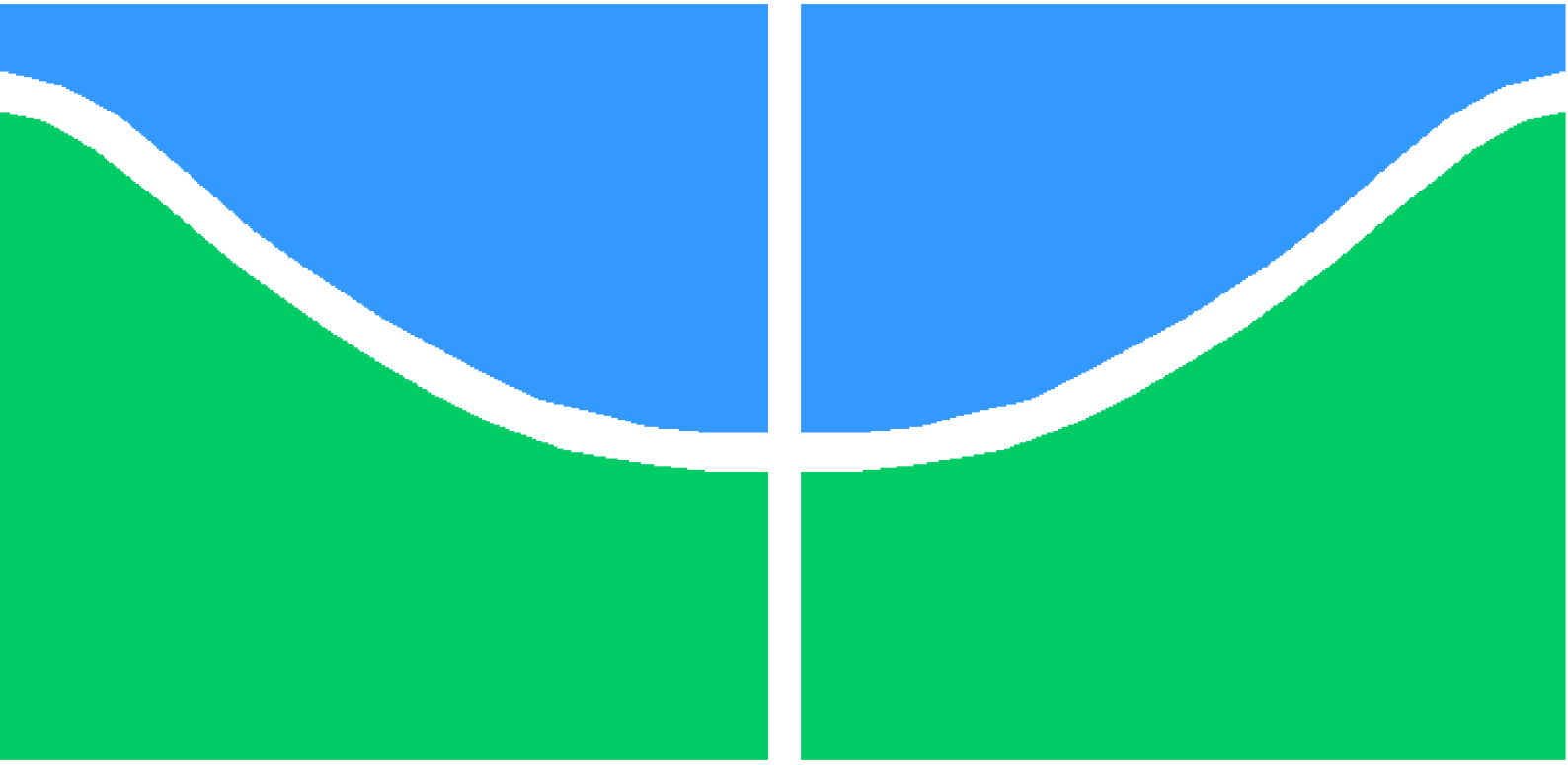


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Criação de uma aplicação para integrar algoritmos para o problema de alocação de salas de aula

Autor: Paulo Batista da Cruz Júnior

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF

2023

Paulo Batista da Cruz Júnior

Criação de uma aplicação para integrar algoritmos para o problema de alocação de salas de aula

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF

2023

Paulo Batista da Cruz Júnior
Criação de uma aplicação para integrar algoritmos para o problema de alocação de salas de aula/ Paulo Batista da Cruz Júnior. – Brasília, DF, 2023-
70 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

1. Alocação. 2. Otimização. I. Prof. Dr. John Lenon Cardoso Gardenghi. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Criação de uma aplicação para integrar algoritmos para o problema de alocação de salas de aula

CDU 02:141:005.6

Paulo Batista da Cruz Júnior

Criação de uma aplicação para integrar algoritmos para o problema de alocação de salas de aula

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 28 de agosto de 2023:

Prof. Dr. John Lenon Cardoso Gardenghi
Orientador

Prof. Dr. Bruno César Ribas
Convidado 1

Prof. Dr. Daniel Sundfeld Lima
Convidado 2

Brasília, DF
2023

Resumo

O problema da alocação de salas (PAS) é um desafio que as universidades costumam enfrentar regularmente, ele envolve a alocação de turmas nas salas de aula de forma a otimizar o uso dos recursos e atender as demandas de cada turma. Esta alocação leva em consideração as necessidades dos professores, dos alunos, das disciplinas e da capacidade das salas. Normalmente, a solução deste problema nas universidades costuma ser feita de forma manual, podendo levar dias para conseguir concluir, e mesmo assim o resultado pode não ser satisfatório. O objetivo desta pesquisa é criar uma aplicação que dê suporte à automação de todo este processo de alocação de salas. Para tanto, será desenvolvida nesta pesquisa uma aplicação que seja capaz de se integrar com vários algoritmos de otimização diferentes para resolver este problema da alocação de salas. A aplicação oferecerá uma interface para que o usuário consiga gerenciar as salas e escolher qual algoritmo de otimização deseja utilizar para resolver o problema. Esta proposta de solução poderia ser utilizada na prática pelos responsáveis pela alocação das salas, e com isso, diminuir o esforço e o tempo gastos na solução do PAS.

Palavras-chaves: alocação de salas; Engenharia de Software; algoritmos de otimização; aplicação web.

Abstract

The Classroom Allocation Problem (PAS) is a challenge that universities face on a regular basis, it involves allocating classes in classrooms in order to optimize the use of resources and meet the demands of each class. This allocation takes into account the needs of teachers, students, subjects, and room capacity. Usually, the solution of this problem in universities is often done manually, which can take days to complete, and even then the result may not be satisfactory. Therefore, the goal of this research is to create an application that can help automate this whole process of room allocation problem. Considering this, an application will be developed in this research that is able to integrate with several different optimization algorithms to solve this problem of room allocation. The application will offer an interface so that the user can manage the rooms and choose which optimization algorithm to use to solve the problem. This proposed solution could be used in practice by those responsible for allocating rooms, and thus reduce the effort and time spent on solving the PAS.

Key-words: classrooms allocation; Software Engineering; optimization algorithms; web application.

Lista de ilustrações

Lista de abreviaturas e siglas

CSS	Cascading Style Sheets
FGA	Faculdade do Gama
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
NP	Não Polinomial
PL	Programação Linear
PAS	Problema de Alocação de Salas
REST	Representational State Transfer
SQL	Structured Query Language
UNB	Universidade de Brasília
ITC-2019	2019 international timetabling competition
USP	University Service Planning

Sumário

1 Introdução

O processo de alocação de salas na Faculdade UnB Gama (FGA) é realizado manualmente pela coordenação dos cursos do campus (GARDENGHI, 2023). Este processo precisa acontecer antes do início do período de matrícula dos alunos e tem como objetivo garantir que exista salas suficientes para atender à demanda de cada disciplina, mesmo que nem sempre seja possível garantir isso.

Embora este processo manual de alocação de salas seja capaz de alcançar o objetivo, ele também pode ser trabalhoso e sujeito a erros. Além disso, este processo manual também pode ser demorado, pois exige que o servidor responsável pela alocação verifique manualmente as informações.

Informações como o histórico do número de alunos matriculados em uma turma são utilizadas para obter informações que são usadas na previsão do número de matrículas em cada disciplina. Por exemplo, para alocar salas para a disciplina de Estruturas de Dados 2, busca-se o histórico do número de alunos matriculados na disciplina de Estruturas de Dados 1 e também a porcentagem de alunos que foram aprovados na disciplina, com base nisso, é estimado quantos alunos provavelmente irão se matricular na disciplina de Estruturas de Dados 2. No entanto, mesmo com essas informações, pode ser que o responsável por alocar as salas encontre dificuldades para garantir que haja espaço suficiente para todos os alunos, já que o número de matrículas pode variar bastante entre os semestres, e portanto, pode ser difícil fazer uma estimativa precisa.

Segundo o Silva (2019), o Problema da Alocação de Salas (PAS) é um caso especial do problema de *educational timetabling*, que envolve a alocação eficiente de recursos educacionais em horários específicos. No contexto do PAS, o objetivo é alocar as turmas em salas de aula que possam comportar todos os alunos, levando em consideração a capacidade das salas e os choques de horário. É um desafio que é frequentemente enfrentado por universidades e outras instituições de ensino. A solução para o PAS é importante para garantir que os alunos tenham acesso a aulas de qualidade e que os professores tenham condições adequadas de trabalho (SILVA, 2019).

O PAS também é um problema que é difícil de ser generalizado, pois existem vários fatores que são únicos de cada instituição que devem ser considerados para se propor uma solução. Dependendo das restrições e peculiaridades de cada instituição, diferentes tipos de soluções podem ser necessários. Por isso, as aplicações que tentam resolver o PAS costumam oferecer soluções específicas para cada instituição (SILVA, 2019).

Considerando dificuldade de generalizar o problema do PAS, não foi possível encontrar uma solução unificada com a proposta de juntar essas soluções. Existem algumas

aplicações que criaram uma linguagem em comum para as soluções, como o *University Service Planning Schema* (BARICHARD et al., 2022) ou o formato usado no *International Timetabling Competition* 2019 (ITC2019, 2023). Mas elas não propõem criar uma arquitetura que consiga juntar todas essas peças.

1.1 Objetivos

O objetivo principal deste trabalho é criar uma interface que permita a integração de dados com os algoritmos criados para resolver o PAS. Este objetivo é detalhado nos seguintes objetivos específicos:

- Implementar a interface do sistema.
- Integrar com algum algoritmo de alocação existente.
- Criar um sistema de *plugin* para facilitar a integração de diferentes soluções.

1.2 Organização do documento

Este trabalho foi dividido nos seguintes capítulos:

- Capítulo 2 - Referencial teórico: explica a base teórica necessária para este trabalho, com foco na explicação dos conceitos de alocação de salas, e conceitos relacionados a estratégia de arquitetura utilizada no projeto.
- Capítulo 3 - Suporte tecnológico: explica quais foram as tecnologias utilizadas durante o desenvolvimento do trabalho.
- Capítulo 4 - Metodologia: explica a metodologia utilizada no projeto para guiar a pesquisa e desenvolvimento deste projeto.
- Capítulo 5 - Solução final: apresenta os componentes que foram desenvolvidos neste trabalho.
- Capítulo 6 - Conclusão: descreve as conclusões finais do trabalho.

2 Referencial Teórico

Para entender melhor os termos desta pesquisa, neste capítulo é explicado de forma detalhada o problema de alocação, e também são apresentados algumas das soluções para este problema. Além disso, também serão apresentados alguns conceitos de arquitetura que serão importantes para entender melhor a solução criada.

2.1 Problema de Alocação de Salas

Existem vários artigos de universidades que lidam com este problema do PAS. Alguns destes artigos são usados nesta pesquisa, como por exemplo CIRINO (2016), que aborda o problema no contexto da Universidade de São Paulo; SUBRAMANIAN et al. (2011) e SILVA (2019) da Universidade Federal da Paraíba; SALES (2015) da Universidade Federal de Santa Maria.

Ao longo deste trabalho os seguintes conceitos relacionados a este problemas são utilizados. Segundo Cirino (2016), a definição destes termos são os seguintes:

- **Aula:** Um encontro planejado e regulado por horários, neste encontro existe um professor responsável e seus estudantes.
- **Salas:** Um local para realizar as aulas.
- **Turma:** Um grupo de estudantes com um professor que tem uma ou mais aulas por semana, como por exemplo uma turma de Estrutura de Dados 1 com duas aulas semanais.
- **Currículo:** Um conjunto de aulas obrigatórias predefinidas para um certo grupo de estudantes.
- **Recurso:** Elementos associados às aulas que são necessários para o ensino ou ergonomia, como projetor, mesa de desenho, etc.

O problema de alocação de salas faz parte dos problemas do tipo *timetabling*. Segundo o Saviniec (2017), os problemas de *timetabling* escolar envolvem alocar horários para encontros entre professores e alunos, levando em consideração uma série de requisitos diferentes. Esses requisitos podem ser divididos em obrigatórios e não obrigatórios. Uma grade escolar é considerada viável quando atende a todos os requisitos obrigatórios, e é considerada ótima quando além de atender a esses requisitos obrigatórios, consegue minimizar as violações nos requisitos não obrigatórios.

Segundo Cirino (2016), o PAS é abordado de várias formas na literatura. Isso ocorre pois cada instituição ou região tem suas próprias necessidades e objetivos específicos, o que leva a formulações diferentes do problema. Porém, existem três restrições fortes que são intrínsecas ao PAS:

- Todas as aulas devem ser alocadas em uma sala.
- Não podem haver sobreposições de horários entre aulas em uma mesma sala.
- As aulas devem ser alocadas somente em salas que consigam atender as necessidades de recursos da sala e capacidade de acomodar todos os participantes.

Cirino (2016) também cita que existem algumas métricas que podem ser utilizadas para tratar o PAS:

- **Redução de deslocamento**: Minimizar os deslocamentos tanto dos professores quanto dos alunos.
- **Estabilidade de salas**: Tem o objetivo de manter um conjunto de salas e um conjunto específico de aulas. Por exemplo, aulas de um currículo devem permanecer em um mesmo bloco didático.
- **Restrição da utilização de salas**: Pretende evitar a subutilização dos recursos das salas, e a utilização da sala em qualquer horário.
- **Melhor utilização da capacidade da Sala**: Minimizar o número de assentos livres nas salas após a alocação de uma aula.

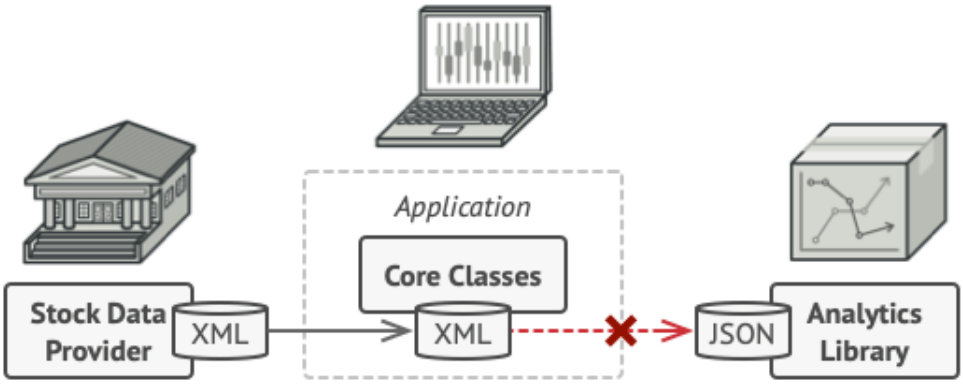
2.2 *Adaptador*

O *Adapter* é um dos padrões de *design* estrutural que permite que objetos com interfaces incompatíveis consigam colaborar entre si.

Shvets (2019) cita um exemplo da aplicação desse padrão no livro dele. No exemplo citado existe um biblioteca externa que faz análises, mas esta biblioteca só funciona com os dados no formato JSON. Então ela espera que os dados estejam em um formato compatível com a outra aplicação. É possível ver este exemplo na Figura 1 (SHVETS, 2019).

Para resolver o problema do exemplo, seria possível alterar a biblioteca para conseguir trabalhar com o formato XML. No entanto, esta mudança poderia causar algum efeito negativo nos códigos existentes da biblioteca. Ou não seria possível conseguir acesso ao código-fonte da biblioteca em primeiro lugar, o que tornaria esta abordagem impossível (SHVETS, 2019).

