UNIVERSIDADE DO OESTE DE SANTA CATARINA CAMPUS DE SÃO MIGUEL DO OESTE
Área: Ciências Exatas e Tecnológicas **Curso: Ciência da Computação**
Semestre Letivo: 2023/02
Disciplinas: Programação II, Engenharia de Software I e Banco de Dados II
Professores: Franciele Carla Petry, Otilia Donato Barbosa e Roberson Junior Fernandes Alves Carga
Alunos: Davi Vargas Ribeiro,Eduardo Antonio Giehl, Eduardo Lang Tessaro, Maick Tonet.

CallTasks: Solução completa para gerenciamento de chamados

O CallTasks é uma solução abrangente para empresas de pequeno e grande porte que buscam aprimorar o controle e a gestão de seu suporte técnico. Com uma interface intuitiva e funcionalidades robustas, este software visa simplificar o processo de atendimento a chamados, garantindo que sua empresa opere sem interrupções imprevistos.

Requisitos funcionais

O CallTasks atende aos seguintes requisitos funcionais:

- *Registro de chamados:* Os funcionários devem poder registrar novos chamados, fornecendo informações detalhadas sobre o problema encontrado.

- *Atribuição automática de chamados:* O sistema deve atribuir automaticamente um técnico disponível para lidar com o chamado.

- *Categorização de chamados por urgência:* O sistema deverá solicitar que o usuário especifique a urgência do chamado, e então ele será categorizado por grau de urgência.

- *Fechamento e arquivamento de chamados:* O sistema deve fechar o chamado e arquivá-lo quando o técnico encerra a última tarefa do chamado.

- *Encaminhamento de chamados não resolvidos:* Em casos de chamados não resolvidos dentro de um determinado prazo, o sistema deverá encaminhar o chamado para outro técnico disponível.

- *Consulta de status de chamados:* Os usuários e técnicos devem poder consultar o status dos seus chamados.

- *Comunicação entre atendentes e usuários:* O software deve facilitar a comunicação entre atendentes e os funcionários que abriram os chamados.

- *Distinção entre chamados de software e hardware:* O sistema deverá distinguir, e separar os chamados que forem no software e chamados envolvendo hardware (computadores, monitores, scanners e impressoras etc.).

- *Histórico de chamados e relatórios:* O sistema deverá manter um histórico de chamados, e permitir a emissão de uma série de relatórios tanto para o uso operacional quanto para os gestores do processo.

Requisitos não-funcionais

O CallTasks atende aos seguintes requisitos não-funcionais:

- *Intuição e facilidade de uso:* O software deve ser intuitivo e fácil de usar, mesmo para funcionários não técnicos.

- *Segurança:* O sistema deve ter medidas de segurança para proteger as informações dos chamados e garantir a privacidade dos usuários.

- *Desempenho:* O software deve ser rápido, mesmo quando estiver com uma grande quantidade de chamados em aberto.

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

- *Escalabilidade:* O software deve ser arquitetado para suportar o crescimento e o aumento da demanda por chamados.

- *Personalização:* O software deve permitir a personalização de campos e fluxos de trabalho de acordo com as necessidades específicas da empresa.

- *Integração:* O software deve ser capaz de se integrar a outros sistemas existentes na empresa, como sistemas de gerenciamento de ativos ou ferramentas de monitoramento de rede.

Implementação

O CallTasks foi implementado usando as seguintes tecnologias:

- JPA: para armazenar os dados dos chamados no banco de dados PostgreSQL.
- Spring Boot: para facilitar o desenvolvimento do aplicativo.
- PostgreSQL: Plataforma usada para o Banco de Dados do APP

Implementação do JPA

O JPA foi usado para armazenar os dados dos chamados no banco de dados PostgreSQL. Para isso, foi criada uma classe para representar cada tipo de dado que será armazenado, a classe Chamado representará um chamado. A classe Chamado terá os seguintes atributos:

- id: O ID do chamado.
- descrição: A descrição do chamado.
- prioridade: A prioridade do chamado.
- status: O status do chamado.

