LUIZ ROBERTO AKIO HIGUTI
MARCEL MAKOTO KONDO

USPOLIS: SISTEMA OPEN SOURCE PARA ALOCAÇÃO E VISUALIZAÇÃO DE SALAS DE AULA

São Paulo
2022

LUIZ ROBERTO AKIO HIGUTI
MARCEL MAKOTO KONDO

USPOLIS: SISTEMA OPEN SOURCE PARA ALOCAÇÃO E VISUALIZAÇÃO DE SALAS DE AULA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro da Computação.

São Paulo
2022

LUIZ ROBERTO AKIO HIGUTI
MARCEL MAKOTO KONDO

USPOLIS: SISTEMA OPEN SOURCE PARA ALOCAÇÃO E VISUALIZAÇÃO DE SALAS DE AULA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro da Computação.

Orientador:

Prof. Dr. Fábio Levy Siqueira

Co-orientador:

Renan de Luca Avila

São Paulo
2022

AGRADECIMENTOS

Ao nosso orientador, Prof. Dr. Fábio Levy Siqueira, que nos acompanhou durante todo o projeto, nos ensinou muito, contribuindo para nosso crescimento científico e intelectual.

Ao Renan de Luca Avila, coorientador deste projeto, pela atenção e apoio durante o processo de definição e orientação.

À Maria Amélia Correia Silva e ao Prof. Dr. Gustavo Pamplona Rehder, por disponibilizarem tempo e vontade para nos ajudar a compreender o problema abordado neste projeto.

A nossas famílias, pelo apoio e suporte em todas as questões externas à faculdade para que pudéssemos focar nos estudos.

RESUMO

As instituições de ensino presencial precisam alocar as turmas nas salas de aula disponíveis periodicamente. Em alguns casos, tal processo é feito de forma manual, obtendo alocações não otimizadas, demandando um grande esforço das pessoas responsáveis e podendo ainda gerar problemas durante o oferecimento das disciplinas como conflitos de horário e utilização não otimizada de salas.

Este trabalho propõe o desenvolvimento de um sistema de gestão e alocação de salas para o prédio do Biênio da Escola Politécnica da USP (EPUSP), aplicando metodologias ágeis e técnicas de engenharia de software, que utiliza de uma abordagem matemática de Programação Linear Inteira para resolução automatizada do Problema de Alocação de Aulas às Salas (PAAS).

Neste trabalho são descritas as metodologias utilizadas na busca da solução, e a especificação obtida. Ademais são detalhadas a arquitetura e as funcionalidades implementadas, bem como as validações de usuário realizadas, sendo uma intermediária focada na interface do sistema, e uma final, utilizando dados reais do ano seguinte (2023). O sistema foi considerado validado perante os usuários alvos em termos de usabilidade e funcionalidade, e confirmou-se a intenção de uso do mesmo para a gestão das alocações do prédio do Biênio a partir do primeiro semestre de 2023.

Palavras-Chave – Problema de Alocação de Aulas às Salas, PAAS, Programação Inteira, Programação Linear Inteira, EPUSP, Requisitos, Scrum, Histórias de Usuário.

ABSTRACT

Institutions need to periodically assign classes to the available classrooms, and in some cases this process is done manually, resulting in non-optimized allocations. Besides demanding a great effort from the administrative staff, this can also generate problems throughout the semester, such as schedule conflicts and suboptimal room usage.

This work proposes the development of a management and room allocation system for the *Biênio* building of the *Escola Politécnica da USP* (EPUSP), through the use of agile methodologies and software engineering techniques, with a mathematical approach of Integer Linear Programming to automatically solve the Classroom Assignment Problem (CAP).

This work describes the methodologies used in the search for the solution, and the specifications obtained. Furthermore, the architecture and the implemented functionalities are detailed, as well as the user validations performed, one focused on the system interface, and a final one, using real data from the following year (2023). The devellopedsystem was considered validated, both in terms of usability and functionalities, and got a confirmation of intent of usage in deployment for the assignment of classrooms in the Biênio building for the first semester of 2023.

Keywords – Classroom Assignment Problem, CAP, Integer Programming, Integer Linear Programming, EPUSP, Requirements, Scrum, User Stories.

SUMÁRIO

1 INTRODUÇÃO

O Problema de Alocação de Salas (PAS), presente em toda organização de ensino presencial, descreve o problema da alocação de turmas em salas de aula dados seus respectivos horários, atendendo a restrições predefinidas, como capacidade e exigências dos professores, por exemplo (SALES, 2015). Tal processo normalmente é feito de forma manual, obtendo alocações não otimizadas, além de demandar um grande esforço das pessoas responsáveis e podendo ainda gerar problemas durante o oferecimento das disciplinas como conflitos de horário e desperdício no uso de salas.

Na Escola Politécnica da USP (EPUSP) a alocação das aulas do Biênio às suas respectivas salas é feita de forma manual, através de planilhas estáticas ou ainda de agendas impressas. Ademais, a falta de um sistema centralizado dificulta a troca de informação com os alunos, devido ao difícil acesso à informação e mudanças repentinas dos locais de aula.

Uma solução que buscou resolver esse problema foi o USPolis, sistema desenvolvido voluntariamente pelo ex-aluno Renan de Luca Avila, e que era utilizado no prédio do Biênio até o início da pandemia. Todavia, o sistema foi desenvolvido em um contexto onde a sua manutenibilidade e escalabilidade não eram prioridades, sendo baseado em metodologias e tecnologias desatualizadas, acarretando assim em sua descontinuação após o término da graduação do aluno.

1.1 Objetivo

O objetivo deste projeto de formatura é desenvolver um sistema centralizado e *open source* de alocação e visualização de salas de aula para a comunidade da EPUSP, baseando-se no projeto anterior e aprimorando-o.

O projeto consiste na concepção e implementação de um sistema de alocação de salas de aula, voltado para as secretarias dos prédios acadêmicos da EPUSP. A aplicação será

responsável por resolver o problema de distribuição das disciplinas nas salas de aula, considerando seus respectivos horários e demais restrições de cada turma, através de um sistema centralizado e de fácil utilização.

O desenvolvimento do projeto será feito seguindo o *Scrum* e usando histórias de usuário, elicitando e refinando continuamente os requisitos de cada tipo de usuário do sistema.

1.2 Justificativa

A alocação de salas sendo um problema inerente a qualquer instituição de ensino, é de interesse de cada instituição possuir de ferramentas e sistemas que ajudem na execução dessa tarefa, tanto do ponto de vista de tempo quanto de complexidade. Contudo, no momento da realização deste trabalho, o prédio do Biênio (ciclo básico da engenharia) da EPUSP, local foco deste trabalho, se utiliza de agendas impressas e escrita em caneta para realizar tal processo, o que torna a tarefa custosa em tempo e esforço, além de dificultar o controle e disseminação de tais informações. O problema descrito está também presente nos demais prédios da EPUSP, porém em graus e contextos variados.

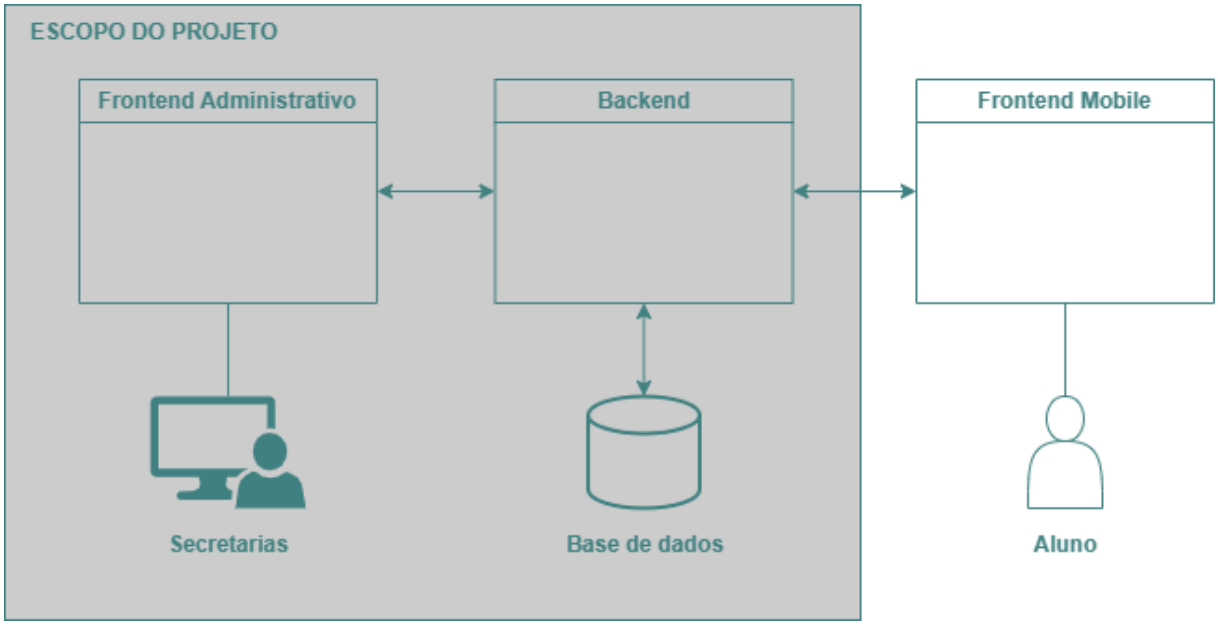
Existe portanto uma oportunidade na criação de uma aplicação capaz de auxiliar no processo de alocação de turmas para toda a comunidade da EPUSP, trazendo uma maior eficiência, minimizando questões como conflitos de horários e providenciando um sistema centralizador das informações, que possa ser utilizado inicialmente no prédio do Biênio e implementado ao longo do tempo no restante dos prédios.

1.3 Escopo

O sistema USPolis Original consistia de dois módulos principais: um sistema administrativo de alocação de salas de aula, voltado para as secretarias dos prédios acadêmicos, e uma plataforma web-mobile para visualização dos locais de aula, voltada para os alunos da graduação e pós-graduação.

O escopo de atuação do projeto será restrito, conforme apresentado na Figura 1, em um primeiro momento, ao prédio do Biênio da EPUSP, e o desenvolvimento será focado na concepção apenas do módulo de alocação de salas. Para tanto foram feitas entrevistas e contatos com pessoas envolvidas com o projeto.

Figura 1: Escopo do projeto



1.4 Organização do Trabalho

Este trabalho é organizado da seguinte forma: no Capítulo 2 são apresentados conceitos teóricos utilizados para desenvolvimento desse projeto; no Capítulo 3 são discutidos produtos semelhantes, indicando os principais diferenciais deste projeto; no Capítulo 4 é descrita a metodologia de trabalho adotada; no Capítulo 5 é apresentada uma especificação dos requisitos do sistema; no Capítulo 6 é definida a arquitetura do sistema desenvolvido; no Capítulo 7 são apresentadas as funcionalidades do sistema desenvolvido; e no Capítulo 8 são discutidos resultados obtidos em relação à usabilidade e performance do sistema desenvolvido.

2 ASPECTOS CONCEITUAIS

Este capítulo apresenta os principais conceitos teóricos empregados no trabalho. Do ponto de vista de Engenharia de Software, são abordados os conceitos de Scrum e Histórias de Usuário, os quais foram utilizados como base para a gestão e desenvolvimento do projeto. Do ponto de vista do problema de alocação de salas de aula, que compõe uma das funcionalidades principais do sistema desenvolvido, são abordados a formulação teórica do problema, conhecida como Problema de Alocação de Aulas às Salas (PAAS), e os principais conceitos de Programação Inteira, abordagem de otimização utilizada na sua solução.

2.1 Scrum

O Scrum é uma metodologia ágil para o desenvolvimento de produtos inovadores (RUBIN, 2012). Ele consiste em um conjunto de práticas que seguem princípios ágeis para organização e gerenciamento de projetos. Além disso, o Scrum pode ser adaptado, sem ignorar seus valores, para diferentes contextos.

Os times no Scrum são formados pelo *Product Owner* (PO), *Scrum Master* e time de desenvolvimento. O PO é o principal responsável pelo produto, decide as *features* que devem ser implementadas para garantir que seja desenvolvido o produto de maior valor possível (RUBIN, 2012). O *Scrum Master* atua como facilitador para adoção das práticas Scrum e é responsável pelo gerenciamento de possíveis mudanças (RUBIN, 2012). Por fim, o time de desenvolvimento, composto por 5 a 9 pessoas de áreas de atuações diversas, fica responsável por se auto-organizar e determinar a melhor forma de atingir os objetivos definidos pelo PO.

No Scrum, as atividades são realizadas em períodos cíclicos com duração fixa de até um mês (RUBIN, 2012), chamados *sprints*. No início de cada sprint o ocorre a *sprint planning* cujo objetivo é criar o *Backlog da Sprint* a partir do *Backlog do Produto*. O *Backlog* é uma lista de itens ordenada por prioridades, sendo que no backlog da sprint

são alocados os itens a serem realizados nesse período.

Com os objetivos definidos, inicia-se a fase de execução da sprint. Há ainda um alinhamento diário, as *daily's*, feito entre o time de desenvolvimento para discutir eventuais alterações nas atividades. Ao final da sprint, ocorre a *sprint review*, quando são validadas as tarefas finalizadas e feitas adaptações no produto.

2.1.1 Requisitos e Itens de *Backlog*

O levantamento de requisitos é tratado de forma diferente em projetos orientados a plano e em projetos Scrum (RUBIN, 2012). Nos desenvolvimentos de produtos orientados a plano, os requisitos são detalhados no início de projeto, então são entregues em forma de um documento altamente detalhado ao time de desenvolvimento, que deve desenvolver um produto conforme o especificado. Nesse caso, o principal é se adequar às especificações, e qualquer necessidade de alteração do plano inicial deve ser feita através de um processo de controle de mudança formal. Portanto, esses desvios se tornam caros e não são desejáveis, podendo causar desperdício (retrabalho ou mesmo descarte).

Por outro lado, no Scrum os requisitos têm um importante grau de liberdade para atingir os objetivos do negócio. Os detalhes são discutidos em conversas, que acontecem de forma contínua durante o desenvolvimento. Além disso, durante o desenvolvimento é possível remover ou adicionar requisitos, o primeiro se dando por questões de escassez de tempo ou de percepção durante o desenvolvimento de que um dado requisito não faz mais sentido, e o segundo ocorrendo de forma análoga quando se percebe que há um requisito de alto valor não visto antes ou se o desenvolvimento encontrar-se em um estágio mais avançado que o esperado por exemplo (RUBIN, 2012).

Em projetos de produtos inovadores, é inevitável o surgimento de novos requisitos durante o processo de desenvolvimento (RUBIN, 2012), portanto não é viável fazer um detalhamento completo dos requisitos previamente. Dessa forma, utilizando o Scrum, ao invés de tentar fazer no início um detalhamento completo dos requisitos, são criados *placeholders*, chamados Itens de Backlog de Produto (PBIs, do inglês *Product Backlog Items*), em que cada item representa algo desejável e de valor ao negócio. Assim, não há um gasto desnecessário com a especificação de requisitos inicial, pois é esperado que os requisitos do projeto mudem ao longo do tempo, conforme um maior entendimento do produto final (RUBIN, 2012).

No começo do projeto, os itens do PBIs são maiores - representam requisitos de alto

nível - e sem muitos detalhes. Esses itens são discutidos entre os *stakeholders* e divididos em menores e mais detalhados, até ser possível de serem alocados em uma *sprint*, para ser desenvolvido e testado. Há diversas formas de representar um item do *backlog* de produto, por exemplo com *histórias do usuário*.

No Scrum, conversas são a chave para garantir a discussão de requisitos. A comunicação verbal traz benefícios como maior velocidade na comunicação, *feedback* rápido, ser fácil e barato para compartilhar informações além de ser bidirecional, permitir descobrir novas ideias e oportunidades sobre o problema (RUBIN, 2012). Apesar de ser uma boa ferramenta, a conversa não substitui todos os documentos, no Scrum o *backlog* é um "documento vivo", disponível em todas as etapas do desenvolvimento.

Outra vantagem trazida pelo Scrum em relação ao desenvolvimento orientado a plano é o "Refinamento Progressivo", em que os requisitos não precisam estar no mesmo nível de detalhes simultaneamente. Nesse caso, os que serão abordados antes estarão menores e mais detalhados comparados aos abordados mais à frente. Há dessa forma vantagens em relação ao desenvolvimento orientado a plano (RUBIN, 2012), em que todos os requisitos devem estar detalhados igualmente ao mesmo tempo, fazendo com que seja necessário prever todos os requisitos antecipadamente, mesmo que o conhecimento sobre eles não seja suficiente. Isso gera um custo de retrabalho caso o entendimento sobre o sistema mude com o tempo ou mesmo caso haja mudanças de requisitos.

2.2 Histórias do usuário

As *histórias do usuário*, mencionadas anteriormente, são um formato conveniente de expressar itens de um *backlog*. Elas são escritas de maneira a serem entendidas por pessoas tanto de negócio como técnicas, estruturalmente simples e servem como *placeholders* para conversas, podem ser de vários níveis de granularidade e fáceis de serem refinadas progressivamente. Podem ser descritas como 3C 's: cartão, conversa e confirmação (RUBIN, 2012).

A ideia do *cartão* é simples: originalmente as histórias eram escritas em *sticky notes*. É comum utilizar um *template* para criação - "Como <perfil do usuário> eu gostaria de <objetivo> para que <benefício>". Um cartão não precisa incluir todas as informações, deve ser breve e conter a essência de um requisito, servindo como *placeholder* para discussões mais detalhadas entre os *stakeholders*.

O terceiro "C" mencionado se refere à *confirmação*: representado por critérios de

satisfação para considerar a história finalizada. Essas condições podem ser descritas como testes de alto nível com a finalidade de ajudar na clareza da comunicação entre o *PO* e o time de desenvolvimento, sendo os testes de baixo nível já verificados internamente pelo segundo.

As histórias do usuário apresentam níveis de detalhes diferentes, o que garante a possibilidade do refinamento progressivo. As maiores histórias e menos detalhadas (épicos) possibilitam uma visão geral do requisito e tem duração de meses. Um pouco mais detalhadas, com duração de semanas classificam-se as *features*. Por fim as histórias, com duração de dias, podem ser alocadas em *sprints* e durante o planejamento divididas em tarefas, em que o foco passa a ser como, ao invés de o que construir.

Wake (2003) propôs o critério INVEST (*Independentes*, *Negociáveis*, *Valiosas*, *Estimáveis*, *Pequenas/Tamanho ideal (Small/Sized-Apropriately)*, *Testáveis*) para garantir que sejam escritas boas histórias:

- **Independentes -** As histórias devem ser independentes ou pelo menos pouco dependentes uma da outra. A alta dependência atrapalha na estimativa, priorização e planejamento. Com esse critério o objetivo não é excluir todas as dependências, e sim escrever histórias de forma a minimizá-las.

- **Negociáveis -** Os detalhes de uma história devem ser negociáveis, o PO deve definir os porquês da história e não a forma de desenvolvê-la. Caso contrário, o time de desenvolvimento perde possibilidades de inovação.

