UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ivan Guimarães Monte

Uma Nova Proposta de *Frontend* para o Sistema *Online* de Distribuição de Disciplinas da FACOM

Uberlândia, Brasil

2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ivan Guimarães Monte

Uma Nova Proposta de *Frontend* para o Sistema *Online* de Distribuição de Disciplinas da FACOM

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Bruno Augusto Nassif Travençolo

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2022

Ivan Guimarães Monte

Uma Nova Proposta de *Frontend* para o Sistema *Online* de Distribuição de Disciplinas da FACOM

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 05 de setembro de 2022:

Prof. Dr. Bruno Augusto Nassif Travençolo
Orientador

Prof. Dr. Henrique Coelho Fernandes

Prof. Dr. Rafael Dias Araújo

Uberlândia, Brasil
2022

Resumo

Na Faculdade de Computação da Universidade Federal de Uberlândia (UFU) se utiliza um Sistema *Online* de Distribuição de Disciplinas (SODD) para seleção dos docentes que irão ministrar as disciplinas em cada semestre respeitando determinadas regras preestabelecidas. A proposta deste projeto é melhorar o sistema atual reescrevendo ele por completo, utilizando tecnologias e linguagens mais atuais para aplicações web e uma interface mais amigável. Como principal foco, neste trabalho foi feito a implementação do lado do *frontend* de algumas funcionalidades para os usuários administradores e para os professores, além de dar suporte no desenvolvimento de algumas funcionalidades no *backend*. Todas essas funcionalidades que foram implementadas, são funcionalidades que já existem no sistema que está em vigência até a data presente.

Palavras-chave: Sistema de Distribuição de Disciplinas, Manutenção do Software, Atualização do Software, Reescrita do Software.

Lista de ilustrações

Lista de abreviaturas e siglas

CDD	Comissão de Distribuição de Disciplinas
COBOL	*Common Business Oriented Language*
CRUD	*Create Read Update Delete*
CSS	*Cascading Style Sheets*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
JS	JavaScript
MVC	*Model View Controller*
ORM	*Object Relational Mapper*
POO	Programação Orientada a Objetos
SODD	Sistema *Online* de Distribuição de Disciplinas
TS	TypeScript
UFU	Universidade Federal de Uberlândia
WEB	*World Wide Web*

Sumário

1 Introdução

Na Faculdade de Computação da Universidade Federal de Uberlândia (UFU) em todos os semestres a Comissão de Distribuição de Disciplinas (CDD) faz o uso de um Sistema *Online* de Distribuição de Disciplinas (SODD) para distribuir as disciplinas entre os docentes. O SODD é um *software web* que foi criado e implantando em 2015 pelos formandos da época para auxiliar e facilitar todo esse processo que antes era feito manualmente por um grupo de docentes que compunham a CDD.

O atual sistema possui o seu *backend* escrito na linguagem de programação JAVA, e para o funcionamento da aplicação, foi-se utilizado o servidor de aplicação JAVA chamado Apache Tomcat (Apache Tomcat, 2021). Já o *frontend*, é composto por HTML, CSS e JS e utiliza como principal *framework* o AngularJS, ainda na sua primeira versão que já está bastante desatualizada e não recebe mais manutenção de sua principal desenvolvedora, a empresa multinacional Google (Wikipedia, 2022a). O sistema também utiliza o *Bootstrap*, *framework* CSS criado e mantido pela empresa Twitter (Guilherme Lima, 2022).

Como todo *software*, o SODD também necessita de manutenção de tempos em tempos, e com o passar do tempo algumas tecnologias vão ficando obsoletas, e com isso possuindo cada vez menos desenvolvedores dispostos à aprendê-la, tornando assim cada vez mais difícil a manutenção do *software* devido à escassez de mão de obra. Um exemplo deste problema são os sistemas escritos na linguem de programação COBOL, que foi criada em 1959 e que ainda é utilizada por grandes instituições financeiras, mas hoje em dia está cada vez mais difícil encontrar novos desenvolvedores que conhecem e entendem esta linguagem, e com a aposentadoria dos mais antigos, as instituições estão sendo aos poucos forçadas a migrar seus sistemas para linguagens mais atuais (Scott Carey, 2021).

Pensando nisso, um grupo de alunos propôs a refatoração de todo o sistema SODD, com o objetivo de atualizar as tecnologias utilizadas nele para facilitar a sua manutenção e atualizações nos próximos anos. Com isso, na parte que se refere ao *backend*, foi proposta utilização do NodeJS, um ambiente de execução JavaScript (JS), mas voltada para desenvolvimento de aplicações *backend* (Ivan de Souza, 2020). Até o momento que esta monografia foi escrita, o JS é uma das linguagens mais utilizada no mundo devido principalmente a sua versatilidade (Redação, 2020), pois com ela é possível desenvolver aplicações *web*, *mobile*, *desktop* e até mesmo para Apple TV e Android TV. Com isso, utilizar essa linguagem traria outro grande benefício para o projeto, que é ter o *frontend* e *backend* da aplicação escritos em linguagens de programação semelhantes, facilitando assim encontrar novos desenvolvedores para mantê-lo. Junto com o NodeJS, foi escolhido o *superset* TypeScript (TS) para substituir todo o código JAVA (Felipe Nascimento, 2021).

Foi definido também o uso do *framework* TypeORM, um *framework* todo escrito em TypeScript e que serve para que a comunicação com com o banco de dados PostgreSQL.

Na na parte do *frontend*, as principais tecnologias escolhidas foram o React, um *framework* criado e mantido pela empresa Facebook (Naélio Freires, 2019), acompanhado do TypeScript e o *framework NextJS* mantido pelos desenvolvedores da empresa Vercel (Guillermo Rauch, 2021).

2 Objetivos

2.1 Objetivo Geral

O objetivo deste projeto é reescrever todo o Sistema Online de Distribuição de Disciplinas, utilizando tecnologias e linguagens que são mais atuais, estão sendo mais utilizadas no mercado e que possuem uma maior comunidade de desenvolvedores para facilitar a manutenção ou implementação de novas funcionalidades no sistema.

2.2 Objetivos Específicos

Baseando-se no objetivo geral, podemos definir os seguintes objetivos específicos:

- Refazer todo o *frontend* da aplicação utilizando o *framework React* junto com o *NextJS*.
- Apoiar no processo de reescrita do *backend*.
- Melhorar usabilidade das funcionalidades existentes.
- Propor uma design mais limpo e amigável.

3 Métodos e Tecnologias

Este capítulo apresenta o ambiente e as ferramentas utilizadas no processo de criação do sistema, bem como, os procedimentos envolvidos no decorrer do projeto.

3.1 Métodos

Como o desenvolvimento deste software é bastante complexo e envolve mais de um desenvolvedor, foi escolhida a metodologia Kanban para gerenciar, distribuir e acompanhar o desenvolvimento das tarefas, assim conseguisse acompanhar de forma mais fácil e clara o progresso no desenvolvimento da aplicação, bem como o que ainda precisa ser feito (Kanban, 2021). Com essa metodologia definida, foi-se escolhido o uso do sistema *web* chamado Trello, um site onde se pode criar um quadro no estilo Kanban e convidar outras pessoas para participarem dele (Wikipedia, 2022b).

Na parte de desenvolvimento do *frontend*, o *framework NextJS*, adota por padrão uma estrutura de pastas compostas por *assets*, *components* e *pages*. Na pasta ***assets*** deve-se colocar todos os arquivos de imagens, fontes e ícones. Na pasta ***components*** ficarão todos os componentes da aplicação que podem ser reutilizados por mais de uma página como cabeçalho, botões ou rodapé. Na pasta ***pages*** ficarão todas as páginas que estão disponíveis na aplicação. A Figura 1 mostra um exemplo da estrutura de pastas do *frontend*.

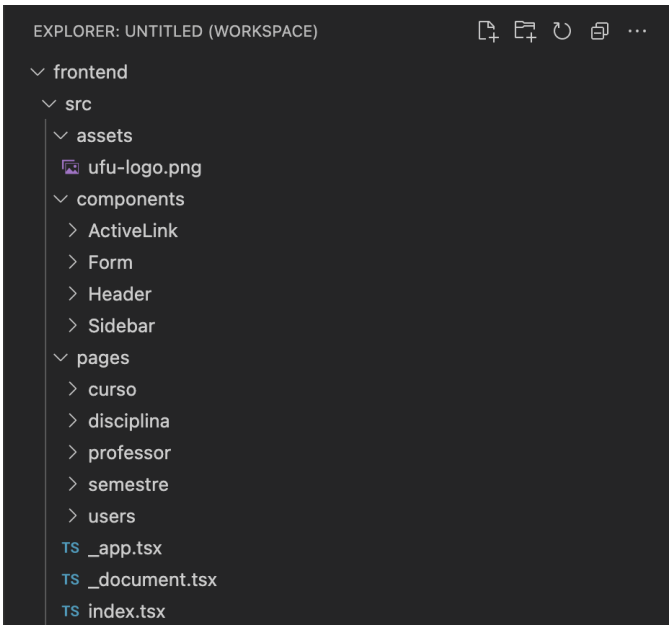

Figura 1 – Estrutura de pastas do *frontend*.

Como padrão de projeto e arquitetura de *software*, foi adotado o modelo *Model View Controller* (MVC). Este padrão é constituído de três camadas. Na camada chamada de ***Model***, encontra-se toda a lógica da aplicação, como regras de negócios e comunicação com banco de dados para inserção, remoção, atualização ou busca dos dados. Na camada ***View***, é apresentada a saída dos dados e a interface do usuário. Na camada de ***Controller***, é feita a comunicação entre a *Model* e a *View* (Vitor Zucher, 2020). Na Figura 2 temos uma representação visual de como as camadas se comunicam (Wikipedia, 2021).

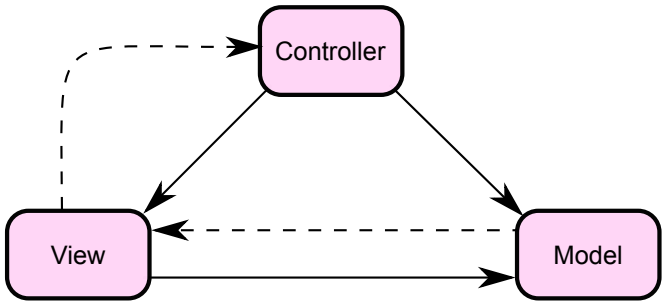

Figura 2 – Diagrama arquitetura MVC

Neste padrão aplicado no sistema, toda a interação começa na camada *view*, onde o usuário interage. A partir deste ponto, a camada *controller* pega as informações enviadas pela camada *view* e as envia para a camada *model* que fica responsável por avaliar e aplicar todas as regras de negócio necessárias. Após todo este processo, a camada *model* se comunica de volta com a camada *controller*, que por sua vez se comunica com a *view*.

3.2 Regras do Sistema

3.2.1 Regras Gerais

O sistema SODD comportará dois tipos de usuários, usuários **admin** e **professor**. Cada tipo de usuário poderá acessar telas específicas e executar tarefas específicas no sistema.

3.2.2 Usuários Admin

Esse perfil de usuário será utilizado pelas pessoas que ficarem responsáveis de administrar o sistema. Como admin, os usuários podem executar as seguintes ações:

- Criar, editar e remover cursos, usuários, semestres, professores, disciplinas.
- Remover a permissão de admin de algum usuário.

3.2.3 Usuários professor

Esse perfil de usuário, será utilizado pelos professores. Esse tipo de usuário pode executar as seguintes tarefas:

- Adicionar e remover restrições de horários para no semestre.

3.2.4 Ambos

Essas funcionalidades podem ser utilizadas por ambos os perfis:

- Visualizar as filas das disciplinas para saber a ordem de prioridade.
- Visualizar um calendário com o o horário das matérias que ele selecionar.

3.3 Tecnologias

3.3.1 JavaScript

Linguagem de programação *frontend* projetada e utilizada inicialmente para o desenvolvimento de aplicações *web*. Com o passar dos tempos, novas propostas surgiram para ela, e hoje essa linguagem é utilizada para se desenvolver aplicações *backend*, *web*, *desktop* e *mobile*.