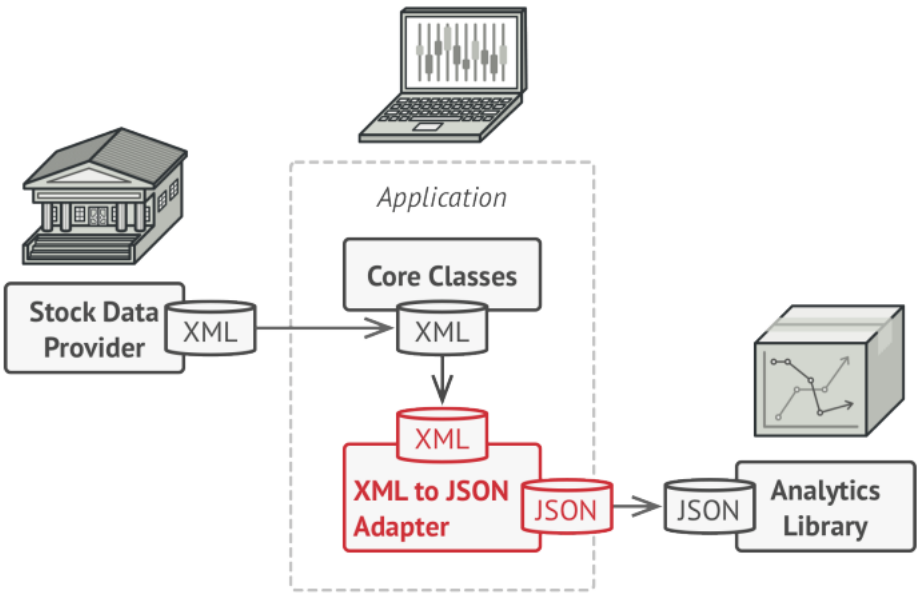
Figura 1 – Exemplo de duas aplicações que não conseguem se comunicar.



Fonte: (SHVETS, 2019).

A proposta do padrão *Adapter* é de resolver este problema. Neste padrão é usado um adaptador que serve como uma camada em cima da aplicação original, e com isso, o adaptador ficaria responsável por fazer a comunicação com os outros sistemas. Dessa forma, as aplicações conseguem se comunicar, mesmo que elas estejam esperando receber dados em formatos diferentes. Um exemplo dessa implementação é mostrado na Figura 2.

Figura 2 – Exemplo de uso do Adapter.



Fonte: (SHVETS, 2019).

As principais vantagens na utilização deste padrão seria de conseguir separar o código de conversão dos dados da lógica de negócio do programa, e a possibilidade da introdução de novos tipos de adaptadores no programa sem quebrar os códigos que já existentes (SHVETS, 2019).

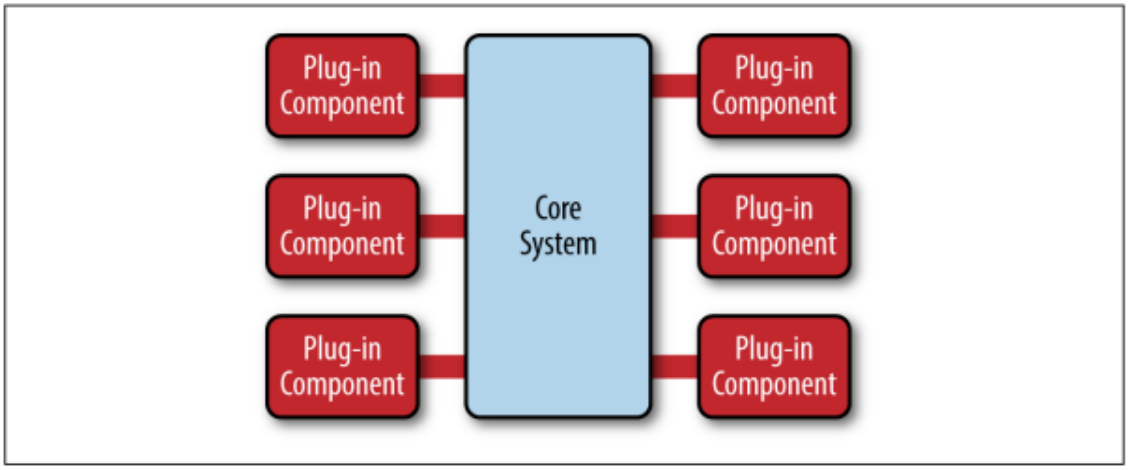
2.3 *Arquitetura de* Microkernel

A arquitetura de *microkernel* (também conhecida como arquitetura padrão *plug-in*) permite que novas funcionalidades sejam adicionadas como *plug-in* na aplicação principal. Dessa forma, melhorando a extensibilidade e isolação do código (RICHARDS, 2015).

2.3.1 Descrição do padrão

Segundo Richards (2015), esse padrão de arquitetura consiste de dois tipos de componentes: o sistema principal e os módulos *plug-in*. A lógica da aplicação dessa forma ficaria dividida da seguinte forma: a lógica básica do sistema ficaria no sistema principal e a lógica adicional nos módulos *plug-in* independentes. A Figura 3 mostra como esses componentes são organizados.

Figura 3 – Componentes do padrão de arquitetura *Microkernel*.



Fonte: (RICHARDS, 2015).

- **Módulos *Plug-in*:** Componentes independentes, que contêm funcionalidades especializadas, adicionais e customizadas. Geralmente é recomendável que os módulos *plug-in* evitem de comunicar entre si, para evitar problemas de dependências.
- **Sistema *core*:** Possui a responsabilidade de saber quais são os módulos *plug-in* disponíveis e como se comunicar com eles. Uma forma de implementar isso pode ser por meio do uso de um ***plug-in registry***, onde ficaria guardado informações sobre cada módulo plug-in.

2.3.2 Comunicação

Os módulos *plug-ins* podem ser conectados com o sistema principal por meio de várias formas, incluindo o uso de um sistema de mensageria. Esse padrão de arquitetura

não chega a especificar os detalhes de implementação, apenas que os *plug-ins* devem ser intendentes (RICHARDS, 2015).

Para que a aplicação seja utilizada por *plug-ins* é necessário que a interface esteja bem definida. Pois com a interface bem definida, novos *plug-ins* podem ser facilmente criados (APPLE, 2023).

No caso dos componentes *plug-in* serem desenvolvidos por terceiros, é comum que exista a necessidade da criação de novos adaptadores para cada *plug-in*, e com isso esses módulos conseguirem seguir a o padrão de comunicação definida pelo sistema (RICHARDS, 2015).

2.3.3 Vantagens

A principal vantagem desta arquitetura seria a remoção da responsabilidade do sistema principal de manter códigos customizados, pois cada *plug-in* poderia ficar com a responsabilidade de oferecer estas funcionalidades customizadas. Fora isso, uma outra vantagem é que os *plug-ins* são independentes, ou seja, podem ser removidos e adicionados sem alterar os sistemas existentes.

2.3.4 Desvantagens

Caso tenha que adicionar um novo *plug-in* que não segue o contrato de comunicação estabelecido, haveria a necessidade da criação de um novo um adaptador para este *plug-in*.

2.4 *Arquitetura Baseada em Eventos*

Segundo Richards (2015), esse padrão de arquitetura baseada em eventos é utilizado para criar aplicações distribuídas altamente adaptáveis e escaláveis, podendo ser aplicado tanto em projetos pequenos quanto em projetos de maior escala. Nesta arquitetura existem vários componentes que fazem o recebimento e processamento de dados de forma assíncrona, apresentando baixo acoplamento com outros componentes.

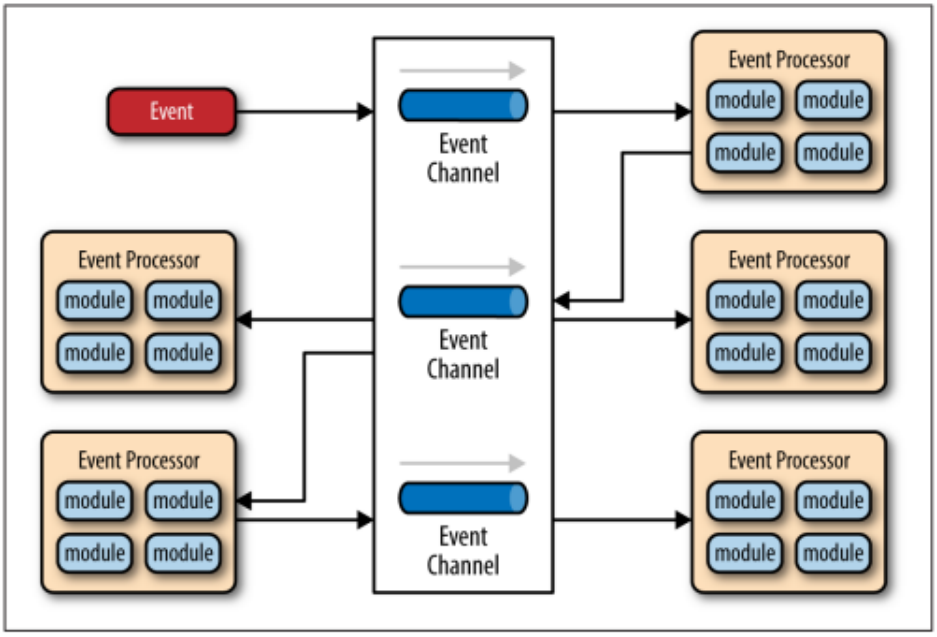
Nesta arquitetura existem duas topologias, o *mediator* e o *broker*. O *mediator* serve para quando é necessário orquestrar os eventos utilizando um mediador central para controle, e o *broker* quando não é necessário este tipo de orquestração. Para este capítulo o foco vai ser maior no *broker*, pois esta foi a implementação escolhida (RICHARDS, 2015).

2.4.1 *Broker*

O *broker* é utilizado para trabalhar com a comunicação entre os diferentes sistemas, é responsável por receber as mensagens enviadas pelos produtores e rotear para

os seus devidos consumidores (RABBITMQ, 2023a). O *broker* pode inclusive ajudar a fazer o controle dessas mensagens, como por exemplo, salvar os dados sobre o estado das mensagens. Dessa forma, é possível verificar com o *broker* quais foram as mensagens enviadas. Uma das tecnologias que utiliza do *broker* é o próprio *RabbitMQ*. Esta topologia é mostrada melhor na Figura 4 (NEWMAN, 2015).

Figura 4 – Exemplo de como funciona a topologia do *broker*.



Fonte: (RICHARDS, 2015).

2.4.2 Contrato entre os componentes

Segundo o Richards (2015), se houver uma decisão de utilizar esta arquitetura, é muito importante estabelecer logo no início o contrato de comunicação entre os componentes, e qual seria a política de versionamento deste contrato.

Neste contrato poderia por exemplo estabelecer que as mensagens devem estar no formato XML, e que uma certa estrutura padrão deve ser seguida.

2.5 Comunicação Síncrona ou Assíncrona

Segundo o Newman (2015), em uma comunicação **síncrona**, uma chamada é feita para um servidor remoto, no qual bloqueia a comunicação até que a operação consiga ser completada. Ou seja, os dois componentes estariam acoplados, mantendo a comunicação aberta até que a operação seja concluída. Na comunicação **assíncrona**, o componente que chama o servidor não precisa esperar pela conclusão da operação e pode até mesmo não

se importar se ela foi finalizada ou não. Sendo assim, o acoplamento entre os componentes é menor.

2.5.1 Comunicação Síncrona

A comunicação síncrona geralmente é mais fácil de compreender, pois permite saber quando o processamento de algum dado foi terminado ou não. Mas acaba criando um alto acoplamento entre os componentes. Se no servidor houver algum processamento prolongado, a aplicação que precisa do servidor estaria bloqueada esperando este processamento terminar.

2.5.2 Comunicação Assíncrona

A comunicação assíncrona costuma ser mais complexa, porém pode ser útil em aplicações que requerem processamentos de longa duração. Pois a aplicação cliente não precisará esperar a resposta do servidor, sendo assim, é possível continuar com outras tarefas enquanto o processamento é realizado. Dessa forma, se um usuário estiver utilizando uma interface *web*, a experiência seria melhor, pois a interface permaneceria responsiva (NEWMAN, 2015).

Fora isso, essa comunicação baseada em eventos também é altamente desacoplada. É possível adicionar novos componentes para os eventos gerados, sem afetar as outras aplicações já existentes (NEWMAN, 2015).

No entanto, apesar dessas vantagens, a implementação desse tipo de comunicação pode ser mais complexa. Ao implementar esse padrão, é necessário considerar diversos problemas que podem surgir, tais como erros no processamento de mensagens, falta de resposta por algum componente e lógica de reconexão em caso de falhas na conexão (RICHARDS, 2015).

2.6 Advanced Message Queueing Protocol (AMQP) 0–9–1

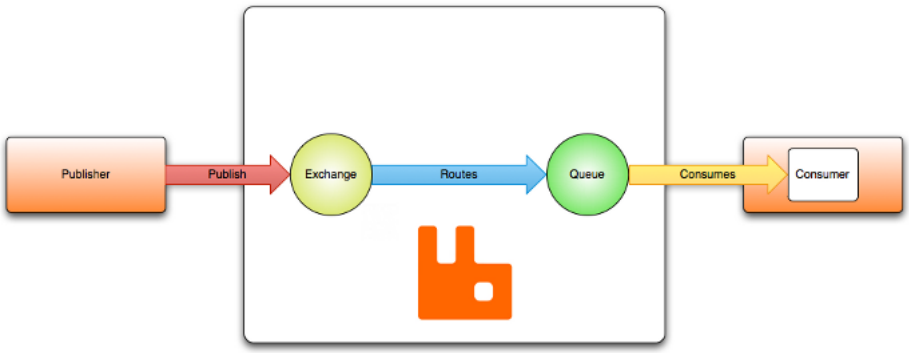
Segundo a documentação oficial do RabbitMQ (2023a), o **AMQP 0-9-1** é um protocolo de mensagem que permite às aplicações do cliente se comuniquem com os *brokers*.

Neste modelo as mensagens são publicadas para as ***exchanges***, e elas são responsáveis por pegar as mensagens e enviar para as filas corretas. As *exchanges* distribuem as cópias dessas mensagens para as filas, seguindo a regra estabelecidas pelo cliente.

As **filas** neste modelo são parecidas com as filas dos outros tipos de sistemas de mensageria. Elas armazenam as mensagens que são consumidas pelas aplicações.

O ***broker*** permite a entrega dessas mensagens diretamente para os consumidores das filas. O *broker* também permite a possibilidade do cliente consumir diretamente essas as mensagens *on demand* que estão nas filas. Na Figura 5 é apresentado os componentes utilizados neste protocolo.

Figura 5 – Exemplo de uso do protocolo.

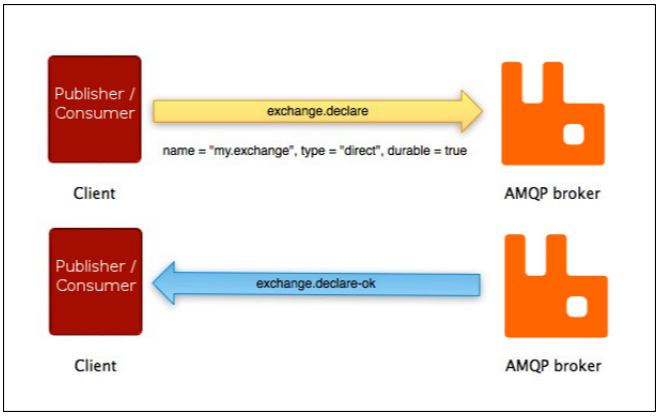


Fonte: (RABBITMQ, 2023a).

Fora isso, este protocolo também permite que as aplicações definam os esquemas de roteamento e entidades, ao invés de depender de um administrador no *broker*. Dessa forma, isso torna o protocolo programável, possibilitando, por exemplo, que uma aplicação possa declarar filas e *exchanges*, definir suas próprias conexões e se inscrever nas filas.

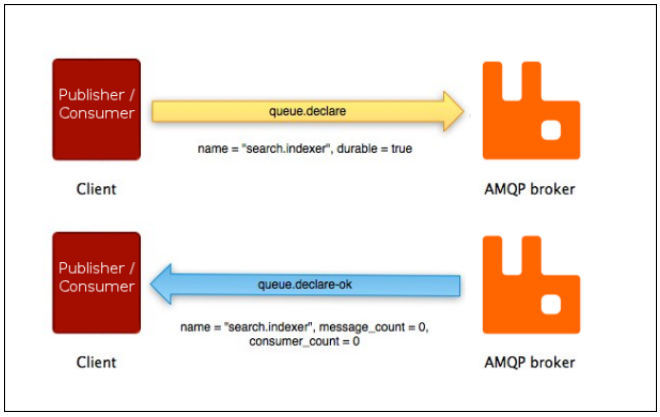
Alguns exemplos dessa programação são mostrados nas Figuras 6 e 7, nestes exemplos um cliente consegue criar novas *exchanges* ou novas filas por meio da programação. Além da criação, este protocolo também permite a deleção.

Figura 6 – Exemplo de declaração de uma *exchange*.



Fonte: (RABBITMQ, 2023a).

Figura 7 – Exemplo de declaração de uma fila.



Fonte: (RABBITMQ, 2023a).

2.7 UniTime

No geral, existem muitas soluções existentes que propõem resolver o PAS, porém elas são específicas para a utilização de um certo algoritmo. Não foi possível encontrar a existência de uma ferramenta com a mesma proposta deste trabalho.