- **Valiosas -** As histórias devem ter valor para o cliente, que paga pelo produto, ou para os usuários, que utilizam o produto. Histórias de viés técnico podem ser explicadas ao PO, que é quem vai decidir se há valor e deve ser incluída no *backlog*. Na prática, a maioria das história técnicas não são incluídas no *backlog*, entram como tarefas associadas a histórias.

- **Estimáveis -** As histórias devem ser estimáveis pelo time desenvolvimento. As estimativas do tamanho da história, do esforço e custo são necessárias para priorização feita pelo PO e para o time de desenvolvimento fazer o refinamento das histórias.

- **Pequenas/Tamanho ideal (Small/Sized-Apropriately) -** As histórias devem ter um tamanho apropriado de acordo com o período em que se planeja desenvolver. Para as *sprints*, devem ser pequenas, mas não há problema haver histórias maiores (épicos) que serão vistas mais a frente.

- **Testáveis -** As histórias devem ser testáveis de forma binária: passam ou falham, o que compõe as condições de satisfação.

2.2.1 Coletando Histórias

Há várias técnicas para coleta de histórias, dentre elas as mais importantes são entrevistas com usuários, questionários, observação e workshop de escrita de histórias (COHN, 2004). A seguir serão descritas as principais técnicas utilizadas nesse projeto.

- *Entrevistas* - entrevista é uma abordagem para coleta de requisitos que consiste em conversas com *stakeholders* para obter informações (IIBA, 2015). São feitas anotações durante as conversas e, a partir delas, criadas as histórias.
- *Story Mapping* - inicialmente é feita uma divisão bidimensional das histórias (RUBIN, 2012). Na horizontal as maiores histórias, chamados épicos, são organizadas pela sequência do fluxo de uso em temas. Em seguida, na vertical, os temas são refinados em histórias menores, de épicos a *features* e tarefas mais detalhadas e ordenados por prioridade para serem alocadas em *sprints*.

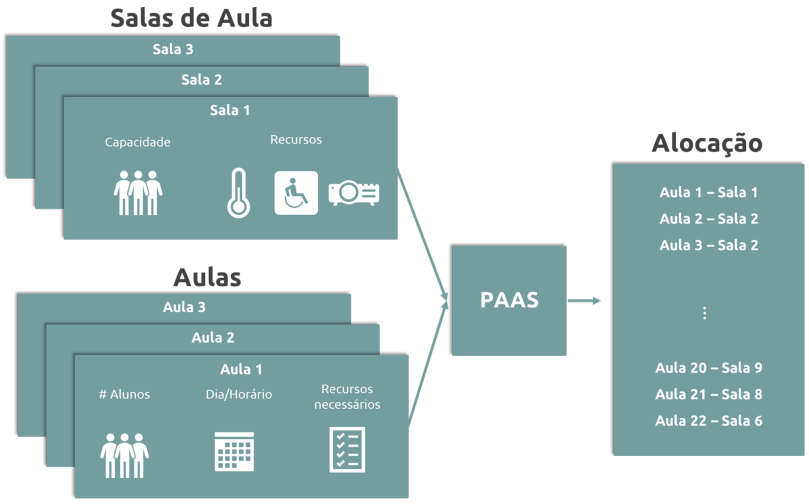
2.3 Problema de Alocação de Aulas às Salas (PAAS)

O Problema de Alocação de Aulas às Salas (PAAS) é um problema clássico de instituições de ensino superior, que atinge as instituições durante o início de seus anos letivos (CIRINO, 2016). O PAAS pode ser considerado como parte integrante do Problema de Programação de Cursos Universitários, que procura uma forma de definir uma distribuição de horários, docentes, e salas para realização das aulas segundo à demanda didática da Universidade (PHILLIPS et al., 2015).

Destaca-se a definição de alguns termos utilizados ao longo do trabalho baseados nos termos utilizados por Cirino (2016):

- **Aula/Evento:** um evento com horário de início, horário de fim e dia da semana definidos, que ocorre semanalmente dentro de um período compreendido por uma data de início e uma data de fim.
- **Sala:** Um espaço dedicado para comportar aulas/eventos.
- **Turma:** um conjunto de alunos com um professor, que possui uma ou mais aulas semanais, por exemplo uma turma 1 de Cálculo I que possua 2 aulas semanais.

Figura 2: Diagrama representativo do PAAS



- **Recursos:** são elementos disponíveis/contidos em uma sala e associados às aulas na forma de exigências didáticas ou ergonômicas. Por exemplo, acessibilidade, projetor e ar condicionado.

- **Decisor local:** indivíduo da instituição responsável por estabelecer e validar alocações de salas e aulas.

Do ponto de vista conceitual o PAAS pode ser modelado como um problema cuja solução visa encontrar a melhor distribuição de aulas (eventos) em salas (espaços), atendendo às restrições e preferências dos decisores locais (Figura 2). Esse problema é da categoria NP-Completo, redutível ao problema de k-coloração de grafo, verificado no trabalho de Carter e Tovey [1] (1992 apud CIRINO, 2016, p. 5).

Enquanto as restrições aplicáveis ao problema são em sua maioria intuitivas e de definição simples (por exemplo, duas aulas não podem ser alocadas em uma mesma sala para um mesmo horário, todas as aulas devem ser alocadas em uma e uma única sala, salas alocadas devem possuir os recursos necessários para ministrar cada aula), existem diferentes formas de se definir uma métrica de qualidade de solução, de forma a se elencar e definir quais configurações de alocação são mais desejáveis que outras, segundo as preferências dos decisores locais.

No contexto deste trabalho, o PAAS foi considerado como um problema com informações de evento dadas, em que as informações de horário, dia e docente responsável por ministrar cada aula já foram pré-estabelecidas, e cada docente pode indicar os re-

[1] CARTER, M. W.; TOVEY, C. A. **When is Classroom assigment problem hard?** *Operations Research*, v. 40, n. 1-supplement-1, p. S28S39, 1992.

cursos necessários para ministrar suas aulas. Estabeleceu-se também o uso da métrica de **Estabilidade de Sala** como métrica usual de qualidade para elencar alocações, onde aulas de uma mesma turma tentam ser mantidas em uma mesma sala durante a semana (CIRINO, 2016).

2.3.1 Formulação matemática do PAAS

Com o intuito de se modelar matematicamente PAAS, definiu-se o seguinte conjunto de variáveis e conjuntos de dados de preprocessamento, baseados nos mesmos utilizados por Cirino (2016):

Conjuntos e dados de Preprocessamento

T	Conjunto de turmas ($t \in \{1, ..., \|T\|\}$)
S	Conjunto de salas ($s \in \{1, ..., \|S\|\}$)
R	Conjunto de recursos ($r \in \{1, ..., \|R\|\}$)
A	Conjunto de aulas ($a \in \{1, ..., \|A\|\}$)
A^t	Conjunto das aulas $a \in A$ da turma $t \in T$
η	$\eta_{as} = 1$ se a sala $s \in S$ pode alocar a aula $a \in A$, 0 c.c.
Θ	$\Theta_{aa'} = 1$ se as aulas $a, a' \in A$ possuem sobreposição de dia e horário, 0 c.c.

Variáveis de decisão

x_{as}	$x_{as} = 1$ se a aula $a \in A$ é alocada na sala $s \in S$, 0 c.c.
y_t	$y_t =$ número de trocas de sala da turma $t \in T$

$$\min_{x,y} \quad F = \sum_{t \in T} y_t \tag{2.1a}$$

$$\text{sujeito a: } \sum_{s \in S} x_{as} = 1 \quad a \in A \tag{2.1b}$$

$$x_{as} \leq \eta_{as} \quad s \in S \text{ e } a \in A \tag{2.1c}$$

$$x_{as} + x_{a's} \leq 1 \quad s \in S \text{ e } a, a' \in A | \Theta_{aa'} = 1 \tag{2.1d}$$

$$\sum_{n=1}^{|A^t|} x_{a_n s_n} \leq 1 + y_t \quad t \in T, a_n \in A^t \text{ e } \{s_1, ..., s_{|A^t|}\} \subset S \tag{2.1e}$$

$$x_{as} \in 0, 1 \quad a \in A, s \in S \tag{2.1f}$$

$$y_t \in \mathbb{Z}_+ \quad t \in T \tag{2.1g}$$

As variáveis de decisão **x** e **y** codificam as possíveis configurações de soluções, caracterizando portanto os graus de liberdade do problema. O conjunto de variáveis x codifica as possíveis combinações de alocações de aulas e salas, de forma que, para uma determinada aula $a_1 \in A$ e sala $s_1 \in S$, $x_{a_1 s_1} = 1$ indica uma que a_1 foi alocada em s_1, e $x_{a_1 s_1} = 0$ indica o contrário. Por sua vez, y codifica a quantidade de trocas de salas incorridas por uma determinada aula (uma vez que o problema procura minimizar y), ou seja, para uma dada turma $t_1 \in T$, $y_t = 2$ indica que a turma t_1 teve duas trocas de salas na configuração analisada. Resolver o problema apresentado significa achar a combinação de valores de **x** e **y** que minimiza o valor da função objetivo (2.1a), ao mesmo tempo satisfazendo as restrições (2.1b) - (2.1g).

A função objetivo (2.1a) minimiza o número de trocas de salas para aulas de uma mesma turma. O conjunto de restrições (2.1b) assegura que todas as aulas devem ser alocadas em uma, e uma única sala. As restrições (2.1c) asseguram que aulas serão alocadas apenas em salas que atendam às necessidades de localidade (prédio onde a aula deve ser ministrada) e de recursos necessários. As restrições (2.1d) garantem que duas aulas com sobreposição de dia e horário não possam ser alocadas em uma mesma sala. As restrições (2.1e) ficam responsáveis por medir a quantidade de trocas de salas realizadas por uma mesma turma.

Essa formulação matemática se assemelha em grande parte àquela proposta no trabalho de Cirino (2016), porém apenas com a métrica de qualidade de estabilidade de turmas, uma vez que chegou-se à conclusão, junto dos decisores locais deste trabalho, de que as demais métricas apresentadas não seriam relevantes no contexto proposto.

Uma vez estabelecida uma formulação matemática para o problema, discute-se a abordagem de resolução do mesmo como um problema de otimização.

2.3.2 Resolução do problema de otimização

A formulação proposta anteriormente se encaixa na classe de problemas de otimização de Programação Linear (WINSTON; GOLDBERG, 2004, p. 53):

> Um **Problema de Programação Linear** (PL) é um problema de otimização para o qual:
>
> 1. Tenta-se maximizar (ou minimizar) uma função *linear* nas variáveis de decisão. A função a ser maximizada ou minimizada é denominada **função objetivo**.

2. Os valores das variáveis de decisão devem satisfazer um conjunto de *restrições*. Cada restrição deve ser uma equação linear ou uma inequação linear.
3. Uma *restrição de sinal* é associada com cada variável. Para cada variável x_i, a restrição de sinal especifica que x_i precisa ser ou não-negativa ($x_i \geq 0$) ou irrestrita em sinal (irs).

Mais especificamente, uma vez que as variáveis de decisão de alocação x_{as} e de troca de salas y_t assumem apenas valores inteiros não negativos, a formulação se encaixa na classe de problemas de **Programação Linear Inteira**. Ter uma formulação que se encaixa nessa classe de problemas apresenta sua utilidade na medida em que é possível se utilizar de métodos de resolução disponíveis na literatura, como o algoritmo Simplex.

No contexto deste trabalho, foram testados dois solucionadores de Programação Linear Inteira já implementados (*solvers*, descritos no capítulo 6): o *Coin-or Branch and Cut* (CBC) (FORREST et al., 2022), um solucionador open-source e de uso gratuito, e o solucionador *Gurobi* (Gurobi Optimization, LLC, 2022), uma solução comercial para problemas de otimização.

2.4 Considerações do capítulo

Neste capítulo foram apresentados o *framework* Scrum e o formato de Histórias de Usuário, os quais foram usados para desenvolver o projeto. Uma parte importante do projeto envolve o Problema de Alocação de Aulas às Salas, uma formulação matemática que providencia uma abordagem para gerar alocações válidas de salas seguindo um conjunto de restrições, e otimizando para uma métrica de qualidade desejada.

Uma vez definidas as bases conceituais utilizadas no decorrer deste trabalho, tanto de um ponto de vista de organização do trabalho, para o desenvolvimento do sistema proposto, quanto de um ponto de vista técnico, para a implementação de funcionalidades do mesmo, passa-se à análise do contexto no qual tal solução estaria inserida, olhando para trabalhos e soluções que se propuseram a atacar problemas parecidos com o aqui abordado.

3 TRABALHOS RELACIONADOS

Neste capítulo são explorados os principais trabalhos e soluções relacionados ao sistema proposto, que serviram de base para tomadas de decisão sobre funcionalidades e aspectos diferenciadores do mesmo. São exploradas duas soluções de gerenciamento e visualização de alocações de sala, ambas já utilizadas por prédios da EPUSP: o sistema USPolis original, que serviu de base para estabelecimento de requisitos e funcionalidades, e ao qual esse trabalho se propõe ser uma refatoração, com a implementação de novas funcionalidades; e o sistema *Meeting Room Booking System* (MRBS), solução open-source, gratuita, e em uso no prédio das Engenharias Elétrica e de Computação da instituição no momento da realização deste trabalho.

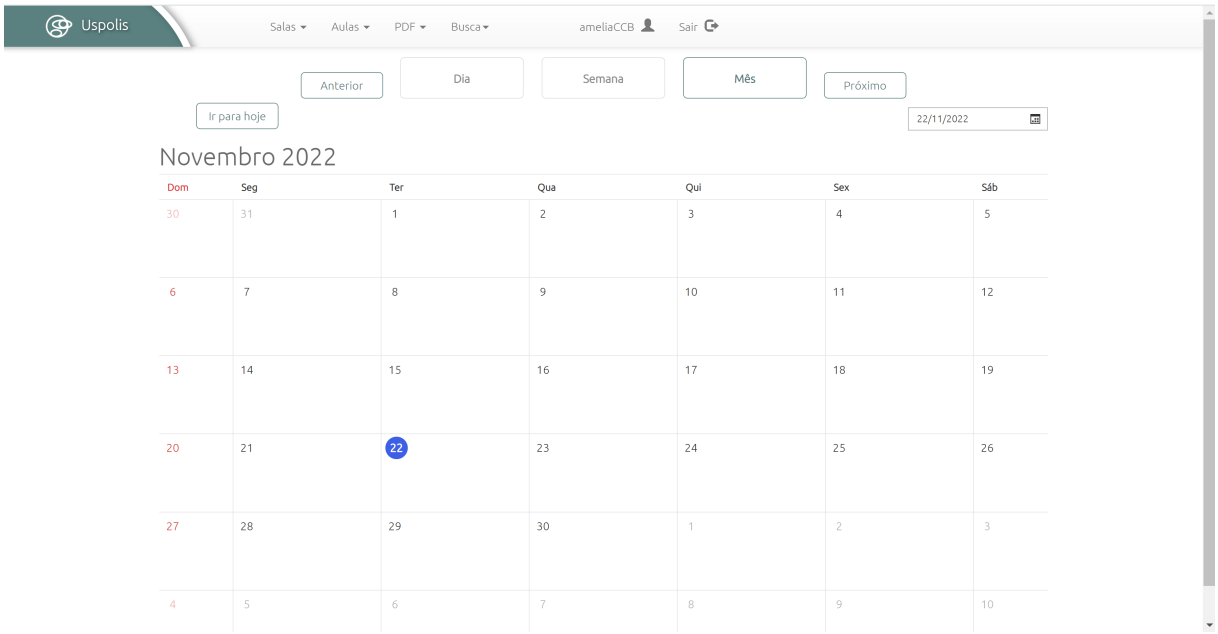
3.1 Sistema USPolis original

O sistema USPolis original (figura 3) foi desenvolvido pelo ex-aluno da Escola Politécnica da USP (EPUSP), e co-orientador deste trabalho, Renan de Luca Avila, durante o período de sua graduação em Engenharia da Computação, com a finalidade de resolver o problema de gestão de alocações de salas dos prédios da EPUSP. Durante os anos de 2017 a 2019 ele foi utilizado no prédio do Biênio da EPUSP como solução para gestão de alocação de salas. Entre os anos de 2020 e 2021, dado principalmente o contexto de pandemia, quando não houveram aulas presenciais, não houve mais demanda pelo sistema, e ele não foi mais atualizado. Este projeto, entre outros aspectos, visa dar continuidade ao sistema original, re-implementando suas principais funcionalidades agregadoras de valor para a EPUSP, e propondo melhorias em diferentes aspectos, principalmente no que se diz respeito à manutenibilidade do mesmo por futuras equipes de desenvolvimento.

3.1.1 Funcionalidades

Dentre outras, o sistema USPolis original apresentava o seguinte conjunto de funcionalidades principais:

Figura 3: Sistema USPolis Original



Fonte: sistema USPolis original.

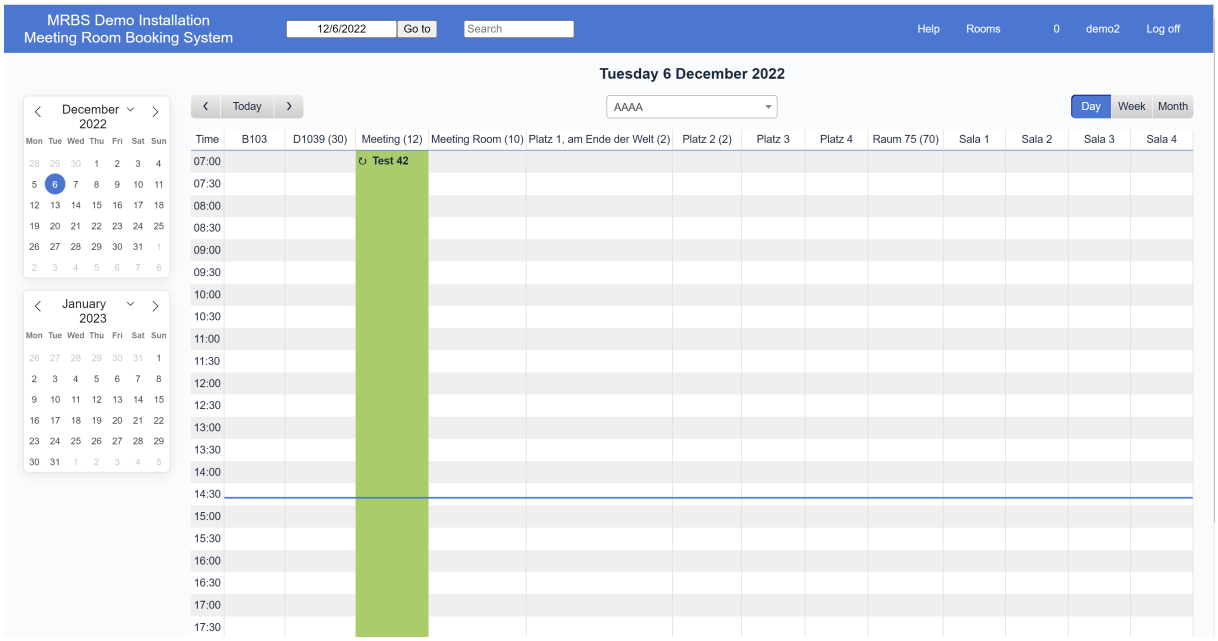
1. Disponibilidade através de navegador web;
2. Cadastro e gestão de informações de salas;
3. Cadastro e gestão de informações de aulas;
4. Visualizações de alocação por dia, semana e mês;
5. Visualizações de alocação por sala;
6. Criação de PDF's de alocações de salas;
7. Controle da gestão de dados do sistema por autenticação de usuário;
8. Aplicativo mobile voltado para os alunos checarem onde ocorreriam suas aulas;

3.2 Meeting Room Booking System - MRBS

O *Meeting Room Booking System* (MRBS) (figura 4) é uma aplicação web gratuita, baseada em PHP e MySQL/pgsql, de reserva de salas de reunião e outros recursos, distribuída sob a Licença Pública Geral GNU (GPL) (J. Beranek et. al, 2022). Com a finalidade de uso voltada para reservas de salas de reuniões, o MRBS apresenta uma proposta mais voltada para o mundo corporativo, porém, dadas as similaridades dos problemas, seu modelo adapta-se bem à realidade de gestão de alocação de salas de aulas

em meios universitários, sendo ele, no período da realização deste trabalho, a solução adotada para a gestão de alocação de salas de aula do prédio das Engenharias Elétrica e de Computação da EPUSP.

