

1 DESENVOLVIMENTO DO SISTEMA DE SUPORTE À DECISÃO

Este capítulo visa apresentar todo o processo realizado para o desenvolvimento do sistema de suporte à decisão. O propósito desta aplicação é auxiliar o coordenador do curso de Ciência da Computação da UENF na alocação interativa de turmas, professores e salas. O auxílio é dado através de duas formas: a primeira é a criação automatizada da grade baseada no histórico das grades anteriores, a segunda é a identificação de conflitos, como a alocação de uma turma grande demais para a capacidade da turma, com o objetivo de auxiliar no aprimoramento da grade inicialmente gerada.

Inicia-se com a apresentação de projetos anteriores que serviram de base para o desenvolvimento do presente trabalho. Em seguida, será apresentado o acesso aos dados acadêmicos da instituição e as dificuldades encontradas para a sua obtenção. Posteriormente, será apresentado o processo de prototipagem do sistema, onde foram criados os designs iniciais das telas do sistema. Em seguida, será apresentado o processo de programação do sistema, que foi dividido em três grandes categorias que visavam entregar o sistema de forma gradual e funcional.

Por fim, será apresentado o cerne do sistema que é a identificação e visualização de conflitos, ou seja, os casos em que a alocação dos recursos gera algum comportamento minimamente indesejável como por exemplo a alocação de uma sala a duas turmas ao mesmo tempo. Serão apresentados os tipos de conflitos que o sistema é capaz de identificar, como eles são visualizados e como o sistema lida com cada um deles. Conclui-se então com a apresentação do preenchimento do banco de dados com dados reais.

1.1 Projetos Anteriores

Antes do desenvolvimento do presente trabalho, outros projetos já foram idealizados e até mesmo desenvolvidos. Dentre eles, dois se destacam por terem sido os precursores do atual projeto. O primeiro deles, que foi idealizado, mas não desenvolvido, almejava apresentar uma visualização da progressão dos alunos na grade curricular. O segundo, que foi desenvolvido, realizava o cálculo da demanda, ou seja, calculava quantos e quais eram os alunos que poderiam se inscrever nas disciplinas. Ambos os projetos se mostram como abordagens paralelas de auxiliar no planejamento do curso de Ciência da Computação da UENF, sendo eles também sistemas de suporte à decisão, mesmo que em menor escala.

Deve-se comentar também sobre os dois trabalhos anteriores desenvolvidos por (??) e (??) que iniciaram os estudos sobre a alocação de turmas no curso de Ciência

da Computação na UENF, mas que, como já foram citados previamente, não se mostra necessária a repetição.

1.1.1 Andamento dos Alunos

Como interesse próprio, cogitou-se o desenvolvimento de uma plataforma onde se pudesse ver em que ponto os alunos se encontram em relação ao andamento de seus cursos. Para isso, seria necessária a obtenção dos dados dos alunos, seja por parte dos mesmos, do coordenador ou por integração com o sistema acadêmico. Com estes dados, seria possível criar uma interface que mostrasse o andamento dos alunos, quais matérias já foram cursadas, quais estão sendo cursadas e quais ainda faltam. Além disso, seria possível mostrar quais matérias são pré-requisitos para outras. Assim, o aluno e a coordenação poderiam ter uma visão geral de seu andamento e de quais matérias ele precisará cursar para se formar. Esse projeto não saiu do mundo das ideias, entretanto, lá permaneceu sendo maturado.

Figura 1 – Andamento do aluno no Sistema Acadêmico

9º Período				
Disciplina	C.H.	Cumprida com		
INFO1130 - Projeto de Monografia	136	INFO1130	Inscrição	2023/1
10º Período				
Disciplina	C.H.	Cumprida com		
INFO1127 - Estágio Supervisionado	204	INFO1127	Inscrição	2023/2
INFO1131 - Monografia	136	INFO1131	Inscrição	EM CURSO
Disciplinas Optativas Eletivas				
Exigidas: 306 horas Obtidas: 272 horas				
<input checked="" type="checkbox"/> Somente aproveitadas				
Disciplina	C.H.	Cumprida com		
INFO1220 - Fundamentos de Processamento de Imagens	68	INFO1220	Inscrição	2023/1
INFO1133 - Tópicos Especiais em Inteligência Artificial II - Sistemas Inteligentes	68	INFO1133	Aproveitamento de atividade	2022/2E
INFO1218 - Tópicos Especiais em Simulação Computacional II: Heurísticas e Complexidade	68	INFO1218	Inscrição	2022/2
INFO1226 - Tópicos Especiais em Computação II: Programação de Aplicações de Microcontrolador Digital	68	INFO1226	Aproveitamento de atividade	2021/2

Fonte: autoria própria

Anos após a idealização dessa funcionalidade, o Sistema Acadêmico da UENF passou a disponibilizar uma funcionalidade semelhante, como pode ser visto na Figura 1. Nela, é possível ver o andamento do aluno por período, incluindo as disciplinas Eletivas Livres e Eletivas Optativas. Entretanto, essa visualização tabular não ilustra o andamento

de todos os alunos simultaneamente, sendo necessário navegar individualmente por cada um deles.

1.1.2 Cálculo de Demanda

Ao longo dos semestres, foi percebido que durante o intervalo entre os semestres, os alunos precisam se inscrever nas matérias que desejam cursar no semestre seguinte. Para isso, é necessário que o coordenador saiba quantos alunos estão interessados em cada matéria para que ele possa definir quantas turmas serão abertas. Com este fim em mente, o coordenador dispõe de algumas alternativas como estimar quantos alunos se inscreverão em cada disciplina, checar manualmente no sistema acadêmico quais alunos podem fazer cada matéria, ou então obter diretamente dos alunos através de um formulário em quais disciplinas cada um dos alunos tem a intenção de cursar.

O método que o atual Coordenador de Ciência da Computação utiliza consiste em baixar o extrato de todos os alunos do curso e tabelar no Excel qual é o andamento de cada um dos alunos, para que assim, através da análise manual, pudesse ver qual é o andamento de cada um e de quantos alunos demandam quais disciplinas.

Entretanto, todas essas alternativas são trabalhosas e propensas a erros. Sendo assim, foi pensado em uma forma de automatizar esse processo. Foi então elaborado um código em Python¹ que atualmente se encontra no GitHub². Este código tem como entrada os extratos de matrícula dos alunos e como saída a listagem das disciplinas demandadas e a listagem dos alunos que demandam cada disciplina.

Código 1.1 – Obter demanda por extratos em PDF

```

1 import code_1_set_working_directory      as swd
2 import code_2_get_pdf_list             as gpl
3 import code_3_get_string_from_pdf     as gsp
4 import code_4_structure_data_from_text as sdt
5 import code_5_filter_structured_data  as fsd
6 import code_6_get_demand_list         as gdl
7 import code_7_merge_demands          as mgd
8 import code_8_output_demand_as_txt   as odt
9
10 swd.set_cwd()
11 pdf_paths      = gpl.get_pdf_list()
12 texts          = gsp.get_pdf_text(pdf_paths)
13 structured_data = sdt.structure_data(texts)
14 approved_codes = fsd.get_approved_codes(structured_data)
15 demands        = gdl.get_demand_list(approved_codes)
16 demands_and_values = mgd.get_merged_demands(demands)

```

¹ <<https://www.python.org>>

² <https://github.com/jvfd3/university_demand>

```
17 odt.output_to_txt(demands_and_values)
```

Este código foi desenvolvido em 8 etapas, vistas no Código 1.1, cada uma com um arquivo separado. Para alcançar a lista das demandas, é necessário primeiro obter a lista dos arquivos em formato PDF que serão processados, em seguida extrair seus dados com a biblioteca *PDFMiner*³, estruturar os dados obtidos, filtrar os dados estruturados, obter a demanda de cada disciplina, juntar as demandas de cada disciplina e salvar os dados obtidos em um arquivo de texto.

Embora o código cumpra com seu objetivo, apresenta algumas características limitantes. A primeira é que os PDFs precisam ser obtidos manualmente, um por um, pelo coordenador, sendo ela por si só uma tarefa extenuante, o que não é desejado. Além disso, o seu uso não é muito intuitivo, sendo necessário que o usuário lide com o prompt de comando e instale as dependências necessárias, o que acaba trazendo uma dificuldade a mais ao usuário. O código também apresenta limitações por sistema operacional, não sendo garantido o seu funcionamento em sistemas operacionais diferentes do Windows.

1.2 Acesso aos Dados Acadêmicos da Instituição

Em sua concepção original, o presente trabalho visaria integrar o sistema desenvolvido com o atual sistema acadêmico da UENF. Essa abordagem foi descartada devido às dificuldades encontradas por parte do setor administrativo da UENF que, devido à Lei Geral de Proteção dos Dados (LGPD)⁴, não podem divulgar dados dos alunos, mesmo anonimizados.

Para confirmação das informações recebidas, a LGPD foi estudada e talvez o presente estudo recaísse na alínea b do inciso 2º do artigo 4º do capítulo 1 da Lei Nº 13.709, de 14 de agosto de 2018. Informando este que esta lei, a LGPD, não se aplica ao tratamento de dados pessoais realizado para fins exclusivamente acadêmicos.

Segundo o Estudo Técnico sobre o tratamento de dados pessoais para fins acadêmicos⁵, é reforçado que “o tratamento de dados pessoais para fins acadêmicos deve ser sempre lícito”.

Apesar das possibilidades de meios legalmente válidos para a aquisição dos dados, optou-se por abandonar a integração com o Sistema Acadêmico e o uso de dados reais dos alunos já existentes na plataforma. Rumando-se então para uma abordagem de inserção de dados manual por parte dos usuários do sistema.

³ <<https://pypi.org/project/pdfminer/>>

⁴ <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>

⁵ <https://www.gov.br/anpd/pt-br/assuntos/noticias/sei_00261-000810_2022_17.pdf>

1.3 Prototipagem

A criação de protótipos, seguindo a abordagem tomada por ??), se mostra como essencial para que se mantenha a constante satisfação por parte dos *stakeholders* e quais mudanças sugerem ao desenvolvimento do projeto, assim reduzindo a necessidade de retrabalho e de não alcançar as expectativas do projeto. Para este fim, foram feitos os designs iniciais das telas do sistema usando o software de design Figma⁶, designs esses que representam apenas o esboço, e não um sistema funcional. Esse esboço serviu como uma base visual de como o sistema estaria no final.

1.3.1 Protótipos de Componentes

Antes de seguir com a descrição das telas, é válido descrever os componentes que foram utilizados na construção dos protótipos. Os componentes principais são os cartões de turma (Figura 2) e as caixas de seleção (Figura 3).

A Figura 2 mostra os cartões de turma que foram utilizados para representar as turmas criadas. O objetivo desses cartões é de apresentar de forma compacta o máximo possível de informações relevantes ao usuário. Considerando o caráter iterativo do desenvolvimento, foram criados diversas versões dos cartões, todos eles visando a apresentação de informações de forma clara, objetiva e intuitiva por parte do usuário.

Essas informações relevantes são as informações relacionadas à identificação da disciplina, do professor e da sala. Além dessas, são apresentadas informações adicionais como a quantidade de alunos que podem se inscrever na disciplina e alertas de possíveis conflitos.

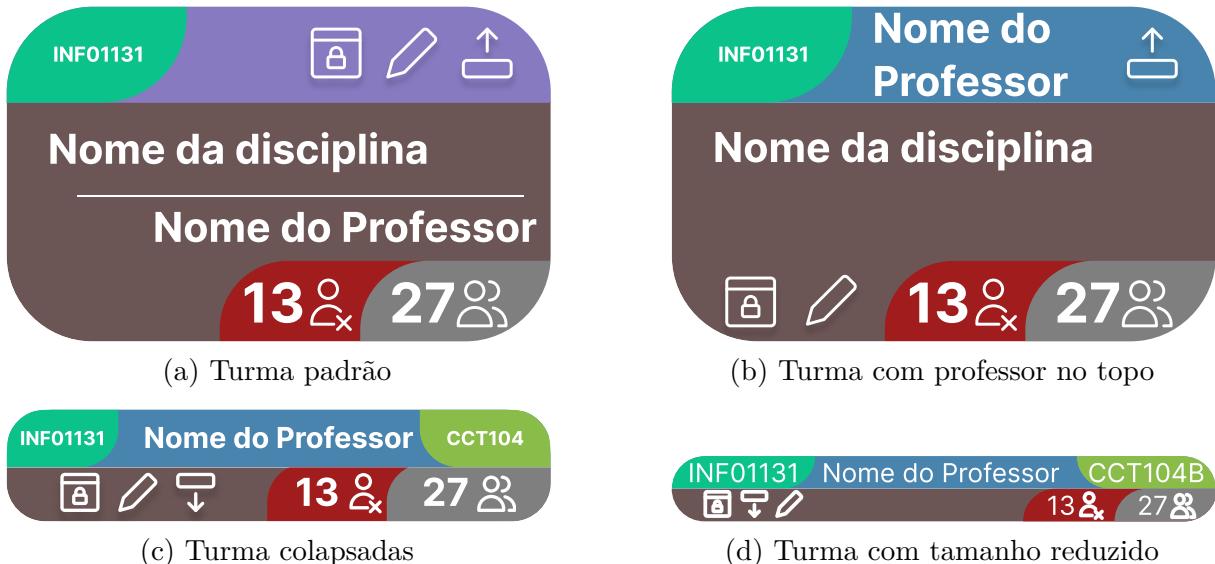
Em todas as versões dos cartões de turma, o código da disciplina está no canto superior esquerdo. No canto inferior direito está a quantidade de alunos que podem se inscrever na disciplina e, à sua esquerda, aqueles alunos que apresentam algum tipo de impedimento, como por exemplo ter alguma outra turma que ele possa se inscrever no mesmo horário. Figura 2c

Nestes cartões, estão presentes também três botões: **travar**, **editar** e **expandir/recolher**. O botão de **travar** tem como função fixar a turma no horário em que ela se encontra, impedindo que ela seja movida. O botão de **editar** tem como função abrir um painel onde é possível alterar as informações da turma. O botão de **expandir/recolher** tem como função expandir ou recolher o cartão, mostrando ou escondendo informações adicionais. Em cada uma das versões dos cartões, a posição dos botões foram alteradas.

