UNIVERSIDADE ESTADUAL DE SANTA CRUZ - UESC
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Alexandre Pedrecal Campos

DESENVOLVIMENTO DE API PARA EXTRAÇÃO DE INFORMAÇÃO TEXTUAL DE ARQUIVOS PDF PADRONIZADOS

ILHÉUS - BAHIA
2020

ALEXANDRE PEDRECAL CAMPOS

DESENVOLVIMENTO DE API PARA EXTRAÇÃO DE INFORMAÇÃO TEXTUAL DE ARQUIVOS PDF PADRONIZADOS

Trabalho de Conclusão de Curso apresentado à Universidade Estadual de Santa Cruz - UESC, como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Mestre Hélder Conceição Almeida

ILHÉUS - BAHIA
2020

ALEXANDRE PEDRECAL CAMPOS

DESENVOLVIMENTO DE API PARA EXTRAÇÃO DE INFORMAÇÃO TEXTUAL DE ARQUIVOS PDF PADRONIZADOS

Trabalho de Conclusão de Curso apresentado à Universidade Estadual de Santa Cruz - UESC, como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

Ilhéus, 30 de Janeiro de 2020

Prof. Mestre Hélder Conceição Almeida
UESC/DCET
(Orientador)

Prof. Doutor Marcelo Ossamu Honda
UESC/DCET

Prof. Álvaro Vinícius de Souza Coêlho
UESC/DCET

Agradeço em primeiro lugar aos meus pais Valter e Indaiá e a minha avó Aurelita que sempre me incentivaram e fizeram o máximo para que eu tivesse uma boa educação e chegasse a Universidade. Como fruto de tudo que aprendi e conquistei na UESC, dedico este trabalho a vocês.

AGRADECIMENTOS

Agradeço a Deus que me deu força, saúde e sabedoria durante toda a minha graduação.

Agradeço a todos os familiares que se alegraram comigo no momento de aprovação e me incentivaram. Principalmente o meu primo Lui que sempre me apoiou e me ajudou quando precisei.

Agradeço a minha namorada Cássia, que esteve comigo durante a maior parte da graduação e foi uma das principais incentivadoras até o presente dia.

À Universidade Estadual de Santa Cruz (UESC) pela infraestrutura e bolsas de Iniciação Científica concedidas.

Ao Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas (NBCGIB) pela infraestrutura disponibilizada para o desenvolvimento de projetos durante a minha graduação.

Os meus amigos Gabriel Figueiredo, Levy Marlon, Adson Cardoso, Aurélio Chaussê, Gabriel Rodrigues, Matheus Almeida, Tulio Campos, Alberto Segundo, Daniel Penedo e Luís Carlos (grupo semi.Pro) que foram verdadeiros irmãos durante essa jornada e sem o apoio deles nada disso seria possível.

A todos os professores que participaram da minha graduação. Ao Professor Hélder Almeida, pela orientação, suporte e conselhos dados na elaboração desse trabalho. Ao Professor Marcelo Ossamu Honda, pelo incentivo e conhecimento compartilhado desde o início da graduação, pelo projeto de iniciação cientifica e conselhos profissionais.

"A dor não diz quando você deve parar.
A dor é a pequena voz em sua cabeça que tenta impedi-lo,
porque sabe que se você continuar, você mudará."
(Kobe Bryant)

RESUMO

Com a constante mudança do papel para o digital, o Portable Document Format (PDF) ganhou muito espaço quando o assunto é documentos digitais. A popularização do formato fez com que surgisse a necessidade da extração de dados de tais arquivos. As existentes ferramentas que realizam este trabalho, em sua maioria, são pagas ou não realizam uma extração personalizável. O objetivo deste trabalho é sanar essas necessidades através de uma API que, de forma configurável, permita a extração de dados textuais de documentos PDF e os entreguem em formato JSON. Assim é possível realizar a extração e diretamente alimentar os dados extraídos a outros sistemas. Através da utilização e manipulação de algumas bibliotecas, foi desenvolvida a API que compreende uma boa documentação e respeita as constrições do estilo de arquitetura REST. Desta forma foi obtida uma ferramenta que é capaz de se integrar a outros sistemas ou simplesmente ser utilizada diretamente da sua rota de documentação, realizando a extração de dados textuais, dada a configuração de onde o conteúdo pode ser encontrado e filtrado. E garantido sua segurança e integridade através da validação de usuário e chave de API. Como forma de validação destas funcionalidades, também foi desenvolvido uma aplicação *front-end* que consome a API e faz utilização de todos seus recursos, com o objetivo de extrair dados dos arquivos PDF de Trabalhos de Conclusão de Curso do curso de Ciência da Computação da Universidade Estadual de Santa Cruz.

Palavras-chave: API, REST, PDF, Extração de Informação

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

PDF	*Portable Document Format*
ISO	*International Organization for Standardization*
API	*Application Programming Interface*
NPM	*Node Package Manager*
JSON	*JavaScript Object Notation*
HTML	*Hypertext Markup Language*
CSS	*Cascading Style Sheets*
CRUD	*Create, Read, Update and Delete*
JWT	*JSON Web Token*
UI	*User Interface*
NBCGIB	Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas

SUMÁRIO

1 INTRODUÇÃO

Com sua primeira versão no início da década de 90, o formato de arquivos Portable Document Format (PDF) foi criado pela Adobe Systems com o intuito de tornar a troca de documentos algo mais prático e seguro (LIN et al., 2011). Conforme sua popularização foi crescendo, setores técnicos passaram a adotar o formato em seus manuais e documentos, o que acarretou na adoção do formato em algumas ISO's, até que em 2008 foi publicada sua própria ISO (ISO 2008). Atingido o objetivo de ser um formato de arquivo que representa documentos independentemente de hardware e software, este é atualizado até os dias atuais.

Proveniente deste crescimento, é natural que surgisse a necessidade de ferramentas que extraíssem os dados de tais documentos. O grande número de arquivos e informações úteis em tais, impulsionou a criação de diversas ferramentas com este objetivo. Porém, não é uma tarefa fácil, a organização interna e as diferentes formas de gerar um arquivo PDF induziram a soluções para extrações de informações específicas, e ainda assim não muito confiáveis.

Existem trabalhos científicos neste formato, como os apresentados por SASIREKHA; CHANDRA, AJEDIG; LI; REHMAN, MARINAI e LIN et al. Porém, com a dificuldade para extração dos dados muitas informações não são aproveitadas. Existem ferramentas que catalogam estes trabalhos, mas com a extração de informações selecionadas diretamente dos arquivos é possível criar um banco de dados onde as informações podem ser utilizadas e facilmente acessadas.

Considerando os documentos PDF como uma grande fonte de informações, e a extração dos dados como uma ferramenta, surgem diversas possibilidades de utilização desta extração. E assim a construção deste projeto como uma API tornaria possível que diversos outros sistemas que necessitem realizar a extração de dados o utilizem.

Neste contexto foi proposta uma solução onde, através da utilização de ferramentas atuais e confiáveis, seria possível definir o que será extraído e filtrar estas informações. Além disso, a utilização de tal software poderia ser feita tanto por usuários quanto por outros sistemas. Assim, dada a relevância deste problema, o sistema aqui descrito deve sanar muitas necessidades já descritas e ser apto a receber novos módulos com novas funcionalidades e melhorias.

1.1 Objetivo Geral

Propor uma API RESTful para a extração de dados textuais de documentos PDF padronizados.

1.2 Objetivos Específicos

- Garantir segurança da API e integridade dos dados.
- Gerar documentação da API.
- Validar a API através de site que a consuma para realizar a classificação de TCC's do curso de Ciência da Computação da Universidade Estadual de Santa Cruz;

1.3 Organização do Trabalho

O Capítulo 2 apresenta o referencial teórico, resultante de pesquisas e estudos de diversas bibliotecas e tecnologias utilizadas para a construção deste sistema. O estudo destas tecnologias foi além do âmbito acadêmico e fundamentado em ferramentas muito utilizadas no mercado de trabalho (Stack Overflow). Através do Stack Overflow Trends foi analisada a popularidade de tais tecnologias, e o estudo detalhado visando a integração ao sistema. Além disso são apresentados sistemas com propósito similar ao trabalho aqui apresentado.

O Capítulo 3 apresenta os Materiais e Métodos, este é dividido em duas principais partes, a primeira lista ferramentas e suas versões, explicando sua importância e o papel exercido no sistema. A segunda descreve como foi a implementação da API, passando desde o estudo das ferramentas até o desenvolvimento do código.

O Capítulo 4 descreve os resultados obtidos com a API, e fundamenta tais resultados com a criação e utilização de um sistema que consome a API. Este tem o objetivo de extrair informações de TCC's do curso de Ciência da Computação da UESC.

O Capítulo 5 apresenta as conclusões e discussões como consequência deste projeto. Assim como também indica possíveis trabalhos futuros.

2 REFERENCIAL TEÓRICO

Para realizar o desenvolvimento da aplicação foi necessária uma fundamentação teórica, apresentando outras literaturas, de forma que a escolha das tecnologias a serem utilizadas fossem guiadas através deste embasamento.

2.1 Extração Automática de Informações em PDF

Desde sua criação até os dias atuais a utilização de arquivos PDF só cresce, e é o formato mais utilizado para troca de documentos. Extrair dados através de textos em documentos é um objetivo de interesse de muitos que desejam juntar informações que se encontram isoladas em tais, porém não é uma tarefa simples, principalmente, se tratando de arquivos PDF. A extração automatizada encontra muitas barreiras, de diversas naturezas, e soluções mais genéricas a este problema, não são tão comuns (AJEDIG; LI; REHMAN, 2011).

2.1.1 PDF - *Portable Document Format*

Como definido no manual da Adobe Systems (BIENZ; COHN; VIEW, 1993), o PDF é um formato de arquivo para representar um documento, de forma que não haja dependências como fonte, imagens e anexos. O que quer dizer que ele tem a capacidade de reunir todos os recursos necessários para que a visualização do documento não dependa de nenhuma fonte externa.

Com o objetivo de compartilhar documentos digitais, em 1992 a Adobe Systems realizou o lançamento do formato PDF. Em 1993 o formato foi disponibilizado de forma gratuita, o que gerou um grande aumento em sua adoção, mas somente em 2008 ele deixou de ser um formato proprietário e passou a ser um Padrão Aberto, isto significa que qualquer pessoa pode ter acesso à implementação deste formato e não é mais possível ser cobrado royalties pelo uso do mesmo. No mesmo ano foi publicada a ISO 32000-1 (ISO, 2008) declarando o PDF como um padrão internacional verdadeiramente aberto.

A capacidade de gerar um documento fiel a sua criação, e possível de ser visualizado em qualquer dispositivo que possua um leitor do formato, contribuiu para que ele se tornasse uma das ferramentas de documento digital mais utilizado no mundo (Sobre o Adobe PDF 2020).

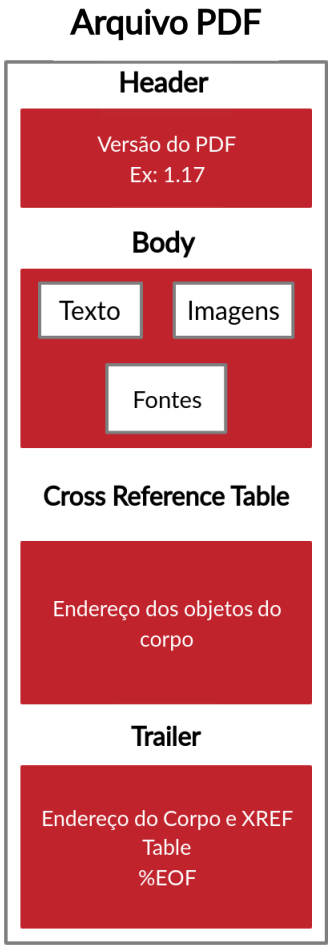
Na **Figura 1** é apresentado um gráfico comparativo do interesse entre os tipos de arquivo do Microsoft Word e do PDF, entre primeiro de janeiro de 2004 e 31 de dezembro de 2019. O gráfico é gerado através de métricas privadas do Google.

Figura 1 – Comparação de Interesse entre o formato PDF Microsoft Word.

Fonte: Google Trends

O PDF é descendente do *PostScript*, uma linguagem de descrição de páginas que inicialmente foi criada para gerar laudas a serem impressas. O *PostScript* foi desenvolvido pela Adobe Systems e os objetos encontrados no código de arquivos PDF possui sintaxe similar a tal linguagem (WARNOCK; GESCHKE, 2001). Como pode ser visto na **Figura 2**, a estrutura do PDF é dividida em quatro partes:

Figura 2 – Detalhamento da estrutura do PDF.



Fonte: Autoria Própria

- *Header*: Contém o número da versão do PDF que foi utilizado para gerar tal documento;
- *Body*: Aqui está presente todo o conteúdo que pode ser visualizado no documento. Toda informação pertencente ao *Body* é dividida em objetos para a informação que será visualizada em cada página. O que engloba texto, imagens, fontes, etc. Para fins de otimização os objetos encontrados no *Body* não se repetem mesmo estando presente em diferentes páginas;
- *Cross Reference Table* (*XREF Table*): Esta tabela é responsável por indicar o posicionamento dos objetos no *Body*. Desta forma não é necessária a leitura de todo o conteúdo para que seja encontrado o objeto necessário para compor a visualização;
- *Trailer*: Contém a localização do *Body*, da *Cross Reference Table* e o fim do arquivo. O leitor de arquivos PDF irá acessá-lo primeiro para saber onde encontrar a tabela e assim carregar os objetos necessários.