No entanto, durante a pesquisa sobre o assunto, descobriu-se aplicações com algumas similaridades, ou seja, com o mesmo objetivo de criar uma interface *web* que permite a utilização de algum algoritmo para solucionar o PAS. A documentação de alguma dessas aplicações encontradas exploram apenas a parte dos algoritmos e não entram em muitos detalhes como a alocação poderia ser resolvida na prática em conjunto com a interface. Algumas destas aplicações foram estudadas, como por exemplo o (UNITIME, 2023), (NAVUDURI, 2016) e (LEGASPI et al., 2012). Nesta seção será apresentada a aplicação do *UniTime*, pois ela foi utilizada como exemplo para a criação deste projeto.

A aplicação **UniTime** é um sistema de código aberto que oferece uma plataforma única para fazer a alocação e agendamento de cursos, exames, eventos e estudantes de uma universidade. Este projeto faz parte da *Apereo Foundation*, uma organização sem fins lucrativos cuja missão é desenvolver e manter software de código aberto para o ensino superior (UNITIME, 2023).

Das soluções encontradas, esta parece ser a mais completa e a mais utilizada na prática pelas universidades, como a *Purdue University*. Ela permite vários modos de alocação, tanto manual quanto totalmente automatizada. Fora isso, ela permite mudanças interativas na alocação (UNITIME, 2023).

A sua única limitação é a utilização de um único algoritmo para fazer a alocação de salas.

2.8 Considerações Finais

Neste capítulo, foram apresentadas algumas universidades que lidam com o problema do PAS, e também alguns exemplos de sistemas que foram desenvolvidos para resolver este problema. No entanto, dentre as soluções encontradas, estas são específicas para a utilização de um certo algoritmo.

Para a criação do projeto deste trabalho de TCC, vários conceitos foram utilizados. Portanto, é fundamental a compreensão desses conceitos expostos neste capítulo, tais como o adaptador, arquitetura *Microkernel* e a arquitetura baseada em eventos. Todos esses conceitos expostos são necessários para o melhor entendimento da solução final.

3 Suporte Tecnológico

Para entender melhor as tecnologias utilizadas, neste capítulo é explicada de forma detalhada cada tecnologia que será utilizada para conseguir implementar a aplicação proposta nesta pesquisa.

3.1 *Node.js*

O *Node.js* é um ambiente de execução *Javascript* que possui um tempo de execução assíncrono, e é projetado para construir aplicações que sejam escaláveis (NODE.JS, 2023).

O principal motivo da escolha desta tecnologia é a possibilidade de utilizar a mesma linguagem de programação, o *Javascript*, tanto no servidor quanto no *frontend*. No projeto, o *Node.js* foi utilizado para a criação do servidor.

3.2 *Angular*

O *Angular* é uma plataforma de desenvolvimento que ajuda a criar interfaces para aplicações web. Ele disponibiliza um *framework* baseado em componentes, uma coleção de bibliotecas com diversas funcionalidades e um conjunto de ferramentas para ajudar o desenvolvedor na escrita do código (ANGULAR, 2023). Essa tecnologia foi utilizada para desenvolver toda a parte que envolve a interface do usuário.

Atualmente, existem outras ferramentas com propósitos semelhantes. A escolha dessa tecnologia foi feita principalmente devido à sua estrutura mais bem definida e também por conta da familiaridade do autor com ela.

3.3 *Express*

Express.js, baseado na linguagem JavaScript, é um *framework* que possui um conjunto de funcionalidades que ajudam a criar um sistema para lidar com solicitações e respostas *HTTP* (EXPRESS, 2023).

Essas funcionalidades adicionais que este *framework* possui são feitas a partir das funções *middlewares*. Essas funções podem ser integradas ao fluxo das requisições, possibilitando a execução de tarefas adicionais, como por exemplo, autenticação e registro de *logs* (EXPRESS, 2023). Com isso, a flexibilidade do uso desta tecnologia na aplicação é aumentada.

Fora isso, uma das suas principais vantagens do *Express.js* é que permite a criação de aplicações *web* de forma mais simples e rápida. Neste trabalho, o *framework* foi utilizado em conjunto com o servidor *Node.js*.

3.4 *PostgreSQL*

O *PostgreSQL* é um sistema de gerenciamento de banco de dados que possui código aberto. Ele foi projetado para ser confiável, robusto e estável. Também possui um número avançado de recursos que podem ser utilizados (GROUP, 2023). Foi o único sistema de gerenciamento de banco de dados utilizado para todo o projeto.

3.5 *Git*

O *Git* é uma ferramenta de controle de versão distribuído utilizado para ajudar no rastreamento das alterações feitas no código-fonte de um projeto. Esta tecnologia ajuda no processo de versionamento de projetos, e com isso, permite o acompanhamento e gerenciamento das mudanças no código de forma mais rápida e eficiente. Ele foi utilizado durante o desenvolvimento deste trabalho para ajudar nos rastreamento das alterações do código-fonte (GIT, 2023).

3.6 *RabbitMQ*

O *RabbitMQ* é um sistema *broker* de mensageria de código aberto, serve como um intermediário entre as mensagens enviadas entre os produtores e consumidores. Ele também oferece uma plataforma em comum para o envio seguro de mensagens e segue o protocolo *AMQP* (Advanced Message Queuing Protocol), definido na seção 2.6. No contexto do projeto, o *RabbitMQ* foi o sistema utilizado para troca de eventos entre os sistemas desenvolvidos (RABBITMQ, 2023b).

3.7 *GraphQL*

É uma linguagem de consulta feita para servidores com interface de programação de aplicações (APIs). Com ela é possível extrair dados de várias fontes em apenas um *request*. Foi utilizada para comunicação da aplicação *frontend* com o servidor (REDHAT, 2023).

3.8 Lista das tecnologias utilizadas

Na tabela 1 é possível observar a lista das tecnologias utilizadas no desenvolvimento do projeto.

Tabela 1 – Ferramentas usadas no projeto

Nome	**Descrição**	**Versão**
Node.js	Ambiente de execução *Javascript*	20.2.0
Angular	*Framework* Javascript para criação de interfaces *web*	16.0.0
Express	*Framework* Javascript de solicitações e respostas *HTTP*	4.18.2
PostgreSQL	Sistema de gerenciamento de banco de dados	14.8
Git	Ferramenta de controle de versão distribuído	2.34.1
RabbitMQ	Sistema *broker* de mensageria	3.12.2
GraphQL	Linguagem de consulta	15.3.0

4 Metodologia

Para entender a metodologia utilizada para cada fase do trabalho, é apresentado neste capítulo todo o detalhamento metodológico utilizado neste trabalho. A primeira seção deste capítulo descreve como o tipo de pesquisa é classificada considerando vários critérios. Em seguida, os fluxos de atividades mostram todas as atividades realizadas no TCC 1 e 2. Depois disso, é definida a metodologia que seria utilizada na fase de desenvolvimento do projeto.

4.1 Fluxo de atividades

Na Figura 8 é mostrado todo o fluxo de atividades utilizado para desenvolver o TCC 1, e na Figura 9 é mostrado uma continuação deste fluxo com a parte do TCC 2. Nesta segunda parte possui a parte prática do projeto.

4.2 Introdução

O objetivo desta seção é classificar de forma correta esta pesquisa em relação a **abordagem**, **natureza**, **objetivos** e **procedimentos**. Para que uma pesquisa possa ser considerada científica é preciso identificar o método utilizado na pesquisa (GIL, 2008). Estas são as classificações citadas pelo Gerhardt e Silveira (2009).

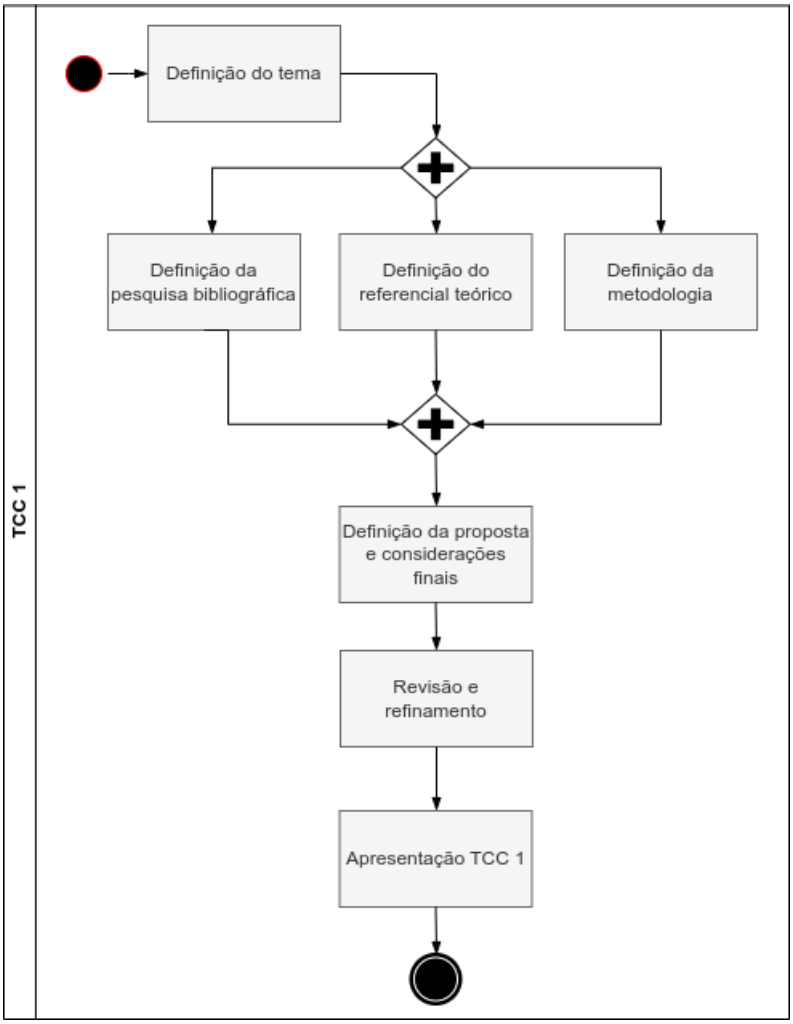
4.2.1 Abordagem

Pela classificação de abordagem, esta pesquisa poderia ser classificada como **quantitativa** e **qualitativa**, pois ela possui características que são únicas para ambas classificações. A quantitativa é mais focada na objetividade, realiza a coleta e análise dos dados sem depender da intuição. Enquanto a qualitativa utiliza o seu lado subjetivo para conseguir analisar e entender o experimento. Ou seja, a qualitativa consegue analisar as informações de uma forma organizada, mas intuitiva (GERHARDT T. E.; SILVEIRA, 2009).

4.2.2 Natureza

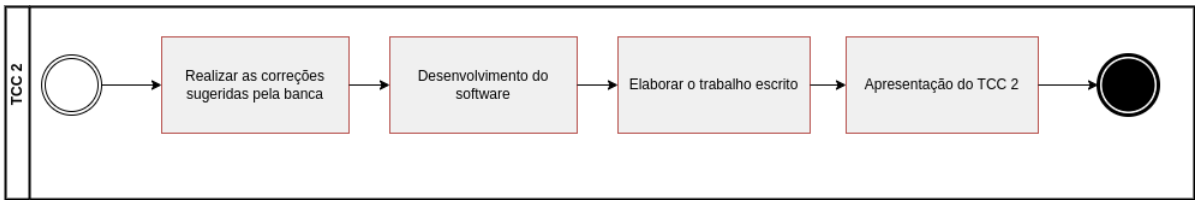
Com base na natureza esta pesquisa pode ser classificada como **pesquisa aplicada**, pois este trabalho envolve o desenvolvimento de uma aplicação para resolver o PAS. Existem dois tipos de natureza da pesquisa: a **pesquisa básica** e a **pesquisa aplicada**. A **pesquisa básica** realiza a pesquisa sem uma aplicação prática. Enquanto a **pesquisa**

Figura 8 – Fluxo de atividades do TCC 1.



Fonte: Autor.

Figura 9 – Fluxo de atividades do TCC 2.



Fonte: Autor.

aplicada envolve a criação de uma aplicação para resolver um certo problema para a pesquisa (GERHARDT T. E.; SILVEIRA, 2009).

4.2.3 Objetivos

Com base nos objetivos esta pesquisa pode ser classificada como **pesquisa exploratória**, pois este trabalho procura entender melhor o problema e como resolvê-lo. Existem três grupos que uma pesquisa pode ser classificada com base nos objetivos (FREITAS,

2013):

- **Pesquisa exploratória**: Procura descobrir melhor o problema e como resolvê-lo.
- **Pesquisa descritiva**: Apenas descreve como uma certa realidade funciona.
- **Pesquisa explicativa**: Procura descobrir explicações por trás das coisas e suas causas, usando os resultados oferecidos.

4.2.4 Procedimentos

Os procedimentos técnicos é a maneira como é feita a busca por conhecimento para elaborar uma certa pesquisa (FREITAS, 2013). Existem vários tipos diferentes de modalidades de pesquisa, como por exemplo a pesquisa experimental, pesquisa bibliográfica, pesquisa documental, pesquisa de campo, pesquisa de caso e etc. Do pontos de vista dos procedimentos técnicos esta pesquisa pode ser classificada nos grupos de **pesquisa bibliográfica** e **pesquisa-ação**.

Na **pesquisa bibliográfica** o pesquisador busca por materiais já publicados sobre o assunto que está sendo pesquisado (GERHARDT T. E.; SILVEIRA, 2009).

Na **pesquisa-ação** o objetivo deve ser conseguir resolver ou esclarecer os problemas da situação observada. E durante o processo, deve ser feito um acompanhamento das decisões e ações de todas as atividades que foram feitas (FREITAS, 2013).

4.3 Metodologia de desenvolvimento

No desenvolvimento desta aplicação será utilizado os métodos ***Kanban*** e ***Scrum***. Ambos os métodos são considerados ferramentas do método ágil baseado no trabalho incremental, ou seja, ferramentas com o objetivo de fazer a entrega do software de forma frequente e incremental (COCCO et al., 2011). Fora isso, o trabalho foi orientado pelos *princípios ágeis*.

Os **princípios ágeis** fornecem a base para orientar o processo de desenvolvimento de software. Uma das maiores características de um desenvolvimento ágil é a habilidade de se adaptar e responder a mudanças, tanto no ambiente, quanto nos requisitos do usuário ou nas restrições das entregas. Devido à melhora na produtividade, qualidade e entrega do software, foi feito a decisão de utilizar este paradigma no trabalho deste TCC (AL-SAQQA; SAWALHA; ABDELNABI, 2020).

O ***Scrum*** é um *framework* da metodologia ágil, que se adapta aos diferentes contextos do desenvolvimento de software. O uso do *Scrum* neste trabalho é relacionado a divisão do trabalho em uma lista de entregáveis menores, e a criação de um ***product***

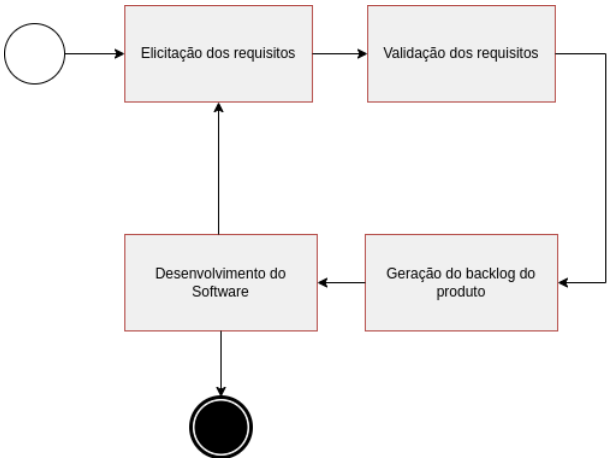
backlog. Fora isso, o trabalho foi dividido em vários ciclos iterativos e incrementais. Neste tipo de prática, existem vários ciclos que potencialmente podem introduzir uma nova funcionalidade ou melhoria, e o conjunto destes ciclos pode acumular em um produto que esteja pronto para entrega. Com isso, é possível entregar de forma mais frequente software funcional (COCCO et al., 2011).

Usar ***Kanban*** significa quebrar o trabalho em vários itens diferentes, colocando a descrição de cada item do trabalho em uma certa carta, e depois disso colocar estas cartas em uma tabela. Nesta tabela teria também os status de cada cada carta. Sendo assim, o fluxo de trabalho de uma pessoa ficaria mais facilmente visível para todo mundo. Seria mais fácil por exemplo de olhar todos os trabalhos que precisam ser feitos, e todo o trabalho que já foi completado (COCCO et al., 2011).

4.4 Processo de desenvolvimento

Todas as fases de desenvolvimento realizadas no projeto podem ser vistas na Figura 10. Esta figura mostra também que o desenvolvimento foi feito de forma iterativa.

Figura 10 – Etapas de desenvolvimento do software.



Fonte: Autor.

- **Elicitação dos requisitos**: Definição de quais são necessidades do usuário, escopo do produto e quais as limitações do software.
- **Validação dos requisitos**: Validação dos requisitos elicitados com o professor orientador.
- **Geração do *backlog* do produto**: Observação dos requisitos licitados, e a criação de uma lista com as funcionalidades desejadas no produto.
- **Desenvolvimento do software**: Desenvolvimento das funcionalidades desejadas.

