Sistema de Apoio à Apuração de Presença e Votação em Reuniões de Conselhos Superiores da UFU

Henrique Braga Alves Pereira

Universidade Federal de Uberlândia

Faculdade de Computação
Bacharelado em Ciência da Computação

Uberlândia - MG
2026

Henrique Braga Alves Pereira

Sistema de Apoio à Apuração de Presença e Votação em Reuniões de Conselhos Superiores da UFU

Trabalho de Conclusão de Curso de Desenvolvimento de Software apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do Título de Bacharel em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Mauricio Cunha Escarpinati

Uberlândia - MG
2026

Resumo

Este trabalho apresenta o desenvolvimento de um sistema de apoio à Secretaria de Conselhos da Universidade Federal de Uberlândia (UFU) para a apuração de presença e votação em reuniões de conselhos superiores, como o CONSUN e o CONGRAD. O sistema automatiza o cruzamento entre os relatórios de presença e votação exportados do Microsoft Teams e a lista de conselheiros aptos a votar em cada reunião, mantendo o quórum atualizado em tempo real e reconciliando participantes identificados de formas diferentes entre as fontes de dados, como por nome em uma planilha e por e-mail em outra, recorrendo à validação manual da secretaria apenas quando essa identificação não é clara. O sistema foi desenvolvido com Java e o *framework* Spring Boot no *back-end*, Angular no *front-end* e banco de dados H2 embarcado, seguindo um processo iterativo e incremental, com cada funcionalidade testada com dados reais antes de se avançar para a seguinte. Ao final do desenvolvimento, os doze requisitos funcionais definidos para o sistema foram implementados e validados por um plano de testes manuais, confirmando que o sistema atinge os objetivos propostos e reduz o trabalho manual antes realizado pela secretaria, ainda que dependa da exportação manual dos relatórios do Teams, por não haver integração direta com a plataforma.

Palavras-chave: Apuração de votação. Quórum. Conselhos universitários. Microsoft Teams. Desenvolvimento de software.

Abstract

This work presents the development of a support system for the Council Secretariat of the Federal University of Uberlândia (UFU) for tallying attendance and voting in meetings of the university's higher councils, such as CONSUN and CONGRAD. The system automates the cross-referencing between attendance and voting reports exported from Microsoft Teams and the list of council members eligible to vote in each meeting, keeping the quorum updated in real time and reconciling participants identified in different ways across data sources, such as by name in one spreadsheet and by e-mail in another, relying on manual validation by the secretariat only when this identification is not clear. The system was developed using Java and the Spring Boot framework on the back-end, Angular on the front-end, and an embedded H2 database, following an iterative and incremental process, with each feature tested with real data before proceeding to the next. At the end of development, the twelve functional requirements defined for the system were implemented and validated through a manual test plan, confirming that the system meets the proposed objectives and reduces the manual work previously performed by the secretariat, although it still depends on manually exporting reports from Teams, since there is no direct integration with the platform.

Keywords: Vote tallying. Quorum. University councils. Microsoft Teams. Software development.

Lista de ilustrações

Lista de Quadros

Lista de siglas

API Application Programming Interface

CLI Command Line Interface

CONGRAD Conselho de Graduação

CONSUN Conselho Universitário

CORS Cross-Origin Resource Sharing

CSV Comma-Separated Values

DDL Data Definition Language

DTO Data Transfer Object

FK Foreign Key

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

IA Inteligência Artificial

JDK Java Development Kit

JPA Java Persistence API

JVM Java Virtual Machine

LTS Long Term Support

REST Representational State Transfer

SBC Sociedade Brasileira de Computação

SPA Single Page Application

TCC Trabalho de Conclusão de Curso

UFES Universidade Federal do Espírito Santo

UFU Universidade Federal de Uberlândia

URL Uniform Resource Locator

Sumário

CAPÍTULO 1

Introdução

1.1 Contextualização do trabalho

Esse trabalho busca facilitar o processo de apuração das reuniões de conselhos superiores da Universidade Federal de Uberlândia (UFU), como o CONSUN e o CONGRAD, hoje conduzido manualmente pela secretaria responsável: conferir quem está presente, verificar se o quórum mínimo foi atingido e contabilizar os votos de cada pauta discutida. Com a migração dessas reuniões para o formato remoto ou híbrido, viabilizada por plataformas como o Microsoft Teams, esse processo passou a depender também dos relatórios de presença e participação exportados da própria plataforma, que oferece nativamente registro de presença e enquetes durante a reunião (MICROSOFT, 2026), mas sem nenhum mecanismo que cruze automaticamente esses dados com a lista oficial de conselheiros aptos a votar em cada reunião.

Soluções voltadas à digitalização de processos deliberativos já existem, mas atacam partes distintas do problema. A Universidade Federal do Espírito Santo (UFES), por exemplo, mantém um sistema próprio de votação online (UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, 2026), utilizado em eleições institucionais, como a escolha de representantes docentes junto ao Conselho Universitário, voltado, porém, a votações pontuais e isoladas, sem relação com o acompanhamento de presença ou quórum de uma sessão em andamento. Já o OpenSlides, software livre voltado à gestão de pautas, moções e votações em assembleias (OPENSLIDES, 2026), cobre boa parte do ciclo de uma reunião deliberativa, mas não oferece integração nativa com plataformas de videoconferência para importação automática de presença e votos, muito menos um mecanismo de validação manual para participantes que não são reconhecidos automaticamente na lista oficial.

Diante desse cenário, o presente trabalho propõe o desenvolvimento de um sistema de apoio à secretaria que automatiza o cruzamento entre os dados de presença e votação exportados do Teams e a lista de conselheiros de cada reunião, com foco em ser simples e didático. Ainda assim, um problema que permanece em aberto é a ausência de uma integração direta com o Teams: o sistema depende da exportação manual dos relatórios de presença e votação da plataforma para posterior importação, não havendo, no momento, comunicação automática entre os dois ambientes. Os objetivos específicos que orientaram o desenvolvimento desse sistema são detalhados na seção a seguir.

1.2 Problema e Justificativas

Os desafios que esse tema propõe estão ligados, principalmente, à forma como os dados de uma reunião são gerados e disponibilizados. A presença e a votação de cada conselheiro vêm de exportações distintas do Microsoft Teams, que nem sempre identificam a mesma pessoa da mesma forma, em uma planilha ela pode aparecer pelo nome, em outra pelo e-mail, e a plataforma não oferece nenhuma integração automática com sistemas externos, de modo que a exportação e a importação desses relatórios continuam sendo feitas manualmente. Outro fator é o caráter dinâmico da própria reunião: o quórum e a lista de conselheiros aptos a votar mudam ao longo da sessão, à medida que novos participantes vão sendo reconhecidos, o que exige que esses números sejam recalculados continuamente, e não apenas capturados uma única vez no início.

Diante desses desafios, o problema que este trabalho pretende resolver é a ausência de uma ferramenta que automatize o cruzamento entre os dados de presença e votação exportados do Teams e a lista oficial de conselheiros de cada reunião, evitando a duplicação de uma mesma pessoa quando ela é identificada de formas diferentes nas duas fontes e mantendo o quórum e as listas de conselheiros aptos sempre atualizados conforme a secretaria valida os casos pendentes durante a reunião.

1.3 Objetivos (Gerais e Específicos)

Esse trabalho tem como objetivo contribuir para tornar mais eficiente e menos manual o processo de apuração de presença e votação das reuniões de conselhos superiores da UFU, por meio do desenvolvimento de um sistema que automatize o cruzamento entre os dados exportados do Microsoft Teams e a lista de conselheiros de cada reunião. Ao longo do desenvolvimento, buscou-se verificar, na prática, se esse sistema realmente reduz esse trabalho manual.

Para alcançar esse objetivo, o trabalho busca atender à necessidade da Secretaria de Conselhos da UFU de gerenciar múltiplos conselhos, como o CONSUN e o CONGRAD, cada um com sua própria lista de conselheiros e reuniões, sem que os dados de um se misturem aos de outro. O sistema também precisa acompanhar o quórum em tempo real, à medida que a reunião acontece, e reconhecer um mesmo participante mesmo quando ele aparece identificado de formas diferentes entre os dados de presença e de votação, recorrendo à secretaria apenas quando essa identificação não é clara. Nenhuma dessas características é

encontrada, em conjunto, nas alternativas já existentes: soluções genéricas de gestão de assembleias, como o OpenSlides (OPENSLIDES, 2026), não separam os dados por conselho; os relatórios nativos do Microsoft Teams (MICROSOFT, 2026) não cruzam presença com a lista de conselheiros aptos; e sistemas institucionais de votação, como o da UFES (UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, 2026), são voltados a eleições pontuais, não ao acompanhamento de uma sessão completa.

1.4 Contribuições Esperadas

Espera-se que este trabalho traga contribuições em duas frentes distintas. A primeira é de natureza prática: a entrega, à Secretaria de Conselhos da UFU, de um sistema funcional que automatiza o cruzamento entre presença, votação e a lista de conselheiros de cada reunião, reduzindo o esforço manual do processo de apuração hoje realizado à mão.

A segunda é focada no aprendizado depois de lidar com o problema na prática: cruzar dados que vêm de fontes diferentes e nem sempre são identificados da mesma forma, como nome em uma fonte e e-mail em outra. Esse é um problema que dificilmente se limita a este sistema, e que pode se repetir em outros contextos no futuro. Também está ligada à forma como o sistema foi estruturado: em vez de uma lista de conselheiros única para todas as reuniões e conselhos, cada lista passou a pertencer à sua própria reunião, o que resolveu o problema aqui e pode servir de referência para outros sistemas com a mesma necessidade. Por fim, está ligada ao comparativo feito entre esse sistema e o que já existe, como o OpenSlides, sistemas de votação institucionais e os recursos nativos do Teams, que mostrou que nenhuma dessas ferramentas cobre o cruzamento automático de presença, votação e quórum.

1.5 Organização do Trabalho

Este trabalho está organizado em cinco capítulos, além deste primeiro, que apresenta a contextualização, o problema, os objetivos e as contribuições esperadas do trabalho. O Capítulo 2 apresenta os requisitos do sistema, incluindo a descrição geral do sistema, o estudo de viabilidade, os requisitos funcionais e não funcionais, e os protótipos das interfaces com o usuário. Já o Capítulo 3 traz a análise e o projeto do sistema, com o modelo de domínio, os diagramas de sequência, a modelagem de dados e o diagrama de classes utilizados no desenvolvimento. O Capítulo 4, por sua vez, descreve o desenvolvimento do sistema propriamente dito: o processo adotado, a arquitetura e os padrões de projeto utilizados, as tecnologias empregadas, as instruções de instalação e

uso, os testes realizados e os resultados alcançados. Por fim, o Capítulo 5 apresenta as conclusões do trabalho, retomando as principais contribuições e apontando os trabalhos futuros.

1.6 IA-Generativa

Neste trabalho, foi utilizada a ferramenta de Inteligência Artificial (IA) Generativa Claude, da Anthropic, em duas frentes. A primeira foi como apoio à revisão do texto, auxiliando na leitura do conteúdo já escrito e na correção ortográfica e gramatical. A segunda foi como apoio à busca de trabalhos relacionados e do estado da arte, auxiliando a localizar publicações e fontes pertinentes ao tema deste trabalho. Todo o conteúdo produzido com o apoio da ferramenta foi revisado e validado pelo autor, que é o único responsável por seu conteúdo.

CAPÍTULO **2**

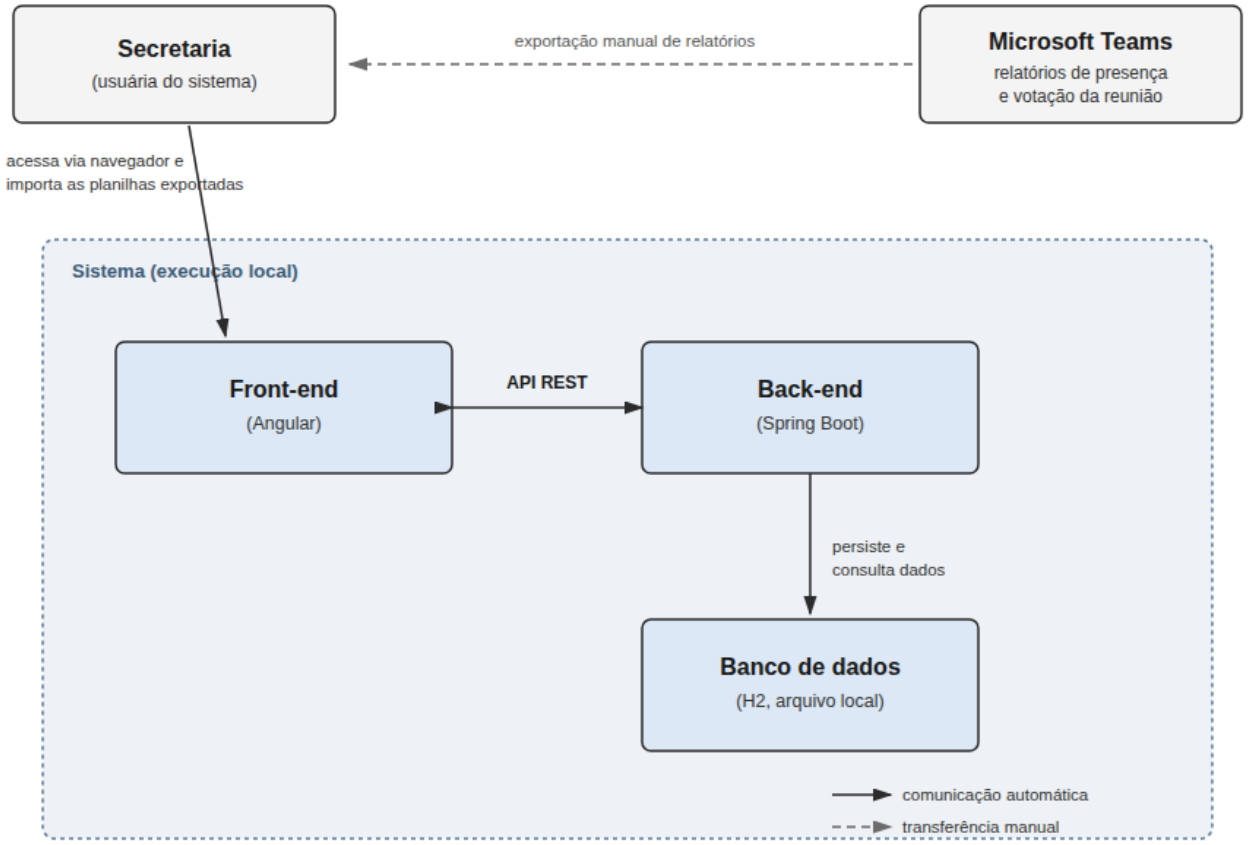
Requisitos do Sistema

Este capítulo apresenta os requisitos do sistema desenvolvido neste trabalho. Inicialmente, será abordada a descrição geral do sistema, incluindo seu propósito, sua abrangência e o perfil dos usuários a que se destina. Em seguida, é apresentado um breve estudo de viabilidade, com os principais benefícios esperados, os custos envolvidos e o cronograma de desenvolvimento adotado. Por fim, são detalhados os requisitos funcionais e não funcionais do sistema, acompanhados dos protótipos das interfaces propostas para os usuários.

2.1 Descrição geral do sistema

O sistema desenvolvido neste trabalho tem como objetivo apoiar a Secretaria de Conselhos da UFU na apuração de presença e votação das reuniões de conselhos superiores, como o CONSUN e o CONGRAD. Seu propósito central é automatizar o cruzamento entre os dados de presença e votação, exportados do Microsoft Teams, e a lista de conselheiros aptos a votar em cada reunião, calculando o quórum e o resultado das votações de forma automática, e submetendo à secretaria apenas os casos em que a identificação de um participante não pode ser resolvida automaticamente. A importância do projeto está em reduzir o tempo e o esforço que a secretaria hoje dedica a esse processo, que é feito manualmente, e em diminuir a ocorrência de erros como duplicação de participantes e contagem incorreta de quórum.

Em termos de arquitetura, o sistema é dividido em dois módulos principais: um *back-end*, responsável por armazenar os dados e processar o cruzamento entre presença, votação e conselheiros, e um *front-end*, por meio do qual a secretaria interage com o sistema, cadastrando reuniões, pautas e conselheiros, importando as planilhas exportadas do Teams e acompanhando os resultados da apuração. Os dados são armazenados localmente em um banco de dados próprio do sistema, e a comunicação entre os dois módulos ocorre por meio de uma API REST. O sistema não depende de conexão com a



internet para funcionar, nem se comunica automaticamente com o Microsoft Teams ou com qualquer outro sistema externo: a entrada de dados do Teams é feita por meio da importação manual dos relatórios de presença e votação exportados pela própria secretaria.

Figura 1 – Diagrama de Arquitetura (Fonte: Autor)

2.1.1 Abrangência e sistemas relacionados

O sistema permite o cadastro de reuniões e das pautas discutidas em cada uma delas, bem como o cadastro da lista de conselheiros aptos a votar em cada reunião. A partir dessas informações, o sistema permite importar planilhas de presença e de votação exportadas do Microsoft Teams, cruzando automaticamente esses dados com a lista de conselheiros cadastrada, calculando o quórum da reunião e apurando o resultado da votação de cada pauta. Nos casos em que a identificação de um participante não pode ser resolvida automaticamente, por exemplo, quando um nome não corresponde a nenhum conselheiro cadastrado, o sistema apresenta essa situação à secretaria para validação manual. O sistema também permite o cadastro manual de votos avulsos e a exportação de listas e relatórios relacionados à apuração, como a lista de conselheiros aptos em formato CSV.

Ficam fora do escopo deste sistema algumas funcionalidades que, a princípio, poderiam parecer relacionadas ao problema tratado. O sistema não realiza a reunião em si, nem grava ou transmite áudio e vídeo, essa função continua sendo provida pelo Microsoft Teams. O sistema também não gera a ata formal da reunião, ficando essa tarefa a cargo da secretaria, a partir dos dados de presença e votação disponibilizados pelo sistema. Por fim, o sistema não inclui controle de autenticação de múltiplos usuários, já que é operado localmente, apenas pela secretaria responsável.

Quanto à interação com outros sistemas, este sistema não realiza nenhuma comunicação automática ou integração direta com outras plataformas: ele é autocontido em sua execução. No entanto, ele depende de dados originados no Microsoft Teams, os relatórios de presença e de votação de cada reunião, que são exportados manualmente da plataforma e, em seguida, importados pela secretaria por meio da interface do sistema, conforme descrito na Seção 2.1.

2.1.2 Descrição dos usuários

Os usuários deste sistema são os membros da Secretaria de Conselhos da UFU responsáveis pela apuração de presença e votação nas reuniões dos conselhos superiores, como o CONSUN e o CONGRAD. Trata-se de um único perfil de usuário, com rotina administrativa e não necessariamente com conhecimento técnico em informática, por isso a interface do sistema precisa ser simples o suficiente para ser operada sem treinamento especializado.

O principal problema que hoje limita a produtividade desses usuários é a natureza manual do processo de apuração: a cada reunião, é preciso conferir, item por item, se cada participante presente ou que votou consta na lista de conselheiros, calcular o quórum a partir dessa conferência e apurar o resultado de cada pauta, cruzando informações vindas de diferentes planilhas exportadas do Microsoft Teams. Esse processo consome tempo que poderia ser dedicado a outras atividades da secretaria, além de estar sujeito a erros como a duplicação de um mesmo participante ou a contagem incorreta do quórum, conforme discutido na Seção 1.2.

2.2 Estudo de Viabilidade do Sistema

Este sistema foi desenvolvido com tecnologias gratuitas e de código aberto, sem custos de licenciamento, e sua adoção pela Secretaria de Conselhos da UFU não exige investimento em infraestrutura adicional, já que ele roda localmente na própria máquina da secretaria. Os principais benefícios esperados com a adoção deste sistema são a

redução do tempo e do esforço manual da secretaria no processo de apuração de presença e votação, a diminuição de erros recorrentes nesse processo, como a duplicação de participantes e a contagem incorreta de quórum, discutidos na Seção 1.2, e o consequente aumento da confiabilidade dos registros produzidos a cada reunião, incluindo os dados usados posteriormente na elaboração da ata.

O sistema foi construído inteiramente com tecnologias gratuitas e de código aberto: o *back-end* utiliza o *framework* Spring Boot, o *front-end* utiliza Angular, e a persistência dos dados é feita em um banco H2 local, sem custo de licença em nenhuma dessas ferramentas. Como o sistema é executado localmente pela própria secretaria, também não há custo de hospedagem ou de infraestrutura de servidor.

O desenvolvimento do sistema ocorreu ao longo de um semestre letivo, dividido em três fases principais. A primeira foi dedicada à implementação das funcionalidades principais do sistema: cadastro de reuniões, pautas e conselheiros, importação de planilhas de presença e votação, cruzamento automático desses dados e validação manual de casos ambíguos. Na segunda fase, foram identificados e corrigidos problemas observados durante o uso do sistema, como a duplicação de participantes e a apuração incorreta de quórum. Por fim, a terceira fase consistiu em testes manuais estruturados, realizados para validar as correções implementadas.