2.1.2 Obstáculos na Análise dos Dados

A extração e análise de dados, principalmente na área acadêmica, é um problema abordado desde a padronização, e reconhecimento dos arquivos PDF como um formato universal. A ampla utilização dos documentos PDF faz com que a análise e extração destes documentos recebam cada vez mais atenção, e este interesse vem de diversas áreas. É possível encontrar pesquisas que abordam o reconhecimento de dados textuais, fórmulas matemáticas, códigos genéticos, gráficos, entre outras informações. Por ser mais conveniente, são buscadas ferramentas que transformem o arquivo em uma forma intermediária, como por exemplo texto ou HTML, para que seja mais fácil a extração e análise de dados (AJEDIG; LI; REHMAN, 2011).

Com essas abordagens, chegam os desafios, como exemplo, a falta de estrutura padronizada, diferentes formatos dado o software que gerou o arquivo, a incapacidade de renderizar o conteúdo corretamente, entre outros. Como foi apresentado no tópico 2.1.1, a codificação do PDF não contém as informações de uma forma coerente a leitura humana, e sim blocos que são distribuídos e montados no momento da visualização do arquivo. Assim, é necessário realizar uma abordagem na qual as informações buscadas sejam coerentes e não blocos soltos. Outra recorrente dificuldade está nas diferentes formas que um software pode gerar um arquivo PDF, como no LaTeX, onde certos símbolos matemáticos são desenhados como objetos e no Microsoft Word são utilizados os símbolos Unicode (SASIREKHA; CHANDRA, 2013). A abordagem através do reconhecimento de imagem é muito comum, mas enfrenta um grande problema como falsos positivos e requer um maior poder de processamento (AJEDIG; LI; REHMAN, 2011).

Com o objetivo de criar algo mais genérico e reaproveitável para o meio acadêmico, foi pensando em uma forma mais configurável de se realizar a análise dos dados diretamente do PDF montado, e superar as barreiras citadas anteriormente.

2.2 Tecnologias

Com o objetivo de criar uma API segura e que possa extrair dados de documentos PDF de forma confiável, foram estudadas diversas tecnologias, que quando unidas, complementam umas as outras e entregam uma boa performance, cumprindo o que desejado.

2.2.1 API - Interface de Programação de Aplicações

Uma API é um software intermediário que permite a interação com diferentes aplicações. Ela vai além do que seu nome sugere, e atua principalmente na questão de interface. O grande foco de tal está na forma em que ela se apresenta, isso se tratando de quem a consome, que pode ser tanto para programas quanto para humanos. E se tratando de uma API web, esta interface pode ser acessada em qualquer lugar do mundo. Ela atua recebendo requisições de outra aplicação, então uma rotina correspondente é executada e uma resposta é devolvida a aplicação que realizou a requisição (JIN; SAHNI; SHEVAT, 2018).

1. REST - Transferência de Estado Representativo

   Fielding et al. (2000) propôs em sua tese de doutorado, comparando com estilos arquiteturais já existentes, um estilo para API onde as necessidades de uma aplicação em rede fosse atendidas da melhor forma, e conseguissem que o desenvolvedor seguisse este padrão sem grandes complicações. Os principais benefícios trazidos pelo REST contam com (MASSÉ, 2012):

   - Performance: Resultante de uma comunicação simples e eficiente;
   - Escalabilidade: Havendo interações simples, é mais prático o crescimento do sistema;
   - Simplicidade de Interface: Uma interface simples garante uma interação simples e consequentemente traz benefícios como os citados anteriormente;
   - Manipulação de componentes: Seguindo este estilo há uma separação de responsabilidades, logo um componente não deve depender do outro o que permite que a alteração de um não cause danos a outros;
   - Portabilidade: Com uma comunicação e Interface simples qualquer API que adote o estilo REST consegue ser consumida por qualquer tipo de tecnologia.

- Confiabilidade: Devido a suas restrições, cada requisição é um processo independente o que facilita na recuperação em caso de falhas;
- Visibilidade: A Restrição Stateless determina que cada requisição deve ser independente, logo, se forem feitas diversas requisições cada uma deve ter sua resposta e a visibilidade das respostas não precisam ser analisadas.

Fielding et al. (2000) abordou a definição do REST a partir de restrições, elas definem limites e regras para os componentes que compõe o sistema. São seis restrições:

1. *Client-Server*: É a restrição mais básica, ela garante que a API irá lidar com a comunicação com o banco de dados, gerenciamento de cache, log, etc. Desta forma são separadas as responsabilidades do *front-end* e do *back-end*;
2. *Stateless*: Como dito anteriormente, esta restrição diz que o cliente pode fazer várias requisições, isso significa que para cada requisição que ele fizer deve ser enviado todas as informações novamente;
3. *Cacheable*: Esta restrição não precisa ser aplicados para todos os componentes, ela define que a informação de resposta deve ser guardada em cache. No caso de muitas requisições a mesma informação não é necessário ocorrer todo o processo de busca no banco de dados e processar os dados;
4. *Uniform Interface*: Tendo uma interface uniforme para todas as requisições o cliente sabe exatamente o formato de resposta que ele deve esperar e é possível ter o mesmo resultado para diferentes plataformas que estiverem consumindo a API;
5. *Layered System*: Tendo em mente que a API está conectada a internet e deve ocorrer um grande tráfego de informação, é aplicado o conceito de camadas para que haja uma simplificação do sistema e uma boa organização dos componentes. Cada camada tem sua responsabilidade, como por exemplo executar a lógica e buscar ou salvar informações;
6. *Code-On-Demand*: Única restrição opcional, ela permite que o cliente execute código da lógica do servidor através de um script.

Estas restrições atuam como paredes invisíveis e conseguem guiar o desenvolvedor. Aplicando estas condições no código é possível obter várias vantagens e construir uma API de ótima performance, coesa e escalável. Quando estas restrições são aplicadas de forma correta no projeto, é utilizada a expressão RESTful para identificar que aquela API está incorporando os limites dado pelo estilo de arquitetura REST.

2. XML vs JSON

Para a troca e leitura dos dados foi necessário escolher um formato. HAQ; KHAN; HUSSAIN descrevem que o massivo aumento da web trouxe uma grande necessidade

de um formato para troca de dados entre diferentes tecnologias. Não havia um formato padronizado de forma que, automaticamente, os dados exportados fossem entendidos ao serem importados em outra aplicação. Com essa necessidade Sperberg-McQueen et al. 2008 junto ao W3C, criaram o XML (*Extended Markup Language*), sendo uma linguagem de marcação, assim como o HTML, mas atendendo a demanda da troca de dados. Atualmente é a forma mais adotada para documentação e registro de dados simples.

Atualmente existem várias alternativas ao XML, sendo as mais famosas JSON e YAML, neste tópico será tratada apenas a comparação com o JSON. O *JavaScript Object Notation* (JSON) é um formato de texto criado para representar um objeto JavaScript, de forma simples, portável e textual (JSON, 2014).

Comparado com o XML, o formato JSON é mais compacto, menos verborrágica e com o crescimento do número de aplicações escritas em JavaScript, este formato se tornou muito popular e vem tomando espaço do XML. HAQ; KHAN; HUSSAIN realizaram uma comparação onde as seguintes vantagens são atribuídas ao JSON:

- Velocidade: A análise e leitura de arquivos XML tendem a ser lentas, devido a sua verborragia, enquanto o JSON entrega os dados de forma simples e direta;
- Serialização e Desserialização: Serialização significa conversão de objeto para string. Normalmente é encontrada uma única forma de serialização e desserialização, no caso do JSON esta já é presente no JavaScript. Já o XML contém várias formas diferentes de realizar tal;
- Conciso: O JSON utiliza a técnica de chave-valor, enquanto o XML utiliza tags, o que dificulta a visualização e aumento o tamanho do arquivo;
- Bibliotecas de fácil utilização: Atualmente é possível encontrar bibliotecas, para a maioria das linguagens, que tornam a utilização do JSON muito orgânica, enquanto a utilização do XML é algo mais complicado.

Dado as vantagens, e a escolha do Node.JS, que será melhor abordada no próximo tópico, foi decidido que todos os dados na aplicação serão trabalhados no formato JSON.

3. *Middleware*

Com a utilização de uma API é necessário definir o que é um *middleware* e quais tipos serão utilizados neste projeto. BAKKEN (2001) diz que *middleware* é uma camada de software que se encontra entre dois sistemas e provê uma abstração comum para o entendimento de ambos.

A ampla utilização deste termo pode trazer dúvidas quanto ao seu significado, mas como definido anteriormente, o *middleware* pode ser visto como uma cola que liga

dois sistemas ou componentes de um software. Tendo esse significado, é possível aplicar esta definição a diferentes níveis. Porém, no caso do projeto em discussão, serão abordados dois tipos.

O *middleware* de rotas, desempenha a função de realizar uma ação antes da requisição chegar ao controlador. Isso proporciona oportunidades como verificação dos dados recebidos, filtragem e preparação para execução da rotina em tal rota.

E o *middleware* a nível de aplicativo, este funciona de acordo com o funcionamento do sistema, estando presente em todo tipo de ação e fazendo a ligação das ações executadas com o ciclo de vida da API.

2.2.2 Node.JS

Node é a forma de utilizar JavaScript em um servidor. A implementação do Node tem base no interpretador JavaScript desenvolvido pela Google, o V8, este interpretador é implementado em C e C++, e se concentra em extrair a melhor performance utilizando menos memória (TILKOV; VINOSKI, 2010).

Motivado a criar uma comunicação simples entre o servidor e a página web, Ryan Dahl criou o Node.JS em 2009 e foi um sucesso imediato. Fundamentado em ser *Event Driven*, o que quer dizer que há sempre um *core* escutando por todos os eventos, e chamando as devidas funções quando tais eventos são acionados(SHAH, 2017).

A introdução do Node.JS trouxe características de outras linguagens que não existem no JavaScript no navegador, como a manipulação do sistema de arquivos, acesso a um banco de dados e a criação de um cliente HTTP, o que faz possível a criação de um *web server* (MEAD, 2018). Pela familiaridade de grande parte dos desenvolvedores web com a linguagem JavaScript e a separação do *Front-End* e *Back-End*, a criação de API's utilizando o Node.JS se tornou muito popular e atualmente, boa parte das ferramentas encontradas para atingir os objetivos deste trabalho podem ser encontradas como bibliotecas de tal.

No mesmo ano de lançamento do Node.JS foi lançado o *Node Package Manager* (NPM), que atua como um gerenciador e distribuidor de pacotes para projetos, e conta com uma interface por linha de comando que auxilia na criação e manejo de tais. Atualmente o NPM é o maior registrador de software do mundo, devido a sua facilidade e a imensa comunidade dos usuários. Em poucos segundos é possível submeter um pacote ao NPM e assim com a visibilidade e apoio da comunidade este pacote é mantido e utilizado livremente por desenvolvedores ao redor do mundo. Muitas tarefas específicas e morosas são evitadas através da utilização de algum pacote que já contém determinada solução para tal tarefa. Uma importante tarefa

que o mesmo executa é o controle de versão de dependências, isto é, um pacote pode utilizar determinada versão de outro, e através de um arquivo chamado *lockfile* é obtida determinada versão da dependência de forma que o pacote não encontre problemas de outras versões da dependência. A utilização do NPM em projetos Node é algo indispensável e de grande utilidade. Desta forma, este foi utilizado para o controle de bibliotecas e organização do projeto (Sobre o Node Package Manager 2020).

2.2.3 NoSQL

Estabelecido que o sistema terá como base uma API em Node.JS, foi necessário escolher um banco de dados. Pensando na melhor integração com o Node.JS, a possibilidade de escalar para uma massiva quantidade de dados e a compatibilidade com chave-valor proveniente dos objetos JSON; foi feita uma análise dos bancos NoSQL.

O NoSQL é uma categoria de banco de dados que surgiu com o objetivo de atender aos requisitos do gerenciamento de grande volume de dados, sem estrutura definida e que necessitam ser disponibilizados rapidamente e preparados para crescer ainda mais. Esta categoria traz algumas características como: (LÓSCIO; OLIVEIRA; PONTES, 2011).

a) Evitamento de Complexidade desnecessária;

b) Escalabilidade Horizontal;

c) Esquema Flexível ou Ausência de Esquema: A ausência ou flexibilidade de esquema trás vantagens como a alta escalabilidade e a oportunidade de mudar ou adicionar campos inesperados. Há a desvantagem da falta de garantia de integridade dos dados;

d) Simples Acesso aos Dados: Com o foco em velocidade é essencial que a entrega de dados ao sistema seja feita de forma simples e enxuta, o modelo NoSQL oferece os dados como interface e facilita o acesso e utilização de tal;

e) Modelo Chave-Valor: O modelo simples torna prático o armazenamento de dados, principalmente quando se trata de algo que não necessita se relacionar com outros valores e independe do tamanho do valor.