Figura 4: Sistema MRBS



Fonte: https://mrbsdemo.uk.to/periodmrbs/

3.2.1 Funcionalidades

O MRBS apresenta uma lista de funcionalidades principais (J. Beranek et. al, 2022):

1. Disponibilidade de acesso através de navegador web;
2. Reservas recorrentes flexíveis;
3. Serviço de autenticação a partir de bases de usuário existentes;
4. Diferentes níveis de visualização - dia, semana, mês;
5. Reservas válidas por períodos de tempo definidos;
6. Bloqueio de reservas conflitantes;
7. Notificação de reservas por e-mail;

3.3 Considerações do capítulo

A Tabela 1 apresenta um resumo das principais funcionalidades e tecnologias presentes nas duas soluções analisadas, comparadas às do sistema proposto (detalhadas no capítulo 7). Nota-se que, em questão de possuir ou não, ambas soluções anteriores apresentam quase o mesmo conjunto de funcionalidades. Contudo, como é analisado nos Capítulos 4 e 5, o formato como tais funcionalidades são apresentadas e dispostas à utilização do usuário possui também um alto grau de importância, e portanto deve ser considerado ao se julgar diferentes opções de solução. Como cada instituição tem suas especificidades em relação à gestão e alocação de suas salas, ter um sistema desenvolvido com um contexto específico em mente (USPolis Original) pode se apresentar mais vantajoso do que um sistema projetado para um contexto mais geral (MRBS). Ademais, ressalta-se o fato de que, mesmo ambas soluções providenciando plataformas convenientes para cadastro e gestão das informações necessárias, ainda há uma grande dependência de ação manual do usuário nesse processo, limitação que o sistema proposto visa atacar.

Uma vez analisadas soluções que propõe resolver problemas análogos ao do sistema desenvolvido neste trabalho, passa-se à descrição do processo de desenvolvimento do mesmo, começando pela metodologia de trabalho utilizada.

Tabela 1: Trabalhos Relacionados - Resumo de funcionalidades e tecnologias

	USPolis Original	MRBS	USPolis - Sistema Proposto
Interface de usuário	Web - PHP	Web - PHP	Web - React JS
Controle/gestão de informações	Mysql	Mysql	Mongo DB
Autenticação de usário	Sim	Sim	Sim
Cadastro de informação de aulas	Manual	Manual	Automático (através de Web Crawler)
Cadastro de ínformação de salas	Manual	Manual	Manual
Processo de alocação salas	Manual	Manual	Automático
Criação de PDF's de alocação	Sim	Não	Sim
Aplicativo mobile	Sim	Não	Não
Notificação por email	Não	Sim	Não

4 METODOLOGIA DE TRABALHO

Seguindo o *framework Scrum*, o desenvolvimento do projeto foi dividido em *sprints*, com 2 ou 3 semanas de duração. A cada sprint, cartões de Histórias de Usuário foram atribuídos à categoria "Em Andamento", indicando que serão o foco de desenvolvimento daquela *sprint*. Cada cartão atribuído à *sprint* em questão deverá ter seus critérios de aceite definidos, de forma a poder ser considerado finalizado uma vez seus critérios atendidos.

4.1 Sprints

A divisão das sprint foi realizada de acordo com o escopo definido para o projeto, totalizando 3 sprints iniciais de 3 semanas e 3 últimas de 2 semanas de duração. O encurtamento das últimas sprints foi feito para um melhor acompanhamento das atividades, com objetivo de obter um MVP ao final do desenvolvimento.

4.1.1 Desenvolvimento inicial

O desenvolvimento de uma versão inicial, utilizada nas primeiras validações, foi realizado majoritariamente nas 3 sprints com maior duração. As histórias foram desenvolvidas seguindo a ordem do processo de alocação, dessa forma foi possível realizar validações intermediárias, descritas no capítulo 8, dos módulos do sistema.

1. **Cadastros**
   Os cadastros de salas e turmas foram priorizados por serem fundamentais tanto para as principais visualizações quanto para o cálculo da alocação.

2. **Visualizações e Algoritmo de alocação**
   Com os cadastros funcionais foram desenvolvidas as telas para visualizar as informações, além das ações para cadastrar, editar e removê-las. Paralelamente foi

iniciada a formulação matemática do problema para automatizar o processo de alocação.

3. **Tradução dos dados de alocação e visualização**
   A seguir foi feita a integração do algoritmo matemático com o sistema, em paralelo com a visualização das salas às quais cada turma foi alocada.

4.1.2 Validações

Com a aplicação parcialmente desenvolvida, foram realizadas validações com os usuários para coletar *feedbacks* e sugestões para novas funcionalidades. Os testes foram realizados de 3 formas:

- **Sem explicar o sistema**
  O sistema foi apresentado ao usuário sem instruções de como utilizá-lo, para avaliar sua intuitividade e usabilidade.
- **Sugerindo novas funcionalidades**
  Foram discutidas ideias para novas funcionalidades, obtendo um melhor entendimento do processo de alocação.
- **Utilização assistida**
  O sistema foi apresentado ao usuário e explicado seu funcionamento, com intuito de verificar o alinhamento das expectativas com as funcionalidades implementadas até então.

O resultado das avaliações, descrito no Capítulo 8, foi positivo, indicando um sistema intuitivo e capaz de ser responsável pelo processo de alocação de salas.

4.2 Levantamento de Requisitos

O levantamento de requisitos foi feito utilizando as técnicas descritas a seguir.

4.2.1 Entrevistas

Considerando o escopo do Biênio e da Engenharia Elétrica, inicialmente definido para o projeto, foram realizadas entrevistas com os principais *stakeholders* para o levantamento de requisitos. Foram feitos diagramas para ajudar na compreensão do problema: a figura

5 representa o processo de alocação; a figura 6 demonstra o problema enfrentado na semana de provas e na alocação de eventos pontuais; na figura 7 apresenta um resumo dos principais pontos levantados.

Figura 5: Anotações de entrevistas do usuário

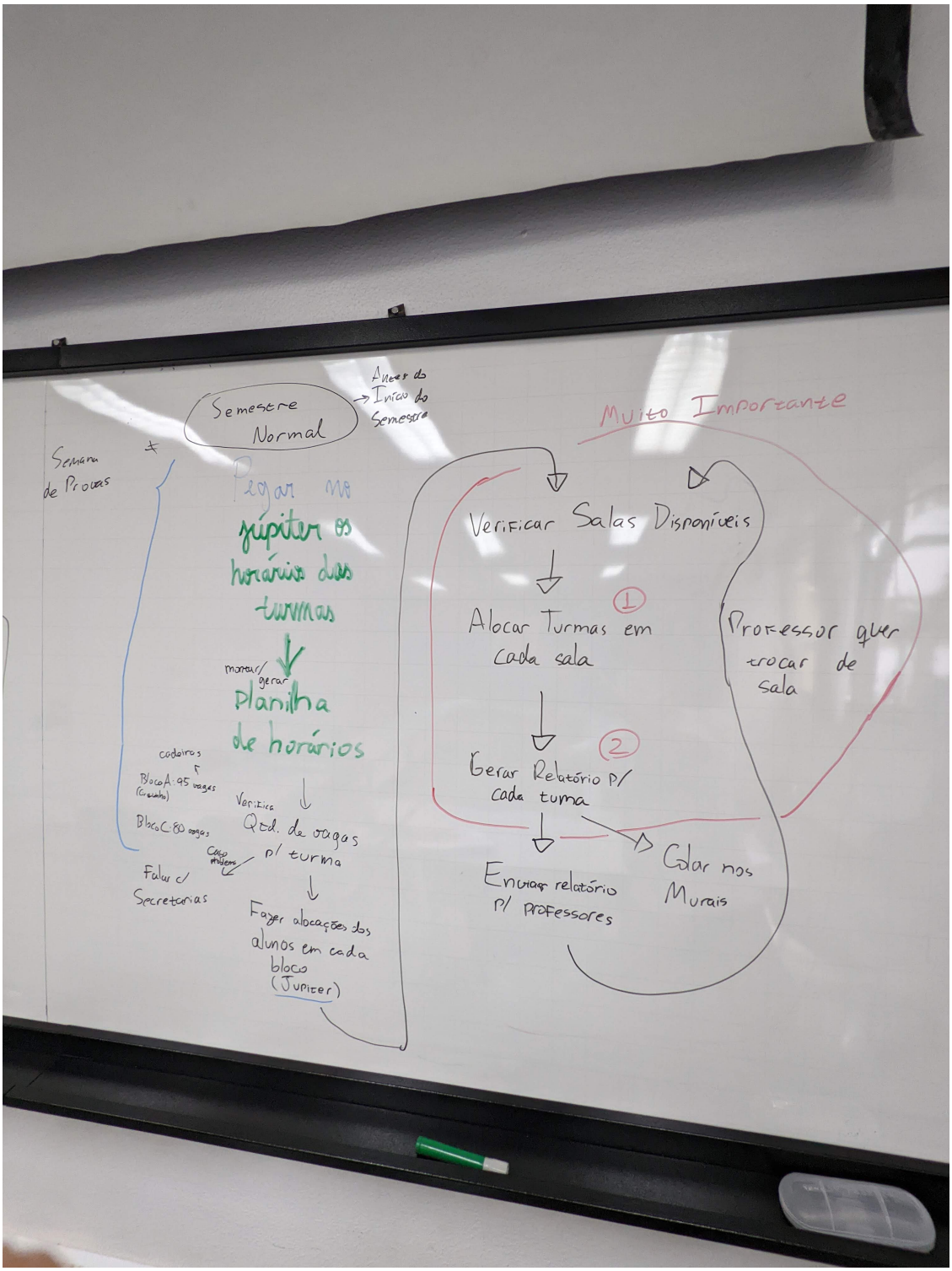


Figura 6: Anotações de entrevistas do usuário

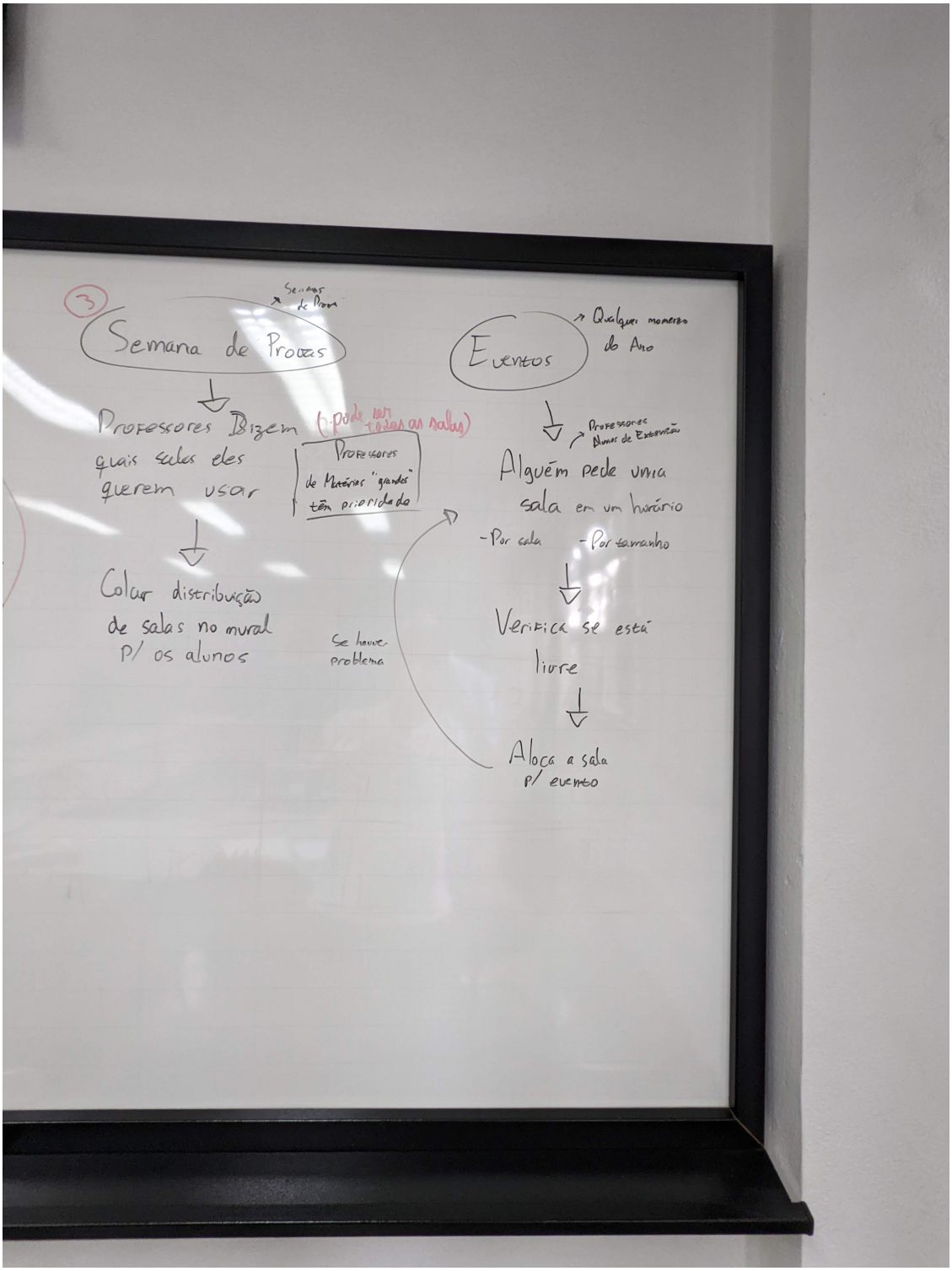
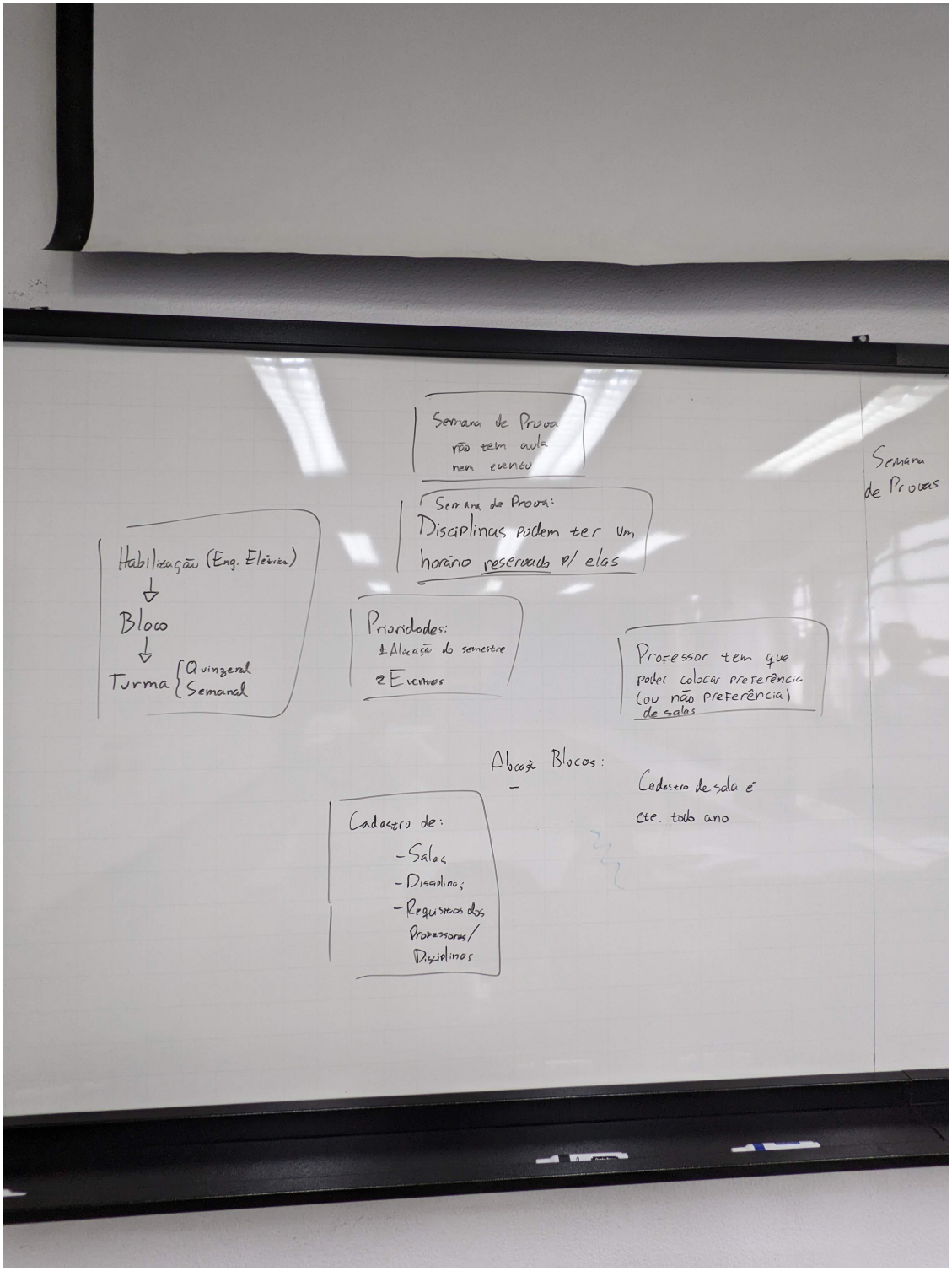


Figura 7: Anotações de entrevistas do usuário



- **Responsável pela alocação no Biênio**

  O objetivo da conversa foi entender o procedimento realizado e encontrar soluções para otimizá-lo. Foi observado que o processo é feito de forma manual, utilizando agendas e planilhas impressas e demandam um esforço devido a problemas observados tal como conflitos de horários, exigências dos professores, mudanças frequentes

tanto de horários quanto de professores e ocasionais indisponibilidades.

A partir da reunião foi possível estabelecer um fluxo principal de atividades como também questões pontuais.

