UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

GABRIEL NOGUEIRA BEZERRA

DESENVOLVIMENTO DE UM BACKEND PARA AGENDAR HORÁRIOS E SALAS NA UNIVERSIDADE FEDERAL DO CEARÁ CAMPUS RUSSAS

RUSSAS

2023

GABRIEL NOGUEIRA BEZERRA

DESENVOLVIMENTO DE UM BACKEND PARA AGENDAR HORÁRIOS E SALAS NA UNIVERSIDADE FEDERAL DO CEARÁ CAMPUS RUSSAS

Trabalho de Conclusão de Curso apresentado ao Curso de bacharelado em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Alexandre Matos Arruda

RUSSAS

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

B469d Bezerra, Gabriel Nogueira.
Desenvolvimento de um backend para agendar horários e salas na universidade federal do Ceará campus Russas / Gabriel Nogueira Bezerra. – 2023.
56 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2023.
Orientação: Prof. Dr. Alexandre Matos Arruda.

1. Alocação de horários. 2. Grade de horários. 3. Satisfazer Horários. I. Título.

CDD 005.1

GABRIEL NOGUEIRA BEZERRA

DESENVOLVIMENTO DE UM BACKEND PARA AGENDAR HORÁRIOS E SALAS NA UNIVERSIDADE FEDERAL DO CEARÁ CAMPUS RUSSAS

Trabalho de Conclusão de Curso apresentado ao Curso de bacharelado em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: 18 de julho de 2023

BANCA EXAMINADORA

Prof. Dr. Alexandre Matos Arruda (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Ms. José Osvaldo Mesquita Chaves
Universidade Federal do Ceará (UFC)

Prof. Dr. Dmontier Pinheiro Aragão Junior
Universidade Federal do Ceará (UFC)

Agradeço a Deus pois sem ele eu não teria forças para concluir essa etapa da minha vida, à minha mãe, por sempre me apoiar e incentivar em todas as decisões que tomei em minha vida. E aos amigos que me ajudaram nessa trajetória de vida.

AGRADECIMENTOS

Ao Prof. Dr. Alexandre Matos Arruda por me orientar nesse Trabalho.

Aos meus pais, familiares, que nos momentos de minha ausência dedicados ao estudo superior, sempre fizeram entender que o futuro é feito a partir da constante dedicação no presente!

Aos amigos que encontrei nesse caminho que me auxiliaram e ajudaram nas disciplinas e trabalhos da faculdade.

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender.

“Sempre que você vir uma pessoa de sucesso, você sempre verá as glórias, nunca os sacrifícios que os levaram até ali.”

(Vaibhav Shah)

RESUMO

As instituições de ensino devem fornecer aos seus discentes e docentes uma relação de turmas e horários que atenda a todos os requisitos estabelecidos, sem conflitos e de maneira eficiente. Isso implica em otimizar o uso do tempo e dos recursos disponíveis. No entanto, a alocação de horários apresenta desafios, exigindo muito tempo e atenção, e qualquer erro pode afetar toda a grade já estabelecida. Este trabalho descreve o desenvolvimento de uma estrutura backend para inserção de dados e alocação de turmas e horários na Universidade Federal do Ceará Campus de Russas. Utilizando o método Design Science, foram realizadas cinco etapas de desenvolvimento, desde a investigação do problema até a avaliação da implementação. O resultado desta pesquisa é uma estrutura de *backend* pronta, que permite uma transição fluida para os desenvolvedores que assumirem o projeto. Além disso, proporciona aos profissionais responsáveis pelo frontend uma experiência de desenvolvimento fácil e ágil, aproveitando a base sólida estabelecida neste trabalho. Esse trabalho também oferece a capacidade de resolver situações de problemas com restrições flexíveis, o que possibilita maior flexibilidade e adaptabilidade na alocação de turmas e horários, atendendo às necessidades específicas da instituição de ensino.

Palavras-chave: alocação de horários; grade de horários; satisfazer horários

ABSTRACT

Educational institutions must provide students and teachers with a list of courses and schedules that meets all established requirements, without conflicts and in an efficient manner. This implies optimizing the use of time and available resources. However, scheduling presents challenges, requiring a lot of time and attention, and any error can affect the entire schedule that has already been established. This work describes the development of a backend structure for data insertion and scheduling of courses and schedules at the Federal University of Ceará Campus of Russas. Using the Design Science method, five development stages were carried out, from problem investigation to implementation evaluation. The result of this research is a ready-made backend structure, which allows a smooth transition for developers who take over the project. In addition, it provides professionals responsible for the frontend with an easy and agile development experience, taking advantage of the solid foundation established in this work. This work also offers the ability to solve problem situations with flexible constraints, which allows greater flexibility and adaptability in the scheduling of courses and schedules, meeting the specific needs of the educational institution.

Keywords: time allocation; schedule grid; meet schedules

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Toda instituição de ensino, no início de qualquer período letivo, deve apresentar a seus docentes e discentes a relação de disciplinas, horários e suas respectivas salas. Nesse processo, origina-se o problema de alocação de horários, conhecido na literatura como *timetabling*, e o *problema de alocação de salas* (PAS).

Schaerf (1995) define o PAS como uma associação de disciplinas, professores e salas, sempre agendadas em um período determinado de tempo, e que devem satisfazer um conjunto de restrições. De fato, a complexidade desse processo é alta, em razão da quantidade de variáveis que devem ser atendidas, acrescido também do número de matrículas e cursos ofertados que aumentam com o decorrer do tempo.

Apesar da complexidade do problema, muitas das instituições ainda realizam a alocação de horários e salas manualmente, causando muitas vezes uma distribuição de horários ou recursos imperfeita, que gera insatisfação dos usuários. Dessa maneira, tem-se a necessidade de desenvolver uma solução tecnológica, para obtenção de uma acurácia maior nos resultados, ganho de velocidade na criação do agendamento e aprimorar manutenções futuras caso alguma mudança seja solicitada.

Embora existam vários trabalhos na literatura, mesmo com a semelhança entre os problemas, não existe uma solução genérica. Isso ocorre pela mudança de requisitos e restrições de cada organização. As soluções propostas são suficientemente amplas, muitos autores podem utilizar de programação inteira (BAGGER et al., 2019), inteligência artificial (BANBARA et al., 2019), lógica proposicional (ACHá; NIEUWENHUIS, 205), entre outros.

Nesta monografia, é apresentada uma proposta para o desenvolvimento de um *backend* que visa otimizar a alocação de horários e salas na Universidade Federal do Ceará (UFC), Campus Russas, através da utilização de um solucionador pseudo booleano. O objetivo é oferecer uma aplicação com documentação abrangente, facilitando assim a construção de uma solução frontend sem a necessidade de conhecimento prévio sobre todas as regras de negócio da instituição. Dessa forma, a alocação poderá ser realizada de maneira rápida e precisa, seguindo os parâmetros estabelecidos pelos usuários.

Cabe ressaltar que a UFC possui cinco cursos distintos, sendo eles: Engenharia de Software, Ciência da Computação, Engenharia Civil, Engenharia Mecânica e Engenharia de Produção. Cada um desses cursos possui sua própria grade curricular, organizada em semestres, sendo que cada semestre é composto por disciplinas específicas.

O desafio central deste projeto é alocar de forma eficiente e inteligente as disciplinas no calendário, levando em consideração a disponibilidade de horários dos professores do campus e respeitando a restrição de não alocar disciplinas do mesmo semestre em um mesmo horário.

Por meio do backend proposto, será possível otimizar essa alocação, garantindo que as disciplinas sejam dispostas de maneira adequada no calendário, atendendo a todos os horários dos professores e seguindo as regras estabelecidas pelos semestres. Essa abordagem proporcionará um processo de alocação mais ágil e eficaz, contribuindo para uma gestão otimizada dos recursos acadêmicos.

O restante do trabalho se divide em 5 capítulos. No capítulo 2 são mostradas as explicações dos conceitos utilizados neste trabalho. No capítulo 3 são mostrados os trabalhos relacionados ao tema de pesquisa. No capítulo 4 é mostrada a metodologia que foi utilizada para construção desta pesquisa. No capítulo 5 são mostrados os resultados desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção estão descritos os principais conceitos e descrições utilizadas nesse trabalho.

2.1 Alocação de Horários e Salas

Qu et al. (2009) classifica o *problema de alocação de horários* ou *timetabling* em educacionais, hospitalares, esportiva e entre outras. Burke, Werra e Kingston (2004) define como um conjunto finito de horários T (*times*); um conjunto finito de recursos R (*resources*); um conjunto de eventos M (*meetings*); e um conjunto finito de restrições C (*constraints*).

A definição de tempo pode ser estabelecida através da criação de intervalos de tempo arbitrários, permitindo que esse fluxo temporal seja subdividido e organizado em um conjunto finito de unidades. Dessa forma, podemos compreender e mensurar a passagem do tempo de maneira mais precisa e estruturada. As instituições possuem horários recorrentes, ou seja, ocorrem em intervalos regulares, como semanalmente.

Os recursos podem ser definidos como professores, salas, equipamentos, alunos e outros. As restrições devem levar em consideração os recursos utilizados. Por exemplo, os professores e as salas devem ter seus cronogramas sem que haja conflitos de horários. Isso garante que sempre haverá horários disponíveis caso seja necessário.

Os eventos serão representados como uma coleção de intervalos de tempo e um conjunto de recursos. A ideia central é atribuir valores aos intervalos de modo que todos os eventos estejam devidamente inseridos na estrutura proposta.

Ao final, encontramos as restrições que tornam a problemática de alocação de horários e recursos complexa e desafiadora, já que cada instituição possui suas próprias restrições e peculiaridades. O desafio de agendar horários em instituições de ensino reside no número de restrições estabelecidas e na escassez de recursos disponíveis.

Geralmente, essas restrições são caracterizadas em dois grupos distintos, conforme definido por Qu et al. (2009): as restrições rígidas, conhecidas como *hard constraints*, e as restrições flexíveis, chamadas de *soft constraints*.

- Restrições Rígidas:
  - Não podem ser violadas em hipótese alguma. Um exemplo seria disciplinas que pertencem a um semestre e curso, essas não podem ser agendadas no mesmo horário.

  - Um cronograma deve satisfazer todas as restrições rígidas, para ser considerado válido.
- Restrições Flexíveis:
  - Essas são apenas desejáveis, em prática é impossível gerar um cronograma que satisfaça todas as restrições rígidas e flexíveis. As restrições flexíveis variam de instituição a instituição, e muitas vezes podem conflitar umas nas outras.
  - Um exemplo de restrição flexível é agendar horários de professores em determinados horários, ou seguindo um padrão já estabelecido.

Na área educacional Schaerf (1995) distribui o problema em três grupos: ***school timetabling***, ***course timetabling*** e ***examination timetabling***. Os grupos ***school timetabling*** e ***course timetabling*** representam o agendamento de salas de instituições do ensino médio e universitário, e o grupo ***examination timetabling*** é sobre a designação de horários para avaliações.

2.2 Desenvolvimento de Software

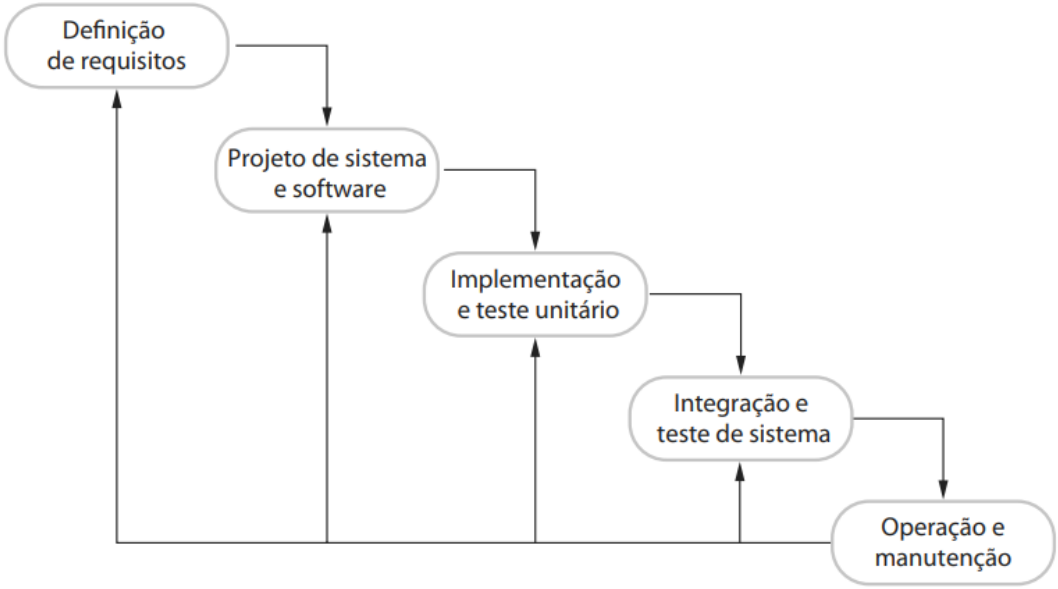
Para a construção de um projeto de software é necessário seguir uma sequencia de passos, como a coleta dos requisitos desse sistema, a construção do código fonte e os testes para determinar se esse projeto está de acordo com os requisitos levantados, e por fim sua instalação, treinamento e a evolução do software. Esses processos são complexos e precisa de pessoas para tomada de decisões e julgamentos. Dessa maneira é necessário criar um processo bem definido, desenvolvendo assim o máximo poder intelectual dessa equipe. (PRESSMAN, 2005)

O primeiro modelo catalogado é o modelo cascata, esse modelo é constituído como uma sequencia de passos que devem ser seguidos até a entrega final do sistema desejado. Cada etapa possui entradas, que seria documentos necessários para a realização da tarefa, e no término das atividade são entregues os artefatos. Esse modelo é altamente compreensível e fácil de usar, e foi o modelo utilizando neste trabalho. Na figura 1 mostra os passos do modelo cascata.