2.2.1 Benefícios Esperados

Os principais benefícios esperados com a adoção deste sistema pelos usuários são a redução do tempo e do esforço manual dedicados à apuração de presença e votação em cada reunião, hoje realizada por meio da conferência manual de planilhas exportadas do Teams. Espera-se também a diminuição de erros recorrentes nesse processo, como a duplicação de um mesmo participante identificado de formas diferentes entre as fontes de dados e a contagem incorreta do quórum, o que deve aumentar a confiabilidade dos registros produzidos em cada reunião. Por fim, como o sistema foi pensado para ser simples e didático, esses benefícios devem se concretizar sem exigir da secretaria qualquer conhecimento técnico prévio ou treinamento especializado para sua utilização.

2.2.2 Cronograma de Desenvolvimento

O desenvolvimento do sistema seguiu um processo incremental, no qual funcionalidades foram implementadas, testadas e corrigidas em ciclos sucessivos, sem fases estanques de requisitos, projeto e implementação. Todas as tarefas foram executadas por um único recurso, o autor deste trabalho. O detalhamento das tarefas

realizadas entre 9 de abril e 8 de agosto de 2026 está no Quadro 1.

Quadro 1 – Cronograma detalhado de desenvolvimento

Tarefa	Recurso	Início	Término
Levantamento de requisitos e planejamento inicial	Autor	09/04	15/04
Modelagem do banco de dados e estruturação do projeto (*back-end* e *front-end*)	Autor	16/04	22/04
Implementação do cadastro de reuniões e pautas	Autor	23/04	29/04
Implementação do cadastro e importação de conselheiros	Autor	30/04	06/05
Implementação da importação de planilhas de presença	Autor	07/05	20/05
Implementação da importação/cadastro de votos e do cruzamento automático	Autor	21/05	03/06
Implementação da validação manual de casos ambíguos	Autor	04/06	10/06
Implementação da apuração de votação por pauta e da lista de votação	Autor	11/06	17/06
Identificação e correção do problema da lista de conselheiros global	Autor	18/06	01/07
Identificação e correção do erro de contagem de quórum	Autor	02/07	08/07
Testes manuais estruturados e validação das correções	Autor	09/07	22/07
Ajustes finais e documentação do sistema	Autor	23/07	08/08

(Fonte: Autor)

2.3 Requisitos funcionais - RF

2.3.1 Diagrama de Caso de Uso do sistema

O sistema possui um único ator, a Secretaria, responsável por todas as interações previstas. O diagrama de casos de uso do sistema (Figura 2) organiza os doze requisitos funcionais nos mesmos quatro grupos: Reuniões e Pautas, Gestão de Conselheiros, Presença e Votação, e Validação Manual. Como há apenas um tipo de usuário, todos os casos de uso se conectam diretamente ao ator Secretaria, sem necessidade de outros atores ou de relações de inclusão/extensão entre os casos de uso.

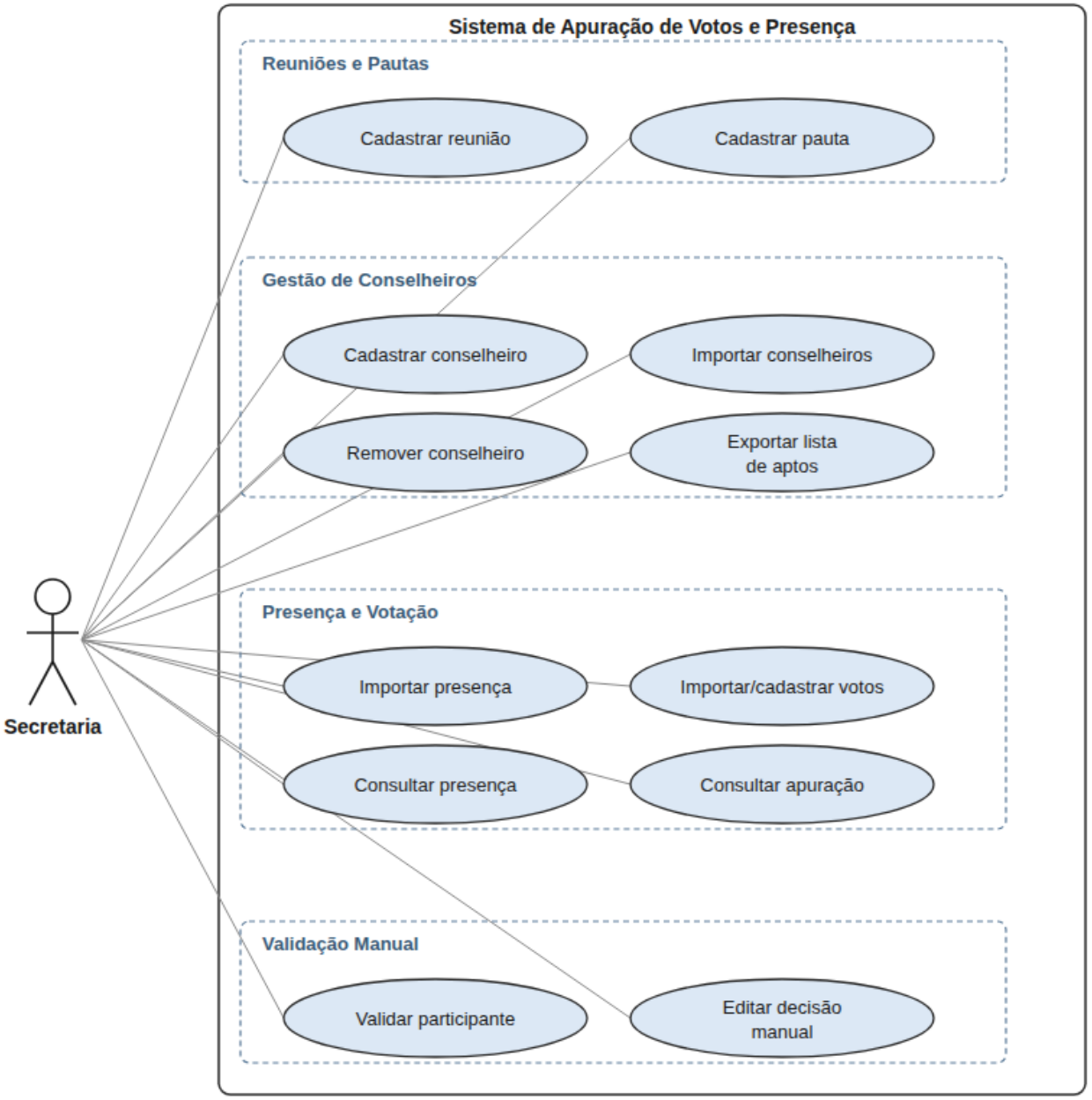


Figura 2 – Diagrama de casos de uso do sistema (Fonte: Autor)

2.3.2 Descrição dos Cenários dos Casos de Uso

2.3.2.1 [RF001] Cadastrar reunião

- **Descrição:** Permite à secretaria cadastrar uma nova reunião de conselho no sistema, servindo como base para o cadastro de pautas, conselheiros, presença e votos daquela reunião.
- **Ator:** Secretaria
- **Prioridade:** Essencial

- **Entradas e pré-condições:** Nome/identificação da reunião e conselho ao qual pertence (ex.: CONSUN, CONGRAD). Não há pré-condição além do acesso à tela "Detalhes da Reunião".
- **Saídas e pós-condições:** A reunião é criada e passa a estar disponível para seleção nas demais telas do sistema (cadastro de pautas, conselheiros, presença e votação).
- **Fluxo de eventos principal**

  1. O caso de uso inicia quando a secretaria acessa a tela "Detalhes da Reunião" e seleciona a opção "Cadastrar Reunião".
  2. A secretaria informa os dados da reunião.
  3. A secretaria confirma o cadastro.
  4. O sistema salva a reunião e a exibe na lista de reuniões cadastradas.

- **Fluxos secundários** (alternativos e de exceção)

  1. Caso algum campo obrigatório não seja preenchido, o sistema exibe uma mensagem de erro e não conclui o cadastro.

2.3.2.2 [RF002] Cadastrar pauta

- **Descrição:** Permite à secretaria cadastrar uma ou mais pautas a serem discutidas e votadas em uma reunião já cadastrada.
- **Ator:** Secretaria
- **Prioridade:** Essencial
- **Entradas e pré-condições:** Reunião já cadastrada no sistema; descrição da pauta.
- **Saídas e pós-condições:** A pauta é associada à reunião selecionada e passa a estar disponível para apuração de votação.
- **Fluxo de eventos principal**

  1. O caso de uso inicia quando a secretaria seleciona uma reunião já cadastrada.
  2. A secretaria acessa a opção de cadastrar pauta e informa a descrição.
  3. A secretaria confirma o cadastro.
  4. O sistema salva a pauta, associando-a à reunião selecionada.

- **Fluxos secundários** (alternativos e de exceção)

    1. Caso a descrição da pauta não seja informada, o sistema exibe uma mensagem de erro e não conclui o cadastro.
    2. A secretaria pode repetir o fluxo principal para cadastrar mais de uma pauta na mesma reunião.

2.3.2.3 [RF003] Cadastrar conselheiro

- **Descrição:** Permite à secretaria cadastrar manualmente um conselheiro apto a votar em uma reunião específica, informando nome e e-mail institucional.
- **Ator:** Secretaria
- **Prioridade:** Essencial
- **Entradas e pré-condições:** Reunião já cadastrada; nome e e-mail institucional do conselheiro.
- **Saídas e pós-condições:** O conselheiro passa a constar na lista de aptos a votar naquela reunião, com status "Válido".
- **Fluxo de eventos principal**

    1. A secretaria acessa a aba "Carregar Lista de Conselheiros" da reunião desejada.
    2. A secretaria seleciona a opção "Adicionar manualmente".
    3. A secretaria informa nome e e-mail institucional do conselheiro.
    4. A secretaria confirma o cadastro.
    5. O sistema adiciona o conselheiro à lista da reunião, com status "Válido".

- **Fluxos secundários** (alternativos e de exceção)

    1. Caso o e-mail informado já conste na lista da reunião, o sistema exibe uma mensagem de erro e não conclui o cadastro.
    2. Caso algum campo obrigatório não seja preenchido, o sistema exibe uma mensagem de erro e não conclui o cadastro.

2.3.2.4 [RF004] Importar conselheiros

- **Descrição:** Permite à secretaria importar, de uma só vez, a lista de conselheiros aptos a votar em uma reunião, a partir de uma planilha.
- **Ator:** Secretaria

- ❑ **Prioridade:** Essencial
- ❑ **Entradas e pré-condições:** Reunião já cadastrada; planilha (.xlsx ou .csv) com as colunas obrigatórias Nome e E-mail institucional.
- ❑ **Saídas e pós-condições:** Os conselheiros da planilha passam a constar na lista de aptos a votar naquela reunião, com status "Válido".
- ❑ **Fluxo de eventos principal**
    1. A secretaria acessa a aba "Carregar Lista de Conselheiros" da reunião desejada.
    2. A secretaria seleciona ou arrasta a planilha para a área de carregamento.
    3. O sistema lê a planilha e cadastra cada conselheiro listado.
    4. O sistema exibe a lista atualizada de conselheiros cadastrados na reunião.
- ❑ **Fluxos secundários** (alternativos e de exceção)
    1. Caso a planilha não contenha as colunas obrigatórias, o sistema exibe uma mensagem de erro e não realiza a importação.
    2. Caso algum e-mail da planilha já conste na lista da reunião, o sistema ignora o registro duplicado.

2.3.2.5 [RF005] Remover conselheiro

- ❑ **Descrição:** Permite à secretaria remover um conselheiro da lista de aptos a votar em uma reunião específica.
- ❑ **Ator:** Secretaria
- ❑ **Prioridade:** Importante
- ❑ **Entradas e pré-condições:** Reunião já cadastrada, com ao menos um conselheiro cadastrado.
- ❑ **Saídas e pós-condições:** O conselheiro deixa de constar na lista de aptos a votar daquela reunião; a remoção não afeta a lista de conselheiros de nenhuma outra reunião.
- ❑ **Fluxo de eventos principal**
    1. A secretaria acessa a aba "Carregar Lista de Conselheiros" da reunião desejada.
    2. A secretaria localiza o conselheiro na lista e seleciona "Remover".

3. O sistema remove o conselheiro da lista daquela reunião.

- **Fluxos secundários** (alternativos e de exceção)

1. Não há fluxos secundários identificados para este caso de uso.

2.3.2.6 [RF006] Exportar lista de aptos

- **Descrição:** Permite à secretaria exportar a lista de conselheiros aptos a votar em uma reunião.
- **Ator:** Secretaria
- **Prioridade:** Desejável
- **Entradas e pré-condições:** Reunião já cadastrada, com ao menos um conselheiro cadastrado.
- **Saídas e pós-condições:** O sistema gera um arquivo (CSV) com a lista de conselheiros aptos a votar naquela reunião, disponível para download.
- **Fluxo de eventos principal**

1. A secretaria acessa a aba "Carregar Lista de Conselheiros" da reunião desejada.
2. A secretaria seleciona "Exportar lista".
3. O sistema gera e disponibiliza o arquivo para download.

- **Fluxos secundários** (alternativos e de exceção)

1. Não há fluxos secundários identificados para este caso de uso.

2.3.2.7 [RF007] Importar presença

- **Descrição:** Permite à secretaria importar a planilha de presença exportada do Microsoft Teams, atualizando a lista de participantes ativos na reunião.
- **Ator:** Secretaria
- **Prioridade:** Essencial
- **Entradas e pré-condições:** Reunião já cadastrada; planilha de presença exportada do Teams.
- **Saídas e pós-condições:** A lista de participantes ativos é atualizada, com indicação de quem está online e apto a votar; o quórum é recalculado.
- **Fluxo de eventos principal**

1. A secretaria acessa a aba "Lista de Presença" da tela "Processar Votos e Presenças".
2. A secretaria carrega a planilha de presença exportada do Teams.
3. O sistema cruza os e-mails da planilha com a lista de conselheiros cadastrada.
4. O sistema atualiza a lista de participantes ativos e o quórum exibido.

- **Fluxos secundários** (alternativos e de exceção)

1. Caso um e-mail da planilha não conste na lista de conselheiros, o sistema o inclui na lista de participantes com indicação de "Não apto".

2.3.2.8 [RF008] Importar/cadastrar votos

- **Descrição:** Permite à secretaria importar a planilha de votos de uma pauta, ou cadastrar um voto individual manualmente, cruzando automaticamente os e-mails votantes com a lista de conselheiros cadastrada.
- **Ator:** Secretaria
- **Prioridade:** Essencial
- **Entradas e pré-condições:** Reunião e pauta já cadastradas; planilha de votos exportada do Teams (ou dados de um voto individual).
- **Saídas e pós-condições:** Os votos são registrados e classificados conforme o reconhecimento do e-mail votante; e-mails não reconhecidos ficam pendentes de validação manual (RF011).
- **Fluxo de eventos principal**

1. A secretaria acessa a aba "Apuração de Votação" da pauta desejada.
2. A secretaria carrega a planilha de votos correspondente.
3. O sistema cruza cada e-mail votante com a lista de conselheiros cadastrada.
4. Para e-mails reconhecidos, o sistema registra o voto automaticamente.
5. Para e-mails não reconhecidos, o sistema exibe um alerta para validação manual pela secretaria.

- **Fluxos secundários** (alternativos e de exceção)

1. A secretaria pode cadastrar um voto individual manualmente, para quem votou fora da planilha (ex.: pelo chat da reunião).
2. Ao validar/negar um e-mail não reconhecido, a secretaria pode aplicar a mesma decisão às

demais pautas restantes da reunião, evitando repetir a validação para o mesmo e-mail.

2.3.2.9 [RF009] Consultar presença

- **Descrição:** Permite à secretaria consultar, em tempo real, quantos conselheiros com direito a voto estão online e quantos participantes estão ativos na reunião.
- **Ator:** Secretaria
- **Prioridade:** Importante
- **Entradas e pré-condições:** Reunião já cadastrada, com lista de presença importada ao menos uma vez.
- **Saídas e pós-condições:** O sistema exibe o número de conselheiros online com direito a voto, o total de participantes ativos e a lista detalhada de cada um.
- **Fluxo de eventos principal**
  1. A secretaria acessa a aba "Lista de Presença" da reunião desejada.
  2. O sistema exibe os indicadores de quórum e a lista de participantes atualizada.
- **Fluxos secundários** (alternativos e de exceção)
  1. Não há fluxos secundários identificados para este caso de uso.

2.3.2.10 [RF010] Consultar apuração de votação por pauta

- **Descrição:** Permite à secretaria consultar o resultado da apuração de votos de uma pauta, incluindo a quantidade de votos por opção, os votos negados e o detalhamento do reconhecimento e status de cada voto.
- **Ator:** Secretaria
- **Prioridade:** Essencial
- **Entradas e pré-condições:** Pauta com votos já importados ou cadastrados.
- **Saídas e pós-condições:** O sistema exibe a quantidade de votos por opção, um gráfico comparativo, o total de votos válidos, a proporção de conselheiros que votaram, os votos negados, os participantes online que não votaram e a tabela de resultado do cruzamento.
- **Fluxo de eventos principal**

1. A secretaria acessa a aba "Apuração de Votação" da pauta desejada.
2. O sistema exibe o resumo da apuração e a tabela de resultado do cruzamento.

- **Fluxos secundários** (alternativos e de exceção)

1. A secretaria pode baixar a planilha validada com o resultado da apuração, em formato CSV.

2.3.2.11 [RF011] Validar participante

- **Descrição:** Permite à secretaria validar ou negar manualmente a participação de um e-mail votante que não consta na lista de conselheiros cadastrada, decidindo se aquele voto deve ser considerado na apuração da pauta.
- **Ator:** Secretaria
- **Prioridade:** Essencial
- **Entradas e pré-condições:** Um e-mail votante não reconhecido durante a importação de votos (RF008).
- **Saídas e pós-condições:** O voto passa a ter status "Válido" ou "Negado" para a pauta (ou para toda a reunião, conforme a decisão da secretaria).
- **Fluxo de eventos principal**

1. O sistema exibe um alerta de "E-mail não reconhecido" ao identificar um voto de e-mail fora da lista de conselheiros.
2. A secretaria avalia o caso e escolhe "Validar" ou "Negar".
3. A secretaria pode optar por aplicar a mesma decisão às demais pautas restantes da reunião.
4. O sistema registra a decisão e atualiza o status do voto.

- **Fluxos secundários** (alternativos e de exceção)

1. Caso a secretaria não marque a opção de aplicar a decisão às demais pautas, a validação vale só para a pauta atual, podendo o mesmo e-mail gerar novo alerta em pautas seguintes.

2.3.2.12 [RF012] Editar decisão manual

- **Descrição:** Permite à secretaria alterar uma decisão de validação manual já registrada, revertendo o status de válido para negado ou vice-versa.
- **Ator:** Secretaria

- **Prioridade:** Importante
- **Entradas e pré-condições:** Participante com decisão de validação manual já registrada (RF011).
- **Saídas e pós-condições:** O status do voto do participante é atualizado conforme a nova decisão.
- **Fluxo de eventos principal**

  1. A secretaria localiza o participante na lista de presença ou na tabela de resultado do cruzamento.
  2. A secretaria seleciona "Alterar decisão".
  3. A secretaria informa a nova decisão.
  4. O sistema atualiza o status do voto.

- **Fluxos secundários** (alternativos e de exceção)

  1. Não há fluxos secundários identificados para este caso de uso.

2.4 Requisitos Não Funcionais - RNF

2.4.1 Usabilidade

Esta seção descreve os requisitos não funcionais associados à facilidade de uso da interface com o usuário, material de treinamento e documentação do sistema.

2.4.1.1 [RNF001] Interface sem necessidade de treinamento técnico

- **Descrição:** O sistema deve poder ser operado pela secretaria sem necessidade de treinamento técnico especializado ou conhecimento prévio em informática, já que seus usuários não possuem, necessariamente, formação na área.
- **Prioridade:** Essencial
- **Caso(s) de uso associado(s):** Não se aplica (requisito do sistema como um todo).

2.4.1.2 [RNF002] Mensagens claras nos casos de validação/erro

- **Descrição:** O sistema deve apresentar mensagens claras e compreensíveis sempre que uma ação não puder ser concluída (por exemplo, campo obrigatório não preenchido, e-mail

duplicado) ou quando a secretaria precisar validar manualmente um participante não identificado automaticamente.

- ❑ **Prioridade:** Importante
- ❑ **Caso(s) de uso associado(s):** Validação Manual – RF011 (Validar participante).

2.4.2 Confiabilidade

Esta seção descreve os requisitos não funcionais associados à frequência, severidade de falhas do sistema e habilidade de recuperação das mesmas, bem como à corretude do sistema.