Outras informações que também tiveram sua posição alterada foram o nome do professor, o nome da disciplina e o código da sala.

⁶ <<https://www.figma.com>>

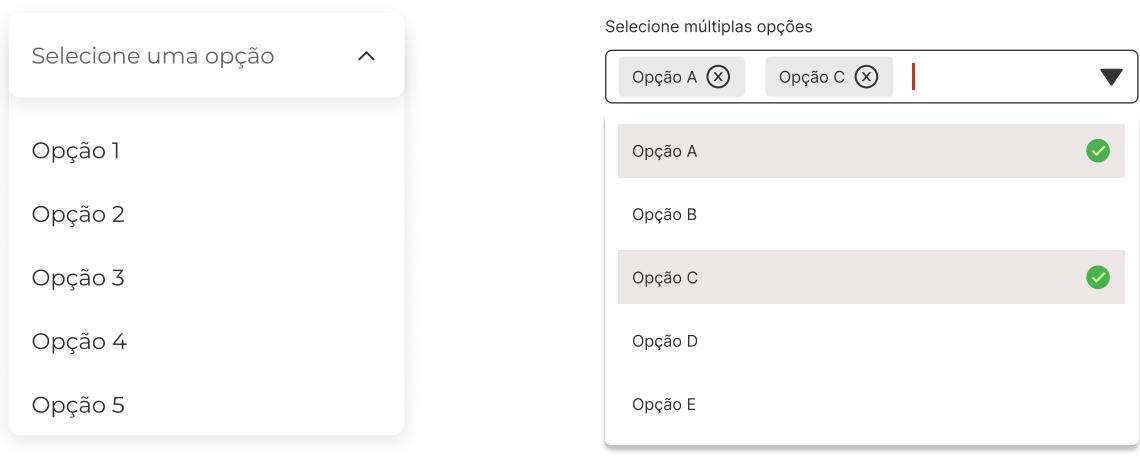
Figura 2 – Protótipos de cartões de turma



Fonte: autoria própria

Cada uma dessas versões dos cartões de turma se mostra com um propósito específico. O cartão padrão (Figura 2a) é um dos maiores cartões, o que facilita a visualização do usuário às informações importantes, o mesmo pode ser dito para o cartão com o professor no topo (Figura 2b), esse, por sua vez, distribui a posição dos botões. Já os botões colapsados (Figura 2c) e o cartão com tamanho reduzido (Figura 2d) prezam por uma maior economia de espaço, sendo úteis para quando o usuário deseja visualizar mais turmas ao mesmo tempo, como é o caso da visualização da grade horária, e por isso informam também o código da sala em que estão aloados.

Figura 3 – Protótipos de caixas de seleção



Fonte: autoria própria

A Figura 3 mostra os protótipos de caixas de seleção que foram utilizados para a seleção de dados. Na Figura 3a está a caixa de seleção onde o usuário pode selecionar apenas uma opção. Já na Figura 3b a caixa de seleção permite ao usuário selecionar várias opções.

1.3.2 Protótipos de Páginas

Aqui serão elecadas as sete páginas esboçadas para o sistema. Sendo elas a página principal, a página de seleção, a página de salas, a página de alunos, a página de disciplinas, a página de professores e a página de turmas.

A primeira, e principal, é a ilustrada pela Figura 4 que permite que o usuário arraste todas as turmas listadas até o horário desejado. A listagem das turmas a serem distribuídas é disposta em um painel à esquerda. Este painel tem como funcionalidade, fixar as turmas ainda não alocadas, como se estivessem presas por fechos de gancho e laço⁷. Assim, dispondo de um local para que turmas que turmas em processo de mudança de horário sejam armazenadas temporariamente sem que sejam perdidas.

Figura 4 – Página principal do sistema



The screenshot shows a user interface for scheduling classes. At the top left, there is a header with 'INFO1131', 'Nome do Professor' (Professor Name), and 'CCT104'. Below the header, there are two red buttons labeled '13' and '27'. The main area features a large grid for scheduling. The columns represent days of the week: SEG (Monday), TER (Tuesday), QUA (Wednesday), QUI (Thursday), and SEX (Friday). The rows represent time intervals from 08:00 ~ 09:00 up to 21:00 ~ 22:00. To the left of the grid, there is a vertical list of class names, each with a small icon and a 'fix' button. To the right of the grid, there is a vertical column of dots connected by lines, representing a temporary storage or backlog for moved items. The entire interface has a dark theme with orange and grey highlights.

Fonte: autoria própria

⁷ <https://en.wikipedia.org/wiki/Hook-and-loop_fastener>

Em seguida, temos a tela de seleção da categoria dos dados que deseja-se modificar (Figura 5), podendo esses serem sobre as turmas, salas, disciplinas, professores ou alunos. Cada uma destas tendo a capacidade de criação, leitura, edição e deleção dos dados.

Figura 5 – Página de seleção



Fonte: autoria própria

Quanto à página das salas, temos primeiro a seleção da sala na qual deseja-se fazer alterações de cadastro. Abaixo desta caixa de seleção única há um filtro de visualização das alocações de determinado ano e semestre. Nessa página pode-se também registrar algumas características da sala, como a quantidade de cadeiras e computadores, e se possui monitor, projetor, quadro de giz e quadro branco. Um exemplo de sala ainda sem turmas alocadas é representado na Figura 6. Essas últimas informações, embora não sejam essenciais para a alocação das turmas, podem ser úteis como forma de filtragem para a alocação de turmas, visto que certas disciplinas demandam salas com características específicas.

Na página dos alunos, pode-se cadastrar novos alunos informando o seu ano de entrada e a sua matrícula. Abaixo temos a visualização da grade, onde pode-se classificar cada uma das disciplinas como aprovada, reprovada e cursando. O exemplo da Figura 7 mostra a grade de um aluno inscrito em 2019.1.

Podemos definir na página das disciplinas qual seu código, nome, e o seu período esperado segundo a ementa. Além dessas informações, pode-se cadastrar quais cursos a possuem em suas ementas, quais seus pré-requisitos, os professores que a ministram e quais requisitos a mesma possui em relação às características de sala. A Figura 8 mostra a página de modificação de disciplinas.

Figura 6 – Página de modificação das informações de salas

Selecionar	Quantidade de cadeiras					
Salas	40					
	<input checked="" type="checkbox"/> Monitor					
	<input type="checkbox"/> Projetor					
Quantidade de Computadores	<input checked="" type="checkbox"/> Quadro de giz					
0	<input checked="" type="checkbox"/> Quadro branco					
Código da sala	Sala 2 - P10 - 2º. Semestre 2023 Capacidade: 40 alunos					
2 - P10	SEGUNDA	TERÇA	QUARTA	QUINTA	SEXTA	SÁBADO
105						
109-A						
109-B						
203-A						
205 E1						
Ano	Semestre					
2024	2					
	1					
	2					
	Verão					
	15:00 - 16:00					
	16:00 - 17:00					
	17:00 - 18:00					
	18:00 - 19:00					
	19:00 - 20:00					
	20:00 - 21:00					
	21:00 - 22:00					

Fonte: autoria própria

Figura 7 – Página de modificação das informações de alunos



Fonte: autoria própria

Figura 8 – Página de modificação das informações de disciplinas

Fonte: autoria própria

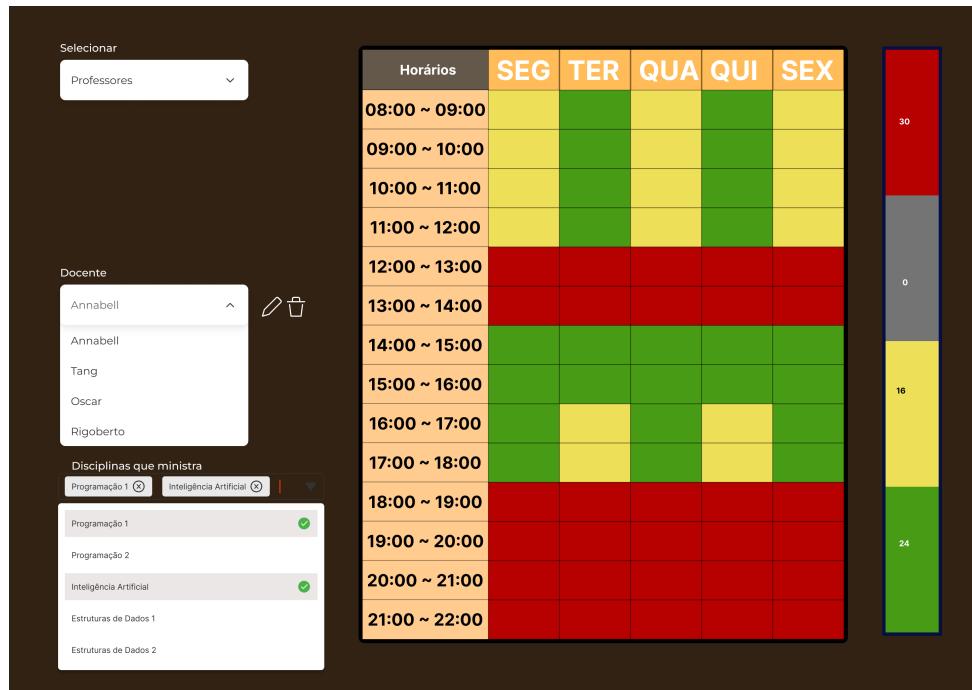
Na página de professores (Figura 9), temos a relação de disciplinas que os mesmos estão passíveis de ministrar, e também quais são suas preferências de horários ao longo da semana. Embora não seja essencial, essa informação pode ser útil para a alocação de turmas, pois alguns professores podem ter preferência, ou até mesmo não estarem disponíveis para ministrar aulas em determinados horários. Um exemplo deste caso seriam os bolsistas para docência complementar, dado que os mesmos podem estar também vinculados a outras instituições. E nesses casos, aquele que estiver desenvolvendo a grade horária pode alocar as turmas de acordo com a disponibilidade dos professores.

Por fim, na Figura 10, temos a junção de todas as informações registradas acima. Nela, podemos alocar os seus horários, definindo o ano, semestre, dia, hora de início e duração em que será ministrada. Também é necessário que seja definido em que sala cada um de seus horários estará alocada. Além de informar, também, qual professor a lecionará e a qual disciplina ela se refere.

Ainda na Figura 10 temos alguns exemplos de conflitos percebidos. O primeiro, com a cor amarelada, informado que o tempo de duração do segundo horário da turma não condiz com a preferência pessoal do professor selecionado. Este conflito não é impeditivo, entretanto, se possível, um outro horário poderia ser encontrado para atender melhor às preferências do professor.

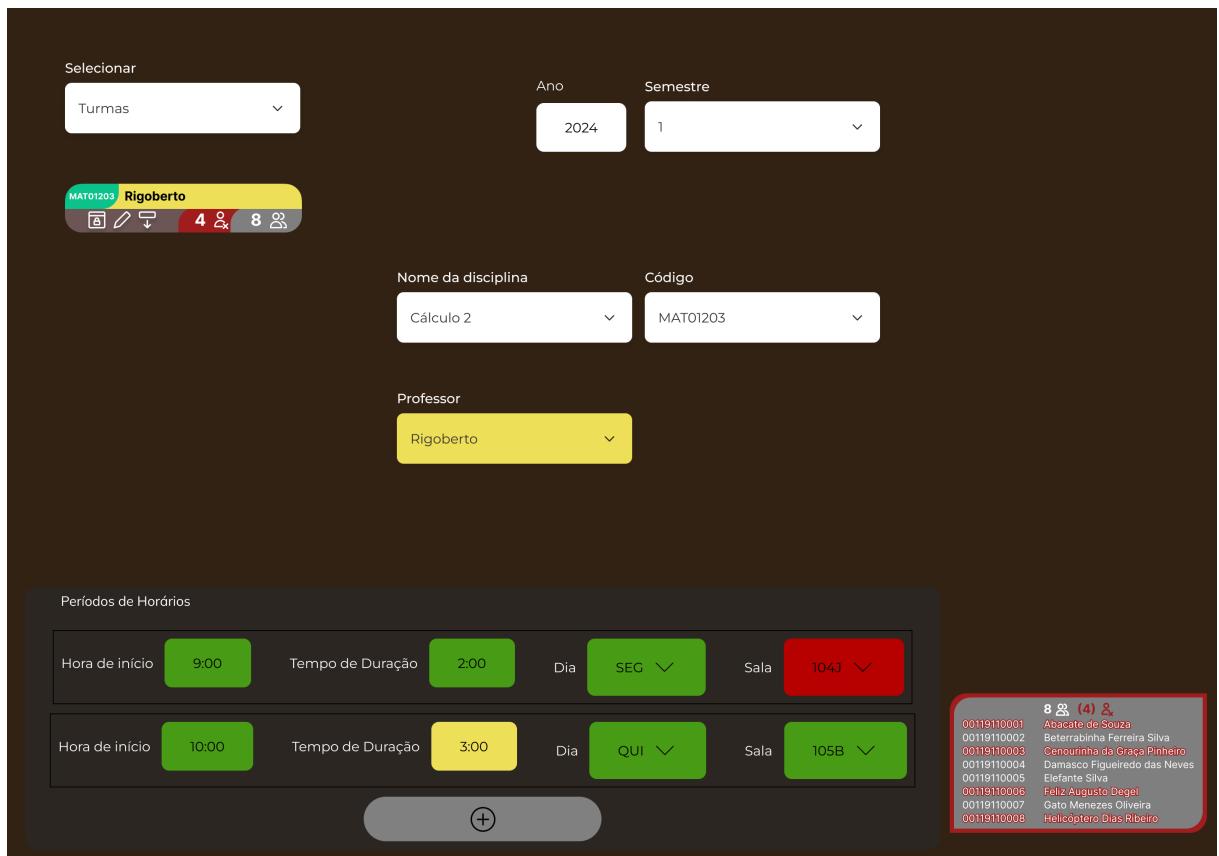
Outro conflito exibido é que a sala em que está alocado o primeiro horário da turma já está ocupada no mesmo horário por outra turma. Este conflito é impeditivo,

Figura 9 – Página de modificação das informações de professores



Fonte: autoria própria

Figura 10 – Página de modificação das informações de turmas



Fonte: autoria própria

sendo então representado na cor vermelha. Neste caso, a turma deve ser realocada para outro horário ou sala.