4.5 Cronograma

As Tabelas 2 e 3 foram criadas para mostrar os cronogramas das atividades realizadas durante o desenvolvimento deste trabalho.

Tabela 2 – Cronograma TCC 1 - 2022/2

Atividades	**Outubro**	**Novembro**	**Dezembro**	**Janeiro**	**Fevereiro**
Definição do tema	X	X			
Pesquisa bibliográfica	X	X	X	X	X
Definição do referencial teórico			X	X	
Definição da metodologia				X	
Definição da proposta			X	X	
Revisão e refinamento				X	X
Apresentação TCC 1					X

Tabela 3 – Cronograma TCC 2 - 2023/1

Atividades	**Março**	**Abril**	**Maio**	**Junho**	**Julho**
Realizar as correções no texto	X				
Desenvolvimento do software	X	X	X	X	X
Elaborar o trabalho escrito				X	X
Apresentação do TCC 2					X

5 Solução Final

O objetivo deste TCC é criar uma ferramenta para ajudar na resolução do Problema de Alocação de Salas (PAS). Este capítulo aborda os requisitos e arquitetura do sistema desenvolvido para atender a este objetivo, e os definidos na Seção 1.1. Esta solução se encontra em um repositório do *Github*, https://github.com/higton/allocate-app.

5.1 Requisitos

Nesta seção são apresentados os requisitos levantados para a ferramenta proposta neste trabalho. Neste levantamento foi levado em consideração o escopo do produto e quem são os usuários desta aplicação. O escopo definido é que a aplicação deve conseguir alocar salas para todos os cursos da FGA, e que deve permitir a utilização de outros algoritmos. Os usuários considerados para essa elicitação de requisitos são os responsáveis pela alocação na universidade.

Os requisitos foram separados em **requisitos funcionais** e **não-funcionais**, a seguir estão descritos os **requisitos funcionais** da aplicação proposta:

- É possível editar ou inserir dados manualmente através da interface do sistema;
- O sistema deve permitir a utilização de diferentes algoritmos de alocação de sala;
- Deve garantir que apenas os usuários autorizados possam acessar e modificar as informações do sistema;
- Permite que o usuário insira novos algoritmos de resolução do PAS em tempo de execução, como se fossem *plugins*;
- Os resultados gerados por cada algoritmo de alocação são disponibilizados através da interface do sistema;

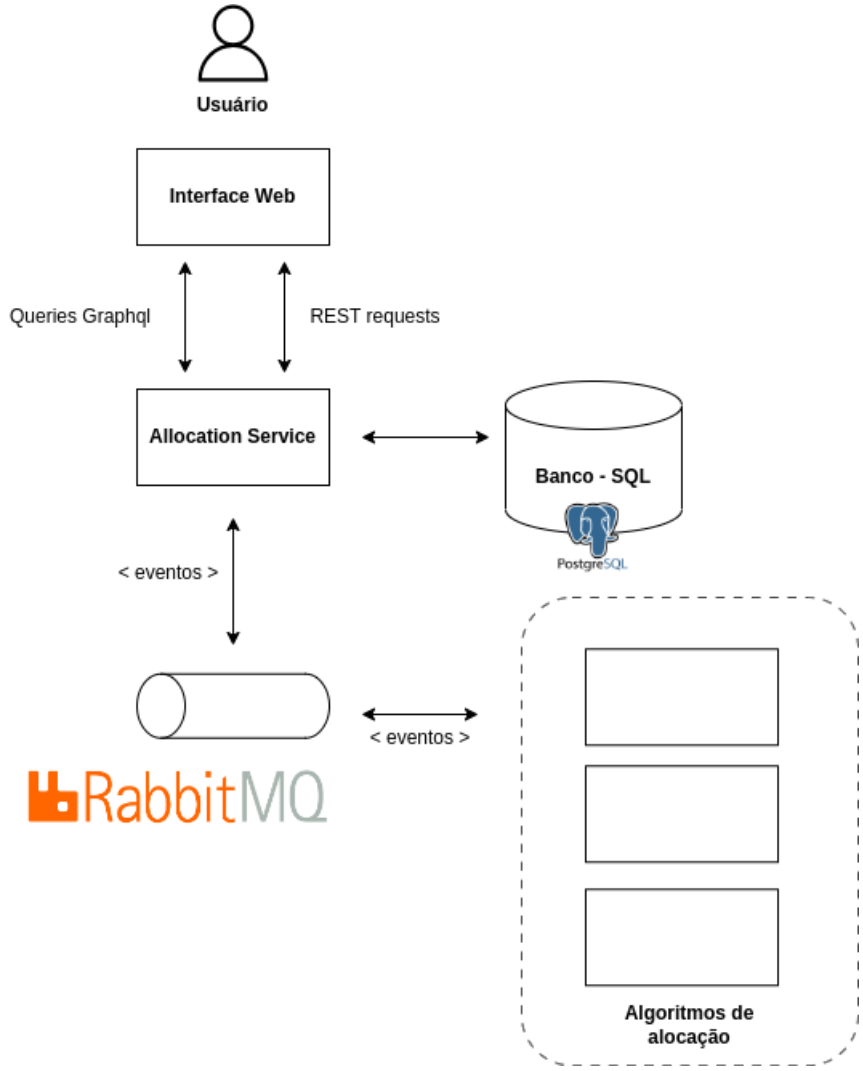
Apresentamos a seguir os **requisitos não-funcionais**:

- Não precisa ter uma alta consistência dos dados, ou seja, consistência eventual;
- Durabilidade, os dados armazenados nos sistema devem ser duráveis e não podem ser perdidos;
- Usabilidade, mesmo enquanto o algoritmo da solução estiver processando os dados, a interface *web* deve permanecer responsiva, não podendo travar;

5.2 Arquitetura do sistema

Nesta seção são descritos todos os componentes que fazem parte da arquitetura da aplicação, e como eles interagem entre si. A arquitetura do sistema pode ser vista na Figura 11.

Figura 11 – Arquitetura do sistema.



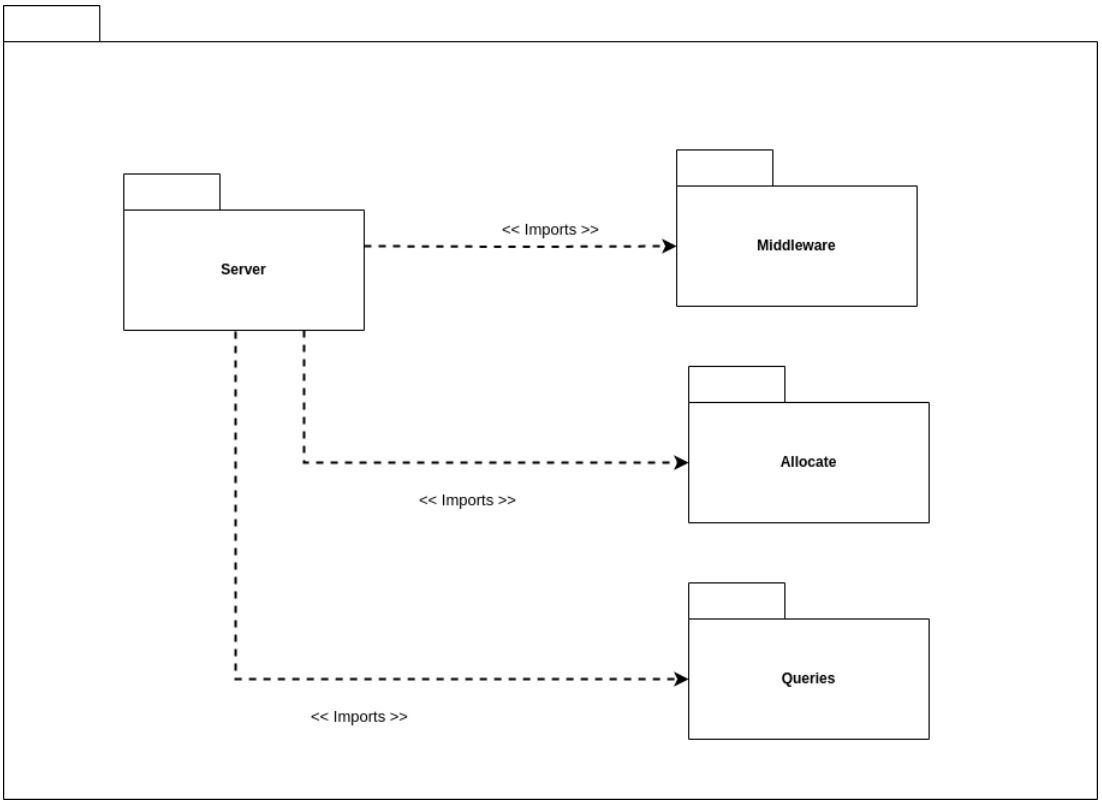
Fonte: Autor.

5.2.1 Servidor

O servidor *allocation server* é o componente que possui a responsabilidade de se integrar com 3 componentes diferentes, a interface *web*, algoritmos externos e o banco de dados. A comunicação do servidor com a aplicação interface *web* é feita com o *GraphQL* e *endpoints REST*. Para a comunicação entre o servidor e os algoritmos externos, foi feita a escolha de utilizar uma arquitetura baseada em eventos com o *RabbitMQ*. A linguagem escolhida para este servidor é o *Node.js*, e as tecnologias utilizadas foram o *framework*

express e *GraphQL*. A Figura 12 ilustra a forma como este componente está organizado atualmente.

Figura 12 – Diagrama de pacotes do *Allocation Service*.



Fonte: Autor.

5.2.2 Banco de dados

Neste componente foi utilizado a linguagem *SQL*, mais especificamente a tecnologia *PostgresSQL*. Foram armazenados informações como salas de aulas disponíveis, cursos criados e contas dos usuários.

5.2.3 Algoritmos externos

Os algoritmos externos são o conjunto de diferentes algoritmos que conseguem resolver o PAS. Todos eles devem seguir uma interface em comum para conseguirem se comunicar com o servidor. A linguagem e as tecnologias utilizadas por este componente será escolhida pelo desenvolvedor responsável pelo algoritmo, seria necessário apenas utilizar do mesmo contrato de comunicação definido com a aplicação central.

5.2.4 Broker

Responsável por rotear as mensagens entre os algoritmos externos e a aplicação central. Para esta função, foi decidido utilizar a tecnologia do *RabbitMQ*.

5.2.5 Arquitetura baseada em eventos

O motivo da escolha desta arquitetura baseada em eventos foi por conta da necessidade de trabalhar de forma assíncrona com os algoritmos. Considerando que cada algoritmo de alocação pode demorar bastante tempo, uma arquitetura baseada em eventos seria a solução mais adequada. Para esta consideração, foram levadas em conta as informações levantadas no referencial teórico, (Capítulo 2).

Caso fosse utilizado algum método de comunicação como por exemplo o *REST*, a aplicação de interface *web* não seria muito responsiva e criaria um forte acoplamento da aplicação central com os algoritmos de alocação.

Nesta arquitetura cada algoritmo seria considerado como um novo micro serviço, onde cada serviço ficaria responsável por receber os eventos, processar e enviar os resultados de forma independente. Novos algoritmos poderiam ser adicionados sem causar alterações na aplicação central, ou nos algoritmos que já existem. O lado negativo desta escolha seria a complexidade adicional de desenvolver esta arquitetura.

5.2.6 Arquitetura de *Microkernel*

O motivo da escolha desta arquitetura também foi baseada nos requisitos existentes, a necessidade da criação de um sistema que consiga facilmente adicionar novos algoritmos em tempo de execução.

Para conseguir implementar esta arquitetura foi utilizado os conceitos do Capítulo 2. Foi utilizado o conceito de *plugin registry* para descobrir quais são as soluções existentes. E na parte de comunicação dos *plug-ins* com a aplicação central, foi escolhido comunicação baseada em eventos.

5.2.6.1 Contrato de comunicação

No Capítulo 2 é citado que para utilizar este tipo de arquitetura, é necessário que primeiro um contrato de comunicação entre os componentes estejam bem definidos.

Na pesquisa realizada no presente trabalho, foram encontrados dois protocolos para estabelecer o contrato de comunição: *University Service Planning Schema* (BARICHARD et al., 2022) e o formato da *International Timetabling Competition* 2019 (ITC2019, 2023).

O ITC 2019 é uma competição com o objetivo de incentivar ainda mais as pesquisas feitas sobre a solução de problemas envolvendo a alocação de cursos, no contexto de uma universidade. Os dados da competição são baseados em dados reais que foram coletados usando a aplicação UniTime (MÜLLER, 2022). Nesta competição foi definido um formato XML de entrada para os algoritmos e de saída (ITC2019, 2023).

Sobre o formato dos dados do ITC 2019, no formato de entrada é possível passar

as salas, cursos, alunos e as restrições de cada curso. As restrições podem ser fortes ou fracas. Agora, no formato do resultado são passados as classes, com as suas respectivas salas e os alunos que foram alocados (ITC2019, 2023).

O USP é um modelo proposto para que algoritmos sigam um certo formato de comunicação. Ele é específico para os algoritmos que desejam resolver uma certa classe de problemas de alocação. Este modelo envolve a alocação de cursos, recursos e estudantes no contexto de uma universidade. É um formato que utilizou como base o formato do ITC 2019, e que também utiliza *XML*. Este modelo parece ainda estar em desenvolvimento, mas já oferece algumas ferramentas para facilitar a criação de adaptadores para este formato e alguns exemplos de soluções que utilizam deste formato (BARICHARD et al., 2022).

Por conta da facilidade do uso do formato ITC 2019, número de aplicações que já utilizam ela e documentação objetiva, foi definido que a interface de comunicação seria feita usando este formato. Fora isso, este formato parece satisfazer todos os requisitos necessários para fazer a alocação, considerando o contexto da solução proposta por este trabalho (ITC2019, 2023).

Assim como foi descrito na Seção 2.2, as aplicações que não estiverem seguindo este formato, seria necessário a utilização de adaptadores para traduzirem o formato usado.

5.2.7 Comunicação da aplicação central com os algoritmos de alocação

A Figura 13 mostra os componentes utilizados na comunicação entre os algoritmos de alocação e a aplicação central. Sempre que um novo algoritmo for adicionado no projeto, estes componentes estarão presentes.

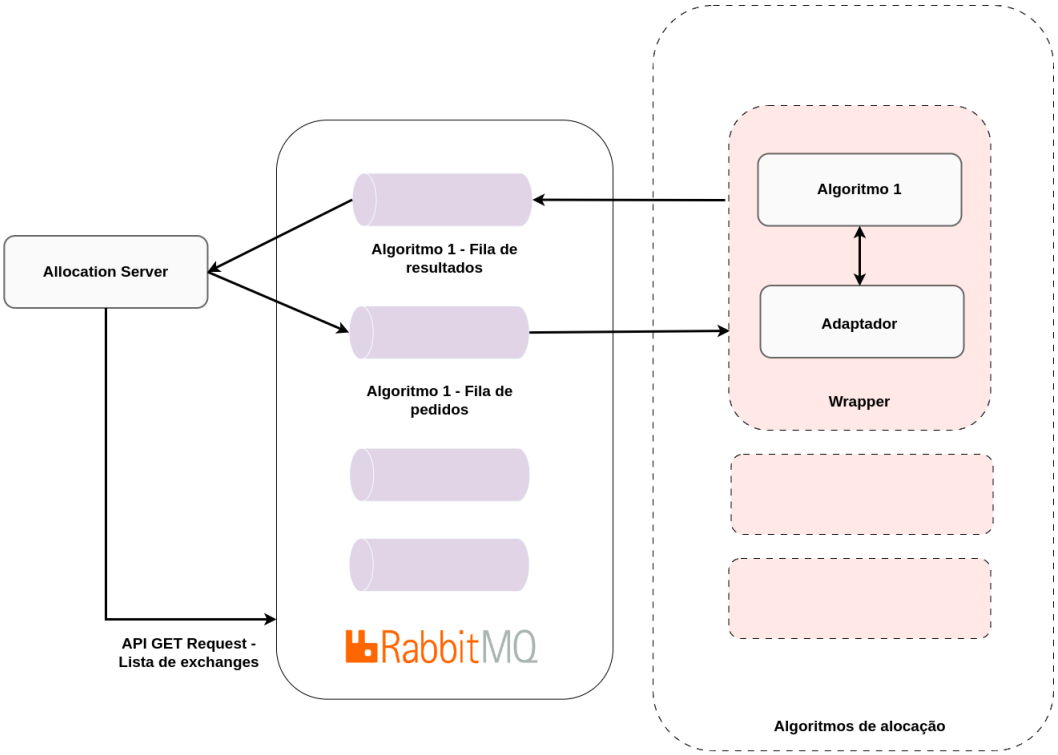
Cada aplicação possui um **adaptador** único responsável por chamar o algoritmo, a responsabilidade deste componente é de garantir que o algoritmo seja chamado e que os dados estejam no formato certo. O motivo disso é que cada algoritmo costuma possuir uma forma única de ser executado e configurado. E nos caso de os dados esperados não estiverem seguindo o formato do contrato de comunicação estabelecido, existiria a necessidade de que os dados sejam traduzidos.