2.4.2.1 [RNF001] Não duplicação de participantes entre fontes de dados

- ❑ **Descrição:** O sistema não deve duplicar o registro de um mesmo participante na apuração quando ele for identificado de formas diferentes entre as fontes de dados (por exemplo, por nome em uma planilha de presença e por e-mail em uma planilha de votação).
- ❑ **Prioridade:** Essencial
- ❑ **Caso(s) de uso associado(s):** Presença e Votação – RF007 (Importar presença), RF008 (Importar/cadastrar votos).

2.4.2.2 [RNF002] Consistência dos números de quórum e apuração

- ❑ **Descrição:** Os números de quórum, presença e apuração exibidos em diferentes telas do sistema (*pop-ups*, tabelas, cartões de resumo) devem ser sempre consistentes entre si, refletindo o estado mais atual dos dados.
- ❑ **Prioridade:** Essencial
- ❑ **Caso(s) de uso associado(s):** Presença e Votação – RF007 (Importar presença), RF010 (Consultar apuração).

2.4.2.3 [RNF003] Persistência dos dados entre execuções do sistema

- ❑ **Descrição:** Os dados cadastrados (reuniões, pautas, conselheiros, presença, votos e decisões manuais) devem ser mantidos entre uma execução e outra do sistema, não sendo perdidos ao fechar e reabrir a aplicação.
- ❑ **Prioridade:** Essencial
- ❑ **Caso(s) de uso associado(s):** Não se aplica (requisito do sistema como um

todo).

2.4.3 Desempenho

Esta seção descreve os requisitos não funcionais associados à eficiência, uso de recursos e tempo de resposta do sistema.

2.4.3.1 [RNF001] Atualização do quórum e das listas em tempo real

- **Descrição:** O cálculo do quórum e as listas de conselheiros online/aptos a votar devem ser recalculados e exibidos de forma atualizada, refletindo decisões de validação tomadas pela secretaria durante a própria reunião, sem exigir reimportação de dados.
- **Prioridade:** Importante
- **Caso(s) de uso associado(s):** Presença e Votação – RF007 (Importar presença), RF009 (Consultar presença).

2.4.4 Segurança

Esta seção descreve os requisitos não funcionais associados à integridade, privacidade e autenticidade dos dados do sistema.

2.4.4.1 [RNF001] Não exposição de dados pessoais fora do ambiente local

- **Descrição:** Como o sistema armazena dados pessoais de conselheiros (nome e e-mail), ele deve ser executado localmente, sem transmitir esses dados pela internet ou expô-los a sistemas externos, reduzindo o risco de acesso indevido a essas informações.
- **Prioridade:** Importante
- **Caso(s) de uso associado(s):** Não se aplica (requisito do sistema como um todo).

2.4.5 Distribuição

Esta seção descreve os requisitos não funcionais associados à distribuição da versão executável do sistema.

2.4.5.1 [RNF001] Execução local via scripts de inicialização

- **Descrição:** O sistema deve poder ser iniciado localmente pela secretaria por meio de scripts

de inicialização simples, sem exigir a configuração de um servidor dedicado ou conhecimento técnico para sua implantação.

- **Prioridade:** Importante
- **Caso(s) de uso associado(s):** Não se aplica (requisito do sistema como um todo).

2.4.6 Padrões

Esta seção descreve os requisitos não funcionais associados a padrões ou normas que devem ser seguidos pelo sistema ou pelo seu processo de desenvolvimento.

2.4.6.1 [RNF001] Padronização das rotas da API seguindo convenções REST

- **Descrição:** A comunicação entre *front-end* e *back-end* deve seguir as convenções REST, com rotas organizadas por recurso (por exemplo, /api/reunioes/{id}/votantes) e uso apropriado dos métodos HTTP (GET, POST, DELETE).
- **Prioridade:** Desejável
- **Caso(s) de uso associado(s):** Não se aplica (requisito do sistema como um todo).

2.4.6.2 [RNF002] Compatibilidade do arquivo exportado com planilhas eletrônicas

- **Descrição:** O arquivo CSV exportado pela funcionalidade de exportação de lista de conselheiros aptos deve seguir um formato (separador e codificação de caracteres) compatível com softwares de planilha eletrônica de uso comum, como o Excel.
- **Prioridade:** Desejável
- **Caso(s) de uso associado(s):** Gestão de Conselheiros – RF006 (Exportar lista de aptos).

2.4.7 Hardware e software

Esta seção descreve os requisitos não funcionais associados ao hardware e software usados para desenvolver ou para executar o sistema.

2.4.7.1 [RNF001] Requisitos de ambiente para execução

- ❑ **Descrição:** Para ser executado, o sistema requer um ambiente com Java (JDK) instalado para o *back-end*, Node.js para o *front-end*, e um navegador web atualizado para acesso à interface. Não há exigência de hardware especializado, sendo suficiente um computador comum.
- ❑ **Prioridade:** Essencial
- ❑ **Caso(s) de uso associado(s):** Não se aplica (requisito do sistema como um todo).

2.5 Protótipos das interfaces com usuários

Esta seção apresenta protótipos das principais telas do sistema, obtidos a partir da execução real da aplicação, associando cada uma às funcionalidades descritas na Seção 2.3.

2.5.1 Protótipo Interface - RF001 - (Cadastrar reunião)

O cadastro de uma reunião é feito informando o conselho, a data, o horário e o link da reunião no Microsoft Teams. Após salva, a reunião passa a constar na lista de reuniões cadastradas, com status "Agendada" (Figura 3).

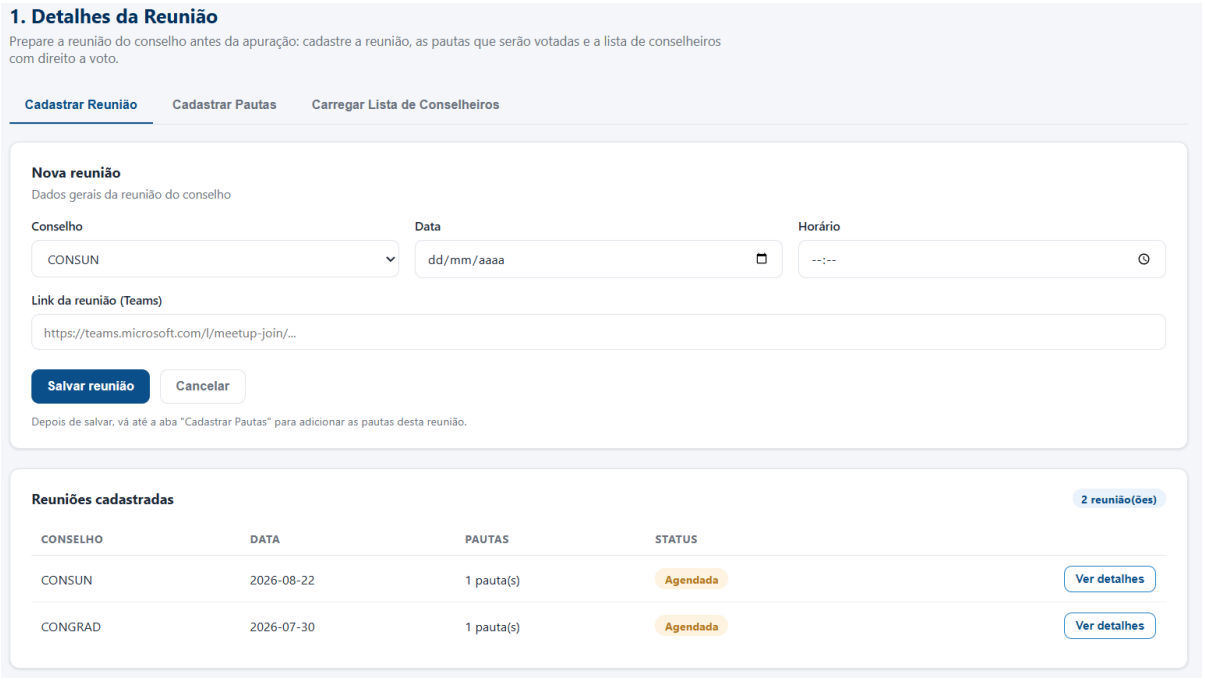


Figura 3 – Protótipo Interface - RF001 (Cadastrar reunião)
(Fonte: Autor)

2.5.2 Protótipo Interface - RF002 - (Cadastrar pauta)

Uma vez cadastrada a reunião, a secretaria pode adicionar as pautas que serão votadas,

informando título e, opcionalmente, uma descrição. As pautas cadastradas ficam listadas abaixo do formulário (Figura 4).

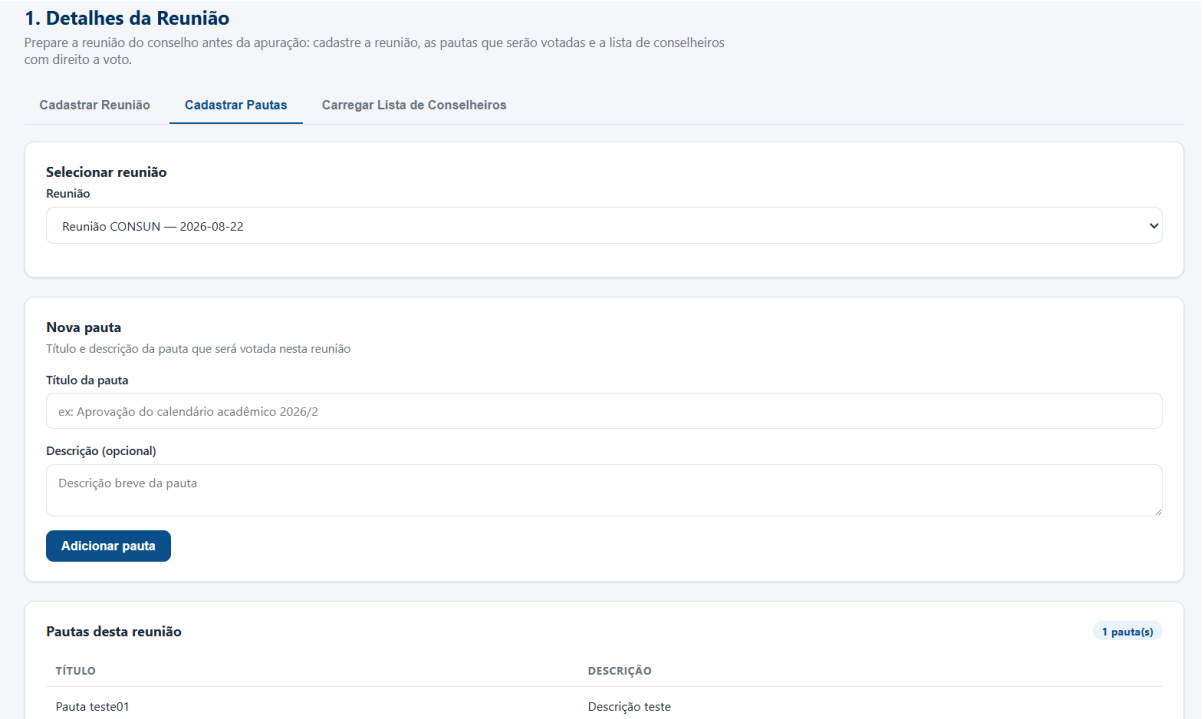


Figura 4 – Protótipo Interface - RF002 (Cadastrar pauta)
(Fonte: Autor)

2.5.3 Protótipo Interface - RF003 - (Cadastrar conselheiro)

O cadastro manual de um conselheiro é feito pelo botão "Adicionar manualmente", disponível na tela de gestão de conselheiros da reunião (Figura 5).

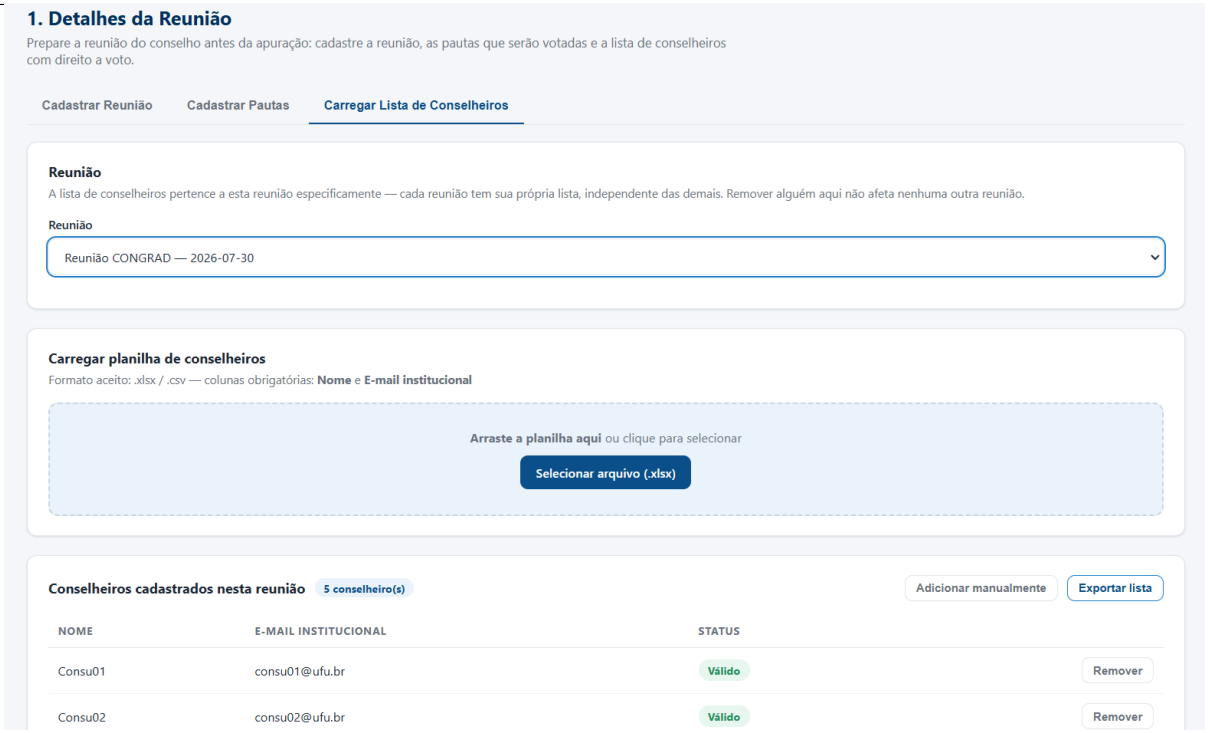


Figura 5 – Protótipo Interface - RF003, RF004, RF005 e RF006 (Cadastrar, importar, remover e exportar lista de conselheiros)
(Fonte: Autor)

2.5.4 Protótipo Interface - RF004 - (Importar conselheiros)

Alternativamente, a lista de conselheiros pode ser importada de uma só vez a partir de uma planilha (.xlsx ou .csv) contendo nome e e-mail institucional, carregada na área "Carregar planilha de conselheiros" (Figura 5).

2.5.5 Protótipo Interface - RF005 - (Remover conselheiros)

Cada conselheiro cadastrado na reunião pode ser removido individualmente por meio do botão "Remover", exibido ao lado de seu registro na lista (Figura 5).

2.5.6 Protótipo Interface - RF006 - (Exportar lista de aptos)

A lista de conselheiros aptos a votar na reunião pode ser exportada a qualquer momento por meio do botão "Exportar lista" (Figura 5).

2.5.7 Protótipo Interface - RF007 - (Importar presença)

A presença dos participantes na reunião é importada a partir da planilha exportada do Microsoft Teams, carregada na área "Carregar planilha de presença" (Figura 6).

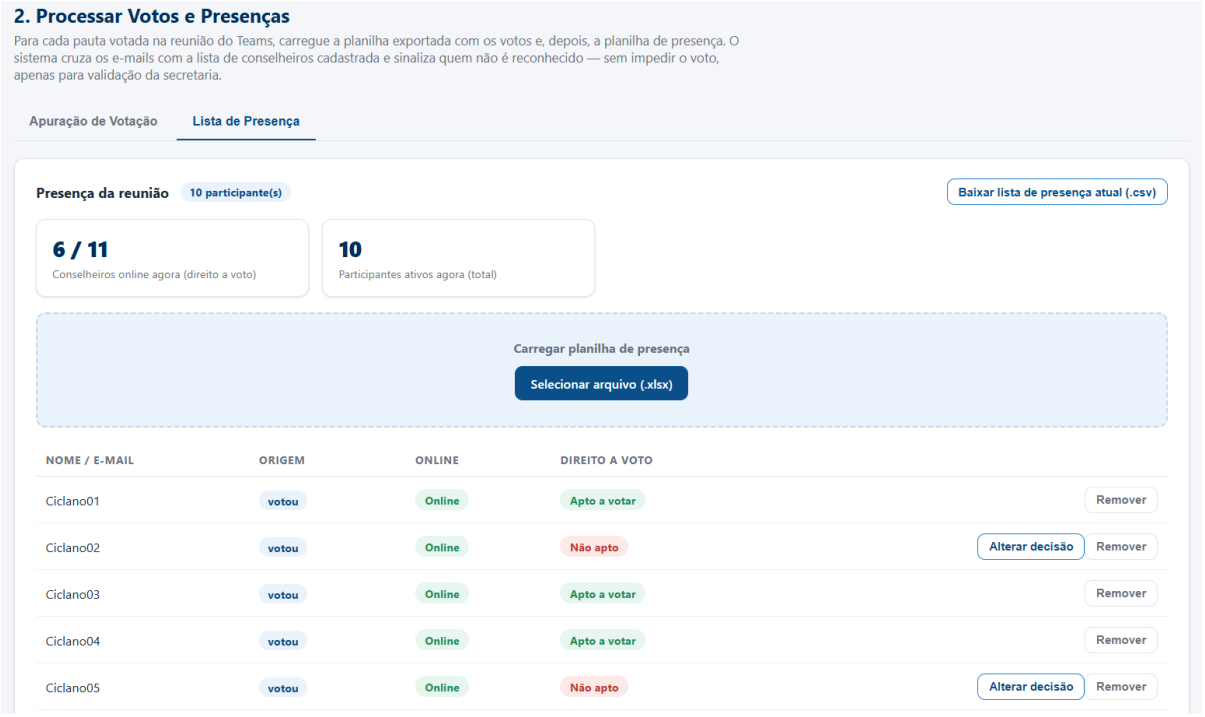


Figura 6 – Protótipo Interface - RF007 e RF009 (Importar e consultar presença)
(Fonte: Autor)

2.5.8 Protótipo Interface - RF008 - (Importar e cadastrar votos)

Ao carregar a planilha de votos de uma pauta, o sistema cruza automaticamente os e-mails votantes com a lista de conselheiros cadastrada. Quando encontra um e-mail que não consta nessa lista, um alerta é exibido para que a secretaria valide ou negue manualmente aquele voto, sem bloqueá-lo. A tela inclui uma opção para aplicar essa mesma decisão às demais pautas restantes da reunião, evitando que o mesmo alerta se repita para o mesmo e-mail a cada nova pauta votada (Figura 7).

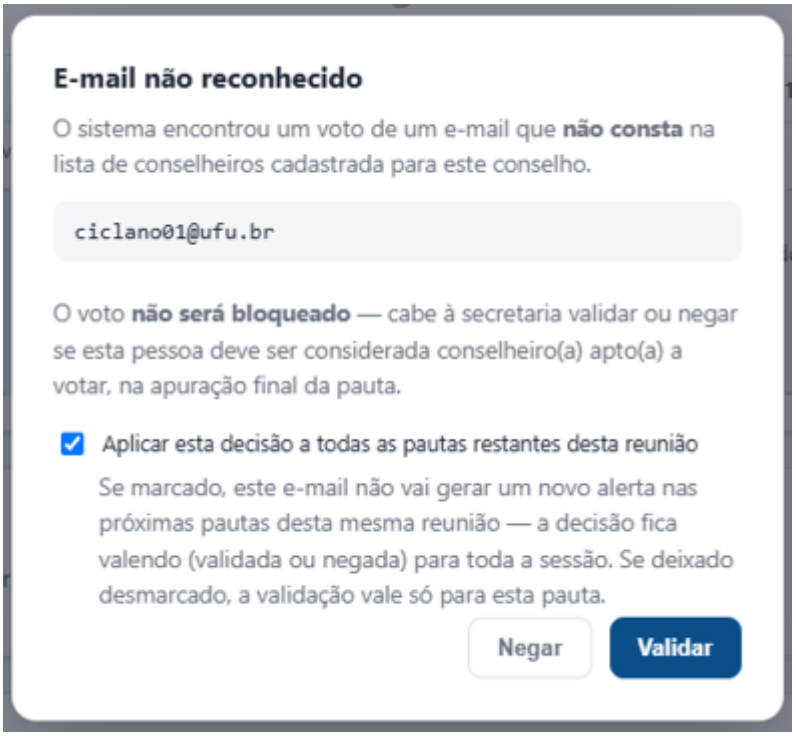


Figura 7 – Protótipo Interface - RF008 (Importar e cadastrar votos)
(Fonte: Autor)

2.5.9 Protótipo Interface - RF009 - (Consultar presença)

O sistema exibe, em destaque, quantos conselheiros com direito a voto estão online no momento e quantos participantes ativos há ao todo, além de listar cada participante com sua origem, status online e se está apto a votar (Figura 6).