A seguir, apresentamos o significado de cada etapa do modelo:

- Definição de requisitos
  - Nessa etapa é realizada uma análise detalhada das necessidades do cliente, a fim de determinar os requisitos que o sistema deve atender. Essa etapa é fundamental para garantir que o sistema desenvolvido atenda às necessidades do cliente e alcance os objetivos definidos. São realizadas entrevistas, reuniões e pesquisas para entender as

Figura 1 – Modelo Cascata



Fonte: Pressman (2005, pag. 20)

expectativas do cliente em relação ao sistema e definir as funcionalidades e recursos necessários. A partir disso, são definidos os requisitos funcionais e não funcionais, que serão utilizados como base para o desenvolvimento do sistema. É importante que essa etapa seja realizada com cuidado e atenção aos detalhes, para evitar problemas futuros e garantir a satisfação do cliente.

- Projeto de sistema e software
  - Na etapa de projeto de sistema e software, são utilizados os requisitos previamente definidos para construir os modelos e documentos necessários à implementação do sistema. Essa etapa é crucial para garantir que o sistema seja desenvolvido de forma eficiente, seguindo as especificações definidas. São criados diagramas, fluxogramas, mapas de navegação, protótipos, entre outros artefatos que ajudam a equipe de desenvolvimento a entender o escopo do projeto e a trabalhar de forma mais organizada e produtiva. É importante que os documentos produzidos nessa etapa sejam claros e precisos para evitar mal-entendidos e erros durante a implementação.
- Implementação e teste unitário
  - Após a conclusão do projeto, inicia-se a etapa de construção e codificação do sistema. Seguindo as especificações e requisitos definidos na fase anterior. É importante que a equipe de desenvolvimento siga as diretrizes estabelecidas para garantir a qualidade

e a eficácia do sistema.

- Testes do Sistema
  - Após a conclusão do projeto, são realizados testes para verificar se o que foi planejado está em conformidade com a implementação. Esses testes são essenciais para garantir a qualidade e a funcionalidade do projeto e garantir que ele atenda aos requisitos e expectativas dos usuários.
- Operação e manutenção
  - Apesar de passar por uma etapa de testes, é possível que problemas não tenham sido identificados antes do usuário final utilizar o projeto. É nessa fase que se busca resolver e melhorar quaisquer problemas que possam surgir, a fim de garantir que o projeto atenda às necessidades do usuário e funcione da maneira mais eficiente possível. É importante estar preparado para essa fase e estar aberto a fazer as melhorias necessárias para garantir a satisfação do usuário.

A medida que foi sendo utilizado esse modelo, foi visto a necessidade de fazer alterações para adaptar o processo a uma determinada situação ou instituição. Dessa forma surgiu alguns outros modelos como por exemplo, o modelo incremental ou o modelo espiral.

2.3 Arquitetura de software

Nos passos iniciais de um projeto de software são definidos possíveis tipos de arquiteturas para serem avaliados e aprovadas para o projeto. Isso é necessário para se obter um entendimento de como o sistema deve estar estruturado e organizado, e dessa forma, melhorar a comunicação dos stakeholders. (PRESSMAN, 2005)

Os requisitos funcionais, regras de negocio e requisitos não funcionais definidos no início do projeto têm um impacto direto na arquitetura do sistema, isso adiciona limitadores no projeto, como por exemplo, desempenho da aplicação, proteção dos dados, segurança das informações, disponibilidade do software e por fim a sua manutenção. Embora cada sistema possua suas particularidades, tornando-o único, as arquiteturas utilizadas muitas vezes se repetem, tornando-as similares em diferentes projetos.

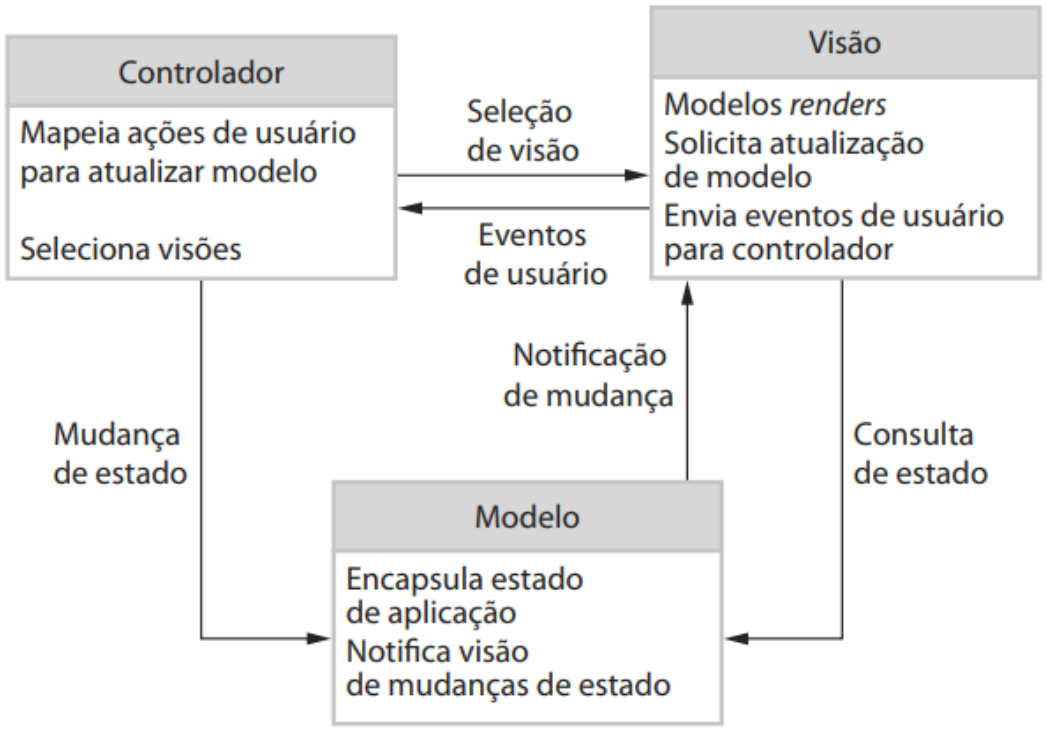
Nesse sentido a Arquitetura de Software bifurca em dois conceitos, padrões de arquitetura e visões de arquitetura. Os padrões de arquitetura servem para mostrar onde o seu projeto será inserido e como ele se comunicará com todas as partes do sistema. Um exemplo de arquitetura de software conhecido na literatura é o Cliente-Servidor, onde os clientes se

comunicam com o servidor afim de requisitar informações e o servidor por sua vez provê essas informações.

Já a visão arquitetural está preocupada em como o seu projeto está estruturado. Por exemplo uma visão bastante comum na literatura é o padrão MVC (Modelo, Visão e Controlador), onde dentro do projeto é feito a separação dos seus arquivos por meio de pacotes conhecidos como: dados do sistema, visões de interface ao usuário e os controladores. Dessa maneira deixando o projeto mais organizado até para futuras manutenções.

- Modelo
  - O modelo será a camada responsável por gerenciar todos os dados utilizados e requeridos do sistema ao longo da sua vida útil.
- Visão
  - A camada de visão será responsável por mostrar aos usuários as informações do sistema.
- Controle
  - O controlador será utilizado nessa estrutura para ligar a parte de visão com os dados do sistema, fazendo assim a gestão interna de cada componente dentro do sistema.

Figura 2 – MVC



Fonte: Pressman (2005, pag. 109)

Com base na facilidade de separação de arquivos e no controle que esse modelo oferece para a aplicação, essa será a visão arquitetural adotada neste trabalho.

2.4 Lógica para Computação

A Lógica Proposicional é uma linguagem formulada para expressar frases de maneira que sua estrutura lógica se torne clara, sendo baseada em proposições ou frases declarativas (FINGER; SILVA; MELO, 2006). Uma proposição é uma frase que pode assumir um valor verdadeiro ou falso.

> ”Dessa forma, a *lógica proposicional clássica* nos permite tratar de enunciados aos quais podemos atribuir valor verdade (as proposições) e as operações que permitem compor proposições complexas a partir de proposições mais simples [...]” (FINGER; SILVA; MELO, 2006, p.8).

Exemplos de proposições:

- O céu é azul.
- O dia está nublado.
- Não ganhei na loteria.

2.4.1 Alfabeto

O alfabeto da linguagem proposicional é composto pelos seguintes elementos: um conjunto de símbolos proposicionais (P = p1, p2, . . .); o conectivo $\neg$ (negação); conectivos binários $\wedge$ (conjunção), $\vee$ (disjunção) e $\Rightarrow$ (implicação); e elementos de pontuação, como parênteses. (HUTH; RYAN, 2008)

2.4.2 Fórmulas

Os elementos da linguagem da lógica proposicional e a relação entre esses elementos é denominado de *fórmulas*. O conjunto das fórmulas proposicionais é definido indutivamente e contém as seguintes regras de formação:

1. Caso básico: Todos os símbolos proposicionais estão no conjunto de fórmulas;
2. Caso indutivo 1: Se A pertence ao conjunto, então $\neg$ A também pertence ao conjunto;
3. Caso indutivo 2: Se A, B pertencem ao conjunto, então (A $\vee$ B) pertencem ao conjunto, (A $\wedge$ B) pertencem ao conjunto, (A $\Rightarrow$ B) pertencem ao conjunto;
4. O conjunto das fórmulas proposicionais é o menor conjunto obtido das regras 1, 2 e 3.

2.4.3 Fórmula Normal Conjuntiva (FNC)

A Fórmula Normal Conjuntiva (FNC) é utilizada como formato de entrada na maioria dos algoritmos de verificação de satisfazibilidade. Uma FNC é uma fórmula na qual segue um modelo fixo. Basicamente são elementos ligados por $\vee$ (disjunções) e essas ligações por sua vez estão conectadas por $\wedge$ (conjunções). O elemento básico de uma FNC é o literal, que pode ser uma variável p positiva ou a sua negação $\neg$p negativa.

2.4.4 O Problema da Satisfazibilidade Booleana (SAT)

O problema da satisfazibilidade booleana (SAT) é formulado por, dado uma fórmula FNC, encontrar um conjunto de atribuições verdadeiro ou falso aos átomos de maneira que a fórmula como um todo seja verdade. Se for possível encontrar uma resposta, então é dito que o problema é satisfazível, caso contrário insatisfazível.

2.4.5 O Método DPLL

O DPLL, criado em 1962, é um algoritmo maleável que tem como objetivo a construção de uma valoração para uma FNC (FINGER; SILVA; MELO, 2006, pag. 94). Ao iniciar, todos os literais recebem a valoração "*", representando seu valor como indefinido. O algoritmo é encerrado quando se encontra uma valoração que a FNC é satisfazível ou insatisfazível.

2.5 O Problema da Satisfazibilidade Booleana (SAT)

O problema da satisfazibilidade booleana (SAT), é formulado por, dado uma fórmula, encontrar um conjunto de atribuições verdadeiro ou falso aos átomos, de maneira que a fórmula como um todo seja verdade. Se for possível encontrar uma resposta, então é dito que o problema é satisfazível, caso contrário insatisfazível. O SAT foi o primeiro a pertencer aos Não determinísticos Polinomialmente completos (NP-completos), todos os problemas que fazem parte desse grupo, não possuem nenhuma solução eficiente conhecida.

2.5.1 Representação do problema pseudo booleano

A fórmula pseudo booleana denota-se como um somatório de constantes inteiras seguidas de um operador relacional (=, >, >=, <, <=). Como por exemplo temos:

$$L_1 + L_2 + L_3 + L_4 \geq 3$$

Uma possível solução para este exemplo é $L_1 = 0$, $L_2 = 1$, $L_3 = 1$, $L_4 = 1$, dessa maneira essa formula é satisfeita quando os elementos a esquerda somados satisfazem ao operador que foi colocado.

Na seção 2.4.3 é definido como FNC um conjunto de cláusulas ligadas por conjunções, e cada cláusula é formada por literais ligados por disjunções. A cláusula para ser verdade basta que apenas um literal que a componha seja verdade, então é possível estabelecer uma relação com a representação booleana da seguinte instrução:

$$L_1 + L_2 + L_3 + ... + L_4 \geq 1$$

Caso qualquer elemento unário da formula tenha valoração verdadeira então essa formula será dita verdade.(MANQUINHO; MARQUES-SILVA, 2005)

2.6 Bancos de dados

A utilização de um banco de dados é importantíssimo nas diversas áreas da tecnologia, tendo relação tanto com dispositivos móveis quanto computadores pessoais. Atendendo ao armazenamento de informações em estruturas de dados relacionais. Segundo Elmasri e Navathe (2011), um banco de dados constitui-se de uma coleção de dados que se relacionam entre si, e para o desenvolvimento de sistemas com qualidade e confiabilidade é necessário entender os conceitos e fundamentos dessa tecnologia.

Um banco de dados é construído e alimentado para um propósito definido, ele possui usuários e programas que estão diretamente interessados nas suas informações.

Um SGBD é um sistema para gerenciar o banco de dados, ele irá facilitar a criação, manutenção e verificação de dados das aplicações e dos usuários. Para a construção e manutenção de um banco de dados é necessário criar estruturas com tipos e restrições de dados a serem inseridos. O SGBD irá auxiliar o projeto para manter os dados e os requisitos de projeto caso mudem.

