UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus José da Costa

Sistema de Equivalência de Disciplinas

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus José da Costa

Sistema de Equivalência de Disciplinas

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 Ciência da Computação.

Orientador: Prof. Dr. Renato Aparecido Pimentel da Silva

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023

Dedico este trabalho a meus pais, Vanusa Maria e José Crispim, que muito se esforçaram para que eu tivesse a oportunidade de ter um curso superior, são o alicerce para que eu tenha chegado ao fim deste ciclo de maneira íntegra.

Agradecimentos

Agradeço primeiramente a Deus por ter me dado forças e sabedoria para chegar até este ponto da graduação. E a meus pais e a meu irmão Gabriel Silva por tudo que fizeram por mim para que o diploma fosse uma realidade. Além da Andressa Oliveira e seus pais Zilda Oliveira e Wilson Bernardes, que estiveram presentes nos últimos anos da faculdade e que contribuíram muito para meus dias serem mais leves, a que me presentearam com meu animalzinho de estimação – uma gatinha linda chamada Lilo – e que tanto me auxiliou a distrair e suportar a ansiedade. Agradeço também imensamente à minha psicóloga Luciene Viana por todo o apoio nesse período, e por ter me encorajado tantas vezes em continuar o curso nos momentos mais difíceis que passei, e que inclusive possibilitou com que eu ajustasse as rotas da navegação dessa jornada na faculdade, em que tantas vezes pensei em desistir. Por fim, meus amigos – João Victor, Thiago, Leonardo, Matheus Cardoso, João Pedro, Victor, Guilherme, Pablo, Raphael, Belchior, Lalesca, Lucas, Pedro, Carlos, Vitória – e familiares – Caio e Fernanda – que estiveram comigo nesse período, vocês fizeram total diferença nos meus dias, dos mais difíceis aos mais alegres.

“Você não pode mudar o vento, mas pode ajustar as velas do barco para chegar onde quer.”, Confúcio

Resumo

As universidades de todo o Brasil, durante certas partes do ano, fazem processos seletivos, seja para o ingresso de novos alunos; ou de transferências, que podem ser tanto internas – de um curso para outro, da própria instituição – ou externas – quando um aluno é oriundo de outra instituição de ensino superior, tanto pública quanto privada. Com isso, os alunos que previamente matriculados noutro curso ou noutra universidade – e que, portanto, já cursaram disciplinas diversas – podem tentar equivalência para determinadas disciplinas do curso em que estão ingressando, por meio de uma solicitação de equivalência. No caso da Universidade Federal de Uberlândia, o aluno preenche um requerimento e preenche uma planilha onde informa, dentre outros dados, qual a carga horária e a ementa das disciplinas, tanto daquelas previamente cursadas quanto das quais deseja obter equivalência, através das anteriores. Um professor, membro do Colegiado do curso, analisa se são de fato equivalentes, em um determinado prazo estabelecido pelas Normas da Graduação. Este trabalho propõe um Sistema de Equivalência de Disciplinas para auxiliar a Coordenação no cadastro das solicitações feitas, alocando-se um professor para a análise de equivalência, estipulando um prazo máximo para a mesma. O professor por sua vez entrará no sistema e fará tal análise, que possui como forma de auxílio um parecer prévio do sistema a partir de Inteligência Artificial. Dado o registro de equivalência ou não, o mesmo é armazenado numa base de dados que posteriormente poderá ser reutilizada em novas solicitações, visando agilizar todo o processo de análise e reduzir a carga horária empenhada tanto pela Coordenação quanto pelos membros do Colegiado.

Palavras-chave: Análise de Equivalência, Graduação, Inteligência Artificial, Banco de Dados

Lista de ilustrações

Lista de abreviaturas e siglas

UFU	Universidade Federal de Uberlândia
UFMG	Universidade Federal de Minas Gerais
JWT	Json Web Token
SMTP	Simple Mail Transfer Protocol
REST	Representational State Transfer
API	Application Programming Interface
YAML	YAML Ain't Markup Language
XML	eXtensible Markup Language
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
DOM	Document Object Model
CPU	Central Processing Unit
GPT	Generative Pre-trained Transformer
AWS	Amazon Web Service
OCR	Optical Character Recognition
PDF	Portable Document Format

Sumário

1 Introdução

Com a crescente entrada de pessoas à universidade, as coordenações de cursos têm de lidar com as mais diversas situações da vida escolar dos alunos. Pode-se citar, por exemplo, jovens iniciando em determinado curso, porém que por razões diversas acabam migrando para outro curso ou até mesmo para outra instituição de ensino, assim preenchendo as vagas ociosas (VAGASOCIOSAS, 2022), e que porventura abrindo solicitação para a coordenação avaliar equivalência de disciplinas. Tal solicitação visa dispensar a matrícula em componentes curriculares com conteúdo e carga horária compatíveis com disciplinas previamente cursadas no curso/instituição de origem. Essas solicitações que alunos encaminham às coordenações acabam gerando bastante trabalho, justamente porque existem inúmeros cenários distintos na análise de equivalência.

Hoje, na UFU, quando um aluno requisita equivalência de disciplinas para a coordenação de curso, a análise é atribuída a um professor membro do Colegiado do curso, que não necessariamente tem conhecimento da ementa ou experiência com determinadas disciplinas. O mesmo então compara a ementa que o aluno trouxe – do curso em que ele concluiu aquela matéria – com a do curso que ele está atualmente cursando na UFU. Essa parte é a mais minuciosa, pois requer uma análise crítica de cada componente da ementa, com base nas normas (NORMASEQUIVALENCIA, 2022). E então é dado um parecer da análise. Se aprovado, é emitida uma certidão comprobatória da equivalência. Caso não seja aceito, o aluno então poderá recorrer em um prazo estipulado.

Essa fase de análise de ementa é a que mais demanda esforço humano, conforme o processo atualmente existente de análise de equivalência, constituído das seguintes etapas. Primeiro o aluno abre uma solicitação à coordenação, e então um membro do colegiado avalia o pedido. Nesse momento, caso algum outro aluno já tenha feito esse processo anteriormente e conseguiu equivalência, envolvendo as mesmas componentes curriculares, pode ser que o avaliador em questão se lembre desse fato e considere a análise de equivalência previamente feita. Caso contrário, então é analisado se a carga horária entre as disciplinas é compatível, além também dos conteúdos da ementa, e essa análise é feita conteúdo a conteúdo. Há casos em que os conteúdos entre as ementas divergem em relação a nomes, e então quem está analisando precisa ter mais cautela. Em caso de sucesso, então é posteriormente emitida uma certidão comprovando que o aluno conseguiu equivalência. Caso não dê equivalência, e essa seja porque na ementa que o aluno mandou – do curso de origem – falte algum conteúdo que precise constar na nova ementa do curso que quer equivalência, então o professor pode pedir que o aluno faça uma complementação de estudos para aquele determinado conteúdo. Em outros casos, o aluno também pode recorrer em um prazo estipulado pela instituição. Dessa forma, como a demanda por solicitações

de equivalência de disciplinas é muito alta, surgiu a ideia de implementar um sistema que consiga analisar se, entre duas ementas de disciplinas, essas sendo da mesma universidade ou não, possuem equivalência.

1.1 Objetivos

Este trabalho tem como objetivo geral desenvolver um sistema *web* para cadastrar tanto disciplinas de outros cursos, seja da UFU ou de outras instituições, como disciplinas do curso de referência, para análise de equivalência e dispensa de disciplinas para discentes recém-transferidos. Isso permitirá agilizar o processo de avaliação e evitar decisões conflituosas entre processos de diferentes alunos.

Como objetivos específicos, é possível enumerar:

1. Disponibilizar o sistema para secretários das coordenações de curso e professores;
2. Realizar pré-análise de equivalência a partir de Inteligência Artificial;
3. Realizar notificações por *e-mail* de análises em atraso e recibos de equivalência registrada.

1.2 Justificativa

Conforme a quantidade de vagas ociosas (VAGASOCIOSAS, 2022) na Universidade Federal de Uberlândia, pode-se perceber que muitos estudantes entram na universidade por transferência. E por entrarem por transferência, será possivelmente requisitada a equivalência de disciplinas com base nas matérias que já foram concluídas na faculdade de origem. Com isso, para cada pedido de equivalência é aberto na coordenação uma solicitação, em que é analisada por um professor membro do Colegiado do curso, sem que necessariamente o mesmo tenha domínio sobre a ementa da disciplina passível de ser dispensada. Com isso, é preciso bastante atenção e tempo para fazer a análise. Além de que, caso o pedido seja indeferido, o aluno pode entrar com recurso para re-analisar. Dessa forma, o Sistema de Equivalência de Disciplinas é bastante útil, pois com ele é possível cadastrar ementas de disciplinas e validar se entre duas disciplinas as mesmas são equivalentes, levando em consideração curso de origem e de destino. E com esse sistema também serão gravadas em uma base de dados as disciplinas que já tiveram equivalência, o que permite consultar num futuro disciplinas que já foram analisadas, agilizando desta forma a validação de um novo processo de equivalência.

1.3 Organização

Este trabalho foi desenvolvido em cinco capítulos, divididos da seguinte forma:

- Capítulo 1: traz a introdução, os objetivos e a justificativa da proposta apresentada neste trabalho;
- Capítulo 2: descreve acerca do referencial teórico, em que abordam-se as tecnologias utilizadas tanto para o desenvolvimento do *backend* quanto do *frontend*;
- Capítulo 3: aborda o desenvolvimento do sistema, em específico sua arquitetura e o banco de dados a ser usado pelo mesmo;
- Capítulo 4: aborda a interface e funcionamento do sistema, abordando o sistema de autenticação e autorização, a página de login e as áreas internas do sistema referentes aos personas de secretário e professor;
- Capítulo 5: apresenta a conclusão deste trabalho, sobre o benefício do sistema de equivalência de disciplinas para as coordenações dos cursos das faculdades, além de descrever algumas funcionalidades a serem desenvolvidas como continuação deste projeto.

2 Referencial Teórico

Neste capítulo, serão abordadas as tecnologias que o sistema utiliza, conceitos sobre o que é o modelo entidade-relacionamento, o que é um caso de uso, dentre outros.

2.1 Tecnologias

Nesta seção será descrito quais são as tecnologias escolhidas para este projeto, sobre o que é cada uma delas e a sua importância no Sistema de Equivalência de Disciplinas.

2.1.1 Java

É uma linguagem de programação orientada a objetos comumente utilizada em grandes projetos. Com Java (JAVA, 2023) é possível a criação de aplicações multiplataforma, ou seja, é possível desenvolver para os seguintes sistemas operacionais: MAC, Linux, Windows; sem precisar preocupar com a arquitetura da máquina a qual está executando a aplicação. Além disso, é possível criar aplicações WEB (servindo como *server-side*) e também *Desktop*. Neste projeto, ela foi utilizada para compor a *stack* do *backend*, em que foram desenvolvidas todas as funcionalidades de cadastro, listagem, edição, deleção, controle de usuários e até mesmo envio de *e-mail*.

2.1.2 Spring

O Spring (JOHNSON et al., 2004) foi criado por Rod Johnson para aumentar a agilidade no desenvolvimento de software. Justamente porque ele traz muitas funcionalidades de forma abstraída. Um exemplo é a criação de uma interface com Java Swing. Antes, precisava-se validar se o cliente clicou em um botão, por exemplo. Com o Spring, basta implementar a ação para o botão, sem se preocupar com o que acontecerá adiante. Além disso, têm-se injeções de dependência, em que uma dada classe se preocupa apenas em utilizar os recursos que a dependência traz ao projeto. E o interessante é que dessa forma não é necessário instanciar os objetos gerenciados pelo Spring. Neste sistema, o Spring foi utilizado para acelerar o desenvolvimento, como a criação dos *endpoints* com suas mais diversas funcionalidades, abstração de injeção de dependências e até mesmo as configurações padrões de base de dados no momento que a aplicação inicia, podendo o desenvolvedor então focar nas tarefas principais, que é desenvolver as *features* com as regras de negócio.

2.1.3 Spring Boot

Assim como o Spring, o Spring Boot (WALLS, 2015) visa facilitar o desenvolvimento de aplicações Java. Justamente porque este é um *framework* cujo objetivo é facilitar a configuração inicial de um projeto. Com o Spring Boot, abstraímos diversas configurações iniciais, como: gerenciamento de dependências, bibliotecas, validação da saúde da aplicação (*health check*), acesso à base de dados com Spring Data JPA e também uma documentação muito bem feita, que facilita bastante na resolução de problemas.

2.1.4 Spring Data

O Spring Data serve como um modelo dentro do Spring para acessar e manipular dados. É possível fazer uma conexão de forma mais fácil com bases de dados relacionais e não relacionais, além da possibilidade de implementar consultas, persistência de dados, atualização e deleção de dados de forma mais fácil. No caso deste projeto iremos utilizá-lo para manipular o banco de dados PostgreSQL. Segundo a (INFOQ, 2013), o Spring Data seria uma solução mais geral para persistência, justamente porque permite manipular os mais diversos bancos de dados, como PostgreSQL, MySQL, MongoDB dentre outros.

