atendimento.

Objetivos do Projeto

O objetivo principal deste projeto foi desenvolver uma aplicação de gerenciamento de chamados que fosse eficiente e de fácil utilização. Para atingir esse propósito, foram implementadas as principais funcionalidades de um sistema de gerenciamento de chamados, incluindo:

Registro de Chamados: Possibilita a inserção e documentação de novos chamados de forma organizada e detalhada.

Acompanhamento de Chamados: Permite o monitoramento em tempo real do status e andamento de cada chamado, proporcionando uma visão abrangente do fluxo de trabalho.

Resolução de Chamados: Facilita o processo de solução, registrando as ações tomadas para resolver cada chamado e garantindo um histórico transparente.

Relatórios: Oferece a geração de relatórios analíticos, fornecendo insights valiosos para a tomada de decisões e melhoria contínua do processo.

Principais Requisitos Funcionais

1. **Registro de Chamados:**
   Funcionários devem ser capazes de registrar novos chamados, fornecendo informações detalhadas, incluindo descrição do problema, data e hora da abertura, usuário responsável, tipo e prioridade do chamado.

2. **Atribuição Automática de Chamados:**
   O sistema deve atribuir automaticamente um técnico disponível com base na disponibilidade, expertise e prioridade do chamado.

3. **Categorização de Chamados por Urgência:**
   Os usuários devem especificar a urgência do chamado, que será categorizado como baixa, média ou alta.

4. **Encaminhamento de Chamados Não Resolvidos:**
   Caso um chamado não seja resolvido dentro do prazo definido, o sistema deve encaminhá-lo para outro técnico disponível, com o prazo sendo configurado pelo administrador.

5. **Consulta de Status de Chamados:**
   Usuários e técnicos devem ter acesso ao status de seus chamados, que pode ser "Aguardando Atendimento", "Em Andamento" ou "Resolvido".

6. **Comunicação entre Atendentes e Usuários:**
   Facilitar a comunicação entre atendentes e funcionários através de chat, sistema de e-mail ou outros meios convenientes.

7. **Distinção entre Chamados de Software e Hardware:**
   O sistema deve distinguir entre chamados de software e hardware para análise e definição de políticas de atendimento.

8. **Histórico de Chamados e Relatórios:**
   Manter um histórico de chamados e possibilitar a emissão de relatórios operacionais e gerenciais. Os relatórios podem incluir dados como número de chamados abertos, resolvidos, tempo médio de resolução e distribuição por tipo.

Além dos requisitos funcionais essenciais para um sistema de gerenciamento de chamados, é imperativo considerar requisitos não-funcionais que garantam a eficiência, segurança e adaptabilidade do software. Este artigo explora os requisitos não-funcionais identificados após a análise dos requisitos funcionais, destacando aspectos como usabilidade, segurança, desempenho e integração.

Requisitos Não-Funcionais

1. **Intuição e Facilidade de Uso**:
   O software deve ser intuitivo e fácil de usar, visando atender tanto a funcionários técnicos quanto não técnicos.

2. **Segurança:**
   Medidas de segurança robustas devem ser implementadas para

proteger as informações dos chamados e garantir a privacidade dos usuários.

3. **Desempenho:**
   O software deve manter um desempenho rápido, mesmo sob carga elevada de chamados em aberto.

4. **Disponibilidade:**
   O sistema deve estar disponível e acessível a funcionários e técnicos sempre que necessário, com um tempo de inatividade mínimo.

5. **Escalabilidade:**
   A arquitetura do software deve ser projetada para suportar o crescimento e o aumento da demanda por chamados ao longo do tempo.

6. **Personalização:**
   O software deve oferecer a capacidade de personalizar campos e fluxos de trabalho, adaptando-se às necessidades específicas da empresa.

7. **Integração:**
   Capacidade de integração com outros sistemas empresariais, como sistemas de gerenciamento de ativos ou ferramentas de monitoramento de rede.

Implementação do JPA para Armazenamento de Dados dos Chamados

Este segmento descreve a implementação do JPA (Java Persistence API) no projeto para armazenamento eficaz dos dados dos chamados no banco de dados PostgreSQL. Uma classe específica, denominada "Chamado", foi desenvolvida para representar cada tipo de dado a ser armazenado. Neste artigo, detalhamos os atributos da classe Chamado e os métodos essenciais implementados para a obtenção de informações específicas

Implementação da Classe Chamado

A classe Chamado foi criada com os seguintes atributos fundamentais:

- **id:** O ID único associado ao chamado.
- **descrição:** A descrição detalhada do chamado.
- **prioridade:** A prioridade atribuída ao chamado.
- **status:** O estado atual do chamado.

Métodos da Classe Chamado

A classe Chamado incorpora os seguintes métodos para acessar e recuperar informações pertinentes:

1. **getId():**
   Retorna o ID único associado ao chamado.

2. **getDescricao():**
   Retorna à descrição detalhada do chamado.

3. **getPrioridade():**
   Retorna a prioridade atribuída ao chamado.

4. **getStatus():**
   Retorna ao status atual do chamado.