Por último, na listagem dos alunos que podem se inscrever, no canto inferior direito, há quatro alunos marcados em vermelho. Estes alunos poderiam se inscrever em outra turma no mesmo horário em que esta turma está alocada. Este conflito é impeditivo, mas apenas para os alunos, assim como no caso da preferência do professor, a turma pode ser realocada para outro horário para atender melhor às demandas dos alunos.

1.4 Programação do Sistema

Após a conceitualização diagramática do banco de dados e a elaboração dos protótipos com o Figma, o desenvolvimento do sistema foi iniciado. Por maior familiaridade com a linguagem foi escolhida a linguagem **JavaScript**, utilizando a biblioteca **React** para a criação dos componentes visuais, ou seja, o *frontend*, e o **Node.js** para a criação do *backend* e a criação de um servidor local que permite visualizar as mudanças no código em tempo real. Suas logos estão ilustrados na Figura 11.

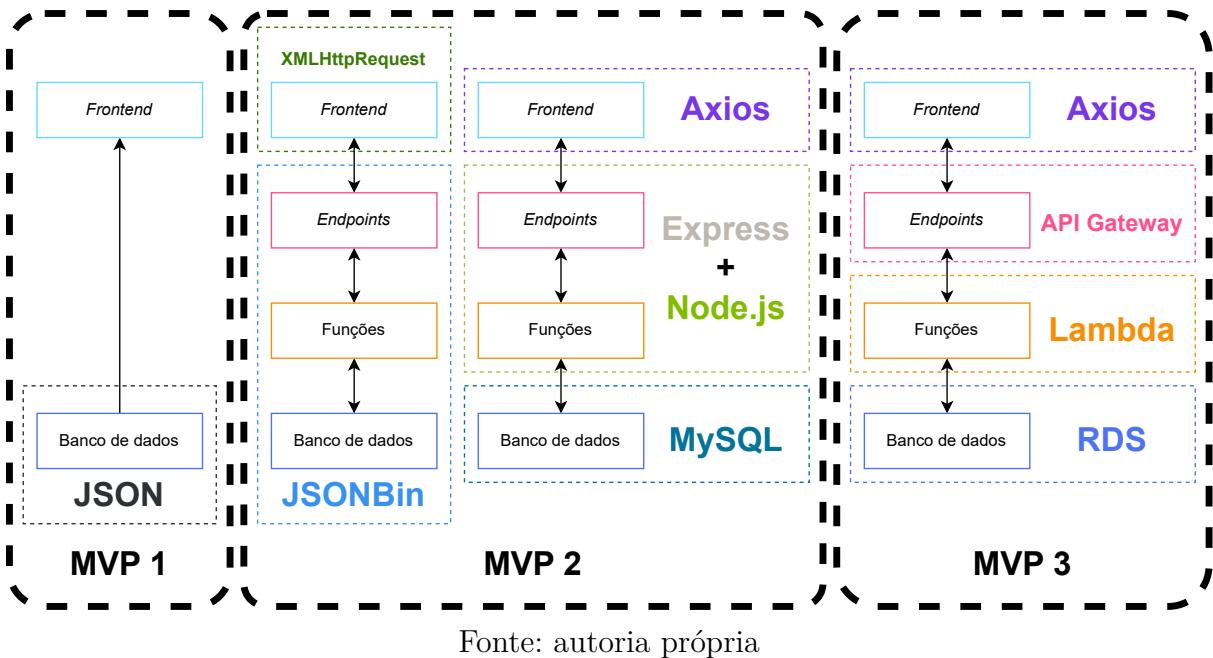
Figura 11 – Recursos usados para o desenvolvimento do sistema

Fonte: autoria própria

Seguindo constantemente o conceito de iteratividade da apresentação do sistema, a programação foi marcada por dois conceitos: blocos de funcionalidade marcas e blocos de funcionalidade apresentadas. O primeiro conceito se refere à grandes grupos de mudanças que estavam relacionadas a um mesmo conceito. O segundo conceito se refere à apresentação esporádica da situação atual do sistema para quem o iria utilizar. Nesta seção serão apresentadas as versões do sistema de acordo com a sua repartição em mudanças relacionadas e sequenciais.

Sua programação foi repartida em três grandes categorias que visavam entregar o sistema de forma gradual e funcional. A primeira versão do sistema foi desenvolvida localmente com o objetivo de se aproximar ao máximo das páginas previstas no protótipo, sem a necessidade de um banco de dados que permitisse alterações. A segunda versão do sistema contou com a utilização de duas abordagens distintas de bancos de dados para se obter a permanência dos dados. Já a terceira versão do sistema foi desenvolvida de forma a estar completamente hospedada na nuvem, incluindo o seu banco de dados, sendo então preenchido com mais dados.

Figura 12 – Diagrama da progressão funcionamento da permanência dos dados



Ao longo da implementação dessas versões, diversos métodos de manutenção dos dados foram utilizados, como a importação de arquivos JSON, a utilização de um banco de dados local e a utilização de um banco de dados hospedado na nuvem. A Figura 12 ilustra a progressão do funcionamento da permanência dos dados ao longo das versões do sistema. Os pormenores de cada versão serão descritos adiante.

1.4.1 Versão 1.0

A primeira versão do sistema foi desenvolvida em um ambiente local, com o objetivo de se aproximar ao máximo das páginas previstas no protótipo. Para isso, foi utilizada a biblioteca *React Router*⁸ para a navegação entre as páginas, e a biblioteca *React Select*⁹ para as caixas de seleção.

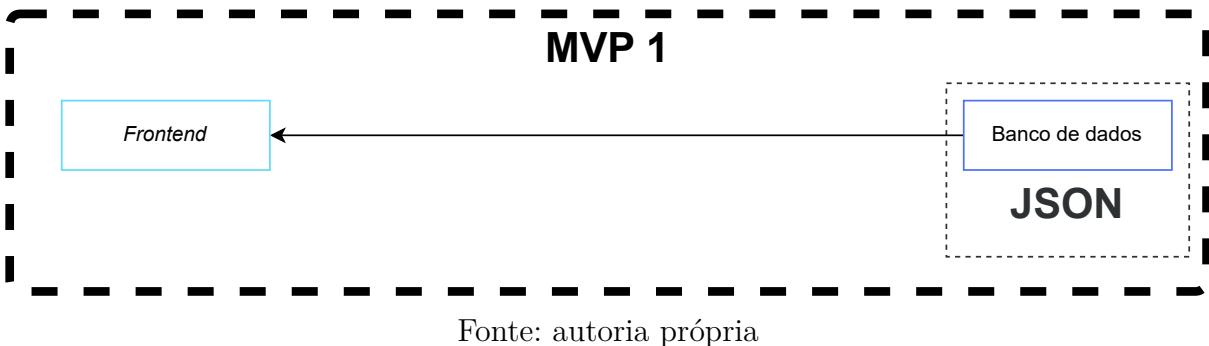
1.4.1.1 Banco de Dados Preliminar

Os dados contidos no sistema foram inicialmente armazenados em arquivos JSON, que eram importados diretamente para o código (Figura 13). Isso foi feito para que fosse possível visualizar o funcionamento do sistema sem a necessidade de um banco de dados real. A partir disso, foi possível visualizar o funcionamento do sistema e realizar testes de usabilidade. Em contrapartida, os dados disponíveis não eram modificáveis, tendo apenas a possibilidade de leitura e mutação temporária, visto que após recarregar ou mudar de página, as mudanças eram perdidas.

⁸ <<https://reactrouter.com>>

⁹ <<https://react-select.com>>

Figura 13 – Diagrama da progressão funcionamento da permanência dos dados



Fonte: autoria própria

Nesse método, cada entidade era armazenada em um arquivo JSON separado, contendo esse um array de objetos, onde em cada objeto haviam as chaves, representando as propriedades da entidade, e os valores, representando os dados da entidade.

Como nesta dinâmica não havia uma forte correlação entre os dados, o *frontend* acabava sendo o responsável por unir todas as informações. Assim, por exemplo, para se obter a lista de professores de uma turma, era necessário importar todos os professores, todas as turmas, e então, a partir do nome do professor alocado àquela turma, buscar na listagem dos professores qual era o professor que correspondia àquele nome, para então agrregar as informações.

1.4.1.2 Funcionalidades Iniciais: CRUD e primeiros conflitos

Nessa primeira versão, algumas funcionalidades já começaram a ser esboçadas, principalmente as funcionalidades CRUD (*Create, Read, Update, Delete*) para as entidades principais do sistema. Embora, como já dito, os dados não fossem persistentes, foi possível visualizar o funcionamento das funcionalidades de criação e leitura de turmas, professores, disciplinas, salas e horários.

Nessa versão, também foi implementada uma checagem bruta de conflitos por alocação simultânea de professores em mais de uma turma e a checagem da quantidade de demanda de alunos em relação à capacidade das salas. Uma descrição mais detalhada das funcionalidades de conflitos está presente adiante na seção 1.5 denominada Detecção e Alerta dos Conflitos.

Além dessas funcionalidades que se mantiveram até a conclusão do sistema, também foram desenvolvidas funcionalidades que não obtiveram o mesmo êxito e que foram deixadas de lado. Dentre elas, podemos citar a definição de níveis de preferência de horários para professores, a definição das características especiais das salas, e o andamento dos alunos em relação às disciplinas. Houveram também outras que nem chegaram a ser desenvolvidas, como a realocação de turmas através de um sistema de arrastar e soltar e o uso de heurísticas para a realocação de turmas.

1.4.1.3 GitHub Pages

Após o desenvolvimento local, como forma de viabilizar o acesso ao sistema por parte de outros usuários, foi feito o *deploy*, ou seja, foi feito o *upload* do sistema para um servidor online. Para isso, foi utilizado o serviço GitHub Pages que, por ser gratuito e de fácil utilização, foi a escolha mais adequada para o momento.

1.4.2 Versão 2.0

Utilizando do *feedback* quanto aos resultados entregues na primeira versão, alguns pontos de melhoria foram identificados, sendo um deles, e o mais importante: o planejamento. Na primeira abordagem, o desenvolvimento foi feito seguindo notas e ideias soltas, sem um planejamento prévio, o que resultou em um sistema que, embora funcional, não atendia a todas as necessidades propostas. Também dispunha de funcionalidades que não eram de todo necessárias, ou, melhor dizendo, que tinham menor prioridade do que muitas outras. Como solução, foi utilizado o GitHub Projects para organizar as tarefas e priorizá-las.

Nessa versão, também foram utilizados dois métodos de manutenção dos dados o JSONBin que não atingiu às expectativas e o MySQL que serviu para a criação de um banco de dados local, já emulando o posterior uso de um banco de dados hospedado na nuvem.

1.4.2.1 GitHub Projects

Utilizando o GitHub Projects, foi organizado uma tabela de tarefas, vista na Figura 14, onde foram unificadas as diversas anotações e ideias, antes soltas. A partir disso, foi possível visualizar o que era mais importante e o que poderia ser deixado de lado.

Tendo este novo sistema de tarefas em prática, foi possível planejar melhor quais eram as funcionalidades que precisavam ser desenvolvidas, as que já estavam prontas, as que poderiam ter melhorias e quais se desejava implementar no futuro.

As tarefas foram inicialmente divididas em três principais categorias: *Status*, *Pages* e *Sequence*. O *Status* reflete o andamento da tarefa, se ela está disponível, em andamento, ou concluída. O *Pages* reflete em qual página do sistema a tarefa se encontra, e o *Sequence* reflete a ordem de prioridade da tarefa.