2.5.10 Protótipo Interface - RF010 - (Consultar apuração de votação por pauta)

Concluída a apuração de uma pauta, o sistema apresenta a quantidade de votos por opção (tanto os apurados pela planilha quanto os informados manualmente via chat), um gráfico comparativo, um resumo com o total de votos válidos, a proporção de conselheiros que votaram, os votos negados e os participantes online que não votaram, além da tabela detalhada de resultado do cruzamento, com o status de cada voto (Figura 8).

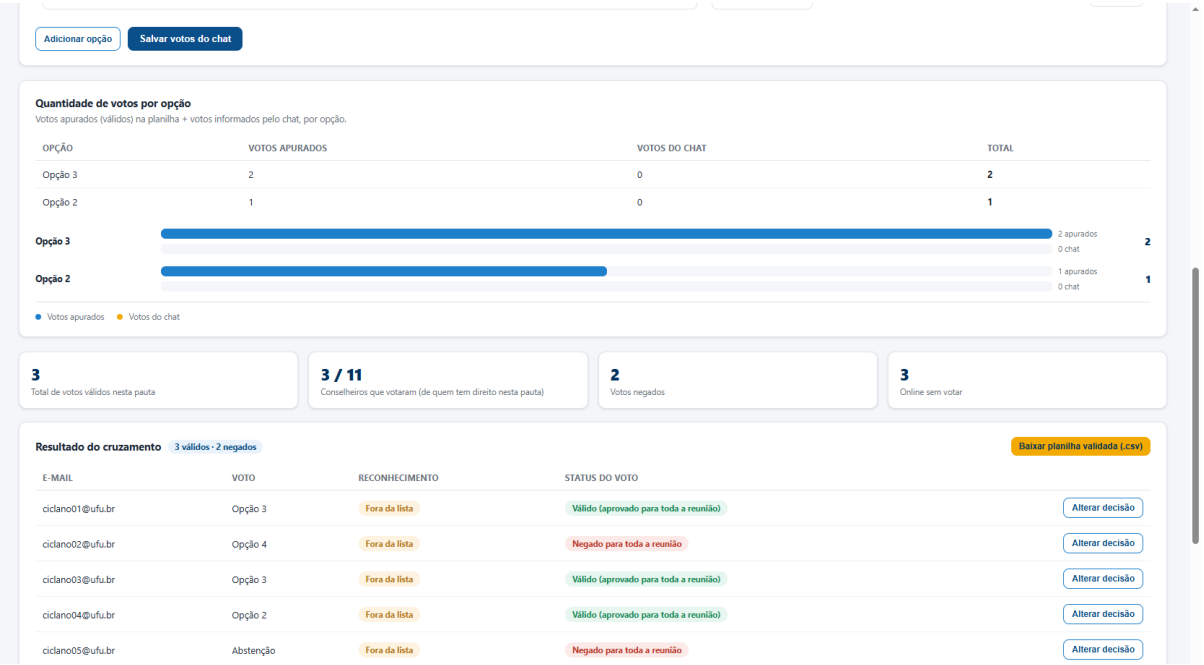


Figura 8 – Protótipo Interface - RF010 (Consultar apuração de votação por pauta)
(Fonte: Autor)

2.5.11 Protótipo Interface - RF011 - (Validar participante)

A validação de participantes não reconhecidos ocorre no mesmo ponto em que o sistema identifica a divergência: o alerta apresentado na Figura 7 permite à secretaria confirmar ("Validar") ou recusar ("Negar") a participação de um e-mail fora da lista de conselheiros.

2.5.12 Protótipo Interface - RF012 - (Editar decisão manual)

Já validado ou negado um voto, a secretaria pode reverter essa decisão a qualquer momento por meio do botão "Alterar decisão", disponível tanto na lista de presença (Figura 6) quanto na tabela de resultado do cruzamento (Figura 8).

2.6 Considerações Sobre o Capítulo

Neste capítulo foram apresentados os requisitos do sistema desenvolvido neste trabalho. Foi descrito o propósito geral do sistema, sua abrangência e o perfil dos usuários a que se destina, seguido do estudo de viabilidade, com os benefícios esperados e o cronograma de desenvolvimento adotado. Em seguida, foram detalhados os doze requisitos funcionais do

sistema, organizados em quatro grupos (Reuniões e Pautas, Gestão de Conselheiros, Presença e Votação, e Validação Manual), bem como os requisitos não funcionais relacionados à usabilidade, confiabilidade, desempenho, segurança, distribuição e padrões adotados. Por fim, foram apresentados os protótipos das interfaces correspondentes a cada requisito funcional, obtidos a partir da execução real do sistema.

CAPÍTULO **3**

Análise e Projeto do Sistema

3.1 Modelo de Domínio do Sistema

O modelo de domínio, apresentado na Figura 9, representa uma primeira visão conceitual das principais entidades do sistema e de seus relacionamentos, servindo de base para o diagrama de classes detalhado apresentado na Seção 3.4.

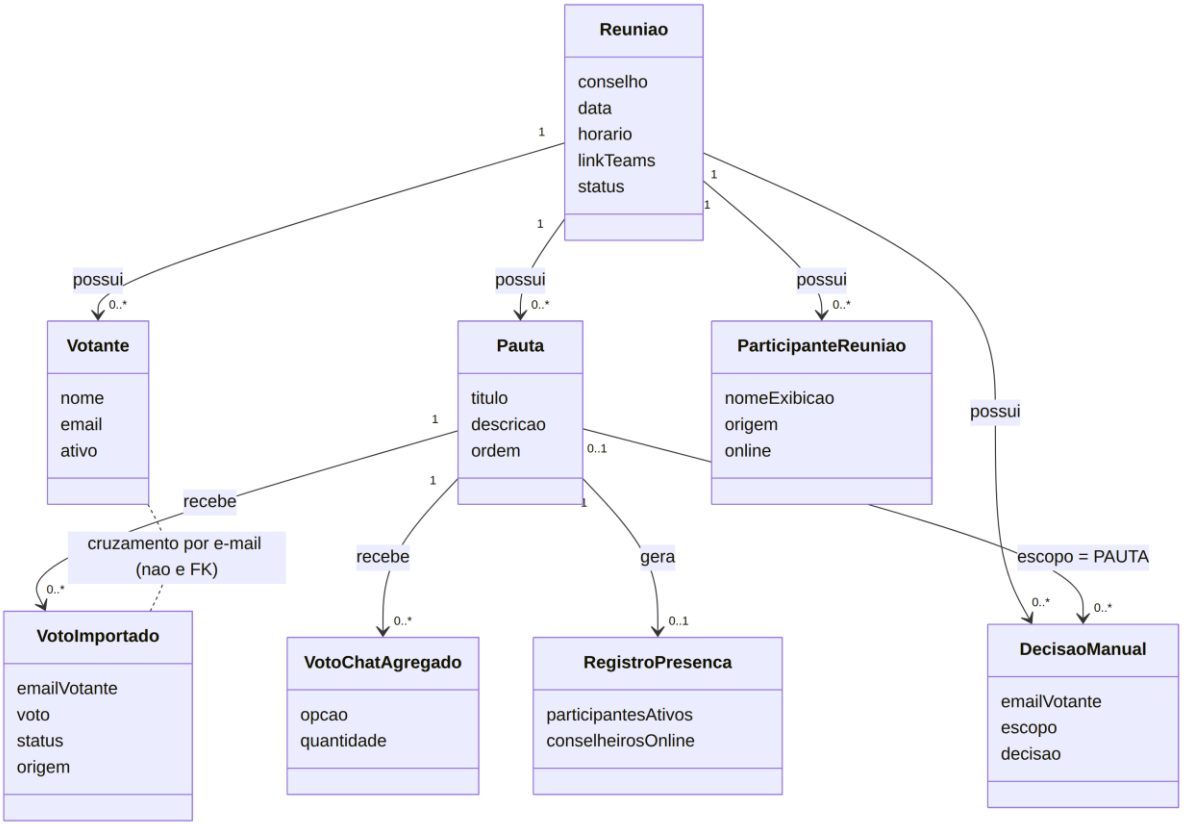


Figura 9 – Modelo de Domínio do Sistema
(Fonte: Autor)

Uma Reunião concentra o escopo de praticamente todas as demais entidades: a ela pertencem as Pautas votadas, os Votantes (conselheiros aptos a votar, cadastrados especificamente para aquela reunião), os Participantes da Reunião (lista de presença editável, obtida a partir das planilhas exportadas do Microsoft Teams) e as Decisões Manuais tomadas pela secretaria. Essa vinculação por reunião é

proposital: a lista de conselheiros de uma reunião não afeta as demais, evitando que uma alteração pontual comprometa o histórico de reuniões anteriores.

Cada Pauta, por sua vez, concentra os elementos do processo de apuração: os Votos Importados (registros individuais extraídos da planilha de votação ou cadastrados manualmente pela secretaria), os Votos do Chat Agregados (contagens anônimas informadas manualmente pela secretaria) e um Registro de Presença, que armazena o retrato numérico do quórum no momento em que aquela pauta foi apurada. Uma Pauta também pode estar associada a Decisões Manuais quando o escopo da decisão é restrito a ela, diferentemente das decisões de escopo "reunião", que valem para todas as pautas, inclusive as futuras.

Por fim, o relacionamento entre Votante e Voto Importado não corresponde a uma referência direta (chave estrangeira), mas sim a um cruzamento realizado em tempo de execução a partir do e-mail do votante. Esse cruzamento é o mecanismo central do sistema, responsável por identificar automaticamente quais votos pertencem a conselheiros cadastrados e quais precisam de validação manual por parte da secretaria.

3.2 Diagramas de Sequência do Sistema

3.2.1 Diagramas de Sequência - RF008 – (Importar e cadastrar votos)

A primeira parte do fluxo desencadeado quando a secretaria importa a planilha de votos de uma pauta está na Figura 10. O `CruzamentoVotosService` lê cada linha da planilha, registra o participante na lista de presença da reunião (caso ainda não conste nela, via `PresencaGeralService`), calcula o status de validação do voto (delegando ao `ReconhecimentoVotanteService`, que verifica, nessa ordem de precedência, se o e-mail consta na lista de conselheiros cadastrada, se há uma decisão manual da secretaria para toda a reunião ou se há uma decisão manual restrita àquela pauta) e grava o voto no banco de dados. Ao final do laço, o retrato de presença da pauta é recalculado.

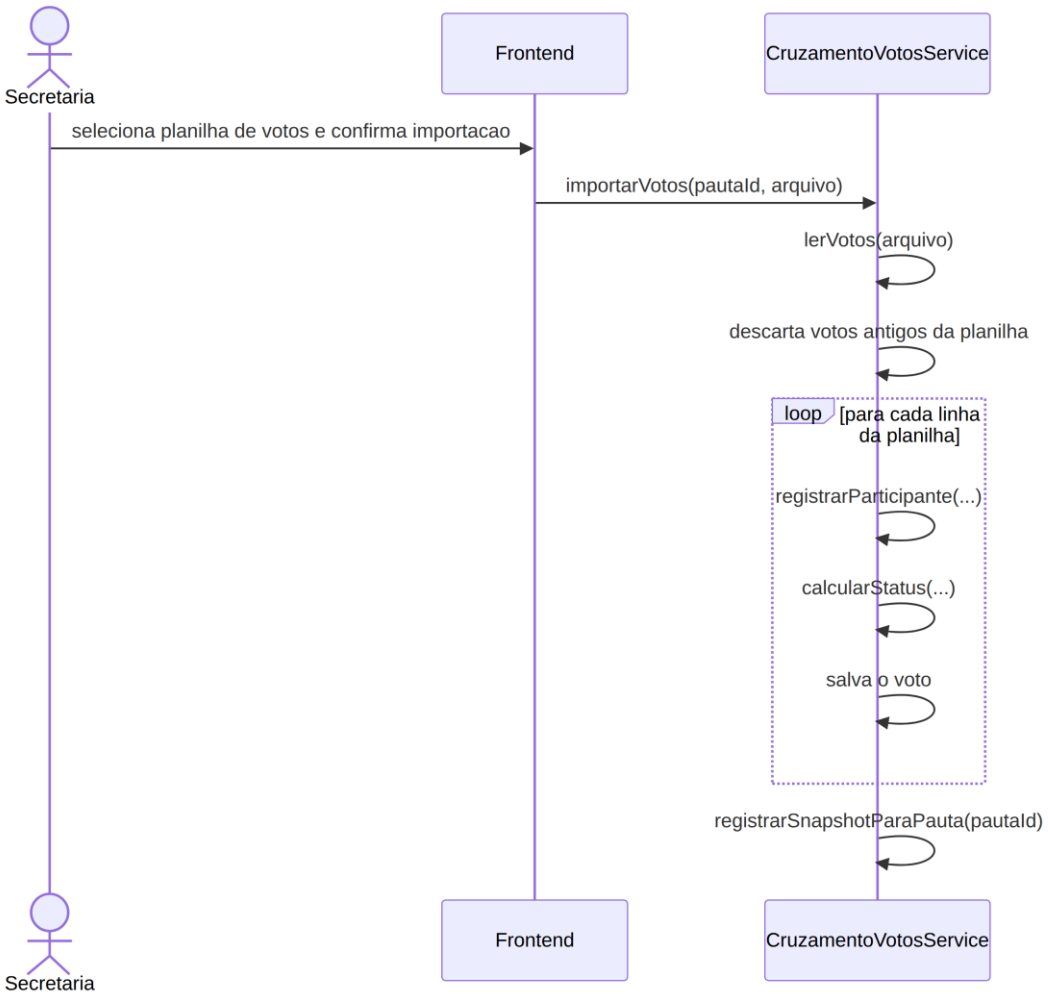


Figura 10 – Diagrama de Sequência - RF008, parte 1 (leitura da planilha e cálculo de status)
(Fonte: Autor)

A segunda parte está na Figura 11: a resposta devolvida ao *front-end* contém o resultado do cruzamento e a fila de e-mails ainda pendentes de decisão. Se houver pendências, o *front-end* abre o *pop-up* de validação (Figura 7), um e-mail de cada vez; caso contrário, exibe diretamente o resultado do cruzamento (Figura 8).

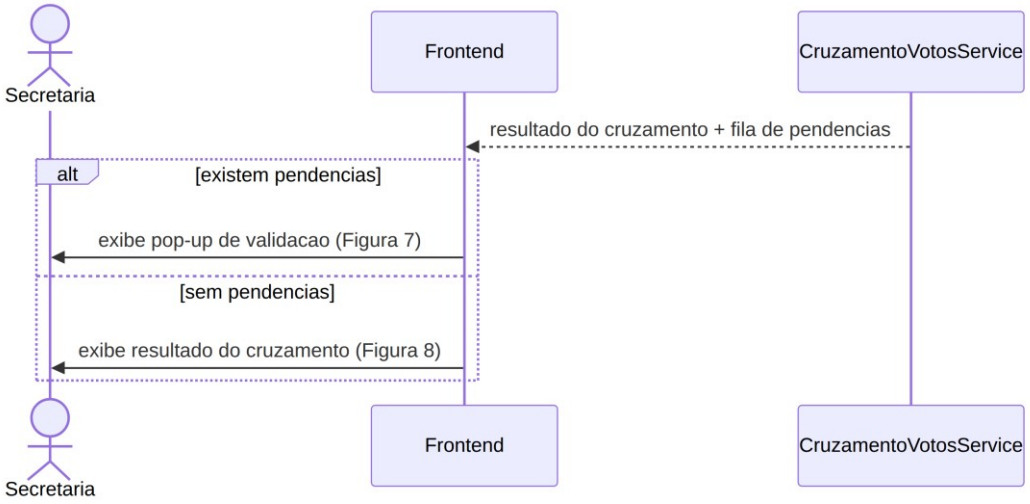


Figura 11 – Diagrama de Sequência - RF008, parte 2 (resposta e fila de pendências)
(Fonte: Autor)

3.2.2 Diagramas de Sequência - RF011 – (Validar participante)

A primeira parte do fluxo desencadeado quando a secretaria valida ou nega, no *pop-up* de validação, um e-mail não reconhecido está na Figura 12. Se o e-mail já constar na lista de conselheiros cadastrada, a operação é rejeitada, pois nesse caso não há necessidade de decisão manual. Caso contrário, o sistema grava a decisão no escopo indicado (apenas a pauta atual ou toda a reunião) e remove qualquer decisão anterior no escopo oposto, para que não permaneça uma decisão desatualizada no banco de dados.

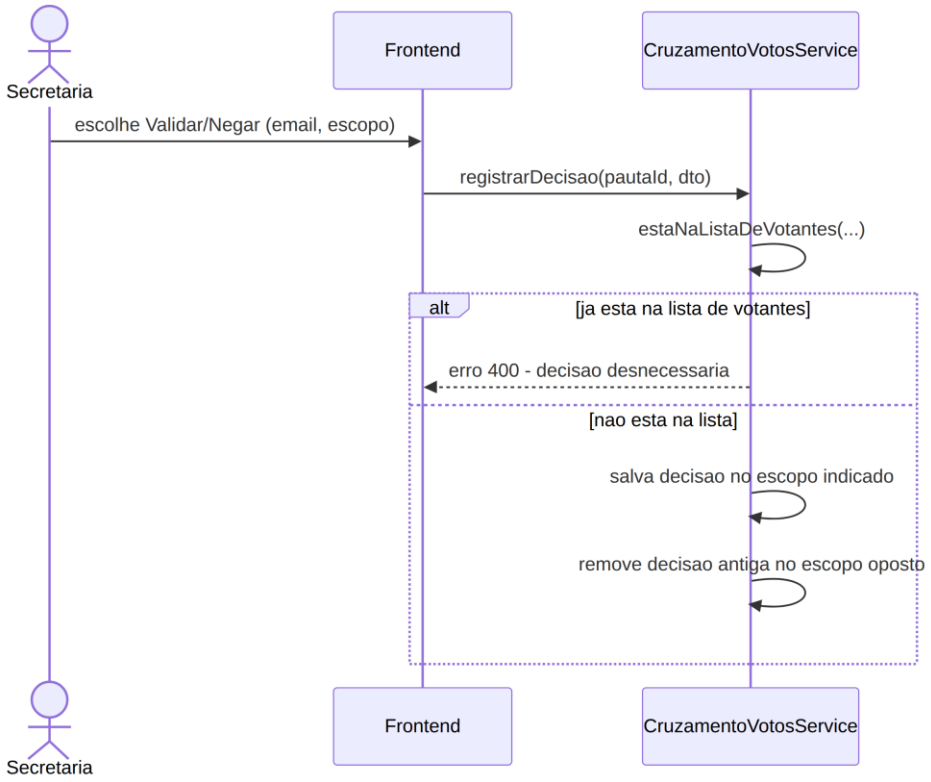


Figura 12 – Diagrama de Sequência - RF011, parte 1 (verificação e gravação da decisão)
(Fonte: Autor)

A segunda parte está na Figura 13: o status de todos os votos já importados daquele e-mail em qualquer pauta da reunião é recalculado, e o retrato de presença correspondente é atualizado (da pauta atual, se o escopo for "pauta", ou de todas as pautas da reunião, se o escopo for "reunião"). Por fim, o *front-end* atualiza a tabela de resultado (Figura 8) e fecha o *pop-up*.