A classe Chamado também terá os seguintes métodos:

- getId(): Retorna o ID do chamado.
- getDescricao(): Retorna a descrição do chamado.
- getPrioridade(): Retorna a prioridade do chamado.
- getStatus(): Retorna ao status do chamado.

Implementação do Spring Boot

O Spring Boot foi usado para facilitar o desenvolvimento do aplicativo. O Spring Boot fornece uma série de recursos prontos para uso que permitem aos desenvolvedores criar aplicativos rapidamente e com facilidade. O Spring Boot fornece um starter para o JPA que facilita a configuração e o uso do JPA.

Implementação do PostgreSQL

O PostgreSQL é um software de código aberto, portanto, pode ser baixado e instalado gratuitamente. Para instalar o PostgreSQL, siga as instruções fornecidas no site oficial do PostgreSQL.

Para Criar a tabela para armazenar os dados dos chamados
O CallTasks armazena os dados dos chamados em uma tabela chamada "chamados". Para criar a tabela “chamados” foi usado o seguinte código:

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

O código criará uma tabela com as seguintes colunas:

- id: O ID do chamado.
- descricao: A descrição do chamado.
- prioridade: A prioridade do chamado.
- status: O status 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 char statusChamado;

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

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

A anotação @Entity indica que a classe Chamado é uma entidade JPA e que a tabela associada é chamado. 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:

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


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

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.

- id: O ID do departamento. É uma chave primária e é auto-incrementável.
- nome_departamento: O nome do departamento. (Campo obrigatório)
- 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.

- cnpj: O CNPJ da empresa.
- nome_empresa: O nome da empresa.
- 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.

- id: O ID do endereço da empresa. É uma chave primária e é auto-incrementável.
- cnpj_empresa: O CNPJ da empresa associada ao endereço.
- cep: O CEP do endereço da empresa.
- rua: A rua do endereço da empresa.
- bairro: O bairro do endereço da empresa.
- cidade: A cidade do endereço da empresa.
- 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.

- id: O ID do telefone da empresa. É uma chave primária e é auto-incrementável.
- cnpj_empresa: O CNPJ da empresa associada ao telefone.
- tel_residencial: O telefone residencial da empresa. (Campo opcional)
- 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.

- codigo_tipo_chamado: O ID do tipo de chamado. É uma chave primária e é auto-incrementável.
- 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: Armazena as tarefas relacionadas aos chamados.

Vantagens do uso do PostgreSQL

O PostgreSQL oferece uma série de vantagens:

- Resistência e escalabilidade: O PostgreSQL é um banco de dados relacional altamente resistente e escalável, o que o torna ideal para aplicações que precisam lidar com um grande volume de dados.
- Segurança: O PostgreSQL oferece uma variedade de recursos de segurança para proteger os dados armazenados no banco de dados.
- Facilidade de uso: O PostgreSQL é um banco de dados relativamente fácil de usar, o que facilita o desenvolvimento e a manutenção do CallTasks.
- Ao usar o PostgreSQL, o CallTasks pode oferecer uma melhor experiência aos usuários e gestores, garantindo que os dados dos chamados sejam armazenados de forma segura e confiável.

Vantagens do CallTasks

O CallTasks oferece uma série de vantagens para empresas que buscam aprimorar o gerenciamento de seus suportes técnicos, incluindo:

- Simplificação do processo de atendimento a chamados: O CallTasks torna o processo de atendimento a chamados mais simples e eficiente, proporcionando aos funcionários e técnicos uma interface intuitiva e funcionalidades robustas.
- Melhor controle e gestão de chamados: O CallTasks fornece aos gestores uma visão abrangente dos chamados, permitindo que eles monitorem o desempenho do suporte técnico e identifiquem oportunidades de melhoria.
- Aumento da satisfação dos clientes: O CallTasks ajuda a garantir que os clientes recebam um atendimento rápido e eficiente, contribuindo para aumentar a satisfação dos clientes.

Referências De Pesquisa:

GitHub: https://github.com/ricamartins/generation/tree/master/spring

GitHub: https://github.com/in28minutes/spring-boot-examples

GitHub: https://github.com/spring-projects/spring-boot

GitHub: https://github.com/alexmanrique/spring-boot-application-example

TreinaWeb: https://www.treinaweb.com.br/blog/o-que-e-o-spring-boot

Material Didático de Programação II

- 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;

SELECT * FROM vw_DetalheChamado;
```

O 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.