Cada aplicação também teria um ***wrapper***, mas neste caso ele seria igual para todas as aplicações. A responsabilidade deste componente é de fazer a comunicação entre o **adaptador** e a aplicação central. Para conseguir fazer isso, cada *wrapper* cria duas filas, uma para receber os dados da alocação e a outra para enviar o resultado. Assim como foi citado no Capítulo 2, o protocolo usado pelo *RabbitMQ* permite que as aplicações consigam definir os esquemas de roteamento e entidades.

A **aplicação central** precisa saber da existência de cada solução criada no sistema, para conseguir fazer isso, este componente faz um *request* para o *broker* pedindo

a lista de *exchanges*, neste caso, cada *exchange* possui uma ligação direta com uma fila. Este componente também possui a responsabilidade de enviar os pedidos de cálculo para cada solução disponível, assim como, a responsabilidade de receber e tratar os resultados recebidos.

Figura 13 – Comunicação entre os componentes.



Fonte: Autor.

5.2.8 Contrato de comunicação implementado

Conforme foi explicado na Seção 5.2.6, é necessário a existência de um contrato de comunicação entre os componentes. Fora isso, algumas soluções foram propostas, como a USP do (BARICHARD et al., 2022) e o formato usado no (ITC2019, 2023). Para a solução final foi decidido utilizar o formato do (ITC2019, 2023).

Para a tomada desta decisão alguns fatores foram levados em consideração. Os principais fatores que levaram a esta decisão foram a qualidade da documentação oferecida e a facilidade da implementação. No caso da necessidade de implementar um cenário de alocação neste formato esperado, a documentação consegue explicar de forma clara como fazer esta transição. Fora isso, o formato utilizado é intuitivo, facilitando o processo de implementação para este formato.

Outro fator considerado para escolher este formato do ITC-2019 foi a quantidade de soluções existentes que já seguem este formato. Por ter sido um formato utilizado em uma competição real, existem algumas aplicações que já seguem este formato, como por

exemplo o (HOLM; MIKKELSON, 2022), o (LEMOS; MONTEIRO; LYNCE, 2020) e o (MÜLLER, 2022).

Fora isso, este contrato de comunicação parece atender a todas as necessidades consideradas para fazer a alocação de salas definidas pelos requisitos deste projeto.

5.2.9 Formato

O formato do *ITC-2019* é baseado no formato já existente na aplicação *Unitime*, mas foi simplificado para ser usado na competição. Este é um formato em *XML* que permite a alocação de salas a partir da descrição de cursos, alunos, salas e restrições. A Figura 14 mostra um exemplo de uso deste formato, neste caso, está sendo feita a descrição de uma certa quantidade de salas, incluindo a disponibilidade de cada uma e o período de tempo necessário para percorrer uma sala para outra.

A Figura 15 mostra que este formato também permite a descrição de algumas restrições, como por exemplo, o *NotOverlap* e *Precedence*. O *NotOverlap* é uma restrição que não permite que um conjunto de cursos aconteçam no mesmo horário da semana. Por outro lado, o *Precedence* estabelece uma ordem dos cursos, e com isso, exigindo que esta ordem seja seguida.

No entanto, este formato pode apresentar algumas fragilidades. Por ser uma versão simplificada do padrão utilizado no *Unitime*, pode ser que todas as possíveis restrições utilizadas não estejam sendo consideradas neste formato. Como por exemplo, no caso de alguns cursos precisarem compartilhar a mesma sala (MÜLLER, 2022).

Figura 14 – Exemplo da descrição de algumas salas, no formato *ITC-2019*.

```
<rooms>
        <room id="1" capacity="50"/>
        <room id="2" capacity="100">
                <travel room="1" value="2"/>  <!-- travel time is in the number of slots -->
        </room>
        <room id="3" capacity="80">
                <travel room="2" value="3"/>  <!-- only non-zero travel times are present -->
                <!-- not available on Mondays and Tuesdays betwen 8:30 - 10:30, all weeks -->
                <unavailable days="110000" start="102" length="24" weeks="1111111111111"/>
                <!-- not available on Fridays betwen 12:00 - 24:00, odd weeks only -->
                <unavailable days="000100" start="144" length="144" weeks="1010101010101"/>
        </room>
</rooms>
```

Fonte: (ITC2019, 2023).

5.2.10 Algoritmos utilizados

O algoritmo utilizado no sistema foi um algoritmo *open source*, o ***Unitime solver*** (MÜLLER, 2022). Este é um algoritmo que é utilizado pela plataforma *Unitime*, o funcionamento desta plataforma foi explicado na Seção 2.7.

Durante o desenvolvimento deste projeto, houve também a tentativa de utilizar outros algoritmos usados no ITC-2019, como por exemplo o (LEMOS; MONTEIRO;

Figura 15 – Exemplo da descrição de algumas restrições, no formato *ITC-2019*.

```
<distributions>
        <!-- classes 1 and 2 cannot overlap in time -->
        <distribution type="NotOverlap" required="true">
                <class id="1"/>
                <class id="2"/>
        </distribution>
        <!-- class 1 should be before class 3, class 3 before class 5 -->
        <distribution type="Precedence" penalty="2">
                <class id="1"/>
                <class id="3"/>
                <class id="5"/>
        </distribution>
</distributions>
```

Fonte: (ITC2019, 2023).

LYNCE, 2020) e o (GASHI; SYLEJMANI, 2020). Mas por conta da falta de manutenção do projeto ou descrição incompleta de como usar o algoritmo, não foi possível utilizá-los.

5.2.11 Interface *Web*

Componente responsável pela interação com a aplicação, e gerenciar os dados para o sistema de alocação de salas. As tecnologias utilizadas neste componente são o *framework Angular*, e as linguagens *Javascript*, *HTML* e *CSS*. A Figura 31 mostra a forma como o projeto está organizado atualmente.

5.2.11.1 Diagrama de fluxo do usuário

Na Figura 32 é mostrado todo o fluxo do usuário na aplicação. O objetivo deste fluxo é de deixar documentado todo o processo que o usuário deveria realizar na aplicação até conseguir chegar no resultado.

5.2.11.2 Protótipos

A criação deste componente alguns protótipos de baixa fidelidade foram criados. As Figuras 33 e 34 mostram alguns protótipos de baixa fidelidade da aplicação proposta. O objetivo deste protótipo foi de mostrar visualmente uma versão inicial da aplicação. E com isso, conseguir validar a ideia e também coletar opiniões.

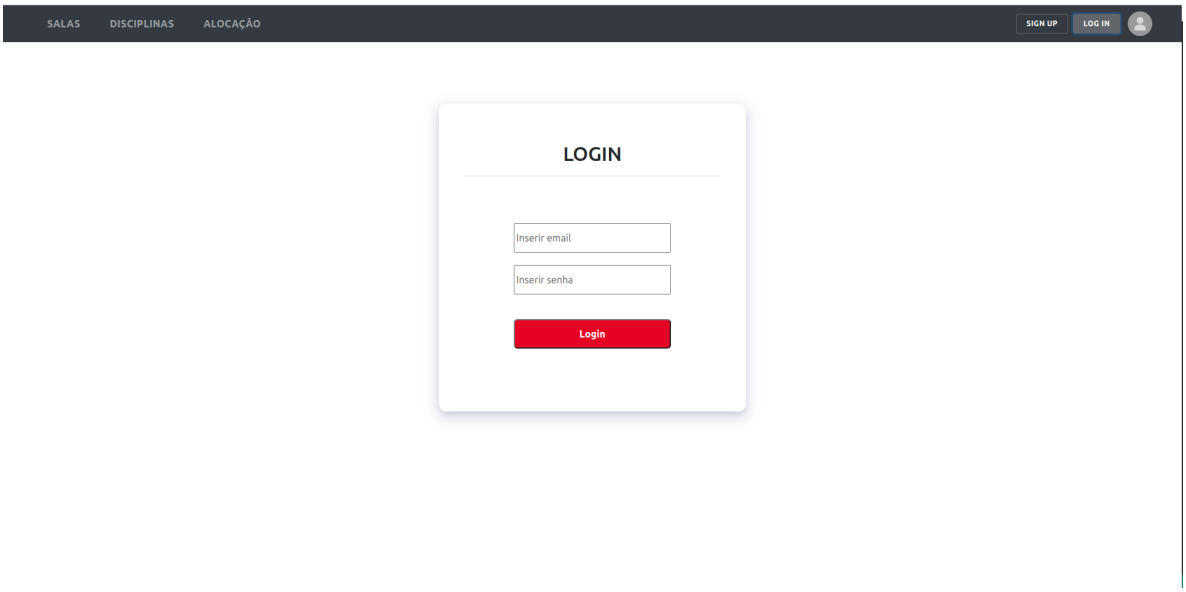
5.2.12 Interface Gráfica

Na interface *web* foram criadas várias telas para conseguir cobrir todos os fluxos existentes do usuário.

5.2.12.1 Criação e login do usuário

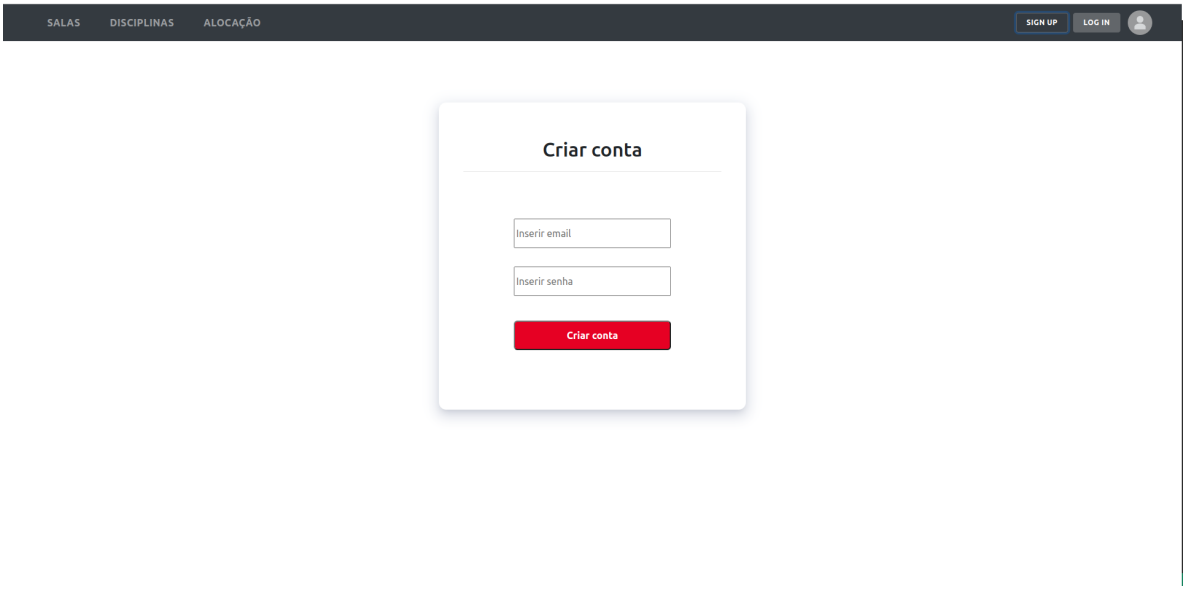
Na Figura 16 é apresentada a primeira tela da aplicação, a página de login. Caso o usuário ainda não tenha uma conta, é necessário ir para a tela que está na Figura 17.

Figura 16 – Tela de login da conta.



Fonte: Autor.

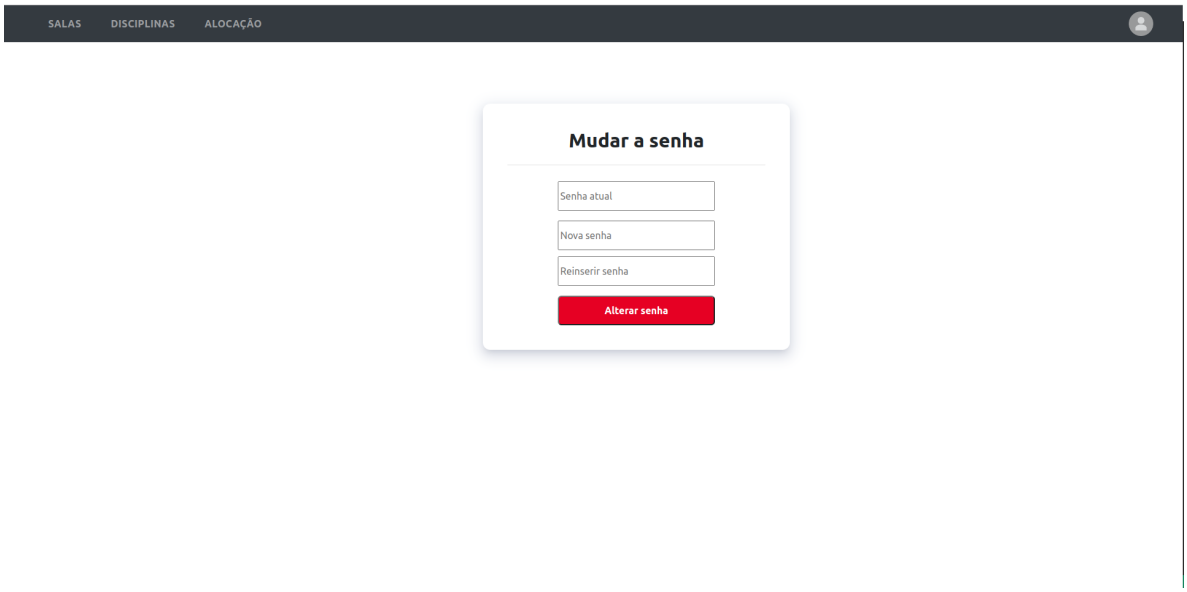
Figura 17 – Tela de criação da conta.



Fonte: Autor.

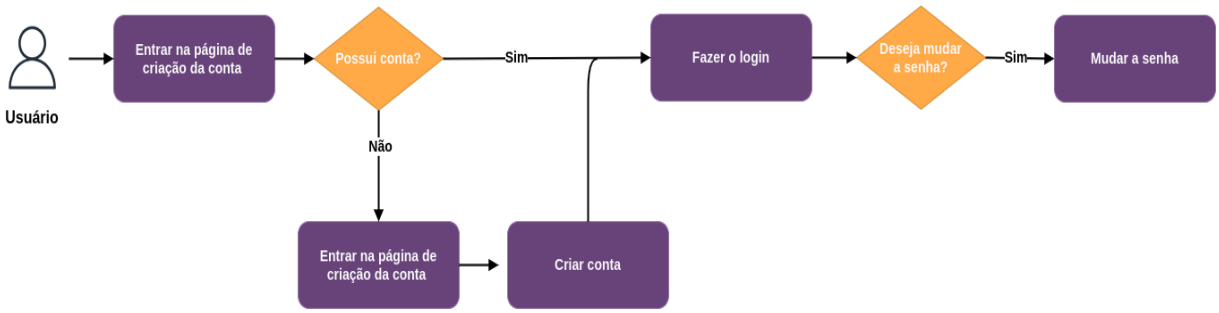
Na Figura 18 mostra que a aplicação também permite que o usuário mude a senha, caso isso seja necessário. A Figura 19 ilustra o fluxo das possíveis etapas para o gerenciamento da conta do usuário.

Figura 18 – Tela de alterar senha.



Fonte: Autor.

Figura 19 – Diagrama de fluxo da parte de gerenciamento da conta do usuário.

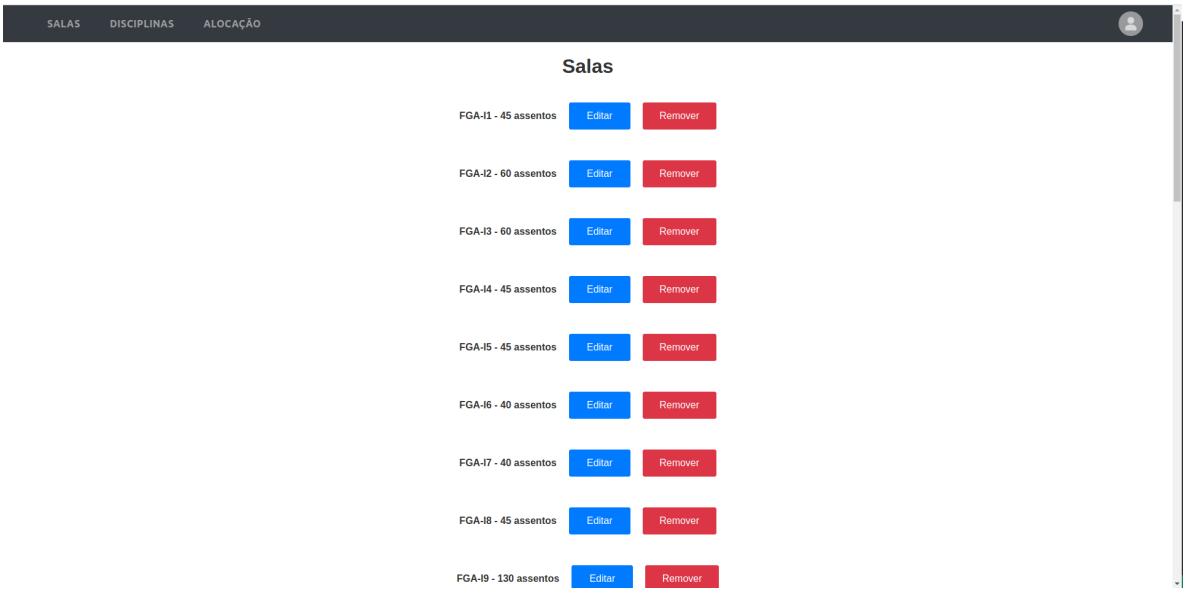


Fonte: Autor.