Implementação do Spring Boot para Facilitar o Desenvolvimento de Aplicativos

Este segmento destaca a implementação do Spring Boot como parte integral do projeto, proporcionando uma série de recursos prontos para uso. O Spring Boot é reconhecido por oferecer ferramentas que permitem aos desenvolvedores criar aplicativos de maneira rápida e eficiente. Neste artigo, focamos especificamente o uso do Spring Boot em conjunto com o JPA, destacando a facilidade que seus recursos proporcionam na configuração e utilização do JPA.

Recursos do Spring Boot

O Spring Boot oferece uma gama de recursos prontos para uso, simplificando o desenvolvimento de aplicativos. Sua integração eficiente com o JPA é especialmente destacada por meio do fornecimento de um "starter" dedicado para o

JPA. Esse starter facilita significativamente a configuração e a incorporação do JPA ao projeto, permitindo uma integração suave e eficaz.

Vantagens da Integração JPA com Spring Boot

1. **Configuração Simplificada:**
   O starter JPA do Spring Boot simplifica o processo de configuração, eliminando complexidades desnecessárias.

2. **Facilidade de Uso:**
   O Spring Boot proporciona uma experiência amigável aos desenvolvedores, simplificando a implementação e o acesso aos recursos do JPA.

3. **Rápido Desenvolvimento:**
   A combinação do Spring Boot e JPA acelera o desenvolvimento do aplicativo, permitindo a criação de funcionalidades robustas com rapidez.

Implementação do PostgreSQL

Neste segmento, detalhamos a implementação do PostgreSQL como sistema de gerenciamento de banco de dados para o projeto. A escolha do PostgreSQL foi fundamentada em conhecimentos adquiridos nas aulas de Banco de Dados I e II, onde a familiaridade com a plataforma foi considerada crucial. Abaixo, apresentamos os passos seguidos para a criação do Banco de Dados, a criação da tabela para armazenar os dados dos chamados e o código utilizado para tal.

```
CREATE TABLE chamados (
    id SERIAL PRIMARY KEY,
    descricao TEXT NOT NULL,
    prioridade INTEGER NOT NULL,
    status INTEGER NOT NULL
);
```

Descrição das Colunas

1. **id (SERIAL PRIMARY KEY):**
   Representa o ID único associado a cada chamado, utilizado como chave primária.

2. **descricao (TEXT):**
   Armazena a descrição detalhada do chamado.

3. **prioridade (VARCHAR(50)):**
   Indica a prioridade atribuída ao chamado.

4. **status (VARCHAR(50)):**
   Reflete o estado atual do chamado.

O Spring Boot vai fornecer um starter para o JPA que vai facilitar a configuração e o uso do mesmo. Para mapear a classe Chamado para a tabela chamados, tivemos que adicionar as seguintes anotações à classe Chamado:

```
@Entity
@Data
public class Chamado {

    @Id
    @Column(name = "codigo_chamado", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long codigoChamado;

    @Column(name = "dataAbertura_chamado", nullable = false)
    private java.sql.Timestamp dataAberturaChamado;

    @Column(name = "status_chamado", nullable = false)
    private String statusChamado;

    @ManyToOne
    @JoinColumn(name = "cpf_usuario")
    private Usuario cpfUsuario;

    @ManyToOne
    @JoinColumn(name = "cnpj_empresa")
    private Empresa cnpjEmpresa;
}
```

A anotação @Entity indica que a classe Chamada é uma entidade JPA e que a tabela associada é chamada. As anotações @Id e @GeneratedValue indicam que a coluna id é a chave primária da tabela e que seu valor é gerado automaticamente. As anotações @Column indicam que as colunas descrição, prioridade e status são colunas da tabela.

Para que o controlador possa acessar o banco de dados, é necessário injetar o EntityManager nele. Para fazer isso, adicione a seguinte anotação ao controlador:

```
@Autowired
private EntityManager entityManager;
```

Esta anotação indica que o EntityManager será injetado no controlador automaticamente pelo Spring Boot

Os métodos do controlador devem ser implementados para realizar as operações necessárias no banco de dados, código a seguir implementa os métodos para registrar um novo chamado e obter o status de um chamado existente:

```
@GetMapping(path = "/listar")
public List<Chamado> listar() {
   return chamadoRepository.findAll();
}

@PostMapping(path = "/criar")
@ResponseBody
public ResponseEntity<Chamado> criar(@RequestBody Chamado chamado) {
   Chamado c = chamadoRepository.save(chamado);
   return new ResponseEntity<>(c, HttpStatus.CREATED);
}
```

O método criarChamado() insere um novo chamado no banco de dados. O método listarChamado() recupera um chamado existente do banco de dados.

Tabela departamento:

```
CREATE TABLE departamento (
id INT NOT NULL AUTO_INCREMENT,
nome_departamento VARCHAR(100) NOT NULL,
cpf_gerente VARCHAR(11) NULL,
PRIMARY KEY (id)
);
```

A tabela departamento armazena os dados dos departamentos da empresa.

1. **id: O ID do departamento.**
   É uma chave primária e é auto-incrementável.