3.3.2 TypeScript

SuperSet utilizado tanto no *backend* quanto *frontend*. Seu objetivo é trazer tipagem forte as variáveis, já que tanto JavaScript quanto o NodeJS por si só, possuem tipagem fraca. Tipagem forte é quando o tipo das variáveis é definida em tempo de desenvolvimento, ou seja, o desenvolvedor tem que dizer explicitamente no programa se uma variável e do tipo inteiro, *string*, *boolean* ente outros. Já a tipagem fraca, o tipo da variável é definida em tempo de execução, ou seja, o desenvolvedor não precisa declarar qual o tipo da variável quando está escrevendo o programa pois esse trabalho ficará por conta do compilador. Além disso, o TypeScript proporciona um melhor uso da programação orientada a objetos (POO), visto que somente com JavaScript ou NodeJS até é possível aplicar os conceitos de POO, mas de forma bastante limitada. Um exemplo de limitação é a utilização de **interfaces** e **classes abstratas**, que só podem ser implementadas em JavaScript e NodeJS utilizando-se TypeScript.

3.3.3 React

Framework JavaScript criado e mantido pela empresa Facebook, ele será utilizado para criar todo o *frontend* da aplicação. Seu objetivo inicial era facilitar e potencializar o desenvolvimento de aplicações *web*, mas com o seu crescimento e aceitação da comunidade de desenvolvimento, este *framework* começou a ser utilizado em outras áreas e hoje com ele é possível se criar aplicações *mobile*, *desktop* e até mesmo para *Smart* TVs.

3.3.4 NextJS

Framework NodeJS criado e mantido pela empresa Vercel. Ele é um complemento do React, tornando mais rápido e fácil o desenvolvimento de aplicações React, pois ele já traz consigo algumas funcionalidades criadas e configuradas, como, roteamento das páginas, *hot code reloading*, que é uma funcionalidade para ajudar os desenvolvedores, onde qualquer alteração feita no código é refletida em tempo real no ambiente de desenvolvimento.

4 Trabalhos Correlatos

4.1 História do SODD

Na monografia escrita por (DAMASCENO, 2021), pode-se ter um conhecimento sobre todo o processo de desenvolvimento que o sistema atual passou, contendo quando cada funcionalidade foi implementada e quem as implementou. No entanto, por se tratar de um sistema que foi sempre construído e melhorado baseado em projetos de conclusão de curso, e a cada solicitação de melhoria no sistema, uma nova equipe sem ligação nenhuma com a anterior era montada para desenvolver os novos requisitos solicitados. Com isso, o sistema foi crescendo de forma desorganizada e sem seguir nenhum padrão específico, o código de algumas funcionalidades que não são mais utilizadas acabaram ficando no projeto, e isso foi tornando cada vez mais difícil dar manutenção no sistema atual, sem contar que as tecnologias utilizadas estão em versões bastante desatualizadas, principalmente a base de código do *frontend*, que utiliza o AngularJS como seu principal *framework* ainda em sua primeira versão, que no momento em que esta monografia foi escrita, versão essa que está descontinuada.

Com essas tecnologias e *frameworks* desatualizados, o projeto perde bastante pois, a sua manutenção fica cada vez mais difícil já que menos pessoas possuem conhecimento sobre essas tecnologias. No caso do AngularJS, o desempenho da sua primeira versão em relação a que temos atualmente é muito inferior como podemos ver na Figura 3 (André Felizardo, 2017).

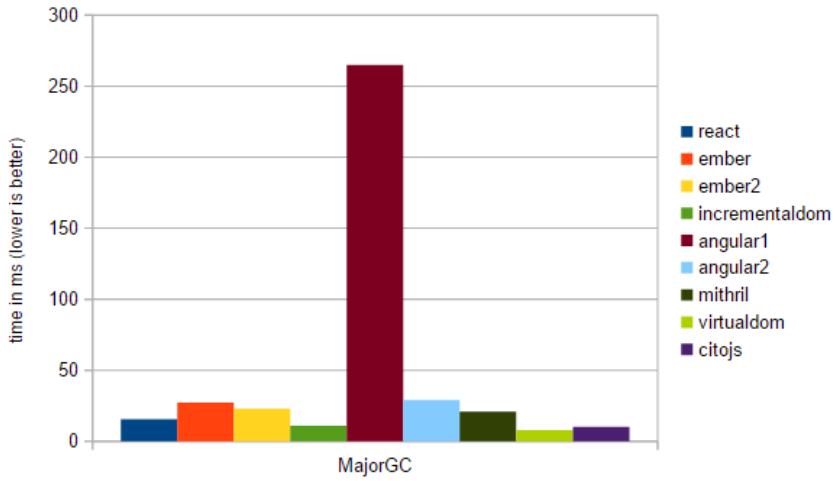

Figura 3 – Gráfico de comparação de desempenho do AngularJS em sua primeira versão em relação aos *frameworks* atuais.

4.2 Nova versão

Após estudar todo o processo de desenvolvimento que o projeto passou até hoje, bem como as tecnologia utilizadas, o trabalho de (MENDES, 2022) propôs reescrever todo o sistema utilizando tecnologias mais atuais, aplicando alguns padrões de projetos, criando e configurando testes unitários e automatizados, bem como, documentando todo o processo de desenvolvimento do sistema e suas funcionalidades existentes. Para isso, alguns *frameworks* foram escolhidos como, NodeJS e TypeScript ao invés do JAVA no backend, e React ao invés do AngularJS no *frontend*. Para escolha dessas tecnologias, foi levado em conta sua popularidade na comunidade de desenvolvimento e sua maturidade em relação as outras tecnologias que estão em alta. Além disso, também levou-se em conta seu conhecimento sobre as tecnologias para trabalhar com uma que o mesmo tivesse mais domínio.

O trabalho de (MENDES, 2022) consistiu em refazer toda a estrutura e a base da aplicação no lado do *backend*. Com isso, o lado do *frontend* não pode ser feito, o (MENDES, 2022) apenas criou uma configuração básica para que os próximos integrantes pudessem dar continuidade. Por esse motivo o sistema não pode ir ao ar quando o (MENDES, 2022) finalizou sua parte. Na implementação que foi feito nesta monografia, uma base do *frontend* foi desenvolvida, está base consiste em algumas páginas para que o usuário admin possa gerenciar os dados dos cursos, professores, semestres, matérias e acessos ao sistema. Além das funcionalidades para o admin, esta disponível também as funcionalidades para que os professores possam acompanhar as filas por disciplina, turma, entre outras, ver os horários de todas as disciplinas que eles quiserem em um calendário e marcar quais horários da semana eles não estarão disponíveis para ministrar qualquer aula.

4.3 Futuras implementações

Para continuidade do projeto, se faz necessário a implementação de algumas funcionalidades que já existem no sistema atual, funcionalidades essas como:

- Permitir aos professores poderem entrar nas filas das matérias.
- Permitir aos professores poderem escolher a ordem de prioridade das matérias que eles querem ministrar.
- Permitir ao admin poder alterar a ordem de prioridade das matérias dos professores.
- Permitir ao admin poder exibir ou esconder as funcionalidades do sistema, tais como os tipos de filas, verificar horários e restrições, escolher prioridades nas matérias e entrar nas filas.

- Permitir ao admin poder gerenciar as filas.
- Permitir ao admin poder atribuir as turmas aos professores.
- Permitir ao admin poder alterar os horários das turmas das disciplinas.

5 Desenvolvimento

Neste capítulo são apresentadas as tarefas que foram executadas para a criação e integração do *frontend* com o *backend*.

5.1 Diagrama caso de uso

As funcionalidades existentes no sistema devem ser acessadas de acordo com o perfil do usuário logado, algumas delas como as filas e o calendário de horários podem ser acessadas por ambos os perfis, enquanto que outras como gerenciar as contas dos usuários, cursos, professores só podem ser acessadas pelos usuários com perfil admin. Na Figura 4 temos um diagrama com as funcionalidades que cada usuário pode acessar no sistema.

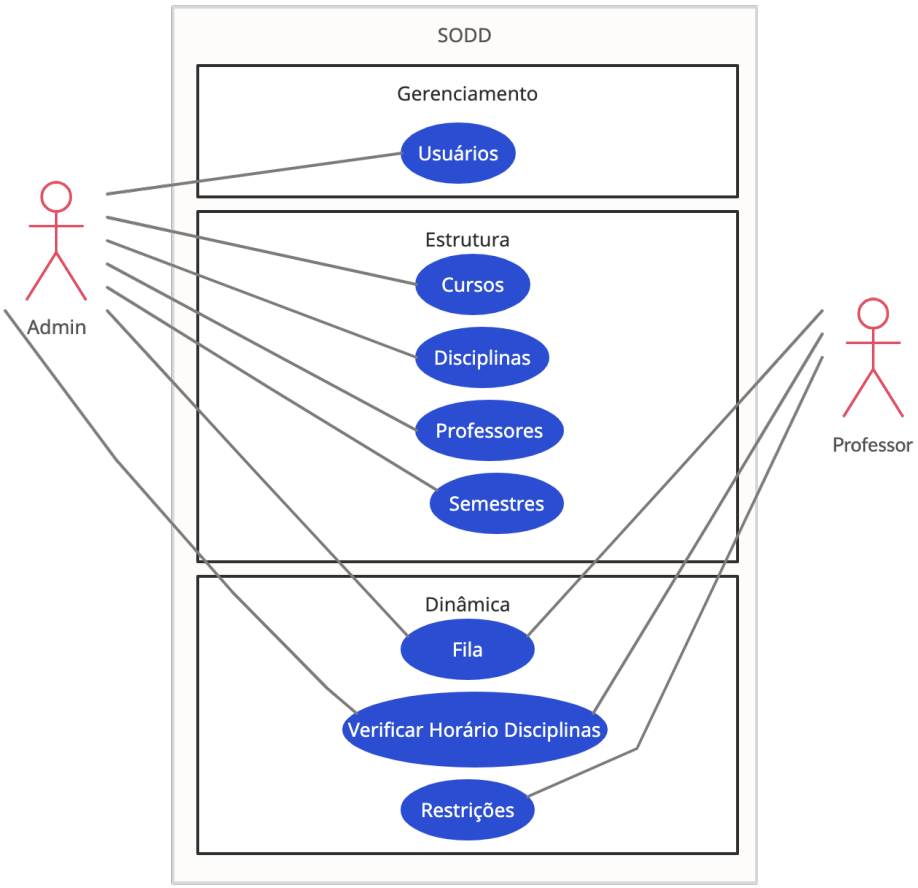

Figura 4 – Diagrama de caso de uso.

5.2 *Login*

Objetivo: Melhorar e integrar a tela de *login* com o *backend* Figura 5.

Nesta etapa, inclui validar se o e-mail e senha estão corretos perante a uma validação básica. Para o e-mail, essa validação básica consiste em verificar se é um e-mail válido. Já para a senha, consiste em validar se ela possui um número mínimo de caracteres. Após essa validação básica, essas informações são enviadas para o *backend* para que ele possa verificar se existe ou não um usuário cadastrado com essas credencias. Caso exista, o *backend* retornará um *token* de autenticação para o *frontend* e o *frontend* irá redirecionar o usuário para a tela de listagem de cursos caso seu perfil seja do tipo admin, se seu perfil for do tipo professor, então o sistema redirecionará o usuário para a tela de fila. Caso o e-mail ou senha estejam inválidos, o usuário será mantido na tela de *login* e o sistema irá exibir uma mensagem de erro avisando que o e-mail ou senha estão incorretos. Como complemento desta tarefa, também será necessário fazer uma validação em todas as rotas para impossibilitar que o usuário acesse uma rota que precisa estar logado sem ter passado pelo processo de *login*.