Contudo, no banco de dados, os seus registros podem ser mantidos com as seguintes operações:

- Criação:
  - Realizar a inserção de dados no banco de dados.
- Leitura
  - Uma vez no banco de dados, esses registros podem ser lidos e apresentados aos

usuários, e utilizados durante a utilização do sistema.

- Alteração
  - Caso seja inserido um registro e algum dados não esteja de acordo, pode ser realizado a alteração.
- Exclusão
  - E por fim, pode ser feito a eliminação de registros que não devem mais fazer parte daquela base.

Figura 3 – Tabelas de um banco de dados

FUNCIONARIO

Pnome	Minicial	Unome	Cpf	Datanasc	Endereco	Sexo	Salario	Cpf_supervisor	Dnr
João	B	Silva	12345678966	09-01-1965	Rua das Flores, 751, São Paulo, SP	M	30.000	33344555587	5
Fernando	T	Wong	33344555587	08-12-1955	Rua da Lapa, 34, São Paulo, SP	M	40.000	88866555576	5
Alice	J	Zelaya	99988777767	19-01-1968	Rua Souza Lima, 35, Curitiba, PR	F	25.000	98765432168	4
Jennifer	S	Souza	98765432168	20-06-1941	Av. Arthur de Lima, 54, Santo André, SP	F	43.000	88866555576	4
Ronaldo	K	Lima	66688444476	15-09-1962	Rua Rebouças, 65, Piracicaba, SP	M	38.000	33344555587	5
Joice	A	Leite	45345345376	31-07-1972	Av. Lucas Obes, 74, São Paulo, SP	F	25.000	33344555587	5
André	V	Pereira	98798798733	29-03-1969	Rua Timbira, 35, São Paulo, SP	M	25.000	98765432168	4
Jorge	E	Brito	88866555576	10-11-1937	Rua do Horto, 35, São Paulo, SP	M	55.000	NULL	1

DEPARTAMENTO

Dnome	Dnumero	Cpf_gerente	Data_inicio_gerente
Pesquisa	5	33344555587	22-05-1988
Administração	4	98765432168	01-01-1995
Matriz	1	88866555576	19-06-1981

LOCALIZACAO_DEP

Dnumero	Dlocal
1	São Paulo
4	Mauá
5	Santo André
5	Itu
5	São Paulo

Fonte: Elmasri e Navathe (2011, pag. 48)

Em um banco de dados é necessário correlacionar dados entre tabelas. Geralmente cada tabela tem uma chave primária para identificar os registros contidos nela. Na figura 3 a tabela funcionário tem como chave primária o CPF do funcionário. Vale ressaltar que chave primária é única, então como mostrado nesse exemplo, não pode haver dois funcionários com o mesmo CPF.

Em determinado momento é necessário também fazer a vinculação de uma ou mais tabelas, dessa maneira não é preciso duplicar informações dentro de um banco de dados, basta colocar chaves para referenciar uma determinada tabela. Na figura 3 é possível visualizar que o departamento tem uma localização e por sua vez, não foi necessário colocar o nome da localização e sim a sua referencia por meio do campo *DNumero*. Essa referência denota-se chave

estrangeira.

Para facilitar a criação de um banco de dados, o primeiro passo é a elaboração de diagramas, também conhecidos como esquemas. Essa etapa marca o início da construção do banco de dados, onde são definidas as tabelas, relações e atributos que serão incorporados. Nessa fase é possível destacar a criação de dois diagramas, o Modelo de Entidade Relacional (MER) que está mostrará as tabelas e suas ligações e o Diagrama de Entidade Relacional (DER) que por sua vez mostrará os atributos necessários para cada tabela.

2.7 Interface de Programação de Aplicação

Após 2015, foi possível ver o mercado de aparelhos mobile crescer, e dessa maneira, o consumo de aplicativos e sites aumentou, e para acessar essas informações as empresas começaram a utilizar de programas para essas aplicações. Interface de Programação de Aplicação mais conhecido como API's podem ser vistas como programas que disponibilizam informações a quem deseja recebe-las ou *backends*, ou seja, o seu dispositivo não acessa as informações de uma empresa diretamente, ele faz solicitações a essa determinada API e recebe as respostas programadas. (JIN; SAHNI; SHEVAT, 2018)

Essa tecnologia irá te entregar informações que você não precisa necessariamente entender sua regra de negócio. Por exemplo não é necessário entender como funciona o Login com o Google, basta você colocar o botão na sua aplicação e utilizar da rota disponibilizada pela Google. Isso também demonstra que você não precisa, ao criar uma aplicação, construir tudo, nesse exemplo, do login, pode-se utilizar dessa API para aumentar o tempo de construção ganhando assim tempo para desenvolver outras funcionalidades mais importantes do projeto.

Para projetar uma API de forma adequada, é fundamental começar definindo sua estrutura, que pode se basear em diversos paradigmas, tais como: REST, RPC, GraphQL, WebHooks e WebSockets.

2.7.1 REST

O REST (Transferência de Estado Representacional) é uma tecnologia que utiliza a web para o tráfego de dados, utilizando os métodos http para fazer esse procedimento como, GET, POST, PUT, PATCH e DELETE.

- GET (Buscar)

  - Quando é preciso acessar informações da API
- POST (Submeter)
  - Caso seja necessário fazer uma requisição de submissão de dados.
- PUT (alterar)
  - Se for preciso fazer a alteração de algo na API.
- PATCH (Atualização)
  - Para atualizações parciais em registros.
- DELETE (remoção)
  - Exclusão de um registro.

Como citado acima essa tecnologia é manipulada pela web, então para acessar as informações de um determinado backend, é utilizado por meio de rotas, e essas rotas são links. Por exemplo, para a criação de um usuário em uma determinada API, pode ser utilizada a rota https://127.0.0.1/users.

No link acima o https é o protocolo de transferência, é uma camada da web para fazer o tráfego de informações de maneira segura e criptografada. Logo em seguida vemos o 127.0.0.1, é um endereço onde está o servidor da aplicação, lembrando que pode ser tanto um IP válido como também um HOST definido pela empresa do *backend*. Em sequencia tem as rotas, apresentado acima pelo */users*.

Os métodos determinam ao servidor como executar a ação. Se a rota do exemplo anterior for acessada usando o método GET, o servidor receberá essa requisição e retornará todos os usuários da sua base de dados. Caso utilize-se o método POST, o servidor receberá um formulário de submissão para adicionar um novo usuário.

Para o trafego de informações em uma API Rest utiliza-se o *Javascript Object Notation* (JSON). É um formato leve para a troca de informação entre sistemas, a inspiração dessa notação veio por meio do *javascript*, e hoje é utilizada pela maioria dos sistemas. A estruturação do JSON é feita por meio de chaves e valores, cada chave contém uma *string*, dois pontos e um valor. Cada par é separado por uma virgula e tudo é delimitado por chaves ({}) para representação de objetos, e colchetes ([]) para representação de listas. O JSON suporta os tipos primários de dados, como por exemplo, textos, inteiros, números flutuantes e valores booleanos. Na figura 4 é mostrado o exemplo de estrutura de um JSON.

Figura 4 – Exemplo de JSON

```
{
        "CODIGO" : 1,
        "DATA" : "23/03/2023"
        "CORES" : [
                {
                        "COR" : "VERMELHO",
                        "HEX" : "#FF0000"
                },
                {
                        "COR" : "AZUL",
                        "HEX" : "#0000FF"
                }
        ]


}
```

Fonte: Elaborada pelo autor (2023)

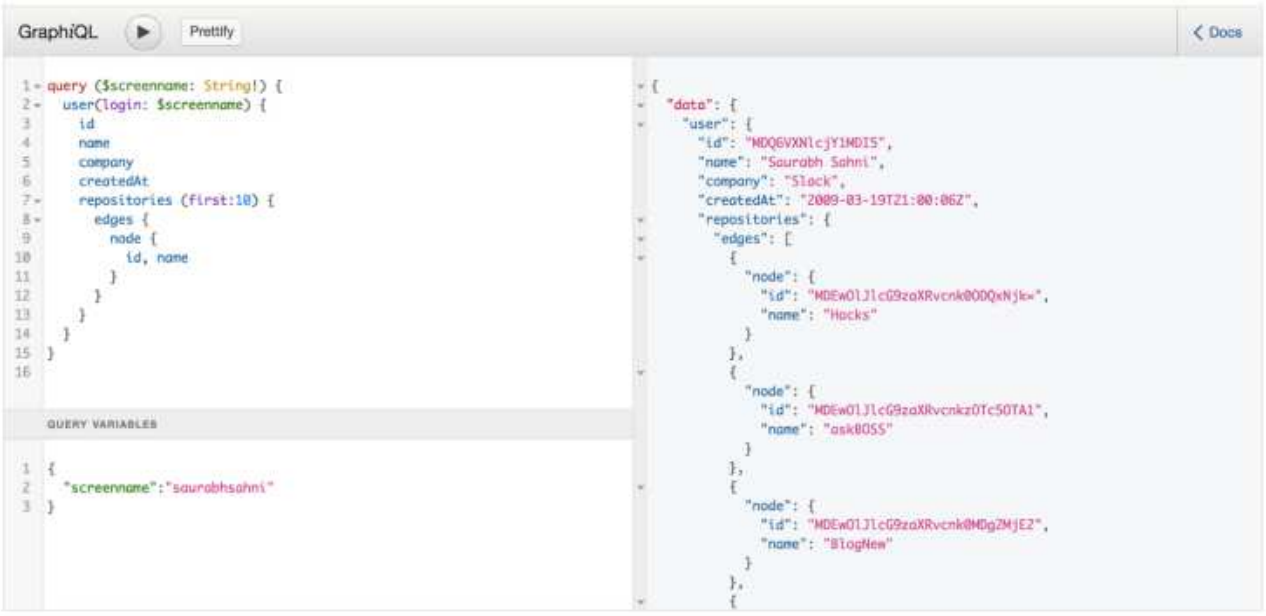
2.7.2 GraphQL

Criada em 2012 pelo Facebook, o GraphQL é uma estrutura que permite aos seus usuários requisitar respostas ao servidor da maneira que o frontend necessita. Basicamente o usuário requisita ao backend já a estrutura de retorno.

A estrutura é parecida com o Rest, porém no campo data da requisição HTTP é necessário passar o que será retornado pela API, dessa maneira fica mais fácil de descontinuar campos caso usuários finais não estejam utilizando.

Porém com essa facilidade de construção de requisições, nasce a dificuldade em entender como funciona a estrutura do backend, pois para que o frontend faça as requisições necessárias ele precisará de um entendimento do que são essas informações, diferente por exemplo do Rest que lhe retornará algo fixo. A figura 5 demonstra como funciona uma requisição usando o GraphQL.

Figura 5 – Requisição GraphQL



Fonte: Jin, Sahni e Shevat (2018, pag. 17)

2.7.3 WebHook

Um Webhook é essencialmente um link que permite o envio de mensagens ao servidor por meio dos métodos HTTP, acionando o envio dessas mensagens quando ocorre um evento específico.

Essa abordagem é especialmente interessante para realizar requisições em tempo real. No entanto, é importante estar ciente das possíveis complexidades envolvidas. Por exemplo, falhas podem ocorrer ao tentar enviar uma mensagem ao servidor devido a interferências externas. Além disso, a segurança é um aspecto crucial, pois a criação de Webhooks ainda é uma área em crescimento em termos de segurança.

Firewalls e outros equipamentos podem bloquear requisições desse tipo. Outro desafio comum é que cada requisição do Webhook é projetada para executar rotinas simples. Portanto, se você deseja realizar múltiplas tarefas, será necessário fazer várias solicitações, o que pode resultar em um aumento significativo no tráfego de comunicação com o servidor.

2.7.4 WebSockets

O protocolo WebSockets serve para fazer a comunicação bilateral entre dois aparelhos, geralmente cliente e servidor, esse padrão é normalmente utilizado por aplicações web.

Essa ferramenta utiliza uma comunicação full-duplex, ou seja, o servidor e o cliente podem se comunicar ao mesmo tempo, e para ser feita essa comunicação utiliza-se portas de comunicação, geralmente as portas 80 ou 443 o que facilita para que o firewall dos dispositivos não faça o bloqueio das comunicações.

WebSockets são ideais para a transferência de informações em tempo real. Entretanto para que essa transferência seja efetuada com sucesso a conexão entre os 2 pontos não pode sofrer qualquer interferência ou corte, pois faria com que as informações que estavam sendo trafegadas no momento da perda não funcionem, e por sua vez terá que reconectar e iniciar o processo novamente. Outro ponto também importante é a conexão entre dispositivos, se a aplicação requisitar 5000 usuários, então serão essas 5000 conexões simultâneas o que pode acarretar em diminuição da eficiência do servidor e lentidão.

2.8 Ciclo Regulador de Wieringa (Design Science)

Design Science proposto por Wieringa (2014) é um método de pesquisa desenvolvido para a criação e avaliação de artefatos tecnológicos. Esse conceito concentra-se em criar artefatos implementados e testados no mundo real.