2. **nome_departamento:**
   O nome do departamento. (Campo obrigatório)

3. **cpf_gerente**:
   O CPF do gerente do departamento. (Campo opcional)

Tabela empresa

```
CREATE TABLE empresa (
cnpj VARCHAR(14) NOT NULL,
nome_empresa VARCHAR(40) NOT NULL,
email_empresa VARCHAR(40) NOT NULL,
PRIMARY KEY (cnpj)
);
```

A tabela empresa armazena os dados das empresas clientes.

1. **cnpj:**
   O CNPJ da empresa.

2. **nome_empresa:**
   O nome da empresa.

3. **email_empresa:**
   O endereço de e-mail da empresa.

Tabela endereco_empresa

```
CREATE TABLE endereco_empresa (
id INT NOT NULL AUTO_INCREMENT,
cnpj_empresa VARCHAR(14) NOT NULL,
cep VARCHAR(8) NOT NULL,
rua VARCHAR(40) NOT NULL,
bairro VARCHAR(20) NOT NULL,
cidade VARCHAR(20) NOT NULL,
estado VARCHAR(2) NOT NULL,
PRIMARY KEY (id)
);
```

A tabela endereco_empresa armazena os dados dos endereços das empresas clientes.

1. **id:**
   O ID do endereço da empresa. É uma chave primária e é auto-incrementável.

2. **cnpj_empresa:**
   O CNPJ da empresa associada ao endereço.

3. **cep:**
   O CEP do endereço da empresa.

4. **rua:**
   A rua do endereço da empresa.

5. **bairro:**
   O bairro do endereço da empresa.

6. **cidade:**
   A cidade do endereço da empresa.

7. **estado:**
   O estado do endereço da empresa.

Tabela telefone_empresa

```
CREATE TABLE telefone_empresa (
id INT NOT NULL AUTO_INCREMENT,
cnpj_empresa VARCHAR(14) NOT NULL,
tel_residencial VARCHAR(20) NULL,
tel_pessoal VARCHAR(20) NOT NULL,
PRIMARY KEY (id)
);
```

A tabela telefone_empresa armazena os telefones das empresas clientes.

1. **id:**
   O ID do telefone da empresa. É uma chave primária e é auto-incrementável.

2. **cnpj_empresa:**
   O CNPJ da empresa associada ao telefone.

3. **tel_residencial:**
   O telefone residencial da empresa. (Campo opcional)

4. **tel_pessoal:**
   O telefone celular da empresa. (Campo obrigatório)

Tabela tipo_chamado

```
CREATE TABLE tipo_chamado (
  codigo_tipo_chamado INT NOT NULL
  AUTO_INCREMENT,
  descricao VARCHAR(255) NOT NULL,
  PRIMARY KEY (codigo_tipo_chamado)
);
```

A tabela tipo_chamado armazena os tipos de chamados.

1. **codigo_tipo_chamado:**
   O ID do tipo de chamado. É uma chave primária e é auto-incrementável.

2. **descrição:**
   A descrição do tipo de chamado. (Campo obrigatório)

Detalhes adicionais

Além das tabelas apresentadas acima, é possível adicionar outras tabelas para armazenar dados adicionais, como:

- **Usuários:**
  Armazena os dados dos usuários do sistema de gerenciamento de chamados.

- **Anexos:**
  Armazena os anexos dos chamados, como documentos, imagens ou vídeos.

- **tarefas:**
  Armazenar as tarefas relacionadas aos chamados.

Vantagens do uso do PostgreSQL

1. **Resistência e Escalabilidade**
   Uma das principais vantagens do PostgreSQL é sua resistência e escalabilidade. Sua capacidade de lidar com grandes volumes de dados faz dele uma escolha ideal para aplicações como o CallTasks, que necessitam de um robusto gerenciamento de chamados. A capacidade de crescimento escalável do PostgreSQL assegura que o CallTasks possa evoluir conforme as demandas, mantendo o desempenho mesmo diante de um aumento substancial na carga de

dados.

2. **Segurança Avançada**
   A segurança é uma prioridade essencial no armazenamento de dados sensíveis. O PostgreSQL destaca-se ao oferecer uma gama abrangente de recursos de segurança. Mecanismos de criptografia, controle de acesso refinado e auditorias são apenas alguns dos recursos que contribuem para a proteção eficaz dos dados no banco. Ao escolher o PostgreSQL, o CallTasks se beneficia dessas camadas de segurança, garantindo a integridade e confidencialidade dos dados dos chamados.

3. **Facilidade de Uso e Desenvolvimento**
   Embora seja poderoso em termos de recursos, o PostgreSQL é conhecido por sua relativa facilidade de uso. Isso se traduz em um ambiente de desenvolvimento e manutenção simplificado para o CallTasks. Desenvolvedores podem se concentrar na lógica da aplicação, enquanto a estrutura amigável do PostgreSQL facilita a interação e a manipulação dos dados. Essa facilidade de uso contribui para um ciclo de desenvolvimento eficiente e para a rápida implementação de novos recursos no CallTasks.