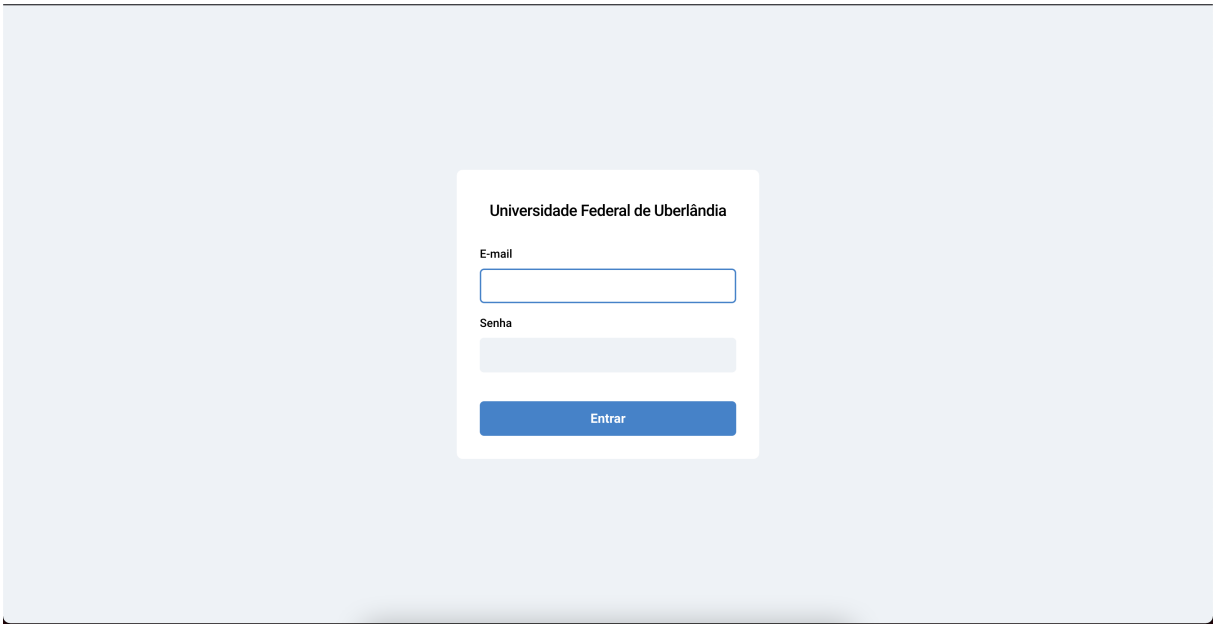

Figura 5 – Tela de *login*.

5.3 Cursos

5.3.1 Campos

- Código: Campo do tipo texto usado para identificar um curso, o valor deste campo deve ser único no sistema.

- Nome: Campo do tipo texto. Neste campo será salvo o nome do curso, pode haver mais de um curso com o mesmo nome.
- Unidade: Campo do tipo texto usado para identificar a qual faculdade o curso pertence.
- Campus: Campo do tipo texto usado para identificar a qual campus o curso pertence.
- Choque de Período: Campo to tipo *boolean*. Se verdadeiro, indica que o curso não permite que um professor ministre mais de uma disciplina do mesmo período.
- Choque de Horário: Campo to tipo *boolean*. Se verdadeiro, indica que o curso não permite que um professor ministre mais de uma disciplina que tenha horários iguais.

5.3.2 Listagem

Objetivo: Melhorar e integrar a tela de listagem de cursos com o *backend* Figura 6.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* busque as informações de todos os cursos disponíveis no banco de dados e exiba-os na tabela. Caso o *backend* encontre um erro durante o processo de busca das informações necessárias, este erro será informado ao *frontend*, que por sua vez, irá exibir uma mensagem de erro para o usuário informando que não foi possível buscar a lista de cursos disponíveis.

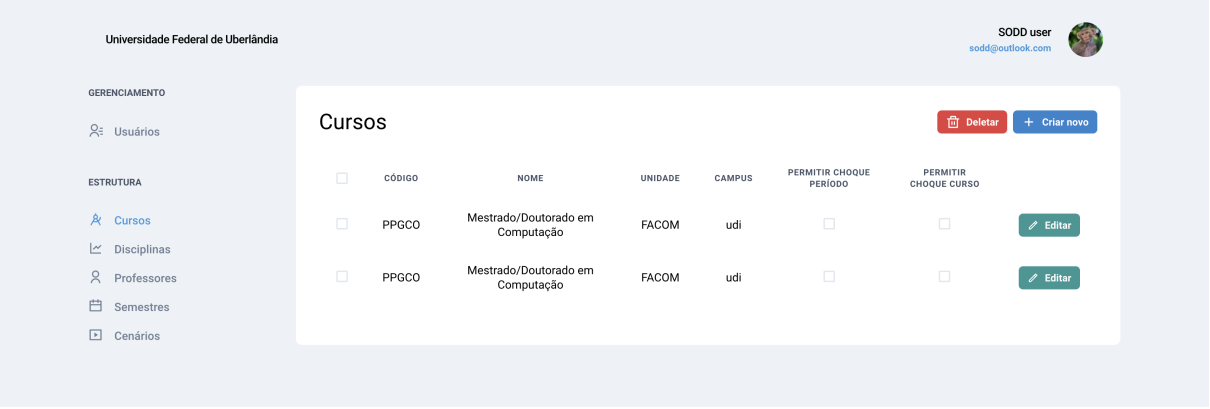

Figura 6 – Tela de listagem de cursos.

5.3.3 Criação

Objetivo: Melhorar e integrar a tela de criar curso com o *backend* Figura 7.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar as informações necessárias para que o *backend* consiga cadastrar um novo curso no banco de dados. No

formulário presente nesta tela, serão feitas algumas validações básicas antes dessas informações serem enviadas para o *backend*. O objetivo dessas validações é evitar que o usuário envie informações erradas ou faltando para o *backend*. Essas validações serão básicas, apenas se os campos necessários para o cadastro de um novo curso foram preenchidos. Após todo o preenchimento correto do formulário, os dados serão enviados para o *backend* para que ele possa validar e cadastrar o novo curso. Com o cadastro sendo realizado com sucesso, o usuário será redirecionado para a tela de listagem de cursos em que poderá ver o curso recém cadastrado. Caso ocorra algum erro no *backend* durante a tentativa de se criar o curso, o *backend* retornará este erro para o *frontend*, com isso, o usuário será mantido na tela de criação de curso e o sistema exibirá uma mensagem de erro para ele informando que não foi possível criar o curso pois ocorreu um erro no lado do servidor.

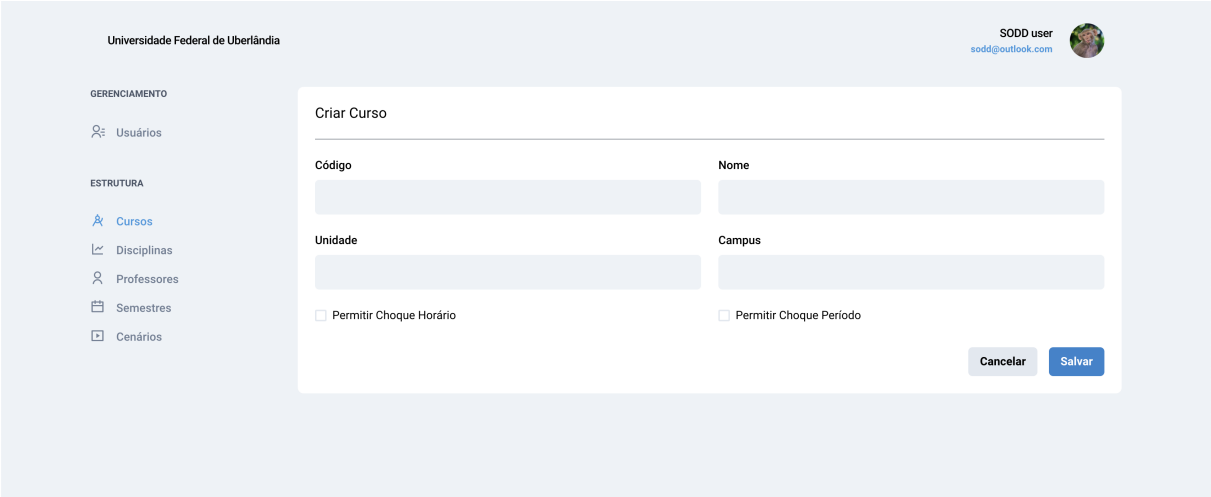

Figura 7 – Tela de criação de curso.

5.3.4 Edição

Objetivo: Melhorar e integrar a tela de editar curso com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa é fazer com que o *frontend* consiga buscar as informações necessárias de um curso específico, para que elas possam ser alteradas e depois enviadas novamente para o *backend*, de forma a atualizar essas informações do curso selecionado no banco de dados. Nesta etapa algumas informações não poderão ser editadas, como por exemplo, o código do curso. Caso o *backend* consiga atualizar os dados do curso com sucesso, ele enviará uma mensagem de sucesso para o *frontend*, que por sua vez, redirecionará o usuário para a tela de listagem de cursos e informará que as alterações foram salvas com sucesso. Caso ocorra algum erro no *backend* durante a tentativa de atualizar as informações do curso, o *backend* informará o *frontend* sobre o erro, o *frontend*

manterá o usuário na mesma tela e exibirá uma mensagem de erro informando que não foi possível atualizar as informações do curso.

5.3.5 Remoção

Objetivo: Melhorar e integrar a funcionalidade de deletar curso com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar uma requisição para o *backend* informando que ele deseja apagar um curso específico. Caso o *backend* encontre algum problema em apagar este curso, ele retornará o erro para o *frontend* que por sua vez exibirá para o usuário que não foi possível apagar este curso, e o usuário sera mantido na mesma tela. Caso o *backend* consiga apagar o curso, uma mensagem de sucesso será enviada para o *frontend*, que por sua vez irá exibir uma mensagem de sucesso e ele irá atualizar a listagem de cursos.

5.4 Disciplinas

5.4.1 Campos

- Código: Campo do tipo texto usado para identificar uma disciplina, o valor deste campo deve ser único no sistema.
- Nome: Campo do tipo texto. Neste campo será salvo o nome da disciplina, pode haver mais de uma disciplina com o mesmo nome.
- Carga teórica: Campo do tipo número usado para salvar o total de horas de aulas teóricas da disciplina.
- Carga prática: Campo do tipo número usado para salvar o total de horas de aulas práticas da disciplina.
- Carga total: Campo do tipo número usado para salvar o total de horas de da disciplina (soma de carga teórica com carga prática).
- Curso: Campo do tipo texto usado para ligar a disciplina com um curso.
- Período: Campo to tipo número. Período ideal em que a disciplina deve ser cursada, de acordo com o projeto pedagógico do curso
- Código Antigo: Campo to tipo texto. Caso a disciplina tenha alterado de código, é possível registrar qual o código anterior.

- Tem fila: Campo to tipo *boolean*. Se verdadeiro, indica que professores podem se candidatar para ministrar a disciplina, e portanto é possível formar uma fila de docentes interessados. Se falso, a disciplina é atribuída diretamente pela coordenação de curso (por exemplo, disciplinas optativas ou disciplinas do programa de pós-graduação).

5.4.2 Listagem

Objetivo: Integrar a tela de listagem de disciplinas com o *backend* Figura 8.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* busque as informações necessárias de todos as disciplinas disponíveis no banco de dados e as exiba na tabela. Caso o *backend* encontre algum erro durante o processo de busca das informações necessárias, este erro será informado ao *frontend*, que por sua vez, irá exibir uma mensagem de erro para o usuário informando que não foi possível buscar os dados para listar as disciplinas disponíveis.

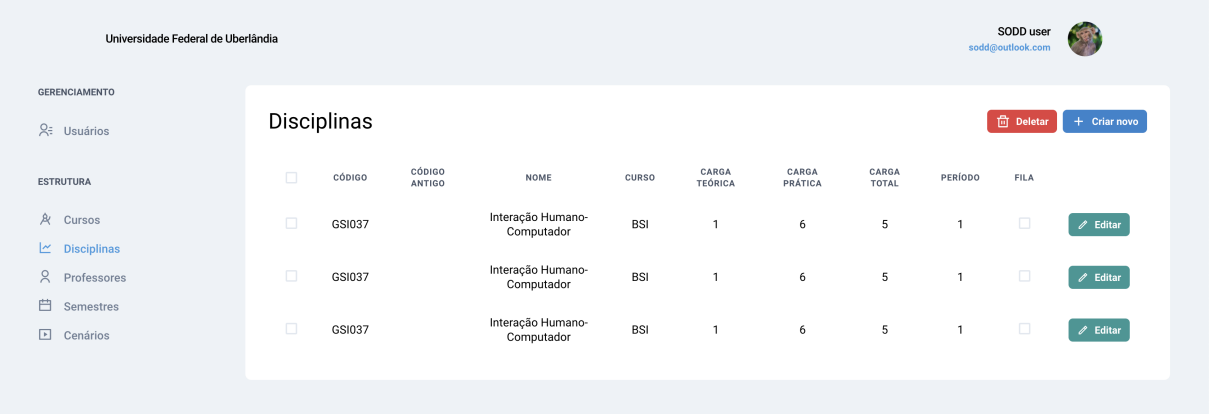

Figura 8 – Tela de listagem de disciplinas.