2.1.5 Spring Security

O Spring Security (MULARIEN, 2010) é amplamente usado em projetos Java, pois fornece um nível de configuração muito robusto em relação a quem pode acessar os recursos do sistema. Através da ferramenta, configura-se quem pode acessar certo *endpoint* a partir de *role* de usuário; se um certo perfil de usuário pode acessar recursos de busca; ou se aquele usuário pode realizar um cadastro, por exemplo. Com o pring Security, também há controle de ataques de injeção de SQL e até mesmo *Cross-Site Scripting*. Ou seja, a aplicação estará mais segura com as regras de segurança aplicadas a partir do Spring Security.

2.1.6 JWT

Na linha do Spring Security, o JWT (JONES; BRADLEY; SAKIMURA, 2015) permite trocar dados de forma segura. Ele é utilizado na construção de sistemas para gerar mais segurança no processo de autenticação. Esse *token* é assinado digitalmente, garantindo que os dados estarão íntegros na comunicação entre as partes, além de que possui uma verificação eficiente de quão autêntico é o *token*.

2.1.7 PostgreSQL

É um sistema de gerenciamento de banco de dados relacional, cujo principal destaque é a confiabilidade e escalabilidade, além de conseguir lidar com grandes volumes de

dados. Com o PostgreSQL (MOMJIAN, 2001), é possível também criar funções e procedimentos. Foi escolhido neste sistema justamente porque há um grande relacionamento de dados entre as entidades, tal como um curso, que está atrelado a uma faculdade, e de uma disciplina, que se relaciona com faculdade e curso.

2.1.8 REST APIs

Uma API REST pode ser dada como um guia de boas práticas que dita como APIs devem ser desenvolvidas, para que as aplicações se comuniquem devidamente. Essa arquitetura tem como alicerce o protocolo HTTP, e possui várias representações de recursos, como XML e JSON. Em uma API REST, os métodos mais utilizados para efetuar operações no sistema é o GET, POST, PUT, DELETE ou PATCH. Definido o método HTTP a ser utilizado, basta definir se haverá um *body, path parameters* e *headers*. Requisitado ao sistema, esse retorna um código indicando se houve sucesso ou não com a solicitação. Exemplos de código são: código 200 indicando sucesso; 201, um dado objeto foi criado com sucesso; 400, houve uma requisição com falha; 404, objeto não encontrado; e 500, uma mensagem que indica que houve um erro no sistema, geralmente causado por exceção do tipo *null-pointer* e afins. Além disso, segundo (MASSE, 2011), uma REST API é um conjunto de princípios de arquitetura de software, já uma API RESTful é a implementação das práticas do REST em uma API, seguindo suas conformidades. Dessa forma, este projeto, por exemplo, seguirá os princípios do REST e implementará APIs RESTful.

2.1.9 Microsserviços

Segundo a AWS (MICROSSERVICOS, 2023), microsserviços consistem em pequenos serviços que são independentes e que se comunicam por APIs bem definidas para gerar o sistema final. Essa abordagem de desenvolvimento de sistemas é interessante, pois o sistema poderá ser criado com base na necessidade, ou seja, em caso de necessidade de desenvolver parte do sistema em *Java*, *Python* ou qualquer outra linguagem, essas poderão se comunicar desde que obedeçam um contrato da definição das APIs. Outra vantagem é na manutenção do código, pois ao se criar pequenos serviços as classes de domínio ficam com a lógica reservada àquele escopo, o que fará com que outro microsserviço não implique diretamente, com isso se precisar fazer qualquer manutenção que seja será mais simples e possivelmente quebrará menos coisas no sistema. Neste sistema, por exemplo, o *backend* é separado do *frontend*, ou seja, ambos serviços são executados de forma independente.

2.1.10 React

O React (GACKENHEIMER, 2015) é uma biblioteca de JavaScript muito poderosa, pois ela permite a construção de interfaces para o usuário que sejam reativas e

interativas. No React, uma das abordagens mais comumente usadas é a de componentes, em que se cria um componente que seja reutilizável, isto é, pode ser utilizado em diferentes partes do *frontend*. O React DOM também contribui bastante para o desempenho, pois há uma resposta rápida a qualquer mudança no estado dos componentes do React, ou seja, o React é muito importante para garantir a eficiência na renderização e manipulação do DOM em aplicações web.

2.1.11 Typescript

O Typescript (BIERMAN; ABADI; TORGERSEN, 2014) é uma linguagem de programação muito utilizada para criar aplicações React de fácil manutenibilidade e robustez. Ela proporciona a utilização de tipagem estática ao JavaScript, ou seja, é possível especificar parâmetros de funções, tipos de variáveis e dentre outros. Na compilação do Typescript ele é convertido para JavaScript puro, com isso o código executado é puramente JavaScript.

2.1.12 NodeJS

O NodeJS (TILKOV; VINOSKI, 2010) é um ambiente de execução JavaScript, isso quer dizer que é possível criar aplicações JavaScript que rodarão em uma máquina e que não dependerá do navegador para rodar. Para o caso deste projeto, o NodeJS é necessário a criação da *engine* da aplicação React no *frontend*. Do NodeJS será utilizado o gerenciador de pacotes, o npm. O npm é similar ao maven, que é muito usado em aplicações Java.

2.1.13 Docker

O *Docker* (ANDERSON, 2015) é uma ferramenta que possibilita a criação e administração de ambientes isolados. Com a ferramenta, é possível empacotar uma aplicação ou ambiente em um *container*, e qualquer ambiente que tenha o Docker instalado conseguirá desempacotar e executar o que há por dentro. Por exemplo, é possível subir um banco de dados relacional PostgreSQL a partir de uma imagem Docker, e então utilizá-lo num sistema que ainda se encontre em desenvolvimento. A vantagem é que não é necessário contratar um banco de dados enquanto a aplicação continua em estágio inicial. Ou seja, ganha-se velocidade no desenvolvimento. Além disso, para se fazer *build* e *deploy* da aplicação, o Docker é uma ferramenta essencial, justamente por realizar tão bem o empacotamento da aplicação.

2.1.14 Docker Compose

O Docker Compose (REIS et al., 2021) é uma ferramenta que permite rodar múltiplas instâncias Docker a serem executadas multi plataforma de forma virtualizada. Define-se em um arquivo de configuração no formato YAML os serviços que serão criados como *containers*, especifica-se a imagem que cada *container* usará, qual porta ele escutará e também fará exposição e qual será a rede utilizada. Acelera-se o desenvolvimento de software principalmente em sistemas que se encontram em desenvolvimento inicial, pois se permite a criação de instâncias que um sistema depende de maneira rápida, como é o caso, por exemplo, da base de dados PostgreSQL que foi definida neste projeto com Docker Compose, sem a necessidade de se adquirir uma base de dados em algum serviço de nuvem ou até mesmo instalar o programa na própria máquina. Criar um *container* Docker possibilita a execução da aplicação na máquina do desenvolvedor de maneira mais otimizada, pois, por atuar como uma espécie de máquina virtual, é até mesmo possível definir-se quanto de memória e CPU aquele *container* utilizará.

2.1.15 Padrão de Projeto *Command*

O padrão *Command*[1] é um padrão de projeto comportamental em que é solicitada uma ação (requisição), há uma execução, e então uma resposta. Pode ser exemplificado mais facilmente pensando-se em, por exemplo, um restaurante: um garçom vai até uma mesa e anota um pedido do cliente. Então, o garçom leva o pedido até a cozinha, que executa o pedido. Posteriormente, é entregue ao cliente o prato. Esse padrão é muito utilizado quando se pensa em desacoplar quem chama a operação de quem sabe como executá-la. Desta forma, tem-se mais flexibilidade no desenvolvimento do sistema. Um exemplo claro de uso de um sistema criado com padrão de Comandos é o de salvar um registro na base de dados. Na criação do comando, passam-se os dados para a criação do registro. Após a criação do comando, é executado o comando e então o receptor (*receiver*) é quem faz a operação, ou seja, o *receiver* que detém a informação de qual tipo de base de dados ele irá integrar, por exemplo, MySQL. Com isso, caso seja necessário em algum momento trocar da base de dados MySQL para outra, como o PostgreSQL, bastará mudar a implementação do *receiver*, não precisando alterar o comando propriamente dito. Além disso, com o padrão de comandos há uma grande reutilização de código, em que para diferentes casos de uso pode-se utilizar um comando já criado, como, por exemplo, para buscar um usuário pelo *e-mail* em diferentes fluxos.

[1] Disponível em: <https://refactoring.guru/pt-br/design-patterns/command>, acesso em nov. 2023

2.1.16 Protocolo SMTP

O protocolo é SMTP[2] é comumente utilizado para enviar *e-mails* pela Internet. Com ele, define-se como será enviado um *e-mail*, o remetente e o destinatário. O remetente envia o *e-mail* para um servidor SMTP, e o servidor envia o mesmo para outro servidor SMTP, do destinatário, e então a mensagem é entregue. Basicamente, para esse envio as etapas são: estabelecer conexão com o servidor SMTP do remetente, envia-se para o servidor SMTP os dados de remetente, destinatário, conteúdo do *e-mail* e então se envia para o servidor SMTP do destinatário, que aceita e encaminha a mensagem à caixa de entrada do destinatário.

2.1.17 Thymeleaf

Para o envio de *e-mail* via protocolo SMTP em Java é possível criar um *template* HTML para deixar o conteúdo do *e-mail* mais amigável para o usuário, e para isso é utilizado o Thymeleaf[3]. Esse mecanismo de *template* é muito utilizado para escrever aplicações com Java e Spring, o Thymeleaf tem suporte nativo do Spring, o que facilita então a visualização dos dados em páginas web.

2.1.18 Open AI

O Open AI[4] é um laboratório de pesquisa em inteligência artificial (IA) que se propõe a entregar tecnologias de IA para a comunidade, como, por exemplo, o GPT. O GPT, como o caso do GPT-3 ou GPT-4, são modelos de linguagem pré-treinados, que podem gerar textos coesos e relevantes. O Open AI disponibilizou uma API para que as pessoas façam integração, em que basta configurar a chave que dá acesso à conta da pessoa ou organização, o modelo que será utilizado e a pergunta que quer ser feita para o GPT.

2.2 Modelagem de Dados e Casos de Uso

Conforme destacado por (PRESSMAN, 2014), uma boa modelagem de dados permite com que uma dada organização chegue em seus objetivos aproveitando ao máximo os seus dados, conseguindo suprir todas as demandas de negócio. Sendo assim, um modelo de dados em suma compõe as regras de negócio do sistema. E é necessária uma boa modelagem de dados para que o sistema se molde e se perpetue no tempo sem muitas complicações. No caso deste sistema, optou-se pela modelagem de dados relacional.

[2] Disponível em: <https://www.baeldung.com/java-email>, acesso em nov. 2023
[3] Disponível em: <https://www.thymeleaf.org/>, acesso em nov. 2023
[4] Disponível em: <https://openai.com/>, acesso em nov. 2023

Paralelamente, em relação aos conceitos de caso de uso como descritos por (JACOBSON et al., 1992), um caso de uso pode ser dito como uma funcionalidade que o sistema deve ter, comportamentos importantes do sistema. Por exemplo, nesse sistema teremos um cadastro de disciplinas. A *API* que será criada para fazer esse cadastro pode ser dada como um caso de uso. Geralmente, para se representar um caso de uso, são utilizados diagramas de sequência para mostrar como se dará a jornada do usuário, desde a sua chamada no *frontend* até os passos que o *backend* fará, seja na base de dados ou até mesmo através da integração com outro sistema.

Como exemplo, pode-se ver na Figura 1 a ilustração de um diagrama de sequência para o caso de uso do cadastro de disciplinas no sistema de equivalência.

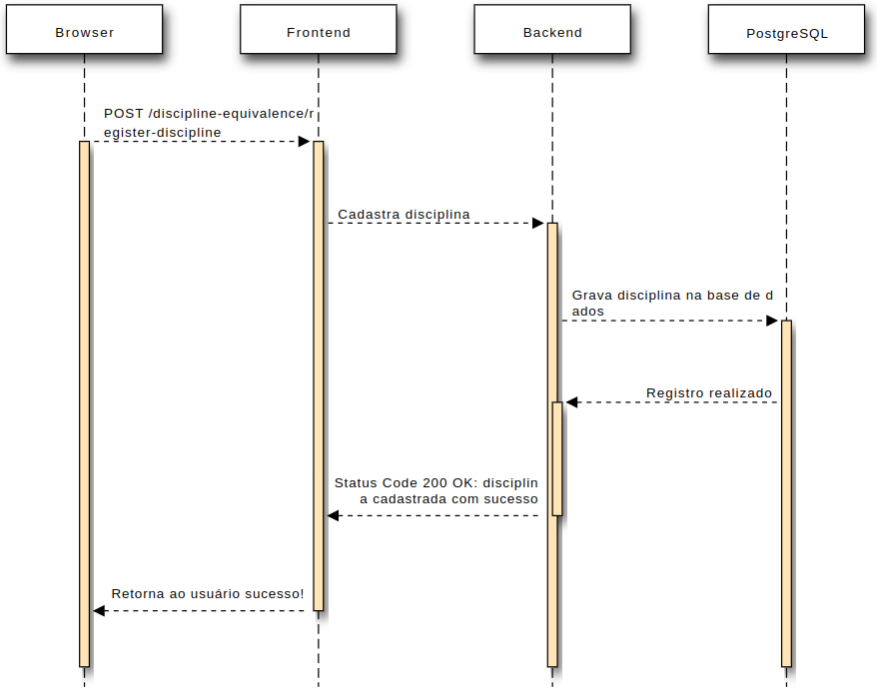

Figura 1 – Diagrama de sequência do cadastro de disciplinas. Fonte: o próprio autor.