1. Principal
   (a) Buscar no JupiterWeb as informações a respeito das turmas
   (b) Verificar salas disponíveis
   (c) Alocar turmas nas salas
   (d) Gerar relatório para cada turma
2. Eventuais problemas
   (a) Professor solicita troca de sala
   (b) Semana de provas
   (c) Eventos como palestras por exemplo

- **Responsável pela alocação na Elétrica**

  Nesse caso, a alocação é feita de forma um pouco menos manual, utilizando uma planilha dinâmica com informações sobre as disciplinas e as salas de aula no Excel em conjunto com um sistema open-source para reserva de salas, Meeting Room Booking System (MRBS)[1], integrado ao site da elétrica. Além disso, foram levantadas diferenças em relação ao procedimento no Biênio, problemas vivenciados anteriormente e sugestões de funcionalidades para o projeto.

  1. Diferenças quanto ao Biênio
     (a) Professores determinam a sala e horário específicos
     (b) As disciplinas têm os horários e professores mais fixos
  2. Problemas observados
     (a) Falta de um lugar centralizado com as informações das disciplinas
     (b) Falta de informações sobre professores e/ou disciplinas
     (c) Principal restrição das salas é a capacidade, necessidade de assertividade na quantidade de alunos inscritos
     (d) Conflitos com as disciplinas de pós-graduação e do quadrimestral (calendários diferentes)
  3. Funcionalidades sugeridas

[1] ⟨https://mrbs.sourceforge.io/⟩

(a) Sugerir alocação que seja facilmente editável
(b) Separação por disciplina e por professor
(c) Busca ativa no JupiterWeb por novas disciplinas
(d) Exportar mapa das disciplinas para divulgação
(e) Exportar relatório interno
(f) Integração com outros serviços como MatrUSP, Google Agenda

4.2.2 Sistema USPolis antigo

Outra fonte de requisitos utilizada foi a versão original do sistema USPolis, mediante a execução do sistema (o qual foi restaurando em uma máquina virtual para a elicitação de requisitos) e também conversas com o criador do sistema, e também coorientador desse projeto. Nas reuniões com o coorientador foram apresentadas as funcionalidades do projeto original, principais problemas enfrentados e a evolução do sistema para solucioná-los.

4.3 Considerações do capítulo

Foram apresentadas as metodologias utilizadas durante o desenvolvimento do projeto, seguindo os princípios ágeis do Scrum e dividindo o desenvolvimento em sprints. Além disso foram mostradas as principais técnicas aplicadas no levantamento de requisitos, que foram fundamentais para definir as principais funcionalidades do sistema. Os resultados serão discutidos na próxima seção.

5 ESPECIFICAÇÃO DE REQUISITOS DO SISTEMA

A partir das entrevistas realizadas, foram escritas histórias do usuário para compor o backlog do produto. Nesse capítulo são apresentados os principais usuários e *stakeholders* identificados, além das histórias escritas a partir do levantamento de requisitos feito.

5.1 Usuários e *Stakeholders*

A partir da ideia e do objetivo do projeto, é possível identificar os principais *stakeholders*:

- **Decisor local -** utilizará o sistema web para fazer a alocação das disciplinas nas salas de aula, considerando a disponibilidade das mesmas, o horário das aulas e possíveis restrições. O projeto busca melhorar a forma como é feita a alocação, realizando a maior parte do processo de forma automática e o decisor irá precisar realizar menos operações manuais. No desenvolvimento do projeto, essa posição foi representada pelos responsáveis por fazer a alocação das salas do Biênio e da elétrica, atuando como principal fonte de requisitos para o sistema.
- **Administrador -** responsável pela manutenção do sistema. O cadastro de usuários do sistema para diferentes prédios é feito de forma manual e deve ser realizado pelo admnistrador.
- **Professor -** interage indiretamente com o sistema, fazendo solicitações de recursos nas salas, mudanças de horário ou mudança de sala ao decisor local.
- **Organizador de eventos -** representa qualquer pessoa interessada em solicitar a reserva do horário em uma sala para realização de eventos, como coordenadores de disciplinas, centros acadêmicos, departamentos e empresas externas.

5.2 Histórias do usuário

Através das entrevistas realizadas com os usuários com papel de "secretário" organizou-se todo o processo de alocação de salas em histórias de usuário que descrevem ações realizadas, agentes realizadores, e sua finalidade. O formato usual para histórias segue o formato sugerido por Cohn (2004): "Como [usuário], eu gostaria de fazer [ação] para [finalidade]". As histórias foram organizadas em temas:

- **Preparação para alocação -** Corresponde às tarefas de aquisição e preparação das informações que precisam estar prontas para serem utilizadas no momento da alocação, e que não têm uma data ou período específico para serem executadas.
- **Alocação de salas -** Corresponde às tarefas diretamente ligadas à atividade de alocação das salas, e que são realizadas historicamente nas semanas que precedem o início das aulas da graduação.
- **Funcionalidades pontuais -** Corresponde a funcionalidades pontuais mencionadas pelos secretários que eram úteis no sistema anterior, ou que seriam úteis de se possuir no sistema novo.
- **Desenvolvimento do projeto -** Corresponde às tarefas relacionadas ao processo de desenvolvimento do sistema e do projeto de conclusão de curso, presentes apenas por questões de organização do projeto.

Também foram atribuídas prioridades para cada uma das histórias. A prioridade 1 foi atribuída às histórias julgadas essenciais para o funcionamento do sistema. A prioridade 2 foi atribuída às histórias cujas funcionalidades seriam importantes para o sistema, porém não essenciais. Por fim a prioridade 3 foi atribuída às histórias cujas funcionalidades representariam um adicional interessante ao sistema, porém poderiam ser despriorizadas em uma primeira versão.

- Cadastro de salas - Prioridade 1

  Como secretário eu gostaria de cadastrar as salas de aula do prédio para que elas possam ser alocadas para uma turma

- Informações de turmas - Prioridade 1

  Como secretário eu gostaria de pegar as informações dos horários das turmas do Júpiter Web de forma automática para não precisar fazê-lo manualmente

- Novas disciplinas - Prioridade 2

  Como secretário eu gostaria de saber se uma disciplina foi adicionada ou excluída do sistema

- Turmas Pós-Graduação - Prioridade 2

  Como secretário eu gostaria de lidar com as turmas de Pós Graduação

- Oferecimentos do quadrimestral - Prioridade 2

  Como secretário eu gostaria que alocação levasse em conta tanto as disciplinas do semestral quanto do quadrimestral (calendários diferentes)

- Feriados - Prioridade 3

  Como secretário eu gostaria de puxar as datas de feriados automaticamente

- Preferências de sala - Prioridade 1

  Como secretário eu gostaria de colocar restrições para que as preferências de sala dos professores sejam atendidas

- Alocar turmas - Prioridade 1

  Como secretário eu gostaria de alocar diferentes turmas de uma disciplina em salas diferentes

- Edição de alocação - Prioridade 1

  Como secretário eu gostaria de editar as alocações das salas a qualquer momento para lidar com mudanças repentinas

- Salas disponíveis - Prioridade 1

  Como secretário eu gostaria de verificar as salas disponíveis para conseguir alocar uma disciplina em uma sala

- Semana de provas - Prioridade 2

  Como secretário eu gostaria de fazer alocações específicas para as semanas de prova

- Gerar cartaz - Prioridade 2

  Como secretário eu gostaria de poder gerar automaticamente cartazes com os horários e locais das disciplinas para poder colar nos murais ou para uso interno

- Alterar ocorrências - Prioridade 2

  Como secretário eu gostaria alterar uma série de ocorrências de uma disciplina de uma vez

- Eventos - Prioridade 3

  Como secretário eu gostaria de poder alocar salas para eventos pontuais

- Visualização de disciplinas - Prioridade 3

  Como secretário eu gostaria de ter uma visualização consolidada de todas as disciplinas do prédio

- Visualização de salas - Prioridade 3

  Como secretário eu gostaria de ter uma visualização consolidada das salas do prédio

- Visualização de professores - Prioridade 3

  Como secretário eu gostaria de ter uma visualização consolidada das disciplinas de um professor

- Escolha das tecnologias - Prioridade 1

  Como desenvolvedor eu gostaria de definir as tecnologias que serão utilizadas no sistema

- Restaurar servidor - Prioridade 1

  Como desenvolvedor eu gostaria de restaurar a versão antiga do USPolis para realizar testes e levantar funcionalidades

- Filtrar matérias - Prioridade 3

  Como aluno eu gostaria de filtrar as disciplinas para visualizar somente as minhas turmas

- Conectar com Matrusp - Prioridade 3

  Como aluno eu gostaria de pegar dados do sistema Matrusp para o USPolis

- Passar conhecimento do sistema - Prioridade 3

  Como secretário eu gostaria de passar o conhecimento de utilização para o próximo secretário

- Exportar para Google Agenda - Prioridade 3

  Como aluno eu gostaria de exportar minhas disciplinas para o Google Agenda

5.3 Planejamento de release

O planejamento de release é um planejamento de longo prazo para determinar as funcionalidades que serão entregues (RUBIN, 2012). No escopo do projeto foi realizado um planejamento para entrega de um produto final com as funcionalidades necessárias para utilização.

5.3.1 Produto Mínimo Viável (MVP)

O MVP foi definido pelo planejamento a partir da priorização e escolha das histórias do usuário. Foram escolhidas as histórias com prioridade 1, fundamentais para o processo de alocação, e definidas funcionalidades para cada.

- Restaurar servidor
  - Realização de testes
  - Fonte de requisitos
- Escolha das tecnologias
  - Escolha da base de dados
  - Criação do setup inicial do backend
  - Criação do setup inicial do frontend
- Cadastro de salas
  - Cadastrar uma nova sala;
  - Editar as informações de uma sala;
  - Excluir uma sala;
- Informações de turmas
  - Buscar as informações das turmas automaticamente do Júpiter Web pelo código da disciplina
  - Excluir uma turma

  - Editar as informações de uma turma

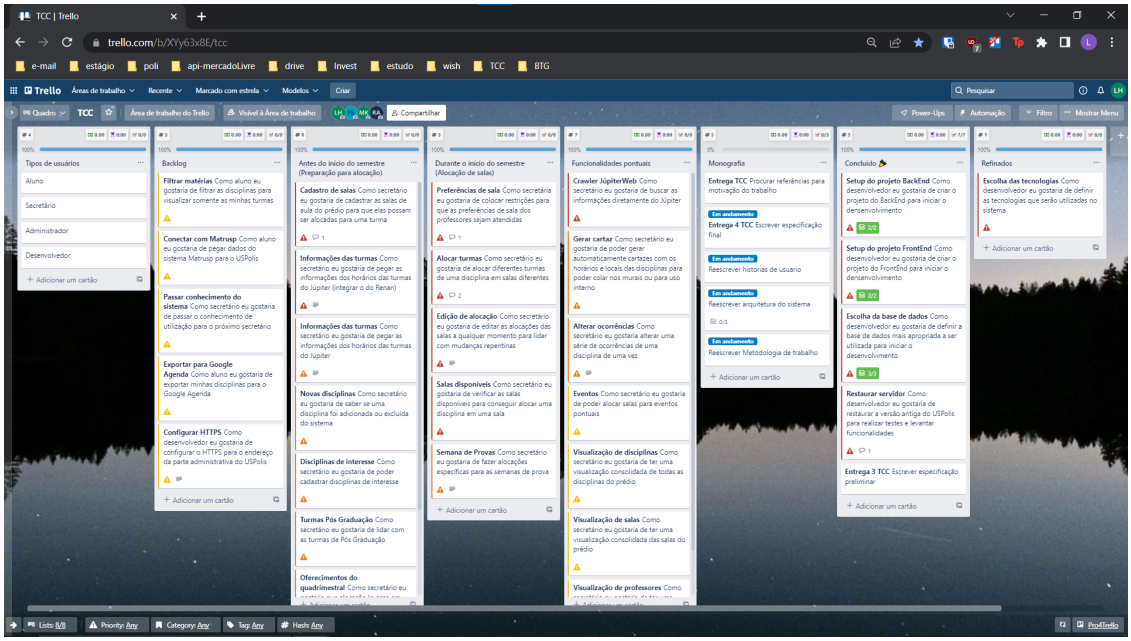
- Preferências de sala
  - Editar as preferências da sala para uma turma
- Alocar turmas
  - Calcular a alocação para as turmas e salas cadastradas, seguindo restrições de horários, capacidade e preferências
- Editar alocação
  - Alterar a sala designada a uma turma
- Salas disponíveis
  - Buscar as salas disponíveis em um horário

5.4 Considerações do capítulo

Nesse capítulo foram apresentados os requisitos elicitados para o sistema, através das entrevistas e análise do sistema antigo. Os requisitos foram documentados através de histórias do usuários, as quais foram colocadas no Trello - figura 8. Além disso as histórias foram priorizadas para composição de um Produto Mínimo Viável e se adequar ao escopo do projeto.

No próximo capítulo será apresentada a arquitetura do sistema, a qual foi criada considerando esses requisitos.

Figura 8: Backlog de histórias do usuário



Fonte: https://trello.com/b/XYy63x8E/tcc

6 ARQUITETURA DO SISTEMA

Neste capítulo é apresentada a arquitetura do sistema e as principais tecnologias usadas, junto da motivação para a escolha das mesmas. Aborda-se inicialmente uma visão geral das partes constituintes da arquitetura, e a forma como elas se comunicam, seguida por uma descrição mais detalhada das tecnologias que compõe o *backend*, parte da aplicação não visível ao usuário e responsável pelo funcionamento, e o *frontend*, responsável pela interface gráfica da aplicação, da solução, assim como a plataforma de hardware utilizada para a hospedagem da mesma. Por fim são apresentadas duas discussões sobre o racional de escolha das tecnologias de banco de dados e de solucionador do PAAS (Problema de Alocação de Aulas às Salas, descrito no Capítulo 2.3), uma vez que representam dois aspectos centrais do projeto – armazenamento das informações de salas e aulas, e solução automatizada do problema de alocação.

A escolha de tecnologias do sistema levou em conta 3 aspectos: a sua adequação para resolução do problema em questão, a sua adequação à metodologia de trabalho empregada no projeto, e a sua adoção pela comunidade, de forma a garantir que o sistema possa ser facilmente mantido por outras equipes no futuro.

6.1 Backend

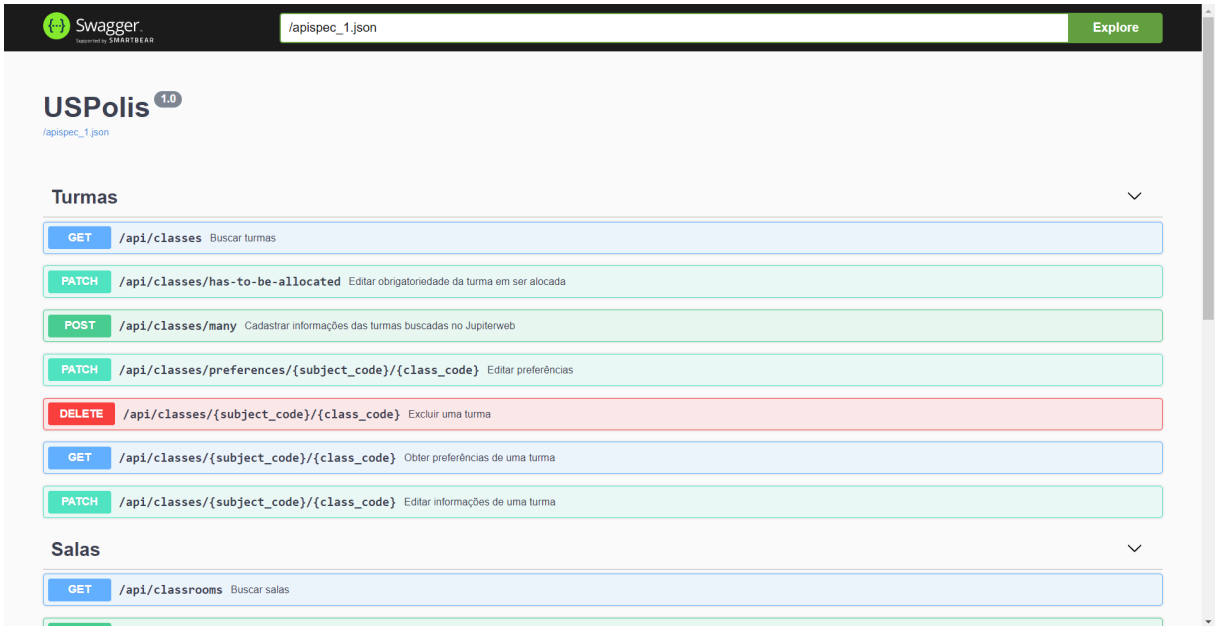
O backend atua por trás da aplicação, é responsável pelas regras internas do sistema e por fazer a comunicação com o banco de dados. Escolheu-se o Flask ⟨https://flask.palletsprojects.com⟩ como *framework* web por se tratar de um módulo de Python, o que possibilita a utilização de uma grande quantidade de outros módulos e ferramentas eventualmente úteis na resolução do problema de otimização de distribuição das salas.

O Flask é um *framework* focado no desenvolvimento de aplicações simples, devido a simplicidade de sua arquitetura, o que proporciona uma maior agilidade no desenvolvimento inicial. Além disso ainda é possível desenvolver aplicações mais robustas, pois o Flask é amplamente personalizável, viabilizando uma alta escalabilidade.

Além do Flask, inicialmente foram utilizadas outras bibliotecas:

- *Marshmallow* ⟨https://marshmallow.readthedocs.io⟩ - biblioteca para auxiliar na manipulação dos dados; mapeamento e validação através da criação de Schemas para as diferentes informações;
- *Pymongo* ⟨https://pymongo.readthedocs.io⟩ - ferramenta de integração do Python com o MongoDB, que foi a base de dados escolhida para o projeto;
- *BeautifulSoup* ⟨https://www.crummy.com/software/BeautifulSoup/⟩ - biblioteca para auxiliar na coleta de informações das turmas buscadas no sistema do Jupiterweb;
- *PuLP* ⟨https://coin-or.github.io/pulp/⟩ - camada de abstração para modelagem de problemas lineares e resolução utilizando um otimizador compatível;
- *GUROBI* ⟨https://www.gurobi.com/⟩ - otimizador comercial utilizado sob licença acadêmica;
- *Apispec* ⟨https://apispec.readthedocs.io/en/latest/⟩ e *Flasgger* ⟨https://github.com/flasgger/flasgger⟩ - bibliotecas auxiliares para a documentação da API a partir da geração de uma especificação OpenAPI - Figura 9.

Figura 9: Interface Swagger da API



6.2 Frontend

O frontend é composto pela parte visual do sistema, onde ocorre a interação com o usuário. O React ⟨https://pt-br.reactjs.org/⟩ foi escolhido como tecnologia para implementação do Frontend Administrativo por sua grande adoção pela comunidade desenvolvedora, adicionado ao fato da equipe ter certa experiência prévia com o *framework*.

A linguagem escolhida para o projeto foi o TypeScript ⟨https://www.typescriptlang.org/⟩, baseada no JavaScript com uma sintaxe adicional para tipagem no código. Tal funcionalidade garante uma maior consistência dos dados, segurança e facilidade na continuação do projeto.