5.4.3 Criação

Objetivo: Melhorar e integrar a tela de criação de disciplina com o *backend* Figura 9.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar as informações necessárias para que o *backend* consiga cadastrar uma nova disciplina no banco de dados. No formulário presente nesta tela, será feita algumas validações básicas antes dessas informações serem enviadas para o *backend*. O objetivo dessas validações, é evitar que o usuário envie informações erradas ou faltando. Essas validações serão básicas, apenas se

os campos necessários para o cadastro de uma nova disciplina foram preenchidos. Após todo o preenchimento correto do formulário, os dados serão enviados para o *backend* para que ele possa validar e cadastrar a nova disciplina. Com o cadastro sendo realizado com sucesso, o usuário será redirecionado para a tela de listagem de disciplinas onde ele poderá ver a disciplina que ele acabou de cadastrar. Caso ocorra algum erro no *backend* durante a tentativa de se criar a disciplina, o *backend* retornará este erro para o *frontend*, com isso, o usuário será mantido na tela de criação de disciplina e o sistema exibirá uma mensagem de erro para ele informando que não foi possível criar a disciplina pois ocorreu um erro no lado do servidor.

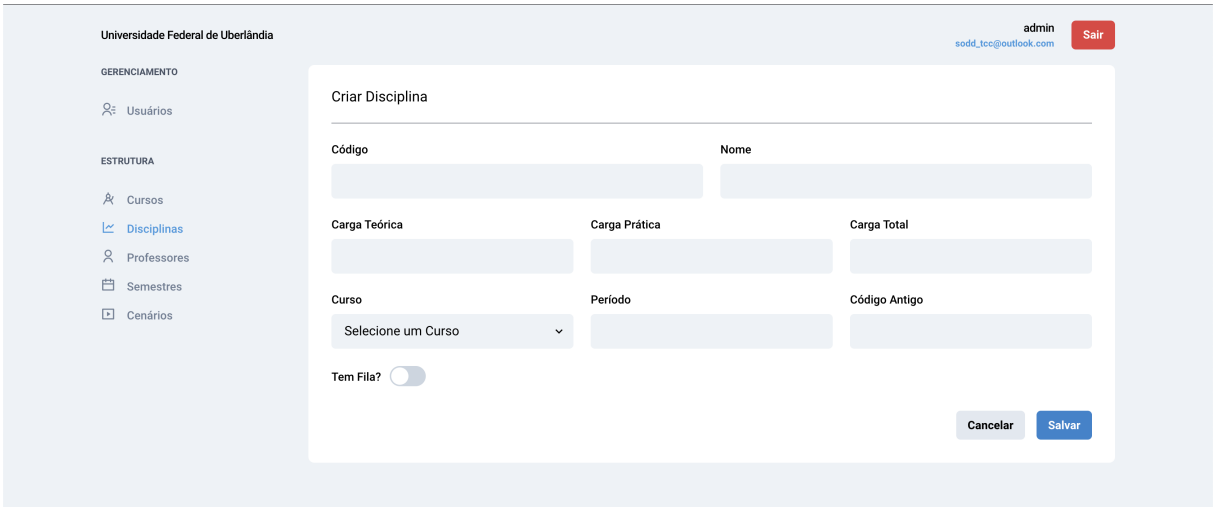

Figura 9 – Tela de criação de disciplina.

5.4.4 Edição

Objetivo: Melhorar e integrar a tela de editar disciplina com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga buscar as informações necessárias de uma disciplina específico para que elas possam ser alteradas e depois enviadas novamente ao *backend*, para que ele possa conseguir atualizar essas informações da disciplina selecionada no banco de dados. Nesta etapa algumas informações não poderão ser editadas, como por exemplo, o código da disciplina. Caso o *backend* consiga atualizar os dados da disciplina com sucesso, ele enviará uma mensagem de sucesso para o *frontend*, que por sua vez, redirecionará o usuário para a tela de listagem de disciplinas e informará que as alterações foram salvas com sucesso. Caso ocorra algum erro no *backend* durante a tentativa de atualizar as informações da disciplina, o *backend* informará o *frontend* sobre o erro, o *frontend* manterá o usuário na mesma tela e exibirá uma mensagem de erro informando que não foi possível atualiza as informações da disciplina.

5.4.5 Deleção

Objetivo: Melhorar e integrar a funcionalidade de deletar disciplina com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar uma requisição para o *backend* informando que ele deseja apagar uma disciplina específica. Caso o *backend* encontre algum problema em apagar esta disciplina, ele retornará o erro para o *frontend* que por sua vez exibirá para o usuário que não foi possível apagar esta disciplina, e o usuário sera mantido na mesma tela. Caso o *backend* consiga apagar a disciplina, uma mensagem de sucesso será enviada para o *frontend*, que por sua vez irá exibir uma mensagem de sucesso e ele irá atualizar a listagem de disciplinas.

5.5 Professores

5.5.1 Campos

- Siape: Campo do tipo texto usado para identificar um professor, o valor deste campo deve ser único no sistema.
- Nome: Campo do tipo texto. Neste campo será salvo o nome do professor, pode haver mais de um professor com o mesmo nome.
- Regime: Campo do tipo texto. Pode ser 20h, 40h ou dedicação exclusiva (*de*).
- Carga atual: Campo do tipo número. Carga horária que o docente deve ministrar no semestre atual
- CNome: Campo do tipo texto. Nome abreviado do docente, para facilitar buscas.
- Status: Campo do tipo texto usado para salvar a situação do professor na faculdade, seus valores podem ser: ativo, aposentado, exonerado e removido.
- Locação: Campo do tipo texto. Indica o local de trabalho do docente. Pode ser *udi* para Uberlândia, *pm* para Patos de Minas e *mc* para Monte Carmelo.
- Data de nascimento: Campo do tipo data, usado para salvar a data de nascimento do professor.
- Data ingresso: Campo do tipo data, usado para salvar em que o professor ingressou na faculdade.

- Data aposentadoria: Campo do tipo data, usado para salvar quando o professor se aposentou, este campo só deve conter valor caso o status do professor seja aposentado.
- Data exoneração: Campo do tipo data, usado para salvar quando o professor foi exonerado, este campo só deve conter valor caso o status do professor seja exonerado.
- Data saída: Campo do tipo data. A ser removido em versões futuras.
- Afastado: Campo do tipo *boolean*, usado para salvar se o professor está afastado ou não.

5.5.2 Listagem

Objetivo: Integrar a tela de listagem de professores com o *backend* Figura 10.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* busque as informações necessárias de todos os professores disponíveis no banco de dados e as exiba na tabela. Caso o *backend* encontre algum erro durante o processo de busca das informações necessárias, este erro será informado ao *frontend*, que por sua vez, irá exibir uma mensagem de erro para o usuário informando que não foi possível buscar os dados para listar os professores disponíveis.

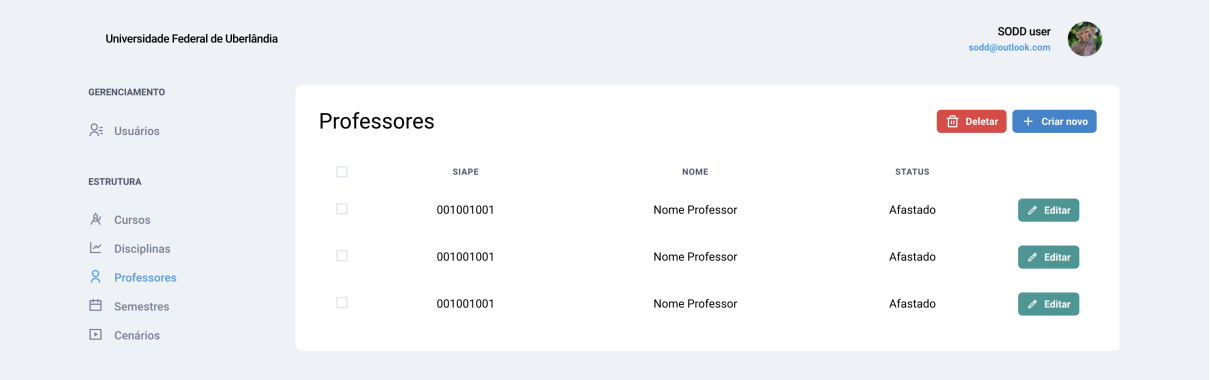

Figura 10 – Tela de listagem de professores.

5.5.3 Criação

Objetivo: Melhorar e integrar a tela de criação de professor com o *backend* Figura 11.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar as informações necessárias para que o *backend* consiga cadastrar um novo professor no banco de dados.

No formulário presente nesta tela, será feita algumas validações básicas antes dessas informações serem enviadas para o *backend*. O objetivo dessas validações, é evitar que o usuário envie informações erradas ou faltando. Essas validações serão básicas, apenas se os campos necessários para o cadastro de um novo professor foram preenchidos. Após todo o preenchimento correto do formulário, os dados serão enviados para o *backend* para que ele possa validar e cadastrar o novo professor. Com o cadastro sendo realizado com sucesso, o usuário será redirecionado para a tela de listagem de professores em que será possível consultar o professor recém cadastrado. Caso ocorra algum erro no *backend* durante a tentativa de se criar o professor, o *backend* retornará este erro para o *frontend*, com isso, o usuário será mantido na tela de criação de professor e o sistema exibirá uma mensagem de erro para ele informando que não foi possível criar o professor pois ocorreu um erro no lado do servidor.

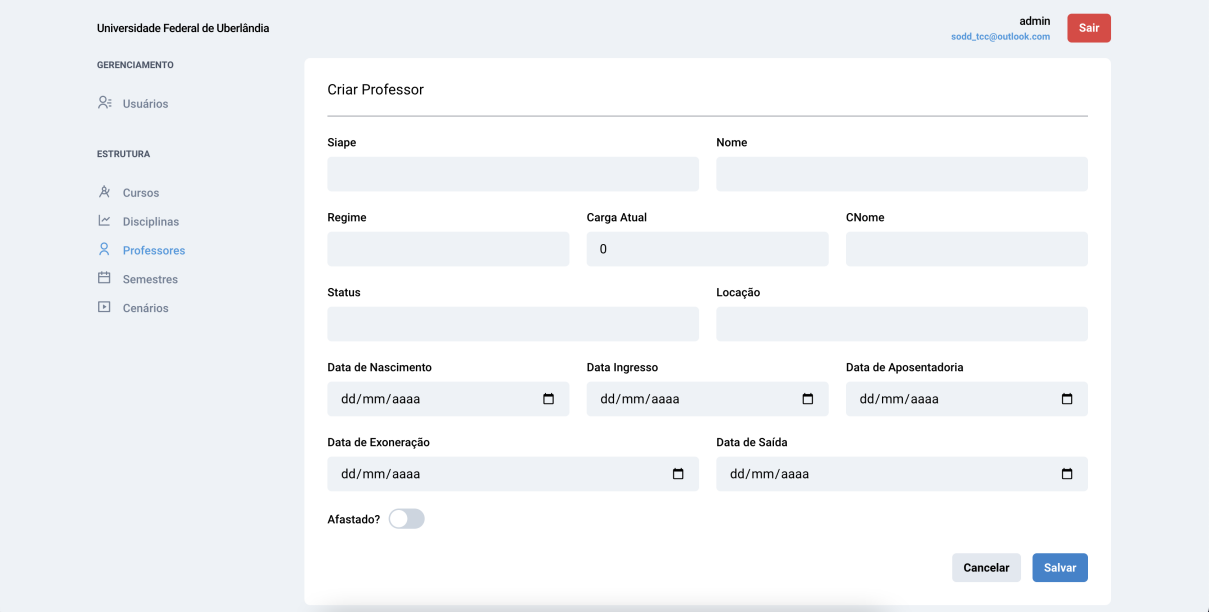

Figura 11 – Tela de criação de professor.

5.5.4 Edição

Objetivo: Melhorar e integrar a tela de editar professor com o *backend*.

O objetivo desta tarefa e fazer com que o *frontend* consiga buscar as informações necessárias de um professor específico para que elas possam ser alteradas e depois enviadas novamente ao *backend*, para que ele possa conseguir atualizar essas informações do professor selecionado no banco de dados. Nesta etapa algumas informações não poderão ser editadas, como por exemplo, o **SIAPE** do professor. Caso o *backend* consiga atualizar os dados do professor com sucesso, ele enviará uma mensagem de sucesso para o *frontend*, que por sua vez, redirecionará o usuário para a tela de listagem de professores e informará que as alterações foram salvas com sucesso. Caso ocorra algum erro no *backend* durante a

tentativa de atualizar as informações do professor, o *backend* informará o *frontend* sobre o erro, o *frontend* manterá o usuário na mesma tela e exibirá uma mensagem de erro informando que não foi possível atualiza as informações do professor.