3 Desenvolvimento

O sistema de equivalência de disciplinas, como dito anteriormente, é um sistema que visa auxiliar as coordenações dos cursos das faculdades em relação ao esforço gasto pelos professores nas análises de equivalência, em que muitas vezes não se tem tanto controle do que já foi ou não analisado, além do aspecto subjetivo quando se considera diferentes pessoas fazendo análises muitas vezes dos mesmos conteúdos.

Nesse contexto, pode-se citar um cenário que acontece muito de pedido de análise de equivalência, que é um aluno que fazia parte de outra universidade como, por exemplo, a UFMG, no curso de Matemática, por exemplo. Esse aluno resolve transferir-se para a Universidade Federal de Uberlândia, para o curso de Engenharia Civil. Quando esse aluno estudava Matemática na UFMG, ele cursou Cálculo I. E na migração para a UFU, ele requisitou na coordenação do curso de Engenharia Civil uma análise de equivalência. A secretaria preparou a solicitação, incluindo as ementas e outras informações como a carga horária de ambas as disciplinas – preenchidas pelo próprio requerente – e encaminhou a um professor membro do Colegiado, para realização da análise e possível dispensa de disciplina.

Essa análise é feita de forma manual, em que se analisam ambas as ementas, bem como a carga horária, e se os requisitos para equivalência estiverem conforme estipulados pelas normas da graduação (NORMASEQUIVALENCIA, 2022) então a equivalência é assegurada. É importante ressaltar que, em geral, não se tem controle do que já foi ou não analisado, podendo ocasionalmente ocorrer uma análise de equivalência envolvendo pares repetidos de disciplinas. Com isso, o sistema de análise de equivalências auxiliará no controle do que já foi ou não analisado.

3.1 Arquitetura

A arquitetura do sistema foi pensada com o *frontend* sendo separado do *backend*, veja a arquitetura de microsserviços, como mostra a Figura 2.

Figura 2 – Modelo da arquitetura do sistema. Fonte: o próprio autor.

Além disso, buscou-se ser o mais reutilizável possível em termos de escrita de código, para assim ganhar mais velocidade no desenvolvimento. Com isso, o *backend* foi escrito no padrão de projeto *Command*, em que é passada a requisição a ser feita, e então o receptor executa esse comando com a lógica de negócio em questão. Outro padrão utilizado é o *Chain of Responsibility*, que permite a criação de um caso de uso utilizando comandos, ou seja, uma composição de comandos para descrever uma funcionalidade do sistema. A Figura 3 mostra sobre o funcionamento do padrão *Command*.

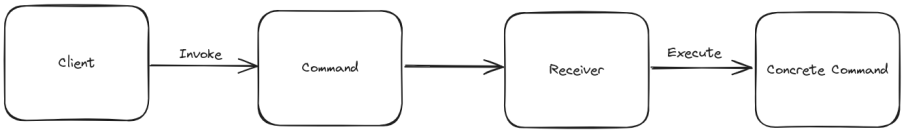

Figura 3 – Diagrama de funcionamento do padrão *Command*. Fonte: o próprio autor.

Em termos práticos de código, na Figura 4 segue uma exemplificação de uma chamada de um comando chamado *FindCourseById* em um *Controller* escrito em linguagem Java. Esse comando inclusive é bastante reutilizado, sendo adotado em outras partes do sistema, como no registro de disciplinas, no cadastro de análise de equivalência e no cadastro de professor.

```
no usages  matheuscosta1-
@Operation(summary = "Find course by college id", description = "Find course by college id")
@DocApiResponsesError
@GetMapping("cursos/{id}")
@PreAuthorize("hasAnyRole('ROLE_ADMIN')")
public ResponseEntity<CourseResponse> findCourseById(@PathVariable(value = "id") Integer id) {

  CourseDocument courseDocument =
      commandGateway.invoke(
          FindCourseById.class, FindCourseById.Request.builder().cursoId(id).build());

  return ResponseEntity.ok(courseDocumentMapper.map(courseDocument));
}
```

Figura 4 – Chamada de um comando de busca de curso pelo id em um *Controller*. Fonte: o próprio autor.

Esse comando tem a seguinte definição de requisição, como mostra a Figura 5, em que basicamente necessita-se do id do curso para ser executado, e a resposta desse comando é uma classe *CourseDocument* em que contém as informações referentes a entidade de curso.

```java
package br.com.tcc.project.command;

import ...

24 usages  matheuscosta1-
@GenerateCommandFactory
public class FindCourseById extends AbstractCommand<FindCourseById.Request, CourseDocument> {

  matheuscosta1-
  @Setter
  @Getter
  @Builder
  public static class Request {
    private Integer cursoId;
  }
}
```

Figura 5 – Comando de busca de curso pelo id. Fonte: o próprio autor.

Além disso, a execução propriamente dita do comando é feita da seguinte forma, como mostra a Figura 6, em que se integra com a base de dados relacional PostgreSQL e busca-se o curso pelo id, e caso encontrado é retornado o documento e caso não encontrado retorna uma exceção.

```java
package br.com.tcc.project.command.discipline.receiver;

import ...

no usages  matheuscosta1-
@CommandReceiver(FindCourseById.class)
public class FindCourseByIdReceiver
    extends AbstractReceiver<FindCourseById.Request, CourseDocument> {

  @Autowired @Setter private CourseRepository courseRepository;
  no usages
  private final CourseDocumentMapper mapper = Mappers.getMapper(CourseDocumentMapper.class);

  matheuscosta1-
  @Override
  protected CourseDocument doExecute(FindCourseById.Request parameter) {

    return courseRepository
        .findById(parameter.getCursoId())
        .orElseThrow(
            () ->
                new CollegeNotFoundException(
                    MessageFormat.format(
                        DisciplineEquivalenceErrors.DEE0003.message(), parameter.getCursoId()),
                    DisciplineEquivalenceErrors.DEE0003.name(),
                    DisciplineEquivalenceErrors.DEE0003.group()));
  }
}
```

Figura 6 – Implementação do receptor do comando de busca de curso pelo id. Fonte: o próprio autor.

É importante ressaltar que com essa definição de arquitetura, caso haja necessidade de mudança da base de dados, por exemplo, a mudança é feita apenas no *receiver*, alterando-o para integrar com outra base de dados, como, por exemplo, o MongoDB.

Como exemplo de um caso de uso utilizando uma cadeia de comandos, pode-se ver pelo código escrito em Java da Figura 7, em que define-se a requisição do comando principal chamado *DisciplineEquivalenceReport*, e dentro desse comando chamam-se outros comandos, como o comando para busca de disciplina por id – *FindDisciplineById*.

```java
@GenerateCommandFactory
public class DisciplineEquivalenceReport
    extends AbstractCommandChain<
        DisciplineEquivalenceReport.Request, EquivalenceDisciplineResponse> {

  matheuscosta1 +1
  @Setter
  @Getter
  @Builder
  public static class Request {
    private Integer idDisciplinaOrigem;
    private Integer idDisciplinaDestino;
  }
  2 usages
  private final DisciplineDocumentMapper mapper = Mappers.getMapper(DisciplineDocumentMapper.class);

  3 usages  matheuscosta1 +2
  @Override
  public void execute() throws IllegalStateException {
    super.execute();

    AnaliseEquivalenciaDisciplineResponse originDisciplineResponse = mapper.mapForResumo(invoke(
            FindDisciplineById.class,
            FindDisciplineById.Request.builder()
                    .id(getParameter().getIdDisciplinaOrigem())
                    .build()));

    AnaliseEquivalenciaDisciplineResponse destinyDisciplineResponse = mapper.mapForResumo(invoke(
            FindDisciplineById.class,
            FindDisciplineById.Request.builder()
                    .id(getParameter().getIdDisciplinaDestino())
                    .build()));
```

Figura 7 – Exemplo de caso de uso para gerar o relatório das análises de equivalência utilizando-se cadeia de comandos. Fonte: o próprio autor.

Essa abordagem é muito interessante, pois se define toda a regra de negócio de uma funcionalidade em um único lugar, que é esse comando principal que invoca os demais para realizar partes do fluxo.

3.2 Banco de dados

Para que este sistema seja funcional, faz-se necessária a modelagem da base de dados relacional entre as entidades, antes mesmo de se iniciar o desenvolvimento do *backend*. Com isso, o sistema consiste nas seguintes entidades principais: faculdade, curso, disciplina, professor, as análises e equivalências.

Uma faculdade não tem dependência de nenhuma outra entidade, mas ela é essencial para que um curso exista, ou seja, o curso está relacionado a uma faculdade. Da mesma forma, uma disciplina relaciona-se com um curso, e este, como dito, está relacionado a uma faculdade. Dado o entendimento das entidades faculdade, curso e disciplina, tem-se a base para a entidade professor e análises.

Um professor é quem leciona uma disciplina. Logo, está relacionado com a entidade disciplina. Esse professor também se relaciona com uma entidade chamada usuário, pois o professor também é um usuário do sistema. Além disso, a entidade usuário é necessária para armazenar os usuários do sistema, seja de professor ou secretário, e para isso um usuário pode ter vários perfis a si associados, então relaciona-se com a entidade perfil.

Além disso, a entidade análises é uma das principais do sistema. Contém as informações de qual professor fará a análise de equivalência e também as informações acerca de quais disciplinas serão analisadas, ou seja, essa entidade relaciona-se com as disciplinas, professor e até mesmo com a tabela de usuário, para saber qual foi o secretário que criou aquela análise – essa informação é necessária para posteriormente notificar o secretário de que uma análise foi feita.

Dessa forma, com uma análise, o professor pode realizá-la e isso é salvo na entidade equivalência. Essa tabela é onde ficam salvos todos os registros de equivalência ou não-equivalência entre duas disciplinas, e há um relacionamento com a entidade de análise, para saber posteriormente de onde essa equivalência foi registrada e por quem foi registrada – armazena-se o identificador do professor, de modo que é possível saber quem fez a análise.

Além disso, há algumas outras entidades secundárias necessárias para o funcionamento do sistema, que são: entidade de notificação, análise de equivalência pelo Open AI e de modificação de ementa. Essa primeira entidade de notificação é a responsável pelas notificações de análises em atraso e também de equivalências realizadas, logo a mesma possui relacionamento com a tabela de equivalência e também de análises. Já a tabela de análise de equivalência pelo Open AI se relaciona com as disciplinas, pois as informações das disciplinas como ementa e carga horária são necessárias para realizar uma análise sistemática se as ementas são equivalentes ou não. E a tabela de modificação da ementa de alguma disciplina relaciona também com uma disciplina, e basicamente ela é utilizada para controlar as modificações feitas em alguma disciplina, pois se isso acontece, então uma análise de equivalência entre duas disciplinas deve ser possível de ser reanalisada. Pela Figura 8 abaixo é possível ver o relacionamento completo entre as entidades do sistema que foram descritas acima.