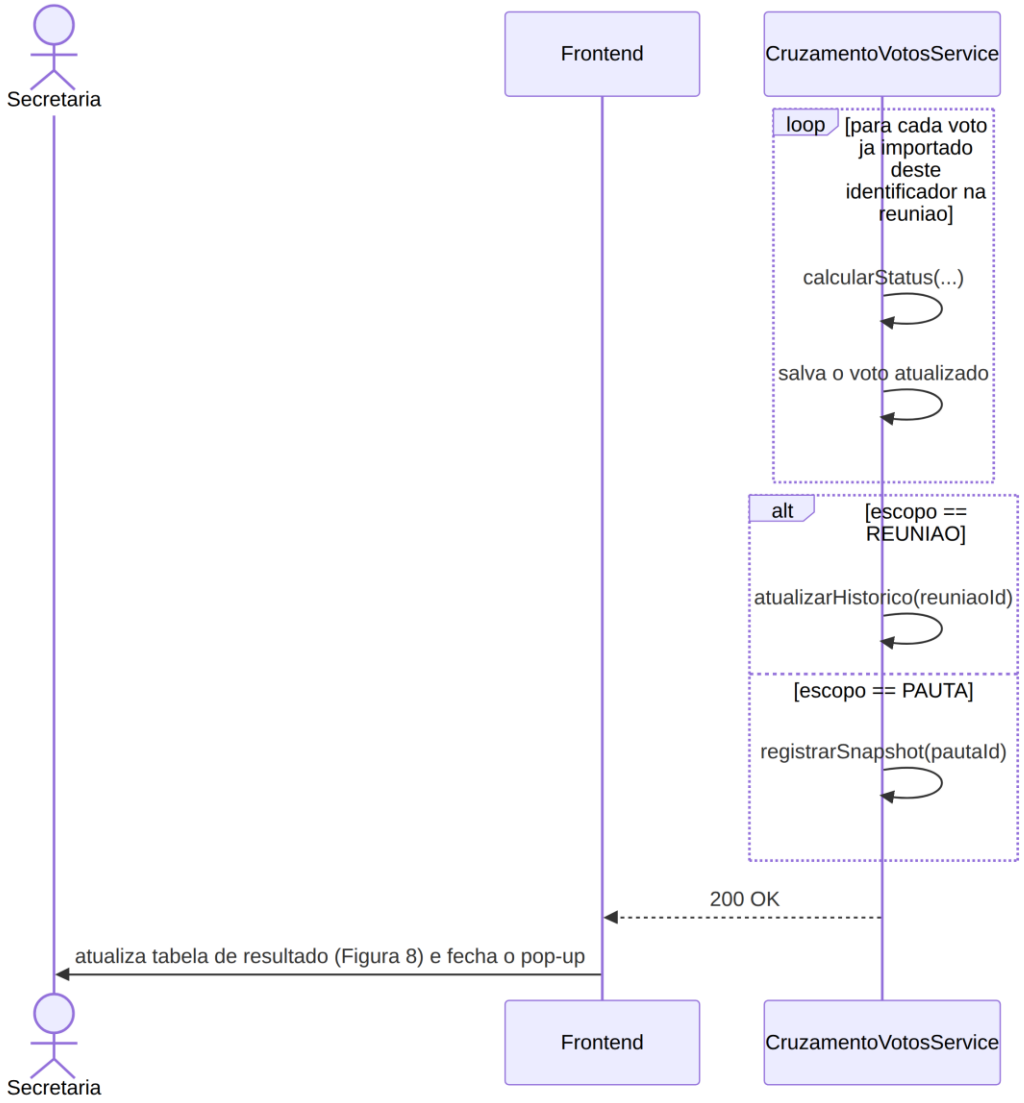


Figura 13 – Diagrama de Sequência - RF011, parte 2 (recálculo de status e atualização de presença)
(Fonte: Autor)

3.3 Modelagem de Dados do Sistema

3.3.1 Modelo Lógico de Dados do Sistema

O modelo lógico resulta do mapeamento objeto-relacional (JPA/Hibernate) do diagrama de classes: cada classe anotada com `@Entity` vira uma tabela, e os relacionamentos viram chaves estrangeiras. O resultado já nasce normalizado até a 3FN: chave primária substituta e ausência de grupos repetitivos (1FN); atributos dependendo integralmente da chave, sempre única por tabela (2FN); e nenhuma dependência transitiva entre atributos não-chave (3FN) e o e-mail de um votante depende só do seu próprio registro, nunca da reunião a que pertence.

Por legibilidade, o modelo foi dividido em duas figuras: reunião (Figura 14) e pauta (Figura 15) com tipos abreviados: id (numérico), str (texto), int (inteiro), bool (booleano), date, time e ts (data e hora).

Na Figura 14 as tabelas de pauta, votante, participante_reuniao e decisao_manual relacionam-se a reuniao por reuniao_id: a lista de conselheiros (votante) e a de presença (participante_reuniao) pertencem a cada reunião individualmente, não ao sistema todo, por isso não existe uma tabela global de conselheiros.

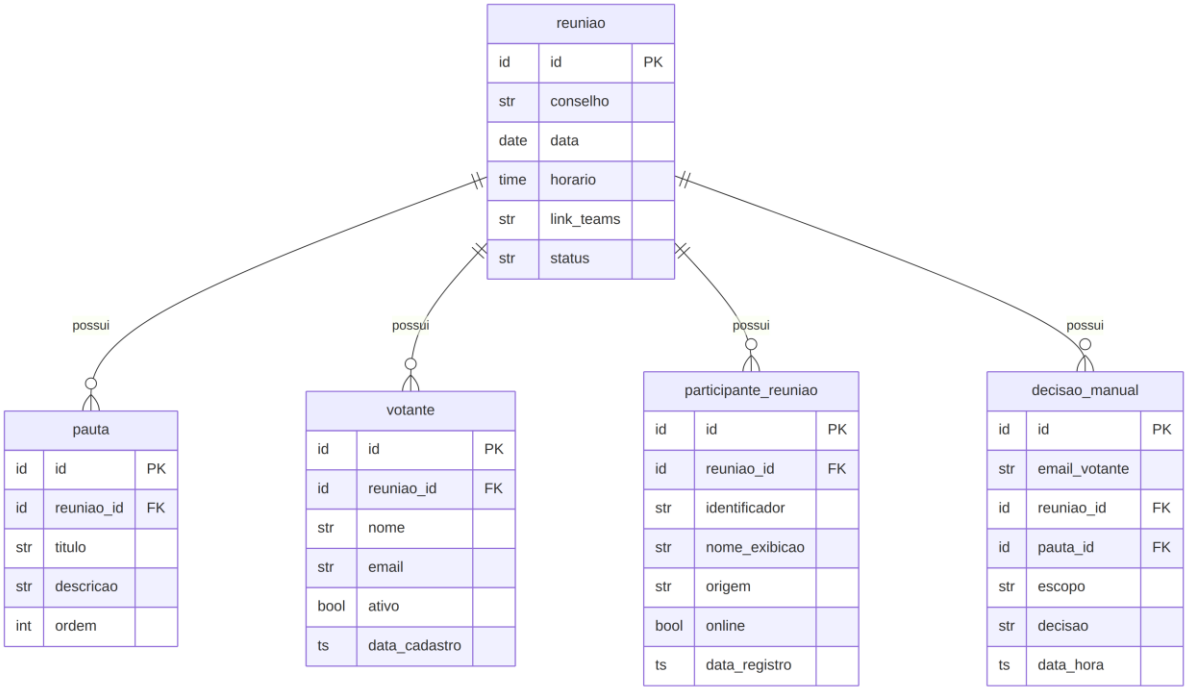


Figura 14 – Modelo Lógico de Dados - tabelas do escopo de uma reunião
(Fonte: Autor)

As tabelas vinculadas a uma pauta específica como voto_importado, voto_chat_agregado e registro_presenca, relacionam-se a pauta por pauta_id (Figura 15). A tabela decisao_manual reaparece porque, quando seu escopo é "pauta" (e não "reunião"), o campo pauta_id (opcional) é preenchido para

restringir a decisão àquela pauta específica; por isso ela é a única tabela do modelo com duas chaves estrangeiras concorrentes: reuniao_id, sempre preenchida, e pauta_id, preenchida apenas quando aplicável.

O campo `email_votante` de `voto_importado` não é chave estrangeira para `votante.email`, e essa ausência de vínculo é proposital: identificar e-mails da planilha sem correspondência na lista de votantes é um dos objetivos do sistema, e uma FK transformaria esse caso em uma violação de integridade.

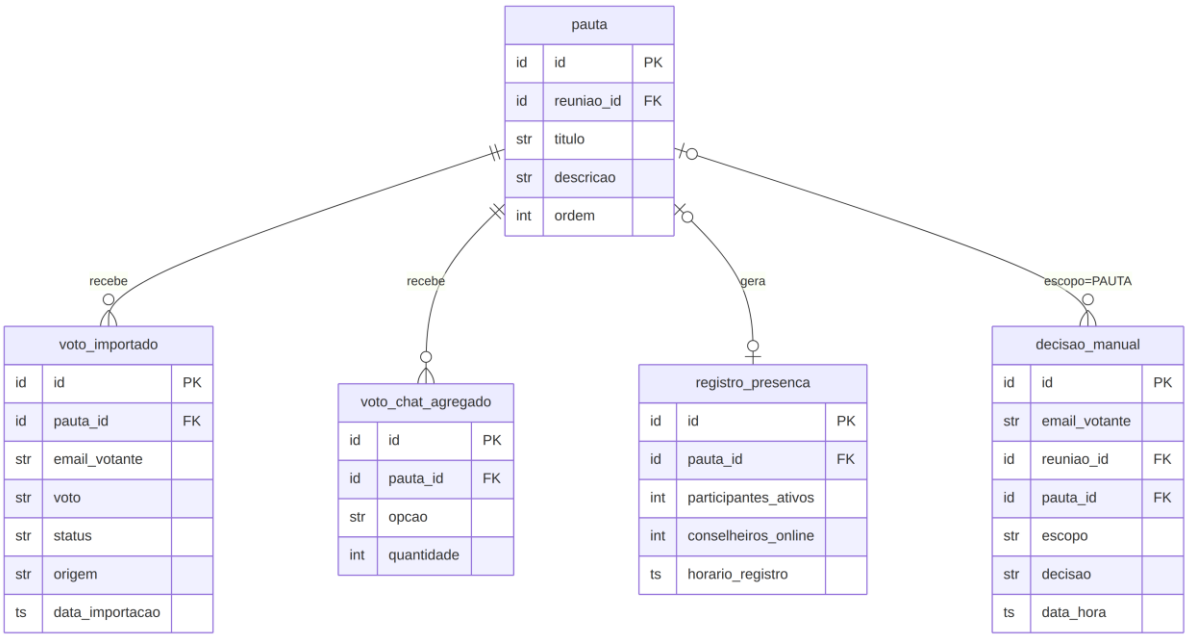


Figura 15 – Modelo Lógico de Dados - tabelas do escopo de uma pauta
(Fonte: Autor)

3.3.2 Modelo Físico de Dados do Sistema

O banco de dados físico é um H2 embarcado, gravado em arquivo local (sem necessidade de instalar um servidor separado), com o schema criado automaticamente pelo Hibernate a partir das entidades `@Entity` (opção `ddl-auto: update`). Não há scripts de migração versionados (Flyway/Liquibase), simplificação deliberada, adequada ao escopo deste TCC. O DDL equivalente ao gerado pelo Hibernate para as oito tabelas do modelo lógico (Seção 3.3.1) está reproduzido no Quadro 2.

Quadro 2 – Script de criação do banco de dados (DDL)

```
create table reuniao (
    id bigint generated by default as identity,
    conselho varchar(255) not null,
    data date not null,
    horario time not null,
    link_teams varchar(255),
    status varchar(255) not null,
    primary key (id)
);

create table pauta (
    id bigint generated by default as identity,
    reuniao_id bigint not null,
    titulo varchar(255) not null,
    descricao varchar(1000),
    ordem integer not null,
    primary key (id)
);

create table votante (
    id bigint generated by default as identity,
    reuniao_id bigint not null,
    nome varchar(255) not null,
    email varchar(255) not null,
    ativo boolean not null,
    data_cadastro timestamp not null,
    primary key (id)
);

create table participante_reuniao (
    id bigint generated by default as identity,
    reuniao_id bigint not null,
    identificador varchar(255) not null,
    nome_exibicao varchar(255) not null,
    origem varchar(255) not null,
    online boolean not null,
    data_registro timestamp not null,
    primary key (id)
);

create table registro_presenca (
    id bigint generated by default as identity,
    pauta_id bigint not null,
    participantes_ativos integer not null,
    conselheiros_online integer not null,
    horario_registro timestamp not null,
    primary key (id)
);

create table voto_importado (
    id bigint generated by default as identity,
    pauta_id bigint not null,
    email_votante varchar(255) not null,
    voto varchar(500) not null,
    status varchar(255) not null,
    origem varchar(255) not null,
    data_importacao timestamp not null,
    primary key (id)
);

create table voto_chat_agregado (
    id bigint generated by default as identity,
    pauta_id bigint not null,
    opcao varchar(500) not null,
    quantidade integer not null,
    primary key (id)
);
```

```
create table decisao_manual (
    id bigint generated by default as identity,
    email_votante varchar(255) not null,
    reuniao_id bigint not null,
    pauta_id bigint,
    escopo varchar(255) not null,
    decisao varchar(255) not null,
    data_hora timestamp not null,
    primary key (id)
);

alter table pauta add constraint fk_pauta_reuniao
    foreign key (reuniao_id) references reuniao (id);
alter table votante add constraint fk_votante_reuniao
    foreign key (reuniao_id) references reuniao (id);
alter table votante add constraint uk_votante_reuniao_email
    unique (reuniao_id, email);
alter table participante_reuniao add constraint fk_participante_reuniao
    foreign key (reuniao_id) references reuniao (id);
alter table participante_reuniao add constraint uk_participante_ident
    unique (reuniao_id, identificador);
alter table registro_presenca add constraint fk_registro_presenca_pauta
    foreign key (pauta_id) references pauta (id);
alter table voto_importado add constraint fk_voto_importado_pauta
    foreign key (pauta_id) references pauta (id);
alter table voto_chat_agregado add constraint fk_voto_chat_pauta
    foreign key (pauta_id) references pauta (id);
alter table voto_chat_agregado add constraint uk_voto_chat_pauta_opcao
    unique (pauta_id, opcao);
alter table decisao_manual add constraint fk_decisao_reuniao
    foreign key (reuniao_id) references reuniao (id);
alter table decisao_manual add constraint fk_decisao_pauta
    foreign key (pauta_id) references pauta (id);
```

(Fonte: Autor)

3.4 Diagrama de Classe do Sistema

O diagrama de classes completo refina o modelo de domínio (Seção 3.1) com os detalhes reais de implementação: tipos de atributo e a distinção entre composição e associação simples, conforme as anotações JPA de cada entidade. Por legibilidade, foi dividido em quatro figuras: classes do escopo de `Reuniao` (Figura 16), classes de `Pauta` vinculadas a `VotoImportado` e `VotoChatAgregado` (Figura 17), classes de `Pauta` vinculadas a `RegistroPresenca` e `DecisaoManual` (Figura 18), e as enumerações usadas como tipo de atributo (Figura 19).

Na Figura 16, `Reuniao` mantém, em composição (losango preenchido), as coleções `pautas` e `votantes`: os objetos `Pauta` e `Votante` são criados e destruídos junto com a `Reuniao` a que pertencem (`cascade = CascadeType.ALL, orphanRemoval = true` no código), refletindo a regra de que a lista de conselheiros de uma reunião não sobrevive nem afeta outras reuniões. Já `ParticipanteReuniao` e `DecisaoManual` apontam para `Reuniao` apenas por associação simples (`@ManyToOne` unidirecional): a classe `Reuniao` não mantém uma coleção desses objetos: eles são consultados sob demanda pelos respectivos repositórios, não carregados automaticamente junto com a reunião.

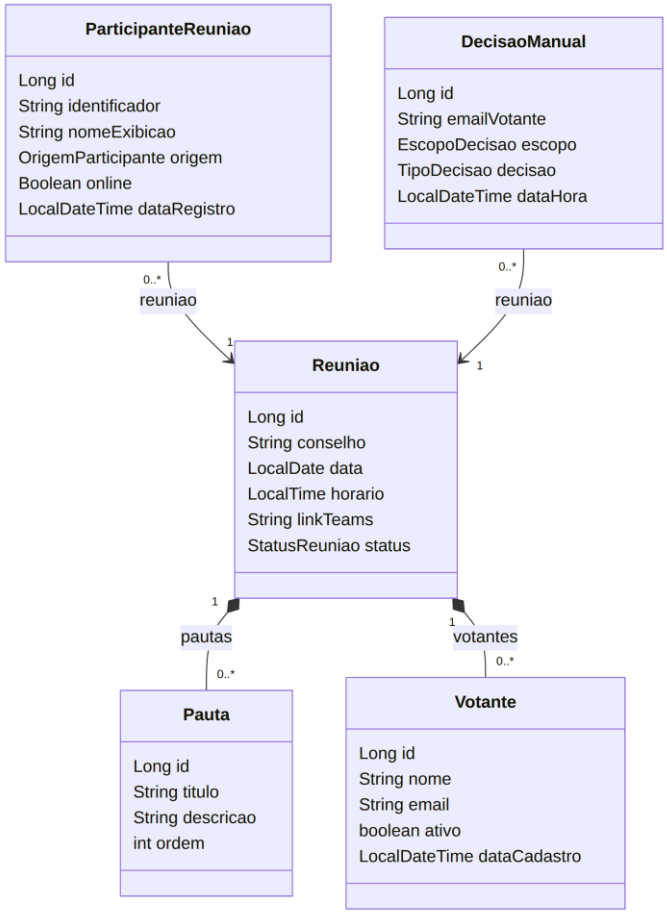


Figura 16 – Diagrama de Classe - escopo de Reuniao (Fonte: Autor)

Nas Figuras 17 e 18, as classes vinculadas a `Pauta` relacionam-se todas por associação simples, pois nenhuma é mantida em coleção por `Pauta`, ao contrário da composição vista em `Reuniao`: `VotoImportado` e `VotoChatAgregado` (Figura 17); `RegistroPresenca` e `DecisaoManual` (Figura 18), esta última repetida da Figura 16 porque sua chave `pauta` é opcional, preenchida só quando o escopo da decisão é `PAUTA`, e não `REUNIAO`.

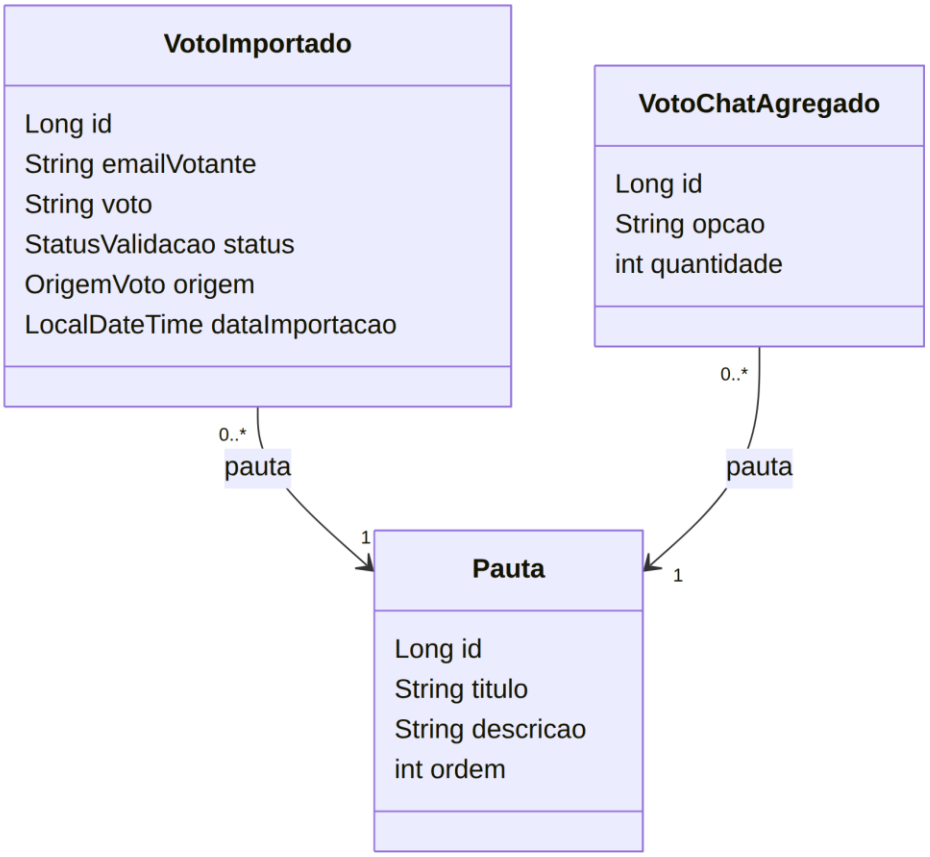


Figura 17 – Diagrama de Classe - Pauta, `VotoImportado` e `VotoChatAgregado` (Fonte: Autor)

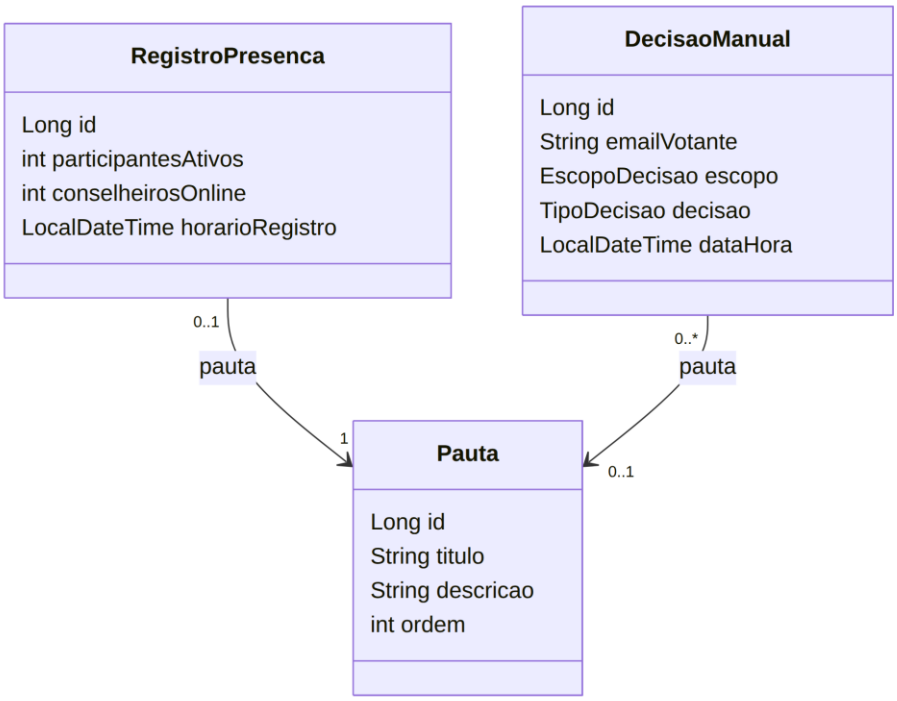


Figura 18 – Diagrama de Classe - Pauta, `RegistroPresenca` e `DecisaoManual` (Fonte: Autor)

Na Figura 19, as seis enumerações usadas como tipo de atributo nessas classes aparecem com seus valores possíveis, já detalhados nas seções anteriores: `StatusReuniao` (`status` de `Reuniao`), `OrigemParticipante` (`origem` de `ParticipanteReuniao`), `EscopoDecisao` e `TipoDecisao`

(`escopo` e `decisao` de `DecisaoManual`), `OrigemVoto` e `StatusValidacao` (`origem` e `status` de `VotoImportado`).