Para o desenvolvimento, além do React, as principais bibliotecas utilizadas foram:

- *Chakra UI* ⟨*https://chakra-ui.com/*⟩ - biblioteca para auxiliar na construção e estilização dos componentes e garantir uma boa experiência de utilização do usuário;
- *React Router* ⟨*https://reactrouter.com/*⟩ - biblioteca responsável pelo controle e gerenciamento das rotas da aplicação;
- *Axios* ⟨*https://axios-http.com/ptbr/docs/intro*⟩ - ciente HTTP para realizar as requisições e troca de dados com o servidor
- *AWS Amplify* ⟨*https://docs.amplify.aws/*⟩ - biblioteca responsável por integrar com serviços da AWS hospedado em cloud, para utilização do AWS Cognito ⟨https://aws.amazon.com/pt/cognito/⟩ como serviço de autenticação e gerenciamento dos usuários;
- *Full Calendar* ⟨*https://fullcalendar.io/*⟩ - biblioteca para auxiliar na visualização dos eventos em calendário, sob a licença GPLv3 ⟨https://www.gnu.org/licenses/gpl-3.0.en.html⟩ para utilização das funcionalidades premium;

6.3 Plataforma de hardware

Para a hospedagem do projeto, usou-se o InterNuvem ⟨https://internuvem.usp.br/⟩, serviço fornecido pela Superintendência de Tecnologia da Informação da USP para criação de máquinas virtuais. Foi feita uma solicitação e obtidos 2 vCPU 2.3Ghz, 4GB vRAM, 100GB HDD, HA, 1 IP público como recursos.

6.4 Banco de dados

Para escolha do banco de dados primeiramente foi preciso definir entre a utilização de um modelo banco de dados relacional ou não relacional. Modelos de banco de dados relacionais dependem da definição de esquemas que definem a estrutura de suas tabelas antes de poder armazenar informações, enquanto que modelos não relacionais não dependem dessa pré definição, fazendo com que esses últimos sejam mais flexíveis em relação à alteração do formato e estrutura dos dados armazenados caso alguma mudança seja necessária a qualquer ponto do processo de desenvolvimento (GYORÖDI; GYORÖDI; SOTOC, 2015). Tendo em vista o objetivo de que o sistema possa ser adaptado para atender novos cenários no futuro, por exemplo sendo adaptado para outros institutos da faculdade, que podem possuir cada um alguma realidade diferente quanto aos tipos de informações relevantes a serem armazenadas, optou-se pela utilização de um modelo de base de dados não relacional visto essa flexibilidade de alteração dos esquemas das bases armazenadas.

Quanto ao tipo de base não relacional escolhido, optou-se pela utilização de um modelo orientado a documentos, uma vez que outros formatos (chave-valor, orientado a colunas e orientado a grafo (GYORÖDI; GYORÖDI; SOTOC, 2015) (JATANA et al., 2012)) não trazem grandes vantagens em relação à forma de armazenamento dos dados para o problema em questão.

Finalmente, quanto às opções disponíveis no mercado, priorizou-se a escolha de uma solução que seja *open-source* e que tenha uma ampla base de utilização, para facilitar que outros desenvolvedores possam editar o sistema no futuro.

Dessa forma escolheu-se o MongoDB como solução, uma das soluções de banco de dados não relacional orientado a documentos historicamente mais utilizada no mercado (GYORÖDI; GYORÖDI; SOTOC, 2015), que promove uma flexibilidade maior na edição dos esquemas das bases conforme necessário, facilitando o desenvolvimento contínuo seguindo o *Scrum*, e eventuais atualizações do sistema que venham a ser necessárias.

6.5 Solucionador do problema de otimização

Dentro do contexto de resolução do PAAS, foram testadas duas opções de solucionadores de problemas de Programação Linear Inteira já implementados: o *Coin-or Branch and Cut* (CBC) (FORREST et al., 2022), um solucionador *open-source* e de uso gratuito, e o

solucionador *Gurobi* (Gurobi Optimization, LLC, 2022), uma solução comercial para problemas de otimização. A interface do sistema com ambos solucionadores foi feita através do pacote PuLP (DUNNING S. MITCHELL, 2011), um modelador de Programação Linear para a linguagem Python que providencia uma camada de abstração para diferentes tipos de solucionadores, incluindo CBC e Gurobi, permitindo que ambas opções fossem testadas em questão de performance a partir de uma mesma base de código.

Como base de comparação, levou-se em conta principalmente a performance na execução da resolução do PAAS (em termos de tempo e capacidade de execução). Nesse sentido, ambos solucionadores foram expostos a 5 cenários de teste:

- **Cenário 1**: conjunto pequeno de salas de aula (5) e aulas (5), com solução possível.
- **Cenário 2**: conjunto pequeno de salas de aula (5) e aulas (5), com solução impossível (i.e. não há configuração possível para a alocação das aulas nas salas disponíveis).
- **Cenário 3**: conjunto grande de salas de aula (25) e aulas (150), com solução possível.
- **Cenário 4**: conjunto grande de salas de aula (25) e aulas (150), com solução impossível.
- **Cenário 5**: conjunto grande de salas de aula (25) e aulas (150), com não obrigatoriedade de alocação de algumas aulas (formulação alternativa do PAAS, capítulo 7.3.1)).

Os cenários 1 e 2 foram criados a partir de dados fictícios de aulas (dias, horários, número de alunos e requisitos de recursos) e salas (lotação e recursos disponíveis), com o intuito de se testar em menor escala a capacidade de solução de cada um dos solucionadores. Já os cenários 3 e 4 foram criados a partir de dados reais de aulas e salas, obtidos a partir da base de dados do sistema USPolis anterior, apenas modificando-se os dados de aulas no cenário 4 de forma a se criar uma configuração de solução impossível, com o intuito de se testar a performance em situações mais próximas à realidade. O cenário 5 utilizou o mesmo conjunto de dados do cenário 4, mas foi testado com a formulação alternativa de não obrigatoriedade de alocação de um dado conjunto de aulas (descrito no capítulo 7.3.1), formulação que visa prover uma configuração de alocação mesmo em casos de solução impossível, removendo a necessidade de alocação de um dado conjunto de aulas menos prioritárias. A Tabela 2 apresenta um resumo dos resultados obtidos.

Tabela 2: Resultados de teste de solucionadores na resolução do PAAS (Resultado obtido/Tempo de execução)

	CBC	Gurobi
Cenário 1	Conclusão esperada / <0.1s	Conclusão esperada / <0.1s
Cenário 2	Conclusão esperada / <0.1s	Conclusão esperada / <0.1s
Cenário 3	Conclusão esperada / ∼30 s	Conclusão esperada / ∼0.5 s
Cenário 4	Sem conclusão / >10 h	Conclusão esperada / ∼0.5 s
Cenário 5	Sem conclusão / >10 h	Conclusão esperada / ∼0.5 s

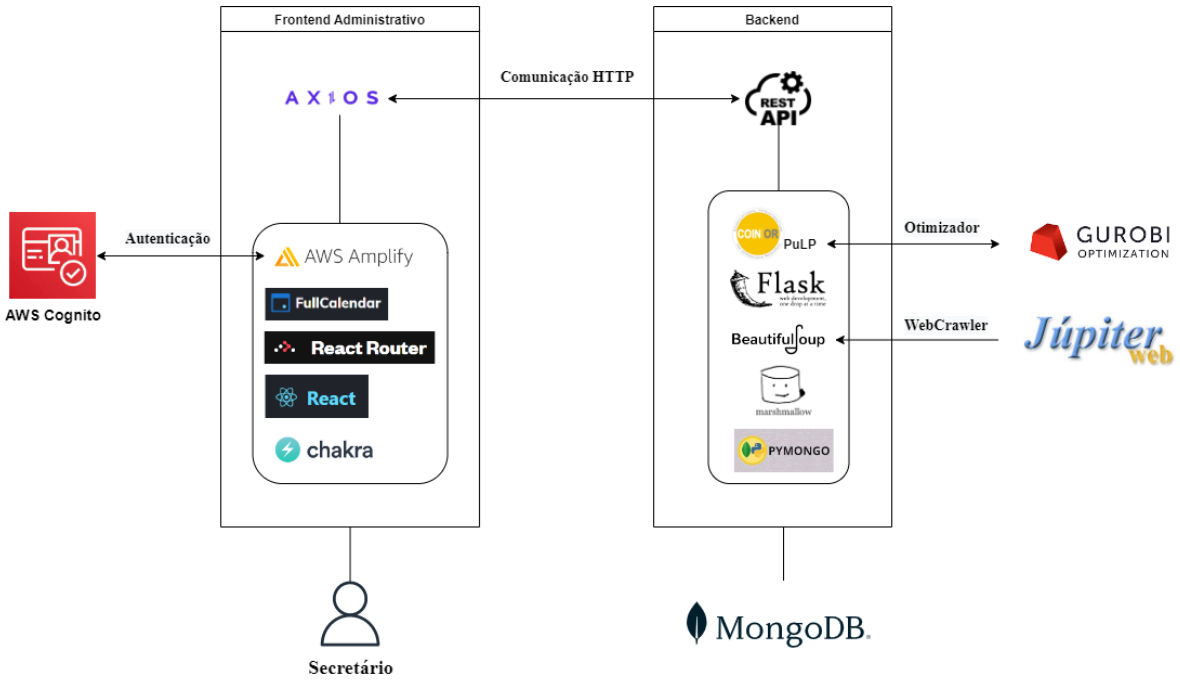
Ambos solucionadores obtiveram os resultados corretos e esperados nos cenários 1 e 2 (ou seja, alocação atendendo às restrições e alocação impossível, respectivamente), em tempos na ordem de mili-segundos. No cenário 3, ambos também chegaram à alocação correta, porém com tempos de execução bem diferentes, com o CBC demorando aproximadamente 100 vezes mais que o Gurobi (mas ainda em uma ordem de grandeza aceitável para a aplicação proposta). Contudo, nos cenários 4 e 5, apenas o Gurobi foi capaz de obter a conclusão correta (alocação impossível), enquanto o CBC não obteve conclusão sobre a possibilidade ou não da alocação em mais de 10 horas de execução - fato que demonstra uma grande limitação do CBC para a aplicação proposta, uma vez que o sistema precisa ser capaz de lidar de alguma forma com casos onde não há configurações de alocação possível (seja alertando o usuário que não há configuração possível, seja providenciando uma alocação sub-ótima).

Com esses resultados em mente, optou-se pela utilização do Gurobi como tecnologia de solucionador do sistema para o PAAS.

6.6 Considerações do capítulo

Nesse capítulo foram discutidas as tecnologias do sistema, levantando as principais características de cada uma, consideradas nas escolhas. A arquitetura final do sistema foi representada no diagrama da figura 10, onde são mostrados os 2 módulos principais do escopo definido para este projeto. Com a arquitetura definida, iniciou-se o desenvolvimento das funcionalidades do sistema, descritas no capítulo a seguir.

Figura 10: Visão geral da arquitetura do sistema



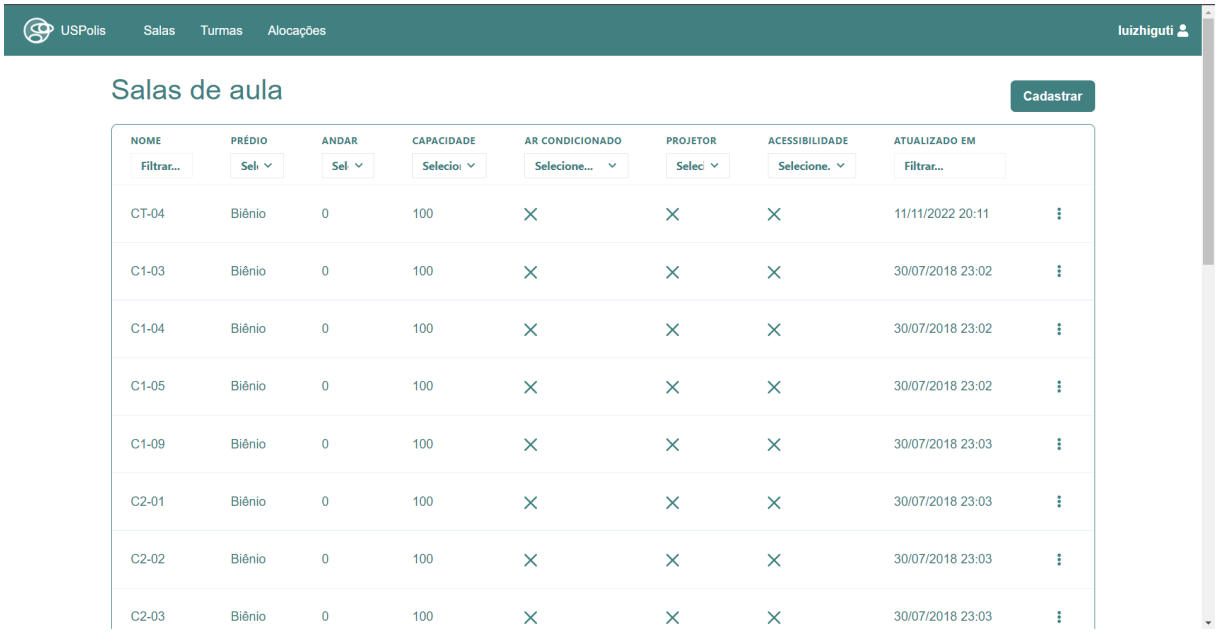
7 FUNCIONALIDADES DO SISTEMA

Nesse capítulo são apresentadas as funcionalidades presentes no sistema desenvolvido, definidas pelo MVP planejado e descrito anteriormente. É feita uma descrição detalhada das visualizações e uma demonstração dos principais fluxos do processo de alocação realizados através do sistema.

7.1 Gestão de salas

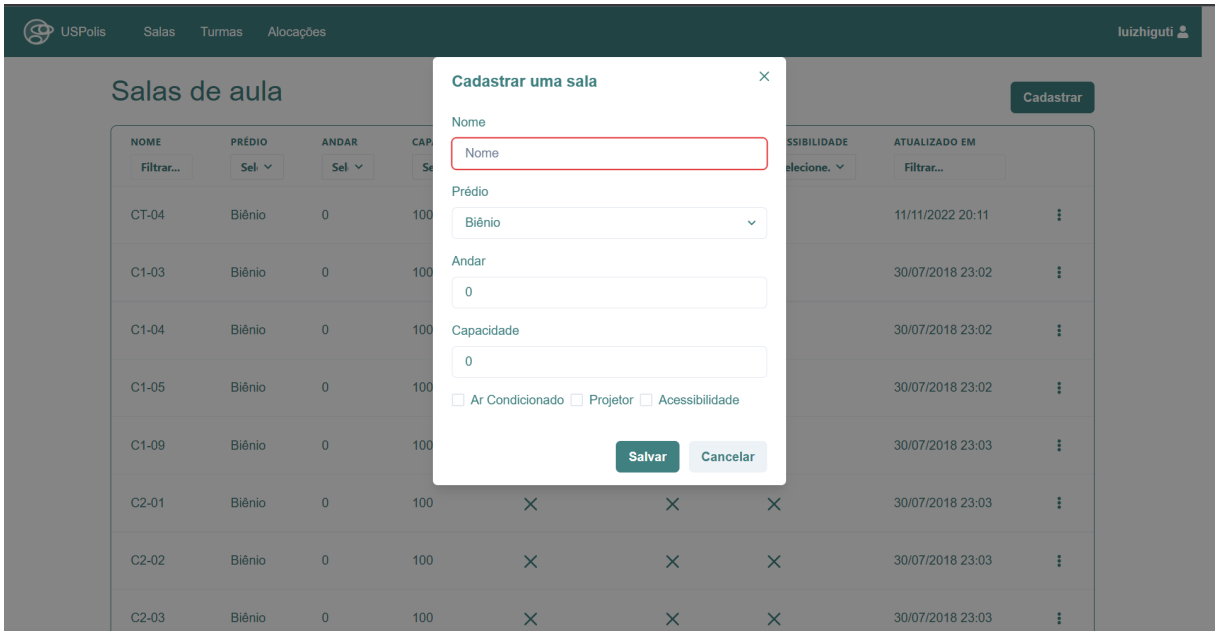
O gerenciamento das salas é realizado através da página de salas - Figura 11 - onde são mostradas as salas cadastradas pelo usuário. A tabela de salas contém o nome da sala na coluna “Nome”, o prédio em que a sala se encontra em “Prédio”, o andar da sala na coluna “Andar”, a capacidade total da sala em “Capacidade”, se a sala possui ar condicionado, projetor e acessibilidade nas colunas com os respectivos títulos e por fim a coluna “Atualizado em” indica o dia e horário que foi feita a última atualização de alguma informação da sala.

Figura 11: Visualização de salas



O cadastro de uma nova sala é feito por meio de um formulário na própria página, aberto através do botão *"Cadastrar"* - Figura 12.

Figura 12: Cadastro de salas



A edição e remoção de uma sala pode ser feita pelo menu de opções (⋮) - Figura 13 - o *"Editar"* e o *"Deletar"* abrem respectivamente um formulário para edição de uma sala

- Figura 14 - e um diálogo de confirmação - Figura 15.