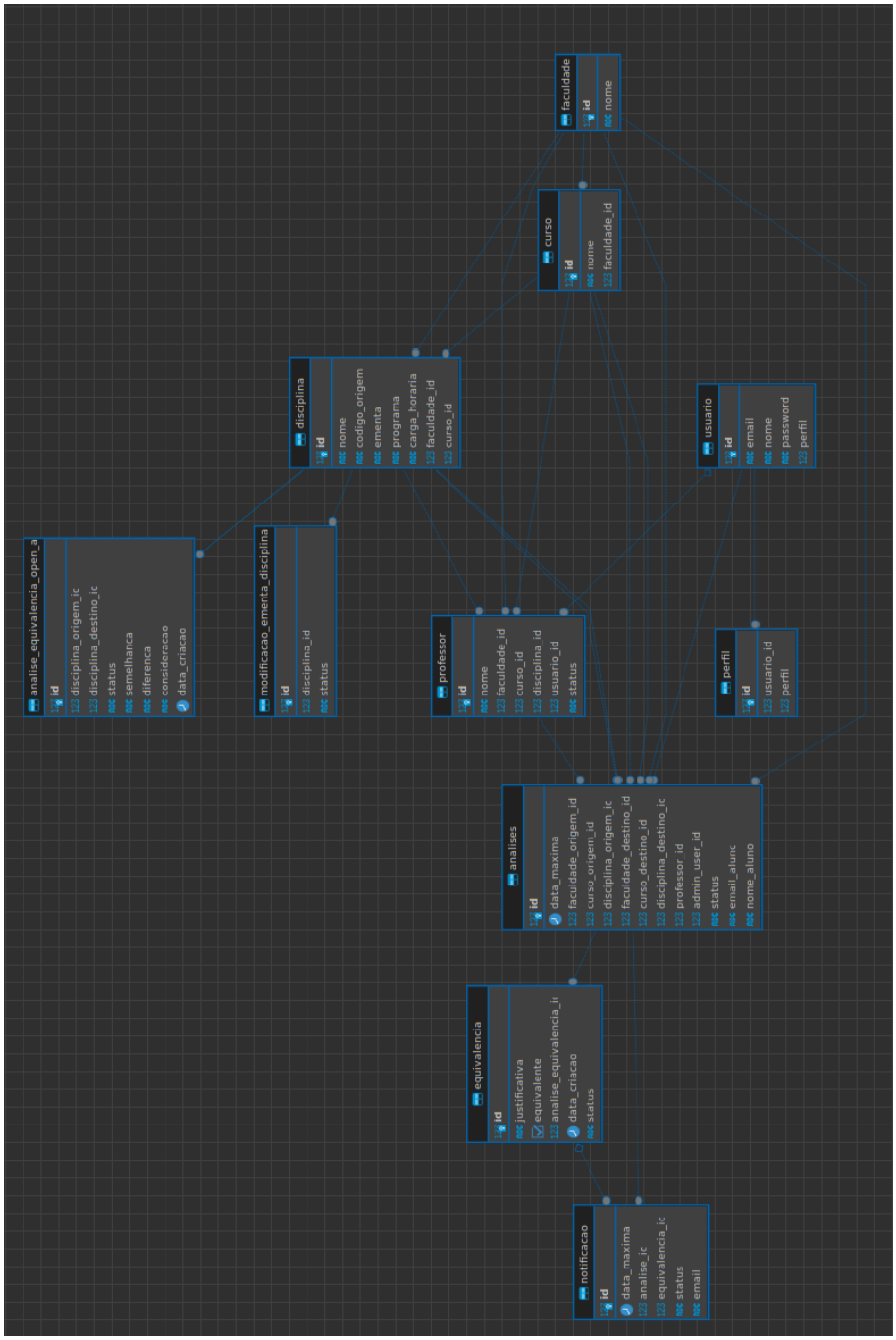

Figura 8 – Diagrama entidade-relacionamento completo do sistema de equivalência de disciplinas. Fonte: o próprio autor.

4 Apresentação e exemplos

Neste capítulo, será mostrada de forma geral a interface do sistema. Também serão ilustrados alguns exemplos de casos de uso ao longo do texto. E esse projeto contempla dois principais Personas, que são: secretário e professor. O secretário é quem vai fazer todo o cadastro básico no sistema, como o cadastro de faculdades, cursos, disciplinas, professores e então as análises de equivalência em que se aloca um professor para analisar. O professor, por sua vez, acessará o sistema para fazer as análises de equivalência que foram a ele alocadas, e que estejam pendentes.

Com base nesses personas descritos, definiu-se que o sistema funcionaria com base em duas jornadas diferentes: a jornada do secretário e a jornada do professor. E para isso foi desenvolvido no *backend* escrito em Java com Spring e padrão de projeto Comandos um controle de autenticação e autorização com *JWT*, além das telas no *frontend* com TypeScript e React.

Importante destacar que hoje o sistema está sendo executado localmente, em que é necessário ter *Java* e o gerenciador de dependências *Maven* na versão 17 para executar o *backend*. Já para executar o *frontend* é necessário ter o *NodeJS* e o gerenciador de dependências *npm*. Todas essas informações estão contidas no *README.md* de cada um dos projetos no *GitHub*[1]. E o código-fonte do *backend* pode ser encontrado em: <https://github.com/matheuscosta1/discipline-equivalence>. E o do *frontend* em: <https://github.com/matheuscosta1/front-discipline-equivalence>.

Será descrito a seguir na seção 4.1 o processo de autenticação e autorização do sistema de equivalências de disciplinas. Em que será abordado o processo de autenticação a partir da geração de um token JWT e a sua utilização para autorizar o acesso aos recursos do sistema.

4.1 Autenticação e autorização

O secretário para o sistema é como se fosse um super-administrador, pois é o mesmo quem possuirá o primeiro acesso à plataforma, ou seja, ele é um usuário do sistema em que o seu tipo de permissão é de super usuário (*ROLE_ADMIN*). O professor, por sua vez, é um usuário que possui permissão apenas de professor (*ROLE_PROFESSOR*). Dito isso, o sistema está preparado para que o usuário tenha mais de um perfil, a implementação do *backend* ficou genérica nesse sentido, porém para o sistema de análises de equivalência é previsto que o usuário professor tenha apenas um tipo de perfil, assim como o secretário.

[1] Disponível em: <https://github.com/>, acesso em dez. 2023

Não podendo assim o professor fazer cadastros que o secretário faz, nem o secretário fazer análises de equivalência que diz respeito ao professor realizar. Na Figura 9 ilustra-se como ficou o diagrama entidade-relacionamento do controle de usuário.

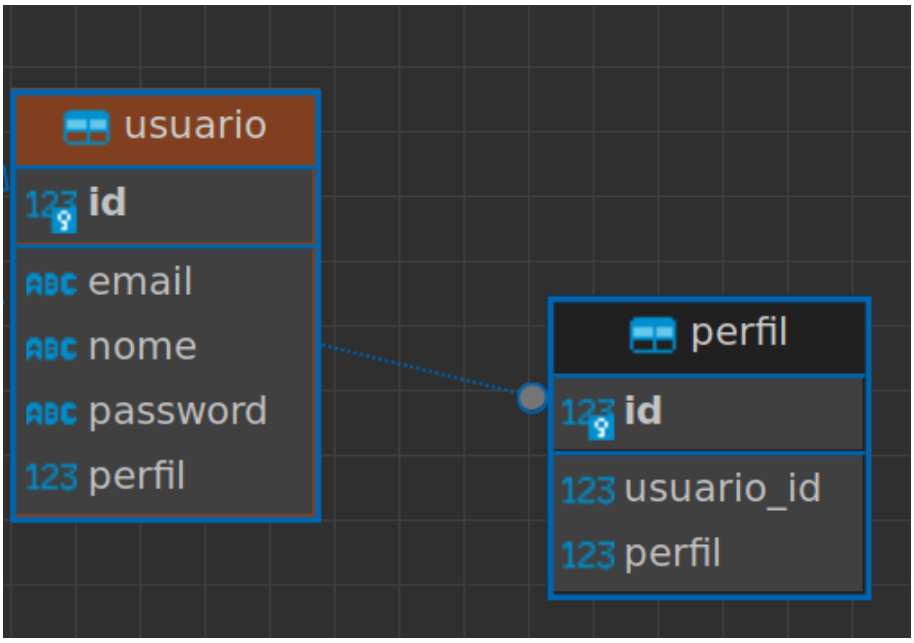

Figura 9 – Diagrama entidade-relacionamento do controle de usuário. Fonte: o próprio autor.

Importante dizer que para que o *frontend* realize qualquer chamada no *backend*, é necessário fazer a autenticação com *e-mail* e senha em um *endpoint* chamado /login, em que será retornado um *token* JWT, como segue na Figura 10.

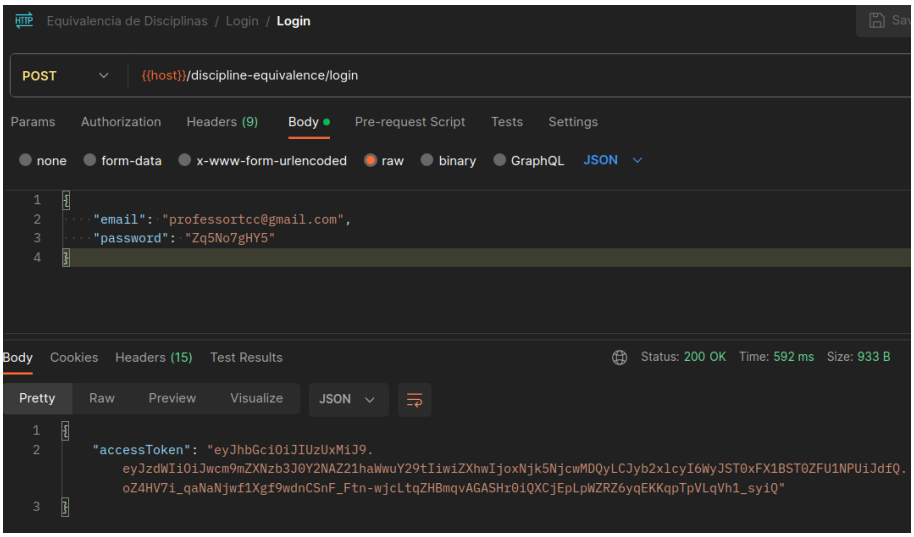

Figura 10 – Geração de token usando e-mail e senha. Fonte: o próprio autor.

Ao abrir esse *token*[2], observa-se pela Figura 11 que no *token* há informações como a *role* de usuário e o *e-mail* da pessoa que fez a autenticação.

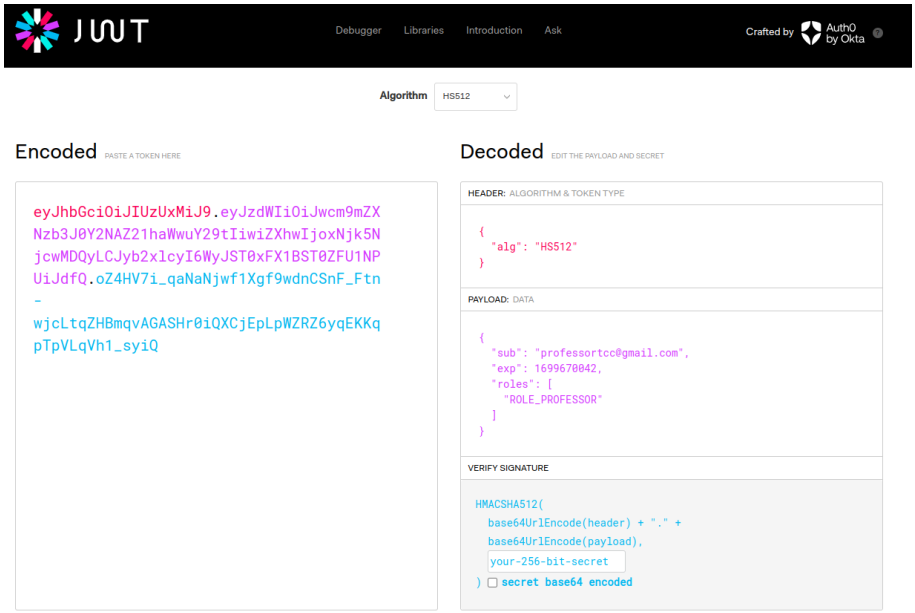

Figura 11 – Token JWT aberto. Fonte: <https://jwt.io/>.

Essas informações de *e-mail* e *role* de usuário são muito importantes para que o *frontend* consiga renderizar as telas específicas para os usuários e inclusive deixar cada usuário acessar apenas as rotas que são referentes ao seu tipo de permissão.

Esse controle do que cada usuário pode acessar foi feito tanto no *frontend* – separando as rotas por tipo de usuário – quanto no *backend*, utilizando *Spring Security*, em que para cada *endpoint* é informado qual *role* de usuário pode autorizar, e caso alguém tente autorizar sem a *role* correta é retornado para aquele usuário que ele não tem acesso àquele recurso, ou seja, um erro com *status code* 403.

Outro aspecto é que todo professor cadastrado na plataforma automaticamente se torna um usuário do sistema, cuja *role* é de professor. O sistema gera uma senha automática para o mesmo, e posteriormente quando o secretário o alocar para fazer uma análise de equivalência ele então já terá acesso para entrar no sistema, em que poderá pedir uma re-definição de senha a partir da tela de login, ou até mesmo trocar os seus dados cadastrais – incluindo a sua senha que recebeu por *e-mail* – após feito o login. No próximo tópico, será abordada a tela de login, redefinição de senha e também a mudança dos dados cadastrais a partir da área logada do sistema.

[2] Disponível em: <https://jwt.io/>, acesso em nov. 2023

4.2 Login

Agora com o conhecimento sobre as personas de secretário e professor, e também sobre o funcionamento do sistema de autenticação e autorização, é possível ilustrar a tela de Login, desenhada como mostra a Figura 12.