2.3 Trabalhos Correlatos

Se tratando de outros programas que em sua essência realizam a análise extração de dados foram encontradas boas alternativas, mas nenhuma que contemplasse todos

os objetivos que foram postos aqui. A seguir serão apresentadas essas alternativas e definido porque ela não se encaixa no objetivo deste projeto.

- **PDF Tables**, **Free PDF Convert** e **Tabula** São boas alternativas, grátis mas que realizam apenas a extração de dados tabulares e geram arquivos *.csv*. Não satisfazendo os quesitos de ser genérico e ter a capacidade de se acoplar a outra aplicação;
- **LA PDF** é uma alternativa desenvolvida por biomédicos que buscavam extrair dados de artigos, ela ainda está em desenvolvimento, porém o foco dela é a extração de dados somente de artigos. Assim, ela está engessada a este tipo de layout e apresenta as mesmas incapacidades das alternativas anteriores, quanto aos objetivos deste projeto;
- **ExtractPDF** é uma alternativa online e grátis que realiza a conversão do PDF em arquivos de texto e imagens. Em testes feitos neste site, foi notado que a extração de texto é feita em blocos quebrados que nem sempre estão na ordem correta de leitura. Não torna possível a utilização em outras aplicações e não possibilita a seleção de quais dados serão extraídos;
- **Doc Parser** apesar de ser uma alternativa paga, pelo que foi apresentado realiza a extração dos dados dada as coordenadas e página, resultando em um arquivo de texto com os dados extraídos;
- **iText PDF** Uma ótima alternativa que pode ser utilizada com aplicações em Java e C++, possui uma API e uma interface amigável caso necessário, e diversos modos de configuração para que se atinja a extração do dado necessário. Porém é uma alternativa paga.

3 MATERIAIS E MÉTODOS

Esta seção descreve as etapas do desenvolvimento da API, assim como a aplicação e utilização das ferramentas e suas tecnologias. Este desenvolvimento foi realizado utilizando a generalização de conceitos de engenharia de software, com o objetivo de um baixo acoplamento e alta coesão assim também como a reutilização de código. As etapas de desenvolvimento se deram através dos estágios necessários para a construção de uma API REST e foram organizadas utilizando a metodologia ágil Kanban.

3.1 Bibliotecas e Ferramentas

A escolha das bibliotecas e softwares deste projeto, devem-se a pesquisas que tiveram como foco:

- Suporte dos desenvolvedores: Dada a necessidade de se ter uma ferramenta que seja confiável e atualizada;
- Alta utilização pela comunidade: Já que a maioria destas ferramentas são de código livre e podendo a comunidade submeter melhorias e realizar a revisão do código que está sendo alterado na mesma;
- Compatibilidade a outros componentes e estrutura do projeto: Um dos objetivos deste software é que ele seja mutável, para que de acordo com as mudanças que podem ocorrer em outras tecnologias e estruturas de documentos PDF, esteja apto a acompanhar tais alterações.

3.1.1 PDF.JS - v2.2.228

O PDF.JS, criado em 2011 pela Mozilla Foundation, é uma biblioteca amplamente utilizada, e se encontra presente na maioria dos websites que fazem qualquer tipo de manipulação e exibição de PDF. Esta foi criada como uma extensão do navegador Firefox, com a intenção de renderizar arquivos PDF, de forma rápida e segura, no navegador do cliente. Apesar da migração de uma biblioteca específica para o navegador da Mozilla, seu formato de renderização permanece o mesmo, onde ela acessa a estrutura do arquivo PDF, coleta o conteúdo e a formatação no *Body*, realiza a conversão para HTML e CSS de cada objeto encontrado e com a tabela XREF o arquivo é remontado no navegador do cliente como um PDF dentro da página web.

Com a evolução das tecnologias e a busca contínua por melhor performance, esta biblioteca conta com centenas de funções e diferentes abordagens para e leitura das mais

variadas formas de PDF. Utilizando o resultado destas funções, é possível realizar extrações para coleta de informações encontradas no arquivo, informações como o texto puro, posição de elementos na página, fonte utilizada, cor e vários outros tipos.

Como o objetivo deste sistema está diretamente ligado a extração dos dados e o foco principal desta biblioteca é a renderização do arquivo no navegador, boa parte das funções mais pesadas nesta biblioteca não são necessárias.

3.1.2 Extract - v0.1.3

Convenientemente, há um módulo criado pela comunidade que encapsula somente a parte e extração de dados contida na biblioteca e entrega de forma leve e prática todas as funcionalidades necessárias para uma boa extração do conteúdo do arquivo PDF.

Fundamentalmente, a leitura acontece de forma que, dado o caminho do arquivo e um objeto de opções, são extraídos metadados do arquivo como o autor, o software utilizado para salvar o arquivo, o software utilizado para gerar o PDF, última data de edição entre outros. Junto ao objeto de metadados é retornado um vetor contendo um objeto JSON para cada página analisada, este objeto contém todas as informações básicas necessária para análise do conteúdo como é possível ver **Figura 3**.

Figura 3 – Objeto JSON após extração de informação de um PDF através da biblioteca Extract.


```
[
  "meta": {
    "info": {
      "PDFFormatVersion": "1.5",
      "IsLinearized": false,
      "IsAcroFormPresent": false,
      "IsXFAPresent": false,
      "IsCollectionPresent": false,
      "Author": "Usuário do Microsoft Office",
      "Creator": "Writer",
      "Producer": "LibreOffice 6.2",
      "CreationDate": "D:20190911193601-03'00'"
    },
    "metadata": null,
    "contentDispositionFilename": null
  },
  "pages": [{ "pageInfo": [Object], "links": [], "content": [Array] }],
  "pdfInfo": undefined,
  "filename": "./TCC-Adson-CIC-Libre.pdf"
},
[
  {
    "x": 231.3,
    "y": 96.30076377952798,
    "str": "ADSON SANTOS CARDOSO",
    "dir": "ltr",
    "width": 161.112,
    "height": 12,
    "fontName": "g_d0_f1"
  },
  {
    "x": 105.3,
    "y": 303.300763779528,
    "str": "ESTUDO METODOLÓGICO DE MINERAÇÃO DE DADOS EDUCACIONAIS",
    "dir": "ltr",
    "width": 412.9559999999999,
    "height": 12,
    "fontName": "g_d0_f1"
  }
]
```

Após extrair essas informações o próximo passo foi a filtragem deste conteúdo.

3.1.3 Extract By Coord

Determinado o objetivo de extrair a informação em um espaço da página que é definido por coordenadas X e Y, foi feita uma pesquisa de formas práticas para se realizar a filtragem do conteúdo, por tais coordenadas. Assim, foi encontrada uma pequena biblioteca no GitHub, com a licença MIT, que realizava uma abordagem similar à necessária para a próxima etapa da extração dos dados.

Esta biblioteca oferece duas funcionalidades: a primeira encapsula a extração feita pela biblioteca Extract, com o intuito de descartar informações desnecessárias para a análise, como a rotação, escala, tamanho da página, e evidenciar o software responsável por gerar o arquivo PDF. Além disso, os objetos na página são ordenados da esquerda a direita e de cima para baixo, e selecionado apenas os dados de coordenadas e string. Desta forma é retornado um objeto JSON enxuto, contendo apenas as informações necessárias e pronto para ser recebido como entrada da função de filtragem por área.

Esta função é responsável por extrair o conteúdo, ela recebe o nome do programa responsável por gerar o arquivo PDF, a importância deste valor é decorrente da forma que diferentes programas atribuem diferentes formas de espaçamento entre as strings. Por exemplo o Microsoft Word realiza o espaçamento dos objetos atribuindo vários objetos de espaço, além disso são recebidos o objeto da página a ser extraída, e as coordenadas X e Y da área como objeto JSON. Ao ser executada, são observadas as coordenadas de cada objeto dentro da página, estando no alcance das coordenadas recebidas, sua string é extraída e tratada. Desta forma é retornada uma string correspondente ao conteúdo textual que se encontra naquela região.

Esta biblioteca foi alterada para que o texto recebido fosse entregue com uma formatação correta, devido a diferença de espaços causada pelo software que gerou o PDF, além disso foi feita uma melhor filtragem pelo sistema de coordenadas e a função de extração passou a aceitar mais de uma página.

3.1.4 *RegEx* - Expressão Regular

O *RegEx*, derivado da expressão regular definida na teoria da computação, entrega uma forma de se identificar cadeia de caracteres, dada uma expressão regular em linguagem formal. Ou seja, dada uma expressão é possível encontrar cadeias que aceitem tais condições. A linguagem JavaScript carrega todo um grupo de funções que auxiliam na utilização de expressões regulares, assim não é necessária nenhuma biblioteca adicional.

Como forma de limitação ou busca de valor específico em determinada string, são utilizadas expressões regulares que tanto removem trechos que podem afetar a busca de um

valor, quanto encontram um valor específico após determinada cadeia de carácteres. Como por exemplo no caso de extração do nome de um orientador. Este nome pode se encontrar depois da palavra "Orientador"e antes da palavra "Coorientador", assim é definido um padrão *RegEx* que encaixe com o valor que se encontra entre as duas palavras.

A filtragem com o *RegEx* assegura o conteúdo encontrado e delimita a informação buscada, no caso de ocorrer um bloco inesperado dentro das coordenadas.

3.1.5 JWT - *JSON Web Token* (v8.5.1)

Com a necessidade de se ter uma forma de autenticação segura e prática, foi optado pela utilização do *JSON Web Token*. Comummente, o desenvolvedor realiza a implementação de seu próprio método de autenticação, porém, segundo a *Open Web Application Security Project* (OWASP), uma organização sem fins lucrativos que trabalha em prol da segurança na web, buscando falhas e formas de combater as mesmas, a "Quebra de Autenticação e Gerenciamento de Sessão" é um dos maiores e mais recorrentes problemas em aplicações web. A falta de uma boa criptografia e métodos seguros de envio e recebimento de dados ocasiona oportunidades para que usuários mal-intencionados consigam invadir contas e ter acesso a informações sigilosas.

O JWT é um padrão aberto (RFC 7519), que define de forma compacta, independente e segura, um método de transmitir informações como objeto JSON. Sendo uma forma segura, com bibliotecas para a maioria das linguagens utilizadas na web e suporte nativo do JSON nos navegadores modernos, o JWT se tornou a escolha ideal como método de autenticação.

Utilizando o algoritmo HMAC *(Hash based Message Authentication Code)*, a informação é encriptada com um segredo que apenas o lado do servidor contém, (também podem ser utilizados pares de chave pública/privada através da encriptação por RSA ou ECDSA), e passada como um *header* HTTP. Desta forma, quando o usuário realiza uma requisição que necessite de validação do usuário, o servidor recebe a *header* e realiza a chamada de um método de verificação do *token* contido na *header* (JWT - JSON Web Token 2020).

3.1.6 MongoDB (v4.0.13)

O MongoDB é um banco de dados não relacional, onde os dados são armazenados em documentos JSON, por isso é dito que este é um banco orientado a documentos. Desta forma não há necessidade da criação de tabelas e colunas na fase de modelagem do banco, assim há uma liberdade maior para que o documento que será salvo represente apenas as informações necessárias e suas relações não sejam pré definidas.

Esta flexibilidade é de grande ajuda quando se trata de uma aplicação que pode

mudar a organização e o tipo de informação a ser salva, diferente de um banco relacional, onde toda a sua estrutura deveria ser mudada para que se adequasse a estas mudanças. No caso dessa aplicação, o usuário irá definir quais informações serão extraídas do PDF e assim o objeto JSON com as informações será montado, além disso, o usuário também tem a liberdade de adicionar ou remover um campo de informação a ser extraída, ou adicionar um novo sem afetar a estrutura do banco.

A escolha do MongoDB entre os bancos não relacionais foi feita devido a sua alta performance, facilidade de manipulação em relação a sua infraestrutura, popularidade e principalmente a biblioteca Mongoose.

3.1.6.1 Mongoose (v5.7.5)

A manipulação do Mongo foi feita através da biblioteca Mongoose, onde a modelagem de dados é baseada em esquemas. Ao realizar uma consulta ao banco de dados, através do mongoose, é realizada conversão do dado para um objeto JavaScript, assim quando recuperada uma informação do banco, sua manipulação se torna algo simples e orgânico no fluxo de desenvolvimento. Também é possível realizar validações através da declaração do *schemas*, isso impõe mais uma camada de segurança entre o usuário e a API.

3.1.7 Express (v4.17.1)

O Express é um *framework* web baseado no modulo de *http* do Node.JS, ele dispõe de vários recursos robustos e fornece muita agilidade e desempenho sem ofuscar as características do Node.JS. Além disso ele proporciona uma estrutura similar ao MVC para a organização do projeto. De forma resumida, ele entrega maneira práticas de lidar com a comunicação da aplicação com a web e fornece a possibilidade de adicionar diversos *middlewares* a nível de aplicação. Algumas das principais funcionalidades oferecidas são (Mardan 2014):

- Processamento de requisições HTTP;
- Processamento de *cookies*;
- Controle de sessão;
- Organizar rotas, de acordo com a URL e o método HTTP utilizado;
- Determinar o *header* de resposta apropriado a cada tipo de dado.