1.4.2.2 Permanência na Alteração dos Dados

Tendo agora uma rota mais clara a ser seguida, o desenvolvimento foi retomado. Uma das características mais marcantes e ainda não atribuídas ao sistema era a manutenção das modificações feitas nos dados. Para exemplificar o funcionamento geral da permanência dos dados, consideremos o uso de uma *REST API*, representada pela Figura 15, utilizando

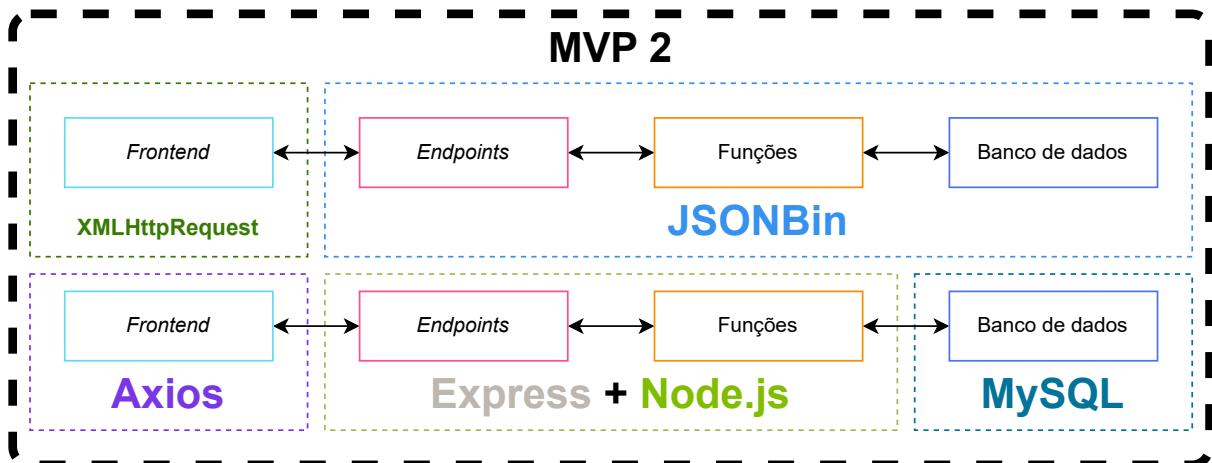
Figura 14 – Tabela de tarefas do GitHub Projects

Title	... Assign... Status ... Labels ... Milestone ... Page ... Sequence ... +
1 🎯 Aprimorar visualização de turmas #1	jlvd3 - Done - new Feature - MVP 1 - Classes - 1
2 🎯 Usar um banco de dados simples para armazenar as informações #2	jlvd3 - Done - new Feature - MVP 1 - Database - 20
3 🎯 Definir paletas de cores #3	- Todo - new Feature, quality of Life - Future - System -
4 🎯 Definir requisitos de características de salas das disciplinas #4	- Todo - new Feature - Future: conflicts - Subjects -
5 🎯 Usar Schema do Banco de Dados de Daniel Brito #5	jlvd3 - Done - abandoned, quality of Life - MVP 3 - Database -
6 🎯 Mostrar a listagem exata de disciplinas demandadas #6	- Todo - new Feature - Future: demand - Multiclasses -
7 🎯 Definir quais disciplinas cada aluno já fez. #7	- Todo - new Feature - Future: demand - Students -
8 🎯 Visualizar as disciplinas passíveis de quebra de requisitos #8	- Todo - new Feature - Future: demand - Students -
9 🎯 Ao adicionar disciplina em "feitas", "cursando" ou "não feitas", atualizar as outras #9	- Todo - new Feature - Future: demand - Students -
10 🎯 Cada item selecionável poderá ter seu próprio link #10	- Todo - new Feature, quality of Life - Future - System -
11 🎯 Cada item em cada página deve ter sua própria página única #11	- Todo - new Feature, quality of Life - Future - System -
12 🎯 Usar o Select Async para seleção de item #12	jlvd3 - Done - abandoned, quality of Life - Future - System -
13 🎯 Usar o JSON base como opções temporárias enquanto não carregam os dados async #13	jlvd3 - Done - quality of Life - MVP 1 - Database -
14 🎯 Reformular o Select para que o JSON que não esteja restrito a Value e Label #14	jlvd3 - Done - new Feature, quality of Life - MVP 1 - System -
15 🎯 Separar o arquivo options que são valores fixos #15	jlvd3 - Done - quality of Life - MVP 1 - System - 1
16 🎯 Ajustar roteamento do GitHub Pages #16	jlvd3 - Done - bug, quality of Life - MVP 1 - System -
17 🎯 Padronizar o visual do código #17	jlvd3 - Done - quality of Life - MVP 1 - System -
18 🎯 Visualizar a quantidade de horas que o professor está ministrando #18	- Todo - new Feature - Future: conflicts - Professors -

Fonte: autoria própria

de 4 “camadas”: o **frontend**, os **endpoints**, as **funções** de execução e o **banco de dados**.

Figura 15 – Diagrama da progressão funcionamento da permanência dos dados



Fonte: autoria própria

O **frontend** é a interface do sistema, onde o usuário interage com o sistema. Ele se encontra em duas formas: a primeira é o chamado “em produção”, que é o sistema que o usuário final acessa, e a segunda é o chamado “em desenvolvimento”, que é o sistema que o desenvolvedor acessa para realizar as modificações necessárias. Ambas precisam se comunicar com o **backend** para realizar as quatro operações básicas no banco

de dados (criação, leitura, atualização e deleção) por sobre as entidades existentes (turmas, professores, disciplinas, salas, etc.). Elas assim o fazer ao enviar requisições HTTP (*GET*, *POST*, *PUT* e *DELETE*), contendo pacotes de informações em formato JSON para os **endpoints**.

Os **endpoints** são as rotas que o *backend* disponibiliza para a recepção das requisições HTTP. Eles são responsáveis por encaminhar as requisições recebidas. Seu funcionamento consiste em rotear as requisições recebidas junto com sua carga útil. Para tanto, as rotas criadas refletem diretamente a qual entidade do banco de dados a requisição se refere, sendo então assim sabido qual **função** deve ser executada.

As **funções** são as responsáveis por executar as operações no banco de dados. Elas processam o pacote de informações recebido, e então realizam a operação desejada no **banco de dados**.

O **banco de dados** recebe a requisição, processa a operação, e então retorna o status da operação. Esse retorno é então repassado camada por camada, até chegar ao *frontend*, onde o usuário final pode visualizar o resultado da operação.

Resumidamente: O *frontend* envia uma requisição HTTP com uma carga de informações a um **endpoint**, que encaminha a requisição a uma **função** específica que executa uma operação no **banco de dados**, assim retornando o status da operação ao *frontend*.

JSONBin¹⁰

Como até então os dados estavam armazenados em formato JSON, imaginou-se que a melhor forma de persistir os dados seria através de um banco de dados que lidasse com JSON, e o escolhido foi o JSONBin.

Esta plataforma permite a criação de *bins*, que são basicamente coleções de dados em formato JSON. A partir disso, é possível realizar requisições HTTP para a leitura, escrita, atualização e remoção dos dados. A utilização do JSONBin foi feita através de requisições HTTP usando o objeto *XMLHttpRequest* do JavaScript, e a comunicação entre o *frontend* e o JSONBin foi feita através de *tokens* de acesso.

Com isso, se tornou possível ler e atualizar os dados de forma remota, e assim, manter os dados mesmo após a recarga da página. Embora cumprisse com o que promete e o que era desejado, o JSONBin não se mostrou a melhor escolha para o sistema, visto que a sua utilização não performou tão bem quanto se esperava. A utilização do JSONBin para a coleta dos dados, fazia com que a tela de carregamento do sistema demorasse alguns segundos para ser exibida, o que não é apropriado para a usabilidade do sistema proposto.

¹⁰ <<https://jsonbin.io>>

MySQL

Embora houvesse o desejo do uso de informações em formato JSON, achou-se por bem utilizar um banco de dados mais usual, recorrendo então ao MySQL, sendo então necessário criar um banco de dados local que armazenasse os dados e que pudesse ser acessado pelo sistema. Essa configuração serviu para estabelecer a supracitada camada de banco de dados. E consistiu basicamente na instalação do MySQL Server.

Migração dos dados

Como os dados se encontravam em formato JSON, primariamente utilizou-se da ferramenta de importação de dados do próprio **MySQL Workbench**. Durante essa importação, o software automaticamente identifica os campos, criando a tabela e suas colunas. Porém, devido à quantidade dos dados, essa importação tendia a ser demorada, e por vezes, falhava .

Com isso, foi necessário recorrer a uma abordagem semimanual, sendo então desenvolvido um código em Python que lê os arquivos JSON e os converte em arquivos SQL para que as *queries* pudessem ser executadas no MySQL Workbench. A partir disso, foi possível importar os dados de forma mais rápida e eficiente.

Apesar da primeira tentativa de importação não ter sido completamente bem sucedida, foi desta forma que as tabelas, representadas pela Figura 16, foram inicialmente criadas. Não seguindo objetivamente a modelagem anteriormente citada. Isso gerou posteriormente a necessidade de ajustes manuais, como a adição de chaves primárias e estrangeiras, e a alteração de tipos de dados. Porém, como neste momento, o sistema visava apenas replicar o funcionamento do JSONBin, essas alterações não foram feitas de imediato.

Acesso ao Banco de Dados

Seguindo a mesma sequência de camadas, o acesso ao banco de dados continua sendo feito através de requisições HTTP, porém, ao invés de serem enviadas ao JSONBin, são enviadas a um servidor local que executa as operações no banco de dados.

No *frontend*, enquanto que para acessar a API já pronta do JSONBin foi utilizado o objeto *XMLHttpRequest*, para a comunicação com o banco de dados local, foi utilizada a biblioteca *Axios* para construir as requisições HTTP. E elas, ao invés de serem enviadas ao JSONBin, são enviadas ao **servidor local**.

Na criação deste **servidor local** utilizou-se a biblioteca *Express* para desenvolver um *backend* local executado paralelo ao *frontend*. Essa biblioteca é responsável por criar todas as rotas necessárias para a comunicação entre o *frontend* e o banco de dados. A

Figura 16 – Diagrama inicial das tabelas de dados SQL

Disciplinas			Professores		
id	idDisciplina	PK	id	idProfessor	PK
int	periodoEsperado		string	apelidoProfessor	
string	apelidoDisciplina		string	curso	
string	codigoDisciplina		string	laboratorio	
string	nomeDisciplina		string	nomeProfessor	

Turmas			Salas		
id	idTurma	PK	id	idSala	PK
int	ano		int	capacidade	
int	demandEstimada		string	bloco	
int	semestre		string	blocoSala	
string	nomeDisciplina		string	codigoSala	
string	nomeProfessor		string	descricaoBloco	

Fonte: autoria própria

partir disso, foi possível criar rotas para cada uma das entidades, e para cada uma das operações CRUD. Com isso, cada operação CRUD em cada uma das rodas é encaminhada para uma **função específica** que executa a operação no banco de dados.

Este uso, embora exemplifique a aplicação da permanência dos dados, está limitado por dois aspectos: em primeira instância, a permanência dos dados é limitada ao servidor local, não sendo este o desejo final do sistema. Em segunda instância, para haver o acesso aos dados, é necessário que, além do banco de dados, o *backend* também esteja em execução, entretanto, o GitHub Pages, onde o sistema está hospedado, não viabiliza essa execução. Com isso viu-se necessária a busca por um novo serviço de hospedagem.

1.4.2.3 Logomarca

Um dia após o Natal, foi criado um dos arquivos do sistema chamado “hourclassMagic.js”, este arquivo agrupava funções que consistiam basicamente em adicionar e remover horários (*hour*) e turmas (*class*) da listagem de turmas e horários.

Porém, considerando que a junção dos nomes *hour* e *class* era consideravelmente similar à palavra *hourglass* (“ampulheta” em inglês) e que a ampulheta é um símbolo que remete ao tempo, e que o sistema é um sistema de alocação de horários, optou-se por utilizar a ampulheta como símbolo do sistema.

Quanto ao nome, como o sistema visa a alocação de turmas, especificamente para

o curso de Ciência da Computação na UENF, decidiu-se pela corruptela da palavra *hour* para que se tornasse *our*, que significa “nossa” em inglês, assim, remetendo ao sentido de individualidade do sistema. Com isso, o nome do sistema se tornou *OurClass*.

Figura 17 – Logomarcas do sistema



Fonte: autoria própria

Juntando o símbolo da ampulheta com o nome *OurClass* e outros elementos gráficos, como as tabelas horárias, foi criada a logomarca do sistema, que pode ser vista na Figura 17. A logomarca foi criada utilizando de inteligências artificiais generativas, principalmente o *Bing*. A partir disso, foi possível criar uma logomarca que refletisse o propósito do sistema, e que fosse agradável visualmente.

Após analisar as possibilidades, foi escolhida a Figura 18 como a logomarca oficial do sistema. A logomarca foi então adicionada ao sistema.

1.4.2.4 Funcionalidades Adicionais: conflitos, filtragem e disciplinas não oferecidas

Acrescendo à visualização de conflitos desenvolvida na primeira versão, foi implementada a visualização de conflito por capacidade de salas, ao comparar com a quantidade de alunos estimados para a turma. Mais detalhes sobre os conflitos podem ser vistos mais adiante na seção 1.5 denominada Detecção e Alerta dos Conflitos.

Adicionou-se também diversas filtragens, principalmente na página de **Grade Horária**. Dessa forma, torna-se possível a visualização específica de turmas que atendam a certos critérios. Essa filtragem é feita através de caixas de seleção, onde é possível selecionar quais critérios se deseja filtrar, sendo eles: ano, semestre, categoria, disciplina, professor e sala. Essa coletânea de filtros viabiliza uma análise mais limpa das informações estruturadas, podendo então gerar *insights* quanto ao posicionamento histórico das turmas.

Outra utilidade adicionada, agora na página **MultiTurmas**, foi a seção de “Disciplinas ainda não oferecidas”. Sua funcionalidade consiste em dispor ao usuário uma lista

Figura 18 – Logomarca oficial



Fonte: autoria própria

de disciplinas que, segundo a ementa de Ciência da Computação, deveriam ser ofertadas naquele semestre. A partir disso, o usuário pode então selecionar o botão correspondente àquela disciplina e, a partir disso, uma turma para esta disciplina é adicionada à lista de turmas ofertadas. Há também um botão no topo que permite a adição de todas as disciplinas de uma vez.