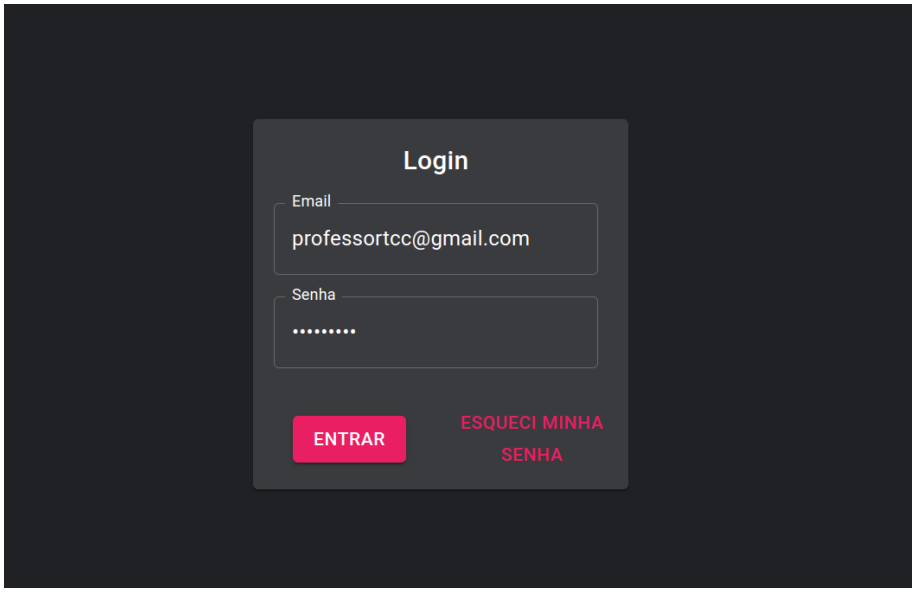

Figura 12 – Tela de login. Fonte: o próprio autor.

Ao informar *e-mail* e senha, secretário ou professor são redirecionados para suas respectivas jornadas. E caso o usuário que está fazendo login tenha esquecido sua senha, então ele pode pedir uma redefinição de senha, como mostra a Figura 13.

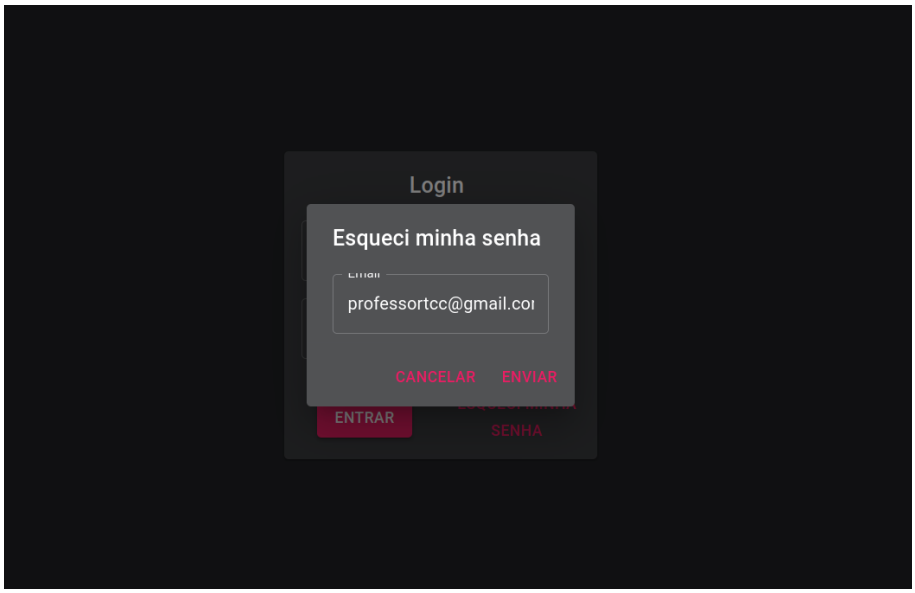

Figura 13 – Tela de esqueci minha senha. Fonte: o próprio autor.

Para que a pessoa peça a redefinição de senha, a mesma deve informar o seu *e-mail*, e então o sistema envia para o mesmo uma nova senha gerada automaticamente, conforme a Figura 14.

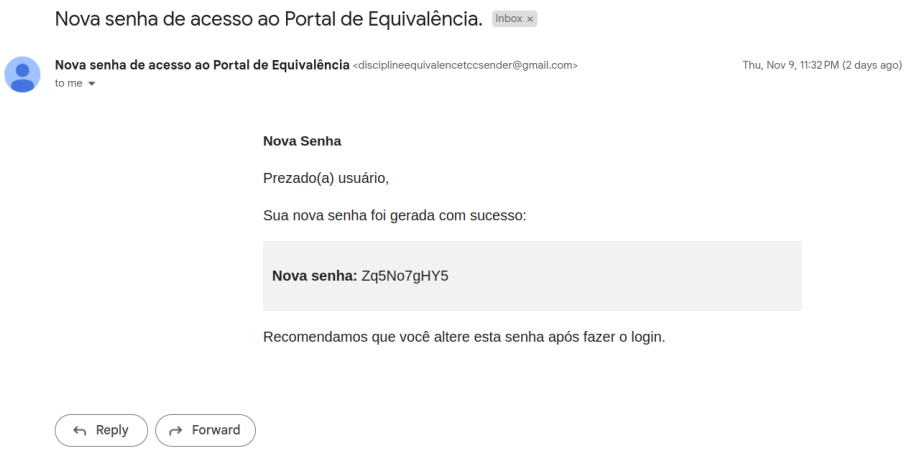

Figura 14 – E-mail com a redefinição de senha. Fonte: o próprio autor.

Com a senha em mãos e feito o login, então o *frontend* diversifica as telas com base no tipo de token, se é professor ou secretário, e que será descrito a seguir, está dividido entre a jornada de secretário e a de professor.

4.3 Secretário

O persona de secretário é o responsável por fazer o cadastro das faculdades, cursos, disciplinas, professores e da principal funcionalidade do sistema: designar uma análise de equivalência a um professor. A seguir será detalhada toda a jornada do secretário dentro do sistema de análise de equivalências.

4.3.1 Página Inicial

Ao fazer login com perfil de secretário, a primeira coisa a ser mostrada no sistema, na página inicial, são as análises de equivalência pendentes, ou seja, tudo que o secretário cadastrou para os professores, onde se mostra o nome do professor que fará a análise, as disciplinas do pedido de equivalência e a data limite para análise. Na Figura 15, além do que foi dito, percebe-se que é possível filtrar pela data máxima e também filtrar os registros pelo nome do professor. Outro aspecto é que quanto mais próximo da data limite para ser realizada a análise, mais se pende para o tom vermelho. Isso foi feito para ficar mais chamativo para o secretário do curso o que está prestes a expirar.

Figura 15 – Página inicial da área logada do perfil de secretário. Fonte: o próprio autor.

A partir da tela inicial é possível também criar uma nova análise de equivalência. Essa análise de equivalência é onde se aloca um professor para avaliar duas ementas de disciplinas, será posteriormente explicado com mais detalhes sobre essa funcionalidade, pois antes será mostrado o funcionamento do cadastro de faculdades, cursos, disciplinas e professores.

4.3.2 Faculdades, cursos e disciplinas

Outro aspecto bem importante para o secretário do curso é o de visualizar, criar e editar faculdades, cursos e disciplinas. Essa é uma etapa bem importante para o sistema, pois para o cadastro de um curso é necessário vincular o curso a uma faculdade pré-cadastrada, e o mesmo ocorre com a disciplina, onde é necessário ter o curso e a faculdade cadastrada. Em seguida, com uma disciplina cadastrada, então é possível fazer o cadastro de um professor e da tarefa principal do sistema: alocar uma análise de equivalência feita entre duas disciplinas a um professor.

A seguir será exemplificado um caso de listagem, criação e edição de disciplina. E na listagem das disciplinas – veja a Figura 16 – tanto como de faculdades e cursos, foi criado um sistema de paginação, devido ao grande número de registros que pode existir no sistema.

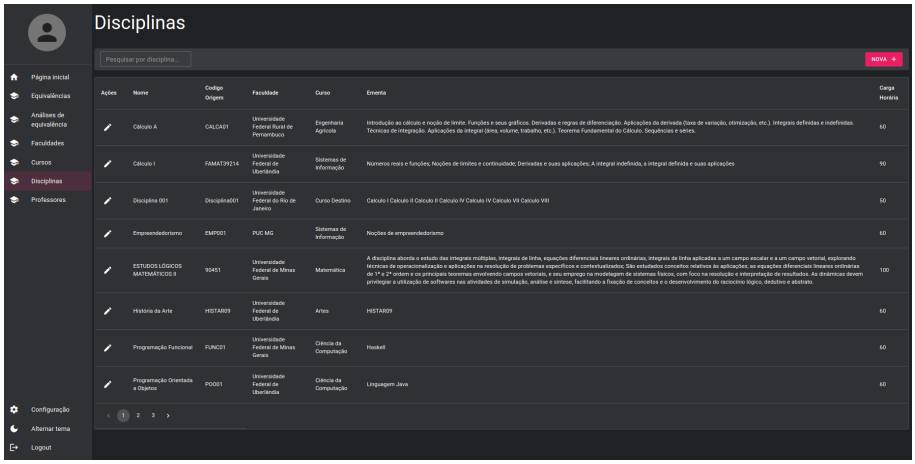

Figura 16 – Listagem de disciplinas. Fonte: o próprio autor.

Além disso, conforme a tela em que o usuário esteja – se tela de disciplina, curso ou faculdade – é possível filtrar pelo nome ou alguma palavra que filtre mais de um registro, como, por exemplo, filtrar todas as disciplinas que comecem com Programação, como mostra a Figura 17.

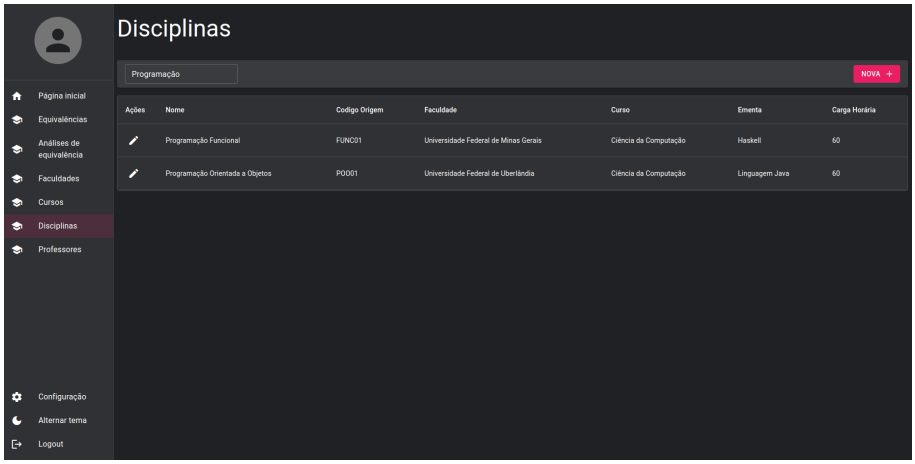

Figura 17 – Listagem de disciplinas com filtro. Fonte: o próprio autor.

Para que esse filtro funcione, no *backend* foi feita uma implementação para que se consulte a base de dados caso tenha sido passada alguma informação no campo de busca, e então são filtrados todos os registros que contenham aquela palavra digitada. Ou seja, o *frontend* chama esse *endpoint* de busca passando o campo digitado na busca. Além disso, todas as funcionalidades de listagem do *backend* estão paginadas, e o *frontend* lida com isso, mostrando de forma paginada para o usuário.

Além da funcionalidade de listagem, também pode-se ilustrar as telas de criação e edição de registros. Para a criação de registro, ao clicar em NOVA+ a partir da tela de listagem, o usuário é encaminhado para a tela de cadastro. Nesta tela, adicionam-se as informações referentes ao que se deseja cadastrar, como mostra a Figura 18.

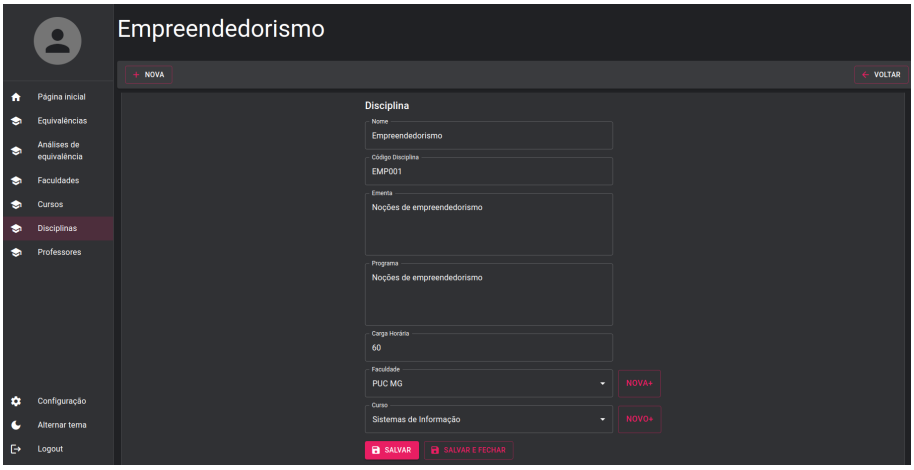

Figura 18 – Cadastro de disciplina. Fonte: o próprio autor.