5.2.12.2 Criação e edição das salas

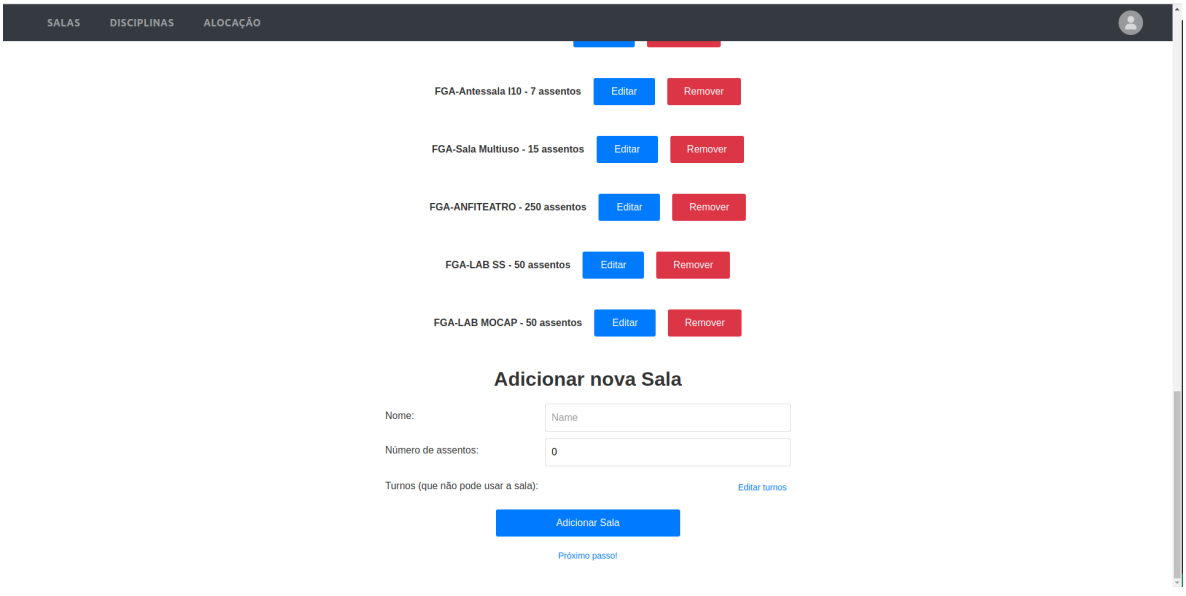
Na Figura 20 mostra o resultado da criação das salas, e na Figura 21 a forma como uma sala pode ser inserida. A interface também possibilita que o usuário consiga editar alguma informação da sala, como é demonstrado na Figura 22. A Figura 23 ilustra como este fluxo de gerenciamento de salas funciona.

Figura 20 – Resultado das salas.



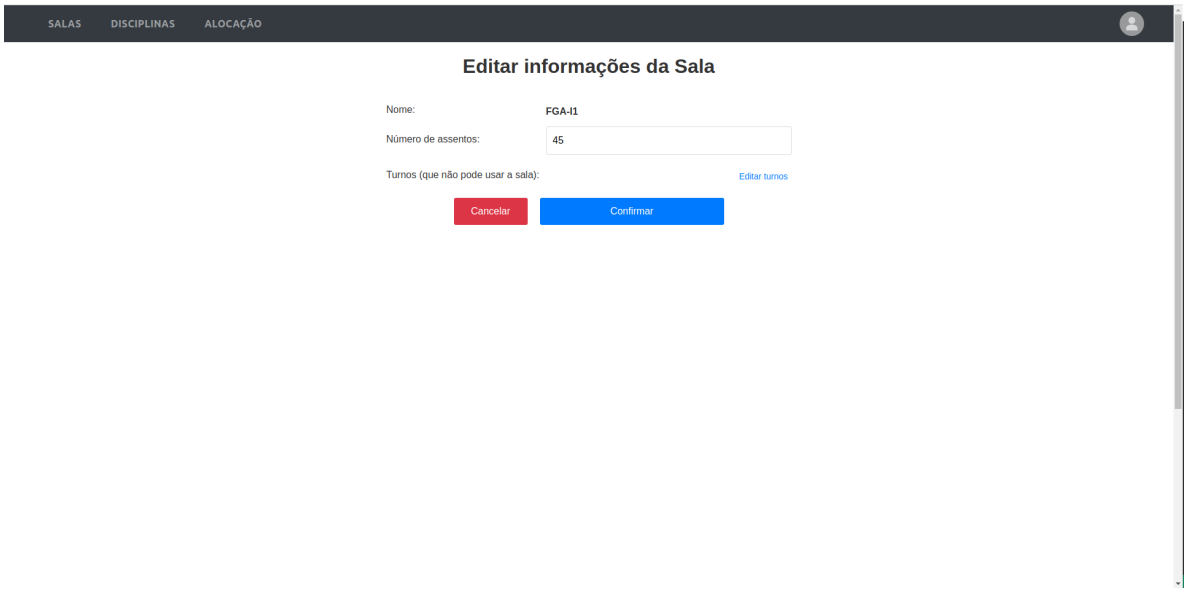
Fonte: Autor.

Figura 21 – Criação de uma sala.



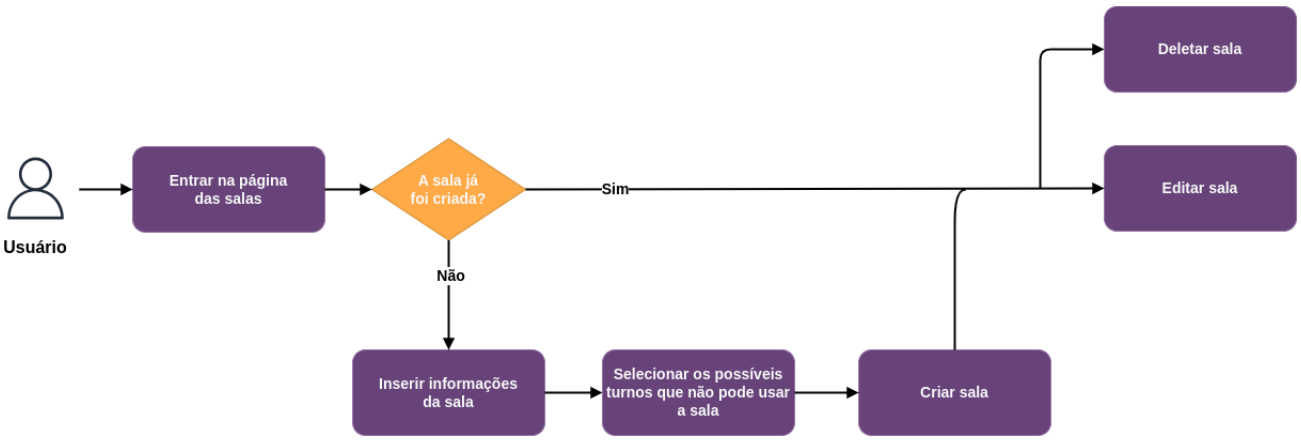
Fonte: Autor.

Figura 22 – Tela de editar informações de uma certa sala.



Fonte: Autor.

Figura 23 – Diagrama de fluxo da parte de gerenciamento das salas.



Fonte: Autor.

5.2.12.3 Criação e edição das disciplinas

A Figura 24 mostra os resultados das disciplinas criadas, seguindo o mesmo formato estabelecido na página de salas, Figura 18. A tela da Figura 25 mostra a tela de edição das informações de uma certa disciplina. A Figura 26 mostra a parte de edição dos turnos. A Figura 27 ilustra como este fluxo de gerenciamento de salas funciona.

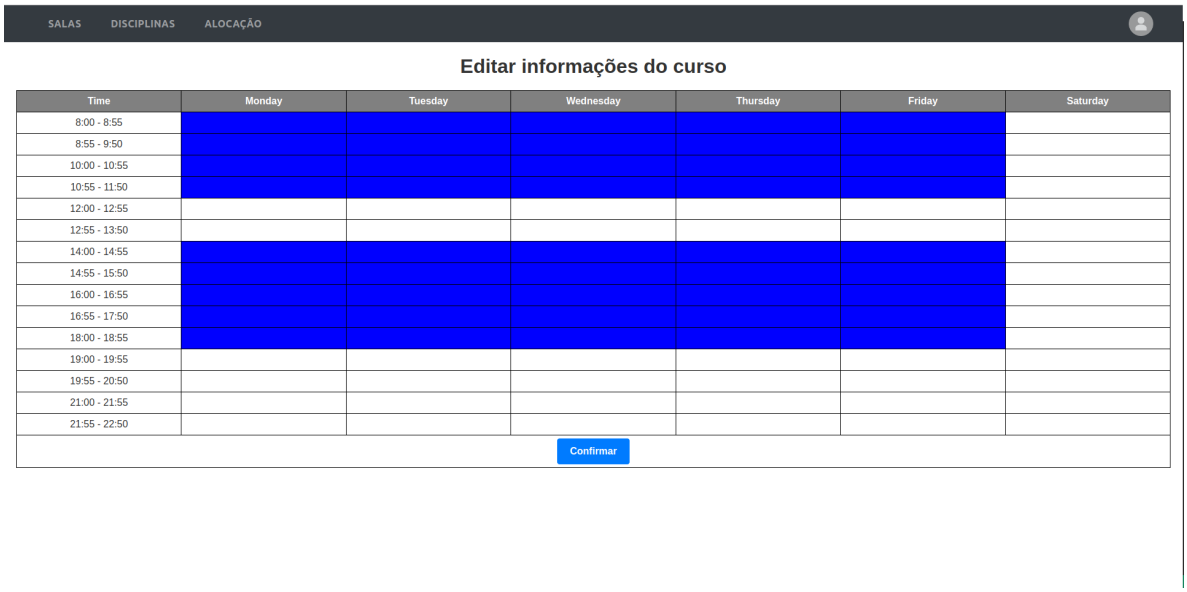
Figura 24 – Resultado das disciplinas criadas.

Fonte: Autor.

Figura 25 – Tela de editar informações de uma certa disciplina.

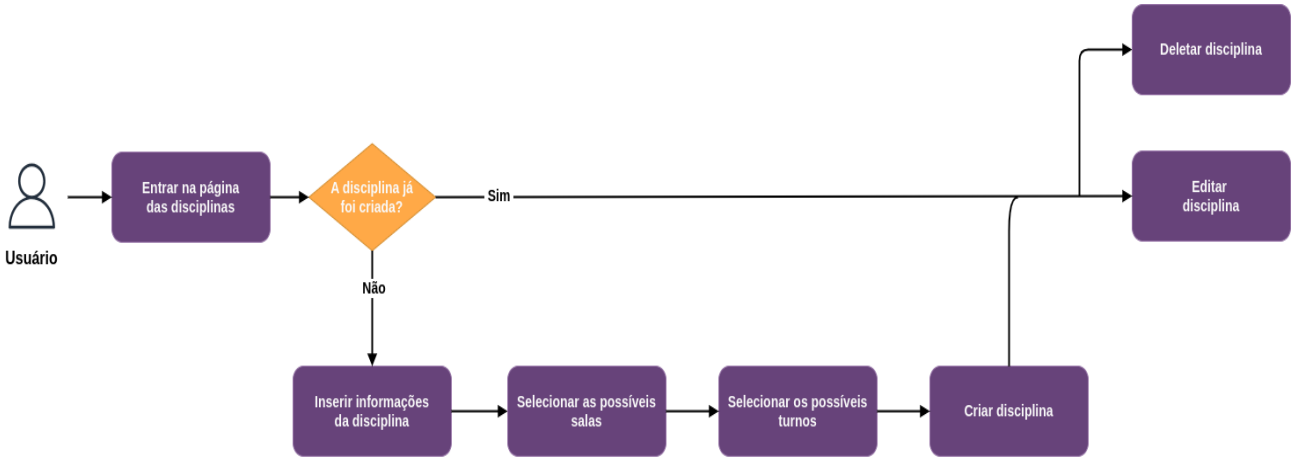
Fonte: Autor.

Figura 26 – Tela de edição dos turnos de um certa disciplina.



Fonte: Autor.

Figura 27 – Diagrama de fluxo da parte de gerenciamento das disciplinas.



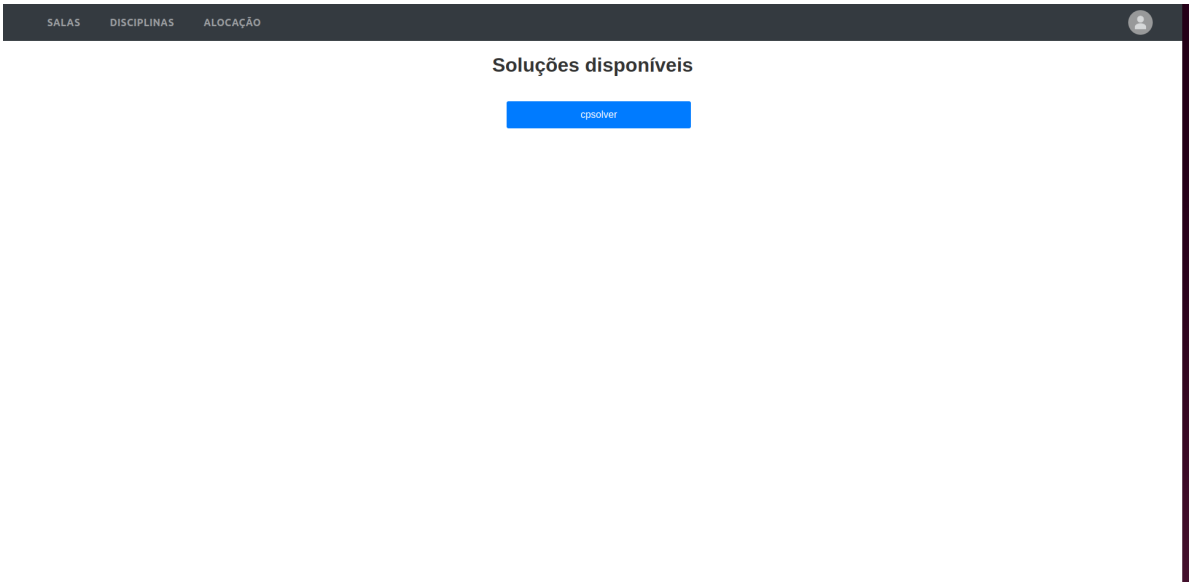
Fonte: Autor.

5.2.12.4 Alocação

Na Figura 28 mostra a página de alocações, onde é possível ver as soluções disponíveis para serem utilizadas pelo usuário. Caso o usuário selecione uma dessas opções, a aplicação vai chamar o algoritmo para começar a fazer o cálculo. O resultado deste cálculo é mostrado em uma outra página, na Figura 29 é possível ver um exemplo do resultado retornado pelo *Unitime Solver*, (MÜLLER, 2022).

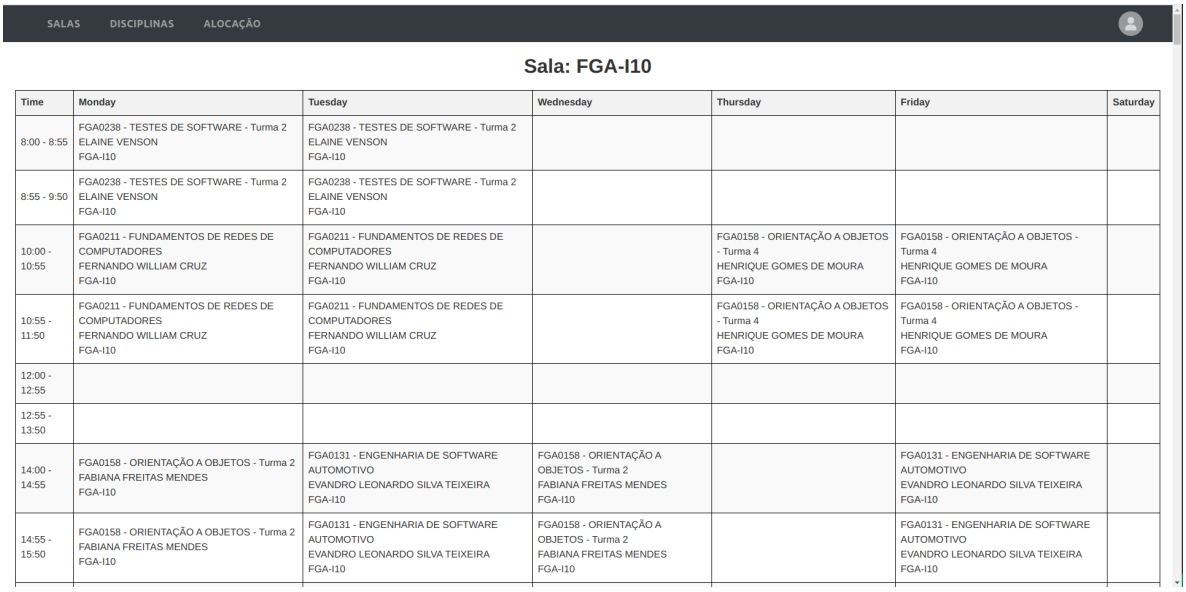
É importante destacar que na Figura 29, a solução encontrada serve apenas para fins de ilustração, não fazendo parte do escopo deste trabalho a análise de viabilidade e aplicabilidade desta solução.