1.4.3 Versão 3.0

Considerando que a segunda versão já apresentava em sua maioria as funcionalidades mínimas desejadas, a terceira versão foi focada em melhorias de usabilidade e na correção de bugs.

1.4.3.1 Mudanças no GitHub Projects

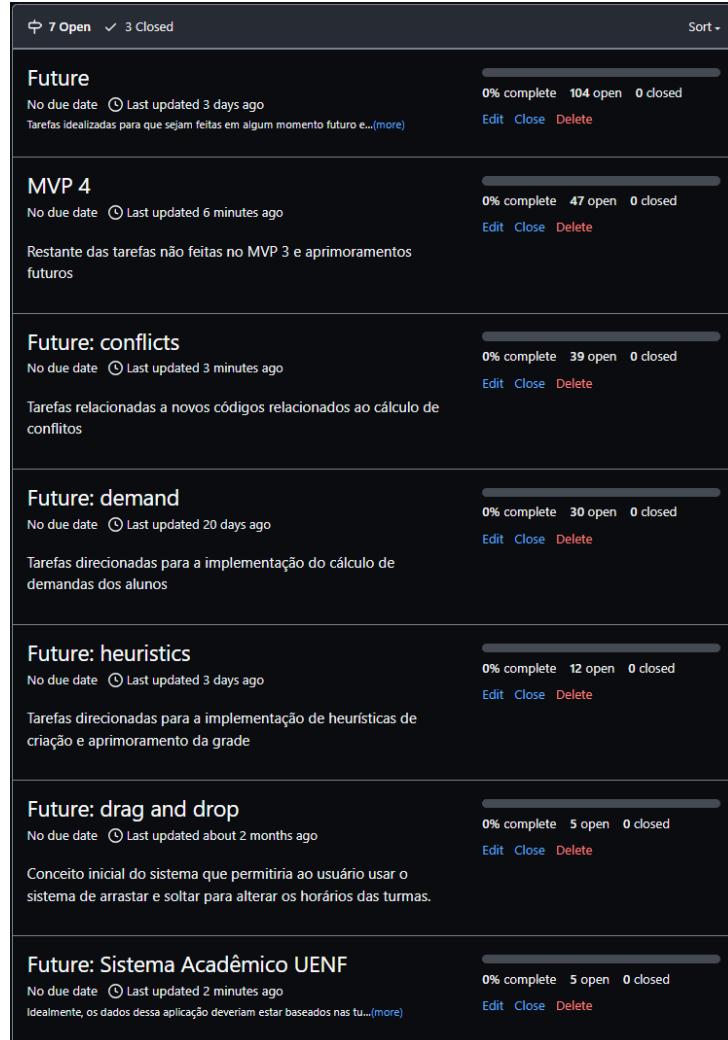
O uso do GitHub Projects se provou como uma excelente forma de organização das tarefas, e assim, foi mantido para a terceira versão. Alguns novos detalhes foram adicionados em sua organização, como a adição da categorização de tarefas por **Marcos** (*Milestones*, Figura 19) e **Etiquetas** (*Labels*, Figura 20).

Marcos e Etiquetas

Os **Marcos** visam distinguir as tarefas por suas versões, e também por seus tipos de funcionalidades futuras, assim, a medida em que surgiam novas ideias, elas eram

adicionadas ao GitHub Projects. Dessa forma, garantindo uma metrificação do andamento de cada categoria de funcionalidades, além de afunilar a quantidade de tarefas realmente prioritárias para o sistema.

Figura 19 – Marcos do GitHub Projects



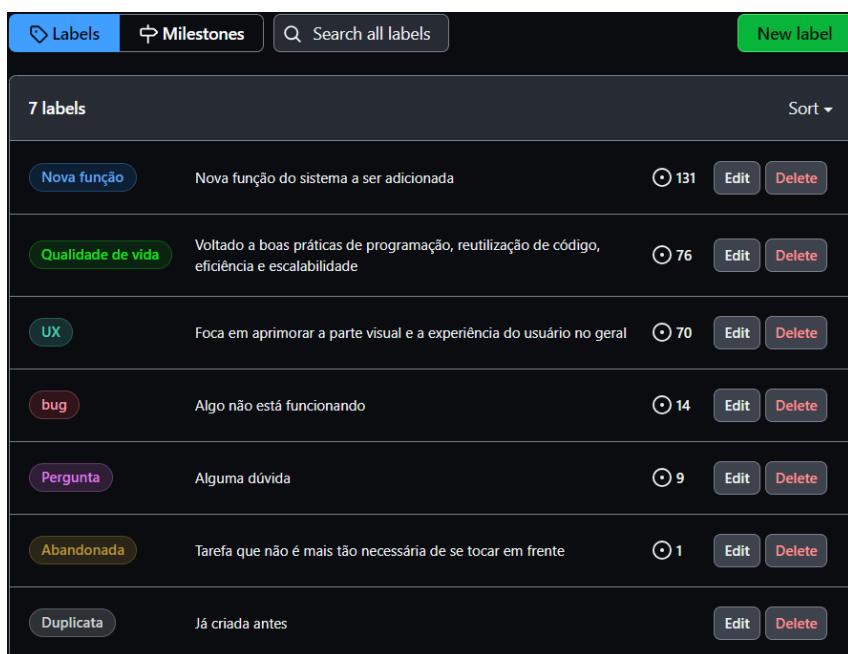
Fonte: autoria própria

1. **MVP 1:** foram concluídas na primeira versão;
2. **MVP 2:** foram concluídas na segunda versão;
3. **MVP 3:** foram concluídas na terceira versão;
4. **Futuro:** foram planejadas para o futuro do sistema;
 - a) **Demandas:** visam calcular a demanda dos alunos por disciplinas;
 - b) **Conflitos:** visam aprimorar a visualização, qualidade e/ou variedade de conflitos;

- c) **Heurísticas:** visam aprimorar a alocação de turmas através de heurísticas;
- d) **Arrasta e solta:** visam aprimorar a alocação de turmas através de um sistema de arrastar e soltar;
- e) **Integração com o Sistema Acadêmico:** visam a integração do atual sistema com o sistema acadêmico da UENF.

No Projects, também foram adicionadas as **etiquetas** que distinguem as tarefas por seu intuito.

Figura 20 – Etiquetas do GitHub Projects



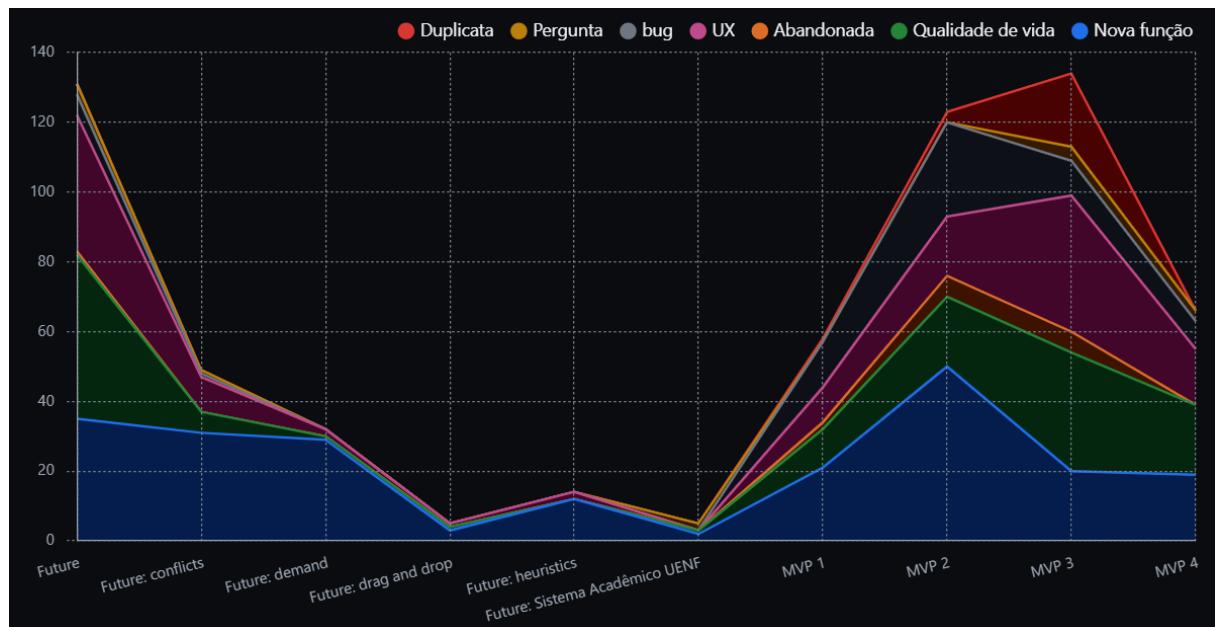
Fonte: autoria própria

1. **UX:** aprimoramento da experiência do usuário;
2. **Bug:** correção de bugs;
3. **Pergunta:** dúvidas sobre a validade da tarefa;
4. **Duplicada:** já foi criada antes e que foi descartada;
5. **Abandonada:** quando criada parecia interessante, mas que se decidiu por não implementar;
6. **Qualidade de vida:** melhorias que não são necessárias, mas que aprimoram o processo de desenvolvimento;
7. **Nova funcionalidade:** funcionalidades que ainda não foram implementadas;

Gráficos

O GitHub Projects também oferece a possibilidade de visualização de gráficos na seção *Insights*, que mostram a quantidade de tarefas em cada uma das categorias. Esses gráficos são úteis para a visualização do andamento do projeto, e para a identificação de possíveis gargalos.

Figura 21 – Gráfico de Marco *versus* quantidade de tarefas separadas por etiqueta



Fonte: autoria própria

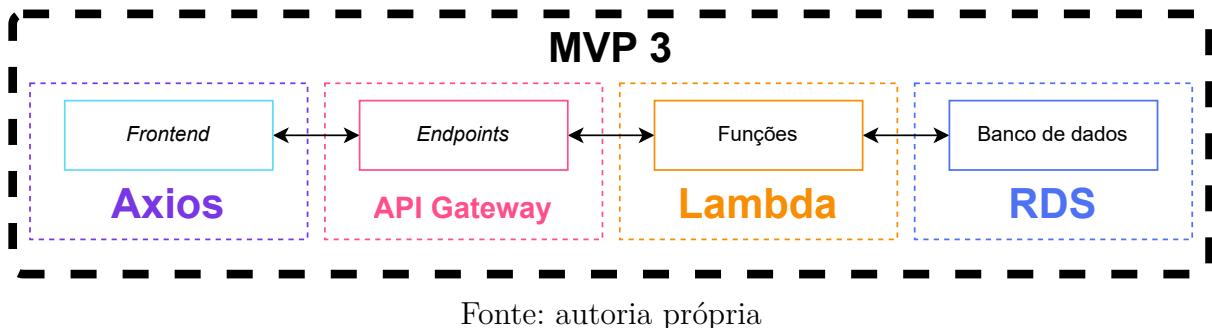
Com eles, pode-se ter uma noção das métricas do projeto, como por exemplo:

- Quantidade de tarefas de determinada etiqueta a cada marco, exemplificado na Figura 21;
- Quantidade de tarefas por marco;
- Quais páginas receberam tarefas de quais etiquetas;
- Quantas são as tarefas em cada um de seus estados (**completa**, **em progresso** ou **pendente**).

1.4.3.2 Amazon Web Services

Para suprir a necessidade de um servidor que pudesse executar o *backend* do sistema em conjunto com o banco de dados, foi escolhido a *Amazon Web Services* (AWS). A AWS é um serviço de computação em nuvem que oferece uma ampla gama de serviços, entretanto, apenas alguns deles foram necessários para o sistema.

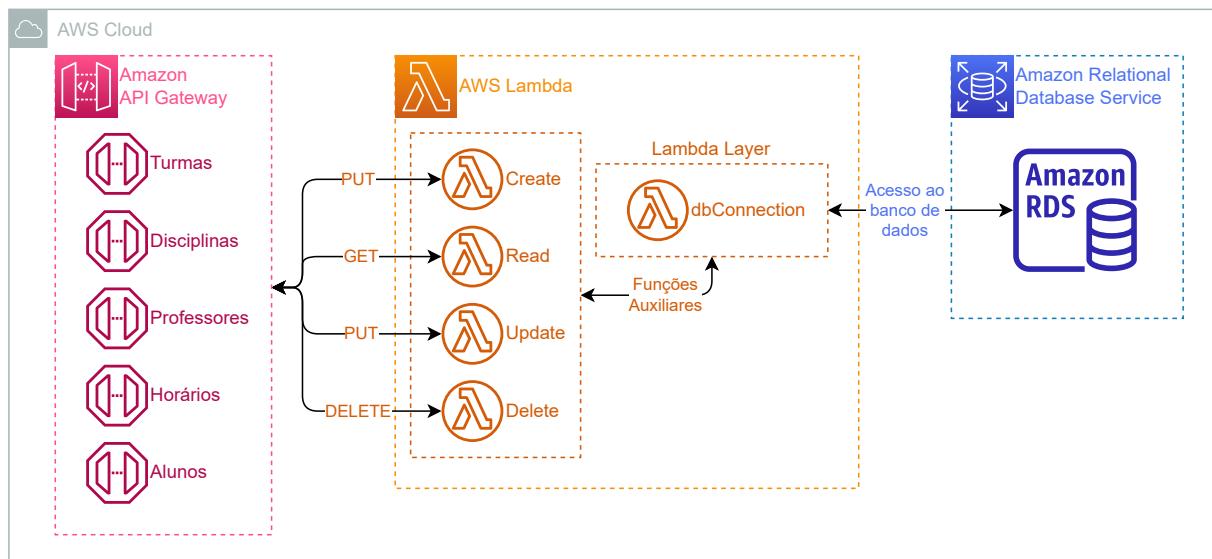
Figura 22 – Diagrama da progressão funcionamento da permanência dos dados



Fonte: autoria própria

O uso da AWS segue a mesma lógica do servidor local, com a diferença de que o servidor está em nuvem, e não localmente, assim resolvendo o primeiro dos dois problemas citados. Neste contexto o uso da AWS, representado pela Figura 23, foi feito através de três serviços principais: o *API Gateway* para a recepção das requisições HTTP, o *Lambda* para a execução das funções que acessam o banco de dados, e o *RDS* para o armazenamento dos dados; serviços estes que serão descritos mais detalhadamente a seguir.