Para toda tela de cadastro do sistema, quando o usuário clica em salvar, é chamado um *endpoint* do *backend* a partir do *frontend* e salvo na base de dados PostgreSQL. Além disso, é retornado um *feedback* para o usuário do que aconteceu com a operação, isto é, se se obteve sucesso ao salvar ou não. Na Figura 19 ilustra-se a modal de *feedback* de sucesso.

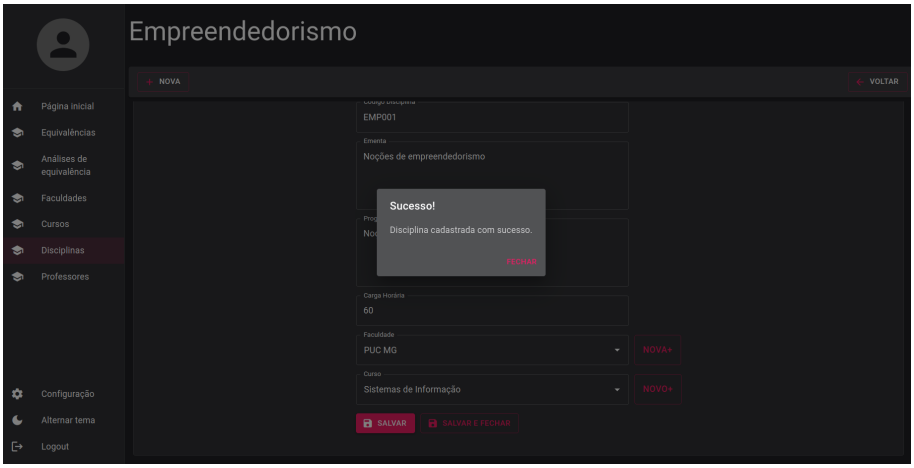

Figura 19 – Modal de *feedback* de sucesso para o usuário no cadastro de disciplina. Fonte: o próprio autor.

Também é retornado um *feedback* de erro para o usuário caso, por exemplo, aquele registro com aquele código ou nome já exista no sistema, como mostra a Figura 20.

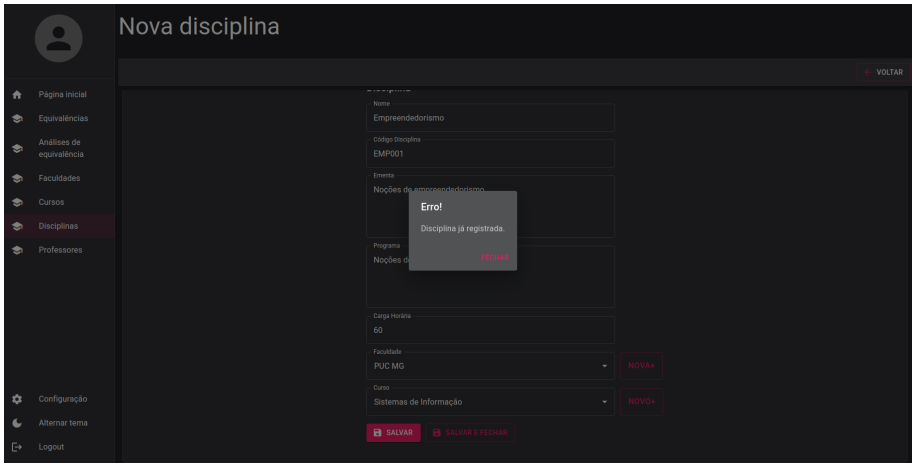

Figura 20 – Modal de *feedback* de erro para o usuário no cadastro de disciplina. Fonte: o próprio autor.

Importante ressaltar que todos esses exemplos ilustrados para a tela de disciplinas segue o mesmo padrão e consistência de informações para as telas de curso e faculdade.

4.3.3 Professores

O professor, para o sistema, é um usuário fundamental, pois é ele quem faz a análise de equivalência após a alocação por um secretário.

Pelo perfil do secretário ele consegue então listar todos os professores e ver a que universidade, curso e qual disciplina ele está registrado.

Além disso, todo professor cadastrado no sistema é um usuário, ou seja, é gerada uma senha para esse professor, e posteriormente ele conseguirá fazer login no portal de equivalências para fazer alguma análise. A Figura 21 mostra um cadastro de professor pela interface *web* do sistema de análises de equivalência.

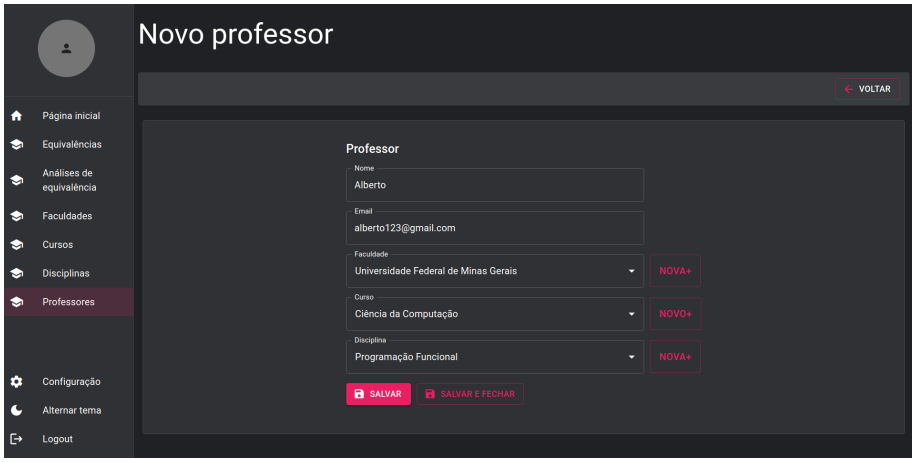

Figura 21 – Cadastro de professor. Fonte: o próprio autor.

É possível observar na Figura 22 que o professor cadastrado virou um usuário do sistema, com uma senha automática gerada.

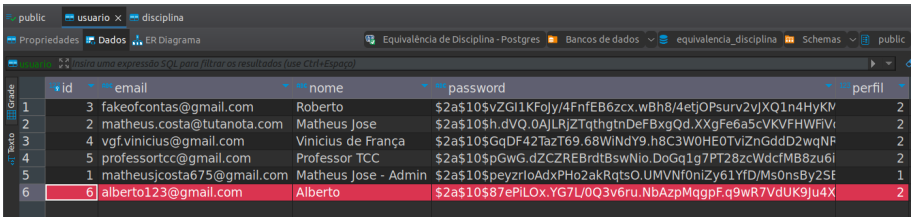

Figura 22 – Exemplo da base de dados de usuário com professor cadastrado. Fonte: o próprio autor.

4.3.4 Análises de equivalência

Agora com o conhecimento da importância das faculdades, cursos, disciplinas e professores cadastrados no sistema, pode-se mostrar sobre o cadastro de uma análise de equivalência entre duas disciplinas a um professor analisador, sendo a tarefa primordial para o perfil de secretário.

Nesta etapa, um aluno dirige-se à coordenação do curso para solicitar uma análise de equivalência, e o secretário utiliza essa tela para colocar os dados que o aluno passou – das duas disciplinas que ele quer a equivalência – para então designar um professor para analisar.

Abaixo, na Figura 23 são ilustradas algumas análises de equivalência, como algumas que já foram analisadas e uma análise que está pendente. As análises de equivalência que já foram efetuadas pelos professores apresentam status na cor verde, e não é possível editar ou apagar esse registro. Já uma análise que está pendente apresenta cor azul, sendo possível editar ou até mesmo apagar o registro.

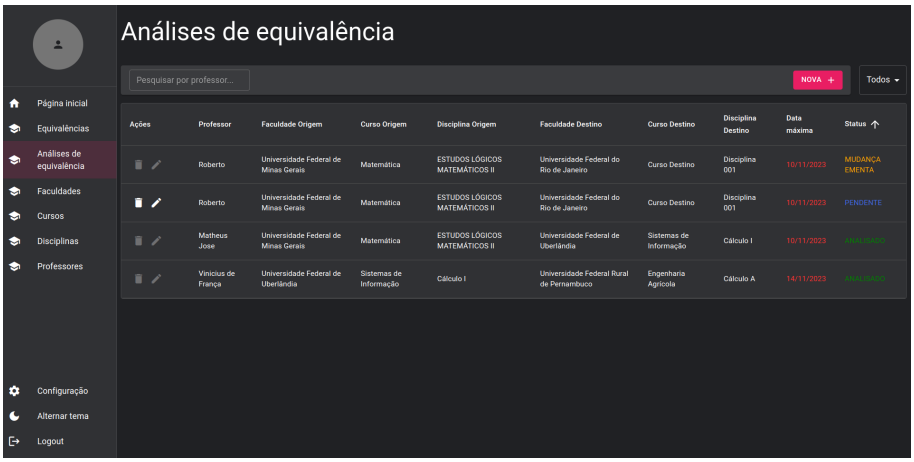

Figura 23 – Listagem das análises de equivalência. Fonte: o próprio autor.

No cadastro de uma análise de equivalência são colocados os dados do aluno que requisitou a análise, além das informações de faculdade, curso e disciplina de origem e de destino, ou seja, as duas disciplinas em que o mesmo solicita equivalência. A Figura 24 ilustra as informações necessárias nesse cadastro.

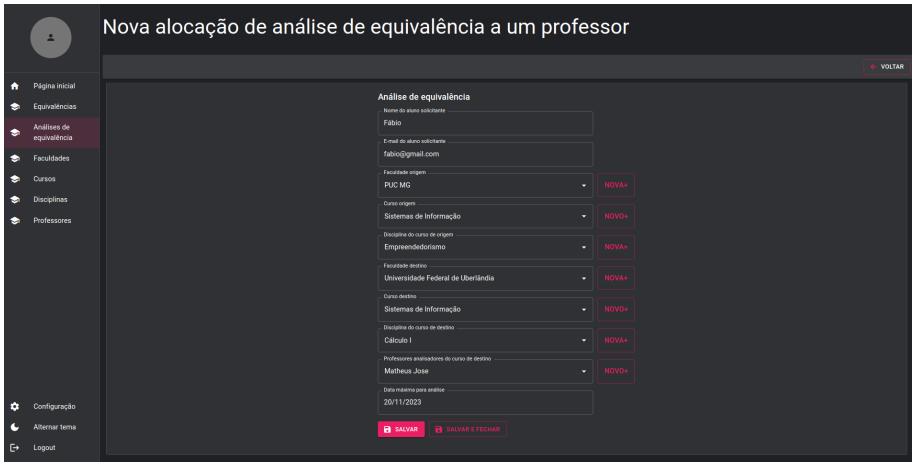

Figura 24 – Cadastro de análise de equivalência. Fonte: o próprio autor.

Nessa tela de cadastro de análise também é possível fazer todo o cadastro de faculdades, cursos, disciplinas e professores. Dessa forma, o secretário não precisa sair da tela de criação de análise de equivalência para acessar uma tela de cadastro de faculdade, curso, disciplina ou professor, o que faz com que a experiência dele como usuário do sistema seja mais fluida, como mostra a Figura 25.

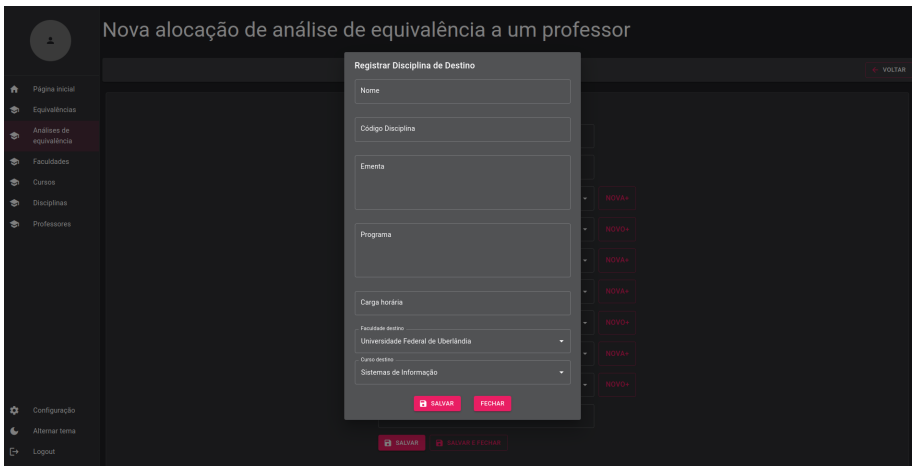

Figura 25 – Modal de cadastro de professor na criação de análise de equivalência. Fonte: o próprio autor.

O secretário informa qual é a data máxima que o professor terá para fazer a análise de equivalência, e uma regra que existe no sistema é que se o professor não fizer a análise, é emitido um comunicado por *e-mail* uma semana antes do prazo expirar. Isso é orquestrado

a partir de uma rotina que fica lendo os dados de uma coleção de notificação, essa coleção de notificação é alimentada no momento que o registro de análise é criado, já calculando a data em que será necessário notificar aquele usuário. Pela Figura 26 tem-se um exemplo de *e-mail* de análise em atraso.