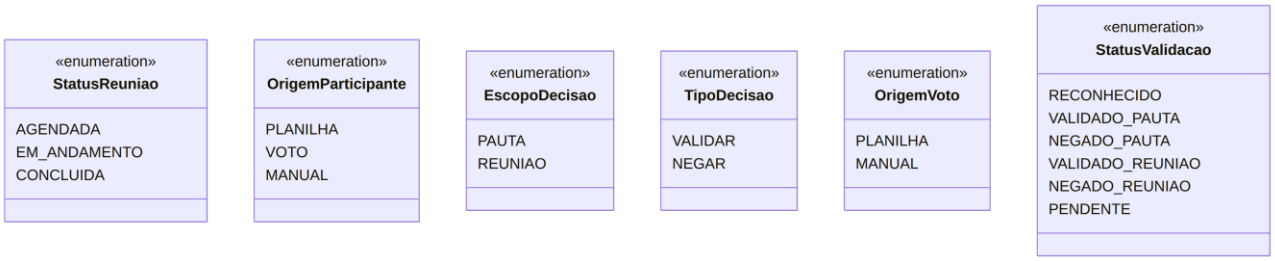


Figura 19 – Diagrama de Classe - enumerações
(Fonte: Autor)

3.5 Considerações Sobre o Capítulo

Neste capítulo, foram abordados os diagramas de domínio, de sequência, de dados e de classes do sistema, todos elaborados a partir do código-fonte real da implementação, que serviram como base e auxílio para o seu desenvolvimento. Os diagramas de domínio e de classes detalham as entidades do sistema e seus relacionamentos, revelando, por exemplo, que apenas `pautas` e `votantes` pertencem de fato a uma `Reuniao`, em composição, enquanto os demais relacionamentos, como o registro de presença e as decisões manuais, são simples associações consultadas sob demanda pelos respectivos repositórios. Os diagramas de sequência detalham os dois fluxos mais relevantes do sistema, o cruzamento automático de votos (RF008) e a validação manual de um participante não reconhecido (RF011), conectando a lógica de negócio descrita no Capítulo 2 às classes e aos métodos que efetivamente a implementam. Já os diagramas de dados detalham o mapeamento dessas entidades para o banco de dados, do modelo lógico, normalizado até a terceira forma normal, ao modelo físico gerado automaticamente pelo Hibernate, incluindo o script de criação das tabelas e das chaves estrangeiras.

Portanto, esses diagramas ajudaram a organizar e compreender a estrutura do sistema, suas entidades, fluxos de uso e banco de dados, servindo de base para a apresentação do seu desenvolvimento no Capítulo 4.

CAPÍTULO 4

Desenvolvimento do Sistema

4.1 Processo de Desenvolvimento

O desenvolvimento do sistema seguiu um processo iterativo e incremental, sem um plano fechado definido por completo antes do início da implementação. Cada funcionalidade foi construída individualmente, do *back-end* ao *front-end*, e testada com dados reais (planilhas de votos e listas de presença exportadas do Teams, e listas de conselheiros dos conselhos da universidade) antes de se avançar para a seguinte. A ordem de implementação acompanhou as dependências naturais dos próprios dados: primeiro o cadastro de reuniões, pautas e conselheiros (RF001 a RF004), depois a importação da presença (RF007), e só então o cruzamento automático de votos (RF008) e a validação manual de participantes não reconhecidos (RF011).

Esse ciclo também incluiu a validação da regra: a precedência entre um e-mail reconhecido na lista de votantes, uma decisão manual de reunião e uma decisão manual de pauta, incluindo o caso do conselheiro que chega atrasado, quando uma decisão tomada para toda a reunião numa pauta anterior precisa valer automaticamente nas pautas seguintes, sem novo *pop-up* de validação. Essa lógica, concentrada na classe `CalculadoraStatusVoto`, chegou a ser verificada, no início do projeto, por meio de um programa em Java puro, sem depender do Maven. Assim que o ambiente passou a ter acesso às dependências necessárias, os mesmos cenários foram formalizados como uma suíte de testes automatizados em JUnit 5, que passou a rodar junto do restante do projeto.

4.2 Arquitetura do Sistema

A arquitetura do sistema segue o modelo cliente/servidor, com a camada de apresentação, a interface Angular, separada da camada de aplicação, a API REST em Spring Boot, que por sua vez concentra o acesso à camada de dados, o banco H2

integrado no próprio processo do servidor, sem um servidor de banco de dados separado. Diferentemente de uma arquitetura distribuída em várias máquinas, aqui as duas camadas lógicas rodam fisicamente no mesmo computador, comunicando-se por HTTP/REST através de http://localhost:8080/api, a URL base definida no *front-end*.

Do lado do servidor, a pilha de software é composta por Java 17, *Spring Boot* 3.2.5 com *Tomcat* integrado e o banco H2 gravado em arquivo local; do lado do cliente, o *front-end* é construído em Angular 17 e executado no navegador da secretaria, provido durante o desenvolvimento pelo próprio `ng serve` na porta 4200. Não há requisito de hardware específico além do necessário para rodar essa pilha. Qualquer computador com JDK e Node.js instalados, e com espaço em disco suficiente para o arquivo do banco H2 — que permanece pequeno dado o volume de dados do sistema, poucas reuniões e poucos conselheiros por conselho —, já atende a esse requisito.

Em termos de rede, toda a comunicação ocorre em *localhost*, sem depender de rede local ou de acesso à internet durante o uso normal do sistema. A internet só é necessária uma vez, na configuração inicial, para o Maven e o npm baixarem as dependências do projeto. O sistema também não foi dimensionado para conexões concorrentes: é operado por uma única pessoa da secretaria por vez, e o próprio *back-end* restringe, por configuração de CORS, as origens aceitas a http://localhost:4200, o que reforça que não há suporte a múltiplos clientes simultâneos.

4.3 Padrões de Projeto Utilizados

O sistema criado não se utiliza de uma coleção extensa de padrões de projeto formais; em vez disso, apoia-se nos padrões já consolidados pelo próprio ecossistema Spring, complementados por algumas escolhas deliberadas nos pontos em que um problema recorrente de design realmente se colocou. O padrão DTO (Data Transfer Object) separa o formato de dados trocado pela API do formato interno das entidades JPA: o pacote `dto` reúne catorze classes, entre elas `PautaDTO`, `VotoResultadoDTO` e `ConselheiroAptoDTO`, cada uma expondo apenas os campos necessários para a tela ou operação correspondente, sem acoplar o *front-end* à estrutura de persistência do banco. O acesso ao banco de dados, por sua vez, segue o padrão Repository, viabilizado pelo Spring Data JPA: as oito interfaces do pacote `repository` (como `DecisaoManualRepository` e `VotanteRepository`) estendem `JpaRepository`, e o Spring gera a implementação e as consultas automaticamente a partir do nome dos métodos, como em `findByReuniaoIdAndEmailVotanteIgnoreCaseAndEscopo`.

As dependências entre as classes de serviço são resolvidas por injeção de dependência, com o

Spring montando o grafo de objetos automaticamente: em vez de campos anotados individualmente, cada serviço declara suas dependências como atributos `private final` recebidos pelo construtor, como em `CruzamentoVotosService`, o que deixa explícito de quais outros componentes cada classe depende. O tratamento de erros, por outro lado, é centralizado num único ponto: a classe `GlobalExceptionHandler`, anotada com `@RestControllerAdvice`, converte exceções como `RegraNegocioException`, falhas de validação e arquivos grandes demais em respostas HTTP padronizadas (`ErroRespostaDTO`), evitando repetir o mesmo tratamento em cada controller e dando ao *front-end* um formato de erro único para consumir.

Por fim, a regra de precedência de validação de voto, já mencionada na Seção 4.1, segue o padrão de classe utilitária: `CalculadoraStatusVoto` é uma classe `final`, com construtor privado e apenas métodos estáticos, deliberadamente isolada do Spring e do JPA. Foi justamente esse isolamento que permitiu validar a lógica de forma independente, como descrito naquela seção.

4.4 Tecnologias Utilizadas no Desenvolvimento

O *back-end* foi desenvolvido em Java 17, sobre o *framework* Spring Boot 3.2.5, usando os módulos `spring-boot-starter-web` (API REST), `spring-boot-starter-data-jpa` (persistência) e `spring-boot-starter-validation` (validação de DTOs). O *front-end* foi desenvolvido em TypeScript, sobre o *framework* Angular 17, com a biblioteca RxJS para a parte reativa da aplicação. O banco de dados é o H2, integrado no próprio processo do *back-end* e gravado em arquivo local, dispensando a instalação de um servidor de banco separado, conforme detalhado na Seção 3.3.2.

A presença das duas bibliotecas adicionais resolve necessidades específicas do sistema: o Apache POI lê e grava planilhas Excel (.xlsx), usado na importação das planilhas de votos e de presença exportadas do Teams, e o Lombok reduz o código repetitivo de getters, setters e construtores nas classes de entidade. O gerenciamento de dependências e o *build* seguem as ferramentas padrão de cada linguagem: Maven no *back-end* e npm com o Angular CLI no *front-end*. Os testes automatizados do *back-end* usam JUnit 5, incluído pelo módulo `spring-boot-starter-test`.

Para o desenvolvimento, não houve necessidade de equipamentos ou rede além de um computador convencional com JDK 17, Maven e Node.js instalados, e acesso à internet para o download inicial das dependências pelos respectivos gerenciadores de pacotes. O mesmo ambiente está descrito na Seção 4.2 para a execução do sistema.

4.4.1 Linguagens de desenvolvimento

O sistema foi desenvolvido em duas linguagens de programação, uma para cada camada da arquitetura descrita na Seção 4.2: Java no *back-end* e TypeScript no *front-end*.

O Java é uma linguagem de programação de propósito geral, orientada a objetos e independente de plataforma, já que o código é compilado para bytecode e executado pela Java Virtual Machine (JVM) (ORACLE, 2021). Foi usado, na versão 17 (LTS), para escrever todo o *back-end* do sistema: a API REST, os serviços de regra de negócio e o mapeamento objeto-relacional das entidades com o banco de dados.

O TypeScript é um superconjunto tipado do JavaScript, mantido pela Microsoft, que adiciona tipagem estática opcional à linguagem e é transpilado para JavaScript puro antes de ser executado no navegador (MICROSOFT, [20--]). Foi usado, na versão exigida pelo Angular 17, para escrever todo o *front-end* do sistema: os componentes de tela, os serviços que consomem a API REST e os modelos de dados compartilhados entre eles.

4.4.2 Frameworks

No trabalho foram utilizados dois *frameworks* que estruturam o desenvolvimento do sistema, um para cada camada da arquitetura descrita na Seção 4.2: Spring Boot no *back-end* e Angular no *front-end*.

O Spring Boot é um *framework* construído sobre o Spring Framework que facilita a criação de aplicações Java autônomas e prontas para produção, com configuração automática e um servidor web embarcado, dispensando boa parte da configuração manual exigida pelo Spring tradicional (SPRING, [20--]). Foi usado, na versão 3.2.5, com o servidor Tomcat embarcado, para expor a API REST do sistema e gerenciar, através do seu contêiner de inversão de controle, o ciclo de vida dos serviços, repositórios e controllers descritos nas Seções 3.4 e 4.3.

O Angular é um *framework* de *front-end* baseado em TypeScript, mantido pelo Google, voltado à construção de aplicações web de página única (SPA), com suporte nativo a roteamento, formulários e comunicação HTTP (GOOGLE, [20--]). Foi usado, na versão 17, para construir toda a interface do sistema, das telas de cadastro ao *pop-up* de validação de participantes, consumindo a API REST do *back-end* através do módulo `HttpClient`.

4.4.3 Bibliotecas

Além dos *frameworks* descritos na Seção 4.4.2, o sistema utiliza três bibliotecas para resolver necessidades específicas que não caberiam no escopo de um *framework* completo. As

bibliotecas utilizadas estão listadas abaixo:

O Apache POI é uma biblioteca Java para leitura e escrita de arquivos no formato do Microsoft Office, incluindo planilhas Excel (APACHE SOFTWARE FOUNDATION, [20--]). Foi usado, na versão 5.2.5, para ler as planilhas .xlsx exportadas do Teams com os votos e a lista de presença de cada reunião, convertendo cada linha da planilha nos objetos processados pelo `CruzamentoVotosService` (Seção 3.2).

O Lombok é uma biblioteca que gera, em tempo de compilação, código repetitivo de uma classe Java a partir de anotações, como getters, setters e construtores (PROJECT LOMBOK, [20--]). Foi usado em todas as entidades JPA do sistema, como `Reuniao`, `Pauta` e `Votante`, reduzindo o código que, de outra forma, precisaria ser escrito manualmente para cada atributo dessas classes.

O RxJS é uma biblioteca de programação reativa para JavaScript, baseada em fluxos de eventos assíncronos chamados Observables (REACTIVEX, [20--]). É usado pelo próprio Angular, na versão 17, para representar as respostas assíncronas das chamadas HTTP à API REST do *back-end*, permitindo que os componentes de tela se inscrevam nesses fluxos e reajam a cada nova resposta.

4.4.4 Ferramentas Auxiliares

Além dos *frameworks* e bibliotecas descritos nas Seções 4.4.2 e 4.4.3, três ferramentas auxiliares deram suporte ao processo de *build* e gerenciamento de dependências do sistema, uma para cada etapa do fluxo de desenvolvimento. As ferramentas utilizadas estão listadas abaixo:

O Maven é uma ferramenta de gerenciamento e automação de *builds* para projetos Java, responsável por baixar as dependências declaradas no arquivo `pom.xml` e compilar, testar e empacotar a aplicação (APACHE MAVEN, [20--]). Foi usado para gerenciar todas as dependências do *back-end*, como o Spring Boot, o H2, o Apache POI e o Lombok, e para gerar o artefato final da aplicação.

O npm é o gerenciador de pacotes padrão do Node.js, usado para instalar e versionar as dependências de um projeto JavaScript ou TypeScript a partir do arquivo `package.json` (NPM, [20--]). Foi usado para gerenciar as dependências do *front-end*, como o Angular e o RxJS, e para executar os scripts de *build* e desenvolvimento do projeto.

O Angular CLI é a ferramenta de linha de comando oficial do Angular, usada para criar, construir e servir uma aplicação Angular a partir de comandos como `ng serve` e `ng build` (ANGULAR, [20--]). Foi usado ao longo de todo o desenvolvimento do *front-end*, tanto para servir a aplicação localmente durante o desenvolvimento quanto para gerar os artefatos de *build* do projeto.

4.4.5 Sistemas e componentes externos utilizados

O sistema se integra a um único sistema externo: o Microsoft Teams, plataforma usada pela universidade para realizar as reuniões dos conselhos, com a votação de cada pauta feita através de uma enquete do Microsoft Forms embutida na própria reunião. Essa integração não ocorre por API, o sistema não faz nenhuma chamada à API do Teams ou do Microsoft Graph. Ela é inteiramente baseada em arquivo, a secretaria baixa manualmente os relatórios gerados pelo Teams e os envia para o sistema pelas telas de importação (RF007 e RF008).

São dois os arquivos consumidos, com formatos bem diferentes entre si. O primeiro é a exportação real de uma enquete do Microsoft Forms/Teams (.xlsx), com colunas como "ID", "Start time", "Completion time", "Email", "Name" e o texto da própria pergunta votada. O segundo é o relatório "Lista de participantes" (Meeting Attendance List) exportado direto do Teams, apesar da extensão .csv ou .xls, não é um arquivo Excel binário de verdade, e sim um texto delimitado por tabulação, normalmente em UTF-16 com BOM, com as colunas "Nome Completo", "Atividade" e "Carimbo de data/hora", sem e-mail, já que muitos tenants do Teams omitem esse dado por padrão de privacidade.

Como a extensão do arquivo não é confiável para identificar qual dos dois formatos foi enviado, o sistema inspeciona os primeiros bytes do próprio arquivo para reconhecer o relatório de presença do Teams, em vez de confiar apenas no nome ou na extensão informados no upload.

4.5 Instalação e Configurações

A instalação do sistema segue dois passos independentes, um para o *back-end* e outro para o *front-end*, já que ambos rodam como processos separados na mesma máquina (Seção 4.2). Os pré-requisitos são JDK 17 ou superior, Node.js 18 ou superior com npm, e acesso à internet na primeira execução, necessário para o Maven e o npm baixarem as dependências dos respectivos repositórios.

O *back-end* é iniciado a partir da pasta `backend` com o comando `mvn spring-boot:run`, que baixa as dependências declaradas no `pom.xml`, compila o projeto e sobe a API em `http://localhost:8080`. Na primeira execução, o Hibernate cria automaticamente o arquivo do banco H2 (`./data/sistema-votos.mv.db`) e as oito tabelas do modelo físico (Seção 3.3.2), sem exigir nenhum script de configuração manual do banco.

O *front-end* é iniciado a partir da pasta `frontend` com os comandos `npm install`, para baixar as dependências declaradas no `package.json`, seguido de `npm start`, que sobe a aplicação em

`http://localhost:4200`. O *back-end* precisa já estar em execução antes desse passo, já que o *front-end* se conecta a ele assim que carrega. Para gerar uma versão de produção, com arquivos estáticos e sem o servidor de desenvolvimento do Angular, usa-se `npm run build`.

As principais configurações do sistema ficam centralizadas no arquivo `application.yml` do *back-end*: a porta do servidor (8080), o domínio de e-mail institucional aceito na validação de votantes (`@ufu.br`), o tamanho máximo de arquivo aceito nas importações (10MB) e o caminho do arquivo do banco H2. Nenhuma dessas configurações precisa ser alterada para rodar o sistema localmente; elas existem para que a instituição possa ajustá-las caso o sistema seja adaptado para outro conselho ou domínio de e-mail.

4.5.1 Pré-requisitos do Sistema

Os pré-requisitos do sistema são: um único computador convencional, sem servidor dedicado e nem requisito específico de hardware, já que o *back-end*, o *front-end* e o banco de dados rodam todos localmente, no mesmo computador.

Do lado do software, é necessário ter instalados o JDK 17 ou superior, para compilar e executar o *back-end*, o Maven, para gerenciar suas dependências e realizar o *build*, e o Node.js 18 ou superior com npm, para instalar as dependências e servir o *front-end*. Para acessar a interface do sistema, também é necessário um navegador web atualizado, compatível com os recursos de JavaScript modernos (ES2022) usados no *build* da aplicação Angular.

Por fim, é necessário acesso à internet apenas na primeira execução de cada parte, para que o Maven e o npm baixem as dependências dos seus respectivos repositórios (Maven Central e npm registry). Depois disso, o sistema roda inteiramente em localhost, sem depender de rede alguma, conforme já descrito na Seção 4.2.

4.5.2 Processo de Instalação/Execução

O processo de instalação e execução do sistema segue a ordem descrita nesta seção, já que o *front-end* depende do *back-end* estar disponível para funcionar corretamente. Primeiro, inicia-se o *back-end*: a partir da pasta `backend`, executa-se `mvn spring-boot:run`, que baixa as dependências do projeto, apenas na primeira execução, compila o código e sobe a API REST em `http://localhost:8080`. Nesse mesmo processo, o Hibernate cria automaticamente o arquivo do banco H2 e as tabelas do sistema, caso ainda não existam (Seção 3.3.2).