4. **Contribuições para o CallTasks**
   Ao adotar o PostgreSQL, o CallTasks eleva sua performance e confiabilidade. A resistência, escalabilidade e segurança avançada do PostgreSQL contribuem diretamente para uma melhor experiência do usuário e para a eficiência operacional do CallTasks. Os gestores podem confiar na solidez do PostgreSQL para garantir que os dados dos chamados sejam armazenados de maneira segura, promovendo a confiabilidade e a satisfação do usuário.

Vantagens do CallTasks

1. **Simplificação do Atendimento a Chamados**
   Uma das principais vantagens do CallTasks reside na simplificação do processo de atendimento a chamados. A plataforma oferece uma interface intuitiva que facilita a interação tanto para os funcionários que abrem chamados quanto para os técnicos que os atendem. Funcionalidades robustas são incorporadas para tornar o fluxo de trabalho mais eficiente, resultando em um atendimento mais rápido e eficaz.

2. **Controle e Gestão Aprimorados**

O CallTasks proporciona aos gestores uma visão abrangente dos chamados em andamento. Essa funcionalidade oferece insights valiosos sobre o desempenho do suporte técnico. Os gestores podem monitorar o status dos chamados, identificar áreas de eficiência e tomar decisões informadas para otimizar o processo de atendimento. O controle aprimorado permite uma gestão mais eficaz, contribuindo para a excelência operacional.

3. **Aumento da Satisfação dos Clientes**
   Um dos objetivos fundamentais do CallTasks é garantir um atendimento rápido e eficiente aos clientes. Ao simplificar o processo de abertura e resolução de chamados, a plataforma contribui diretamente para o aumento da satisfação dos clientes. A agilidade no suporte técnico cria uma experiência positiva, estabelecendo a confiança dos clientes na capacidade da empresa de resolver seus problemas de forma eficaz

Banco de dados

Os bancos de dados desempenham um papel importante no gerenciamento eficaz de informações em um ambiente de negócios. Este artigo foca as etapas envolvidas na criação e otimização de um banco de dados relacional, desde o design inicial até a implementação de medidas de segurança. A implementação dos SQLs foi feita da seguinte forma:

```
CREATE VIEW vw_DetalheChamado AS
SELECT
    c.codigo_chamado,
    c.cnpj_empresa,
    e.nome_empresa,
    e.email_empresa,
    c.cpf_usuario,
    u.nome_usuario,
    u.email_usuario,
    c.dataAbertura_chamado,
    c.status_chamado
FROM
    chamado c
JOIN usuario u ON
    c.cpf_usuario = u.cpf_usuario
JOIN empresa e ON
    c.cnpj_empresa = e.cnpj_empresa
ORDER BY
    c.codigo_chamado DESC;
```

```
CREATE VIEW vw_ResumoChamados AS
SELECT
    t.cod_tipo_chamado,
    t.codigo_chamado,
    t.grau_chamado,
    t.classe_chamado,
    COUNT(*) AS call_count
FROM tipo_chamado t
GROUP BY t.cod_tipo_chamado, t.codigo_chamado, t.grau_chamado,
t.classe_chamado;
```

Modelo de negócios

Modelo de negócios, incluindo as junções e visualizações, foi convertido em SQL para garantir a precisão e a eficiência da consulta. Modificando a criação de tabelas, adicionando índices e visualizações para otimizar o desempenho do sistema.

1. **Acesso**

   Além de criar usuários, grupos e atribuir permissões, enfatizamos a importância de restringir o acesso com base nas funções e responsabilidades do usuário. Descrever como as políticas de acesso contribuem para a segurança e integridade dos dados.

2. **Gatilhos**

   Os gatilhos são implementados para manter a consistência e auditar as alterações. Criamos dois gatilhos para gerenciar regras específicas de integridade e auditoria para poder monitorar com precisão o desempenho do banco de dados.

3. **Eficiência**

   A eficiência do trabalho foi melhorada com a introdução de dois procedimentos armazenados. Esses procedimentos ajudam a orientar algumas das regras operacionais do sistema e a tornar tarefas complexas mais fáceis e eficientes.

4. **BackUp**

   Políticas claras de backup e recuperação melhoram a segurança dos dados. Em caso de falha ou perda de dados, garantimos uma recuperação eficiente e minimizamos o impacto nos processos de negócio. Optamos por fazer um backup em nuvem toda Segunda-Feira de manhã às 09:00 e outro backup offline (Cold backup) toda Sexta-feira de tarde as 17:00.