O processo de utilização do Design Science, tem como consistência a execução das seguintes atividades:

- Identificação do problema: tentar entender completamente o problema que terá que ser resolvido e buscar requisitos e objetivos desse artefato.
- Invenção: projetar a solução desse problema, atendendo aos requisitos impostos no passo anterior.
- Avaliação: avaliar esse artefato, testando e comparando seu desempenho com outras soluções.
- Justificação: Argumentar sobre a validade desse artefato, demonstrando assim que ele resolve o problema proposto de acordo com princípios científicos estabelecidos
- Implementação: desenvolver a implementação prática, por sua vez tornando disponível para o contexto real.
- Reflexão: Refletir sobre esse processo e sobre todos os artefatos criados, e buscar melhorias e correções para projetos futuros

Essa abordagem é para ser utilizada de maneira prática, os resultados são validados por meio de estudos de caso, protótipos, ou implementações reais. E essa metodologia é útil

quando é necessário criar soluções tecnológicas, para resolução de problemas complexos no mundo real.

3 TRABALHOS RELACIONADOS

Nessa seção são destacados três trabalhos relacionados a essa pesquisa. Na seção 4.1 o *Curriculum-based course timetabling with SAT and MAXSAT* (ACHá; NIEUWENHUIS, 205), na seção 4.2 *teaspoon: Solving the Curriculum-Based Course Timetabling Problems with Answer Set Programming* (BANBARA et al., 2019) e na seção 4.3 *Flow Formulations for Curriculum-based Course Timetabling* (BAGGER et al., 2019). A tabela 1 mostra a relação de trabalhos relacionados com o proposto por essa monografia.

3.1 *Tabela de horários do curso com base no currículo com SAT e MAXSAT*

Em Achá e Nieuwenhuis (205) é apresentado a resolução do problema de alocação de horários, utilizando MaxSAT ponderado, ponderado parcial e parcial. Foram definidas 4 variáveis proposicionais, que relaciona os cursos aos dias, aos horários e salas, e criadas 15 relações para as variáveis. Com isso, o artigo conseguiu resolver os problemas propostos.

Com a modelagem completa, verificada e validada, foram iniciados os testes. Todos os MaxSAT utilizaram de uma base com 32 registros, onde cada um, tinha um problema de alocação de horários e salas. Na primeira fase foi retornado como satisfazível 6 dos 32 registros.

Na segunda fase houve um aumento de 6 para 20. Em sequência houve uma reformulação na modelagem dos dados, e foram obtidos melhores resultados em 4 instâncias. Na quarta fase houve uma melhora em 3 registros. Logo após, foi feito a introdução de mais uma variável proposicional e 5 relações, para melhorar a eficiência do agendamento de salas.

No final é obtido a comparação de eficiência dos algoritmos MaxSAT ponderado, MaxSAT parcial e MaxSAT ponderado parcial apresentado por Achá e Nieuwenhuis (205, p.88).

3.2 teaspoon*: Solving the Curriculum-Based Course Timetabling Problems with Answer Set Programming*

No trabalho de Banbara et al. (2019), utiliza-se da programação de conjunto de resposta para resolver o problema de alocação de horários, na qual, o problema é descrito como um programa lógico, onde é formado por um conjunto de regras para sua resolução.

É definida uma base de dados, nome, cursos, salas, dias, horários, currículos, mínimo e máximo dias de disciplinas, número de restrições inválidas, e número de restrições para salas, e as relações entre os fatos.

As relações são formadas por cursos e professores, salas e seus locais, os cursos e seus determinados currículos, as restrições do curso e restrições das salas. Por fim existe uma relação de atribuição que basicamente une as disciplinas, com seus horários e salas. Por fim foi criado a relação de penalidade, onde cada violação da resolução é criado um custo de penalidade. Essa penalidade impacta nas escolhas de restrições flexíveis.

Para essa pesquisa foi utilizado a noção de *k-way configurations* (maneiras de configuração), escolhendo assim 15. O resultado das configurações para cada situação determina o melhor resultado do conjunto de problemas selecionados.

Com essa noção, encontrou-se 13 melhores configurações que resolviam o problema. A partir dessas configurações, o algoritmo foi executado e comparado com outras abordagens, e houve um sucesso de 55,3%. Depois foi comparado os novos resultados com os trabalhos anteriores, e foi visto um total de 90.8% de melhorias.

3.3 *Flow Formulations for Curriculum-based Course Timetabling*

Em Bagger et al. (2019) o problema de alocação de horários é resolvido utilizando programação de números inteiros, onde é separada a programação inteira em dois modelos distintos *Minimum Cost Flow* e *Multi-Commodity Flow*, dessa maneira resolvendo os mesmos em sequência, após a conclusão, os modelos são unidos, usando técnicas de programação inteira.

No *Minimum Cost Flow*, formula-se o problema de cursos e períodos como um subproblema, criando assim um modelo. Em sequência é criado um modelo para tratar do problema de salas. Finalmente com os dois modelos formados, foi checado a capacidade de salas mínimas para os cursos. Na seção de *Multi-Commodity Flow*, é formulado o modelo para resolver o agendamento dos cursos, e em quais salas os mesmos devem ficar semanalmente.

Para os testes dessa modelagem, foram utilizado 3 *datasets*, com um total de 32 instâncias do problema. No trabalho é mostrado o número de vezes que se obteve um ótimo resultado, e é comparado com outros diferentes algoritmos, mostrando melhores resultados em muitos casos.

3.4 Comparação Entre os Trabalhos

Nessa seção é mostrada a relação dos trabalhos citados com esse trabalho. Os tópicos que serão utilizados para a comparação são, Resolução do problema de alocação de horários,

modelar o problema utilizando lógica computacional, comparação da resolução com outras soluções, resolução do problema para um propósito geral e resolução do problema para um propósito específico.

Tabela 1 – Trabalhos Relacionados

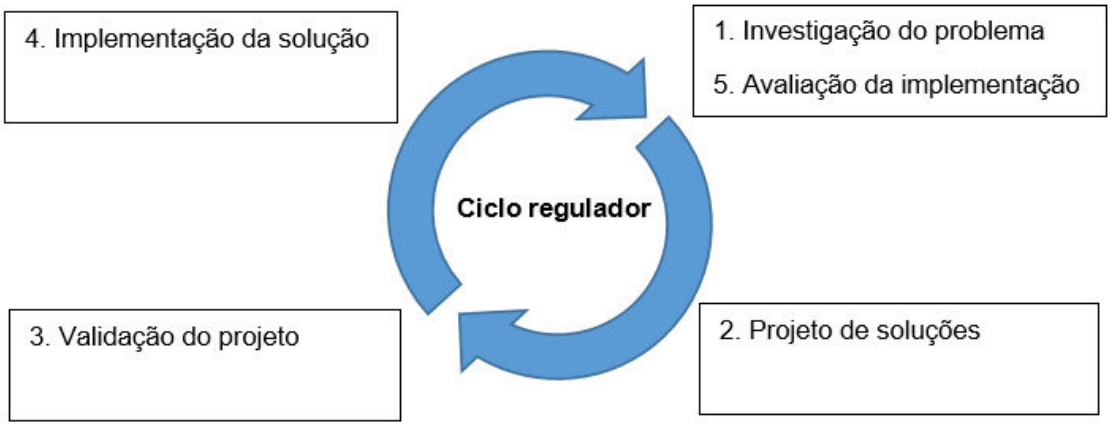
Atividades	Trabalhos Relacionados			
	ACHÁ e NIEUWENHUIS (2012)	Banbara M. (2019)	Bagger NC.F. (2019)	TCC Gabriel
Resolução do problema de alocação de horários	X	X	X	X
Modelando o problema utilizando lógica proposicional	X	X		X
Comparação da resolução com outras soluções	X	X	X	
Resolução do problema para propósito geral	X	X	X	
Resolução do problema para propósito específico				X

Fonte: Elaborada pelo autor (2023)

4 PROCEDIMENTOS METODOLOGICOS

Dada a natureza do estudo que essa pesquisa está inserida, será utilizado o procedimento metodológico proposto por Wieringa (2014), o *Design Science*, que contém cinco etapas na sua estrutura, como é mostrado na figura 6.

Figura 6 – Ciclo Regulador de Wieringa



Fonte: Barbosa e Bax (2017, pag. 38)

4.1 Investigação do Problema

Primeiramente foi levantado as informações com o coordenador do curso de Ciência da Computação, estudando como funciona o problema de alocação de salas na UFC. Procurar na literatura utilizando as strings de busca "Timetabling", "MaxSAT", "Sat and SatSolvers"e "Pseudo Boolean Solver", formas existentes de soluções. Assim no final dessa etapa, obtém-se o entendimento completo do problema. Nessa etapa foi feito o levantamento de requisitos funcionais e não funcionais da aplicação em questão.

4.2 Projeto de Soluções

Com o problema formulado, foi trabalhado no modelo para a resolução, sendo assim, nessa etapa teremos que criar os diagramas necessários para a implementação do problema. Para a criação dos artefatos é necessário utilizar dos requisitos funcionais e não funcionais coletados no passo anterior. Os diagramas dessa etapa são, diagrama de casos de uso, diagrama de classes e a diagramação do banco de dados o *MER* e o *DER*

4.3 Validação do Projeto

Com o problema definido, salvado e processado, então é feito a validação das informações, utilizando o teste de requisitos e dos critérios de aceite, como os ditos na investigação do problema. No objetivo de verificar se todos os parâmetros definidos pela instituição estão levantados.

4.4 Implementação da Solução

Então com tudo validado e verificado, tem-se o desenvolvimento da solução. Será utilizado o python para desenvolvimento. Utilizando o conceito de Pseudo Boolean Solver. Dessa maneira, o backend tem como principal função salvar as informações no banco de dados, e fazer a recuperação de tais informações para alimentar a classe de alocação. E retornar ao frontend ou a qualquer sistema que solicite esses dados.

4.5 Avaliação da Implementação

Com a solução criada e implementada, é feita uma avaliação, verificando assim se está atendendo ao problema, e se é necessário rever alguma parte que esteja faltando, seja de requisitos, ou até mesmo de programação. Utilizando os testes de requisitos, verificando se o que foi pedido realmente está de acordo com o desenvolvido e testes de caixa preta, para verificar se as rotas do projeto estão devolvendo o que é prometido.

5 RESULTADOS

5.1 Investigação do Problema

Atualmente o campus da Universidade Federal do Ceará (UFC) situado na cidade de Russas / CE tem um total de 5 cursos, Engenharia de Software, Ciências da Computação, Engenharia de Produção, Engenharia Civil e Engenharia Mecânica.

Cada curso dispõe de uma grade, que é preestabelecida pela instituição, e é composta por todas as disciplinas que o graduando deve fazer. Ajudando assim a organizar os horários para se formar em tempo hábil, ou seja, no tempo estabelecido para formação do curso.

Cada grade é separada por semestres, e cada semestre tem suas respectivas disciplinas que o compõem. Cada curso tem um total de N de semestres, e por sua vez cada semestre tem um total Y de disciplinas.

Abaixo é definido algumas regras de negócio que devem ser atendidas para o agendamento dos horários:

- Disciplinas que compõem o mesmo semestre, devem estar alocadas em horários distintos
- Disciplinas de um mesmo professor devem estar alocadas em horários distintos
- Disciplinas que acontecem no mesmo horário e dia, devem ser alocadas em salas separadas
- Não é para ser possível agendar um laboratório para duas disciplinas diferentes no mesmo período

Cada disciplina tem um total de créditos. Os créditos definem quantas horas uma disciplina deve ser lecionada. 2, 3, 4 e 6 são os créditos que podem ser associados. Abaixo é explicado como funciona cada crédito:

- Disciplinas de 2 créditos, tem um total de 32 horas, e geralmente acontece apenas uma vez na semana
- Disciplinas de 3 créditos, com um total de 48 horas. E pode encontrar em dois horários na semana, dividido em uma aula de dois horários, e outra de apenas um, ou pode-se encontrar em 3 horários.
- Disciplinas de 4 créditos, as mais comuns, tem um total de 64 horas. Essas disciplinas ocorrem 2 vezes por semana, e geralmente seguem uma regra:
  - Se ela é agendada em qualquer horário na segunda-feira, então deve ser associar no mesmo horário na quarta-feira, e vice-versa
  - Se ela é agendada em qualquer horário na terça-feira, então deve ser associada no

mesmo horário na quinta-feira, e vice-versa

  - Se ela é agendada na quarta-feira, então ela pode ser agendada na segunda-feira, ou ela pode ir para sexta-feira no mesmo horário

- Disciplinas de 6 créditos detêm da mesma funcionalidade da de 4 créditos, mas deve também agendar mais um horário na semana.

Após compreender como as regras de negócio estão inseridas na instituição, realizou-se o levantamento dos requisitos que o sistema deve incorporar. Esse levantamento de requisitos foi feito com ajuda na época do coordenador do curso de Ciências da Computação do campus.

Ao concluir esta etapa, compilamos os requisitos funcionais e não funcionais necessários para o pleno funcionamento do sistema. Esses requisitos são apresentados nas Tabela 2 e 3 a seguir.