Em seguida, com o *back-end* já em execução, inicia-se o *front-end* em um terminal separado: a partir da pasta `frontend`, executa-se `npm install`, apenas na primeira execução, para baixar as dependências do projeto, e depois `npm start`, que sobe a aplicação Angular em `http://localhost:4200`.

Por fim, acessa-se o sistema pelo navegador, no endereço `http://localhost:4200`, que já se conecta automaticamente à API do *back-end* (Seção 4.2). Nenhuma etapa manual de configuração de banco de dados ou de rede é necessária: a primeira execução já deixa o sistema pronto para uso. Para encerrar o sistema, basta interromper os dois processos. Como o banco de dados é gravado em arquivo (Seção 3.3.2), os dados cadastrados permanecem entre uma execução e outra: reiniciar o *back-end* não recria as tabelas do zero, apenas reconecta ao mesmo arquivo já existente.

4.5.3 Variáveis de Ambientes Necessárias

O sistema não depende de nenhuma variável de ambiente do sistema operacional. Como o *back-end* e o *front-end* rodam sempre na mesma máquina, sem distinção entre ambientes de desenvolvimento, homologação e produção (Seção 4.2), toda a configuração fica centralizada em arquivos do próprio projeto, em vez de variáveis externas.

No *back-end*, os parâmetros configuráveis, como a porta do servidor, o domínio de e-mail institucional e o caminho do banco H2, ficam definidos diretamente no arquivo `application.yml` (Seção 4.5), lidos pelo Spring através de anotações `@Value`, como em `@Value("${sistema.dominio-email-institucional}")`. No *front-end*, a URL base da API é uma constante fixa no arquivo `api-base.ts`, em vez de um arquivo de `environment` por ambiente, já que não há ambiente diferente para alternar.

4.5.4 Arquivos de Configuração

Como visto na Seção 4.5.3, o sistema não utiliza variáveis de ambiente do sistema operacional, concentrando toda a sua configuração em arquivos do próprio projeto. No *back-end*, esse arquivo é o `application.yml`; no *front-end*, são o `angular.json`, que configura as ferramentas de *build* e execução do Angular CLI, e o `tsconfig.json`, que configura o compilador TypeScript.

No `application.yml`, ficam definidos a porta do servidor (`8080`), o caminho do arquivo do banco H2 (`./data/sistema-votos`), o modo de geração automática das tabelas pelo Hibernate a partir das entidades (`ddl-auto: update`, Seção 3.3.2), o console web do H2 usado durante o

desenvolvimento para inspecionar o banco (`/h2-console`), o tamanho máximo de arquivo aceito nas importações de planilhas (`10MB`), o domínio de e-mail institucional aceito na validação de votantes (`@ufu.br`) e o nível de log da aplicação (`INFO`).

No *front-end*, o `angular.json` define, entre outras opções, a pasta de saída do *build* de produção (`dist/sistema-votos-frontend`), a porta usada pelo servidor de desenvolvimento (`4200`) e os limites de tamanho dos arquivos gerados na *build* de produção, usados pelo Angular CLI para avisar caso o pacote final cresça além do esperado.

Já o `tsconfig.json` configura o compilador TypeScript em modo estrito (`strict: true`), com verificações adicionais como `noImplicitReturns` e `noFallthroughCasesInSwitch`, além da opção `strictTemplates`, que estende essas mesmas verificações de tipo aos templates HTML dos componentes Angular. Diferentemente dos demais arquivos citados, essa configuração não afeta o comportamento do sistema em execução, apenas garante, em tempo de compilação, um padrão mais rígido de tipagem no código do *front-end*.

4.6 Utilização do Sistema

Esta seção detalha a utilização do sistema em execução, complementando a Seção 4.5, que trata da instalação e da configuração inicial. Os exemplos apresentados a seguir reaproveitam as capturas de tela reais já introduzidas na Seção 2.5, descritas sob a perspectiva do uso do sistema, e não da validação dos requisitos.

4.6.1 Processo de Execução do Sistema

Uma vez concluída a instalação (Seção 4.5.2), o acesso ao sistema é feito pelo navegador, no endereço `http://localhost:4200`. A aplicação exibe uma barra de topo fixa, com o nome do sistema e a identificação da secretaria, e uma barra lateral com as duas etapas do fluxo de uso: "1. Detalhes da Reunião" (cadastro de reunião, pautas e lista de conselheiros) e "2. Processar Votos e Presenças" (apuração de votação e lista de presença). Não há tela de login: o sistema é de uso interno da secretaria, em um único computador (Seção 4.2), e as duas etapas ficam acessíveis diretamente pela barra lateral, a qualquer momento.

O uso do sistema segue, em geral, a mesma ordem de dependência entre os dados já descrita no processo de desenvolvimento (Seção 4.1): primeiro o cadastro da reunião, da pauta e dos conselheiros aptos a votar, na etapa "Detalhes da Reunião"; em seguida, já na etapa "Processar Votos e Presenças", durante a reunião, a importação da presença; e, por fim, a importação dos votos de cada pauta, seguida da consulta à apuração. A Seção 4.6.2 detalha cada uma dessas etapas.

4.6.2 Exemplos de Utilização das Principais Funcionalidades do Sistema

4.6.2.1 Requisito Funcional 001 (Cadastro de reunião)

Para começar a usar o sistema, a secretaria cadastra uma reunião informando o conselho, a data, o horário e o link da reunião no Microsoft Teams. Salva a reunião, ela passa a constar na lista de reuniões cadastradas, com status "Agendada" (Figura 20), e passa a estar disponível para as demais etapas do fluxo.

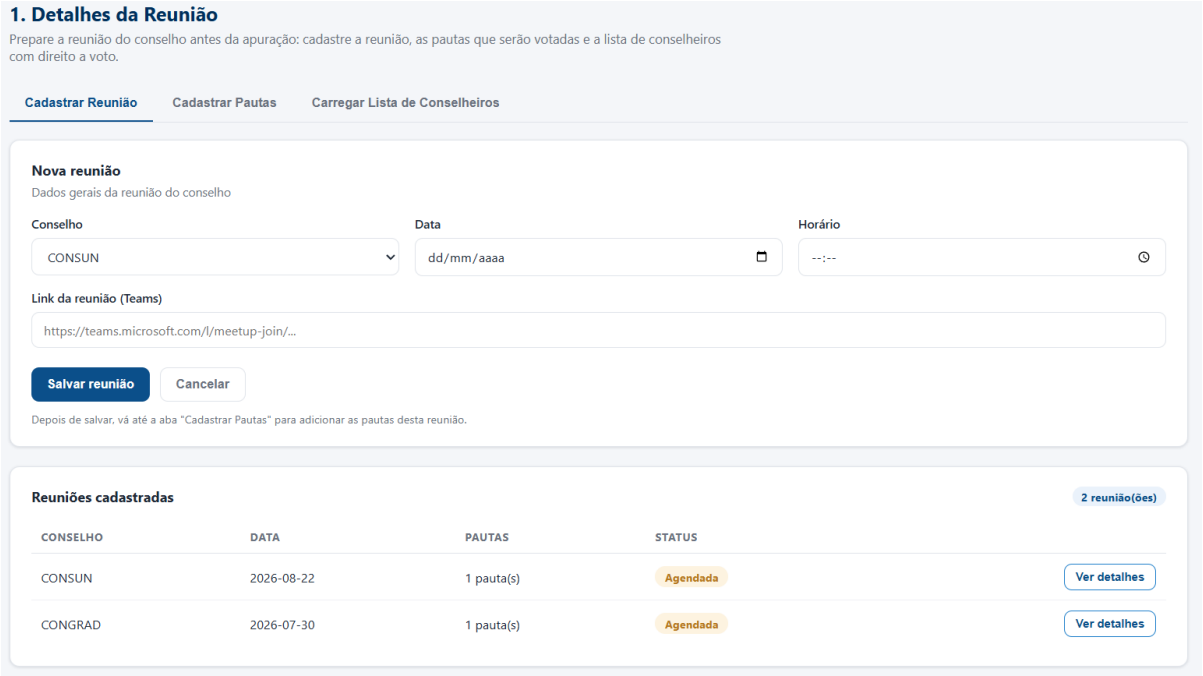


Figura 20 – Cadastro de reunião (RF001)
(Fonte: Autor)

4.6.2.2 Requisito Funcional 002 (Cadastro de pauta)

Com a reunião cadastrada, a secretaria adiciona as pautas que serão discutidas e votadas, informando título e, opcionalmente, uma descrição. As pautas cadastradas ficam listadas na mesma tela (Figura 21), disponíveis para a importação de presença e de votos nas etapas seguintes.

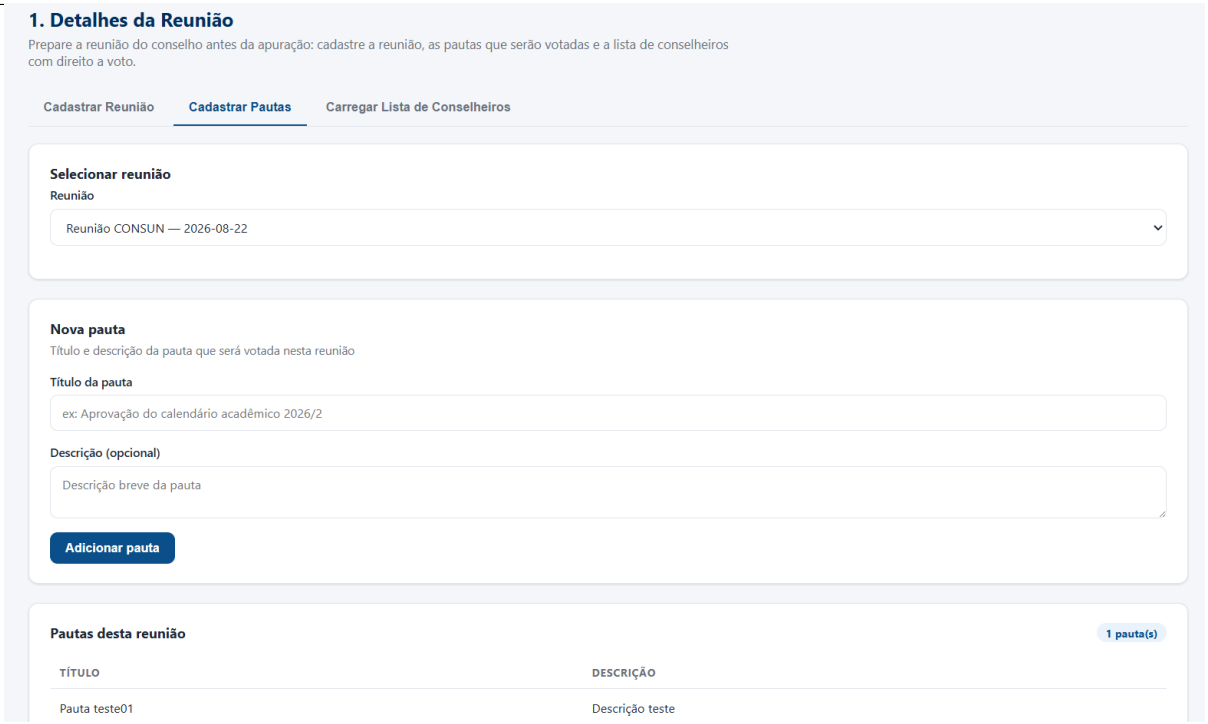


Figura 21 – Cadastro de pauta (RF002)
(Fonte: Autor)

4.6.2.3 Requisito Funcional 003/004/005/006 (Gestão de conselheiros)

Ainda na etapa de preparação, a secretaria gerencia os conselheiros aptos a votar na reunião: pode cadastrá-los manualmente, um a um, pelo botão "Adicionar manualmente", ou importar a lista completa de uma só vez a partir de uma planilha (.xlsx ou .csv) com nome e e-mail institucional. Cada conselheiro cadastrado pode ser removido individualmente, e a lista de aptos a votar pode ser exportada a qualquer momento pelo botão "Exportar lista" (Figura 22).

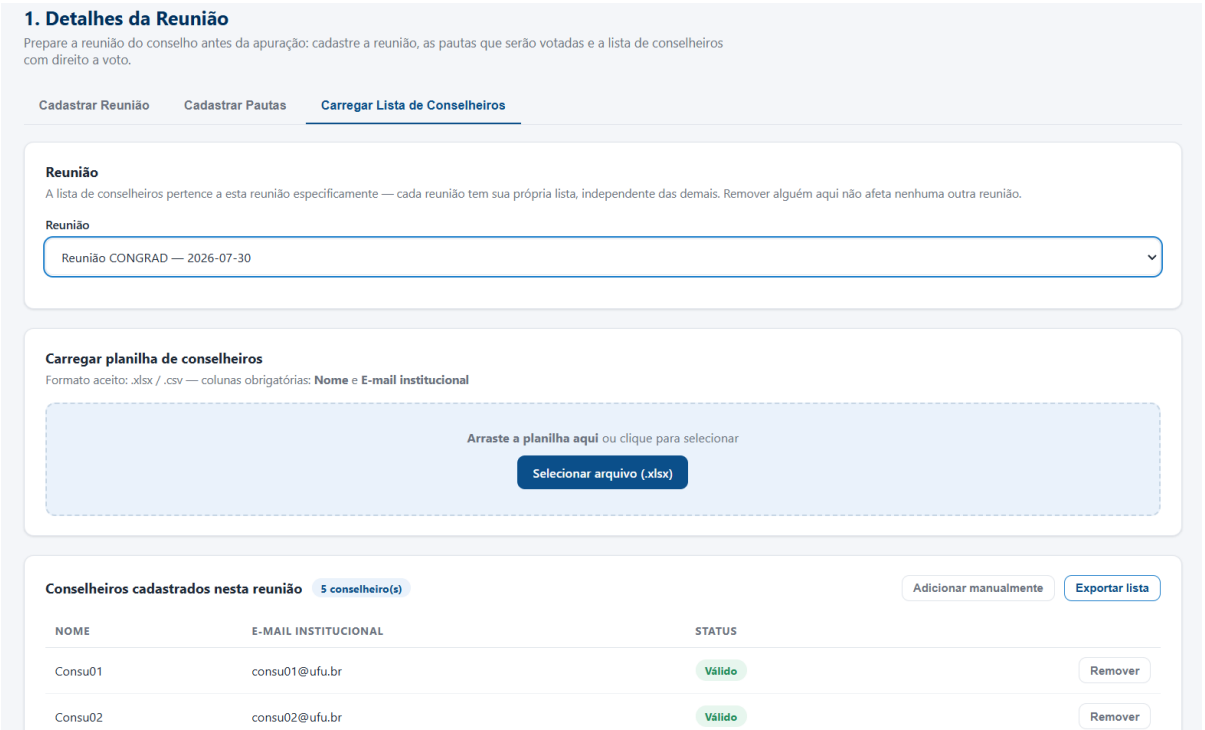


Figura 22 – Gestão de conselheiros (RF003, RF004, RF005 e RF006) (Fonte: Autor)

4.6.2.4 Requisito Funcional 007/009 (Importação e consulta de presença)

Já durante a reunião, a secretaria importa a planilha de presença exportada do Microsoft Teams. A partir dela, o sistema passa a exibir, em destaque, quantos conselheiros com direito a voto estão online no momento e quantos participantes ativos há ao todo, além de listar cada participante com sua origem, status online e se está apto a votar (Figura 23).

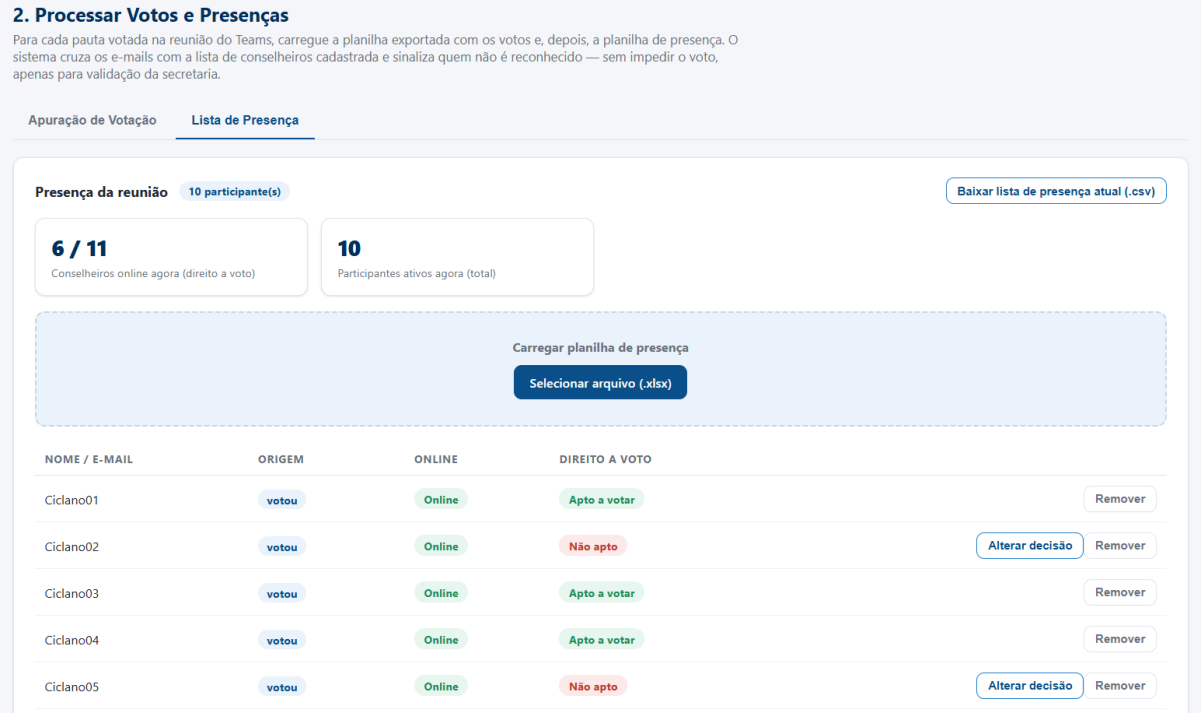


Figura 23 – Importação e consulta de presença (RF007 e RF009)
(Fonte: Autor)

4.6.2.5 Requisito Funcional 008/011 (Importação de votos e validação de participantes)

Ao apurar uma pauta, a secretaria importa a planilha de votos correspondente. O sistema cruza automaticamente os e-mails votantes com a lista de conselheiros cadastrada e, ao encontrar um e-mail que não consta nessa lista, exibe um alerta para que a secretaria valide ou negue manualmente aquela participação, sem bloquear o cruzamento. A decisão pode ser aplicada apenas à pauta atual ou a todas as pautas restantes da reunião, evitando que o mesmo alerta se repita para o mesmo e-mail a cada nova pauta votada (Figura 24).

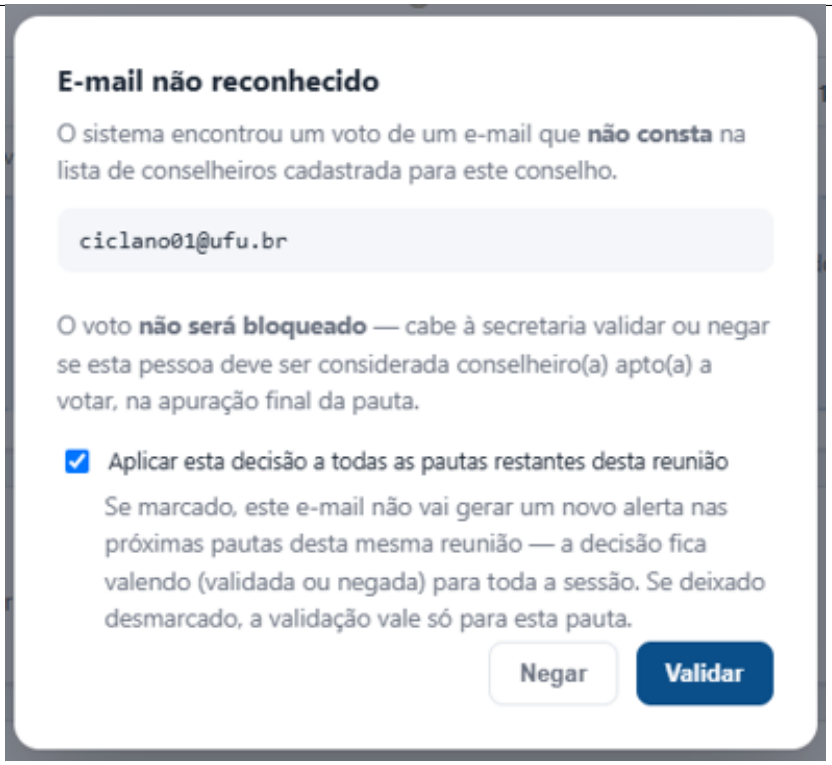


Figura 24 – Importação de votos e validação de participante (RF008 e RF011)
(Fonte: Autor)

4.6.2.6 Requisito Funcional 010/012 (Consulta de apuração e edição de decisão manual)

Concluído o cruzamento de uma pauta, o sistema apresenta a apuração: a quantidade de votos por opção, um gráfico comparativo, um resumo com o total de votos válidos, a proporção de conselheiros que votaram, os votos negados e os participantes online que não votaram, além da tabela detalhada com o status de cada voto (Figura 25). Caso necessário, a secretaria pode reverter uma decisão manual já tomada a qualquer momento, tanto na lista de presença quanto nessa própria tela de apuração, pelo botão "Alterar decisão".