3.1.8 *Multer* (v1.4.2)

O *Multer* é um *middleware* de rota que é responsável por lidar com requisições POST do tipo *multipart/form-data*, este tipo de requisição é utilizado para o envio de arquivos. Quando presente em uma rota POST, ele irá observar se é do tipo *multipart/form-data*, caso não seja, a requisição será ignorada. São definidas algumas configurações para lidar com o arquivo a ser recebido, como destino do arquivo, nome, tamanho máximo e mínimo, tipo de arquivo que deve ser aceito. Desta forma ele foi utilizado para receber os arquivos PDF.

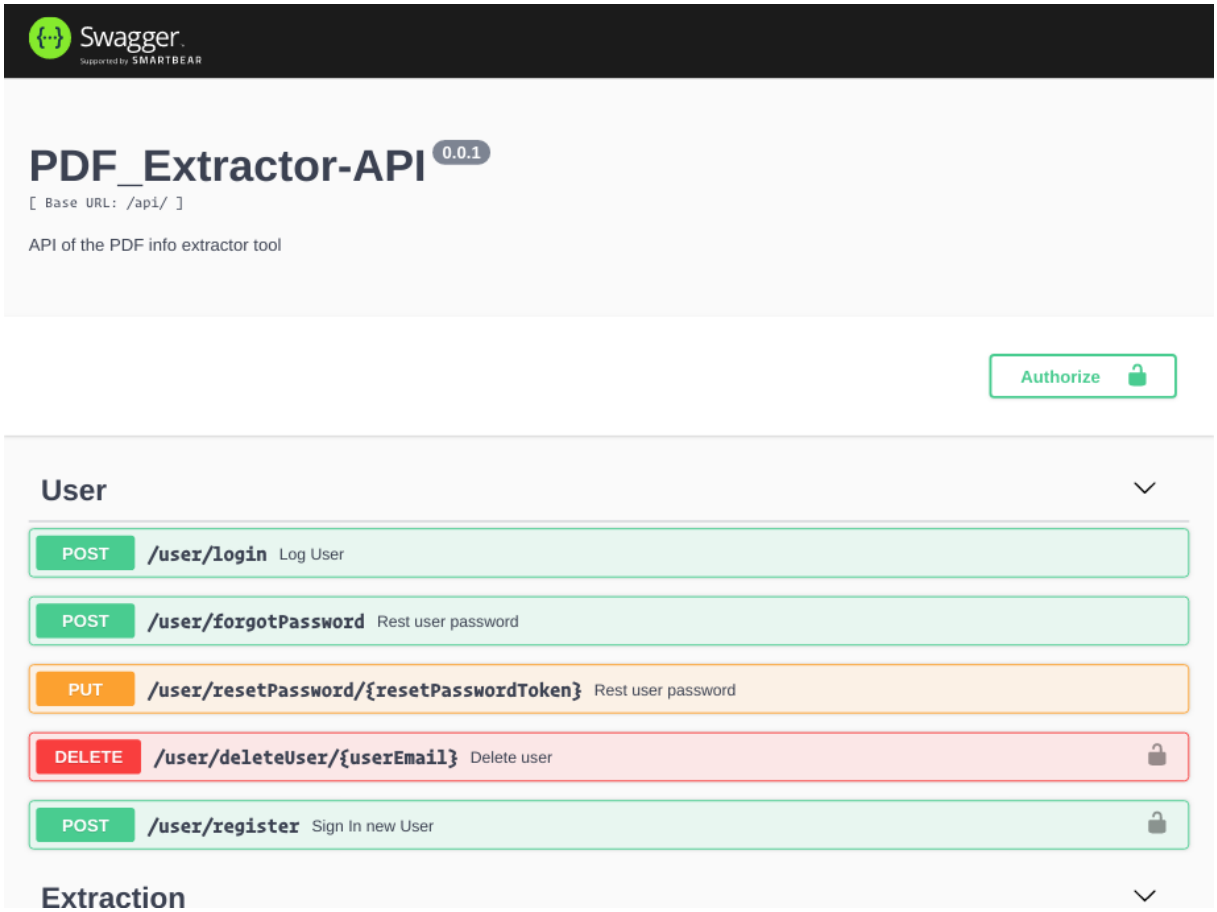
3.1.9 *Swagger-JS* (v3.4.0) e *Swagger-UI* (v4.1.2)

O *Swagger* é um conjunto de ferramentas, construídos em torno da especificação do *OpenAPI*. A organização *OpenAPI* define um padrão de descrição para API's de forma que tanto humanos quanto máquinas consigam compreender e utilizar todas suas funcionalidades sem necessidade de acessar o código fonte (OPENAPI, 2017).

O *Swagger-JS* é uma biblioteca, que por meio de anotações no código fonte, gera uma documentação onde podem ser definidas entidades, formato de requisições, definição de rotas, respostas esperadas de determinada rota e todo tipo de capacidade da API.

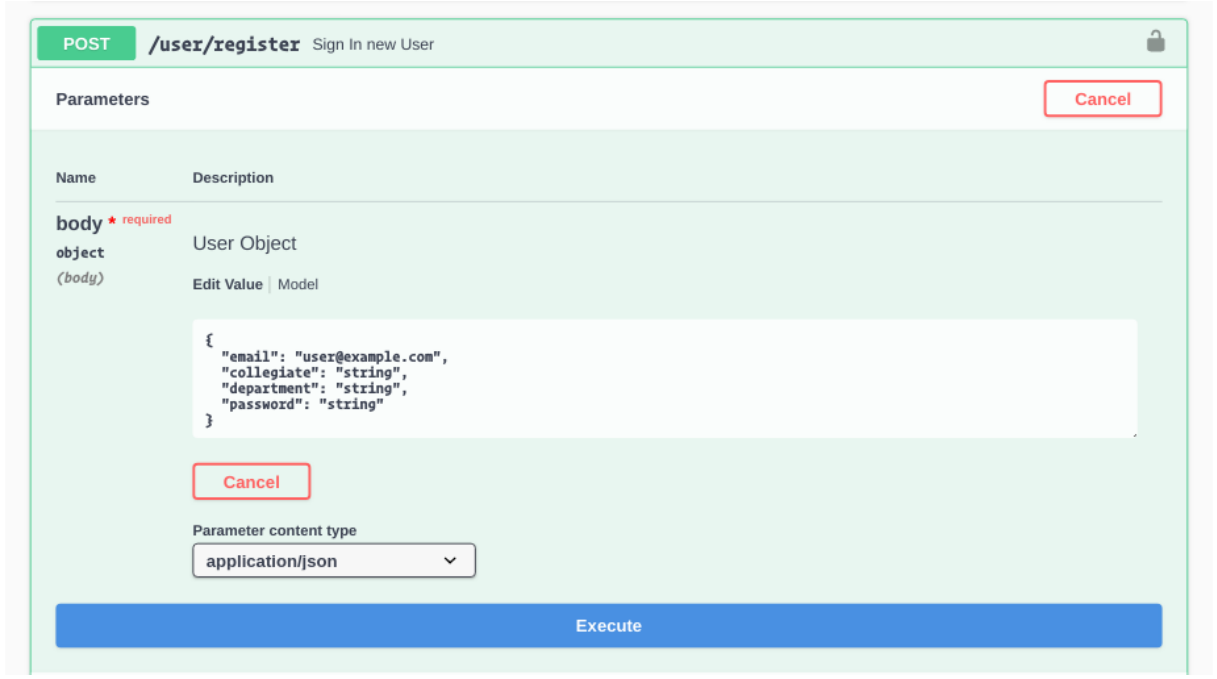
Uma das ferramentas do Swagger é o *Swagger-UI*, este adiciona uma rota a aplicação onde estará disponível uma interface possuindo toda a documentação, como representado na **Figura 4**, definida através das anotações citadas e juntamente é fornecido a possibilidade de realizar requisições diretamente desta interface como pode ser visualizado na **Figura 5** e consultar os dados dos modelos definidos no banco de dados como apresentado na **Figura 6**.

Figura 4 – Página Inicial - Swagger UI.



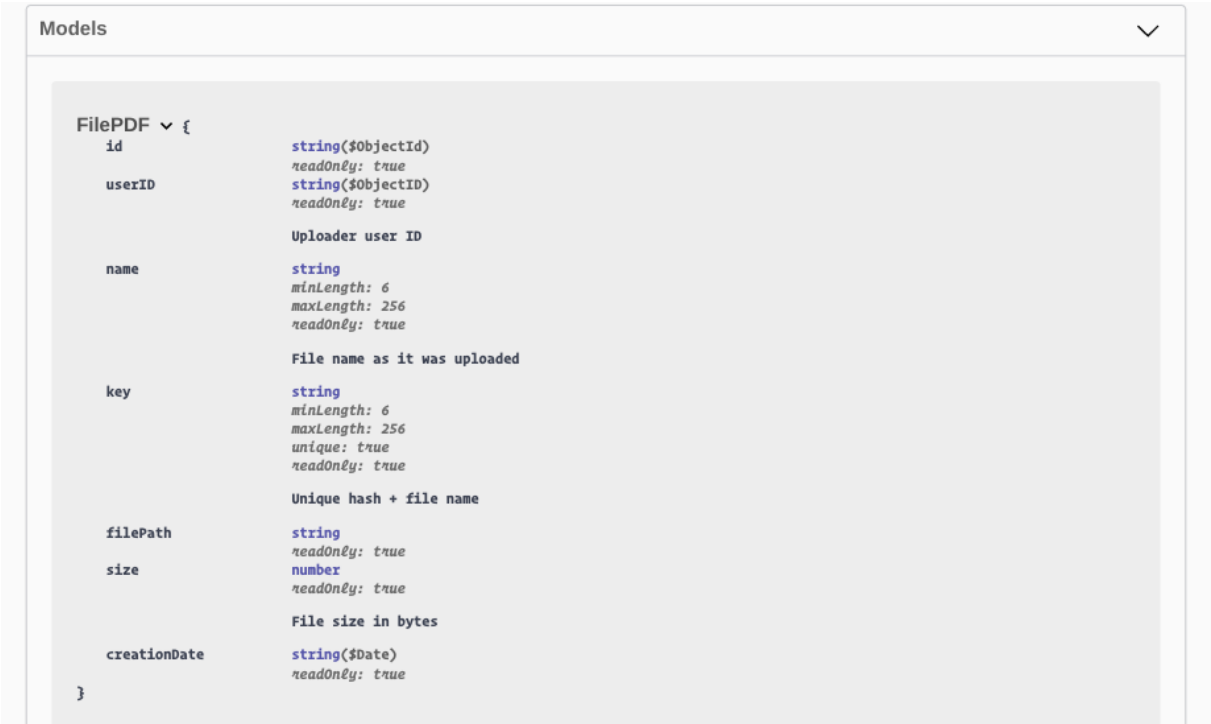
Fonte: Autoria Própria

Figura 5 – Exemplo de Requisição - Swagger UI.



Fonte: Autoria Própria

Figura 6 – Exemplo de Requisição - Swagger UI.



Fonte: Autoria Própria

3.2 Metodologia e Desenvolvimento

Neste tópico, serão explicados como cada ferramenta foi utilizada. Os detalhes mais específicos podem ser encontrados no repositório deste projeto no GitHub, publicamente aberto e é possível enviar mudanças para que sejam avaliadas e incorporadas no código. Também será descrito como foi modelada a estrutura deste sistema e dividia as etapas de desenvolvimento.

3.2.1 Kanban

Para a organização e definição das tarefas que objetivam a completude deste projeto foi escolhida a metodologia ágil Kaban. Criado pela Toyota nos anos 70, o Kanban (que pode ser traduzido como Cartão), sinalizava a disponibilidade para se trabalhar em uma peça, quando se fazia necessário para outras etapas da produção. Dessa forma havia uma maior organização na ordem de trabalho, não havia desperdício de peças e através dos cartões era possível ver o progresso geral.

Com estas vantagens é possível utilizar o Kanban no desenvolvimento de software e assim trazer mais vantagens ao fluxo de trabalho para a programação. Aplicando esta metodologia é feito um quadro consistente de três colunas, uma de tarefas **A fazer**, em seguida as tarefas **Em andamento** e finalizando com as tarefas **Concluídas**.

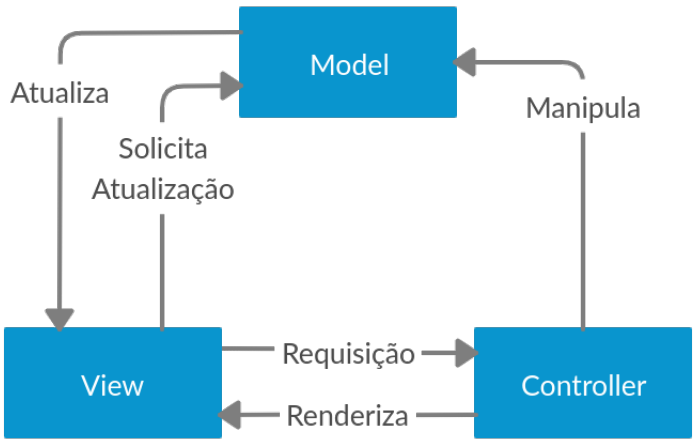
O GitHub oferece uma implementação do Kanban, diretamente no repositório para que outros colaboradores possam interagir e a visualização e utilização sejam práticas.