Tabela 2 – Requisitos Funcionais

Identificador	Nome	Descrição
RF001	Realizar Login	O sistema tem que ter a possibilidade de se realizar login, para que somente pessoas autorizadas usem as funcionalidades
RF002	Professores	No sistema deve ser possível cadastrar professores e também a possibilidade de remover professores.
RF003	Cursos	No sistema é para ser possível colocar os cursos da universidade e adicionar e remover caso necessário.
RF004	Disciplinas	É preciso que no sistema tenha a lista de todas as disciplinas que existam no campus, dessa maneira ficaria mais facil de associar os professores as disciplinas.
RF005	Laboratório	A UFC tem laboratórios para que as cadeiras práticas sejam feitas por eles, dessa maneira é importante que seja possivel lançar esses laboratorios e associar também as turmas
RF006	Semestres	Cada curso da instituição tem semestre, e pode ser preciso fazer mudança em determinados semestres, como por exemplo o que aconteceu com engenharia de software, então é ideal que o sistema mantenha essa grade para que seja possivel fazer mudanças.
RF007	Realizar alocação	O sistema deve alocar os horários que derem certo para as disciplinas dos professores. E essa alocação também precisa alocar os laboratórios para as turmas que tem aulas práticas na sua disciplina. A alocação deve ter como regra não deixar horários de turmas do mesmo professor iguais, e levar em consideração o semestre também do curso, para deixar essas turmas com horários separados.
RF008	Gerenciar Turmas	Com as turmas alocadas e seus horários definidos é importante o sistema deixar fazer alterações desses horários para possíveis mudanças que possam ocorrer antes do semestre iniciar. E conseguir alterar as turmas como por exemplo alterar o professor dessa turma. A turma deve ter um limitador de laboratórios, isso se da por conta do número de alunos dessa disciplina.
RF009	Restrições	Com base nas regras de negócios é essencial que o sistema tenha restrições de horários para turmas do mesmo professor e para turmas do mesmo semestre.
RF010	Restrições Fracas	É importante também que seja possível informar horários restrições para determinadas situações, como por exemplo professores que não querem determinado horário. Também é importante colocar que essas restrições não precisam ser obrigatórias pelo algoritmo e sim que caso dê certo faça dessa forma.
RF011	Alocação Horarios Agrupados	O sistema deve manter o padrão da UFC, disciplinas que aconteçam em 2 vezes na semanas deverão ter um 1 dia entre as 2 aulas. Exemplo seria uma disciplina que ocorrer na segunda feira das 08:00 as 10:00, deverá ter a terça como dia livre, e a próxima aula acontecer somente na quarta no mesmo horário. Disciplina com 3 horários podem acontecer separado na semana ou podem acontecer também em sequencia.

Tabela 3 – Requisitos Não Funcionais

Identificador	Nome	Descrição
RNF001	API	O sistema precisa ser backend, dessa maneira será utilizado para receber requisições. Assim um possível frontend poderá utilizar para implementar um sistema web, mobile, desktop, etc.

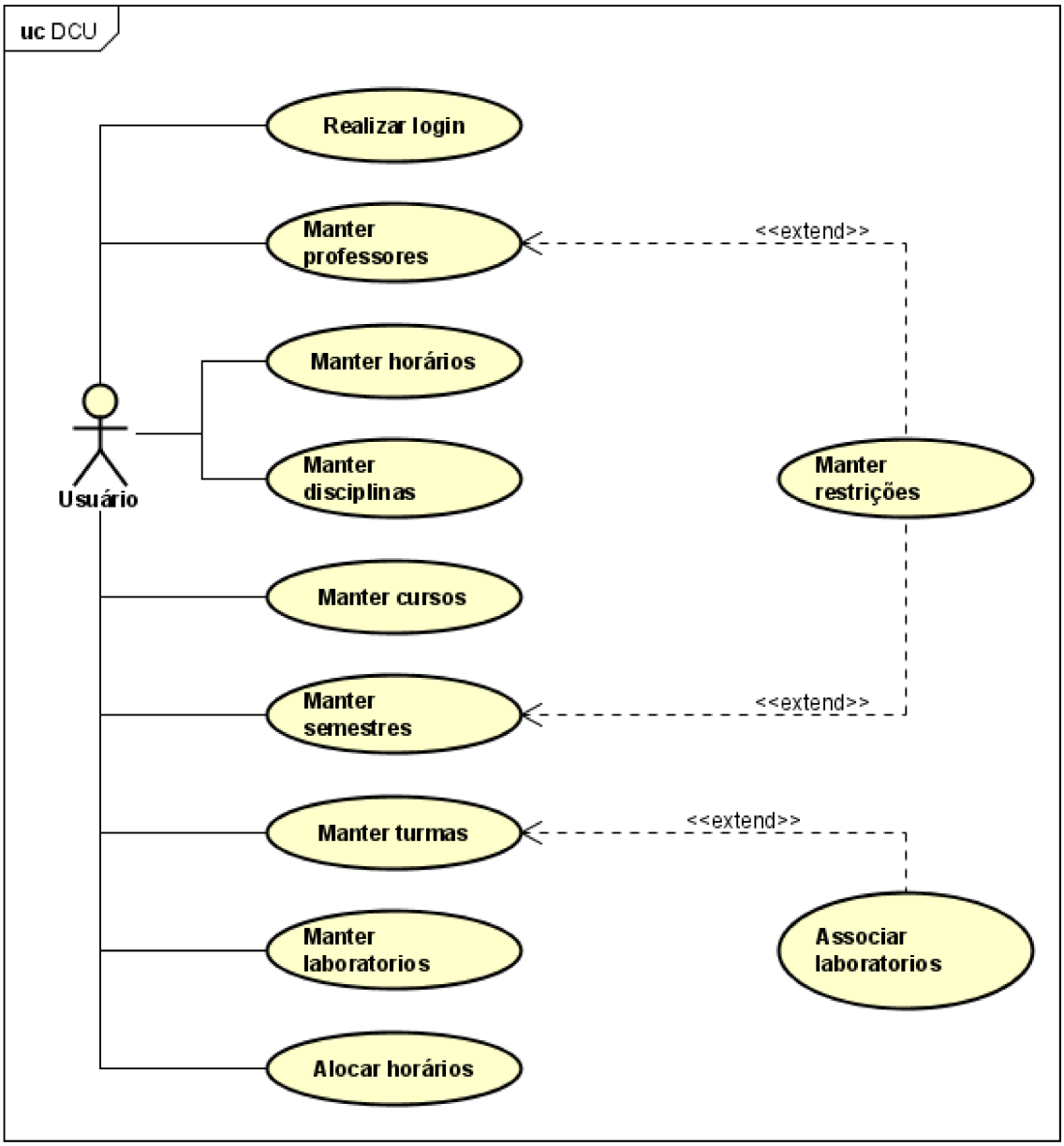
5.2 Projeto de Soluções

5.2.1 Casos de uso

Depois de entender o problema que a UFC está inserida, e a construção dos requisitos funcionais e não funcionais, foi retirado um momento para a construção do diagrama de caso de uso.

No diagrama de caso de uso da figura 7 vimos que a aplicação deve atender esses principais requisitos.

Figura 7 – Diagrama de caso de uso



Fonte: Elaborada pelo autor (2023)

5.2.1.1 Detalhamento dos casos de uso

Tabela 4 – Manter professores

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá manter os professores no sistema. Incluindo, editando e/ou excluindo professores neste processo.
Informações preenchidas	neste caso necessita do nome do professor, quantidade máximo dos dias que o professor necessita, e as restrições de horários caso exista.

Tabela 5 – Manter disciplinas

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá manter as disciplinas no sistema. Incluindo, editando e/ou excluindo cursos neste processo.
Informações preenchidas	neste caso necessita do nome da disciplina.

Tabela 6 – Manter cursos

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá manter os cursos no sistema. Incluindo, editando e/ou excluindo cursos neste processo.
Informações preenchidas	neste caso necessita do nome do curso.

Tabela 7 – Manter semestres

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá manter os semestres no sistema. Incluindo, editando e/ou excluindo semestres neste processo.
Informações preenchidas	neste caso necessita do curso, as disciplinas daquele semestre, as restrições de horários e a descrição daquele semestre.

Tabela 8 – Manter laboratórios

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá manter os laboratórios no sistema. Incluindo, editando e/ou excluindo laboratórios neste processo.
Informações preenchidas	neste caso necessita da descrição do laboratório.

Tabela 9 – Manter turmas

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá manter as turmas no sistema. Incluindo, editando e/ou excluindo turmas neste processo.
Informações preenchidas	neste caso necessita do professor, da disciplina, do semestre na qual esta turma está inserida, os horários, os laboratórios que esta turma necessita, e a quantidade máxima de laboratórios que tal turma necessita.

Tabela 10 – Manter horários

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá manter os horários no sistema. Incluindo, editando e/ou excluindo horários neste processo.
Informações preenchidas	neste caso necessita do período e o dia.

Tabela 11 – Realizar login

Ator primário	usuário
Pré-condições	nenhuma.
Fluxo principal	o usuário irá entrar no sistema e irá para a tela de login.
Informações preenchidas	neste caso necessita do e-mail e senha.

Tabela 12 – Alocar horários

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá realizar a alocação dos horários.
Informações preenchidas	neste caso necessita informar os professores, disciplinas, cursos, semestres, turmas, laborátorios e restrições.

Tabela 13 – Manter restrições

Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação. E informar um semestre ou professor para associar a restrição.
Fluxo principal	o usuário irá manter as restrições de horários dos semestres e professores.
Informações preenchidas	neste caso necessita informar selecionar o professor ou semestre.

Tabela 14 – Associar laboratórios

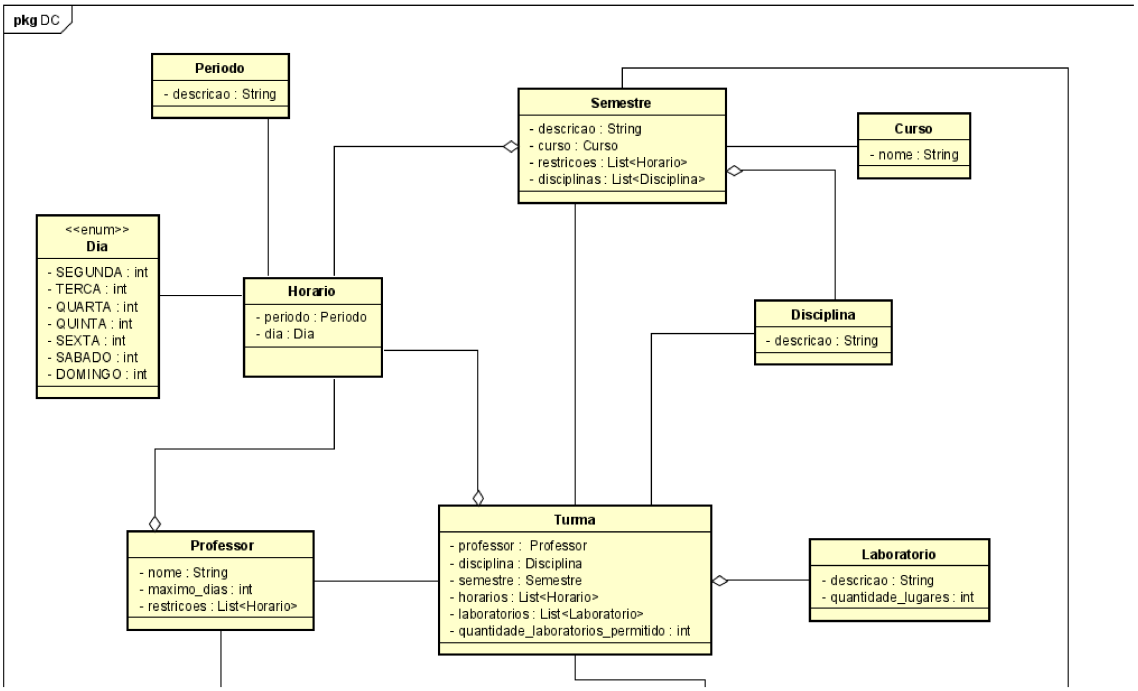
Ator primário	usuário
Pré-condições	o usuário necessita logar no sistema com usuário e senha para realizar essa ação.
Fluxo principal	o usuário irá associar laboratórios a turma.
Informações preenchidas	neste caso necessita informar selecionar a turma para informar os laboratórios.

A validação do diagrama de casos de uso foi feito com o coordenador do curso de ciências da computação. Verificou que o diagrama atendia ao requisitos impostos na investigação do problema.

5.2.2 Diagrama de classe

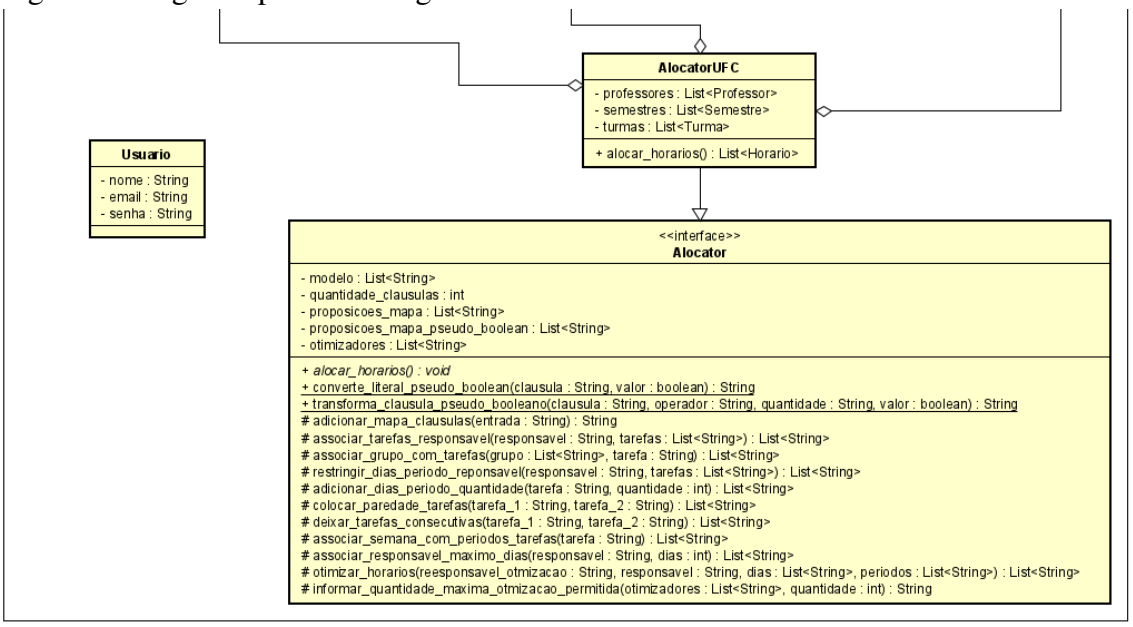
Após a elaboração do diagrama de caso de uso em conjunto com a sua validação, foi produzido o diagrama de classes, para total entendimento de como as classes e funções iriam se relacionar no projeto.