Figura 13: Menu de opções de uma sala

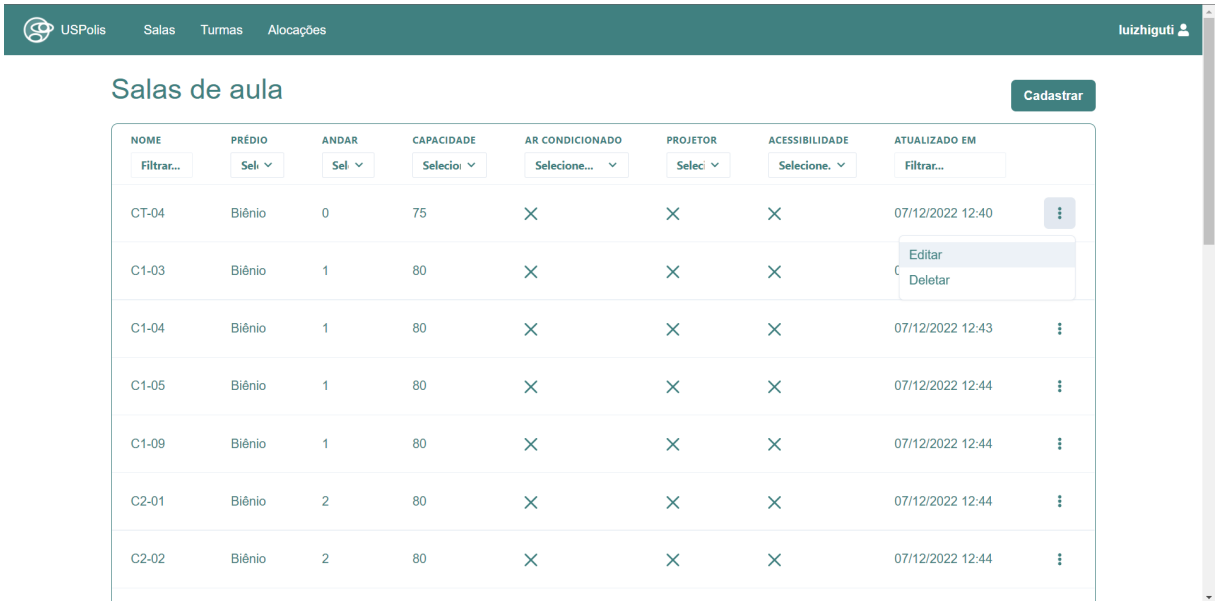


Figura 14: Edição de uma sala

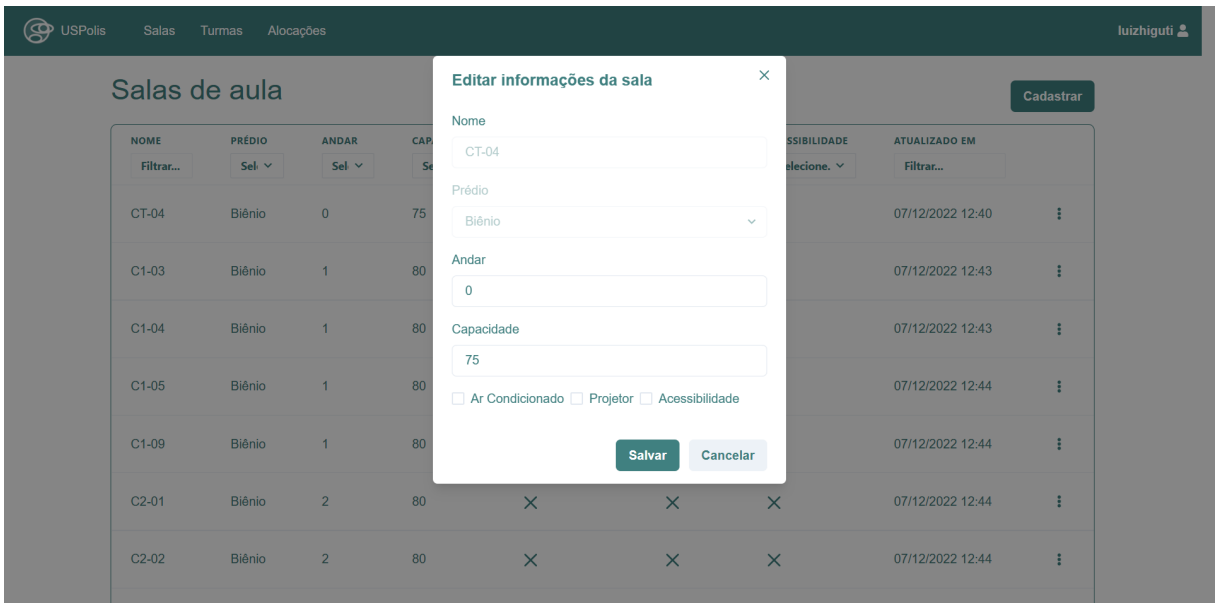
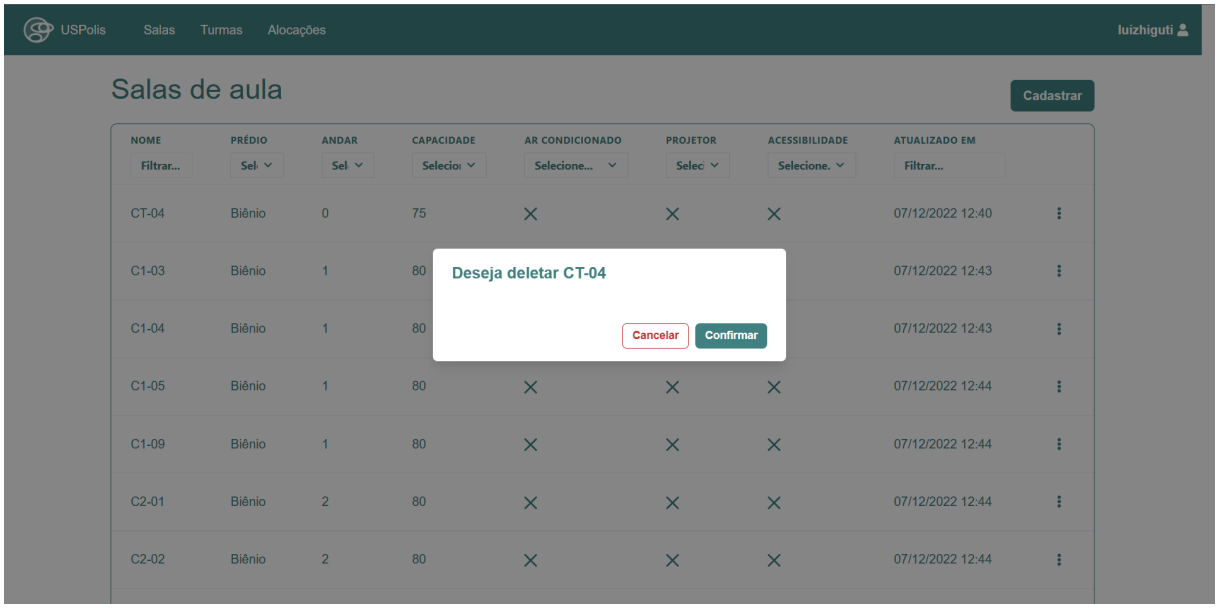
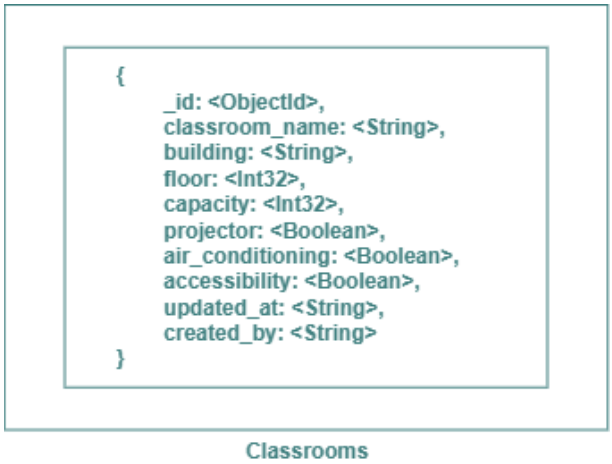


Figura 15: Confirmar deleção de uma sala



As informações das salas são salvas na base de dados na coleção, análogo a tabelas de bases de dados relacionais (MongoDB, Inc., 2022), *classrooms*, apresentado na Figura 16, onde cada documento guarda as informações de uma sala.

Figura 16: Coleção que guarda informações das salas



7.2 Gestão de turmas e eventos

O gerenciamento das turmas é realizado pela página de turmas - Figura 17. A tabela de turmas mostra o código da disciplina na coluna “Código”, o nome da disciplina em “Disciplina”, o código da turma na coluna “Turma”. O código é composto por 7 dígitos,

em que os 4 primeiros representam o ano, o 5º dígito o semestre do ano (1 ou 2) e os 2 últimos correspondem ao número da turma. Para cada horário da turma, a coluna "Horários" indica o dia da semana, horário de início e fim e os nomes dos professores em "Professores".

Figura 17: Visualização de turmas

A partir do menu de opções (⋮) - Figura 18 - é possível editar as informações - Figura 19 - e as preferências de uma turma - figura 20 - além de remover uma turma, analogamente às salas.

Figura 18: Menu de opções de uma turma

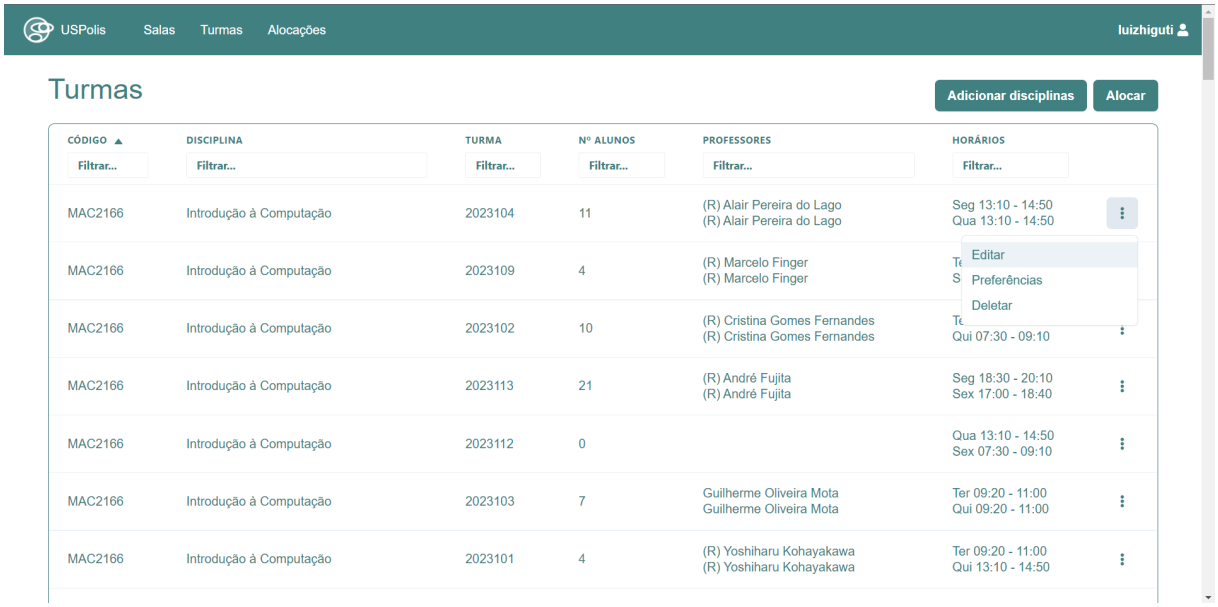


Figura 19: Editar informações de uma turma

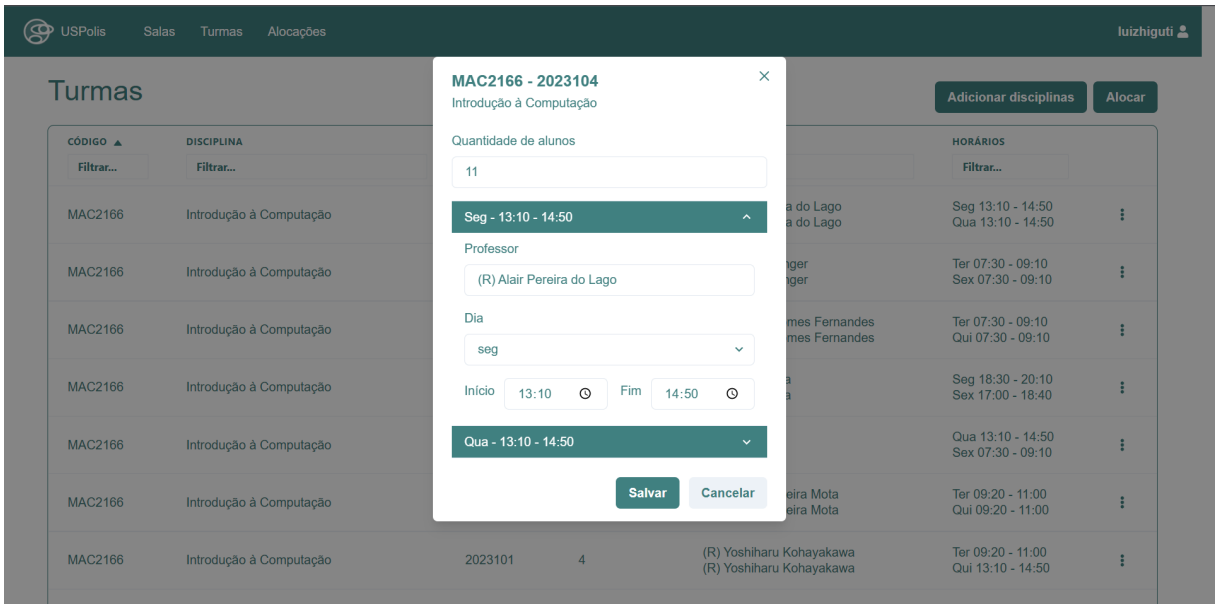
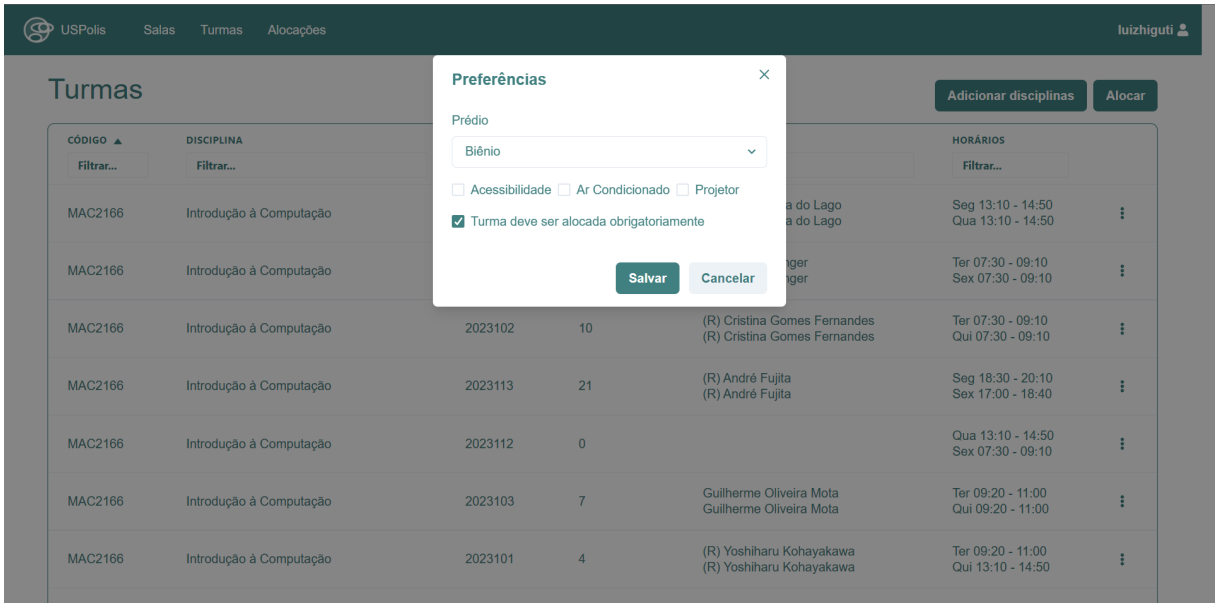
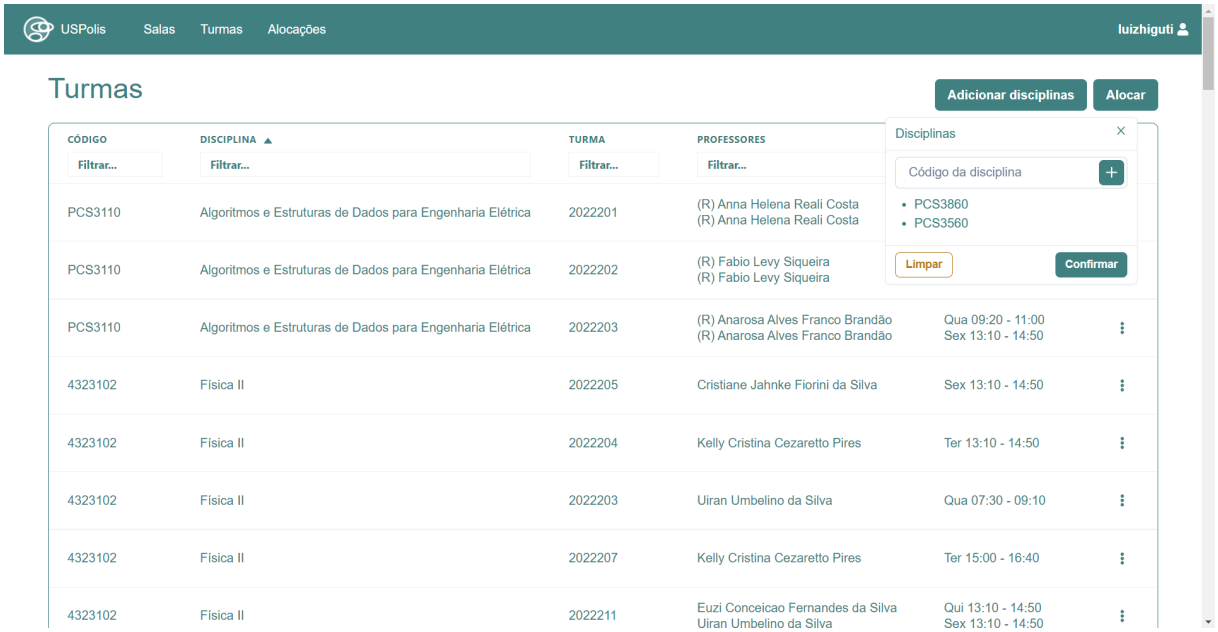


Figura 20: Editar preferências de uma turma



O cadastro das informações das turmas é realizado pelo menu em *"Adicionar disciplinas"* - Figura 21.

Figura 21: Cadastrar turmas



As turmas são buscadas no Júpiter Web pelo código das disciplinas através do Web Crawler desenvolvido e mapeadas em eventos, em que cada horário de uma turma corresponde a um evento. As informações são salvas como eventos na coleção *events* do banco

de dados, a qual é exemplificada na Figura 22.

Figura 22: Coleção que guarda informações dos eventos

7.3 Alocação de salas

A alocação de salas é calculada pelo sistema de forma automática por meio da resolução da formulação matemática para o problema da alocação de salas.

7.3.1 Formulação alternativa do PAAS: não obrigatoriedade de alocação

Durante o desenvolvimento do sistema percebeu-se a possibilidade de haver configurações de aulas e salas que não possuem alocação possível durante o semestre letivo, uma vez que em determinado semestre podem haver um número de aulas em um mesmo horário e dia específicos superior à quantidade salas existentes. Nessa situação, os decisores locais informaram que historicamente esse problema é resolvido retirando um conjunto pré definido de aulas, marcadas como sendo de baixa prioridade, para serem e realocadas em outros prédios, onde a alocação dessas fica sob responsabilidade dos decisores locais de cada prédio.

Dado o contexto, foi proposta uma formulação multi-objetivo alternativa à descrita no capítulo 2.3 de forma a se lidar com essa situação:

Conjuntos e dados de Preprocessamento

T	Conjunto de turmas ($t \in \{1, ..., \|T\|\}$)
S	Conjunto de salas ($s \in \{1, ..., \|S\|\}$)
R	Conjunto de recursos ($r \in \{1, ..., \|R\|\}$)
A	Conjunto de aulas ($a \in \{1, ..., \|A\|\}$)
A^t	Conjunto das aulas $a \in A$ da turma $t \in T$
A_{opc}	Conjunto das aulas $a \in A$ cuja alocação não é obrigatória
η	$\eta_{as} = 1$ se a sala $s \in S$ pode alocar a aula $a \in A$, 0 c.c.
Θ	$\Theta_{aa'} = 1$ se as aulas $a, a' \in A$ possuem sobreposição de dia e horário, 0 c.c.