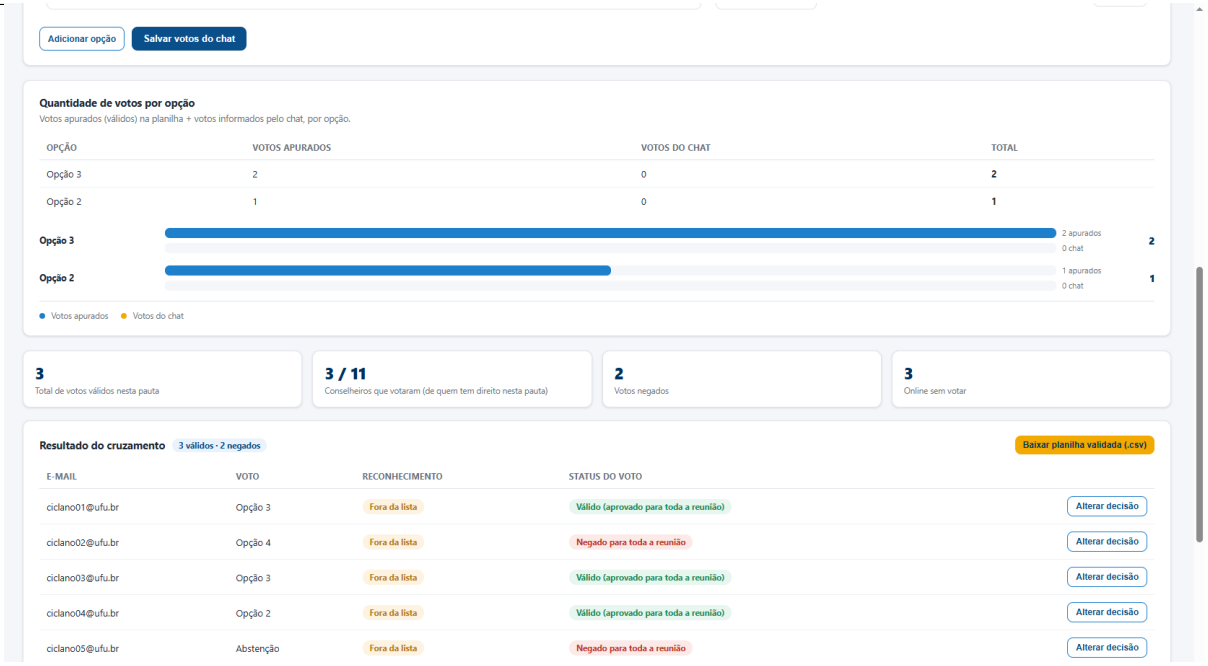


Figura 25 – Consulta de apuração e edição de decisão manual (RF010 e RF012)
(Fonte: Autor)

4.7 Testes do Sistema

Esta seção descreve os testes realizados sobre o sistema, com o objetivo de identificar defeitos, validar suas funcionalidades e verificar se os requisitos funcionais (Seção 2.3) foram implementados corretamente. Diferentemente de um processo automatizado, os testes aqui descritos são majoritariamente manuais, executados diretamente na interface do sistema com dados reais, na mesma linha do processo de desenvolvimento incremental já descrito na Seção 4.1, em que cada funcionalidade era validada antes de se avançar para a seguinte.

4.7.1 Plano de Testes

O plano de testes a seguir cobre os doze requisitos funcionais do sistema (RF001 a RF012, Seção 2.3) e um conjunto de casos encontrados durante o desenvolvimento, relacionados sobretudo à identificação de participantes por nome ou por e-mail e à consistência dos números exibidos entre as diferentes telas. A descrição de cada caso e o resultado esperado estão no Quadro 3.

Quadro 3 – Plano de testes do sistema

Nº	Descrição	Resultados Esperados	Requisito(s)
TC01	Cadastro de reunião, pauta e conselheiros sem duplicação: cadastrar uma reunião, duas pautas e conselheiros, manualmente e via importação de planilha.	Todos os cadastros são salvos e listados corretamente, sem registros duplicados.	RF001, RF002, RF003, RF004

Nº	Descrição	Resultados Esperados	Requisito(s)
TC02	Isolamento da lista de conselheiros por reunião: cadastrar conselheiros em uma reunião, criar uma segunda reunião e verificar a lista de cada uma.	A lista de conselheiros de cada reunião é independente; cadastrar, remover ou consultar em uma reunião não afeta as demais.	RF003, RF005
TC03	Trava de e-mail único por reunião: tentar cadastrar dois conselheiros com o mesmo e-mail na mesma reunião, e repetir o mesmo e-mail em reuniões diferentes.	O sistema recusa o cadastro duplicado dentro da mesma reunião, mas permite o mesmo e-mail em reuniões diferentes.	RF003
TC04	Consulta e exportação da lista de aptos: selecionar uma reunião, consultar a lista de conselheiros aptos a votar e exportá-la em .csv.	A tabela exibida e o arquivo exportado refletem exatamente os conselheiros cadastrados na reunião selecionada.	RF006
TC05	Ordem de validação na importação de presença: importar uma planilha de presença contendo ao menos uma pessoa não cadastrada como conselheiro.	O alerta de validação do participante não reconhecido é exibido antes do resumo de quórum; os números do quórum coincidem com a tabela de presença exibida em seguida.	RF007, RF009, RF011
TC06	Importação aditiva de presença: importar uma segunda planilha de presença, com um subconjunto diferente de participantes.	Nenhum participante da importação anterior é removido; quem consta nas duas importações não duplica; quem não consta na nova planilha mantém o último status conhecido.	RF007, RF009
TC07	Consolidação de participante identificado por nome e por e-mail: importar a presença de uma pessoa apenas pelo nome e, em seguida, um voto dessa mesma pessoa identificado por e-mail (e o caminho inverso).	A pessoa não é duplicada: a mesma linha passa a ser identificada pelo e-mail, preservando qualquer validação manual já realizada.	RF007, RF008, RF011
TC08	Voto manual não exige nova validação: cadastrar manualmente o voto de uma pessoa não reconhecida e de um conselheiro já cadastrado.	Em ambos os casos, o voto é salvo como válido sem abrir novo alerta de validação, e a pessoa passa a constar como apta e online na lista de presença.	RF008, RF011
TC09	Atualização do status online ao registrar voto: registrar o voto de um participante cujo último status de presença conhecido era offline.	O participante passa a ser exibido como online na lista de presença, mesmo sem uma nova importação de planilha de presença.	RF008, RF009
TC10	Lista de votação da pauta atual: consultar, com uma pauta selecionada, os conselheiros online com direito a voto que ainda não votaram, incluindo a ordenação pelas colunas disponíveis.	Somente conselheiros online no momento aparecem na lista; quem ainda não votou é destacado; a ordenação por nome, presença e voto funciona nos dois sentidos; o contador exibido bate com as linhas visíveis.	RF009, RF010
TC11	Consistência dos números entre cartões e tabelas: comparar os cartões de resumo da apuração e da presença com as respectivas tabelas detalhadas.	Os valores exibidos nos cartões (votos válidos, conselheiros que votaram, votos negados, presença online) coincidem exatamente com a contagem das linhas das tabelas correspondentes.	RF010
TC12	Exportação da presença sem participantes removidos: remover um participante da lista de presença e, em seguida, exportar a lista em .csv.	O arquivo exportado não inclui o participante removido.	RF006, RF009
TC13	Propagação de alteração de decisão manual: alterar uma decisão manual já registrada (de validado para negado, ou o inverso) pela lista de presença.	A alteração é refletida tanto na apuração da pauta atual quanto no histórico de decisões da reunião.	RF012
TC14	Não fusão automática de nomes semelhantes: registrar dois participantes com nomes parecidos, porém distintos.	O sistema mantém os dois registros como participantes diferentes, sem fundi-los automaticamente; a fusão, se necessária, ocorre apenas por validação manual da secretaria.	RF011
TC15	Recarregamento ao trocar de pauta ou reunião: alternar repetidamente entre pautas e reuniões diferentes na tela de apuração.	Os cartões, tabelas e listas exibidos são sempre recarregados para os dados da pauta e reunião atualmente selecionadas, sem reter dados da seleção anterior.	RF010

(Fonte: Autor)

4.7.2 Execução do Plano de Testes

Todos os quinze casos descritos no Quadro 3 foram executados manualmente, com o *back-end* e o *front-end* em execução real na máquina do autor (Seção 4.5.2), e não apenas com dados simulados. Os quinze casos passaram, incluindo os casos de borda relacionados à identificação de participantes por nome ou e-mail (TC07 e TC14) e à consistência dos números exibidos entre cartões de resumo e tabelas (TC11), que motivaram correções durante o próprio desenvolvimento do sistema, antes desta execução final do plano.

4.8 Repositório de Código

4.8.1 Link para o Repositório

O código-fonte completo do sistema está hospedado no GitHub, no endereço `https://github.com/Henrique-Braga/sistema-verificacao-votos-TCC-`.

4.8.2 Estrutura de Diretórios

O repositório está dividido em duas pastas principais, *back-end* e *front-end*, cada uma com seu próprio arquivo de dependências (`pom.xml` e `package.json`, respectivamente), além dos arquivos de apoio na raiz do projeto (Quadro 4). No *back-end*, o pacote br.ufu.conselhos.votos segue a separação por camada já descrita na Seção 4.3 (DTO, Repository, Service, além dos pacotes de configuração, controladores REST, tratamento de exceções e entidades JPA). No *front-end*, cada pasta em features corresponde a uma das duas etapas do fluxo de uso descritas na Seção 4.6.1, com um serviço HTTP em core/services por controlador do *back-end*.

Quadro 4 – Estrutura de diretórios do repositório (simplificada)

```
sistema-verificacao-votos-TCC-/
|-- LEIA-ME.md              guia de instalacao para a secretaria
|-- PLANO_DE_TESTES.md      roteiro de testes manuais (Secao 4.7.1)
|-- backend/
|   |-- pom.xml
|   `-- src/main/java/br/ufu/conselhos/votos/
|       |-- config/         configuracao (CORS)
|       |-- controller/     endpoints REST
|       |-- core/           logica de negocio isolada (Secao 4.3)
|       |-- dto/            objetos de transferencia de dados
|       |-- exception/      tratamento centralizado de erros
|       |-- model/          entidades JPA
|       |-- repository/     interfaces Spring Data JPA
|       `-- service/        regras de negocio
`-- frontend/
    |-- angular.json
    |-- package.json
    `-- src/app/
        |-- core/           modelos, servicos HTTP e configuracao da API
        |-- features/       telas (detalhes-reuniao, processar-votos)
        `-- shared/         componentes reutilizaveis (toast, modal de validacao)
```

(Fonte: Autor)

4.8.3 Guia de Contribuições (se aplicável)

Não se aplica. Por se tratar de um Trabalho de Conclusão de Curso desenvolvido individualmente, sem outros colaboradores previstos, o repositório não possui um guia formal de contribuições: toda a autoria e o histórico de versionamento pertencem ao autor deste trabalho.

4.9 Resultados Alcançados com o Sistema

Os resultados obtidos foram positivos: os doze requisitos funcionais definidos na Seção 2.3 (RF001 a RF012) foram implementados e validados pelos quinze casos de teste do Quadro 3 (Seção 4.7.1), cobrindo desde o cadastro de reuniões, pautas e conselheiros até a apuração automática de votos e a validação manual de participantes não reconhecidos. Na prática, o sistema atende aos objetivos definidos no Capítulo 1: cada conselho e cada reunião mantém sua própria lista de conselheiros, sem misturar dados entre si (RF003 a RF006); o quórum e a lista de participantes aptos são recalculados em tempo real, conforme a secretaria valida os casos pendentes (RF009, RF011); e um mesmo participante é reconhecido mesmo quando identificado de formas diferentes entre a planilha de presença e a de votação, sem duplicar seu registro (RF007, RF008, RF011).

Uma limitação, já apontada na Seção 1.1, permanece: o sistema não se integra diretamente com o Microsoft Teams, dependendo da exportação e da importação manual das planilhas de presença e votação a cada reunião. Ainda assim, o cruzamento em si, que antes era feito manualmente pela secretaria, conferindo linha a linha quem estava presente e como cada um votou, passa a ser automático

a partir do momento em que essas planilhas são carregadas no sistema.

4.10 Considerações Sobre o Capítulo

Neste capítulo, foi apresentado o desenvolvimento do sistema, desde o processo e a arquitetura adotados até a instalação, a utilização e os testes realizados, sempre a partir do código-fonte real da implementação. O processo de desenvolvimento e a arquitetura explicaram as decisões técnicas tomadas, como a divisão entre *back-end* e *front-end* numa única máquina, e os padrões de projeto e as tecnologias empregados em cada camada do sistema. Em seguida, foram detalhadas a instalação e a configuração do sistema, e a sua utilização já em execução, com exemplos reais de cada uma das principais funcionalidades. Por fim, o plano de testes validou os doze requisitos funcionais do sistema, o repositório de código disponibilizou o código-fonte completo ao leitor, e os resultados alcançados confirmaram que os objetivos definidos no Capítulo 1 foram atendidos.

Portanto, este capítulo mostrou, na prática, como o sistema descrito nos capítulos anteriores foi de fato construído, instalado e validado, servindo de base para as conclusões apresentadas no Capítulo 5.

CAPÍTULO **5**

Conclusões

O objetivo geral deste trabalho era tornar mais eficiente e menos manual o processo de apuração de presença e votação das reuniões de conselhos superiores da UFU, automatizando o cruzamento entre os dados exportados do Microsoft Teams e a lista de conselheiros de cada reunião (Seção 1.3). O desenvolvimento do sistema, descrito no Capítulo 4, e a execução do plano de testes (Seção 4.7) confirmam esse objetivo: o cruzamento que antes era feito manualmente, linha a linha, passou a ser automático a partir da importação das planilhas de presença e votação.

Quanto aos objetivos específicos, a necessidade de gerenciar múltiplos conselhos, cada um com sua própria lista de conselheiros e reuniões, sem misturar os dados entre si, foi atendida pela separação da lista de conselheiros por reunião (RF003 a RF006), validada pelo caso de teste TC02 (Seção 4.7.1). O acompanhamento do quórum em tempo real, à medida que a reunião acontece, foi atendido pela atualização automática da lista de presença e do status online dos conselheiros a cada nova planilha importada ou voto registrado (RF008, RF009), validado pelos casos de teste TC09 e TC10.

Por fim, o reconhecimento de um mesmo participante identificado de formas diferentes entre os dados de presença e de votação, recorrendo à secretaria apenas quando essa identificação não é clara, foi atendido pela consolidação de registros por nome ou e-mail (RF007, RF008, RF011), validada pelos casos de teste TC07 e TC14, que também confirmaram que o sistema não funde automaticamente participantes apenas parecidos, preservando a decisão da secretaria nesses casos.

5.1 Principais Contribuições

As principais contribuições deste trabalho se confirmam nas mesmas duas frentes anunciadas na Seção 1.4. Na frente prática, a Secretaria de Conselhos da UFU passa a contar com um sistema

funcional, instalado e testado (Capítulo 4), capaz de resolver o problema identificado na Seção 1.2: o cruzamento manual entre presença, votação e a lista de conselheiros aptos de cada reunião, hoje substituído por um processo automático, que só recorre à secretaria nos casos em que um participante não é reconhecido automaticamente.

Na frente de aprendizado, a forma como o sistema resolve a identificação de participantes por nome ou e-mail, e a decisão de vincular a lista de conselheiros a cada reunião em vez de mantê-la única, se mostraram soluções replicáveis para problemas semelhantes em outros contextos, conforme já discutido na Seção 1.4.

O comparativo com soluções já existentes, feito na Seção 1.1, também se confirma ao final do desenvolvimento: nem o OpenSlides, nem os recursos nativos do Microsoft Teams, nem sistemas de votação institucionais como o da UFES oferecem, em conjunto, o cruzamento automático de presença, votação e quórum por conselho e por reunião que este sistema entrega.

5.2 Trabalhos Futuros

Apesar de atender aos objetivos propostos, o sistema apresenta pontos que podem ser melhorados em trabalhos futuros. O principal deles é a ausência de integração direta com o Microsoft Teams (Seção 1.1): hoje, a secretaria precisa exportar manualmente as planilhas de presença e votação e importá-las no sistema; uma integração via API do Teams eliminaria essa etapa manual, automatizando também a coleta dos dados, e não apenas o seu cruzamento. Outro ponto é a cobertura de testes automatizados: apenas a lógica de cálculo de status de voto, concentrada na classe `CalculadoraStatusVoto` (Seção 4.3), possui uma suíte de testes em JUnit (Seção 4.1); o restante do sistema foi validado manualmente (Seção 4.7). Ampliar essa cobertura, com testes automatizados de integração para os *endpoints* REST e, no *front-end*, para os componentes Angular, tornaria o sistema mais seguro para futuras alterações. O sistema também não possui controle de acesso: qualquer pessoa com acesso ao computador da secretaria pode operá-lo livremente, sem distinção entre usuários (Seção 4.6.1). Um trabalho futuro poderia introduzir autenticação e um histórico de auditoria, identificando qual membro da secretaria realizou cada validação manual.

Do ponto de vista de infraestrutura, o uso do H2 em arquivo e a ausência de scripts de migração versionados (Seção 3.3.2) são adequados ao escopo de uma única máquina, mas limitariam uma eventual expansão do sistema para atender outras secretarias da universidade simultaneamente; migrar para um banco de dados cliente-servidor, como o PostgreSQL, com *migrations* versionadas, seria um passo necessário nesse cenário.

Por fim, como já apontado na Seção 1.4, a forma como este sistema resolve a identificação de participantes por nome ou e-mail, e o vínculo entre a lista de conselheiros e a reunião, não são específicos deste problema. Um projeto futuro poderia generalizar essa solução para outras plataformas de videoconferência, além do Microsoft Teams, ou para outros contextos de apuração de presença e votação fora do âmbito de conselhos universitários.

Referências

ANGULAR. **Angular CLI**. [S.l.: s.n.], [20--]. Disponível em: https://angular.dev/tools/cli. Acesso em: 31 jul. 2026.

APACHE MAVEN. **Welcome to Apache Maven**. [S.l.: s.n.], [20--]. Disponível em: https://maven.apache.org/. Acesso em: 31 jul. 2026.

APACHE SOFTWARE FOUNDATION. **Apache POI - the Java API for Microsoft Documents**. [S.l.: s.n.], [20--]. Disponível em: https://poi.apache.org/. Acesso em: 31 jul. 2026.

GOOGLE. **Angular**. [S.l.: s.n.], [20--]. Disponível em: https://angular.dev/. Acesso em: 31 jul. 2026.

MICROSOFT. **Manage the attendance and engagement report for meetings and events in Microsoft Teams**. [S.l.], 2026. Disponível em: https://learn.microsoft.com/en-us/microsoftteams/teams-analytics-and-reports/meeting-attendance-report. Acesso em: 31 jul. 2026.

MICROSOFT. **TypeScript Documentation**. [S.l.: s.n.], [20--]. Disponível em: https://www.typescriptlang.org/docs/. Acesso em: 31 jul. 2026.

NPM. **npm Docs**. [S.l.: s.n.], [20--]. Disponível em: https://docs.npmjs.com/. Acesso em: 31 jul. 2026.

OPENSLIDES. **OpenSlides: the digital motion and assembly system**. [S.l.: s.n.], 2026. Disponível em: https://openslides.com/. Acesso em: 31 jul. 2026.

ORACLE. **The Java® Language Specification: Java SE 17 Edition**. [S.l.], 2021. Disponível em: https://docs.oracle.com/javase/specs/jls/se17/html/index.html. Acesso em: 31 jul. 2026.

PROJECT LOMBOK. **Project Lombok**. [S.l.: s.n.], [20--]. Disponível em: https://projectlombok.org/. Acesso em: 31 jul. 2026.

REACTIVEX. **RxJS**. [S.l.: s.n.], [20--]. Disponível em: https://rxjs.dev/. Acesso em: 31 jul. 2026.

SPRING. **Spring Boot Reference Documentation**. [S.l.: s.n.], [20--]. Disponível em: https://docs.spring.io/spring-boot/docs/3.2.5/reference/htmlsingle/. Acesso em: 31 jul. 2026.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO. **Votação On-line**. Secretaria dos Órgãos Colegiados Superiores (SOCS). [S.l.], 2026. Disponível em: https://daocs.ufes.br/votacao-line. Acesso em: 31 jul. 2026.