DIAGRAMA DE USO

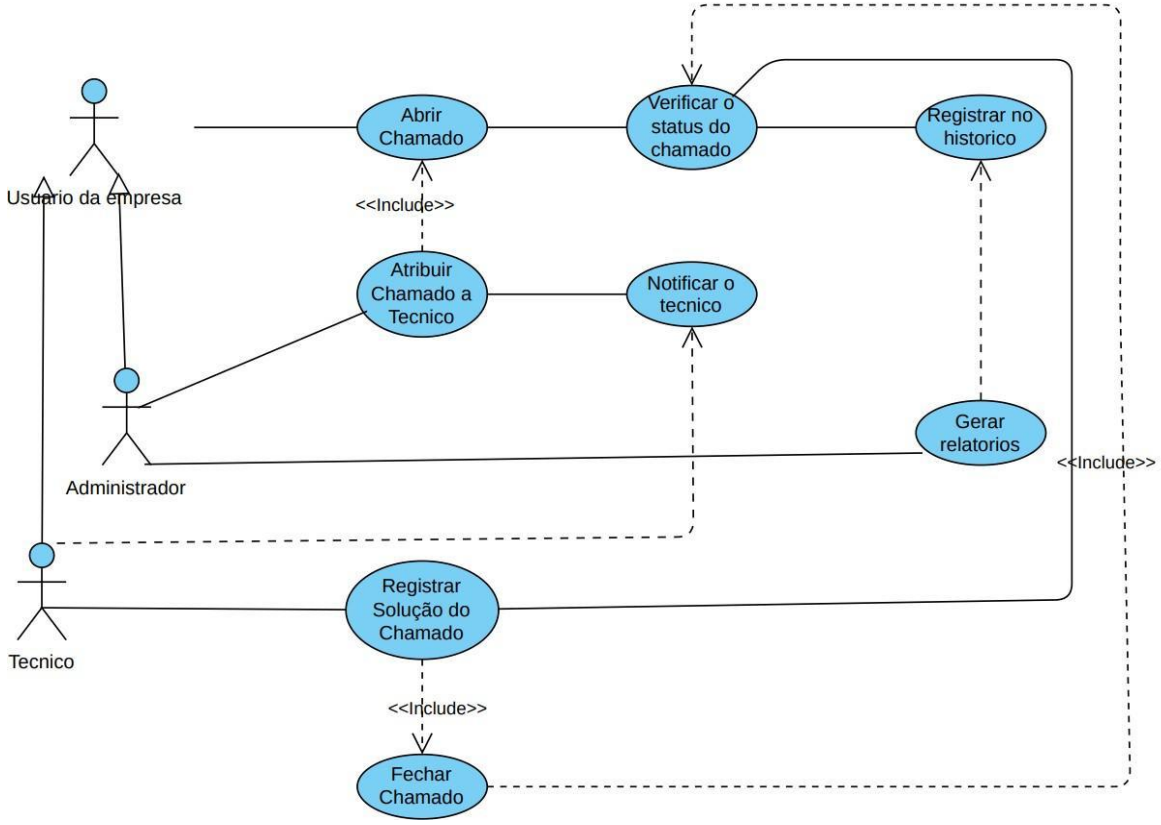


DIAGRAMA DE SEQUÊNCIA

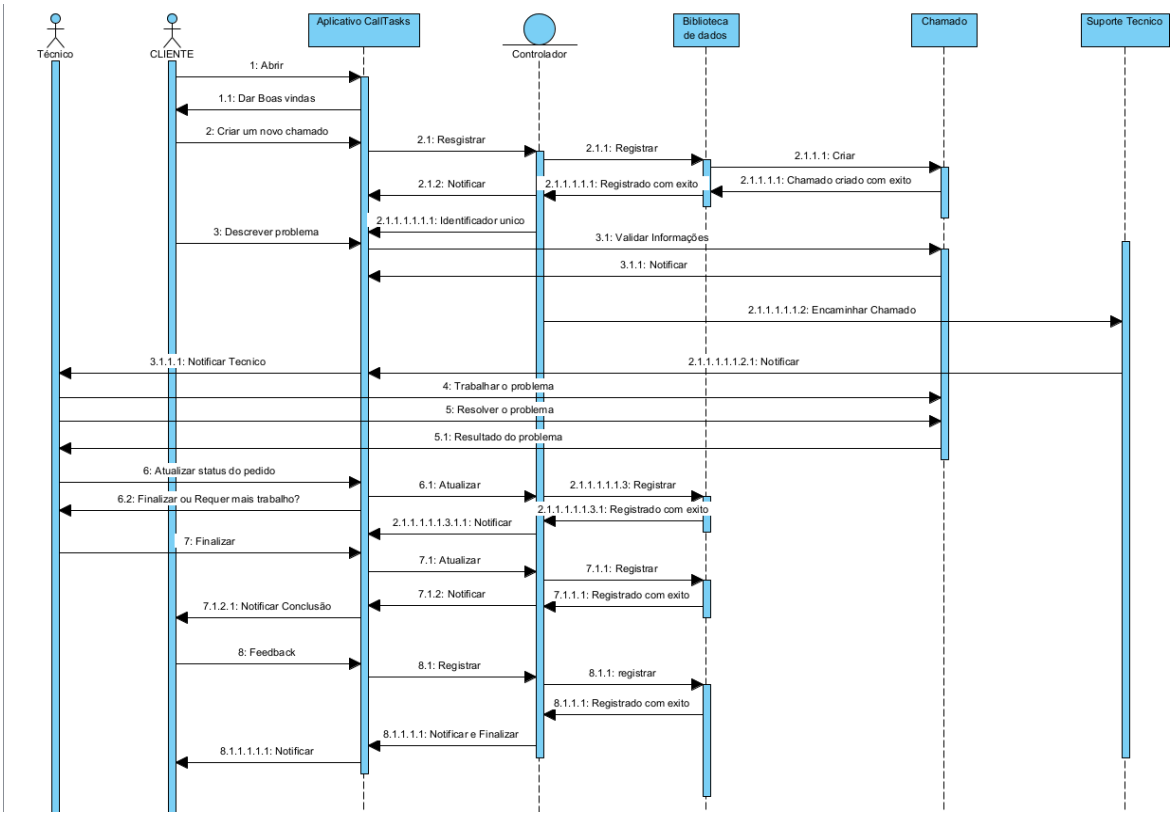