Foram separados por ordem de prioridade as tarefas necessárias para atingir a completude da API, e adicionadas nas lista de A fazer no *kanban*, foram priorizadas aquelas que eram necessárias para o funcionamento e teste da API, assim foi feito uma por vez até que houvessem mais tarefas que impedissem o funcionamento de tal.

3.2.2 MVC - *Model View Controller*

A estrutura deste sistema foi elaborada se baseando no padrão MVC, que tem como objetivo separar a modelagem do banco de dados, a interface do usuário e o controlador dos dados. Se baseando neste padrão foi adicionada mais uma camada chamada *services*, assim é obtido mais um nível de abstração. Na **Figura 7** é possível ver uma representação clássica do MVC e na **Figura 8** a representação utilizada neste projeto.

Figura 7 – Exemplo de Requisição - Swagger UI.

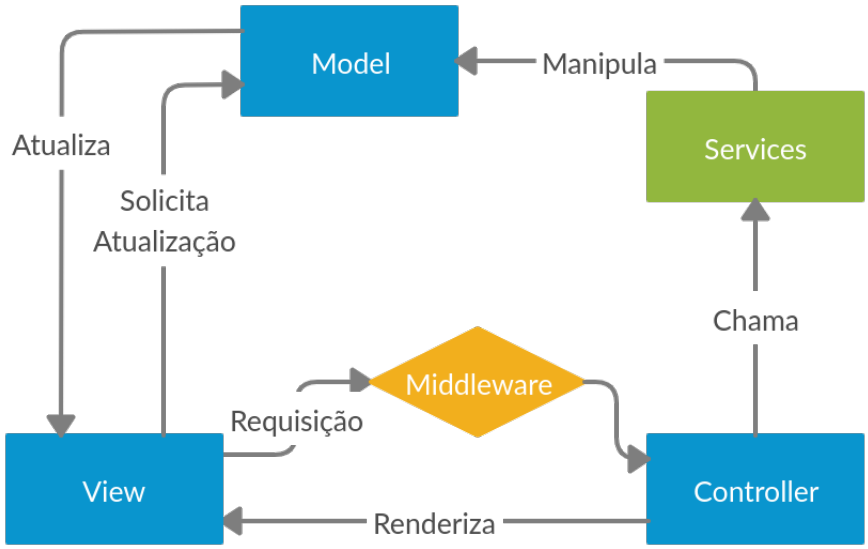


Fonte: Pressman 2005, Adaptado pelo Autor

O objetivo desta camada é retirar as regras de negócio do *controller*. Desta forma, o *controller* possuirá apenas a responsabilidade de administrar o fluxo de dados e indicar qual o *service* que deve lidar com a requisição. Dentro desta camada estão concentradas as funções que devem possuir apenas uma única responsabilidade, e ocorrendo a necessidade de uma ação complexa deve ser feita uma função responsável por chamar funções menores.

Duas principais consequências são alcançadas com esta abstração, a primeira é o baixo acoplamento, já que a lógica de negócio não está mais presente no *controller* e sim uma referência a função que irá executar essa lógica. Caso haja necessidade de modificar algo na lógica, só será necessária modificar a função em questão, ou seja, não há dependência de tal para com o resto do sistema. A segunda consequência é a coesão, na camada de *services* é feita uma divisão por componentes, cada componente tem uma

Figura 8 – Exemplo de Requisição - Swagger UI.



Fonte: Autoria Própria

responsabilidade específica na regra de negócios, e em cada componente estão divididas funções com apenas uma única responsabilidade.

Neste projeto foram desenvolvidos seis componentes no módulo de *services*, a seguir serão apresentados estes módulos e suas finalidades:

1. ***authTokenService*** - Possui as responsabilidades de atribuir um *JSON Web Token* ao usuário no momento de *login*, e verificar se o *token* é valido ao realizar requisições que necessitem de autenticação;

2. ***cryptService*** - Similar ao *authTokenService*, porém possuindo as responsabilidades de realizar o *hash* da senha do usuário e a verificação se determinada senha é válida para aquele usuário;

3. ***extractService*** - Aqui estão as funções que realizam a extração de dados. Dado o arquivo e o conjunto de parâmetros para a extração dos dados, são aplicadas as devidas configurações, provenientes destes parâmetros, e então são extraídos os dados e montado um objeto JSON com os resultados;

4. ***mailService*** - Este *service* tem a única finalidade de enviar o email de recuperação de senha, ao usuário, quando solicitado;

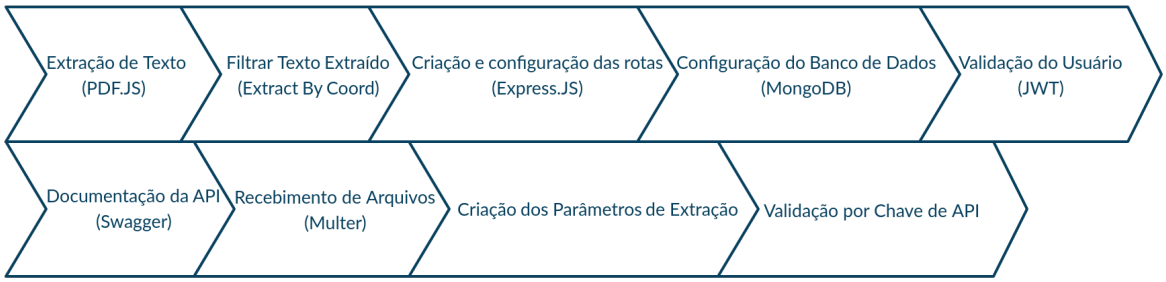
5. ***filesServices*** - Neste *service* é feito o recebimento do arquivo, proveniente do usuário, coletado dados como nome, tamanho, usuário que realizou o envio, e então renomeado com um *hash* para que não ocorra problemas com duplicação;

6. ***userService*** - Aqui estão concentradas várias funções relacionadas ao usuário, sendo elas cadastro, acesso, gerar token para criação de nova senha, criar nova senha e desabilitar usuário.

3.2.3 Desenvolvimento da API

Nesta seção será abordado como foi realizado o desenvolvimento da API desde o começo com o estudo das ferramentas, escolhas, modificações e como as ferramentas foram aplicadas no desenvolvimento do código. Na **Figura 9** pode ser visualizada as etapas realizadas para a conclusão da API.

Figura 9 – Organograma das etapas de desenvolvimento.



Fonte: Autoria Própria

Após definido os objetivos e o que precisava ser feito para alcança-los. Foi realizado o estudo para definir qual plataforma/linguagem seria utilizada. Conforme apontado os benefícios anteriormente, foi escolhido o Node.JS, levando em conta também a habilidade do desenvolvedor.

Tendo escolhido o Node.JS, foi necessário buscar ferramentas compatíveis com o mesmo. A primeira ferramenta estudada foi o PDF.JS, pois para viabilizar o projeto deveria haver uma forma rápida e confiável para a extração dos dados. Muitas outras bibliotecas com este mesmo objetivo foram encontradas e testadas. Porém através destes testes foi definido que o PDF.JS realizava a extração que mais se aproximava do desejado. Havendo aspectos da extração que necessitavam melhora, foi encontrado a biblioteca que encapsulava funções do PDF.JS, o Extract. Como o PDF.JS foi criado com o foco de remontar o arquivo PDF em HTML e CSS para visualização no navegador, esta biblioteca possuía uma carga desnecessária e trazia algumas ferramentas irrelevantes ao tipo de extração em questão.

Provada a viabilidade e funcionamento da extração, foi inciada a etapa de filtragem (até então era extraído todo o conteúdo da página). Como discutido anteriormente, o tipo de filtragem que melhor funcionaria para a extração de qualquer tipo de documento PDF seria orientada a localização. Dado que o foco da extração em massa é ser guiado

pela formatação da página. A extração até esta parte continha muitos dados e nenhum tipo de filtragem, então através de pesquisas no NPM foi encontrada uma biblioteca que realizava a extração utilizando coordenadas da área desejada e retornava o conteúdo que se encontrava ali. Contudo, devido aos diferentes softwares que realizam a montagem do PDF, o texto encontrado continha espaços e quebras de linha indesejadas e que não condiziam com o que era visualizado no documento.

A partir daí foi iniciada a modificação a biblioteca Extract. Foram analisados os métodos de montagem utilizados pelos softwares mais populares e, definidos os casos onde ocorria o erro de formatação. Então foi desenvolvida uma função que realizava o tratamento do conteúdo, dado o software utilizado na montagem do arquivo. Além disso foi necessário adaptar a função responsável por receber o objeto dá página do PDF para que também funcionasse com mais de uma página.

A próxima etapa foi fundamentada na criação e configuração das rotas, o estudo para a decisão de qual framework seria utilizado nesta etapa foi simples, tendo em vista que o ExpressJS é o framework mais baixado no NPM (Sobre o Node Package Manager 2020) e com mais visibilidade no GitHub. A partir daí, ao comparar seus benefícios com outros frameworks ficou claro que esta seria a melhor escolha. Decidido que a próxima etapa seria a criação do CRUD de usuário, veio a necessidade do banco de dados.

Como dito previamente, um banco noSQL se adéqua melhor a necessidade do sistema, considerando seu crescimento e a necessidade de uma coleção livre de esquemas. Pois o conteúdo extraído será definido pelo usuário. Dentre os bancos noSQL, o MongoDB se destacou em performance e sua organização dos dados em objetos JSON se encaixou perfeitamente com as necessidades.

Havendo a validação do usuário, se fez necessário encontrar uma forma de garantir que o mesmo estava autenticado no sistema, e para isso foi feito o estudo de tecnologias de validação. Similar ao caso do ExpressJS, foi encontrado uma tecnologia que se destacava de suas concorrentes, o JWT. Desta forma, o usuário ao se logar no sistema recebe uma *header* contendo um *token* único e criptografado. E para que ele possa realizar certas requisições ele deve possuir um *token* válido.

Acompanhando o crescimento da API, foi necessário iniciar o processo de documentação. Procurando fugir do moroso método clássico de documentar apenas descrevendo cada funcionalidade e suas propriedades, foi buscada uma ferramenta para que isso fosse feito de forma mais rápida e obtendo um melhor resultado. Assim foi encontrado *framework* de documentação Swagger. Este contém uma grande base de usuários e adotado por grandes empresas como a Microsoft e a National Geographic. Além de disponibilizar as funcionalidades da API para que humanos possam ler e testar, também disponibiliza de uma interface para outras aplicações que sigam o padrão estabelecido pela The Linux Foundation.

Desta forma, a utilização da API durante o desenvolvimento, foi feita utilizando a página fornecida pelo Swagger, em vista da sua facilidade e documentação apresentada.

O próximo passo foi poder receber um arquivo através de um requisição da API, e novamente, estudando tecnologias atuais, e bem utilizadas, foi encontrado o Multer. Com uma configuração simples, este *middleware* foi adicionado a uma rota específica para o recebimento de arquivos, e todo o auxílio necessário para o registro deste arquivo no servidor foi feito com sucesso. Todos os cuidados como, definir o tipo de arquivo que deve ser recebido, limite de tamanho, renomear para que não haja duplicatas, foram feitos nesta configuração do Multer. Então o arquivo é recebido, se dentro da conformidade ele é aceito, renomeado com um *hash* na frente, e todas suas informações são armazenadas no banco de dados.

Como o intuito é que a extração seja definida pelo usuário foi estabelecido que o mínimo que se precisa para extrair alguma informação de um documento é o número da página e as coordenadas definindo a área onde se encontra a informação. Para que o usuário possa identificar de forma prática o que foi retornado, ele deve adicionar um título, e para que ele seja capaz de filtrar o texto extraído ele pode adicionar uma *RegEx*, e assim será retornado o resultado da expressão. Por fim, foi necessário acrescentar uma forma de aplicar este conjunto de parâmetros ao arquivo que o usuário deseja ter as informações extraídas. Para isso foi adicionado um campo de identificação do arquivo. Isso quer dizer que no momento de envio do arquivo, deve ser passada uma string de identificação, então será feita uma busca no banco de dados por todos os Parâmetros de Extração com esta identificação, e estes parâmetros que serão utilizados para a extração.

Assim foi desenvolvido um CRUD de Parâmetros de Extração, o objeto referente a este conta com os seguintes campos:

- ***extractionTitle*** - Contém o título que irá corresponder a informação extraída.
- ***page*** - Corresponde a página onde se encontra a informação desejada.
- ***coordinatesStart*** **e** ***coordinatesEnd*** - Coordenadas X e Y da área a ser extraída.
- ***RegEx*** - Expressão regular para um filtragem mais aprimorada do trecho extraído. Este campo é opcional, caso não tenha seu valor preenchido, todo o conteúdo encontrado na área será retornado.
- ***docType*** - Uma tag que será usada no momento de envio de um arquivo PDF. Ao enviar um arquivo deve ser passado um identificador, da escolha do usuário, junto a requisição. Todos os parâmetros que possuírem a tag correspondente a tal identificador serão utilizados para a extração dos dados.