Figura 8 – Primeira parte do Diagrama de classes



Fonte: Elaborada pelo autor (2023)

Figura 9 – Segunda parte do Diagrama de classes



Fonte: Elaborada pelo autor (2023)

Neste diagrama de classe indicado pela figura 8 temos a construção da resolução do problema de alocação de horários no backend.

Uma das classes principais deste diagrama é a Turma, ela está associada a um professor, uma disciplina, o semestre na qual ela faz parte, os horários que podem ou não já estão inseridos na mesma, os seus laboratórios que podem ou não estar associados a ela e a quantidade de laboratórios que ela terá permissão de usar.

Cada horário estará associado a um dia da semana e um período, este definido pela instituição de ensino. Por exemplo na UFC temos 4 períodos H1, H2, H3 e H4 que são 08:00 às 10:00, 10:00 às 12:00, 13:30 às 15:30 e 15:30 às 17:30 respectivamente.

Os semestres estão associados a um curso, e cada semestre terá suas disciplinas associadas, e restrições, caso seja necessário restringir um horário. Por exemplo as disciplinas dos primeiros semestre não podem cair em uma sexta feira.

O professor terá o seu nome, o máximo de dias que podem ser associados a ele na semana, essa opção seria para otimização da própria alocação, e suas restrições. Por exemplo um professor que tenha que estar em Fortaleza segunda-feira e terça-feira, já poderia colocar na sua restrição, fazendo com que o algoritmo já entenda e mapeie para o restante dos dias.

Por fim nesta primeira parte temos o laboratório, que terá sua descrição e a quantidade de lugares, a disciplina que terá somente sua descrição e o curso com o seu nome.

Na Figura 9 temos o restante das classes do sistema.

A interface Alocator é a principal classe do projeto, nela está concepção da resolução do problema. Nela está contido os seguintes atributos e métodos:

5.2.2.1 Atributos do Alocator

- Modelo
  - Conjunto de cláusulas formadas para gerar a fórmula.
- Quantidade de cláusulas
  - Quantidade total de cláusulas geradas pelo modelo
- Proposições mapa
  - Listagem de todas as proposições geradas
- Proposições mapa MaxSat
  - Listagem de todas as proposições geradas na forma de MaxSat
- Otimizadores
  - Listagem de todos os otimizadores que foram utilizados na construção do problema.

5.2.2.2 Métodos do Alocator

- Alocar horários
  - Método responsável por alocar os horários, é um método abstrato, então as classes serão filhas do Alocator deverão implementar a mesma com as suas regras para fazer a alocação.
- Converte literal em MaxSat
  - Método estático, responsável por pegar uma cláusula na escrita de inteiros, por exemplo, 3009 V 3001 V 3010 e converte para o algoritmo responsável iniciar a resolução.
- Transforma cláusula MaxSat
  - Pega o MaxSat feito pelo método descrito anteriormente e associa a um operador matemático e sua quantidade necessária para a cláusula ser satisfeita.
- adicionar mapa cláusulas
  - Responsável por pegar cada literal da alocação e colocar nos mapeadores, por exemplo, o literal $PROFESSOR-TURMA-H1$ ele pegará e colocará na lista de proposições de mapa e associará a um número inteiro maior que zero e colocará esse número no mapa de pseudo booleano, dessa maneira o algoritmo no final saberá quais literais

são verdadeiros para alimentar os horários do sistema.

- Associar tarefas a responsável
  - Esse método terá a responsabilidade de pegar um responsável e associar a lista de tarefas. Exemplo seria um professor e associa-lo a suas turmas, ou um semestre associado com as turmas.
- Associar grupo com tarefas
  - Caso exista grupos que precisam associar 1 ou mais para um responsável, por exemplo, a turma de Fundamentos de programação necessita de até dois laboratórios, então esse método pegará o grupo, no caso laboratórios e irá associar a turma.
- Restringir dias ou períodos para responsável
  - Caso haja necessidade de informar ao algoritmo que tal professor ou semestre não pode ocorrer em um período ou horário específico.
- Adicionar dias para período e quantidade
  - Para literais que necessitam aparecer em uma grade mais vezes, pode ser colocar a quantidade de dias que será preciso para tal. Exemplo, POO necessita acontecer duas vezes na semana.
- Colocar tarefas pareadas
  - Existem tarefas que são executadas no mesmo tempo, por exemplo, caso haja dois professores de uma mesma disciplina, pode-se colocar duas turmas no mesmo horário.
- Deixar tarefas consecutivas
  - Existem tarefas que necessitam ficar próximas, um exemplo são as mesmas disciplinas, de cursos diferentes, para o mesmo professor, então esse método fará com que essas turmas fiquem uma após a outra no calendário de horários.
- Associar semana com períodos e tarefas
  - Esse método é bem especifico para a grade da UFC, basicamente pegará as tarefas, associando ao número de dias, e irá forçar as tarefas a ficarem nesses dias, por exemplo, POO necessita ficar no horário de segunda 08:00 às 10:00, então deverá também ficar na quarta no mesmo horário.
- Associar responsável a máximo de dias
  - Basicamente esse método é para colocar um máximo de dias, para que o professor possa se organizar com outras pendências.
- Otimizar horários

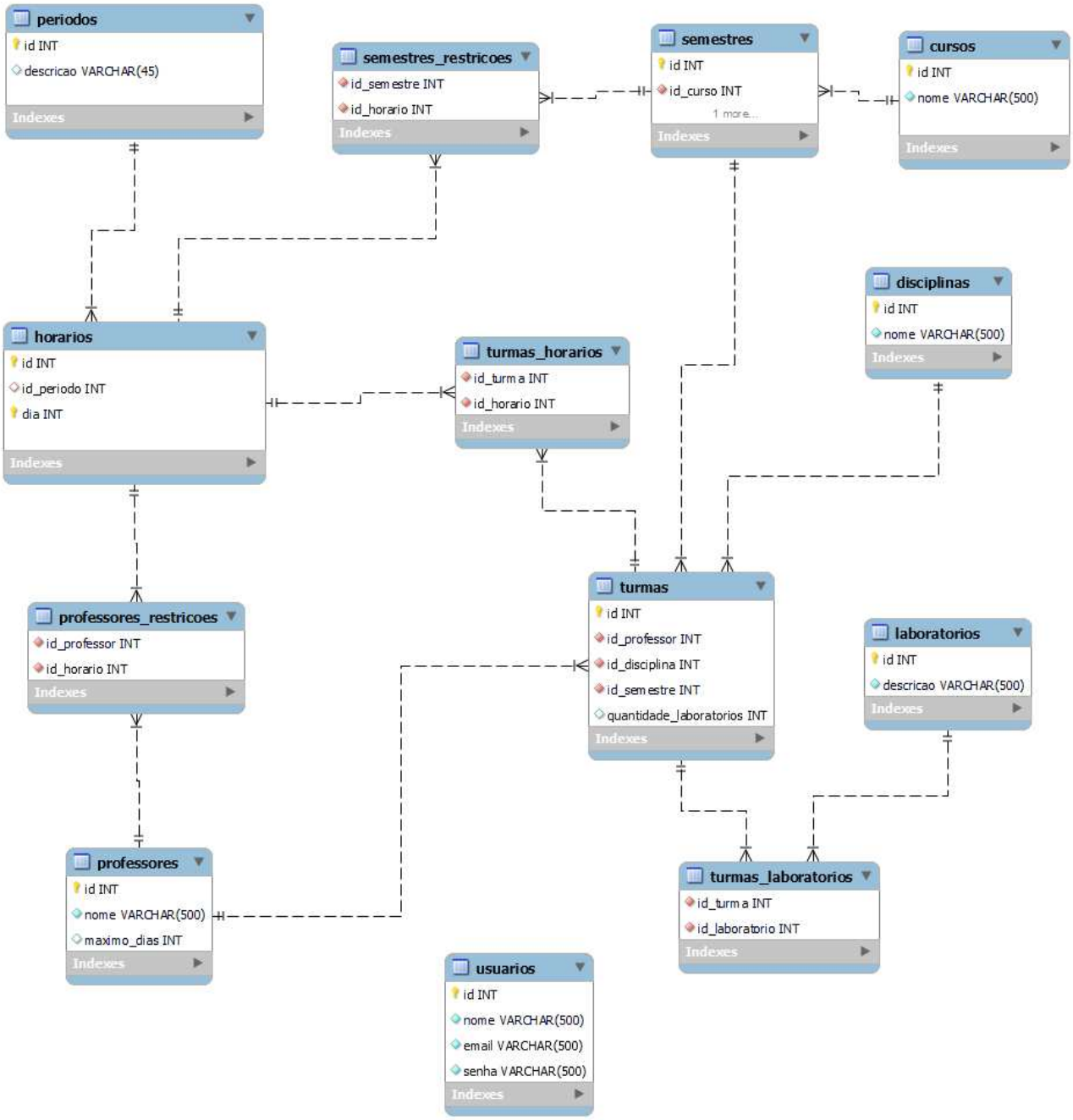
  - Caso você queira passar uma lista de horários e/ou dias para um responsável para ajudar na otimização do mesmo.
- Informar quantidade máxima de otimização permitida
  - Informa ao solver a quantidade máxima de otimizações que são permitidas.

Lembrando que a construção da classe de Alocator é pensada para ser algo genérico, então as subclasses que herdarão ela poderão trabalhar da forma exigida, dessa forma não há necessidade de trabalhar com todos os métodos, mas eles estarão acessíveis, e caso as subclasses trabalhem de maneiras totalmente diferentes então as mesmas podem utilizar de seus próprios métodos e variáveis.

5.2.3 Diagrama de banco de dados

Feito a construção das classes do sistema, e mapeado todas as iterações entre elas, foi então criado a diagramação do banco de dados, dessa forma mapeando as características das classes como por exemplo: a criação de tabelas auxiliares para criação do conceito de muitos para um.

Figura 10 – Diagrama entidade relacional



Fonte: Elaborada pelo autor (2023)

5.2.4 Informações quanto à programação

5.2.4.1 Linguagem de programação

Para iniciar o desenvolvimento do backend, foi feito um levantamento de linguagens em potencial para tal desenvolvimento.

No início, a construção do código foi realizada em C++. No entanto, à medida que o projeto avançava, tornou-se evidente a necessidade de transferir a responsabilidade para a web. Utilizar o C++ para esse desenvolvimento se tornaria complexo e demandaria recursos adicionais em termos de tempo e desenvolvimento.

Nesse contexto, foi considerado explorar soluções já utilizadas por outros programadores da área. Assim, optou-se pela construção do projeto em JavaScript, utilizando o Node.js. O Node.js seria uma opção viável, pois oferecia um conjunto de recursos necessários para a construção de uma API.

Entretanto, durante o uso do Node.js, surgiram dificuldades na integração com o algoritmo necessário para resolver o problema de alocação. Tornou-se inviável prosseguir com a construção da API sem resolver esse problema.

Por fim, foi decidido criar a API em Python, que se mostrou a melhor opção. Além de fornecer todas as ferramentas necessárias para a construção de APIs, o Python também oferecia a capacidade de executar o algoritmo e resolver o problema de alocação utilizando o pseudo boolean solver.

Os pacotes que foram levantados para a criação do backend em python foram:

- *flask*
  - para a criação do servidor HTTP.
- *flask restx*
  - para introduzir a documentação juntamente com a programação, facilitando assim um futuro front-design construir sem necessitar entender do backend.
- *peewee*
  - ORM para construção de banco de dados.

5.3 Validação do Projeto

A validação do projeto ocorreu nas seguintes etapas:

1. A validação da ideia, foi feito a elaboração da investigação do problema visto na seção 5.1. Dessa maneira, foi considerado o problema, montado uma ideia inicial e revisado com o Prof. Dr. Alexandre Arruda Matos responsável pela alocação de salas e horários desde 2019.
2. Foi construído a diagramação de caso de uso, onde se teve a ideia inicial de funcionalidades e o que o sistema deveria atender e validado pelo coordenador de ciências da computação.