DIAGRAMA DE ATIVIDADES

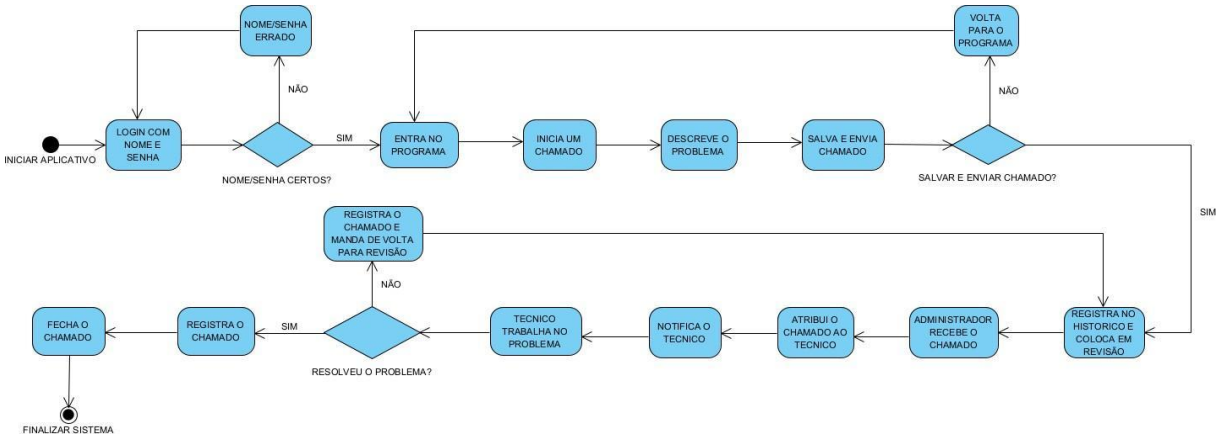


DIAGRAMA DE ESTADO

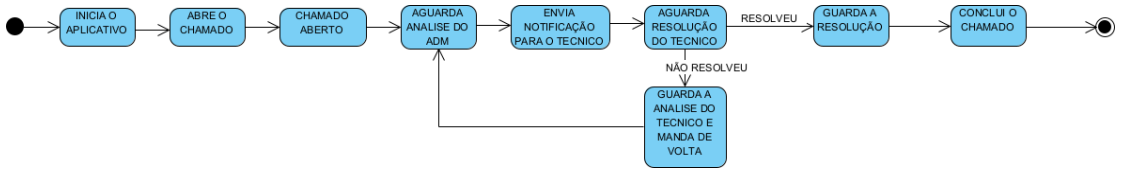