Foi encontrado um obstáculo ao enviar um *RegEx* à API, devido a sua formatação o valor é descaracterizado no momento que chega ao servidor e também quando salvo no banco de dados. Visando contornar este problema, foi encontrada a alternativa de codificar o valor para base64, e utilizando métodos já implementados no Node.JS, converter o mesmo para uma string e da string para um *RegEx* no momento em que for utilizado.

Com o sistema sendo capaz de receber um arquivo e realizar a extração de dados em pontos específicos do documento, foi notada a necessidade de se realizar a busca de alguns dados nos casos onde a página na qual a informação se encontra não é exata. Assim surgiu a necessidade de uma função capaz de extrair informações de uma página não especificada. Ou seja, buscar a página e então realizar a extração. A solução encontrada consistiu em adaptar o objeto de Parâmetros de Extração para ser capaz de receber um intervalo de páginas, para que seja encontrada a página desejada o usuário deve adicionar palavras-chave que estejam contidas na página. Assim foi adicionado o campo de ***keyWords***, este campo recebe uma ou mais *RegEx's*, e quando encontrada uma página que retorne resultados destas expressões, isto significa que foi encontrada a página desejada. Logo, no momento da extração dos dados, caso o parâmetro contenha um intervalo de páginas, uma função será executada, buscando neste intervalo, uma página que corresponda aos *RegEx's* contidos no campo *keyWords*. Encontrada a página, a extração da informação ocorre da mesma forma explicada anteriormente.

Para assegurar que a criação de novos usuários seja controlada, foi desenvolvido um *middleware* para ser utilizado na rota de criação de usuário, o que faz obrigatório o envio de uma chave de API no momento do cadastro do usuário.

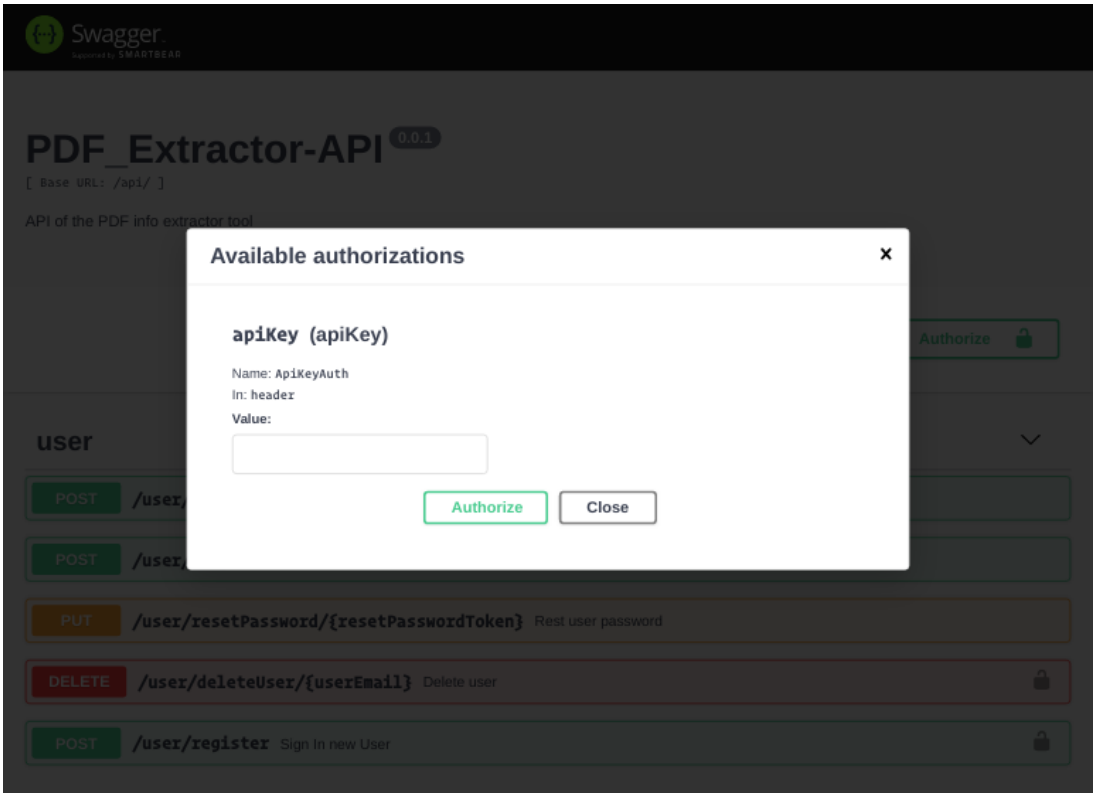
Somando as ferramentas até aqui apresentadas, e os meios encontrados para a criação desta API, foi obtido um ótimo resultado e sua empregabilidade é vasta. Tendo em vista que o estudo de tecnologias modernas não é algo simples, pode ser dito que toda escolha foi bem fundamentada e sua utilização bem descrita até o presente tópico. Conhecendo o que foi utilizado e como foi aplicado, é dado inicio aos testes e utilização da API.

4 RESULTADOS

A API pode ser encontrada em funcionamento em um ambiente de testes no link http://bit.ly/tcc-pdf. Suas funcionalidades de leitura encontram-se livres para serem acessadas, porém as funcionalidades que alterem o estado do banco de dados estão restritas aos usuários cadastrados. Apesar de estar em funcionamento e livre para ser acessada, este é apenas um ambiente de testes, e sua hospedagem no servidor do Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas (NBCGIB) está sendo configurada.

Para o controle destas utilidades existem duas camadas de segurança. A camada de chave de API, apresentada na **Figura 10** assegura que somente os administradores do sistema, em posse da chave, possam registrar e remover usuários da API.

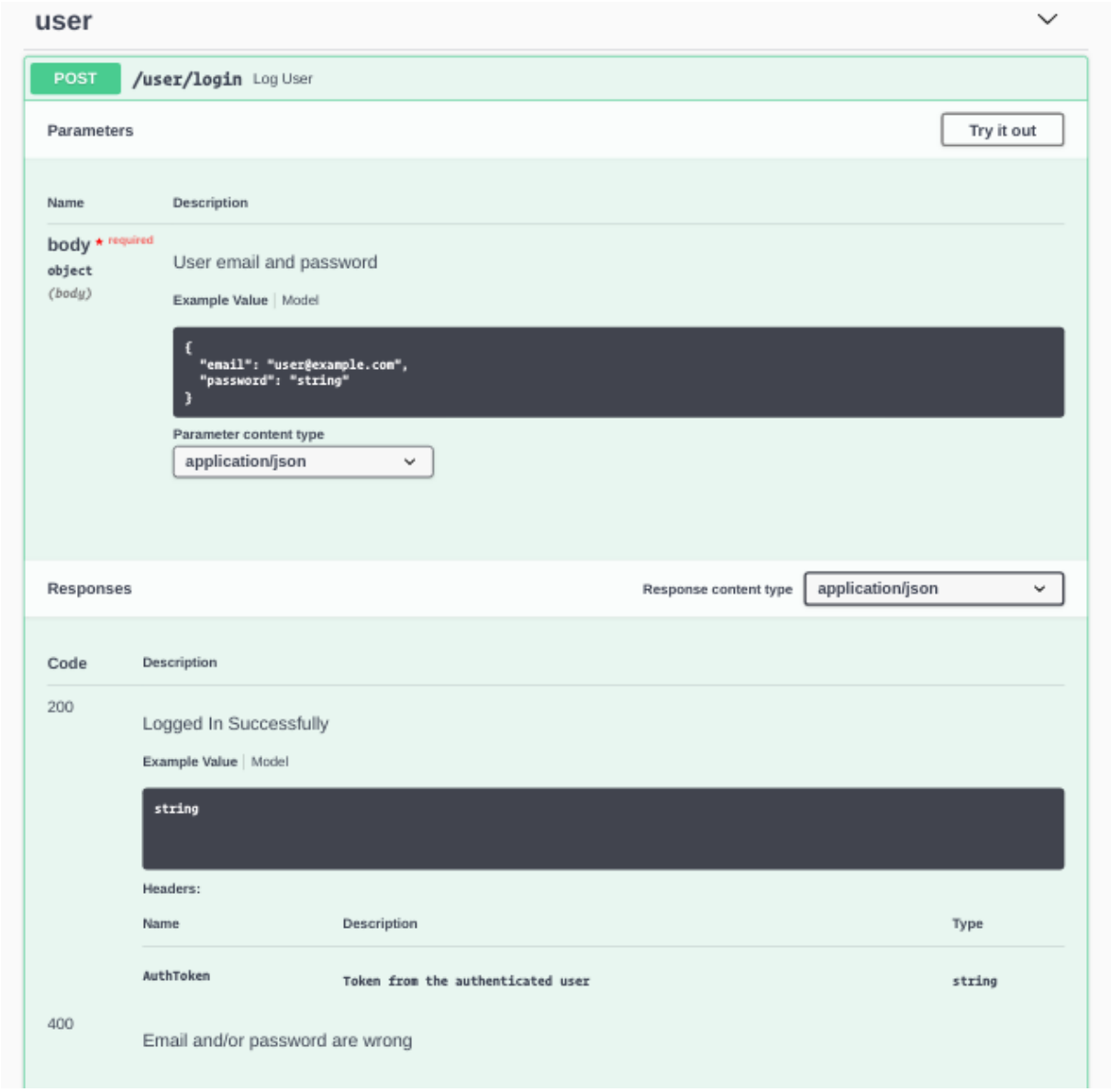
Figura 10 – Autorização por Chave de API - Swagger UI



Fonte: Autoria Própria

A camada de usuário garante a integridade do sistema, pois esta impede que usuários não autenticados tenham acesso a funcionalidades que alterem o estado do banco de dados. Na **Figura 11** é possível ver o método de login, e suas possíveis respostas.

Figura 11 – Login de Usuário - Swagger UI



Fonte: Autoria Própria

Para o categoria de funções do usuário, assegurando o acesso e contendo funcionalidades necessárias para o administrador do sistema apresenta as seguintes funcionalidades:

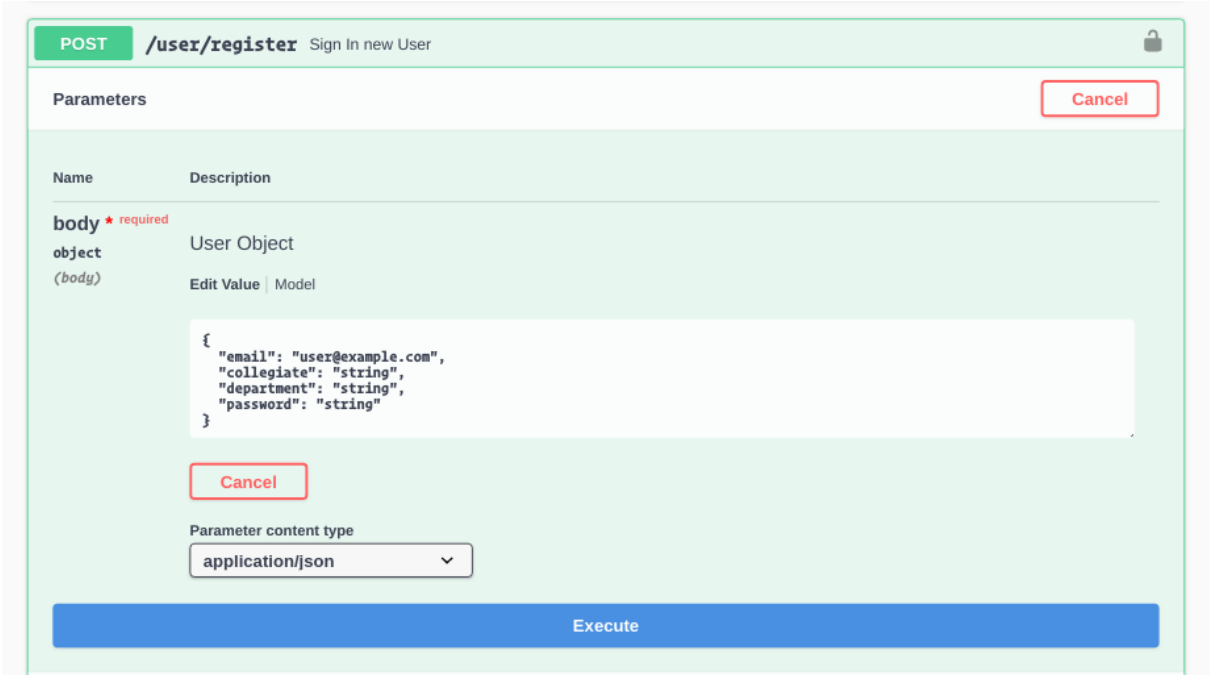
- **Login**: Como explicado no tópico 3.2.3, recebe e-mail e senha e retorna um *token* caso a autenticação o ocorra com sucesso.
- **Senha Esquecida**: Recebe o e-mail do usuário, sendo ele valido, é disparado um e-mail contendo um identificador único para a recuperação da senha.
- **Resetar Senha**: Continuando o fluxo da função anterior, recebe o identificador e uma nova senha, caso sejam válidos, a nova senha é atribuída ao usuário.