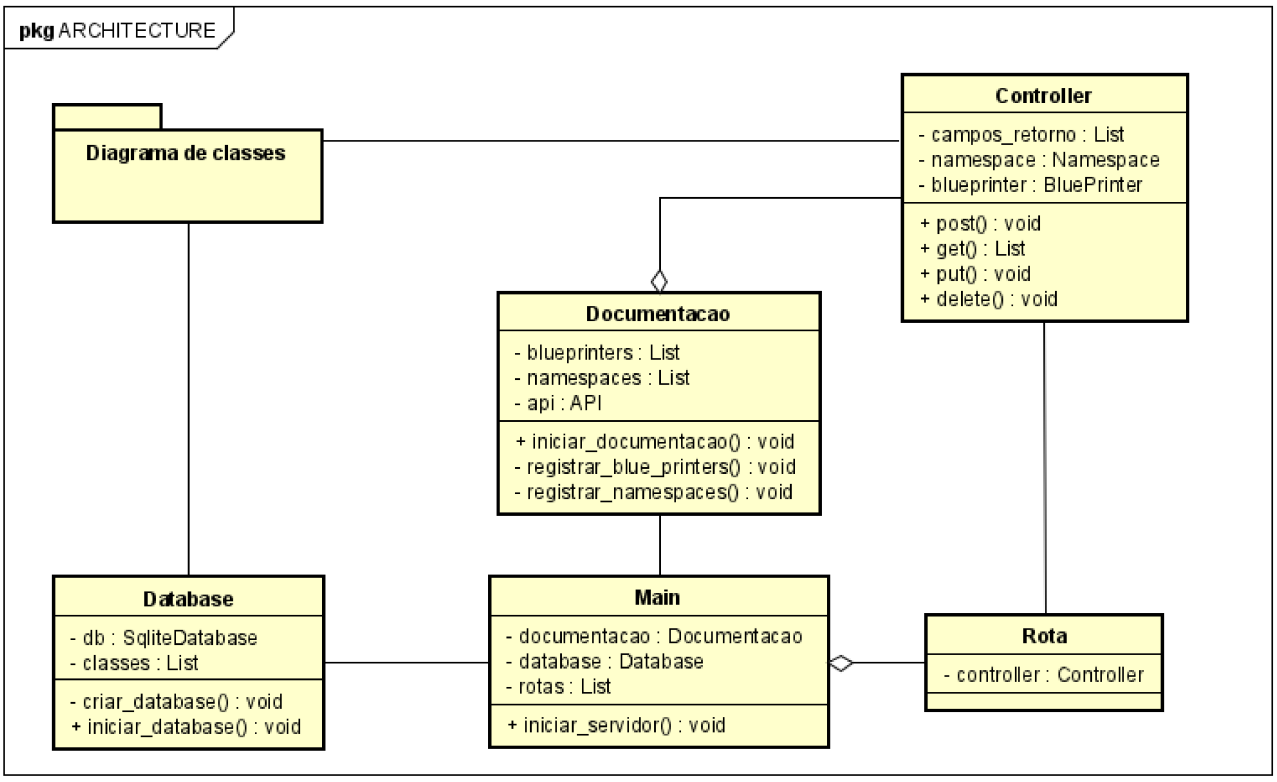
3. Após a construção dos diagramas de classe e banco de dados foi validado também pelo coordenador de ciências da computação para realmente verificar se não havia informações fora do escopo abordado.
4. Realizado a construção inicial do projeto em C++, para validar a manutenção de dados que o projeto necessitava, e verificar se todo requisitos seriam atendidos.

5.4 Implementação da Solução

Após a construção da diagramação foi iniciada a fase de desenvolvimento, dessa maneira foi utilizado as ferramentas que estão listadas na subseção 5.2.4.1.

Na Figura 11 está a estrutura de criação do backend.

Figura 11 – Diagrama da arquitetura do backend



Fonte: Elaborada pelo autor (2023)

O projeto inicia na classe Main, nela está contido a estrutura inicial e o início do servidor. Nela está associada a classe documentação que por sua vez controla toda a parte visual e de dados para que futuros desenvolvedores frontend para aplicação possam utilizar das ferramentas sem necessariamente entender o código do backend.

O pacote descrito como diagrama de classes são o conjunto de classes descritas nas figuras 8 e 9.

A classe documentação tem a API responsável por criação da documentação em SI, esta que foi feita utilizando a biblioteca do python flask rest, tem a lista de blueprinters e namespaces, onde é a localização física dos controladores e sua referência na memória respectivamente. E por fim terá o registro dos mesmos.

A classe Main ficará responsável por escutar as requisições, dessa maneira terá uma listagem de rotas, essas rotas são URL's acessadas pelo frontend ou usuários interessados pela API, e cada rota terá um controlador para a determinada tarefa.

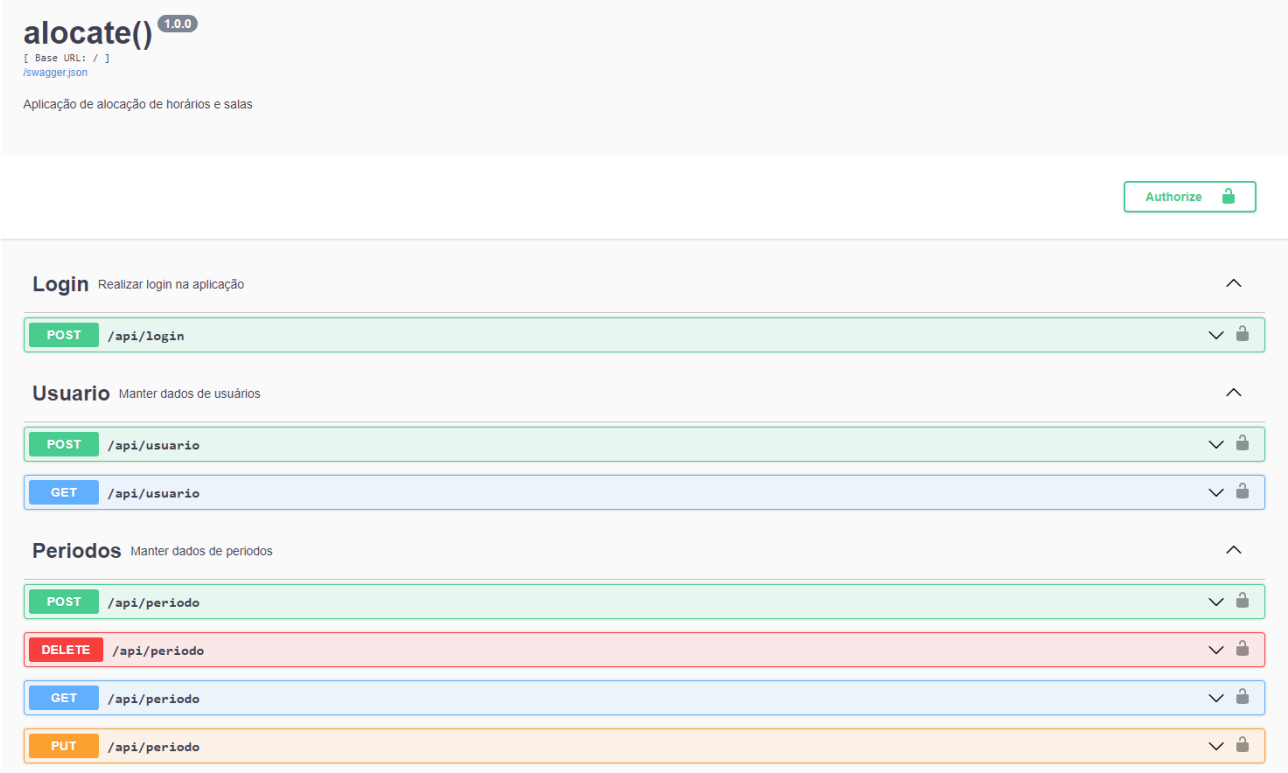
O controlador será responsável por estar repassando para a rota os seguintes métodos: POST, GET, PUT e DELETE. Esses métodos estão associados as requisições que o usuário fará na API e a sua funcionalidade. Cada controlador deverá ter uma lista de campos de retorno, para indicar na documentação como se dará os parâmetros. Terá o seu namespace e blueprinter já mencionados acima. Ele também irá se associar com o diagrama de classes, pois será necessário para utilizar da lógica por trás da alocação.

Por fim a Main manterá a associação com a classe de database. Esta classe está responsável por criar e iniciar o banco de dados para a persistência das informações presentes na aplicação. Ela terá vínculo com o diagrama de classes, então as classes irão ser repassadas para elas para serem criadas, alteradas, deletadas e listadas.

5.4.1 Documentação

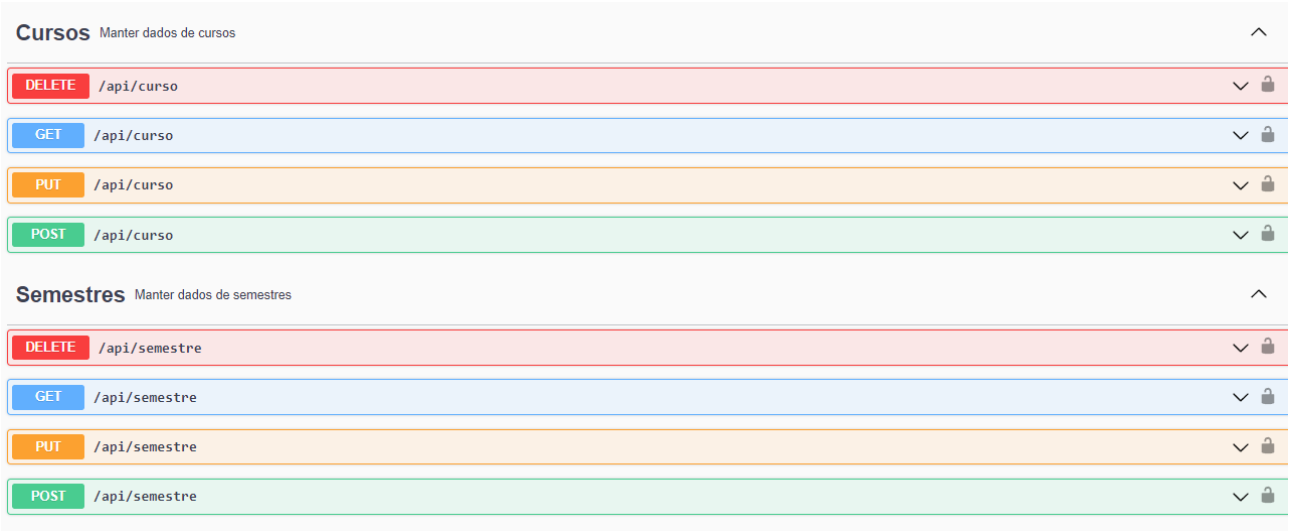
Para que o frontend consiga utilizar as rotas da aplicação, foi criado juntamente com o projeto em python a documentação, então, nessa documentação é mostrada todas as rotas e o método para realizar a autenticação no backend. A parte visual pode ser vista nas figuras 12 à 17 mostradas a seguir.

Figura 12 – Documentação do backend



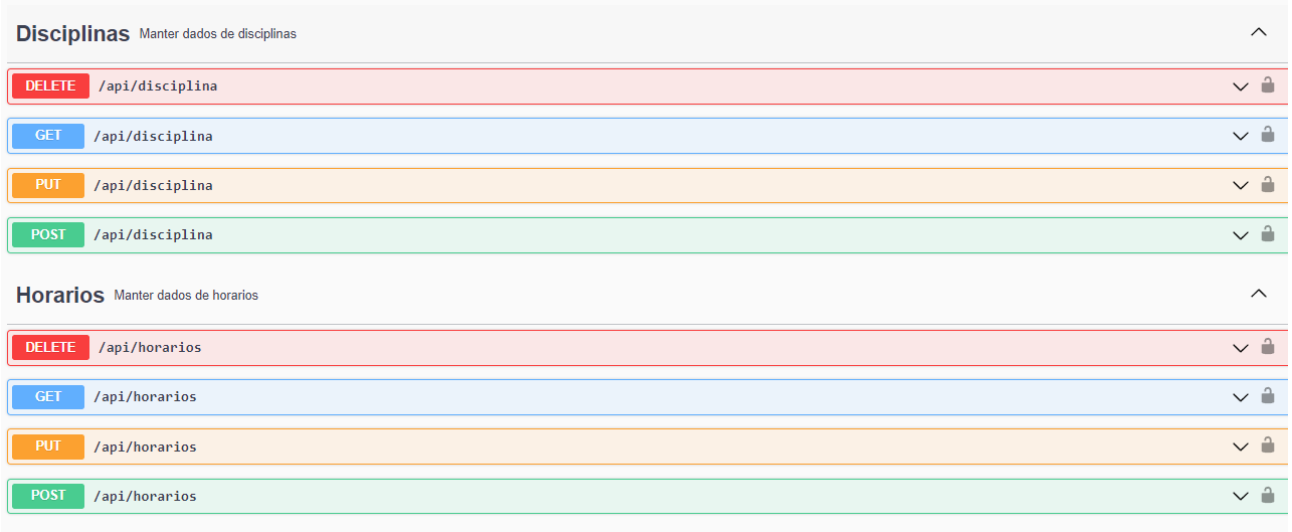
Fonte: Elaborada pelo autor (2023)

Figura 13 – Documentação do backend 2



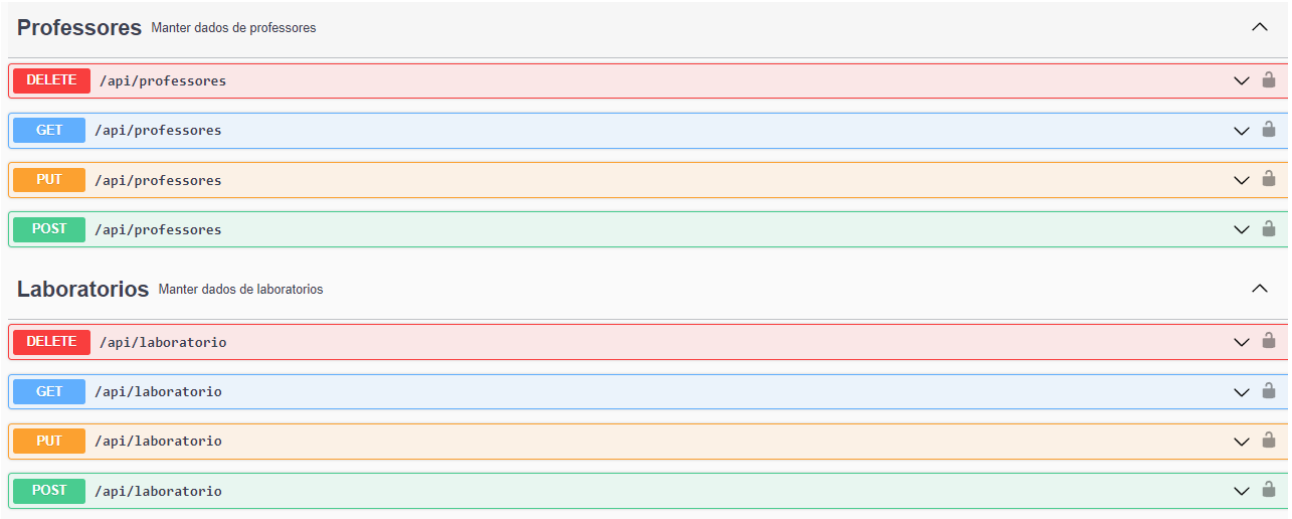
Fonte: Elaborada pelo autor (2023)