5.5.5 Remoção

Objetivo: Melhorar e integrar a funcionalidade de remover professor com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar uma requisição para o *backend* informando que ele deseja apagar um professor específico. Caso o *backend* encontre algum problema em apagar este professor, ele retornará o erro para o *frontend* que por sua vez exibirá para o usuário que não foi possível apagar esta professor, e o usuário sera mantido na mesma tela. Caso o *backend* consiga apagar o professor, uma mensagem de sucesso será enviada para o *frontend*, que por sua vez irá exibir uma mensagem de sucecsso e ele irá atualizar a listagem de professores.

5.6 Semestres

5.6.1 Campos

- Ano: Campo do tipo número usado para salvar a qual ano aquele semestre pertence.
- Semestre: Campo do tipo número usado para salvar qual semestre é no ano.
- Ativo: Campo do tipo *boolean* usado para identificar se é o semestre atual.

5.6.2 Listagem

Objetivo: Integrar a tela de listagem de semestres com o *backend* Figura 12.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* busque as informações necessárias de todos os semestres disponíveis no banco de dados e as exiba na tabela. Caso o *backend* encontre algum erro durante o processo de busca das informações necessárias, este erro será informado ao *frontend*, que por sua vez, irá exibir uma mensagem de erro para o usuário informando que não foi possível buscar os dados para listar os semestres disponíveis.

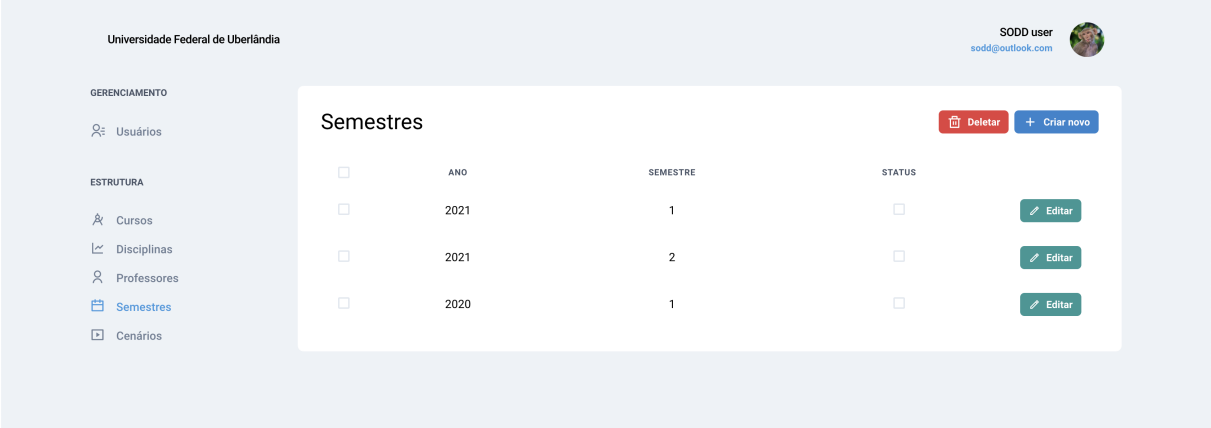

Figura 12 – Tela de listagem de semestres.

5.6.3 Criação

Objetivo: Melhorar e integrar a tela de criação de semestres com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar as informações necessárias para que o *backend* consiga cadastrar um novo semestre no banco de dados. No formulário presente nesta tela, será feita algumas validações básicas antes dessas informações serem enviadas para o *backend*. O objetivo dessas validações, é evitar que o usuário envie informações erradas ou faltando. Essas validações seram básicas, apenas se os campos necessários para o cadastro de um novo semestre foram preenchidos. Após todo o preenchimento correto do formulário, os dados seram enviados para o *backend* para que ele possa validar e cadastrar o novo semestre. Com o cadastro sendo realizado com sucesso, o usuário será redirecionado para a tela de listagem de semestres, onde ele poderá ver o semestre que ele acabou de cadastrar. Caso ocorra algum erro no *backend* durante a tentativa de se criar o semestre, o *backend* retornará este erro para o *frontend*, com isso, o usuário será mantido na tela de criação de semestre e o sistema exibirá uma mensagem de erro para ele informando que não foi possível criar o semestre pois ocorreu um erro no lado do servidor.

5.6.4 Edição

Objetivo: Melhorar e integrar a tela de editar semestre com o *backend* Figura 13.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga buscar as informações necessárias de um semestre específico para que elas possam ser alteradas e depois enviadas novamente ao *backend*, para que ele possa conseguir atualizar essas informações do semestre selecionada no banco de dados. Nesta etapa algumas informações não poderam

ser editadas, como por exemplo, o id do semestre. Caso o *backend* consiga atualizar os dados do semestre com sucesso, ele enviará uma mensagem de sucesso para o *frontend*, que por sua vez, redirecionará o usuário para a tela de listagem de semestres e informará que as alterações foram salvas com sucesso. Caso ocorra algum erro no *backend* durante a tentativa de atualizar as informações do semestre, o *backend* informará o *frontend* sobre o erro, o *frontend* manterá o usuário na mesma tela e exibirá uma mensagem de erro informando que não foi possível atualiza as informações do semestre.

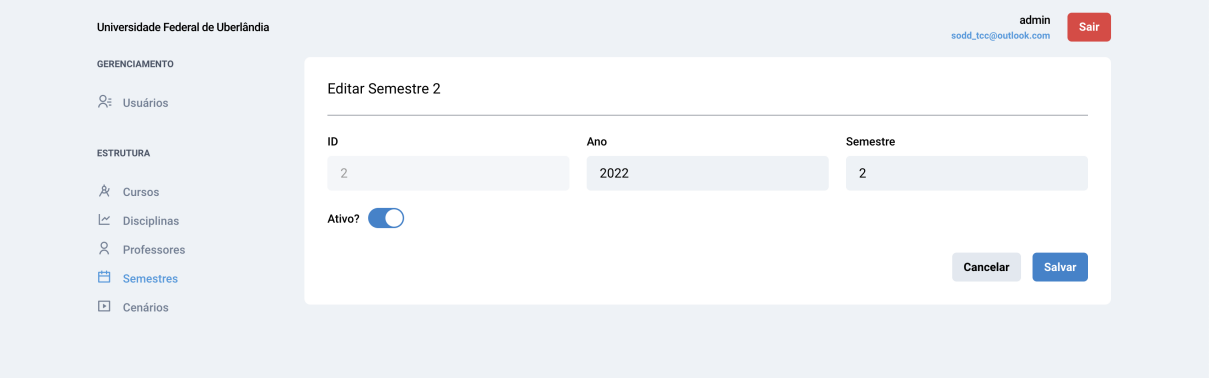

Figura 13 – Tela de edição de semestre.

5.6.5 Remoção

Objetivo: Melhorar e integrar a funcionalidade de remover semestre com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar uma requisição para o *backend* informando que ele deseja apagar um semestre específico. Caso o *backend* encontre algum problema em apagar este semestre, ele retornará o erro para o *frontend* que por sua vez exibirá para o usuário que não foi possível apagar este semestre, e o usuário sera mantido na mesma tela. Caso o *backend* consiga apagar o semestre, uma mensagem de sucesso será enviada para o *frontend*, que por sua vez irá exibir uma mensagem de sucecsso e ele irá atualizar a listagem de semestre.

5.7 Usuários

5.7.1 Campos

- Nome: Campo do tipo texto usado para identificar o nome do usuário no sistema.
- E-mail: Campo do tipo texto usado para que o usuário consiga fazer *login* no sistema.

- Senha: Campo do tipo texto usado para que o usuário possa fazer *login* no sistema.
- É admin: Campo do tipo *boolean* usado para identificar se a conta do usuário é do tipo admin ou não.
- Professor: Campo do tipo texto usado para ligar a conta a um professor cadastrado.

5.7.2 Listagem

Objetivo: Melhorar e integrar a tela de listagem de usuários com o *backend* Figura 14.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* busque as informações de **id**, **nome**, **e-mail** e se é **admin** de todos os usuários cadastrados no banco de dados e exiba-os na tabela. Caso o *backend* encontre um erro durante o processo de busca das informações necessárias, este erro será informado ao *frontend*, que por sua vez, irá exibir uma mensagem de erro para o usuário informando que não foi possível buscar a lista de usuários cadastrados.

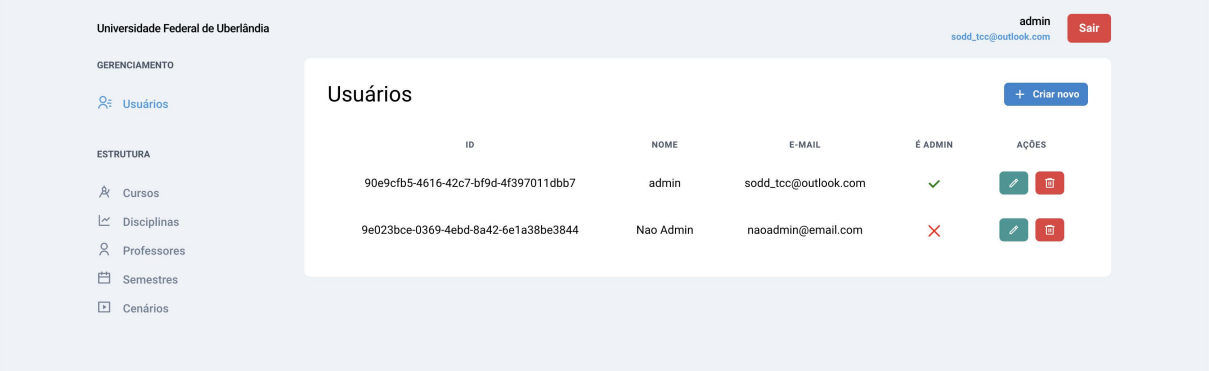

Figura 14 – Tela de listagem de usuários.

5.7.3 Criação

Objetivo: Melhorar e integrar a tela de criar usuário com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar as informações necessárias para que o *backend* consiga cadastrar um novo usuário no banco de dados. No formulário presente nesta tela, será feita algumas validações básicas antes dessas informações serem enviadas para o *backend*. O objetivo dessas validações, é evitar que o usuário que está preenchendo o formulário envie informações erradas ou faltando para o *backend*. Essas validações seram básicas, apenas se os campos necessários para o cadastro de um

novo usuário foram preenchidos. Após todo o preenchimento correto do formulário, os dados seram enviados para o *backend* para que ele possa validar e cadastrar o novo usuário. Com o cadastro sendo realizado com sucesso, o usuário que preencheu o formulário será redirecionado para a tela de listagem de usuários onde ele poderá ver o usuários que ele acabou de cadastrar. Caso ocorra algum erro no *backend* durante a tentativa de se criar o novo usuário, o *backend* retornara este erro para o *frontend*, com isso, o usuário será mantido na tela de criação de usuário e o sistema exibirá uma mensagem de erro para ele informando que não foi possível criar o usuário pois ocorreu um erro no lado do servidor.

5.7.4 Edição

Objetivo: Melhorar e integrar a tela de editar usuário com o *backend* Figura 15.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga buscar as informações necessárias de um usuário específico para que elas possam ser alteradas e depois enviadas novamente para o *backend*, para que ele possa conseguir atualizar essas informações do usuário selecionado no banco de dados. Nesta etapa algumas informações não poderam ser editadas, como por exemplo, o id do usuário. Caso o *backend* consiga atualizar os dados do usuário com sucesso, ele enviará uma mensagem de sucesso para o *frontend*, que por sua vez, redirecionará o usuário para a tela de listagem de usuários e informará que as alterações foram salvas com sucesso. Caso ocorra algum erro no *backend* durante a tentativa de atualizar as informações do usuário, o *backend* informará o *frontend* sobre o erro, o *frontend* manterá o usuário na mesma tela e exibirá uma mensagem de erro informando que não foi possível atualiza as informações do usuário.