Figura 23 – API REST no AWS



Fonte: autoria própria

O uso desses três serviços permitiu a execução do *backend* do sistema em nuvem, e assim, atingindo a permanência dos dados. Com isso, o sistema passou a ser capaz de manter os dados mesmo após a recarga da página, e assim, atender a uma das principais necessidades do sistema.

Implantação

O conjunto de funcionalidades da AWS envolve em grande parte o objetivo de manter um sistema constantemente acessível através da internet, ainda assim, durante o desenvolvimento, ou até mesmo durante o ciclo de vida do software, é esperado que ocorram manutenções periódicas nas quais é compreensível que o sistema fique fora do ar. Sendo assim, para manter-se visando ao máximo a acessibilidade do sistema, espera-se que o mesmo fique desconectado o mínimo possível.

Tanto o API Gateway quanto as funções Lambda precisaram sofrer diversas modificações ao longo do desenvolvimento. A aplicação dessas modificações é chamada de implantação (*deploy*), onde a AWS substitui a versão atual do sistema pela nova versão. Essa aplicação de modificações foi inicialmente feita através da interface *web* da AWS, porém, com o tempo, foi percebido que essa abordagem era ineficiente. No caso das implantações do API Gateway, visto que assim que as rotas estiverem configuradas, não há a necessidade de alterá-las, não havia grande impacto no fluxo de trabalho. Já no caso das funções Lambda, cada mínima mudança no código requisitava um novo *deploy* para cada uma das funções alteradas.

Outro detalhe percebido, foi que boa parte do código se repetia entre as funções que interagiam diretamente com o banco de dados. Sendo assim, passou-se a utilizar das *Lambda Layers*, que são camadas que podem ser compartilhadas entre diversas funções, e assim, diminuir a quantidade de código repetido. Essa nova abordagem permitiu que as funções fossem mais enxutas, e que as mudanças fossem aplicadas de forma mais rápida, visto que bibliotecas e funções comuns entre as funções eram compartilhadas entre elas.

AWS CLI

Nas primeiras tentativas de *deploy*, duas abordagens eram utilizadas: a primeira era a de copiar e colar o código diretamente na interface *web* da AWS, e a segunda era a de fazer o upload de um arquivo zip contendo o código.

Código 1.2 – Código de *deploy* de Lambda

```
1 aws lambda create-function \
2 --function-name createProfessor \
3 --runtime nodejs20.x \
4 --role arn:aws:iam::375423677214:role/LambdaRole \
5 --handler index.handler \
6 --zip-file file:///Files/AWS/lambdas/createProfessor/createProfessor.zip
```

Como mostrado no Código 1.2, usa-se o software AWS CLI para criar uma função lambda. O comando *create-function* é o comando que cria a função, e os argumentos que seguem são os parâmetros necessários para a criação da função. O *function-name* é o nome

da função, o *runtime* é a versão do Node.js que a função utiliza, o *role* é o conjunto de permissões criadas na seção *AWS Identity and Access Management* (IAM), o *handler* é o nome do arquivo principal que contém a função, e o *zip-file* é o arquivo zip que contém o código da função.

Com isso, ao executar o comando, a função é criada, e então, a nova versão do código é aplicada. Com este fluxo de trabalho, embora permita o *deploy* sem a direta conexão ao sistema AWS, ainda assim é necessário executar comandos específicos para cada uma, tendo que manualmente compactar o código e fazer o upload de cada um dos arquivos das diversas funções coexistentes.

SAM

Como solução, foi utilizado o AWS SAM (*Serverless Application Model*), que é uma extensão do *AWS CloudFormation* que simplifica o desenvolvimento de aplicações sem servidor. O AWS SAM permite a definição de aplicações sem servidor de forma mais simples, e a partir disso, é possível fazer o *deploy* de toda a aplicação de uma vez só. O uso do AWS SAM foi feito através de um arquivo *template.yaml*, que contém a definição de todas as funções Lambda, e de recursos necessários para o funcionamento do sistema.

Como mostrado no ?? disposto no ??, o arquivo *template.yaml* contém a definição de algumas das estruturas utilizadas para este sistema, principalmente as funções Lambda, visto que são quatro funções para cada uma das seis entidades, totalizando 24 funções. O arquivo contém a definição de cada uma das funções, e de cada uma das rotas que elas atendem. A partir disso, é possível fazer o *deploy* de todas as funções que foram alteradas de uma vez só, e assim, diminuir o tempo de *deploy* e a quantidade de comandos necessários.

Após o preparativo do arquivo *template.yaml*, o *deploy* é feito através do conjunto de comandos *sam build; sam deploy*, que primeiro combina o *CloudFormation Template* com o código da aplicação, e em seguida realiza o *deploy* de todas as funções definidas no arquivo. Com isso, o sistema passou a ser mais facilmente atualizável, e mais facilmente mantido.

1.4.3.3 Melhorias no Sistema

O sistema passou por diversas pequenas mudanças, e algumas maiores. Uma considerável parte delas foi relacionada à forma com que as informações eram estruturadas internamente, mudanças essas feitas com o intuito de tornar o sistema mais fácil de ser mantido posteriormente. Em seguida estão listadas algumas das várias melhorias feitas no sistema.

Filtros e ordenações

Levando em consideração a multidimensionalidade da estrutura dos dados, a possibilidade de realizar “curvas de nível” e as ordenar por diferentes critérios se mostrou uma funcionalidade essencial para a compreensão dos dados. Dessa forma, em diferentes páginas do sistema, foram adicionadas ordenações padrões e seletores de filtragem manual, assim permitindo ao usuário a visualização de dados específicos.

Alguns exemplos de casos de uso dessas funcionalidades seria: “Na página **Multi-Turmas** o Professor A está tendo conflitos em suas turmas, então o usuário pode filtrar as turmas do Professor A e visualizar apenas as suas turmas. Em seguida, o usuário percebe que o Professor A está em conflito com a Sala B123, então o usuário visualiza apenas as turmas da Sala B123. Por fim, o usuário encontra um outro horário disponível para o Professor A e a Sala B123, e então, o conflito é resolvido.”

Uma das solicitações presentes ao final da versão anterior foi quanto a uma distinção mais clara entre disciplinas de Ciência da Computação e disciplinas de outros cursos. Para isso, embora não pareça apresentar grande robustez na forma como foi feito, apresenta suficiente clareza para o usuário final. A distinção foi feita ao utilizar o campo **Período Esperado** presente na entidade **Disciplina**, para definir que:

- **1 ≤ Período Esperado ≤ 10:** Disciplina obrigatória de Ciência da Computação;
- **Período Esperado = 11:** Disciplina eletiva optativa para Ciência da Computação;
- **Período Esperado = 12:** Disciplina eletiva livre para Ciência da Computação;
- **Período Esperado = 13:** Disciplina não ofertada para Ciência da Computação;

Com essa divisão, definiu-se o sistema para que, por padrão, apenas exibisse as turmas voltadas para Ciência da Computação, e que, caso o usuário desejasse, poderia visualizar as turmas de outros cursos.

MultiTurmas

Dentre as tarefas realizadas, uma das páginas que mais sofreu alterações foi a página de **MultiTurmas**. Nela, foram feitas diversas melhorias, como a adição de filtros, ordenações e aprimoramento dos textos contidos nas caixas de seleção.

Adição da propriedade **descrição**

Disciplinas oferecidas em um mesmo semestre por vezes são oferecidas para alunos demais para que uma única turma os comporte, e assim, é necessário a criação de mais de

uma turma. Para que os alunos e professores possam identificar facilmente a qual turma pertencem, foi adicionada a propriedade descrição, que é uma breve descrição da turma. Esse código descritor já se encontra no Sistema Acadêmico, porém com a limitação de apenas 3 caracteres. No presente trabalho, a descrição pode conter até 255 caracteres.

Adicionando esse campo, a visualização linear das informações da turma se tornou mais difícil de apresentar na tela. Considerando que em sua maioria as turmas possuem dois horários, dispõe-se então as informações em conjuntos de dois elementos, ao invés de uma lista única, tornando então a visualização mais densa.

Aprimoramento dos identificadores dos conflitos

Os conflitos ocorridos indicavam quais eram os identificadores (ids) das turmas que estavam em conflito, porém, esses ids eram referentes ao banco de dados, sendo ele um valor numérico, não continha valor semântico suficiente para ser facilmente identificado. Estes ids eram visualizados ao posicionar o ponteiro do mouse por sobre os componentes cujo conflito foi verificado. Com isso, foi feita a adição de um novo identificador, que é composto pelas informações contidas na turma, sendo elas o ano, semestre, nome da disciplina, nome do professor, e o código descritor da turma. A partir disso, tornou-se mais fácil identificar quais turmas estavam em conflito, e assim, corrigi-las. Essa identificação foi adicionada também aos horários, onde o identificador passou a ser composto pela sala, dia da semana, horário de início e fim.

Criação e deleção de turmas e horários

Embora seja uma funcionalidade básica e existente desde a primeira versão, a criação e deleção de turmas apresentou diversos problemas ao longo do desenvolvimento. Um dos mais cruciais era devido à assincronicidade intrínseca ao uso de um banco de dados remoto. O problema era que durante a criação sequencial de duas turmas, apenas a segunda era mostrada, mesmo que ambas tivessem sido criadas.

O que ocorria era que, ao começar com a lista de $Turmas = [A, B]$ tenta-se adicionar a turma C à lista de $Turmas$, mas para isso, a requisição enviada ao banco de dados deve retornar com o status de sucesso, e para que, só assim, fosse adicionada à listagem apresentada no sistema. Então, caso fosse feita a tentativa de se adicionar a turma D antes da confirmação anterior ser recebida, a adição seria realizada novamente na listagem inicial ($[A, B]$). Por fim, assim que a primeira requisição retornasse bem sucedida, por um breve instante a listagem seria $[A, B, C]$, e então, após a adição da turma D , a listagem seria $[A, B, D]$.

Para resolver esse problema, foi feita a adição de uma função de *callback* que passou a utilizar o estado mais atual da listagem de turmas, e não mais a listagem inicial.

Outra característica aprimorada, foi a velocidade de adição e deleção, principalmente a de deleção. Antes, a aprovação do banco de dados era necessária para que a lista de turmas fosse atualizada, e isso tornava o processo de deleção lento. Para resolver isso, foi feita a adição de uma função de deleção que remove a turma da listagem de turmas antes mesmo da confirmação do banco de dados. Essa não se mostra como a solução mais adequada, visto que em caso de falha na deleção, a turma poderá ser restaurada com a simples atualização da página, os pontos positivos na usabilidade superam os negativos.

Solução inicial

A funcionalidade anteriormente denominada “Disciplinas ainda não oferecidas” foi aprimorada de tal forma que agora, além de criar uma turma para a disciplina selecionada, o sistema automaticamente analisa o histórico de criação de turmas, definindo previamente o professor, a demanda estimada, e os horários, incluindo seus dias, horas de início e sala em que é alocada.

O cálculo da demanda estimada é dado pela média de demandas de todas as turmas anteriores que possuem demanda estimada.

O botão de adição de todas as disciplinas foi mantido, e agora, ao ser clicado, cria as turmas com suas características já preenchidas, não considerando, entretanto, a descrição, visto que esta é uma característica única de cada turma e deve ser adicionada manualmente em caso de necessidade. Então, após a criação de todas as turmas referentes ao curso de Ciência da Computação, uma solução inicial foi obtida.

É esperado, entretanto, que esta solução apresente problemas em sua execução, visto que nem todas as alocações de turmas apresentam um padrão. Então, fica a cargo do usuário a verificação e a correção dos possíveis erros alertados pelo sistema.

Grade Horária

A página que detinha o nome “CCTable” e que visava apresentar exclusivamente as disciplinas do curso de Ciência da Computação, foi renomeada para “Grade Horária”, e passou a ser possível de apresentar disciplinas de todos os cursos, embora ainda não seja possível distinguir as disciplinas de um curso para o outro.