- **Remover Usuário**: Recebe um e-mail de usuário e a chave de API, caso sejam válidos, o usuário correspondente a aquele e-mail é desativado.
- **Registrar Usuário**: Recebe e-mail e senha do usuário a ser cadastrado. Também deve ser enviada a chave de API para validação do administrador. A coleção de usuários, no banco de dados, se encontra livre de *schema* devido aos diferentes tipos de usuários.

Para que o usuário tenha controle das configurações do conteúdo a ser extraído, foi criada a seção de Extração:

- **Registrar Parâmetros**: Conforme explicado anteriormente no tópico 3.2.3, esta requisição recebe um objeto JSON, com as informações necessárias para realizar um extração, e o *token* de usuário, como pode ser observado na **Figura 12**. Este objeto é salvo no banco de dados e utilizado ao realizar extrações.
- **Parâmetros Registrados**: Realiza uma consulta no banco de dados e retorna um objeto JSON com um vetor de parâmetros que já estão registrados.
- **Remover Parâmetro**: Recebe um ID de parâmetro e o *token* de usuário, sendo válidos, o parâmetro é removido do banco de dados.

Figura 12 – Parâmetros de Extração - Swagger UI



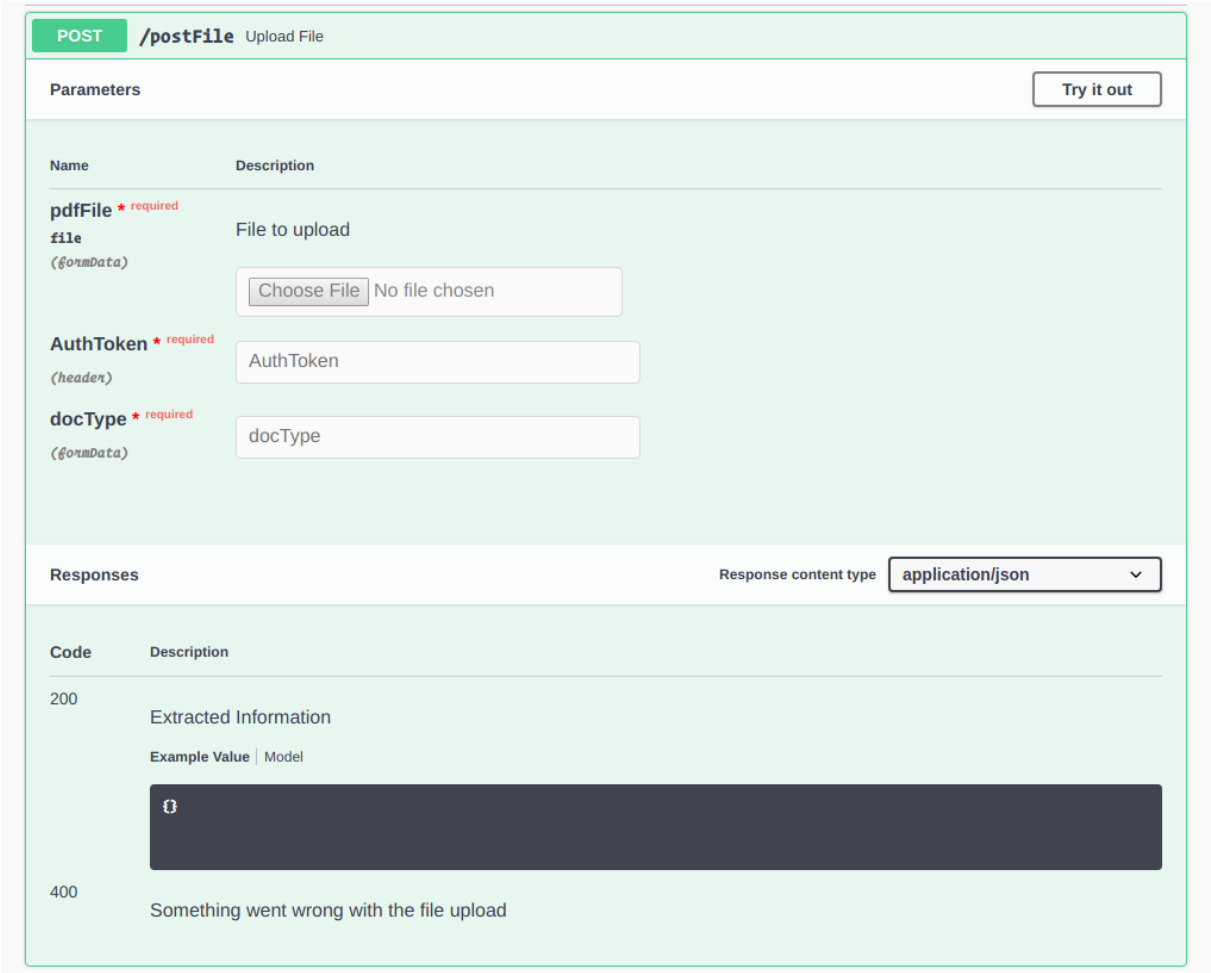
Fonte: Autoria Própria

Em relação a informação já extraída, são entregues as seguintes funcionalidades:

- **Informações Extraídas**: Consulta no banco de dados todas as informações extraídas e as retorna em um vetor de objetos JSON correspondente a cada arquivo, contendo as informações que foram extraídas de tal.
- **Remover Informação Extraída**: Recebe o ID correspondente a um objeto de informações extraídas, e um *token* de usuário, sendo válido, remove o objeto do banco de dados.

Por fim há a função responsável por receber um arquivo, o *token* de usuário e a identificação do arquivo, como pode ser visto na **Figura 13**. Ao ser recebido, são buscados os parâmetros que possuem o mesmo valor do arquivo enviado no campo *docType*, e assim serão extraídas as informações para cada parâmetro definido. A requisição é respondida com um objeto JSON contendo todas as informações extraídas, em caso de sucesso.

Figura 13 – Envio de Arquivo - Swagger UI



Fonte: Autoria Própria

Como pode ser notado nas imagens apresentadas neste capitulo, toda a API está documentada e pronta para uso na interface do Swagger, que pode ser encontrada no caminho */api/docs* (partindo do endereço raiz da aplicação). Neste endereço é possível

fazer a utilização da API e ter o entendimento do que cada função faz e o que deve ser retornado.

No curso de Ciência da Computação da UESC há um projeto em desenvolvimento para a criação de uma rede de egressos. Sendo o trabalho de conclusão de curso aprovado, este é o primeiro passo para o aluno se tornar um egresso. Desta forma foi notado que há uma demanda por informações referentes aos egressos que podem ser encontradas no arquivo PDF do trabalho, assim é possível extrair um espécie de ficha do egresso contendo nome do autor, orientador, título do trabalho, palavras-chave e resumo.

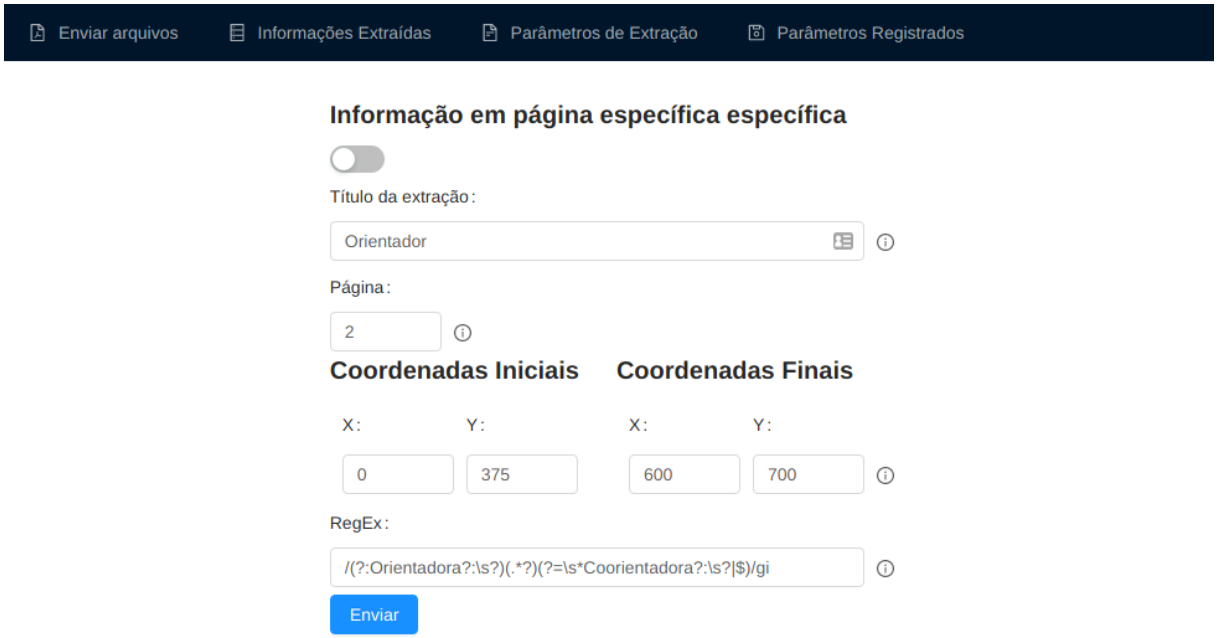
Com esta demanda e a necessidade de por a prova a API, foi desenvolvida uma aplicação *front-end* para a utilização do colegiado do curso.

Ao acessar a aplicação a tela de login é apresentada, e todas as funcionalidades de usuário da API estão conectadas a tal. Esta tela representa por onde o colegiado deve acessar o sistema.

Após efetuar o login, o usuário deve cadastrar os parâmetros de extração. Como pode ser visto na **Figura 14a**, é apresentada a tela para cadastro de parâmetros em página específica, e na **Figura 14b** para parâmetros de página indefinida. Ao enviar o formulário, o objeto JSON é montado e a RegEx codificada para base64 e assim o cadastro é realizado. Como resultado das informações preenchidas na **Figura 14a** é apresentado o objeto JSON da **Figura 15**.