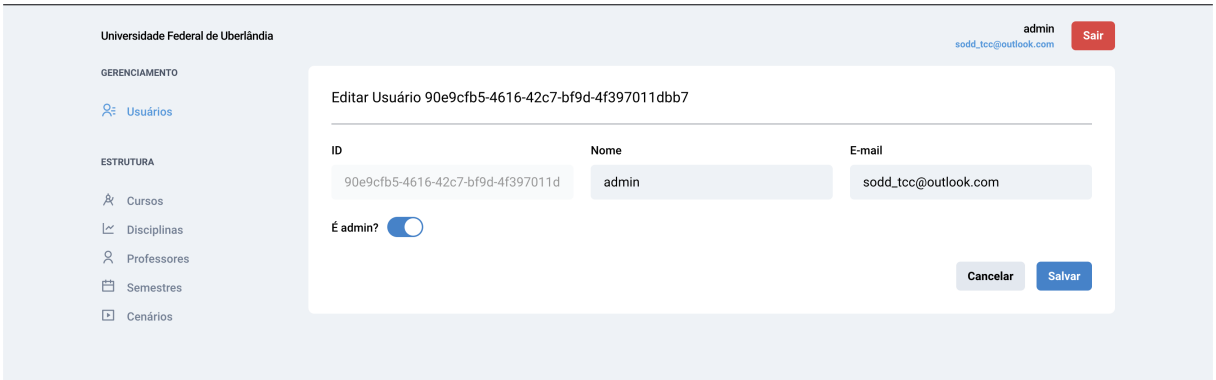

Figura 15 – Tela de edição de usuário.

5.7.5 Remoção

Objetivo: Melhorar e integrar a funcionalidade de remover usuário com o *backend*.

Regra: Somente usuários do tipo **admin** podem acessar esta tela.

O objetivo desta tarefa e fazer com que o *frontend* consiga enviar uma requisição para o *backend* informando que ele deseja apagar um usuário específico. Caso o *backend* encontre algum problema em apagar este usuário, ele retornára o erro para o *frontend* que por sua vez exibirá para o usuário que não foi possível apagar este usuário, e o usuário sera mantido na mesma tela. Caso o *backend* consiga apagar o usuário, uma mensagem de sucesso será enviada para o *frontend*, que por sua vez irá exibir uma mensagem de sucecsso e ele irá atualizar a listagem de usuários.

5.8 Fila

Objetivo: Exibir para os usuários as filas de prioridade de acordo com cada tipo.

Regra: Qualquer usuário pode acessar esta tela.

5.8.1 Tipos de filas

5.8.1.1 Fila por disciplina

Nesta tela, o usuário pode selecionar uma disciplina, e logo em seguida será exibido os professores que desejam ministrar essa disciplina. Esses dados serão exibidos por ordem de prioridade Figura 16.

Nesta tela, ao clicar no botão verde de informações que existe para cada professor listado, o usuário poderá ver toda a fila de disciplinas que aquele professor está na fila.

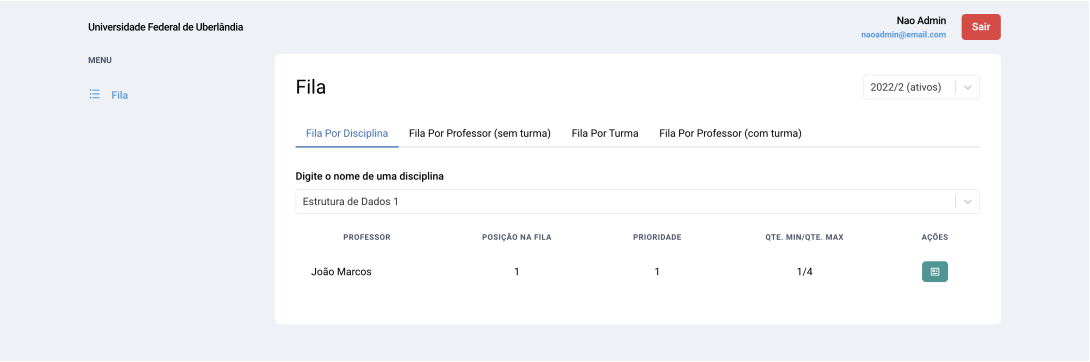

Figura 16 – Tela da fila por disciplina.

5.8.1.2 Fila por professor (sem turmas)

Nesta tela, o usuário pode selecionar um professor, e logo em seguida será exibido todas as disciplinas que este professor deseja ministrar sem levar em consideração as turmas. Esses dados serão exibidos por ordem de prioridade Figura 17.

Nesta tela, ao clicar no botão verde de informações que existe para cada disciplina listada, o usuário poderá ver toda a fila daquela disciplina.

Figura 17 – Tela da fila por professor (sem turmas).

5.8.1.3 Fila por turma

Nesta tela, o usuário pode selecionar uma turma de uma disciplina específica, e logo em seguida será exibido todos os professores que desejam ministrar nessa turma. Esses dados serão exibidos por ordem de prioridade Figura 18.

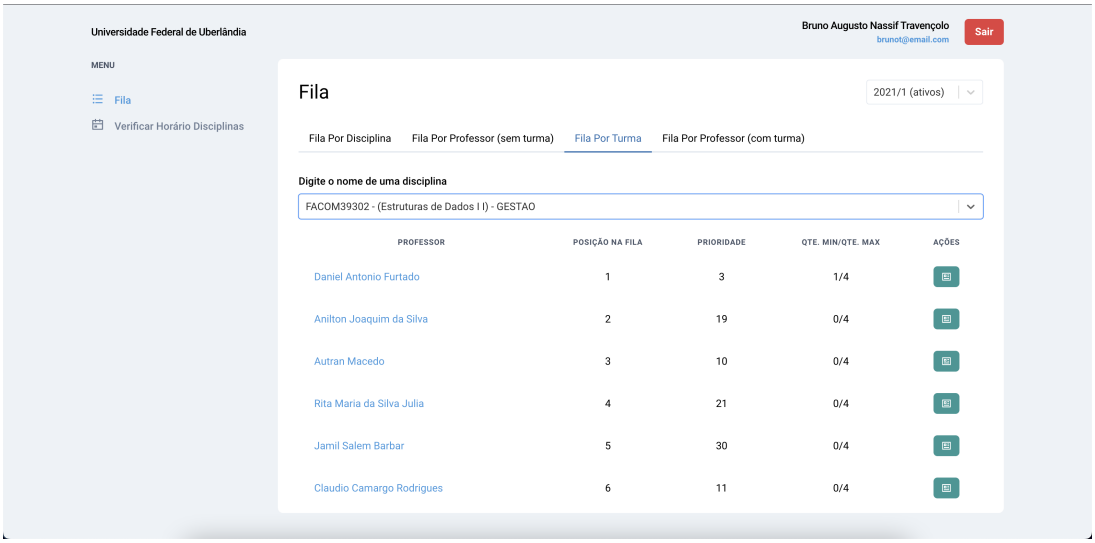

Figura 18 – Tela da fila por turmas.

5.8.1.4 Fila por professor (com turmas)

Nesta tela, o usuário pode selecionar um professor, e logo em seguida será exibido todas as disciplinas que o professor selecionado deseja ministrar levando em consideração as turmas. Esses dados serão exibidos por ordem de prioridade Figura 19.

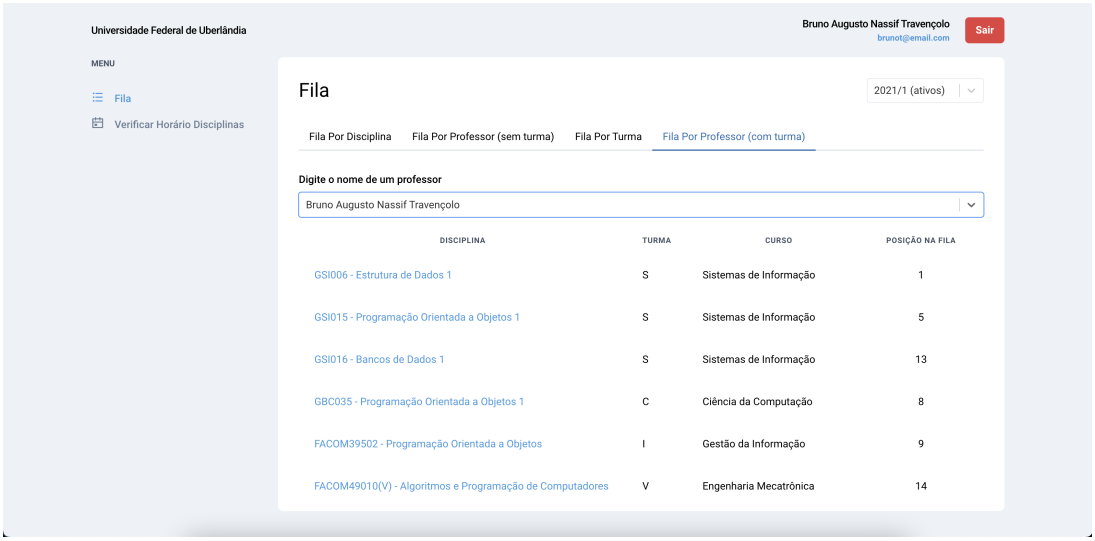

Figura 19 – Tela da fila por professor (com turmas).

5.9 Verificar Horário das Disciplinas

Objetivo: Exibir para os usuários um calendário onde ele possa ver os horários de cada disciplina Figura 20.

Regra: Qualquer usuário pode acessar esta tela.

Nesta tela, o usuário pode personalizar um calendário para conseguir ver os horários das disciplinas disponíveis e se existe choque de horário entre elas para que ele possa simular um calendário letivo com os horários das disciplinas que ele quer lecionar. Quando houver o choque de horário entre duas ou mais disciplinas, as disciplina que estão no mesmo horário serão exibidas de vermelho para se destacarem das demais.

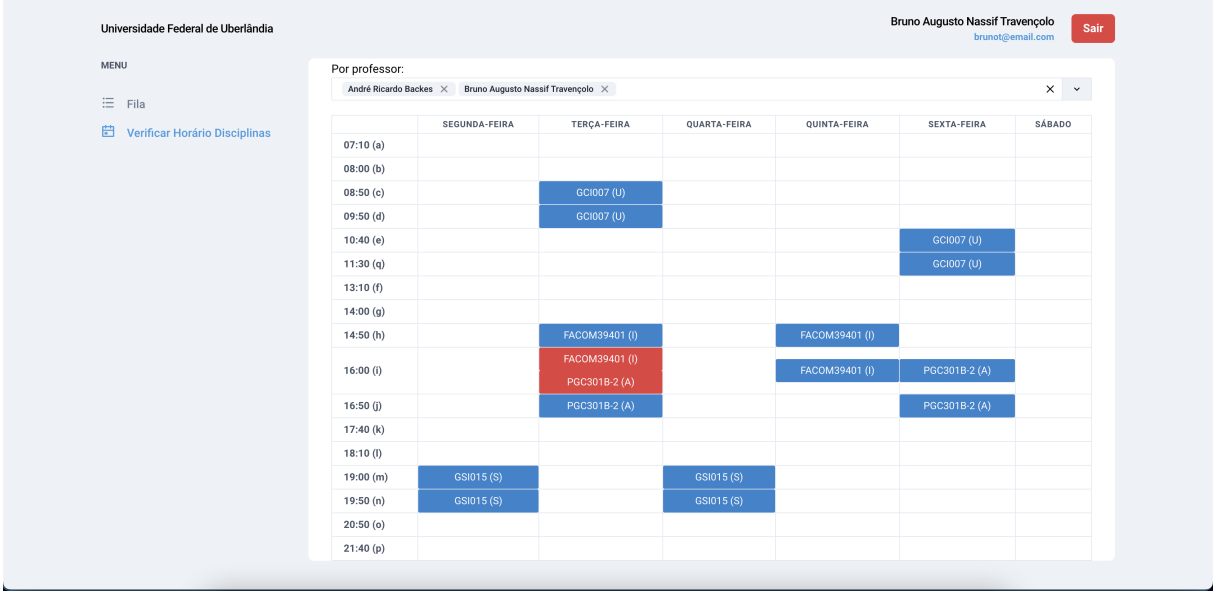

Figura 20 – Tela com o calendário das disciplinas.