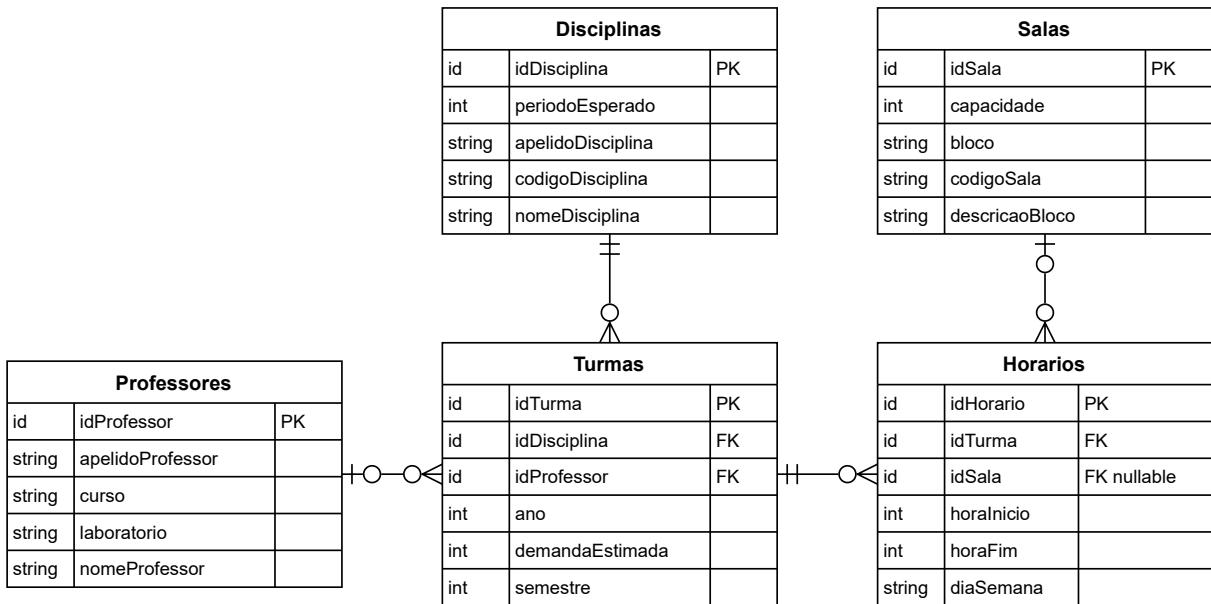
As células das turmas sofreram um ligeiro aprimoramento visual e foram ordenadas primariamente por seu período esperado.

Banco de dados

Quanto ao banco de dados, os dados antes desconexos passaram a ser interligados adequadamente por chaves estrangeiras. Apresentando então restrições em casos de deleções

inapropriadas no banco de dados. A API, por sua vez, deixou de retornar as listas de turmas, professores, disciplinas e salas, e passou a retornar os dados de forma mais estruturada em formato JSON.

Figura 24 – Novo Diagrama de Banco de Dados



Fonte: autoria própria

Na Figura 24 vemos as inter-relação entre as propriedades das entidades do banco de dados. Com o uso mais apropriado das chaves estrangeiras, o anterior uso dos nomes das disciplinas e professores como elementos de identificação foi substituído pelo uso de seus respectivos códigos identificadores. Houve também o surgimento da entidade **Horarios**, que interliga as turmas às salas.

Gerais

Neste tópico estão descritas algumas melhorias gerais do sistema que não se enquadram nos tópicos anteriores.

Aprimoramento na forma de criação de itens

Antes, ao acessar a página de criação de entidades, uma entidade era previamente selecionada. E para a criação de uma nova entidade, os valores da entidade anterior deveriam ser alterados, e, ao clicar em adicionar, esses valores alterados eram cadastrados no banco de dados. Essa sequência de ações apresentou intuitividade suficiente e, portanto, foi alterada.

A versão atual passou a não selecionar previamente a entidade. Assim, ao clicar em adicionar um novo item, uma nova entidade é criada. No caso das turmas, a entidade é criada com os valores de ano e semestre predefinidos baseado na filtragem selecionada.

Boas práticas

Além das funcionalidades voltadas para o usuário final, algumas mudanças classificadas como **Qualidade de vida** foram realizadas visando a manutenção do sistema. Dentre elas estão:

- **Repadronização de componentes:** como o conhecimento relacionado a boas práticas de programação foi adquirido ao longo do desenvolvimento, partes de componentes criados anteriormente foram reformulados para que estivessem estruturados de acordo com a estrutura recentemente desenvolvida no código.
- **Externalização de informações:** algumas informações constantes como cores de fundo e textos foram externalizadas para variáveis, assim, caso haja a necessidade de alteração, não será necessário a busca manual por todas as ocorrências. Um outro caso desses é referente aos textos dispostos nas caixas de seleção, que foram convertidos em funções que retornam o texto desejado.
- **Inglês:** como as linguagens de programação de modo geral se apresentam no idioma inglês, é considerado uma boa prática que as variáveis e funções também, o que não foi a abordagem inicialmente tomada. Assim, foi feita uma gradual migração para o inglês, e embora não tenha sido concluída, a maior parte do código já se encontra em inglês. Para facilitar essa migração, foi elaborado um sistema de “*getters*” como forma de obter a propriedade de determinado objeto, independente de qual língua ele esteja.
- **Remoção de estruturas obsoletas:** ao longo do desenvolvimento, algumas estruturas foram criadas e não utilizadas. Um exemplo dessas foi a propriedade “Ordem” que visava definir a ordem em que os horários da turma apareceriam, porém o resultado desta propriedade foi obtido com a ordenação dos horários por dia e em seguida por horário, não sendo então necessária. Ela então foi removida do sistema e do banco de dados.

1.5 Detecção e Alerta dos Conflitos

Uma das principais funcionalidades do sistema é a detecção de conflitos. Seu objetivo é auxiliar ao usuário a identificar possíveis problemas na alocação das turmas, e assim, permitir que ele possa corrigi-los antes de finalizar a grade horária. Diversas

situações podem ser consideradas como conflitos, e cada uma delas é tratada de forma diferente.

Os conflitos aqui se colocam como uma forma de alerta ao usuário, e não como uma restrição, assim viabilizando ao usuário que uma ação seja tomada, ou não, a partir do alerta. O conceito da não restrição é importante, visto que embora idealmente espera-se que o processo de alocação disponha de todas as informações para que seja otimamente alocado, na prática, isso atualmente não se mostra uma realidade.

Essa flexibilização das restrições que poderiam ser tidas como rígidas em um problema de otimização, é uma característica do problema de alocação de turmas da UENF. Como na realidade da instituição as grades precisam ser criadas enquanto ainda se tem informações incompletas, certas decisões precisam ser tomadas manualmente, sendo então necessária esta flexibilidade para permitir que o usuário possa tomar essas decisões.

Além disso, diversos casos atípicos acabam por ocorrer na realidade da universidade, e que, embora possam não ser aconselháveis ou até mesmo tidos como conflituosos pelo sistema, não seriam de fato um problema para a execução prática das alocações.

1.5.1 Típicos Conflitos Atípicos

Para ilustração, abaixo estão descritos alguns exemplos de conflitos que poderiam ser alertadas pelo sistema, mas que não seriam realmente um restritor para a execução prática das alocações:

Considerando o corpo docente do curso de Ciência da Computação, que atualmente conta com seis professores doutores, é recorrente a solicitação de professores bolsistas para ministrar disciplinas. Devido aos prazos existentes ao longo do processo de criação da grade horária, é comum que ainda não se saibam quais e quantos professores bolsistas serão disponibilizados para quais turmas. Porém, como o Sistema Acadêmico requere a inserção de professores para a criação de turmas, uma solução encontrada foi a inserção de um desses professores permanentes como responsável pela turma. E, mesmo após se obter a informação quanto a quais e quantos bolsistas estarão disponíveis, ainda assim o sistema acadêmico não os permite serem inseridos, visto que eles não têm um vínculo permanente com a instituição. Com isso, seria possível ver, por exemplo, um conflito entre duas turmas que possuem o mesmo professor em um mesmo horário, mas que na prática, uma delas será ministrada por um professor bolsista.

Outras situações que podem ocorrer giram em torno da alocação das salas. Duas situações que podem ilustrar sua atipicidade são: a possibilidade de alocar uma turma a uma determinada sala, mesmo que se tenha a intenção de ministrá-la em outra, e também a possibilidade de se repartir a turma em duas salas de aula ocorrendo simultaneamente.

Esses e outros são exemplos de situações recorrentes ao longo do processo flexível

da organização da tabela horária.

1.5.2 Conflitos Tratados pelo Sistema

Para a implementação, primeiro visou-se a detecção de conflitos que poderiam ser considerados restritores para a alocação das turmas. Sendo eles os de alocação simultânea de salas e professores, visto que um professor não pode ministrar duas turmas simultaneamente, nem uma sala deve comportar duas turmas simultaneamente (embora ambos sejam teoricamente possíveis).

Além disso, também foi implementada a detecção de conflitos de capacidade, onde a quantidade de alunos de uma turma é maior do que a capacidade da sala alocada, e alguns outros indicativos visuais que serão descritos abaixo.

Os conflitos calculados são representados de três formas diferentes. A primeira e mais perceptível é a mudança de cor de fundo das propriedades conflituosas. A segunda, visando evitar sobreposição de conflitos, é a adição de uma borda inferior que se estende por toda a largura da propriedade. E a terceira, mais descriptiva, é o uso do atributo *title* dos elementos HTML, que exibe uma mensagem de alerta flutuante ao passar o mouse sobre a propriedade conflituosa, assim dispondo de mais detalhes sobre os conflitos buscados e encontrados.

Embora o sistema seja projetado para ser permissivo quanto a inexistência de certas informações, é sempre esperado que a maior quantidade de informações possíveis seja inserida, assim, caso algum campo não tenha sido preenchido a cor de fundo do elemento será alterada para um tom acinzentado.

Figura 25 – Paleta de cores do sistema

Período da disciplina	Semestre			Entidade	Conflito		
	Verão	Correto	Errado		Nulo	Sem	Com
1	#FFFF00	#00FF00	#FF0000	Disciplina	#E0E0E0		
2	#EFEF00	#00EF00	#EF0000	Professor	#D0D0D0	#47D902	#C72508
3	#DFDF00	#00DF00	#DF0000	Demandá	#C0C0C0	#3EC200	#DD3333
4	#CFCF00	#00CF00	#CF0000	Sala	#B0B0B0	#38B000	#E3580E
5	#BFBF00	#00BF00	#BF0000	Dia	#A0A0A0	#008000	
6	#AFAF00	#00AF00	#AF0000	Hora	#909090	#007200	
7	#9F9F00	#009F00	#9F0000	Duração	#808080	#006400	
8	#8F8F00	#008F00	#8F0000				
9	#7F7F00	#007F00	#7F0000				
10	#6F6F00	#006F00	#6F0000				

Categoria	Disciplina
Não Computação	#D66615
Eletiva Optativa	#0055FF
Eletiva Livre	#4800FF
Sem Categoria	#EEC0C0

Fonte: autoria própria

Os conflitos que são representados por cores, têm sua paleta de cores representada

na Figura 25. Nessa paleta, dispõe-se 3 conjuntos principais: a distribuição de cores para as disciplinas obrigatórias do curso de Ciência da Computação, que variam de acordo com o semestre em que são ofertadas; a categoria das disciplinas não obrigatórias para o curso de Ciência da Computação; e os conflitos das outras entidades, que são classificados amplamente entre “com conflito”, “sem conflito” e “conflito nulo”.

Cinco campos se encontram sem cores, sendo eles a **Disciplina sem conflito**, **Disciplina com conflito**, **Dia com conflito**, **Hora com conflito** e **Duração com conflito**. No caso dos dois primeiros, melhor explicado a seguir, a representação dos conflitos é substituída pela representação de seu período esperado e de suas categorias. Já os três últimos também não possuem conflitos por si só, o que ocorre é que herdam a cor de fundo das entidades que têm conflito em determinado dia, hora e duração, como é o caso da alocação múltipla de turmas em uma mesma sala ou de professores em turmas simultâneas.

Figura 26 – Exemplo de conflito nulo



Fonte: autoria própria

A Figura 26 ilustra os **conflitos nulos**. Esse tipo de conflito representa a incompletude de informações, e como durante parte do processo de criação da grade não se tem todas as informações sobre as alocações das turmas, ele acaba por ser um dos conflitos mais comuns. Esse conflito, representado pela cor acinzentada, é detectado quando um dos campos não se encontra preenchido.

1.5.2.1 Conflitos de Professores

O sistema contempla a checagem de conflitos de alocação simultânea de professores em mais de uma turma. Ou seja, considerando todas as turmas ao qual o professor está atribuído no ano e semestre selecionados, o sistema compara todos os horários das turmas deste professor, e verifica se há alguma interseção entre horários que estão no mesmo dia, levando em conta a duração da aula.

Caso haja algum conflito, o sistema destaca o professor em questão, tornando a sua cor de fundo avermelhada. Além disso, ao passar o mouse sobre o nome do professor, é exibido um alerta flutuante, informando que quais são as turmas e horários que estão em conflito. Esse comportamento é exemplificado na Figura 27, onde o professor Tang é alocado em duas turmas que ocorrem simultaneamente durante algum intervalo de tempo

Figura 27 – Exemplo de conflito de alocação de professor



Fonte: autoria própria

durante os horários de quarta e sexta-feira, assim informando no alerta flutuante quais são as turmas e horários que estão alocados simultaneamente.

1.5.2.2 Conflitos de Salas

As salas também apresentam a verificação do conflito de alocação simultânea. Porém, diferente dos professores, a checagem é feita conferindo todos os horários na qual a sala está alocada, e então é feita a mesma verificação de interseção citada anteriormente. Havendo o conflito, é exibida uma borda alaranjada na parte inferior das propriedades referentes ao conflito, além de, assim como no caso dos professores, exibir o alerta flutuante. A Figura 28 representa um caso de conflito de alocação de salas.

Além disso, também é feita a comparação entre a quantidade máxima de alunos comportados na sala e a quantidade de alunos estimados para a turma. Este conflito por sua vez é ilustrado tornando avermelhado o fundo da demanda estimada e da seleção de salas. Caso uma turma tenha mais de um horário, é calculada a quantidade remanescente dos alunos que demandam a disciplina com relação a cada uma das capacidades das salas destes horários, mostrando cada um deles no alerta flutuante.

Então, como pode-se perceber na Figura 29, a turma ilustrada apresenta demanda estimada de 31 alunos não poderia ser adequadamente alocada às salas “P5 - inf1”, nem na “P3 - Bcct”, visto que a primeira apenas comporta 24 alunos e a segunda, 30 alunos. O alerta flutuante, ao ser acionado, informa ainda quantos são os alunos que não poderiam ser alocados em cada uma das salas.