Figura 14 – Documentação do backend 3



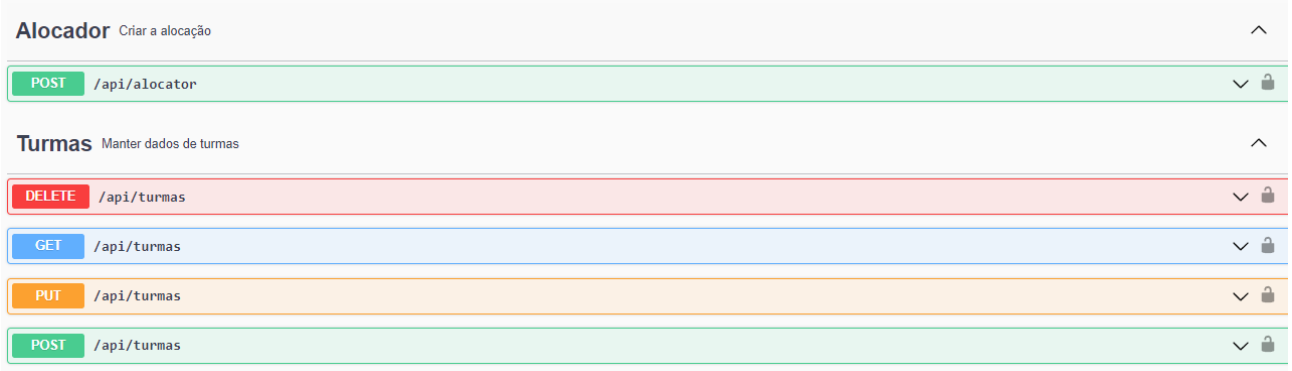
Fonte: Elaborada pelo autor (2023)

Figura 15 – Documentação do backend 4



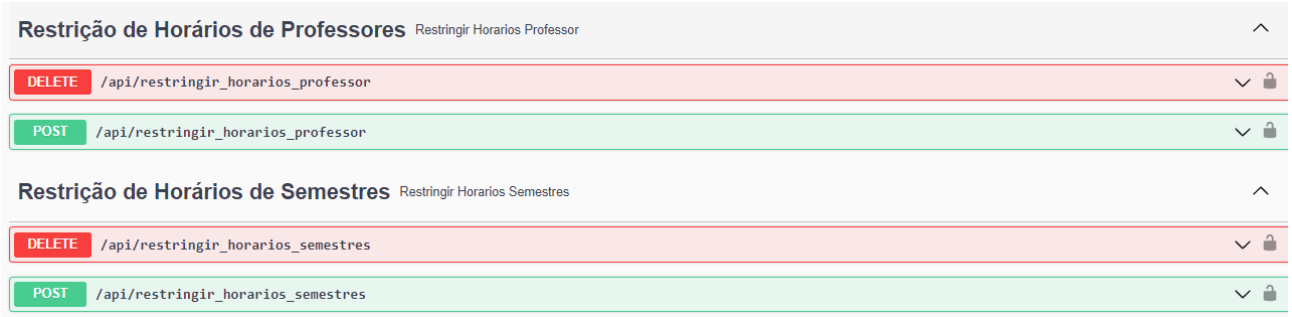
Fonte: Elaborada pelo autor (2023)

Figura 16 – Documentação do backend 5



Fonte: Elaborada pelo autor (2023)

Figura 17 – Documentação do backend 6

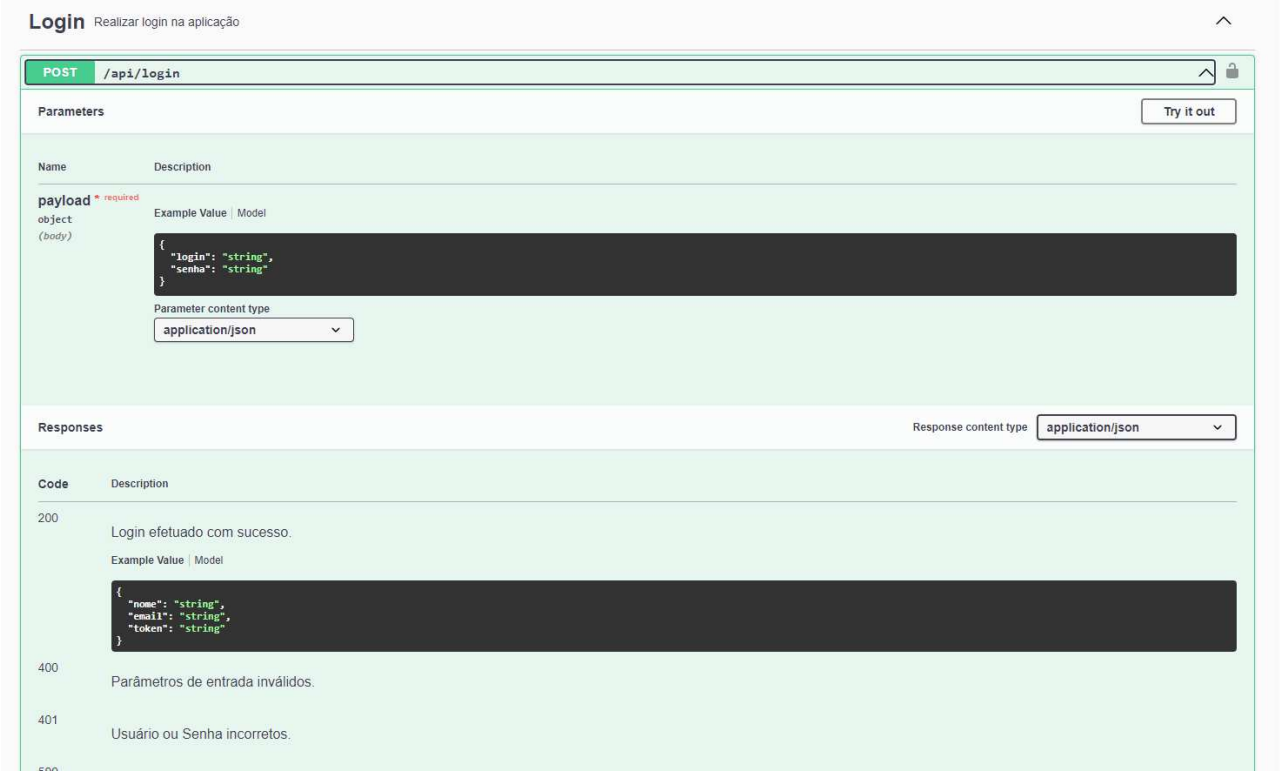


Fonte: Elaborada pelo autor (2023)

Essa documentação conta com a versão que ela está, a URL base para acesso dos eventos, a autorização para que seja utilizado pelo frontend, e por fim as rotas juntamente com seus métodos de GET, POST, PUT e DELETE.

É informado também na documentação para cada URL de acesso os parâmetros que o desenvolvedor precisa passar para que o servidor retorne a informação correta, e quais são os dados retornados, e caso ocorra erros durante o processamento daquele evento também vai estar dispostos as mensagens de erro e seus códigos. A figura 18 mostra como é documentado essa parte.

Figura 18 – Parâmetros de requisição



Fonte: Elaborada pelo autor (2023)

5.5 Avaliação da Implementação

A avaliação da implementação se deve nos semestre de 2022.1 e 2022.2, onde foi utilizado um algoritmo para a resolução da grade de horários da UFC, então estamos colocando esse algoritmo para o *backend* agora conseguir executar, dessa forma será possível quando o *frontend* estiver completo a disponibilizar essa ferramenta para os funcionários da UFC.

6 CONSIDERAÇÕES FINAIS

O problema da alocação de recursos em horários é uma questão bastante recorrente na literatura, e, como resultado, muitas instituições enfrentam dificuldades para realizar alocações de forma abrangente e ágil. Investir em soluções tecnológicas pode reduzir significativamente o esforço necessário para executar esse trabalho.

Com o propósito de abordar o desafio da alocação de horários e salas, este trabalho utilizou como cenário a Universidade Federal do Ceará (UFC). Nesse contexto, foram identificados os requisitos e regras de negócio específicas impostas pela instituição. Além disso, foram projetados os diagramas necessários para compreender a problemática e orientar o desenvolvimento da aplicação. Por fim, foi implementado o *backend* em questão, resultando em uma solução completa e funcional adaptada ao ambiente da UFC.

Neste trabalho, é proposta a utilização do material desenvolvido não apenas para a Universidade Federal do Ceará, mas também para outras instituições de ensino interessadas. No documento apresentado, foi abordada uma técnica que envolveu a utilização da classe *Alocator* para permitir a refatoração do código, tornando possível a inserção de novas ideias que possam ser aplicáveis em diferentes contextos educacionais.

Além disso, uma das principais metas deste trabalho foi fornecer uma documentação abrangente para orientar os futuros desenvolvedores, tornando o uso da ferramenta mais acessível, sem a necessidade de compreender detalhes do código interno. Isso possibilita uma maior facilidade de uso e adaptação da solução não apenas na Universidade Federal do Ceará, mas também em outras instituições de ensino que queiram utilizar a abordagem proposta, incluindo a utilização da classe *Alocator* para facilitar a alocação de horários e salas de forma eficiente e flexível.

Ao longo deste projeto, um dos principais desafios encontrados foi incorporar as restrições impostas pela instituição de forma flexível, permitindo que fossem alteradas e adaptadas para diferentes cenários. Essa tarefa exigiu um período significativo de reflexão e ajustes para ser concluída satisfatoriamente. A necessidade de garantir a adaptabilidade das restrições para outras situações adicionou complexidade ao processo, o que foi cuidadosamente analisado e aceito para a conclusão bem-sucedida do projeto.

Com isso, o trabalho desta pesquisa alcança de forma efetiva os seus objetivos estabelecidos e apresenta potencial para contribuir em futuros projetos. Essas possibilidades incluem a adaptação do trabalho para outras instituições de ensino e a criação de um frontend

web ou mobile para utilizar o backend desenvolvido. Com essas perspectivas em mente, a pesquisa oferece uma base sólida para o desenvolvimento contínuo e aprimoramento do sistema, visando atender a diferentes necessidades e proporcionar uma experiência ainda mais abrangente e funcional para os usuários.

REFERÊNCIAS

ACHá, R. A.; NIEUWENHUIS, R. **Curriculum-based course timetabling with SAT and MaxSAT**. 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05), 205.

BAGGER, N.-C. F.; KRISTIANSEN, S.; SORENSEN, M.; STIDSEN, T. R. **Flow Formulations for Curriculum-based Course Timetabling**. Springer US, 2019.

BANBARA, M.; INOUE, K.; KAUFMANN, B.; SCHAUB, T.; SOH, T.; TAMURA, N.; WANKO, P. **teaspoon: Solving the Curriculum-Based Course Timetabling Problems with Answer Set Programming**. Springer US, 2019.

BARBOSA, D. M.; BAX, M. **A Design Science como metodologia para a criação de um modelo de Gestão da Informação para o contexto da avaliação de cursos de graduação.** Revista Ibero-Americana De Ciência Da Informação, 2017.

BURKE, E.; WERRA, D. de; KINGSTON, J. **Applications to timetabling. Handbook of Graph Theory**. London: Chapman Hall/CRC, 2004.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 6th. ed. [S.l.]: Pearson, 2011.

FINGER, M.; SILVA, F. S. C. da; MELO, A. C. V. de. **Lógica para Computação**. 2. reimpr. da 1. ed. de 2006: Cengage Learning, 2006. v. 2.

HUTH, M.; RYAN, M. **Lógica em Ciência da Computação**. 2. reimpr. da 1. ed. de 2004: Syndicate of the press of the university of Cambridge, England, 2008. v. 2.

JIN, B.; SAHNI, S.; SHEVAT, A. **Designing Web APIs**. 1th. ed. [S.l.]: O'REILLY, 2018.

MANQUINHO, V. M.; MARQUES-SILVA, J. **Satisfiability-Based Algorithms for Pseudo-Boolean Optimization Using Gomory Cuts and Search Restarts**. Ann Open Res, 2005.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. 5th. ed. [S.l.]: McGraw-Hill, 2005.

QU, R.; BURKE, E. K.; MCCOLLUM, B.; MERLOT, L.; LEE, S. **A survey of search methodologies and automated system development for examination timetabling**. Springer, US, 2009.

SCHAERF, A. **A Survey of Automated Timetabling**. Kluwer Academic Publishers, 1995.

WIERINGA, R. J. **Design Science Methodology for Information Systems and Software Enginnering**. 1th. ed. [S.l.]: Springer, 2014.