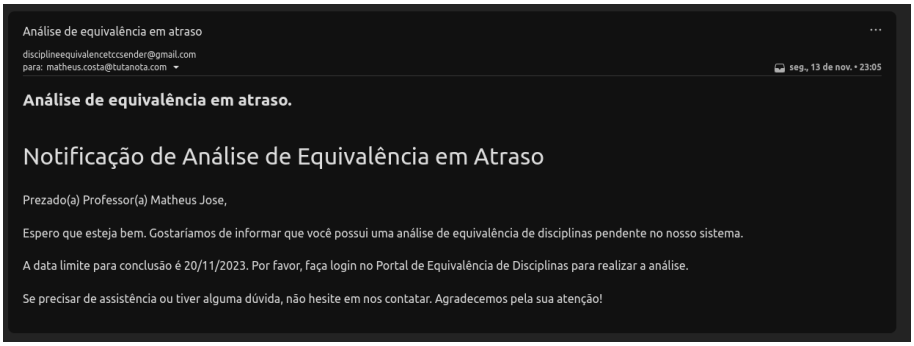

Figura 26 – Exemplo de e-mail que notifica o professor de uma análise em atraso. Fonte: o próprio autor.

Essa análise de equivalência somente pode ser registrada uma única vez no sistema, com aquele mesmo código de disciplina de origem e de destino. Ou seja, se porventura chegar outra solicitação para o secretário para realizar novamente aquela análise de equivalência, o sistema entenderá que aquela análise está pendente para ser analisada, ou já foi devidamente analisada e registrada no sistema, como mostra a Figura 27.

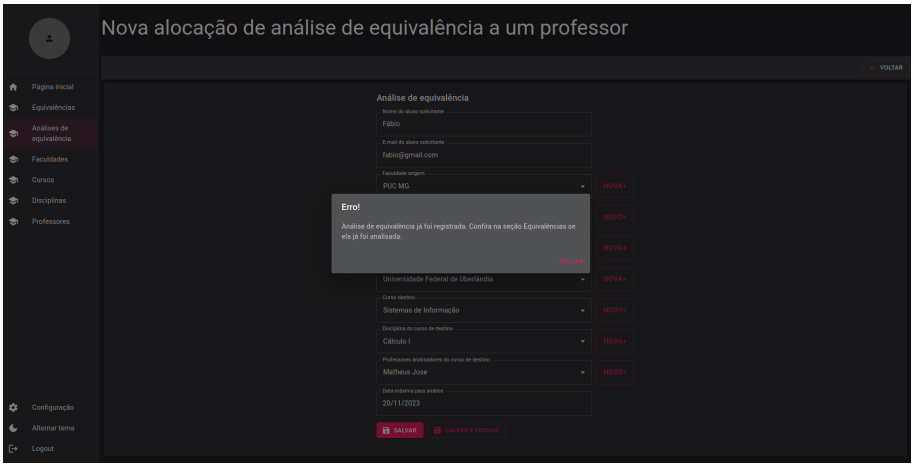

Figura 27 – Exemplo de *feedback* para o secretário de análise de equivalência já registrada. Fonte: o próprio autor.

A partir desse cadastro de análise de equivalência também inicia-se outro processo, que ocorre de forma assíncrona dentro do sistema, o qual é a análise das ementas a partir de uma inteligência artificial generativa, em que foi feita uma integração com o *Open AI*, utilizando o modelo GPT 3.5-turbo, em que basicamente faz-se uma pergunta para o GPT em relação as duas ementas das disciplinas – origem e destino – e também a carga

horária, e pede-se para ele retornar as semelhanças, diferenças e um parecer da Inteligência Artificial se aquelas ementas são equivalentes ou não, e o retorno do GPT é tratado pelo *backend* em campos separados de semelhança, diferença e o parecer do sistema em relação à equivalência. Essa pré-análise será utilizada num relatório que posteriormente será mostrado no perfil de Professor, justamente para facilitar a análise do professor.

Basicamente, quando se cadastra uma análise de equivalência, é registrado em uma tabela que fará o controle do que já foi ou não processado pelo *Open AI*. Pela figura 28 tem-se um exemplo de uma análise de ementa pendente para ser feita. Esse processo assíncrono foi feito a partir de uma rotina executada de tempos em tempos, e analisa todos os registros pendentes de serem analisados pelo *Open AI*.

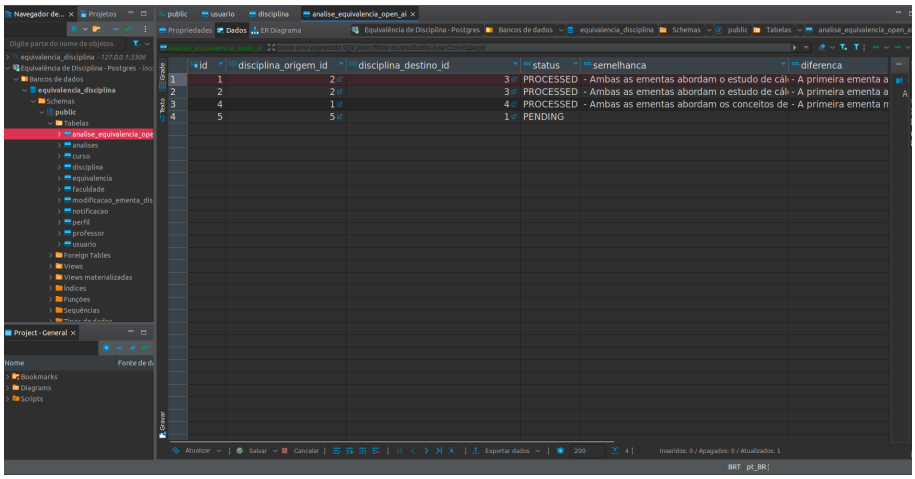

Figura 28 – Registro na tabela de análises do *Open AI*. Fonte: o próprio autor.

A Figura 29 mostra um exemplo de análise de ementa processada pelo *Open AI*. Com os dados processados, tem-se então informações do que é semelhante, diferente e se as ementas são equivalentes, ou seja, um parecer sistêmico a partir da inteligência artificial.

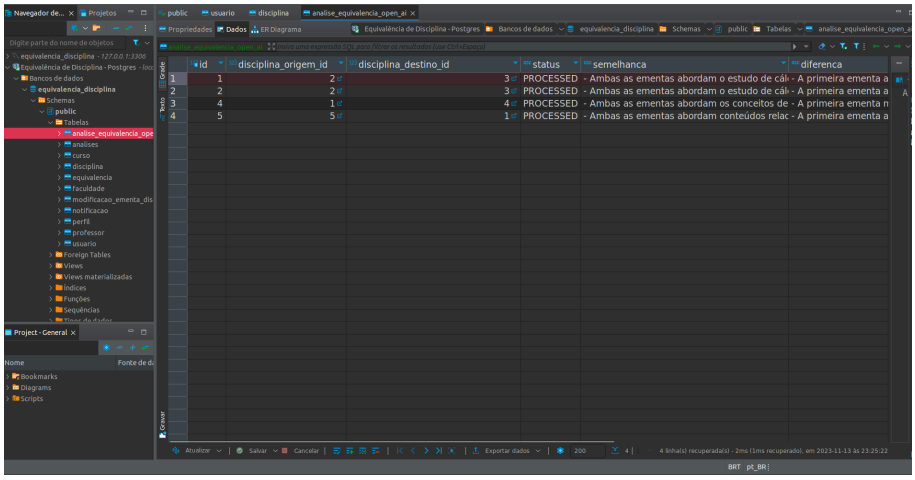

Figura 29 – Registro na tabela de análises processado pelo *Open AI*. Fonte: o próprio autor.

Dessa forma, com a análise de equivalência criada e alocada a um professor, então ele poderá seguir com o seu trabalho, acessando o sistema e registrando uma análise como equivalente ou não equivalente.

4.3.5 Equivalências

Outra funcionalidade importante para o secretário é o de visualizar as equivalências registradas no sistema a partir da análise de equivalência feita por um professor. Pela Figura 30, é possível ver análises em que a equivalência foi verificada, e outra onde não se obteve a mesma. O secretário inclusive consegue filtrar essas equivalências por todas as equivalentes, todas as não equivalentes ou até mesmo filtrar por código de disciplina. Essa tela inclusive possibilita ao secretário ver se uma nova solicitação feita por algum aluno já consta no sistema ou não, pois inclusive não é possível registrar novamente uma mesma análise de equivalência caso essa já tenha sido analisada, como dito anteriormente.

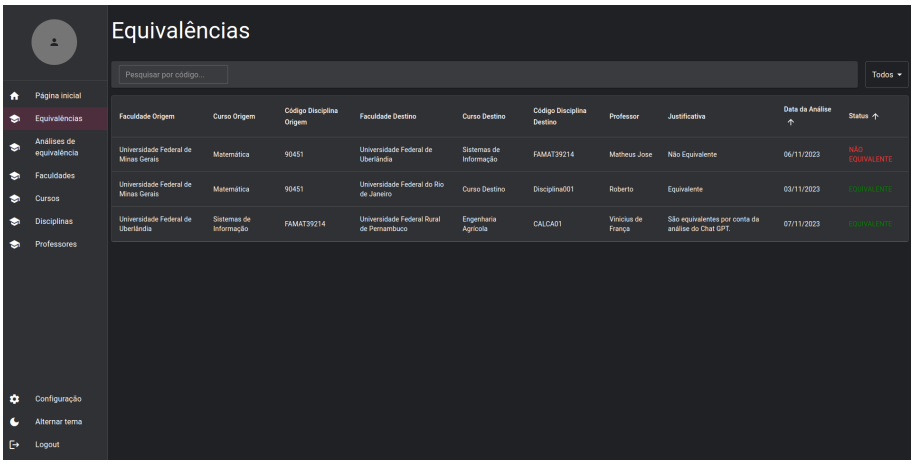

Figura 30 – Equivalências registradas no sistema. Fonte: o próprio autor.

4.3.6 Configuração

Como o usuário pode desejar alterar alguns de seus dados, essa seção foi dedicada a isso. Anteriormente foi falado sobre a criação do professor, em que ele se torna um usuário do sistema com uma senha automática. Inclusive, recomenda-se após essa geração de usuário professor e até mesmo após alguma redefinição de senha a partir da tela de login em que é enviado uma nova senha por *e-mail*, alterar a mesma pela área logada nesta seção de configuração. E nessa tela como mostra a Figura 31 então é possível mudar os seguintes dados: nome, *e-mail* e senha.

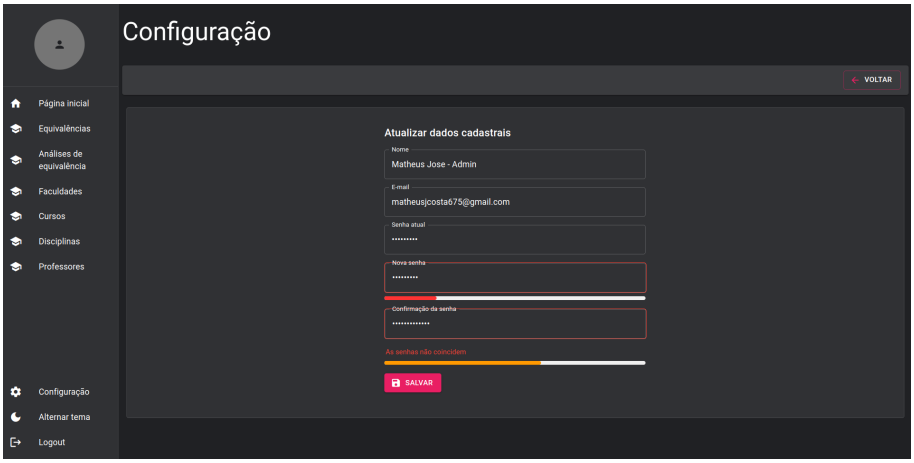

Figura 31 – Alteração de dados cadastrais. Fonte: o próprio autor.

4.4 Professor

O professor é o responsável pela análise de equivalência que foi a ele previamente alocada pelo secretário. É sua tarefa acessar o sistema, verificar suas pendências e fazer as análises requeridas. Abaixo, será ilustrado todo o fluxo do professor dentro do sistema.

4.4.1 Análises de equivalência

Inicialmente, ao fazer login – processo apresentado previamente na seção 4.2 – são listadas todas as análises pendentes. É identificado que esse usuário é um professor por conta do seu tipo de *token*, que possui uma *role* de professor. Com isso, é possível buscar todas as informações daquele usuário pelo seu *e-mail*, que também está presente no *token*, como mostrado na seção 4.1. Nesta tela, é mostrado um resumo daquela análise, como quais são as disciplinas em que a equivalência foi solicitada, e a data máxima para a análise, como mostra a Figura 32.