```
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;

SELECT * FROM vw_ResumoChamados;
```

- 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.

- 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.

- 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.

- 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:

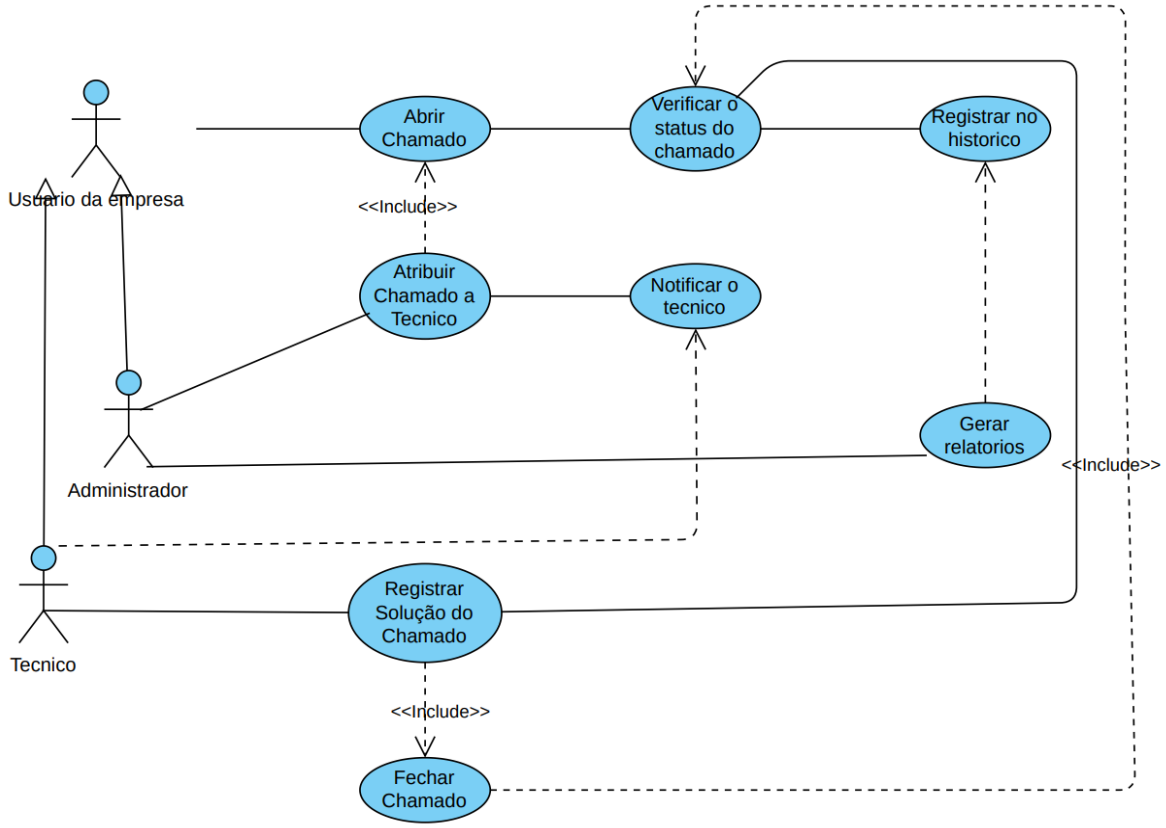