Para execução desta tarefa, no *frontend*, foi necessário criar a tela com o calendário e suas respectivas funcionalidades para que o usuário consiga selecionar uma ou mais disciplinas que ele esteja na fila no respectivo semestre selecionado, consiga também selecionar uma ou mais disciplinas que ele **não** esteja na fila, consiga selecionar um ou mais professores para que ele possa ver o horário das matérias que esses professores estão lecionando. Além disso foi necessário criar um calendário para exibir todas as matérias com uma regra de que quando existe mais de uma matéria no mesmo horário, ambas fiquem marcadas de vermelho para que fique explícito que existe mais de uma matéria no mesmo horário. No lado do *backend* para que este calendário possa ser exibido foi necessário criar uma ligação entre usuário e professor para que fosse possível buscar as matérias somente do professor logado. Além disso, foi necessário criar dois *endpoints*, um para buscar as matérias que o professor logado está ou não na fila de acordo com o filtro enviado na requisição, e outro para buscar os horários das disciplinas de acordo com as turmas para que essas disciplinas possam ser exibidas no calendário. Abaixo temos uma

demonstração de como ficou o fluxo de interações para que o professor consiga ver no calendário o horário das turmas que foram selecionadas ou das turmas dos professores que foram selecionados Figura 21.

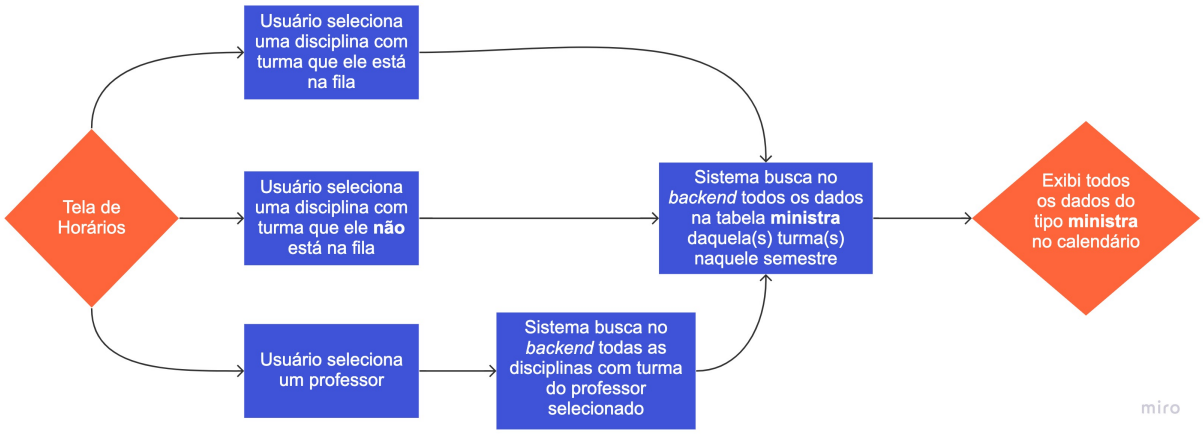

Figura 21 – Fluxo da busca de dados para o calendário de horários.

5.9.1 Filtros

Para auxiliar o usuário no momento de exibir as disciplinas no calendário, esta disponível para os usuários admin um filtro e para os professores três filtros Figura 22.

- Adicionar as disciplinas que estão na fila do professor logado.
- Adicionar as disciplinas que **não** estão na fila do professor logado.
- Adicionar todas as disciplinas que algum professor está ministrando.

5.9.1.1 Adicionar as disciplinas que estão na fila do professor logado

Neste filtro, o professor pode selecionar somente as disciplinas com as turmas que o ele entrou na fila no semestre selecionado.

5.9.1.2 Adicionar as disciplinas que **não** estão na fila do professor logado

Neste filtro, o professor pode selecionar somente as disciplinas com as turmas que ele **não** entrou na fila no semestre selecionado, ou seja, o professor pode selecionar as disciplinas coma as turmas que não se encontram no filtro anterior.

5.9.1.3 Adicionar todas as disciplinas que algum professor está ministrando

Neste último filtro, o usuário poderá selecionar qualquer professor e ver todas as turmas das disciplinas que o professor selecionado está ministrando no semestre selecionado.

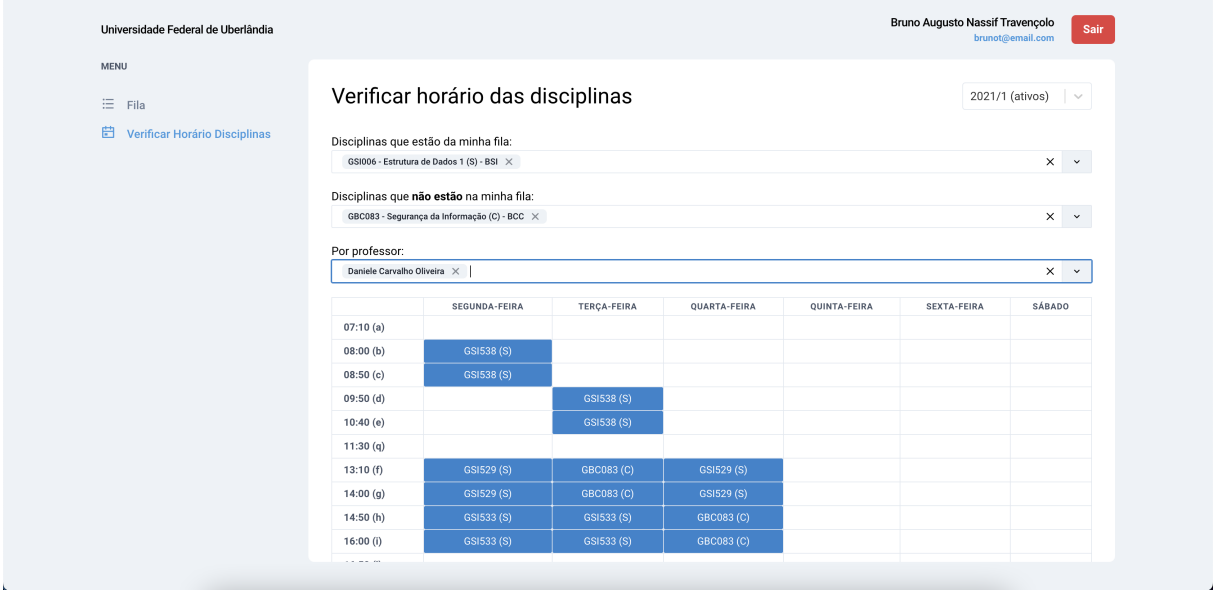

Figura 22 – Tela com os filtros calendário das disciplinas.

5.10 Restrições

Objetivo: Exibir para os usuários do tipo **professor** um calendário onde ele possa ver e marcar quais horários ele quer reservar Figura 23.

Regra: Somente usuários do tipo **professor** podem acessar esta tela.

Nesta tela, o usuário pode selecionar no calendário quais horários ele quer reservar para ficarem bloqueados no momento de distribuição de matérias, ou seja, o professor não pretende lecionar nenhuma matéria nos horários selecionados. Para que o professor execute esta tarefa, basta selecionar no calendário quais horários ele quer reservar, os horários selecionados ficam marcados de vermelho.

Para execução desta tarefa, no *frontend*, foi necessário criar a tela com o calendário e suas respectivas funcionalidades para que o professor consiga marcar e desmarcar um horário no calendário, bem como enviar essas informações para o *backend* para serem salvas. No lado do *backend* foi necessário alterar duas rotas. A primeira, foi a rota que cria restrição, antes nela só era possível criar uma restrição por requisição, mas com a alteração feita, é possível criar somente uma ou várias ao mesmo tempo. A segunda, foi a rota de apagar restrição, antes nesta rota só era possível apagar uma restrição por vez informando o siape do professor, e o horário específico, mas após as alterações feitas, é possível apagar todas as restrições do professor logado de uma só vez.

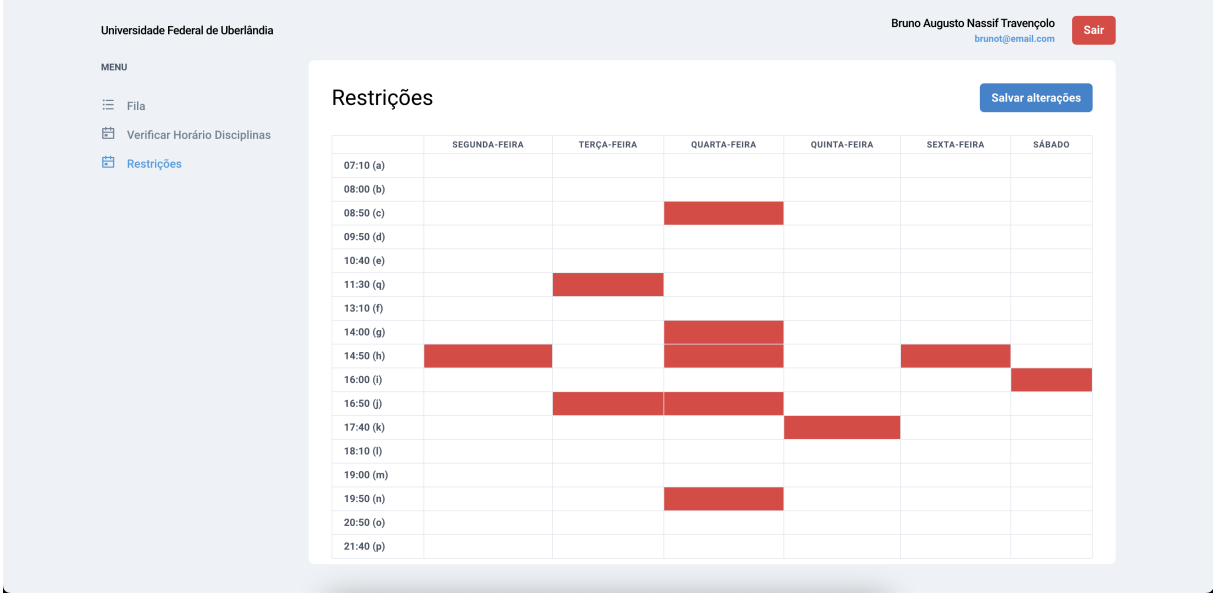

Figura 23 – Tela com o calendário de restrições.

Abaixo do calendário com os horários selecionados, o usuário pode ver de forma mais clara os dias e horários que estão marcados como bloqueados no banco de dados. Esta lista só é atualizada no momento que o professor carrega a tela pela primeira vez ou quando ele salva uma nova regra de restrições, isso é feito para que ele consigo comparar

as restrições que estão no banco com a nova regra que está sendo criada Figura 24.

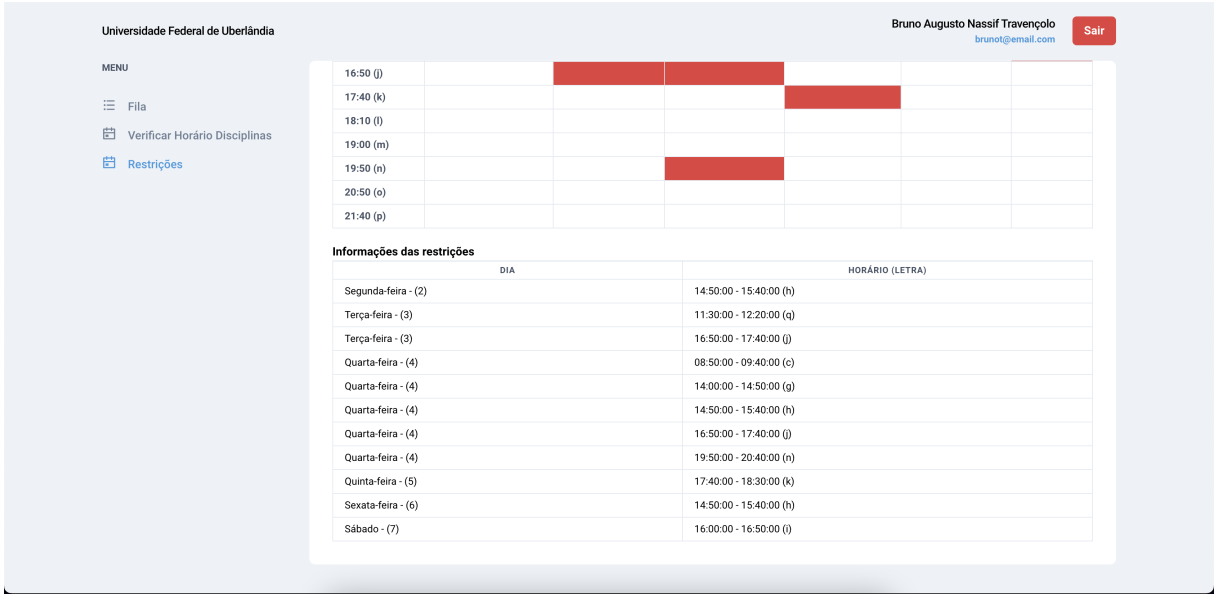

Figura 24 – Tela com a lista de restrições.