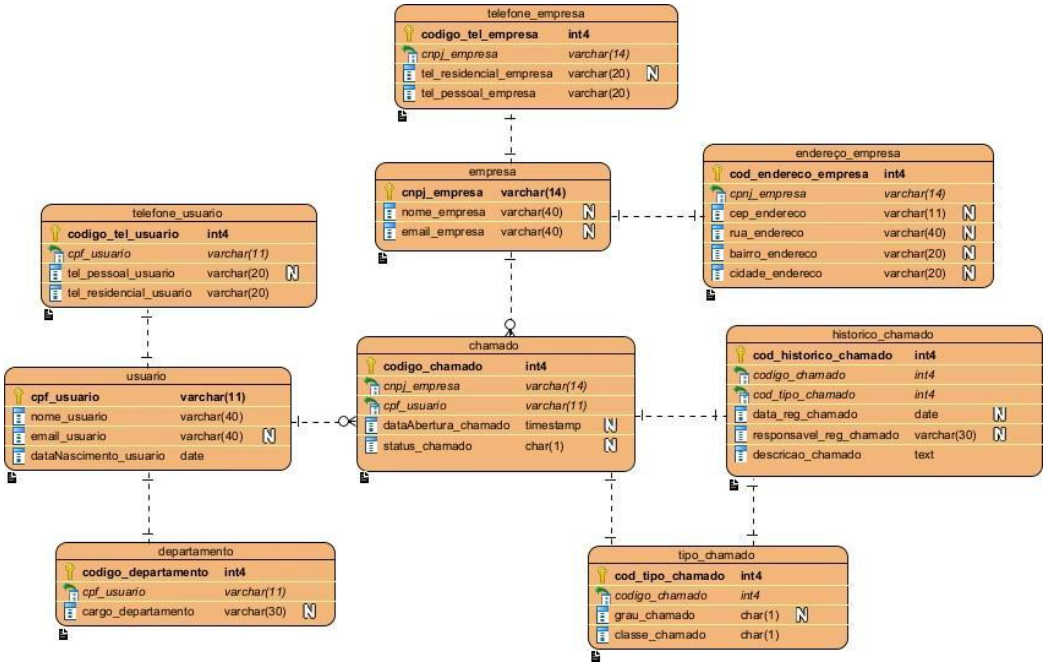
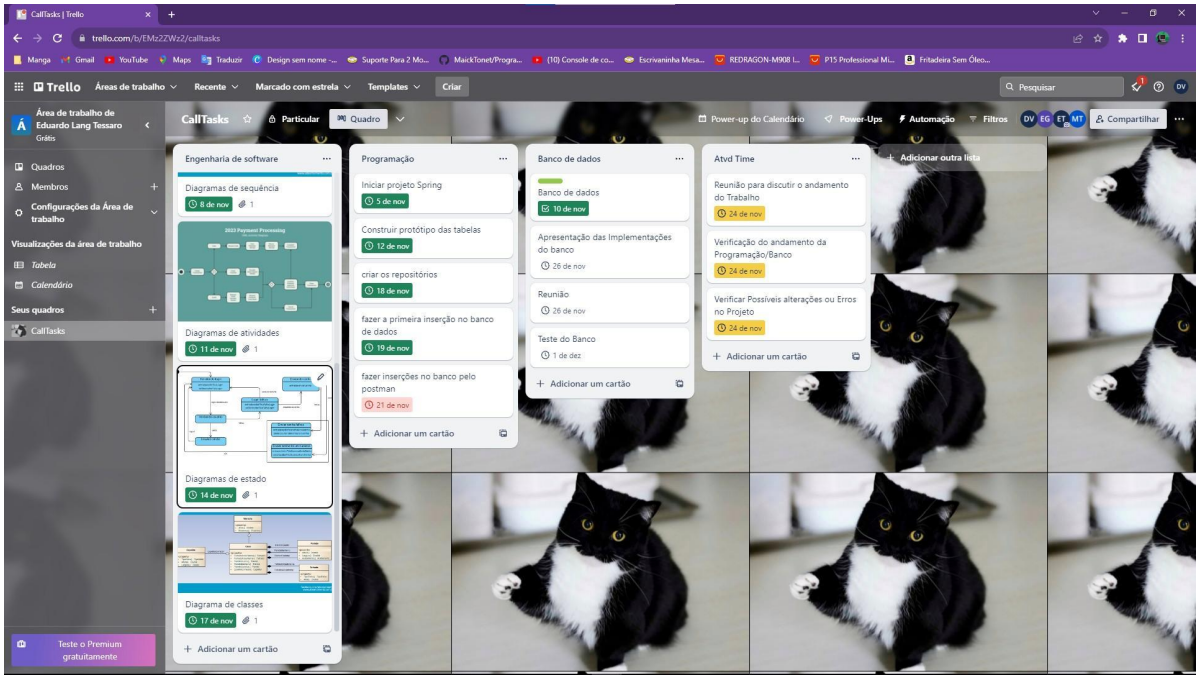
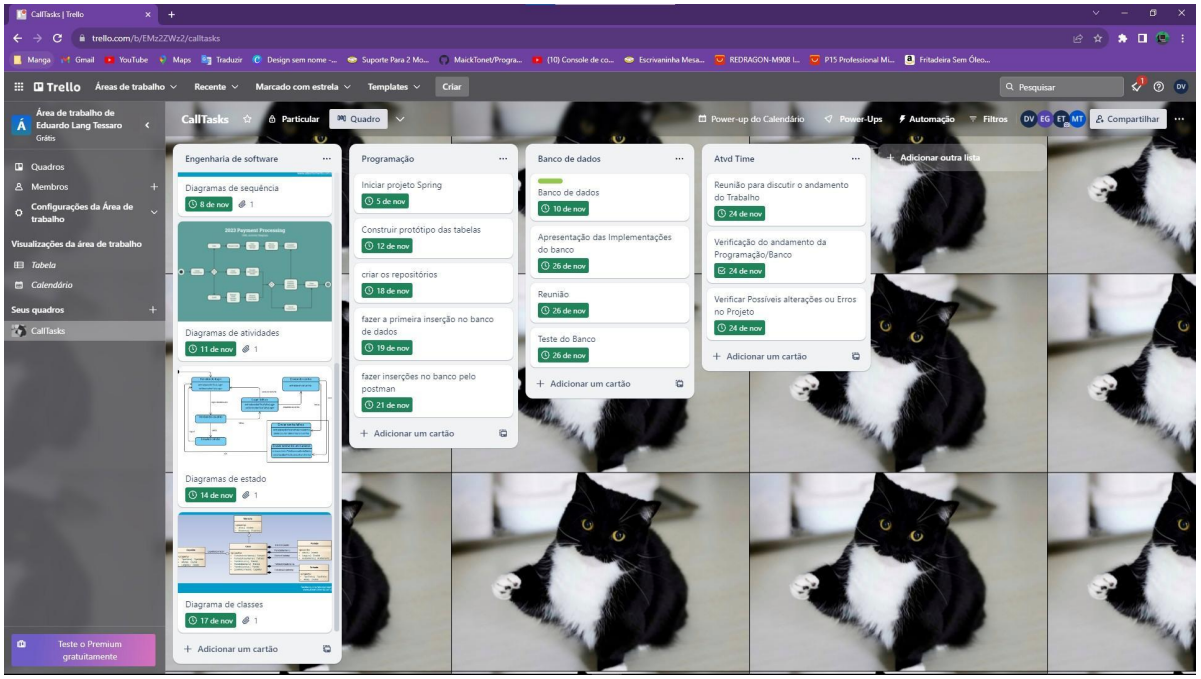