Variáveis de decisão

x_{as}	$x_{as} = 1$ se a aula $a \in A$ é alocada na sala $s \in S$, 0 c.c.
y_t	$y_t =$ número de trocas de sala da turma $t \in T$

$$\min_{x,y} \quad F = \alpha \left(\sum_{t \in T} y_t \right) + \beta \left(\sum_{s \in S} \sum_{a \in A_{opc}} (1 - x_{as}) \right) \tag{7.1a}$$

$$\text{sujeito a:} \quad \sum_{s \in S} x_{as} = 1 \quad a \in A \setminus A_{opc} \tag{7.1b}$$

$$\sum_{s \in S} x_{as} \leq 1 \quad a \in A_{opc} \tag{7.1c}$$

$$x_{as} \leq \eta_{as} \quad s \in S \text{ e } a \in A \tag{7.1d}$$

$$x_{as} + x_{a's} \leq 1 \quad s \in S \text{ e } a, a' \in A | \Theta_{aa'} = 1 \tag{7.1e}$$

$$\sum_{n=1}^{|A^t|} x_{a_n s_n} \leq 1 + y_t \quad t \in T, a_n \in A^t \text{ e } \{s_1, ..., s_{|A^t|}\} \subset S \tag{7.1f}$$

$$x_{as} \in 0, 1 \quad a \in A, s \in S \tag{7.1g}$$

$$y_t \in \mathbb{Z}_+ \quad t \in T \tag{7.1h}$$

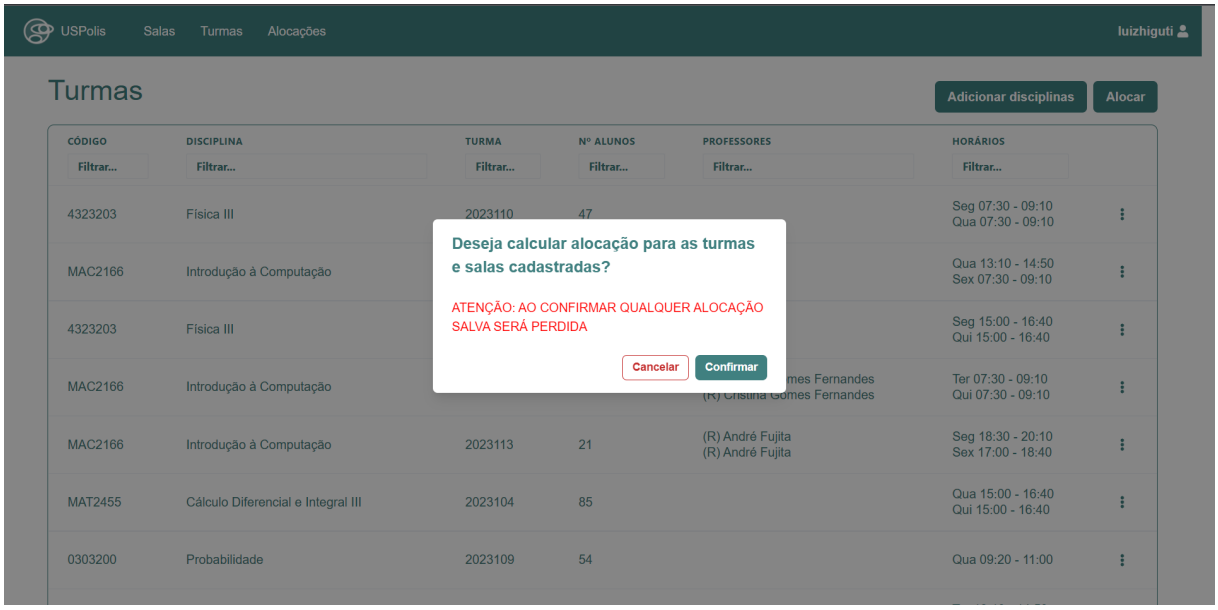
As principais mudanças em relação à formulação original (Capítulo 2.3) se encontram na adição de um conjunto pré-definido A_{opc} de aulas cuja alocação não é obrigatória no problema, na função objetivo (7.1a), e no conjunto de restrições (7.1b) e (7.1c). As restrições (7.1b) e (7.1c) foram modificadas de forma que, aulas pertencentes ao conjunto A_{opc} podem ser alocadas em uma ou zero salas, enquanto as aulas fora do conjunto continuam sendo necessariamente alocadas em uma e uma única sala. A função objetivo

(7.1a) é composta agora por uma combinação linear (mantendo-se portanto na classe de problemas de Programação Linear) de duas métricas de qualidade, a de estabilidade de turmas e uma métrica adicional que visa minimizar o número de aulas opcionais não alocadas, cada uma multiplicada por um fator de importância relativa α e β. Sendo um problema multi-objetivo, é necessário estabelecer como definir os valores para os fatores α e β de importância de cada objetivo. No contexto do sistema desenvolvido, foi proposta a utilização de valores t.q $100 * \alpha = \beta$, uma vez que nessa configuração a função objetivo priorizaria a alocação de aulas opcionais acima da estabilidade de turmas, e seria necessário que houvessem 100 trocas de sala de turmas para equivaler à importância de alocação de uma aula opcional (situação que não deveria ocorrer na prática).

7.4 Gestão de alocações

A alocação das turmas é gerada ao clicar no botão *"Alocar"* na página de turmas - Figura 17. Os dados da alocação são salvos na coleção *events* - figura 22 - indicando a sala e o prédio designados para cada evento nos campos *"classroom"* e *"building"*. Ao tentar calcular uma alocação, as informações de uma alocação anterior, sala e prédio dos eventos, são excluídas, notificado no diálogo de confirmação exibido com a ação de alocar - Figura 23.

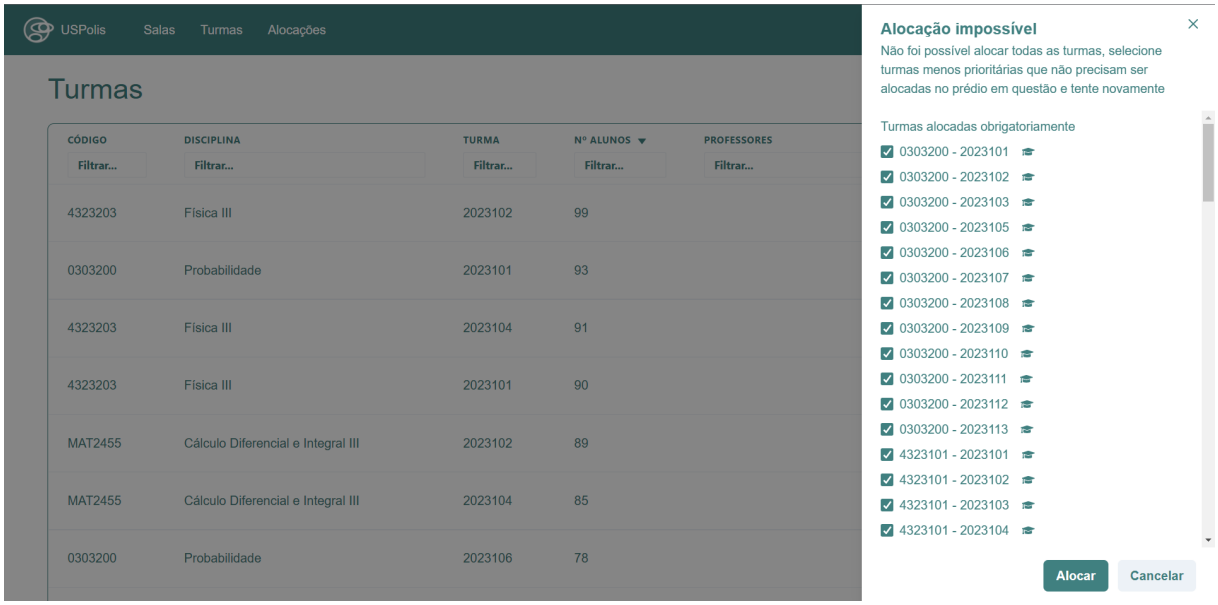
Figura 23: Diálogo para confirmar alocação



7.4.1 Solução de alocação não encontrada

O algoritmo de alocação pode não encontrar uma solução para as turmas e salas cadastradas por dois motivos: indisponibilidade de horário ou falta de salas com capacidade para a quantidade de alunos de alguma turma. Em ambos os casos, o sistema apresenta um menu lateral - Figura 24 - onde é possível editar a obrigatoriedade das turmas a serem alocadas, com finalidade de tentar achar uma configuração de turmas obrigatórias alocadas e não obrigatórias, não alocadas, e contornar a indisponibilidade de horários. Para o segundo caso, deve ser alterada a quantidade de alunos nas turmas - Figura 19 -, de modo que as salas sejam capazes de receber as turmas.

Figura 24: Menu para edição da obrigatoriedade das turmas



7.4.2 Solução de alocação encontrada

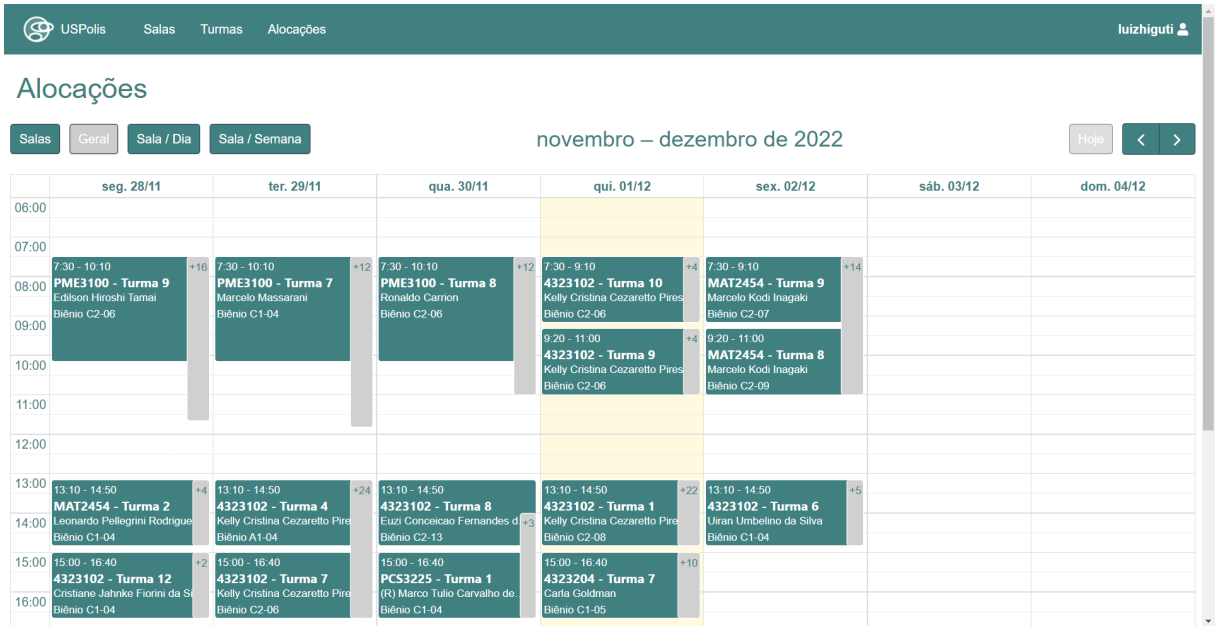
Com uma alocação encontrada, a parte do sistema responsável passa a ser a a página de alocações - Figura 25.

Figura 25: Alocações por salas

Nessa visualização os eventos são divididos pela sala a qual foram alocados. Há outras 3 visualizações disponíveis, acessadas através dos botões na parte superior esquerda: *"Geral"*, *"Sala/Dia"* e *"Sala/Semana"*. Os botões presentes na direita são utilizados para navegar no calendário, e alterar o período para visualizar os eventos agendados.

A visualização Geral - Figura 26 - mostra os eventos que ocorrem na semana, indicando o dia e hora de cada evento.

Figura 26: Visualização da alocação Geral



As visualizações Sala/Dia - Figura 27 - e Sala/Semana - figura 28 - são similares, com a diferença no período (dia ou semana) de visualização.

Figura 27: Visualização da alocação por salas no dia

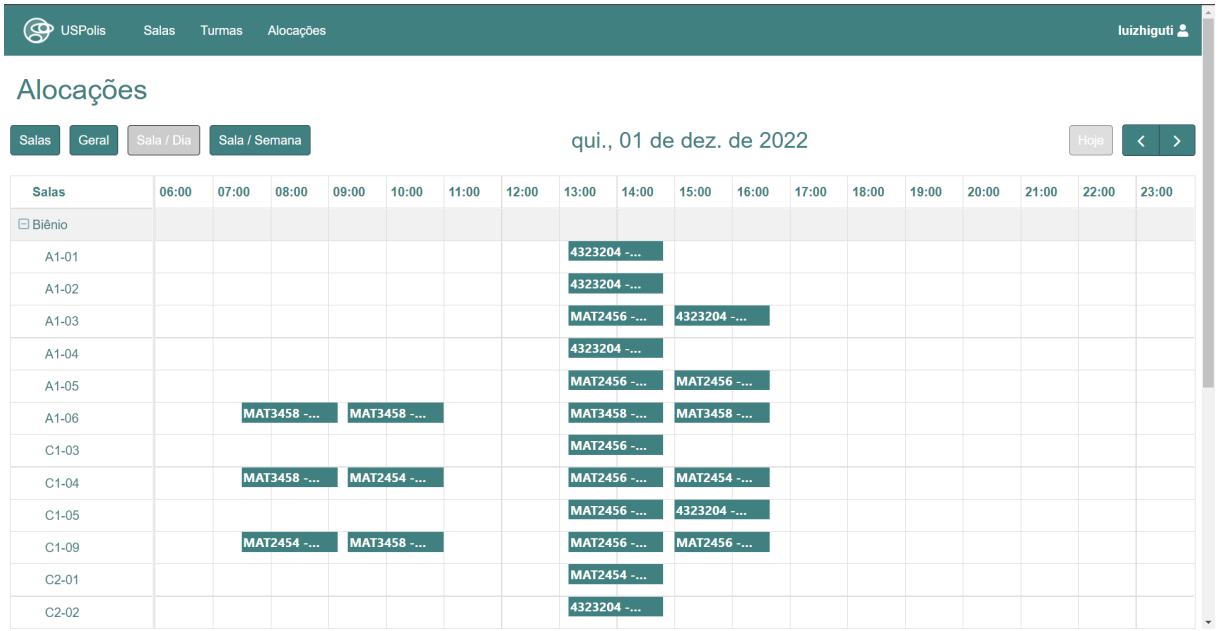
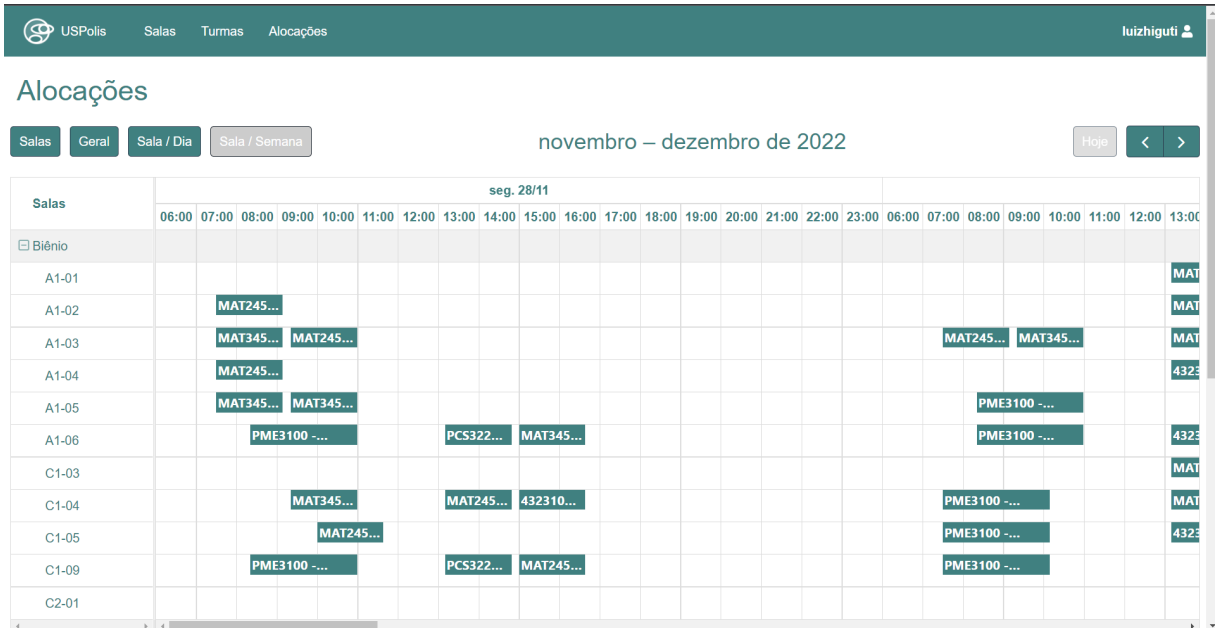


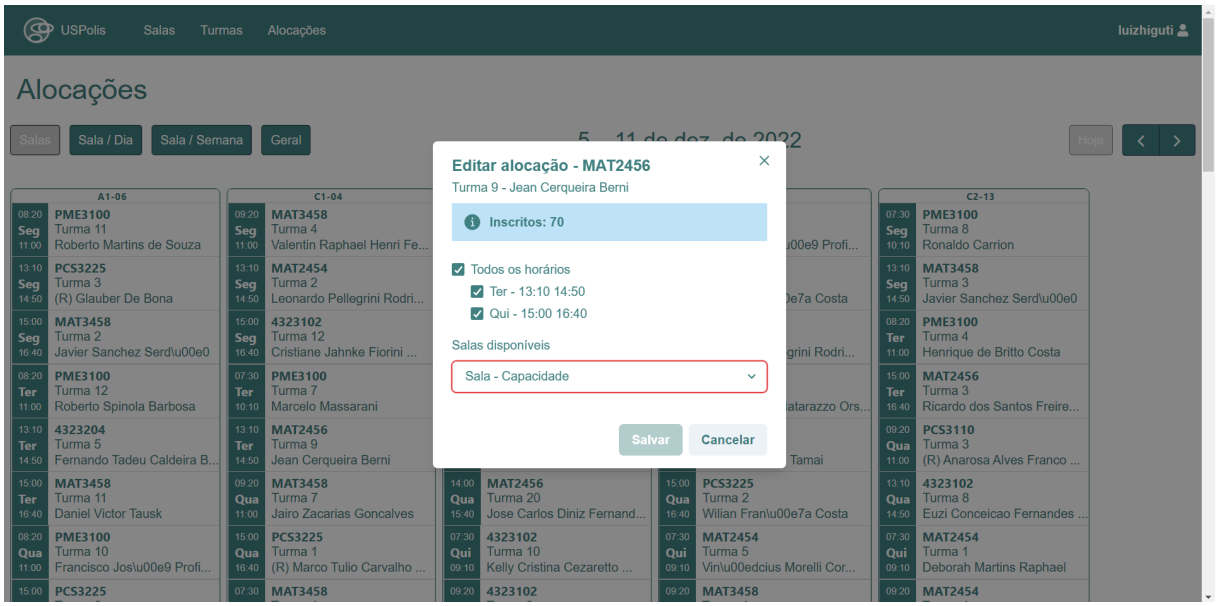
Figura 28: Visualização da alocação por salas na semana



Nessas visualizações os eventos são separados por sala na vertical e cronologicamente na horizontal, o que facilita realizar uma validação visual de conflitos de horário entre eventos, além de observar os horários em que as salas estão disponíveis.