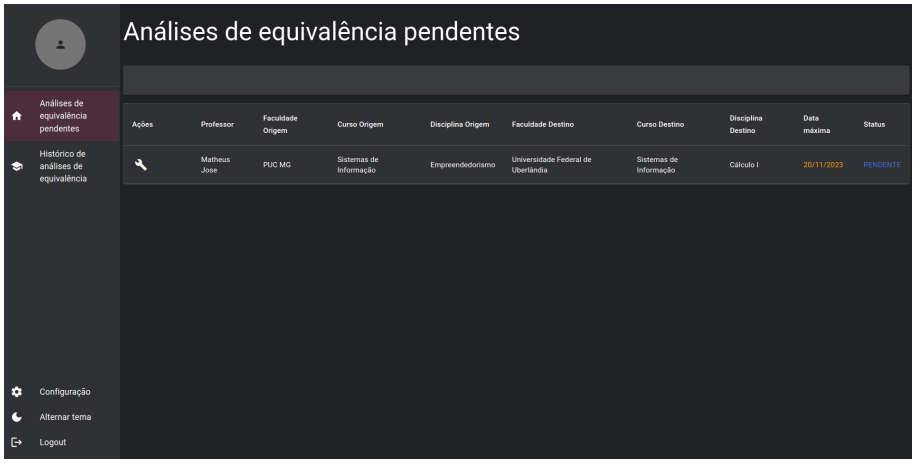

Figura 32 – Análises de equivalência pendentes para análise do professor. Fonte: o próprio autor.

Ao clicar na linha da análise pendente, então é renderizada outra tela, a de relatório de equivalência. Nessa tela, será mostrado um resumo para o professor das disciplinas que ele precisa analisar. Além do resumo, é feita uma pré-análise, esta apresentada anteriormente na subseção 4.3.4, que consiste num parecer do sistema em relação às duas ementas e também em relação à carga horária, feito pela inteligência artificial do *Open AI*, como mostra a Figura 33.

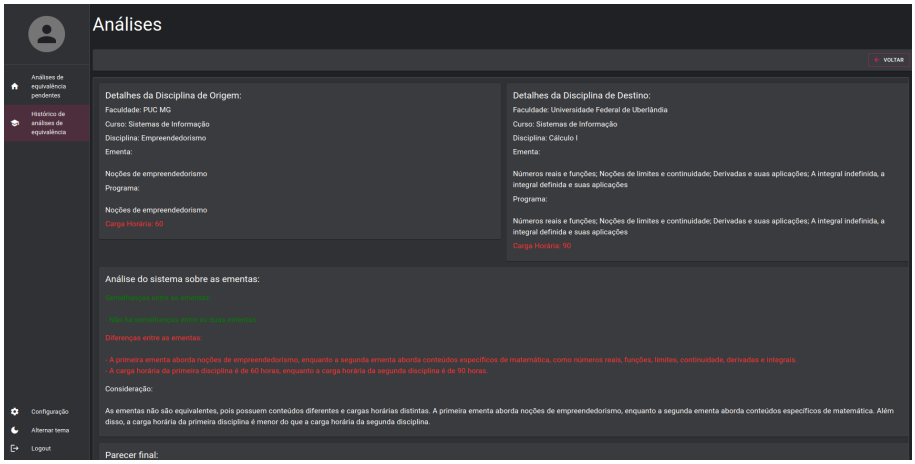

Figura 33 – Relatório de análise de equivalência que é mostrado para o professor. Fonte: o próprio autor.

É importante ressaltar que essa análise do *Open AI* é feita de forma assíncrona, momentos antes do professor fazer a análise, o que torna a experiência do professor na plataforma mais fluida, sem muitos gargalos do *backend* processando dados para retornar ao *frontend*, justamente porque o tempo de resposta do *Open AI* muitas vezes é demorado.

Após o professor efetuar a sua própria análise, é possível emitir o seu parecer final, em que justifica a sua conclusão e cadastra aquela análise como equivalente ou não

equivalente. Na Figura 34, ilustra-se um exemplo de registro de equivalência em que o parecer final do professor é de não equivalência. Como essa é uma operação sensível e que não possível de ser desfeita, o sistema abre uma modal de alerta para que o professor tenha certeza do cadastro, e então confirmar caso esteja tudo correto.

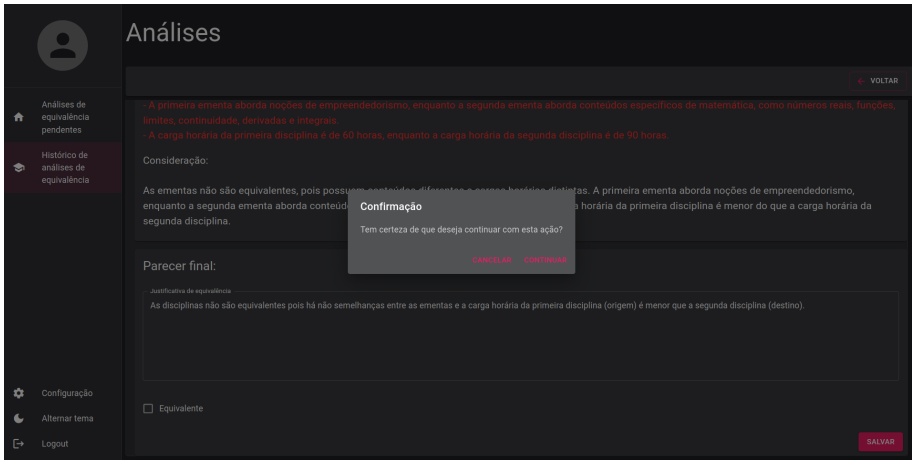

Figura 34 – Cadastro de equivalência. Fonte: o próprio autor.

No momento em que o professor clica em confirmar, o registro é salvo na base de dados na tabela de equivalência. Em seguida, é enviado um recibo de registro de equivalência tanto para o professor quanto para o secretário que fez o registro daquela análise de equivalência, como mostra a Figura 35. Esse envio de *e-mail* é também feito de forma assíncrona, ou seja, após o cadastro da análise é retornado para o usuário que o registro foi cadastrado, e no *backend* é executada uma rotina de tempos em tempos para notificar esses recibos de equivalência. Isso foi pensado e implementado para que o usuário não fique esperando na tela enquanto o *backend* processa o envio de *e-mail*.

Figura 35 – Exemplo de *e-mail* com recibo da equivalência registrada. Fonte: o próprio autor.

E com esse cadastro de equivalência, então posteriormente caso alguém solicite novamente análise entre essas duas disciplinas, o secretário poderá dar um parecer mais rápido para o aluno sobre se são equivalentes ou não.

4.4.2 Histórico de análises de equivalência e configuração

Por fim, o professor pode visualizar um histórico de todas as análises que ele realizou, e filtrar pelas equivalentes ou não equivalentes, e não é possível mais fazer uma análise daquele registro que já foi computado, como mostra a Figura 36.

Figura 36 – Histórico de análises de equivalência do professor. Fonte: o próprio autor.

E para o professor também é permitido mudar alguns dados básicos do usuário pela tela de configuração, tais como seu nome, *e-mail* e senha, a exemplo do foi mostrado na seção 4.3.6 para a persona de secretário.

5 Conclusão

Com o sistema de análise de equivalências e as suas funcionalidades, a rotina da Coordenação e do Colegiado de curso será simplificada. Uma vez que o sistema consegue orquestrar as análises pendentes por professor e também sobre o que já foi analisado, não deixando assim que um trabalho já feito seja novamente um item de uma fila de análises de equivalência. E sabe-se que há alta demanda, proveniente de alunos transferidos, que solicitam essas análises de equivalência para convalidar algumas disciplinas previamente cursadas em outros lugares.

Além disso, as notificações que o sistema faz para o professor em caso de expiração de uma análise de equivalência acaba retirando da Coordenação e sua secretaria a responsabilidade de notificar os professores envolvidos no processo. E com a notificação dos recibos das equivalências registradas pelos professores tem-se uma maior segurança sobre o que já foi feito, o que inclusive é também enviado para o secretário, ou seja, ambos possuem o recibo e em caso de algum questionamento por parte da coordenação é possível analisar esse histórico.

Em suma, este é um sistema que agregará para a rotina às vezes sobrecarregada das coordenações de curso, e principalmente dos professores, que já têm uma carga de trabalho muito grande, a inteligência artificial, aplicada na pré-análise das ementas, e que será um diferencial na agilidade e na qualidade das análises. Além disso, como continuação deste projeto há algumas funcionalidades que podem ser listadas e que contribuiriam imensamente para o uso do sistema, como:

1. Criar mecanismo para adicionar as ementas das disciplinas a partir de uma leitura de PDF;
2. Criar mecanismo para adicionar as ementas das disciplinas a partir de uma imagem utilizando tecnologia OCR;
3. Notificar o aluno com um recibo da análise realizada;
4. Cadastrar secretário/coordenador(a) por faculdade e curso.

Com essas funcionalidades o sistema ganhará mais corpo, principalmente no que diz respeito aos alunos que solicitarem análise serem notificados após efetivação da mesma. E também o cadastro das ementas com *input* de PDF e imagens, em que o *backend* automatizará o processo de cadastro de ementas e retirando mais um trabalho manual do secretário que é o de cadastrar as ementas das disciplinas.

Referências

ANDERSON, C. Docker [software engineering]. **Ieee Software**, IEEE, v. 32, n. 3, p. 102–c3, 2015. Citado na página 17.

BIERMAN, G.; ABADI, M.; TORGERSEN, M. Understanding typescript. In: SPRINGER. **ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28**. [S.l.], 2014. p. 257–281. Citado na página 17.

GACKENHEIMER, C. **Introduction to React**. [S.l.]: Apress, 2015. ISBN 9781484212462. Citado na página 16.

INFOQ. **Spring Data: A solução mais geral para persistência?** 2013. Maleabilidade ao manusear bancos de dados. Disponível em: <https://www.infoq.com/br/articles/spring-data-intro/>. Acesso em: 19 ago. 2023. Citado na página 15.

JACOBSON, I.; CHRISTERSON, M.; JONSSON, P.; ÖVERGAARD, G. Object-oriented software engineering: A use case driven approach. **ACM Transactions on Information Systems (TOIS)**, v. 10, n. 2, p. 168–237, 1992. Citado na página 20.

JAVA. **O que é tecnologia Java e por que preciso dela?** 2023. Desenvolvimento backend. Disponível em: <https://www.java.com/pt-BR/download/help/whatis_java.html>. Acesso em: 19 nov. 2023. Citado na página 14.

JOHNSON, R.; HOELLER, J.; DONALD, K.; SAMPALEANU, C.; HARROP, R.; RISBERG, T.; ARENDSEN, A.; DAVISON, D.; KOPYLENKO, D.; POLLACK, M. et al. The spring framework-reference documentation. **interface**, v. 21, p. 27, 2004. Citado na página 14.

JONES, M.; BRADLEY, J.; SAKIMURA, N. **Json web token (jwt)**. [S.l.], 2015. Citado na página 15.

MASSE, M. **REST API design rulebook: designing consistent RESTful web service interfaces**. [S.l.]: "O'Reilly Media, Inc.", 2011. ISBN 9781449310509. Citado na página 16.

MICROSSERVICOS. **O que são microsserviços?** 2023. Sistemas distribuídos. Disponível em: <https://aws.amazon.com/pt/microservices/>. Acesso em: 19 nov. 2023. Citado na página 16.

MOMJIAN, B. **PostgreSQL: introduction and concepts**. [S.l.]: Addison-Wesley New York, 2001. v. 192. ISBN 9780201703313. Citado na página 16.

MULARIEN, P. **Spring Security 3**. [S.l.]: Packt Publishing Birmingham,, England, 2010. ISBN 9781847199744. Citado na página 15.

NORMASEQUIVALENCIA. **Resolução Congrad Nº 46**. 2022. Normas equivalência de disciplinas. Disponível em: <http://www.reitoria.ufu.br/Resolucoes/resolucaoCONGRAD-2022-46.pdf>. Acesso em: 27 jun. 2022. Citado 2 vezes nas páginas 11 e 21.

PRESSMAN, R. S. **Engenharia de Software: Uma Abordagem Profissional**. [S.l.]: McGraw-Hill, 2014. ISBN 9786558040101. Citado na página 19.

REIS, D.; PIEDADE, B.; CORREIA, F. F.; DIAS, J. P.; AGUIAR, A. Developing docker and docker-compose specifications: A developers' survey. **Ieee Access**, IEEE, v. 10, p. 2318–2329, 2021. Citado na página 18.

TILKOV, S.; VINOSKI, S. Node. js: Using javascript to build high-performance network programs. **IEEE Internet Computing**, IEEE, v. 14, n. 6, p. 80–83, 2010. Citado na página 17.

VAGASOCIOSAS. **Transferências para a UFU para preenchimento de vagas ociosas**. 2022. UFU tem inscrições abertas para preenchimento de quase 2 mil vagas ociosas. Disponível em: <https://g1.globo.com/mg/triangulo-mineiro/noticia/2022/03/24/ufu-tem-inscricoes-abertas-para-preenchimento-de-quase-2-mil-vagas-ociosas.ghtml>. Acesso em: 27 jun. 2022. Citado 2 vezes nas páginas 11 e 12.

WALLS, C. **Spring Boot in action**. [S.l.]: Simon and Schuster, 2015. ISBN 9781617292545. Citado na página 15.