6 Mudanças visuais

Este capítulo apresenta uma comparação entre a aparência do sistema atual e da nova versão que foi desenvolvida.

6.1 Tela após o *login*

Na Figura 25 é apresentada a antiga tela que os usuários com perfil professor visualizavam após o *login* e a sua nova versão.

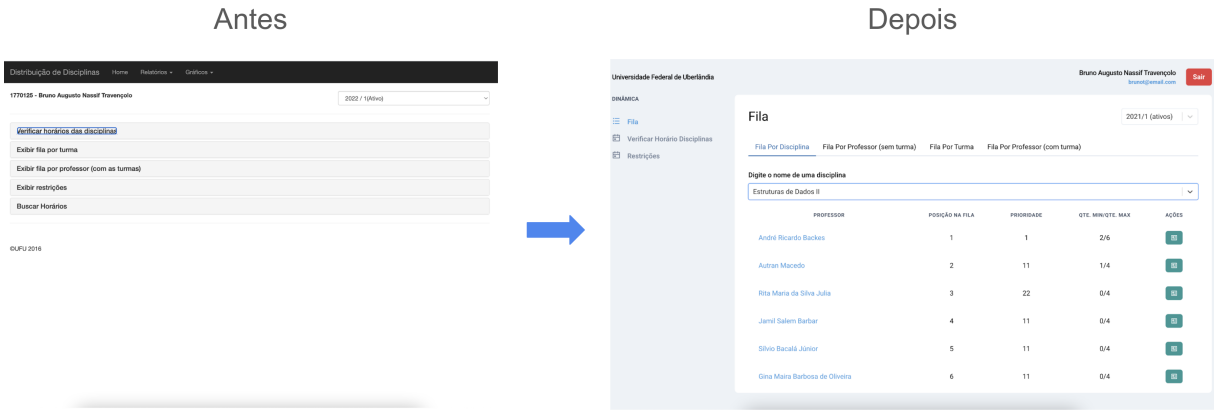

Figura 25 – Mudança visual na tela após o *login*.

6.2 Fila

Na Figura 26 é apresentada a antiga tela que os usuários com perfil professor ou admin visualizavam as filas e a sua nova versão.

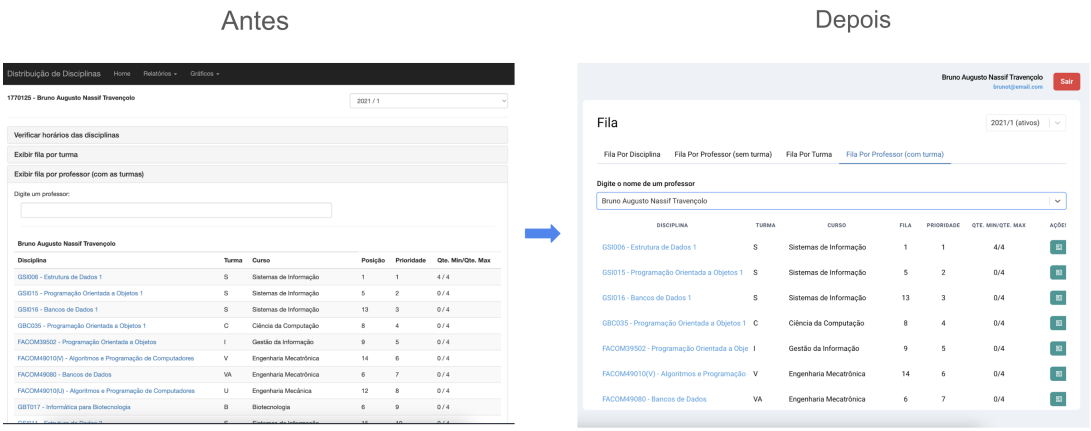

Figura 26 – Mudança visual na tela de filas.

6.3 Gerenciar cursos

Na Figura 27 é apresentada a antiga tela que os usuários com perfil admin gerenciava os cursos cadastrados no sistema e a sua nova versão.

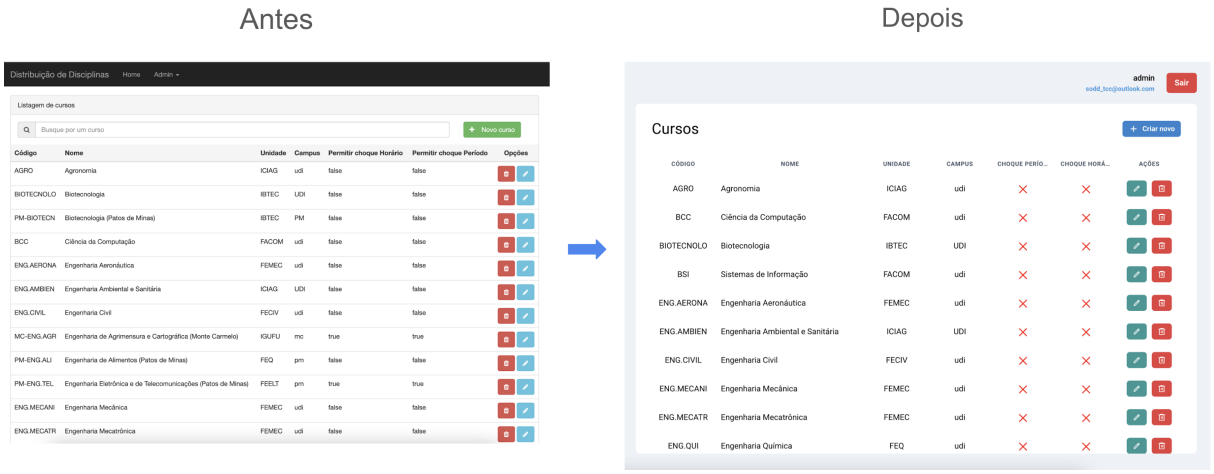

Figura 27 – Mudança visual na tela de listagem de cursos.

7 Conclusão

Com o passar do tempo e a mudança constante de pessoas que fazem manutenção em um software, é comum que padrões diferentes sejam aplicados em um mesmo software, ainda mais quando esses desenvolvedores ainda não possuem muita experiência em desenvolvimento de software. No SODD não foi diferente, com o avanço do projeto e novas funcionalidades surgindo e sendo feitas a cada semestre por alunos diferentes, o software acabou crescendo de uma forma despadronizado e alguns códigos legados que não são mais utilizados foram largados nele. Quando falamos de código legado, o banco de dados também sofreu com este tipo de problema, ficando com tabelas antigas e desatualizadas que não eram mais utilizadas. Com a nova versão do SODD, novos padrões serão aplicados e levados mais a risca, uma vez que os *frameworks* utilizados tanto no *backend* quanto no *frontend* já possuem uma padronização predefinida facilitando para os novos desenvolvedores que forem fazer manutenção no sowftware. Além de um software mais padronizado, a nova versão utiliza de tecnologias mais atuais, facilitando assim para que novos desenvolvedores façam manutenção nele, uma vez que esses novos desenvolvedores estão mais familiarizados com as tecnologias atuais. O banco de dados também passará por um processo de limpeza, eliminando assim todas as tabelas e informações que não são mais utilizadas.

O objetivo principal deste projeto foi poder disponibilizar uma versão inicial da aplicação, uma vez que o *backend* já estava com seu desenvolvimento bastante avançado, este projeto teve como foco principal criar as funcionalidades no lado do *frontend*, dando suporte para algumas novas implementações que foram necessárias no lado do *backend*.

Neste projeto, houve um grande avanço no desenvolvimento da nova versão do SODD, possibilitando assim a disponibilização de uma versão inicial, mesmo que ela ainda não possua todas as funcionalidades que o sistema atual possui. Nesta primeira versão, estará disponível para os usuários as funcionalidades básicas mais utilizadas do sistema anterior como as filas, calendário de matérias e restrição de horários. Além das funcionalidades que já existiam no software anterior que foram passadas para esta nova versão, estará disponível também algumas novas funcionalidades, funcionalidades essas voltadas para os usuários admin, onde ele poderá gerencias os usuários que acessam o sistema, além dos cursos, professores, semestres e turmas cadastradas.

A principal dificuldade encontrada no desenvolvimento desta nova versão do sistema, foi estruturar e organizar os dados no calendário da tela de horários. Para poder montar o calendário, foi utilizado uma matriz onde as linhas são os horários disponíveis para ministrar aulas e as colunas os dias da semana. Para cada combinação de horário

e dia da semana pode haver uma ou mais matéria de acordo com os filtros selecionados. Para resolver este problema, foi necessário a utilização da estrutura de dados *HashMap*, nela salvamos como chave o horário, e como valor uma matriz de matérias, assim cada posição na matriz significava um dia da semana e em cada posição está salvo as matérias que deve ser exibidas.

Referências

André Felizardo. **Angular 2 – A não continuação do Angular 1**. 2017. Disponível em: <andrefelizardo.com.br/blog/angular-2-nao-continuacao-do-angular-1>. Acesso em: 20/03/2022. Citado na página 16.

Apache Tomcat. **Apache Tomcat — Wikipédia, a enciclopédia livre**. 2021. Disponível em: <https://pt.wikipedia.org/wiki/Apache_Tomcat>. Acesso em: 28/01/2022. Citado na página 9.

DAMASCENO, M. I. R. **Manutenção do Sistema Online para Distribuição de Disciplina**. Monografia (Bacharelado em Ciência da Computação) — Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, 2021. Citado na página 16.

Felipe Nascimento. **Javascript ou Typescript?** 2021. Disponível em: <https://www.alura.com.br/artigos/javascript-ou-typescript>. Acesso em: 11/09/2022. Citado na página 9.

Guilherme Lima. **Bootstrap: O que é, Documentação, como e quando usar**. 2022. Disponível em: <https://www.alura.com.br/artigos/bootstrap>. Acesso em: 11/09/2022. Citado na página 9.

Guillermo Rauch. **Next.js — Wikipédia, a enciclopédia livre**. 2021. Disponível em: <https://pt.wikipedia.org/wiki/Next.js>. Acesso em: 11/09/2022. Citado na página 10.

Ivan de Souza. **Saiba o que é Node.js, como ele funciona e como usá-lo no seu site**. 2020. Disponível em: <https://rockcontent.com/br/blog/node-js>. Acesso em: 12/09/2022. Citado na página 9.

Kanban. **Kanban — Wikipédia, a enciclopédia livre**. 2021. Disponível em: <https://pt.wikipedia.org/wiki/Kanban>. Acesso em: 28/01/2022. Citado na página 12.

MENDES, M. dos S. **Proposta de Nova Implementação do Sistema Online de Distribuição de Disciplinas**. Monografia (Bacharelado em Sistemas de Informação) — Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, 2022. Citado na página 17.

Naélio Freires. **Um guia para usar React JS**. 2019. Disponível em: <https://blog.geekhunter.com.br/um-guia-para-usar-react-js/>. Acesso em: 11/09/2022. Citado na página 10.

Redação. **JavaScript é a linguagem mais popular da atualidade e domina aplicativos web e na nuvem**. 2020. Disponível em: <https://itforum.com.br/noticias/javascript-e-a-linguagem-mais-popular-da-atualidade-e-domina-aplicativos-web-e-na-nuvem>. Acesso em: 11/09/2022. Citado na página 9.

Scott Carey. **Como as empresas estão deixando o Cobol**. 2021. Disponível em: <https://cio.com.br/tendencias/como-as-empresas-estao-deixando-o-cobol>. Acesso em: 17/01/2022. Citado na página 9.

Vitor Zucher. **O que é padrão MVC? Entenda arquitetura de softwares!** 2020. Disponível em: <https://www.lewagon.com/pt-BR/blog/o-que-e-padrao-mvc>. Acesso em: 28/01/2022. Citado na página 13.

Wikipedia. **MVC — Wikipédia, a enciclopédia livre**. 2021. Disponível em: <https://pt.wikipedia.org/wiki/MVC>. Acesso em: 11/09/2022. Citado na página 13.

_____. **AngularJS — Wikipédia, a enciclopédia livre**. 2022. Disponível em: <https://pt.wikipedia.org/wiki/AngularJS>. Acesso em: 11/09/2022. Citado na página 9.

_____. **Trello — Wikipédia, a enciclopédia livre**. 2022. Disponível em: <https://pt.wikipedia.org/wiki/Trello>. Acesso em: 12/09/2022. Citado na página 12.