1.5.2.3 Conflitos de Disciplina

Além desses conflitos, outras características analisadas e representadas se referem às disciplinas atribuídas às turmas, que, embora não representem necessariamente um

Figura 28 – Exemplo de conflito de alocação de sala

	Sala	Dia	Hora de início	Duração
	(24) P5 - inf1	Segunda	10	1 hora
	(24) P5 - inf1	Quarta	10	2 horas
	(24) P5 - inf1	Segunda	11	2 horas
	(24) P5 - inf1	Quarta	11	2 horas
	(24) P5 - inf1	Quarta	10	2 horas
	(24) P5 - inf1	Sexta	10	2 horas

Conflitos de alocação de sala avaliados:

- Sem conflitos de sala não definida
- Conflito: Alocação simultânea de Salas
 - Sala sobreposta com 2 turmas
 - Turma: (2025.1, Prog 1, Annabell); horários: [(P5-inf1, QUA, 10~12)]
 - Turma: (2025.1, ED1, Tang); horários: [(P5-inf1, QUA, 10~12)]
- Sem conflitos de demanda de sala

Fonte: autoria própria

Figura 29 – Exemplo de conflito de capacidade na sala

Demand Estimada	Sala
31	(24) P5 - inf1
	(30) P3 - Bcct
	(65) P3 - 106

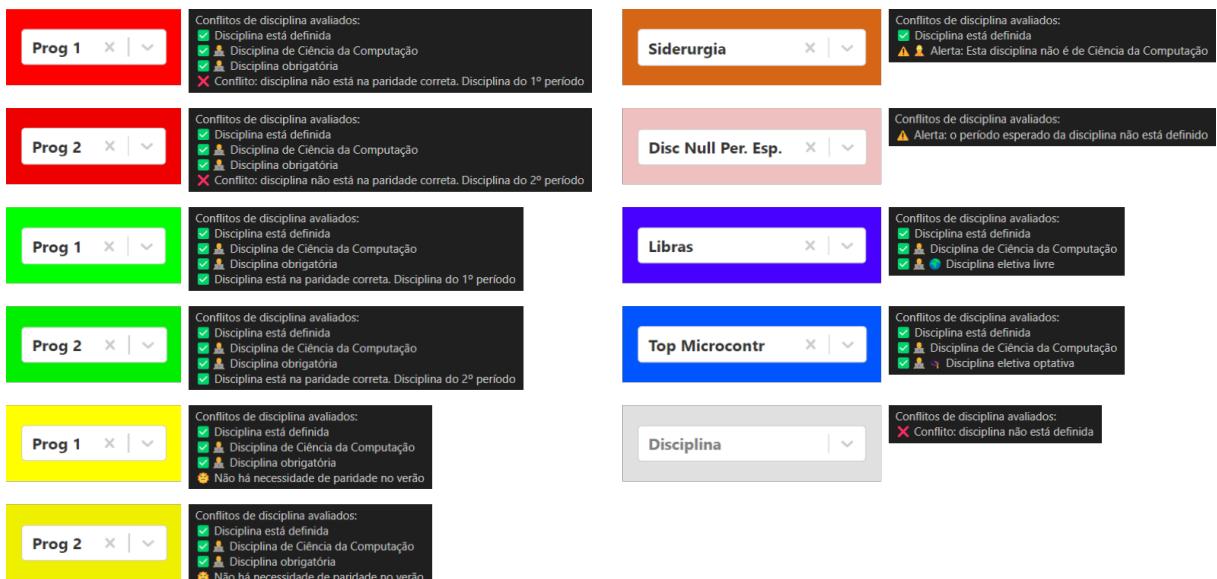
Conflitos de demanda avaliados:

- Sem conflitos de demanda não definida
- Conflito: há sala que não comporta a demanda
 - Há 2 conflitos de demanda.
 - No horário(P5-inf1) sobraram 7 alunos
 - No horário(P3-Bcct) sobraram 1 alunos

Fonte: autoria própria

conflito, mas sim um indicativo, ainda assim serão tratados como conflitos por motivos de simplificação. Esse indicativo leva em consideração o semestre selecionado e o período esperado da disciplina de certa turma. Utilizando de lógica similar, também é indicado caso não tenha sido atribuído um período à disciplina, e se, para o curso de Ciência da Computação, a disciplina é considerada como **Eletiva Livre**, **Eletiva Optativa**, ambas em tons azulados, ou se não é uma disciplina para o curso de Ciência da Computação, sendo então representada em tons alaranjados. Estas características são ilustradas no lado direito da Figura 30.

Figura 30 – Avisos flutuantes dos conflitos de disciplinas



Fonte: autoria própria

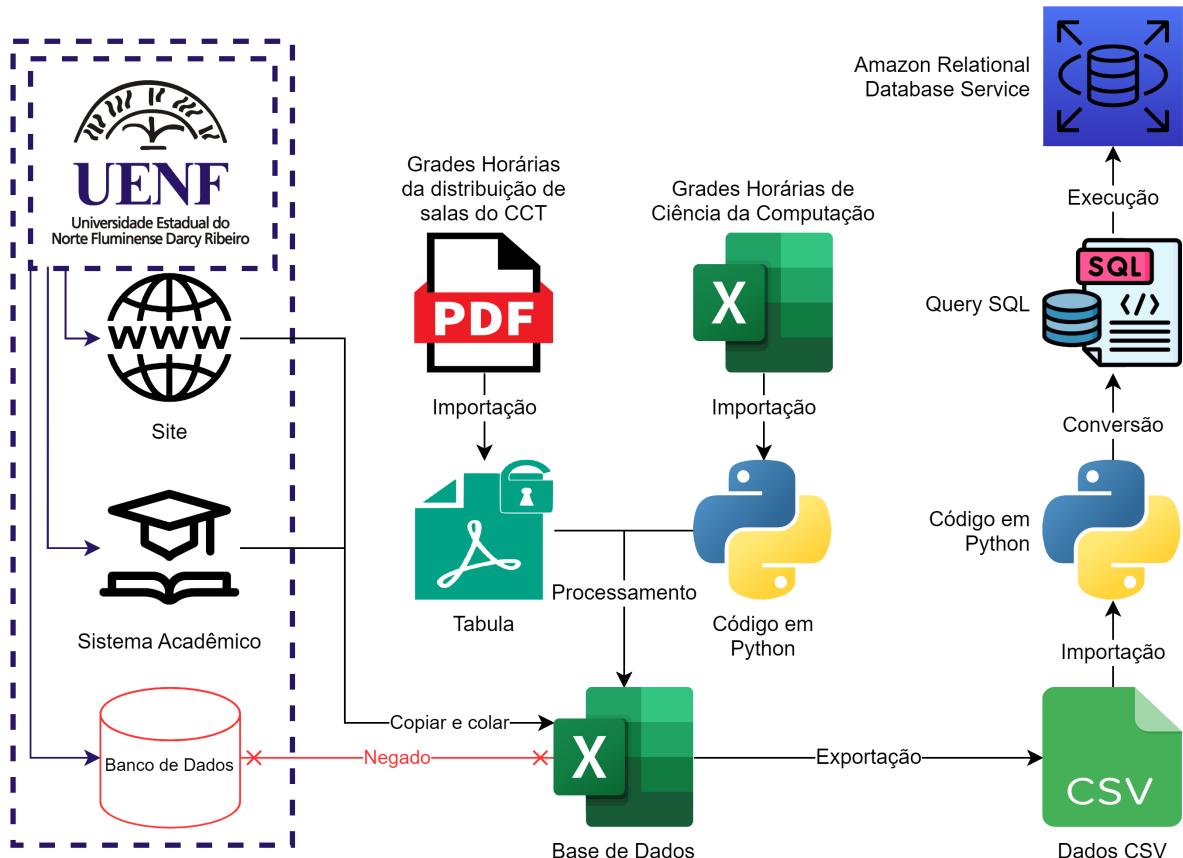
Já no lado esquerdo da Figura 30, vemos os conflitos que correlacionam os períodos esperados das disciplinas obrigatórias do curso de Ciência da Computação com o semestre em que foram ofertadas. Os semestres possíveis são três: o primeiro semestre, o segundo semestre e o “período de verão”. No caso do período de verão, as disciplinas que têm o seu período esperado neste semestre são marcadas com um tom amarelado, visto que não há relevância da sua paridade em um período de férias. Já nos casos das disciplinas de paridade ímpar (disciplinas dos períodos 1, 3, 5, 7 e 9) no primeiro semestre, ou as disciplinas de paridade par (disciplinas dos períodos 2, 4, 6, 8 e 10) no segundo semestre, estas são marcadas com um tom esverdeado, sendo aquelas referentes aos períodos finais do curso marcadas com um tom mais escuro. Já as disciplinas pares em semestres ímpares, ou as disciplinas ímpares em semestre pares, são ilustradas com a cor avermelhada, seguindo a mesma lógica de gradiente escuro nos últimos períodos.

1.6 Preenchimento de dados

Inicialmente, os dados adicionados faziam jus diretamente às disciplinas, professores, salas e turmas do curso de Ciência da Computação. Porém, como para a análise completa dos conflitos é necessário que seja feita também a adição das turmas de outros cursos, foi feito o preenchimento de dados para as entidades de professores, disciplinas e salas.

Para acumular mais dados referentes às entidades do banco de dados, foram tomadas algumas abordagens: requisição dos dados diretamente do Sistema Acadêmico, processamento de tabelas, processamento de PDFs e *web scraping*, todos eles ilustrados pela Figura 31.

Figura 31 – Diagrama do fluxo de obtenção de dados



Fonte: autoria própria

A forma teoricamente mais direta e eficiente para se obter os dados das entidades é a obtenção das informações contidas no banco de dados do sistema acadêmico. Para este fim, foi feita uma solicitação ao responsável pela Secretaria Acadêmica (SECACAD) da UENF, que direcionou a solicitação ao desenvolvedor do Sistema Acadêmico. A resposta obtida do desenvolvedor foi que a solicitação não poderia ser atendida, visto que não detinha a posse dos dados, e que para que pudesse fornecê-los, seria necessária uma solicitação formal à reitoria da UENF. Essa solicitação foi então passada à Coordenação do curso

de Ciência da Computação, com o qual ficou decidido abandonar a ideia e buscar outras formas de obtenção dos dados.

Paralelamente à abordagem anterior foram feitas tentativas individuais de obtenção dos dados. O processamento de tabelas e PDFs e o *web scraping* foram as abordagens utilizadas. Inicialmente os dados foram coletados e armazenados em tabelas, e então, convertidos para o formato CSV. Os arquivos CSV, por sua vez, foram utilizados em *scripts Python* que os convertiam em *queries SQL*, para que assim então fossem adicionados ao banco de dados.

Os PDFs analisados dispunham de tabelas referentes à oferta de turmas para o CCT, mas os dados advindos do processamento de PDF não são tão estruturados quanto os de uma tabela Excel, por este motivo, a primeira abordagem foi a solicitação dos arquivos tabulares para aquele que os produziu. Não havendo resposta favorável quanto a isso, diversos *softwares* de conversão de tabelas em PDF para Excel foram testados, porém, nenhum deles foi capaz de converter as tabelas de forma satisfatória. Um dos agravantes é a existência de células mescladas, o que torna mais complicada a conversão direta. Outra abordagem testada foi a de importação direta dos PDFs através do Excel e também o simples copiar e colar. Nenhum desses métodos foi eficiente, então com isso alguns dados foram coletados, mas sem certeza quanto à sua precisão. Deste método foram coletadas as informações referentes à nomes de professores e disciplinas, capacidades das salas, e horários de aulas.

Além dos PDFs anteriores, haviam também os arquivos referentes à oferta de turmas para o curso de Ciência da Computação. Estes sim dispunham também de sua versão em Excel, e assim, foram processados utilizando *scripts Python*. A abordagem apresentou falhas, visto que a notação das informações não apresentava o mesmo padrão ao longo dos anos, então foi necessário fazer ajustes manuais, resultando em uma tabela normalizada com diversas pastas de trabalhos referentes a cada um dos semestres desde 2019, não considerando os semestres de verão. Deste método foram coletadas as informações referentes à nomes e apelidos de professores e disciplinas, demandas estimadas dos alunos pela turma, descrição da turma, e horários de aulas.

Por fim, foi feita o *web scraping* que consistiu na busca por informações em diversos sites, principalmente o Sistema Acadêmico¹¹, o site da UENF¹² e outros sites. Nessa etapa, foi possível encontrar lotes de informações estruturadas. Um dos lotes foi a listagem de disciplinas e suas características que se encontram disponíveis no Sistema Acadêmico da UENF, essas informações foram copiadas e coladas no arquivo Excel unificado. Outros lotes foram encontrados dispersos ao longo do site da UENF e consistiam basicamente em listagem de professores e seus respectivos laboratórios. Essas informações estavam

¹¹ <<https://academico.uenf.br>>

¹² <<https://uenf.br/portal>>

dispersas dos sites dos diversos centros e cursos, alguns disponíveis no próprio site, outros em formato de arquivo. Deste método foram coletadas as informações referentes à nomes de professores, seus laboratórios e centros; e disciplinas e seus nomes, códigos e períodos de vigência. Além disso, foram encontrados documentos oficiais que referenciam a capacidade de ocupantes das “salas” disponíveis do Centro de Convenções, popularmente conhecido como “Apitão” que mesmo não sendo propriamente uma sala de aula, já foi utilizado previamente para tal fim. Outras salas já obtiveram alocações similares, assim como a Sala dos Professores.