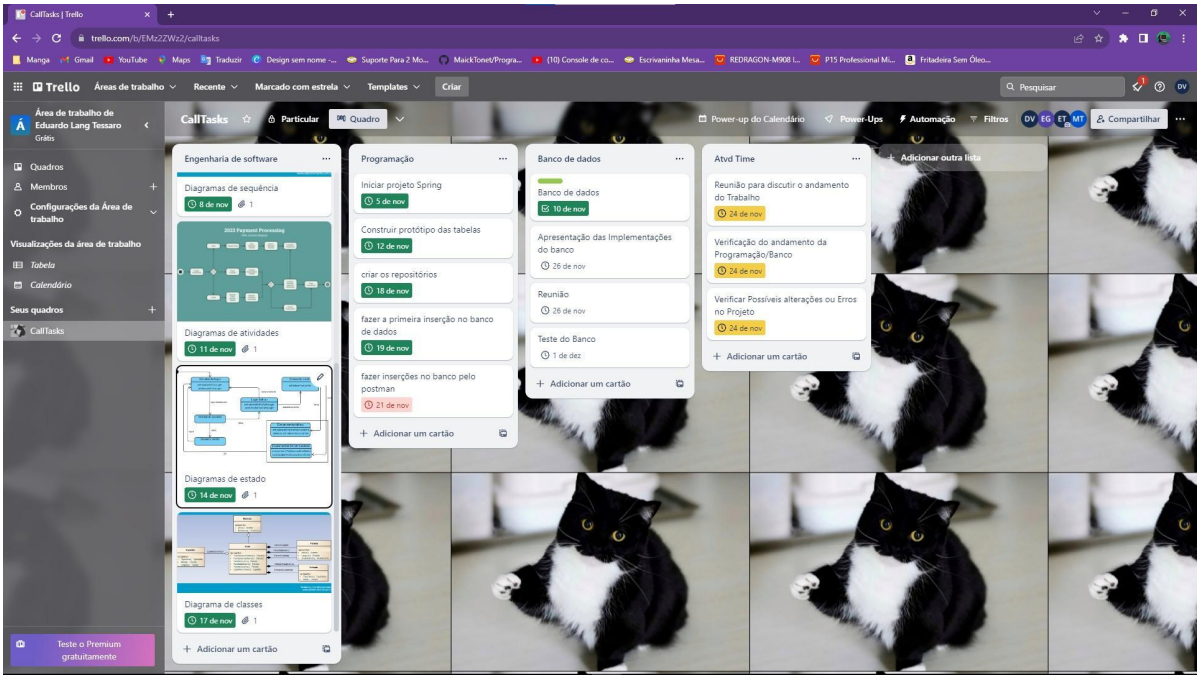
DIAGRAMA DE CLASSES



Utilização de metodologia de planejamento de atividades: TRELLO







Versionamento usando Git e GitHub: https://github.com/MaickTonet/CallTasks

Referências de Pesquisa:

(2023). Generation: Exemplos do Framework Spring. GitHub. https://github.com/ricamartins/generation/tree/master/spring

(2023). Exemplos do Spring Boot. GitHub. https://github.com/in28minutes/spring-boot-examples

(2023). Spring Boot. GitHub. https://github.com/spring-projects/spring-boot

(2023). Exemplo de Aplicação Spring Boot. GitHub. https://github.com/alexmanrique/spring-boot-application-example

(2023). O que é o Spring Boot. https://www.treinaweb.com.br/blog/o-que-e-o-spring-boot

(2023). Material Didático de Programação II e Banco de Dados II.

(2023). Ferreira, Paulo Henrique; Cardoso, Thiago. "O uso do PostgreSQL em um sistema de gestão de ativos de TI." Revista de Sistemas de Informação da FGV, v. 16, n. 2, p. 1-15, 2019.

(2023). Lima, Fabrício Santos, Carlos Magno. "Avaliação de desempenho de consultas SQL no PostgreSQL e no MySQL." Revista Brasileira de Computação Aplicada, v. 12, n. 1, p. 41-52, 2018.

(2023). Ribeiro, José Carlos; Silva, José Roberto. "Desenvolvimento de aplicações web com Java e Spring Boot." Revista de Sistemas de Informação da FGV, v. 19, n. 1, p. 1-15, 2022.