Figura 28 – Tela das soluções de algoritmo disponíveis.



Fonte: Autor.

Figura 29 – Tela do resultado do cálculo feito por um certo algoritmo.



Time	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
8:00 - 8:55	FGA0238 - TESTES DE SOFTWARE - Turma 2 ELAINE VENSON FGA-I10	FGA0238 - TESTES DE SOFTWARE - Turma 2 ELAINE VENSON FGA-I10				
8:55 - 9:50	FGA0238 - TESTES DE SOFTWARE - Turma 2 ELAINE VENSON FGA-I10	FGA0238 - TESTES DE SOFTWARE - Turma 2 ELAINE VENSON FGA-I10				
10:00 - 10:55	FGA0211 - FUNDAMENTOS DE REDES DE COMPUTADORES FERNANDO WILLIAM CRUZ FGA-I10	FGA0211 - FUNDAMENTOS DE REDES DE COMPUTADORES FERNANDO WILLIAM CRUZ FGA-I10		FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 4 HENRIQUE GOMES DE MOURA FGA-I10	FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 4 HENRIQUE GOMES DE MOURA FGA-I10	
10:55 - 11:50	FGA0211 - FUNDAMENTOS DE REDES DE COMPUTADORES FERNANDO WILLIAM CRUZ FGA-I10	FGA0211 - FUNDAMENTOS DE REDES DE COMPUTADORES FERNANDO WILLIAM CRUZ FGA-I10		FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 4 HENRIQUE GOMES DE MOURA FGA-I10	FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 4 HENRIQUE GOMES DE MOURA FGA-I10	
12:00 - 12:55						
12:55 - 13:50						
14:00 - 14:55	FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 2 FABIANA FREITAS MENDES FGA-I10	FGA0131 - ENGENHARIA DE SOFTWARE AUTOMOTIVO EVANDRO LEONARDO SILVA TEIXEIRA FGA-I10	FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 2 FABIANA FREITAS MENDES FGA-I10		FGA0131 - ENGENHARIA DE SOFTWARE AUTOMOTIVO EVANDRO LEONARDO SILVA TEIXEIRA FGA-I10	
14:55 - 15:50	FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 2 FABIANA FREITAS MENDES FGA-I10	FGA0131 - ENGENHARIA DE SOFTWARE AUTOMOTIVO EVANDRO LEONARDO SILVA TEIXEIRA FGA-I10	FGA0158 - ORIENTAÇÃO A OBJETOS - Turma 2 FABIANA FREITAS MENDES FGA-I10		FGA0131 - ENGENHARIA DE SOFTWARE AUTOMOTIVO EVANDRO LEONARDO SILVA TEIXEIRA FGA-I10	

Fonte: Autor.

5.3 Guia de como criar *Plug-ins*

A parte de *plugins* foi feita de acordo com a arquitetura proposta na Seção 5.2. A implementação desta arquitetura permite a adição, em tempo de execução da aplicação, de novos algoritmos.

No caso da adição de um novo *plugin*, seria necessário a utilização de dois componentes principais: o **adaptador** e o ***wrapper***. Considerando esta possível necessidade da adição de novos *plugins*, foram criados modelos dos arquivos do *wrapper* e do adaptador

no projeto. Sendo assim, não é necessário criar esses componentes a partir do zero, uma vez que seria possível se basear nesses arquivos modelos existentes.

Os códigos referentes a esses arquivos modelos estão disponíveis no repositório do projeto no *Github*, e também estão anexados no Anexo B e C. Esses códigos existentes foram usados para integrar com o algoritmo do *Unitime Solver*, mas podem ser usados como modelo para a integração com outros tipos de algoritmos.

O ***wrapper*** modelo pode ser utilizado em qualquer solução. No caso da necessidade de reutilização para uma novo algoritmo, seria necessário apenas mudar o nome do arquivo, pois o nome é utilizado pela aplicação central como identificador do algoritmo. O *wrapper* fica responsável por manter a conexão do *plugin* com a aplicação central, e também pelo transporte das informações.

O **adaptador** é o mesmo utilizado no *plugin* do algoritmo *Unitime Solver*, e pode ser usado de exemplo para outros algoritmos. O adaptador é único para cada solução, sendo sua responsabilidade conseguir fazer a comunicação do *wrapper* com o algoritmo. Caso o algoritmo não esteja seguindo o formato do ITC-2019, será necessário modificar o adaptador para converter as informações recebidas da aplicação central para o formato esperado pelo algoritmo, e também transformar as informações geradas pelo algoritmo no formato do ITC-2019.

5.3.1 Passos para Implementação

Na prática, caso seja necessário criar um novo *plugin* no projeto, é possível reutilizar esses exemplos de componentes para integrar o novo algoritmo à aplicação. O fluxo definido na Figura 30 mostra todos os passos necessário para inserção de algum algoritmo no sistema como *plugin*.

Para a criação do *wrapper*, o arquivo de exemplo do *wrapper* poderia ser usado, apenas copiando esse arquivo e modificando o nome do arquivo para algum nome que identifique o algoritmo solução.

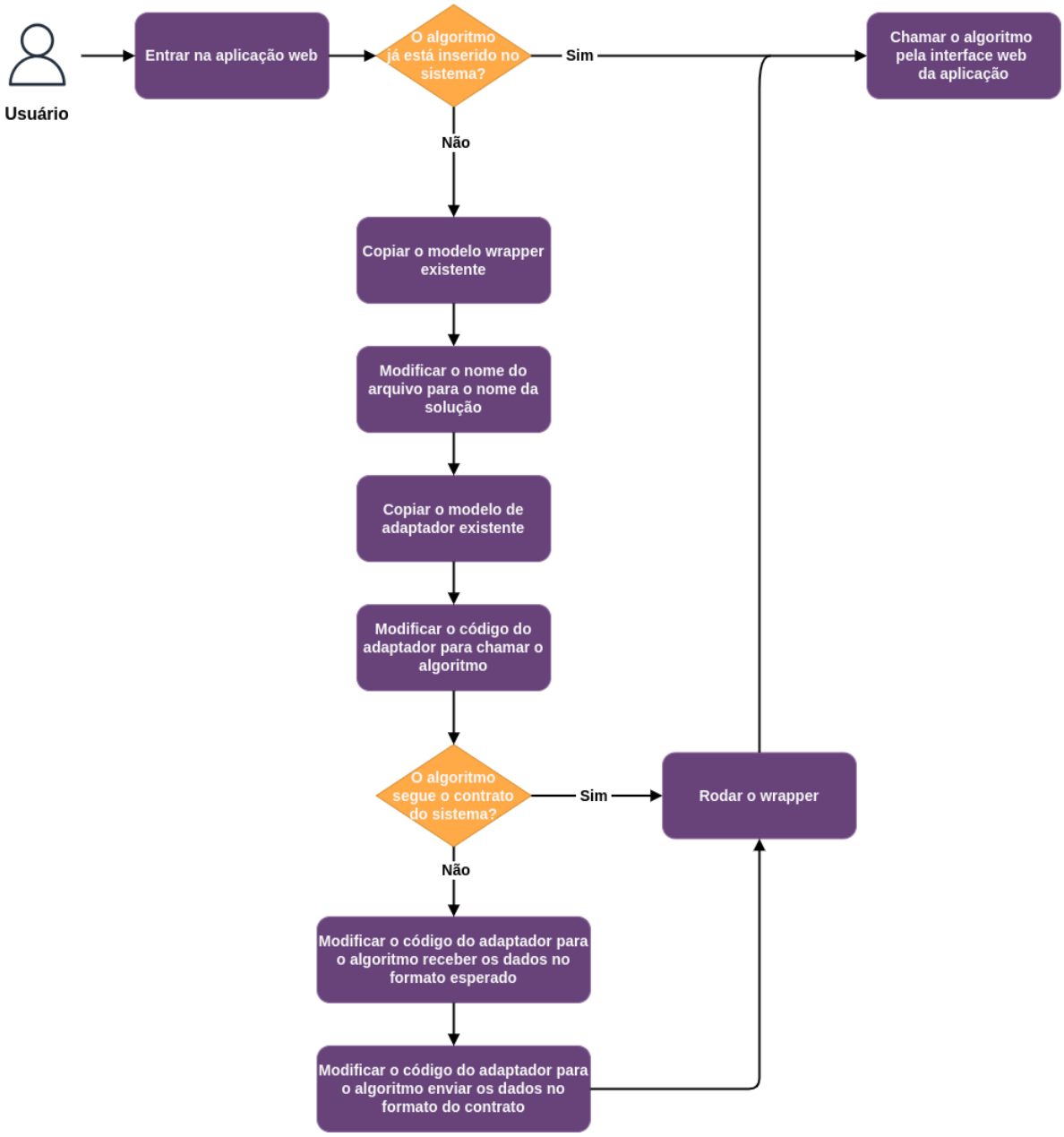
Para a criação do adaptador, o exemplo modelo do adaptador também poderia ser reutilizado, com alterações no formato das informações, nos comandos para chamar o algoritmo e também passar as informações na forma como o algoritmo espera receber. Por exemplo, se o algoritmo espera receber o resultado de um arquivo, como é o caso do *Unitime Solver*. Nesse cenário, seria preciso modificar o código que está no adaptador para criar esse arquivo, e também para transmitir as informações necessárias para o formato esperado pelo algoritmo. Sendo assim, se os dados não estiverem em um formato em que o algoritmo espera, como o formato ITC-2019, teria que fazer esta tradução dentro do adaptador.

Dessa forma, o *wrapper* possui a responsabilidade de chamar o adaptador e esta-

belecer a comunicação com o algoritmo. Além disso, para que a integração ocorra sem problemas, o *wrapper* precisa ser capaz de chamar o adaptador.

De acordo com a Seção 5.2.9, a aplicação receberá os dados no formato padrão ITC-2019, e os dados do resultado devem também retornar no mesmo formato. A partir deste formato, o algoritmo consegue receber os dados para a alocação, como por exemplo, as salas e disciplinas que foram criadas pela interface *web*.

Figura 30 – Diagrama de fluxo da parte de inserção de um novo algoritmo no sistema.



Fonte: Autor.

6 Conclusão

O processo de alocação de salas costuma ser muito trabalhoso e gasta muito tempo, pois existe todo um processo manual que deve ser feito. Fora isso, atualmente não existe uma solução única que consiga permitir a utilização de diferentes algoritmos.

É importante resolver esse problema, pois isto irá facilitar e agilizar o trabalho dos servidores que fazem a alocação de salas. E com isso, tornar este processo mais eficiente. Sendo assim, o objetivo desta proposta de TCC é oferecer uma solução para esse problema.

Segundo a Seção 1.1, a proposta criada neste TCC é a criação de uma aplicação *web* que consiga coletar todas as informações necessárias e com isso fazer a alocação de salas, podendo utilizar algoritmos diferentes para resolver o problema.

6.1 Conclusão dos objetivos

- Implementar a interface do sistema
  Este objetivo pode ser considerado como concluído, pois foi criado uma nova interface web que consegue fazer a alocação das salas.

- Integrar com algum algoritmo de alocação existente
  Este objetivo pode ser considerado como concluído. Houve a integração do *Unitime Solver* (MÜLLER, 2022).

- Criar um sistema de *plugin* para facilitar a integração de diferentes soluções
  Concluído, pois foi possível propor e implementar um sistema de *plugin* que facilita a integração de diferentes algoritmos disponíveis.

6.2 Trabalhos futuros

Tendo em vistas os possíveis trabalhos futuros que podem surgir, os seguintes itens podem servir como possíveis pontos de aprimoramento:

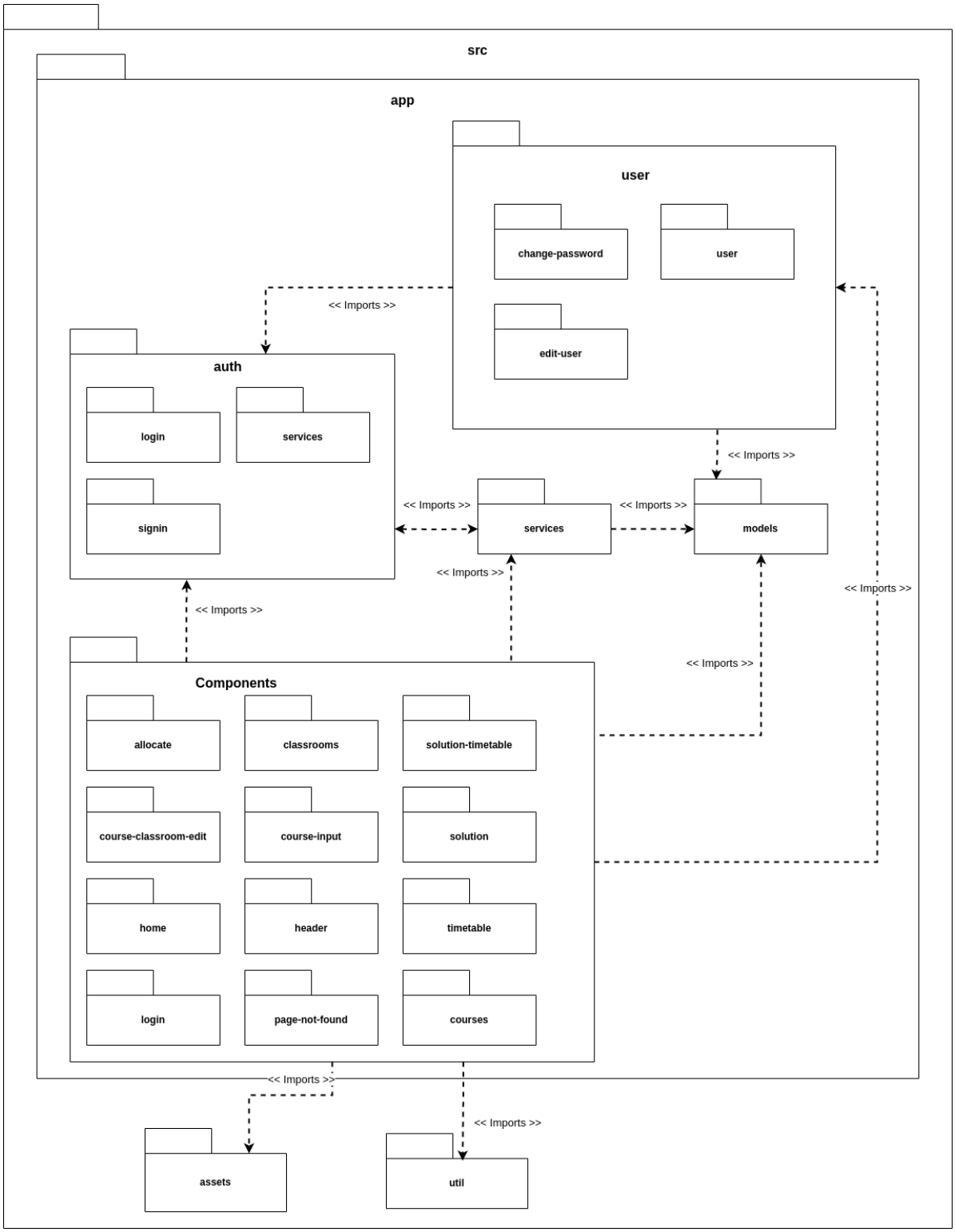
- Melhorar a orquestração da comunicação da aplicação central com os *plugins*
- Fazer a integração de mais algum algoritmo no sistema
- Salvar os resultados em algum lugar
- Permitir a descrição das restrições por meio da interface de usuário

- Considerar usar o USP (BARICHARD et al., 2022) para fazer o contrato de comunicação
  Este é um modelo *open source* que ainda está em desenvolvimento, além disso, os desenvolvedores deste modelo criaram algumas ferramentas de *parsing* que facilitam a criação de adaptadores.