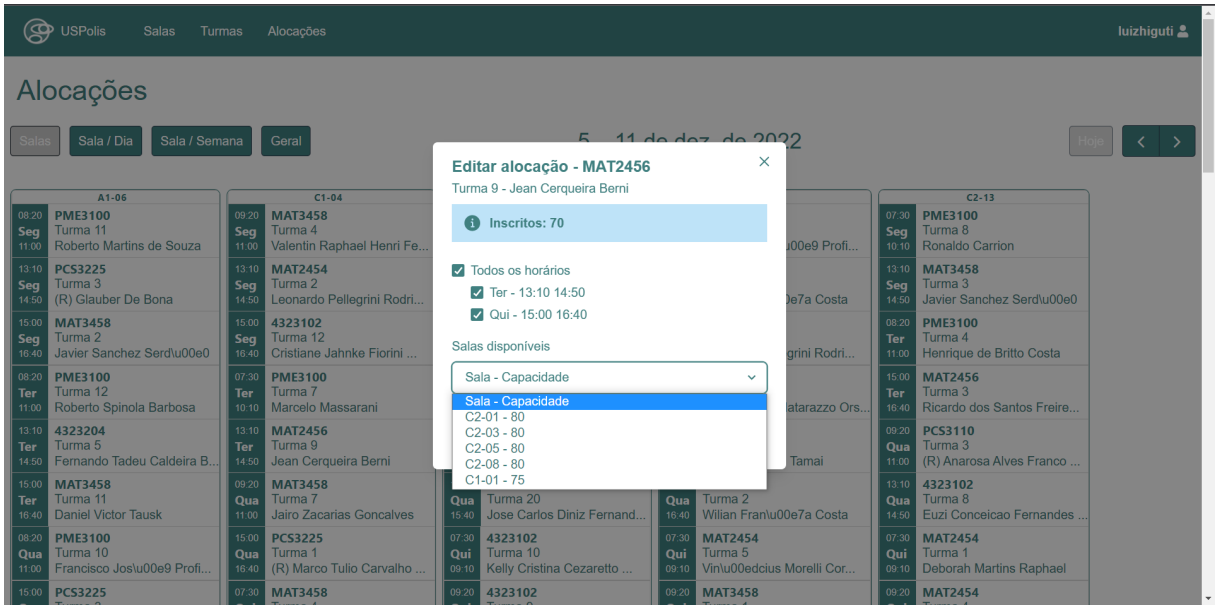
Além da solução calculada, um dos requisitos do sistema é a possibilidade de editar a alocação. A edição é feita na visualização por salas - Figura 25 - selecionando um evento pelo formulário de edição - figura 29.

Figura 29: Editar alocação



O formulário mostra todos os horários da turma do evento selecionado e as salas disponíveis nos horários dos eventos selecionados - Figura 30.

Figura 30: Salas disponíveis



As salas disponíveis mostradas são relativas aos horários selecionados, na figura 30 são referentes ao dois horários selecionados, disponíveis tanto para o primeiro (terça) - Figura 31 - quanto para o segundo (quinta) - figura 32 - horário.

Figura 31: Salas disponíveis na terça

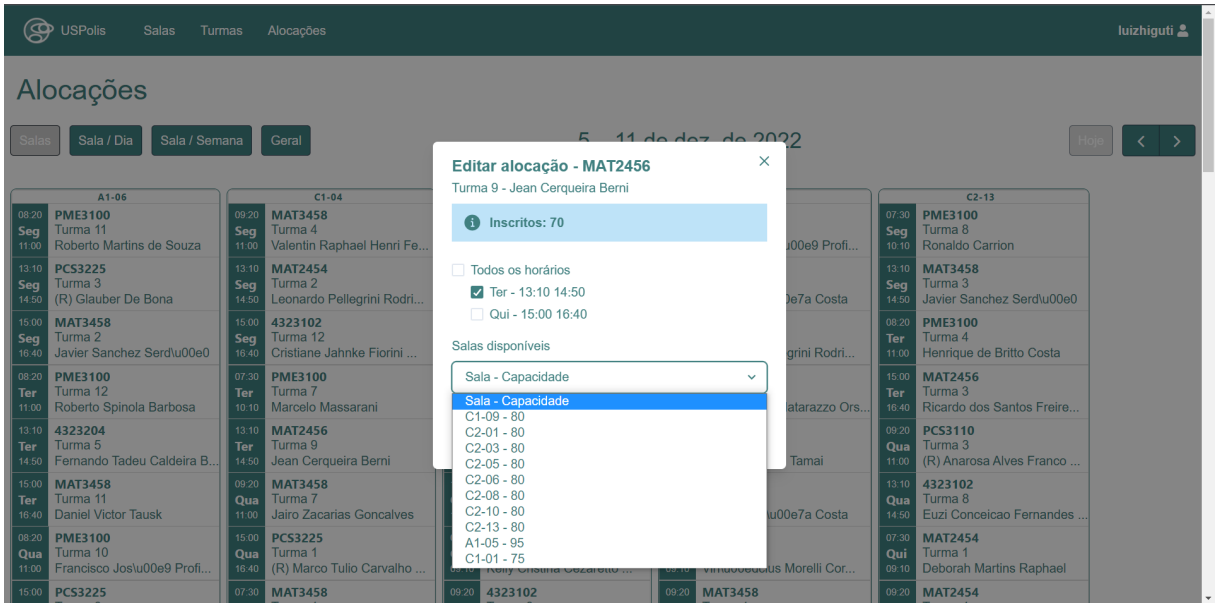
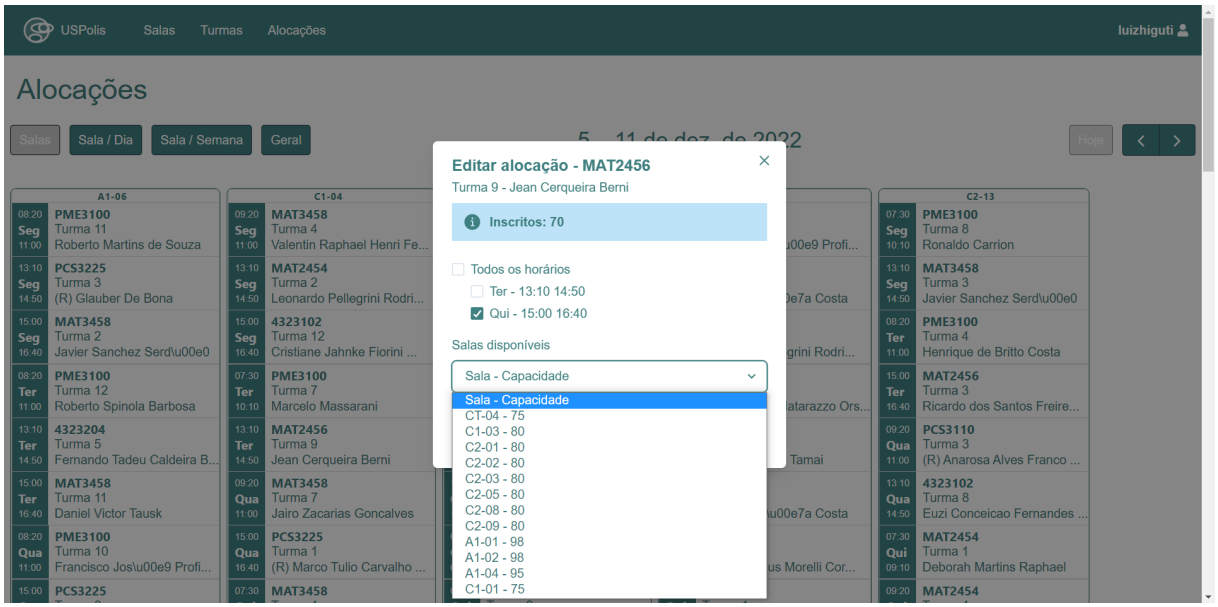


Figura 32: Salas disponíveis na quinta



A sala selecionada pode ter capacidade inferior ao número de alunos na turma. Nesse caso é indicado um alerta - figura 33. A possibilidade de realizar esse tipo de edição

é importante para casos especiais, que envolvem expertise da pessoa responsável pela alocação. Caso a capacidade da sala seja superior, o formulário apresenta um alerta positivo - Figura 34.

Figura 33: Sala selecionada com capacidade inferior

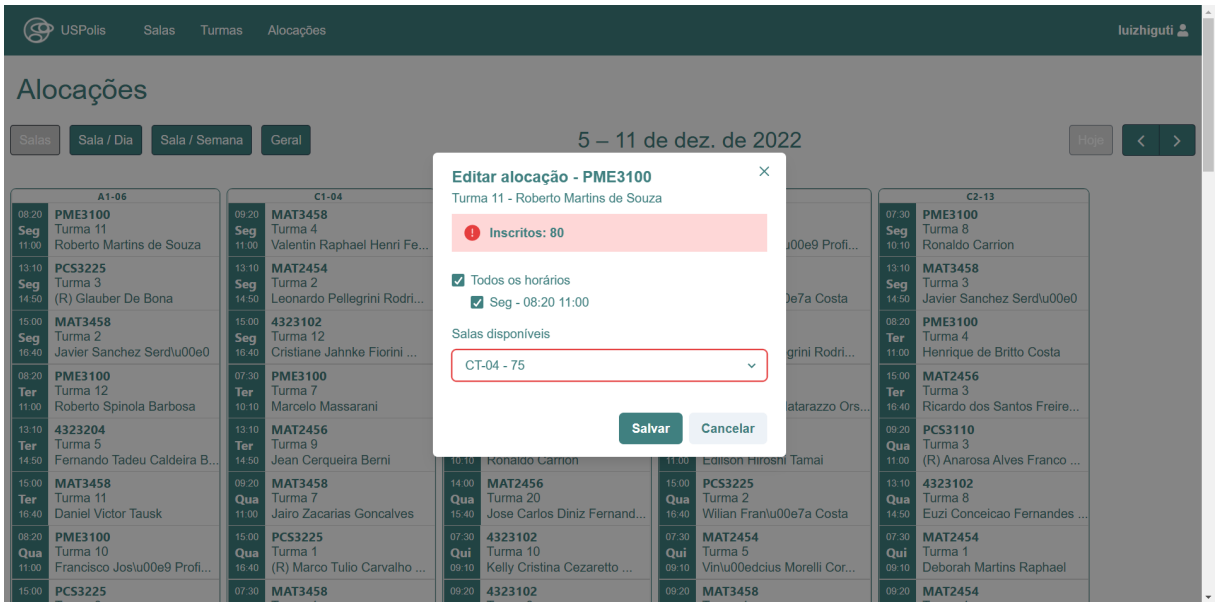


Figura 34: Sala selecionada com capacidade superior

7.5 Considerações do capítulo

Nesse capítulo foram apresentadas as principais funcionalidades do sistema desenvolvido, considerando os principais requisitos levantados e definidos para um Produto Viável Mínimo (MVP) que possa ser utilizado dentro do escopo definido para o projeto trazendo melhorias ao processo de alocação de turmas.

Algumas funcionalidades foram adicionadas e/ou modificadas durante o desenvolvimento a partir de feedbacks coletados em validações feitas com usuários do sistema, que serão discutidas na próxima seção.

8 RESULTADOS OBTIDOS

Este capítulo visa analisar os principais resultados obtidos ao longo do desenvolvimento do trabalho, a partir de dois ângulos: da qualidade e usabilidade do sistema para o usuário, as quais foram medidas através de validações de interface, e da performance da alocação obtida em comparação ao processo manual realizado normalmente.

8.1 Validações de Interface

Foram realizadas validações intermediárias de interface para cada uma das três interfaces de usuário desenvolvidas no sistema: de Salas, de Aulas e de Alocação. Uma vez o desenvolvimento definido como terminado, possuindo todas as funcionalidades principais (definidas no Capitulo 5, e detalhadas no Capítulo 7, envolvendo os processos de gestão de informações de salas, importação e edição de informação de aulas, alocação de salas e edição de alocação), realizou-se também uma validação final do sistema como um todo, simulando um cenário de utilização real de alocação de salas, com os dados de aulas do período do primeiro semestre de 2023. Todas as validações foram realizadas com o decisor local do prédio do Biênio da EPUSP, local onde foi prevista a primeira implantação em produção do sistema USPolis a partir do ano de 2023, com um caráter de avaliação qualitativa, medindo-se usabilidade e intuitividade do processo.

8.1.1 Validações intermediárias

As validações intermediárias foram realizadas com a finalização do desenvolvimento de cada uma das telas de funcionalidades. Elas ocorreram em duas seções, a primeira para a validação das telas de Salas e Aulas, e a segunda para a tela de Alocação, com durações de aproximadamente 1h30 e 1h respectivamente. Além de averiguar se o usuário era capaz de realizar o processo associado a cada uma das telas, também averiguou-se:

1. Se todas as informações esperadas estavam presentes;

2. Se a finalidade de elementos de interface (textos, botões, áreas de texto) era clara;
3. O design das telas era visualmente agradável;
4. Se o formato de apresentação das informações atendia às expectativas do usuário;

Os resultados obtidos em cada validação são apresentados a seguir:

- Tela de Salas: Foram validados os processos de cadastro de salas e edição de informações de salas. Ambos foram julgados como intuitivos e de fácil execução.
- Tela de Turmas: Foram validados os processos de importação de disciplinas do JupiterWeb, edição de informações de turmas, e edição de preferências de turmas. Todos foram julgados como intuitivos e de fácil execução.
- Tela de Alocação: Foram validados os processos de alocação e de edição de alocação. Ambos foram considerados como intuitivos e de fácil execução.

8.1.2 Validação final do sistema

Como validação final foi feita uma simulação do período de alocação de salas do semestre seguinte, em uma seção de aproximadamente 2 horas. Para tanto utilizou-se dos dados reais de salas do prédio do Biênio e de turmas das disciplinas do semestre seguinte (primeiro semestre de 2023), e a tarefa a ser executada pelo decisor local foi de realizar a alocação das aulas às salas, visualizá-la e julgar se ela estaria pronta para uso.

A tarefa foi apresentada ao decisor local, e propôs-se uma tentativa de execução da mesma ao máximo sem intervenção do time de desenvolvimento, que ficaria encarregado de observar o desenrolar da execução, e anotar pontos de atenção.

A execução da tarefa ocorreu sem grandes influências do time de desenvolvimento, com apenas a percepção de algumas melhorias pontuais, que foram corrigidas ou implementadas no local, sendo validadas logo em seguida. O sistema foi portanto considerado validado.

9 CONSIDERAÇÕES FINAIS

Este trabalho propôs o desenvolvimento do USPolis, um sistema *open source* de alocação e visualização de salas de aula para a comunidade da EPUSP, baseando-se no projeto USPolis anterior e aprimorando-o. O projeto consistiu na concepção e implementação de um sistema de alocação de salas de aula, voltado para as secretarias dos prédios acadêmicos da EPUSP, com o intuito de resolver o problema de distribuição das disciplinas nas salas de aula, considerando seus respectivos horários e demais restrições de cada turma, através de um sistema automatizado e de fácil utilização. Para resolver o problema de alocação de aulas às salas propôs-se uma abordagem de otimização, com uma formulação matemática da classe de Programação Linear Inteira, o que permitiu a utilização de solucionadores já implementados e otimizados para a resolução desse tipo de problema. O desenvolvimento se deu seguindo a metodologia *Scrum* e usando histórias de usuário, elicitando e refinando continuamente os requisitos de cada tipo de usuário do sistema.

O desenvolvimento do sistema proposto foi realizado de forma bem sucedida, tendo-se implementado todo o conjunto de funcionalidades e requisitos principais levantados com o usuário. Validações de implementação de funcionalidades e de usabilidade em geral do sistema foram realizadas com os usuários alvo, tendo obtido resultados positivos, e com uma confirmação de intenção de implantação e utilização no prédio do Biênio da EPUSP para a alocação de aulas do primeiro semestre de 2023.

Em questão de resultados obtidos, desenvolveu-se um sistema julgado de utilização intuitiva, que atinge o objetivo proposto em um tempo muito menor do que o processo manual utilizado anteriormente - passando de um total aproximado (informado pelo decisor local) de 27 horas não contínuas para realização da alocação de salas, para um processo que leva aproximadamente meio dia de trabalho, considerando todo o fluxo de importação de dados de aulas, alocação, edição e validação da configuração de aulas e salas obtida.

Com o foco do projeto sendo o desenvolvimento de uma versão mínima do sistema, com funcionalidades que atendessem às necessidades principais mapeadas no contexto do

prédio do Biênio (histórias de prioridade 1, capítulo 5), propõe-se para trabalhos futuros o desenvolvimento tanto de funcionalidades secundárias mapeadas (prioridades 2 e 3) e não mapeadas, que possam melhorar a usabilidade e experiência do usuário, quanto de funcionalidades visando estender o uso do sistema para outros prédios da EPUSP, da USP ou de instituições de ensino como um todo.

REFERÊNCIAS

SALES, E. d. S. e. a. *Problema de Alocação de Salas e a Otimização dos Espaços no Centro de Tecnologia da UFSM*. Dissertação (Mestrado), 2015.

RUBIN, K. S. *Essential Scrum: A practical guide to the most popular Agile process*. [S.l.]: Addison-Wesley, 2012.

WAKE, W. C. *INVEST in Good Stories, and SMART Tasks*. 2003. ⟨https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/⟩. Acesso em 2022-06-04.

COHN, M. *User stories applied: For agile software development*. [S.l.]: Addison-Wesley Professional, 2004.

IIBA. *A Guide to the Business Analysis Body of Knowledge - BABOK Guide v3*. [S.l.]: International Institute of Business Analysis, 2015.

CIRINO, R. B. Z. *Abordagens de solução para o problema de alocação de aulas a salas*. 1–13 p. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2016. Acesso em 2022-11-09.

PHILLIPS, A. E. et al. Integer programming methods for large-scale practical classroom assignment problems. *Computers Operations Research*, v. 53, p. 42–53, 2015. ISSN 0305-0548. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0305054814001956⟩.

WINSTON, W.; GOLDBERG, J. *Operations Research: Applications and Algorithms*. Thomson Brooks/Cole, 2004. 53 p. (Operations Research: Applications and Algorithms). ISBN 9780534423582. Disponível em: ⟨https://books.google.com.br/books?id=tg5DAQAAIAAJ⟩.

FORREST, J. et al. *coin-or/Cbc: Release releases/2.10.8*. Zenodo, 2022. Disponível em: ⟨https://zenodo.org/record/6522795⟩.

Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2022. Disponível em: ⟨https://www.gurobi.com⟩.

J. Beranek et. al. *Meeting Room Booking System*. 2022. Acesso em 2022-12-06. Disponível em: ⟨https://mrbs.sourceforge.io/⟩.

GYORÖDI, C.; GYORÖDI, R.; SOTOC, R. A comparative study of relational and non-relational database models in a web-based application. *International Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 6, n. 11, p. 78–83, 2015.

JATANA, N. et al. A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*, v. 1, n. 6, p. 1–5, 2012.

DUNNING S. MITCHELL, M. O. I. *PuLP: A Linear Programming Toolkit for Python*. [S.l.]: Department of Engineering Science The University of Auckland September, 2011. ⟨https://optimization-online.org/?p=11731⟩.

MongoDB, Inc. *Databases and Collections*. 2022. Acesso em 2022-12-07. Disponível em: ⟨https://www.mongodb.com/docs/manual/core/databases-and-collections/⟩.