DIAGRAMA DE SEQUÊNCIA:

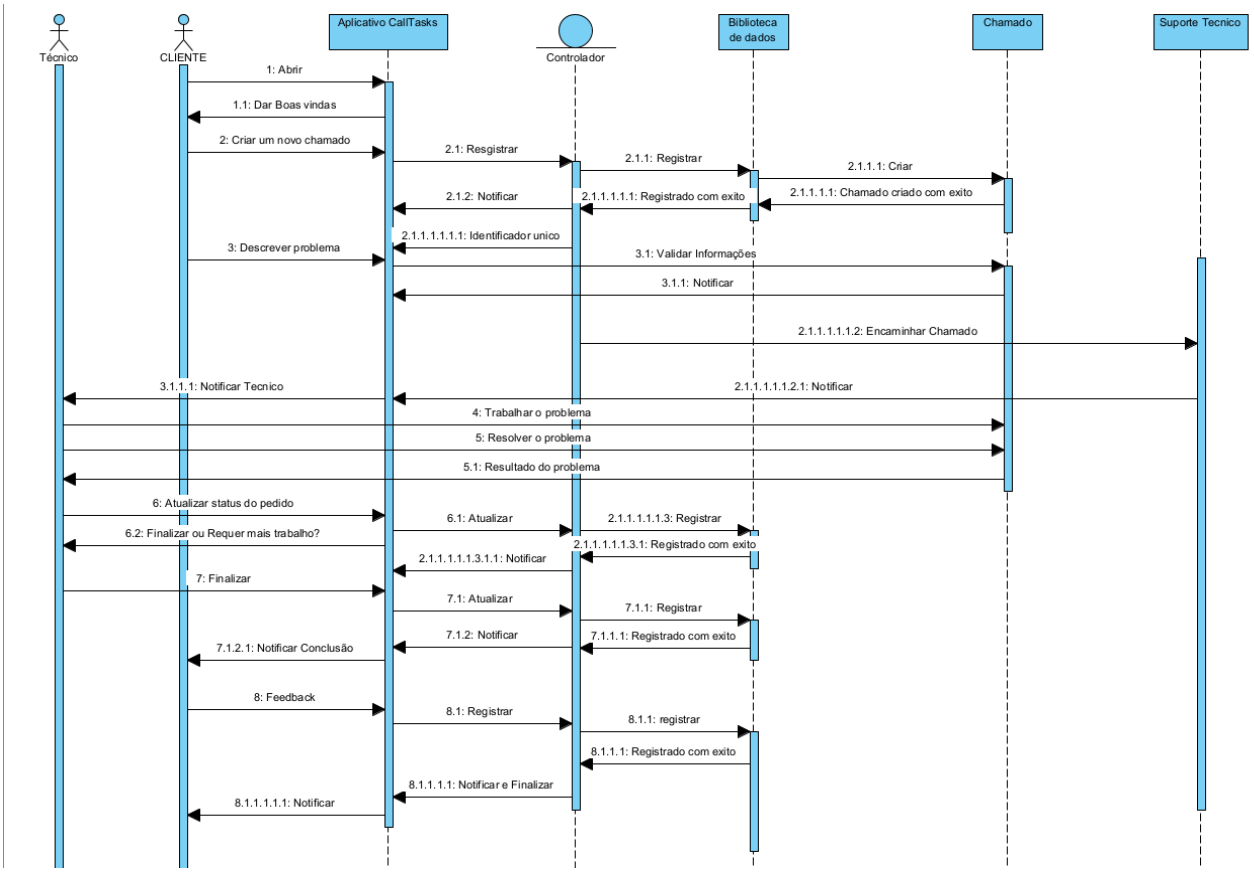


DIAGRAMA DE ATIVIDADES:

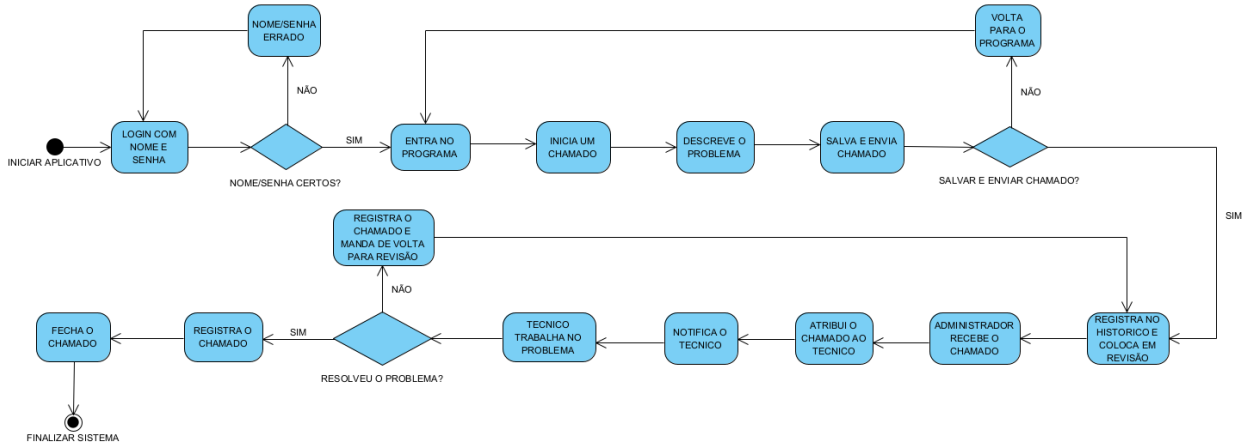


DIAGRAMA DE ESTADO:

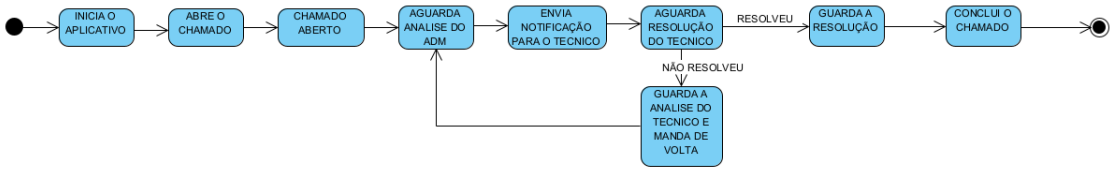
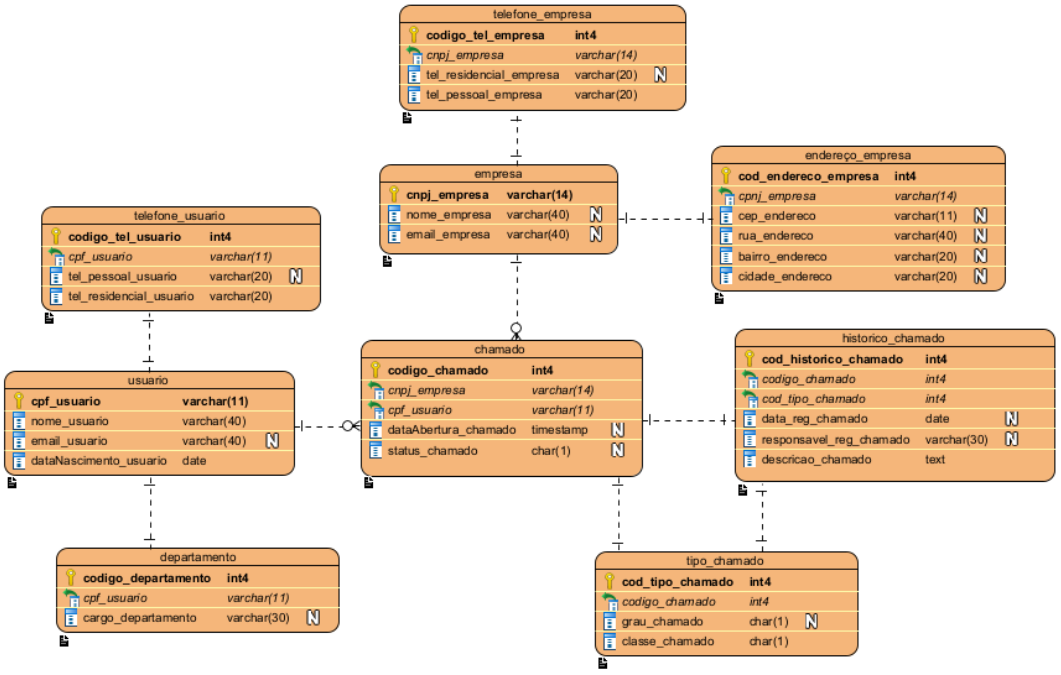
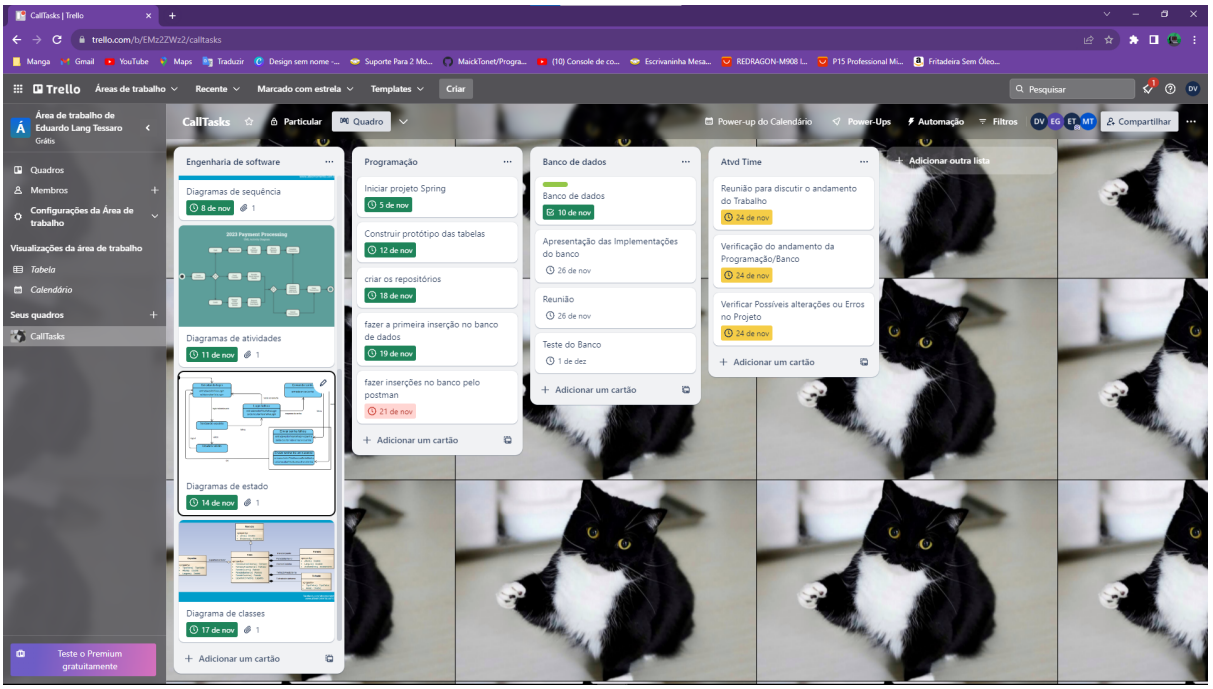
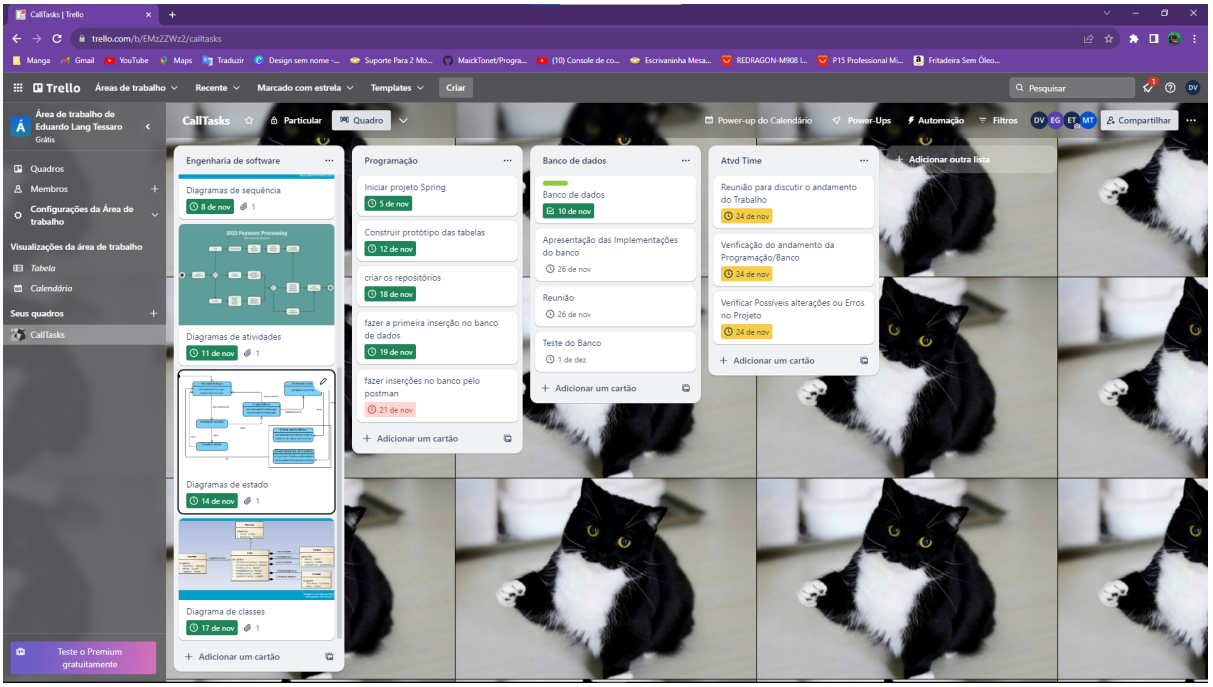