Anexos

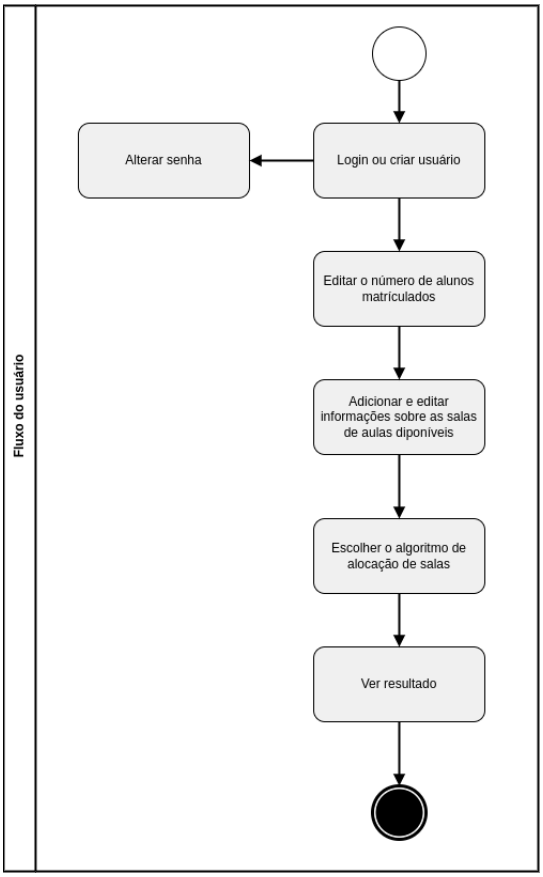
ANEXO A – Interface *Web*

Figura 31 – Diagrama de pacotes da Interface Web.



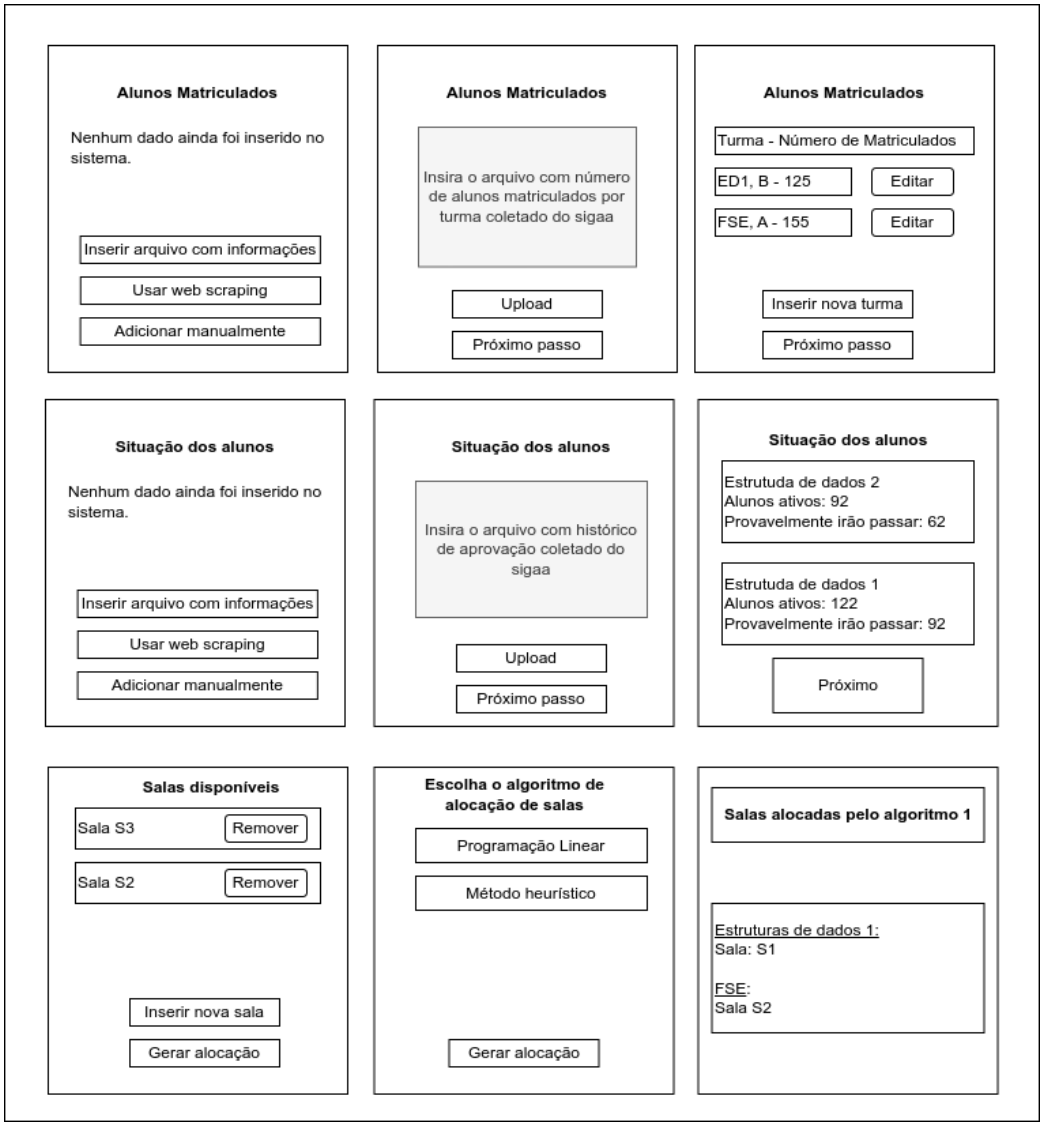
Fonte: Autor.

Figura 32 – Fluxo do usuário na aplicação.



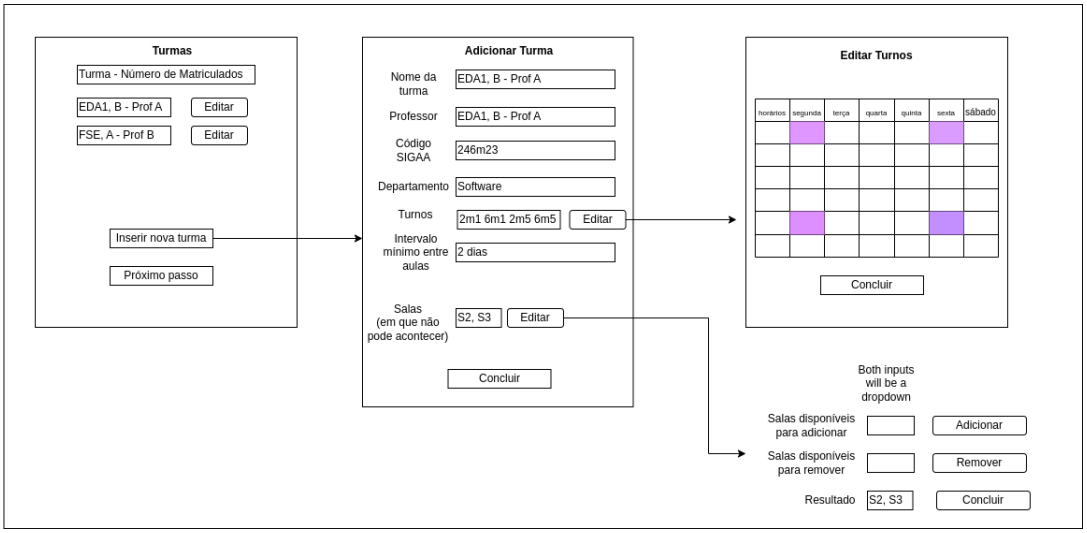
Fonte: Autor.

Figura 33 – Protótipo de baixa fidelidade da aplicação.



Fonte: Autor.

Figura 34 – Protótipo de baixa fidelidade do fluxo de adição de turmas.



Fonte: Autor.

ANEXO B – Código de Exemplo de um Adaptador

Figura 35 – Código existente no repositório do projeto de um adaptador que foi implementado.

```python
import subprocess
import os

class CommandRunner:
    def __init__(self):
        self.output_dir = './output'
        self.instances_dir = './instances'

    def run_java_jar_command(self):
        command = "java -jar target/cpsolver-itc2019-1.0-SNAPSHOT.jar configuration/default.cfg instances/input.xml " + self.output_dir
        subprocess.run(command, shell=True)

    def get_last_created_folder(self):
        folders = sorted([folder for folder in os.listdir(self.output_dir) if os.path.isdir(os.path.join(self.output_dir, folder))])
        if folders:
            last_folder = folders[-1]
            return os.path.join(self.output_dir, last_folder)
        return None

    def get_file_content(self, file_path):
        with open(file_path, "r") as f:
            return f.read()

    def get_last_created_folder_files(self):
        last_folder = self.get_last_created_folder()
        if last_folder:
            solution_xml_path = os.path.join(last_folder, "solution.xml")
            if os.path.isfile(solution_xml_path):
                file_content = self.get_file_content(solution_xml_path)
                return file_content
        return None

    def create_input_file(self, body):
        input_file_path = os.path.join(self.instances_dir, 'input.xml')

        # Write the body content to the input file
        with open(input_file_path, 'w') as f:
            f.write(body.decode('utf-8'))

    def execute(self, body):
        self.create_input_file(body)
        self.run_java_jar_command()
        result = self.get_last_created_folder_files()
        if result:
            return result
        else:
            return "No solution.xml file found in the last created folder."
```

Fonte: Autor.

ANEXO C – Código de Exemplo de um *Wrapper*

Figura 36 – Primeira parte do código de exemplo de um *wrapper* que está no repositório do projeto.

```python
import os
import pika
from command_runner import CommandRunner

file_name = os.path.basename(__file__)
prefix = "wrapper_"
suffix = ".py"
if file_name.startswith(prefix):
    SOLVER_NAME = file_name[len(prefix):]
    SOLVER_NAME = SOLVER_NAME.split(suffix)[0]
else:
    SOLVER_NAME = "solver1"

print(f"Running: {SOLVER_NAME}")

def connect_rabbitmq():
    credentials = pika.PlainCredentials('user', 'password')
    parameters = pika.ConnectionParameters('localhost', credentials=credentials, port=5672)
    connection = pika.BlockingConnection(parameters)
    channel = connection.channel()
    return connection, channel

def declare_exchanges(solver_name, channel):
    exchange_names = ['new_solver_exchange', f'{solver_name}_start_calculating_exchange', f'{solver_name}_result_exchange']
    for exchange_name in exchange_names:
        channel.exchange_declare(exchange=exchange_name, exchange_type='fanout')
        create_and_bind_queue(channel, exchange_name)

def create_and_bind_queue(channel, exchange_name):
    queue_name = f"{exchange_name}_queue"
    channel.queue_declare(queue=queue_name)
    channel.queue_bind(exchange=exchange_name, queue=queue_name)
    return queue_name

def consume_messages(channel, queue_name, connection):
    def callback(ch, method, properties, body):
        print(f"Received message: {body}")

        # wait 5 seconds and send a message to the result exchange
        runner = CommandRunner()
        message = runner.execute(body)

        publish_message(channel, f'{SOLVER_NAME}_result_exchange', message)

    channel.basic_consume(queue=queue_name, on_message_callback=callback, auto_ack=True)

    try:
        channel.start_consuming()
    except KeyboardInterrupt:
```

Fonte: Autor.

Figura 37 – Segunda parte do código de exemplo de um *wrapper* que está no repositório do projeto.

```python
def publish_message(channel, exchange_name, message):
    channel.basic_publish(exchange=exchange_name, routing_key='', body=message)

def close_connection(connection):
    print("Closing connection...")
    connection.channel().queue_delete(queue=f'{SOLVER_NAME}_start_calculating_exchange_queue')
    connection.channel().exchange_delete(exchange=f'{SOLVER_NAME}_start_calculating_exchange')
    connection.channel().exchange_delete(exchange=f'{SOLVER_NAME}_result_exchange')

def main():
    connection, channel = connect_rabbitmq()

    declare_exchanges(SOLVER_NAME, channel)

    message = f'{SOLVER_NAME}_start_calculating_exchange'
    channel.basic_publish(exchange='new_solver_exchange', routing_key='', body=message)
    channel.basic_publish(exchange=f'{SOLVER_NAME}_result_exchange', routing_key='', body='')

    queue_name = f'{SOLVER_NAME}_start_calculating_exchange_queue'
    consume_messages(channel, queue_name, connection)

    close_connection(connection)

if __name__ == "__main__":
    main()
```

Fonte: Autor.

Referências

AL-SAQQA, S.; SAWALHA, S.; ABDELNABI, H. Agile software development: Methodologies and trends. 2020. Citado na página 35.

ANGULAR. *What is Angular?* 2023. Disponível em: <https://angular.io/guide/what-is-angular>. Citado na página 29.

APPLE. *Plug-in Architectures*. 2023. Disponível em: <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/LoadingCode/Concepts/Plugins.html>. Citado na página 23.

BARICHARD, V. et al. A constraint language for university timetabling problems. 2022. Citado 5 vezes nas páginas 18, 42, 43, 44 e 58.

CIRINO, R. B. Z. Abordagens de solução para o problema de alocação de aulas a salas. 2016. Disponível em: <https://teses.usp.br/teses/disponiveis/55/55134/tde-16112016-142336/publico/RafaelBernardoZanettiCirino_revisada.pdf>. Citado 2 vezes nas páginas 19 e 20.

COCCO, L. et al. Simulating kanban and scrum vs. waterfall with system dynamics. 2011. Citado 2 vezes nas páginas 35 e 36.

EXPRESS. *Express - Fast, unopinionated, minimalist web framework for Node.js*. 2023. Disponível em: <https://expressjs.com/>. Citado na página 29.

FREITAS, E. C. P. C. C. . Metodologia do trabalho científico: Métodos e técnicas da pesquisa e do trabalho acadêmico. 2013. Citado na página 35.

GARDENGHI, J. L. Depoimento [Entrevista cedida a] Paulo Batista. Entrevista com o Prof. John L. Gardenghi, coordenador em exercício em Março/2023 (período de matrícula do semestre 2023.1 na UnB). 2023. Citado na página 17.

GASHI, E.; SYLEJMANI, K. Simulated annealing with penalization for university course timetabling. 2020. Citado na página 46.

GERHARDT T. E.; SILVEIRA, D. T. Métodos de pesquisa. 2009. Disponível em: <https://lume.ufrgs.br/handle/10183/52806>. Citado 3 vezes nas páginas 33, 34 e 35.

GIL, A. C. Métodos e técnicas de pequisa social. 2008. Citado na página 33.

GIT. *Git - Fast Version Control System*. 2023. Disponível em: <https://git-scm.com/>. Citado na página 30.

GROUP, P. G. D. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. 2023. Disponível em: <https://www.postgresql.org/>. Citado na página 30.

HOLM, D.; MIKKELSON, R. A parallelized matheuristic for the international timetabling competition 2019. 2022. Citado na página 45.

ITC2019, I. T. C. *ITC 2019: International Timetabling Competition*. 2023. Disponível em: <https://www.itc2019.org/>. Citado 6 vezes nas páginas 18, 42, 43, 44, 45 e 46.

LEGASPI, J. et al. Web based course scheduling system using greedy algorithm. 2012. Citado na página 27.

LEMOS, A.; MONTEIRO, T. P.; LYNCE, I. Itc 2019: University course timetabling with maxsat. 2020. Citado 2 vezes nas páginas 45 e 46.

MÜLLER, T. Itc 2019: Results using the unitime solver. 2022. Citado 4 vezes nas páginas 42, 45, 52 e 57.

NAVUDURI, S. Design and implementation of a classroom allocation system prototype. 2016. Citado na página 27.

NEWMAN, S. Building microservices: Designing fine-grained systems . 2015. Citado 2 vezes nas páginas 24 e 25.

NODE.JS. *About Node.js*. 2023. Disponível em: <https://nodejs.org/en/about/>. Citado na página 29.

RABBITMQ. *AMQP 0-9-1 Model Explained*. 2023. Disponível em: <https://www.rabbitmq.com/tutorials/amqp-concepts.html>. Citado 4 vezes nas páginas 24, 25, 26 e 27.

RABBITMQ. *What can RabbitMQ do for you?* 2023. Disponível em: <https://www.rabbitmq.com/features.html>. Citado na página 30.

REDHAT. *GraphQL: linguagem de consulta para APIs*. 2023. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-graphql>. Citado na página 30.

RICHARDS, M. Software architecture patterns. 2015. Citado 4 vezes nas páginas 22, 23, 24 e 25.

SALES, E. S. Problema de alocação de salas e a otimização dos espaços no centro de tecnologia da ufsm. 2015. Disponível em: <https://repositorio.ufsm.br/handle/1/4751>. Citado na página 19.

SAVINIEC, L. Models and algorithms for high school timetabling problems. 2017. Citado na página 19.

SHVETS, A. Dive into design patterns. 2019. Citado 2 vezes nas páginas 20 e 21.

SILVA, L. F. A. C. Modelo de programação linear inteira para o problema de alocação de salas: Estudo de caso em uma instituição de ensino superior. 2019. Disponível em: <https://repositorio.ufpb.br/jspui/bitstream/123456789/19205/1/LucianoFernandesAcioliCabralESilva_Dissert.pdf>. Citado 2 vezes nas páginas 17 e 19.

SUBRAMANIAN, A. et al. Aplicação da metaheurística busca tabu ao problema de alocação de aulas a salas em uma instituição universitária. 2011. Citado na página 19.

UNITIME. *University Timetabling System Highlights*. 2023. Disponível em: <https://www.unitime.org/unitime_intro.php>. Citado na página 27.