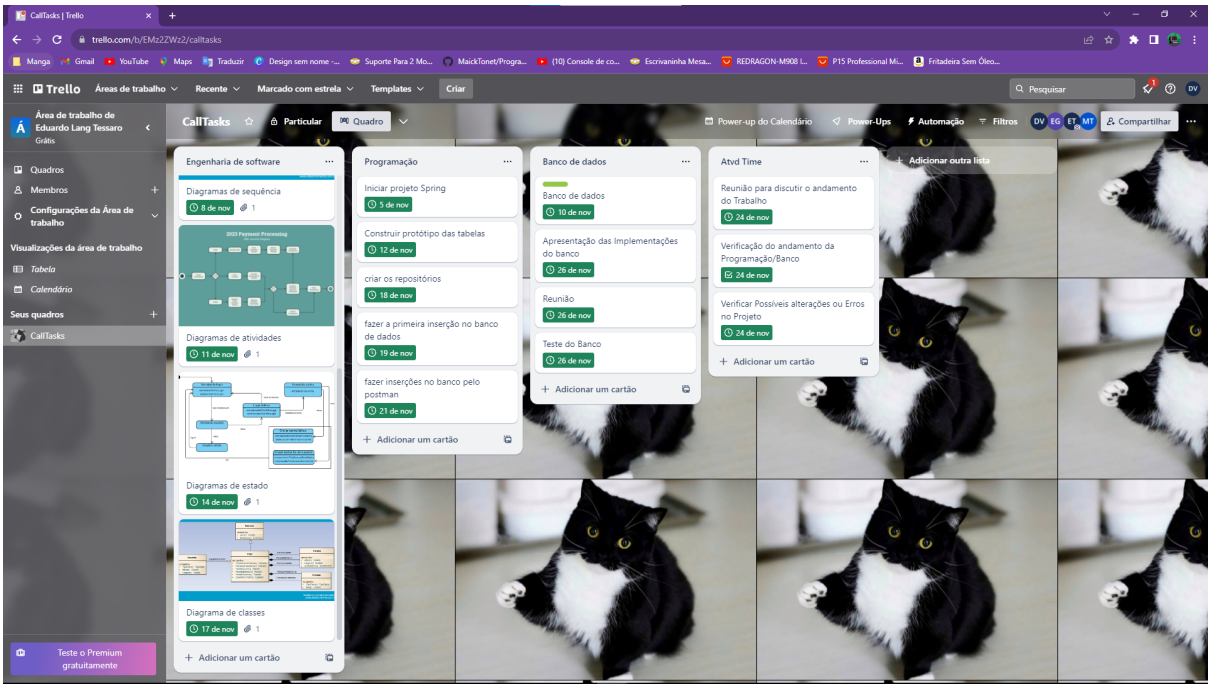
DIAGRAMA DE CLASSES:



Utilização de metodologia de planejamento de atividades: TRELLO







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