Figura 14 – Registro de Parâmetro - Sistema de Extração de dados de TCC

(a) Registro de Parâmetro de Página Específica



(b) Registro de Parâmetro de Página Indefinida

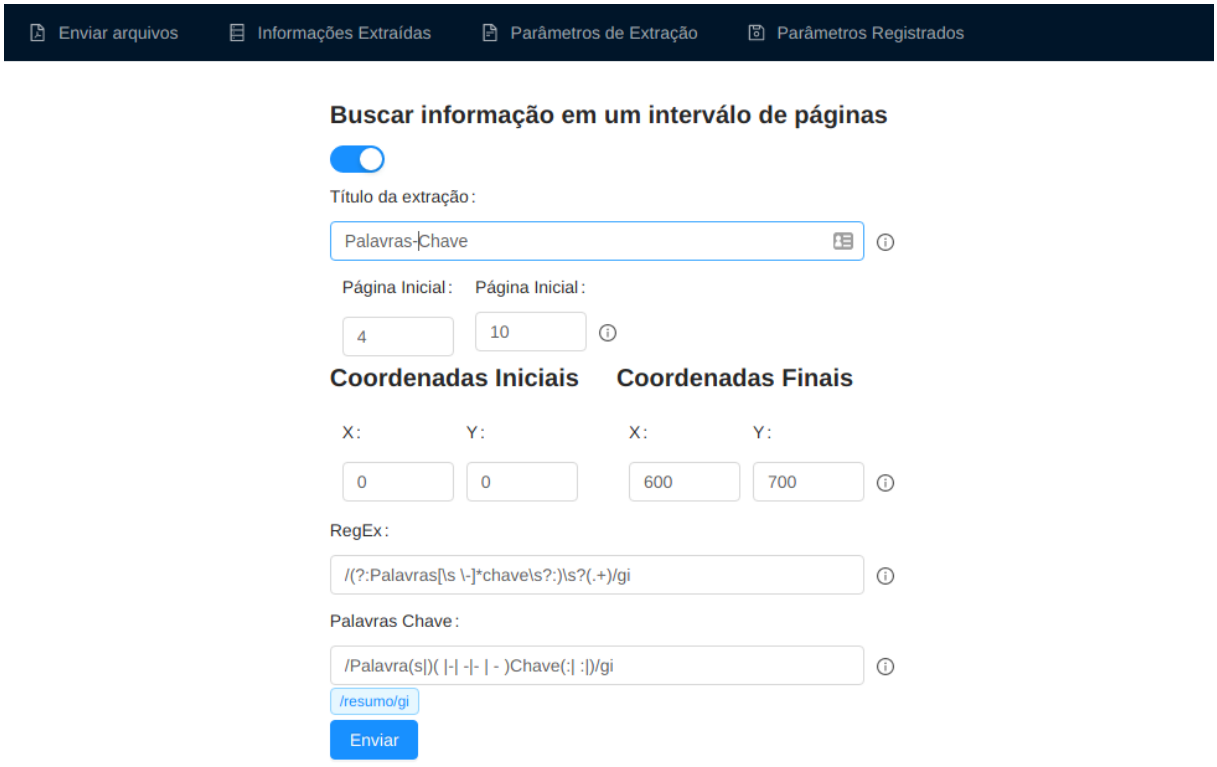


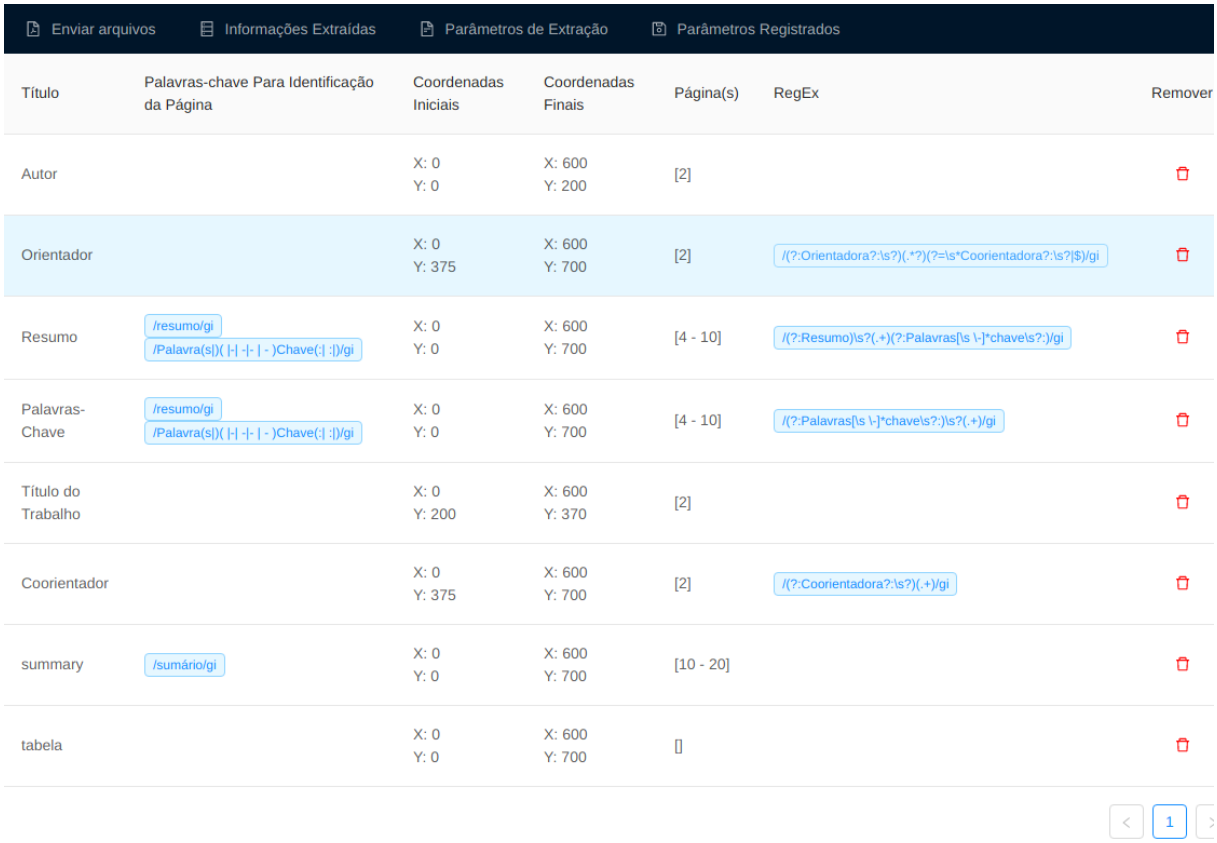
Figura 15 – Objeto JSON Resultante do Formulário de Parâmetros de Extração - Sistema de Extração de dados de TCC


```
{
  "extractionTitle":"orientador",
  "page":2,
  "coordinatesStartX":0,
  "coordinatesStartY":375,
  "coordinatesEndX":600,
  "coordinatesEndY":700,
  "regex":"Lyg/Ok9yaWVudCFkb3JhPzpccz8pKC4qPykoPz1ccypDb29yaWVudCFkb3JhPzpccz98JCkvZ2k="
}
```


Fonte: Autoria Própria

Os parâmetros registrados podem ser consultados na próxima aba, como apresentado na **Figura 16**, lá a informação é recuperada do banco de dados e é possível remover um registro, caso desejado.

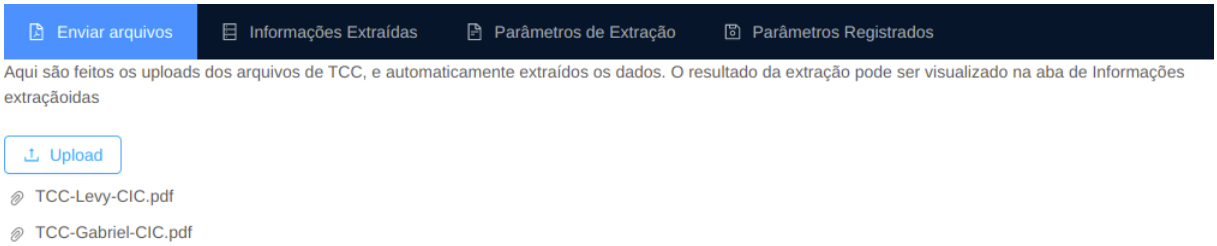
Figura 16 – Parâmetros Registrados - Sistema de Extração de dados de TCC



Fonte: Autoria Própria

Com os parâmetros devidamente registrados, é possível submeter novos arquivos, na **Figura 17** o botão de upload apresentado aceita um ou mais arquivos PDF, mas respeitando uma das constrições de API RESTful, é feita apenas uma requisição para cada arquivo.

Figura 17 – Envio de Arquivos - Sistema de Extração de dados de TCC



Fonte: Autoria Própria

Após enviado, o resultado da extração pode ser consultado na aba seguinte. A tabela apresentada na **Figura 18** é montada de acordo com os parâmetros que foram definidos como primordiais para o projeto de egressos.

Havendo mais informações extraídas, estas podem ser acessadas no formato de objeto JSON através do botão *Mais* a direita. E o resumo no botão presente à esquerda.

Figura 18 – Informações Extraídas - Sistema de Extração de dados de TCC

Fonte: Autoria Própria

5 CONCLUSÕES E TRABALHOS FUTUROS

A partir deste trabalho foi possível realizar o desenvolvimento de uma API que tenha a capacidade de extrair dados textuais de arquivos PDF. E em vista da existência de uma grande demanda por informações encontradas em tais arquivos, nota-se a importância desta API.

Apesar da complexidade interna destes arquivos, existem várias ferramentas que são capazes de auxiliar na leitura dos objetos que o compõe. E a utilização de uma ferramenta de auxilio a leitura do arquivo mostrou o caminho para uma abordagem mais simples e prática do que as apresentadas por SASIREKHA; CHANDRA, AJEDIG; LI; REHMAN, MARINAI e LIN et al.

Este trabalho também contou com um forte estudo de ferramentas atuais e relevantes como, por exemplo o Swagger e JWT, e assim foi demonstrado que a união destas pode resultar em um software conciso e confiável. Além disso, foi possível notar a importância de uma comunidade ativa de desenvolvedores. O que contribui para a evolução e discussão destas ferramentas, já que sua maior utilização vem de empresas.

Após a construção do sistema *front-end* para a classificação de TCC's do curso de Ciência da Computação da UESC, foram testadas as capacidades da API. Sendo algumas delas, a autenticação de usuário, o registro de parâmetros, o envio de arquivos, o retorno das informações esperadas dado os parâmetros, e o acesso à informação extraída. Com estes testes foi notado o bom funcionamento e desempenho do sistema, e como pode ser notado nas imagens apresentadas no capítulo anterior, as extrações foram bem sucedidas.

O sistema de classificação de TCC's provou a fácil utilização da API e se mostrou apto para utilização do colegiado. Deve-se observar que é um sistema *front-end* de baixa complexidade, porém aliado a API, com poucos recursos este se tornou uma poderosa ferramenta para extração de dados de TCC, e assim rapidamente consegue cumprir a demanda da rede de egressos.

A conclusão deste projeto demonstra que há possibilidades de diferentes abordagens para a extração de dados em documentos PDF. O desenvolvimento aqui realizado comprovou que uma aproximação mais modular pode ser muito benéfica e sendo bem executada resulta em um sistema preciso.

Ademais com a criação desta aplicação foi possível imaginar cenários de múltiplos benefícios através do uso da API, e quesitos em que ela pode ser melhorada. Possíveis trabalhos futuros são:

1. Aprimorar o módulo para extração de dados tabulares, mantendo a organização

original das informações;

2. Desenvolver sistema intuitivo para criação de parâmetros de extração; Auxiliando na criação de RegEx e coletando as coordenadas de forma gráfica;
3. Aceitar outros formatos de arquivo
4. Realizar testes de integração e escalabilidade;
5. Criar teste automatizados.

REFERÊNCIAS

AJEDIG, M. A.; LI, F.; REHMAN, A. A pdf text extractor based on pdf-renderer. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. [S.l.: s.n.], 2011. v. 1. Citado 4 vezes nas páginas 12, 14, 16 e 45.

BAKKEN, D. Middleware. *Encyclopedia of Distributed Computing*, Kluwer Academic, Dodrecht, The Netherlands, v. 11, 2001. Citado na página 19.

BIENZ, T.; COHN, R.; VIEW, C. A. S. M. *Portable document format reference manual*. [S.l.]: Citeseer, 1993. Citado na página 14.

DOC Parser. <https://docparser.com/>. [Acesso em: 03 jan. 2020.]. Citado na página 22.

EXTRACTPDF. <https://www.extractpdf.com/>. [Acesso em: 03 jan. 2020.]. Citado na página 22.

FIELDING, R. T. Rest: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*, 2000. Citado 2 vezes nas páginas 17 e 18.

FREE PDF Convert. <https://www.freepdfconvert.com/>. [Acesso em: 03 jan. 2020.]. Citado na página 22.

HAQ, Z. U.; KHAN, G. F.; HUSSAIN, T. A comprehensive analysis of xml and json web technologies. *New Developments in Circuits, Systems, Signal Processing, Communications and Computers*, p. 102–109, 2012. Citado 2 vezes nas páginas 18 e 19.

ISO. *Document management — Portable document format*. Geneva, CH, 2008. v. 2008. Citado 2 vezes nas páginas 12 e 14.

ITEXT PDF. <https://itextpdf.com/>. [Acesso em: 03 jan. 2020.]. Citado na página 22.

JIN, B.; SAHNI, S.; SHEVAT, A. *Designing web APIs: building APIs that developers love*. [S.l.]: OReilly Media, 2018. ISBN 978-1-492-02692-1. Citado na página 17.

JSON, J. I. Disponível em:< http://json. org/>. *Acesso em: 20 dez. 2019*, v. 20, n. 10, 2014. Citado na página 19.

JWT - JSON Web Token. 2020. <https://jwt.io/introduction/>. [Acesso em: 03 jan. 2020.]. Citado na página 26.

LA PDF. <https://github.com/BMKEG/lapdftext>. [Acesso em: 03 jan. 2020.]. Citado na página 22.

LIN, X. et al. Mathematical formula identification in pdf documents. In: IEEE. *2011 International Conference on Document Analysis and Recognition*. [S.l.], 2011. p. 1419–1423. Citado 2 vezes nas páginas 12 e 45.

LÓSCIO, B. F.; OLIVEIRA, H. d.; PONTES, J. d. S. Nosql no desenvolvimento de aplicações web colaborativas. *VIII Simpósio Brasileiro de Sistemas Colaborativos*, v. 10, n. 1, p. 11, 2011. Citado na página 21.

MARDAN, A. *Express. js Guide: The Comprehensive Book on Express. js*. [S.l.]: Azat Mardan, 2014. Citado na página 27.

MARINAI, S. Metadata extraction from pdf papers for digital library ingest. In: IEEE. *2009 10th International conference on document analysis and recognition*. [S.l.], 2009. p. 251–255. Citado 2 vezes nas páginas 12 e 45.

MASSÉ, M. *REST API design rulebook*. Sebastopol, CA: O'Reilly, 2012. ISBN 978-1-449-31050-9. Citado na página 17.

MEAD, A. *Learning Node.js development : learn the fundamentals of Node.js, and deploy and test Node.js applications on the web*. Birmingham, UK: Packt Publishing, 2018. ISBN 1788395549. Citado na página 20.

OPENAPI. Openapi specification. *Retrieved from GitHub: https://github. com/OAI/OpenAPI-Specification/blob/master/versions/3.0*, v. 1, 2017. Citado na página 28.

PDF Tables. <https://pdftables.com/>. [Acesso em: 03 jan. 2020.]. Citado na página 22.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. [S.l.]: Palgrave macmillan, 2005. Citado na página 31.

SASIREKHA, D.; CHANDRA, E. Text extraction from pdf document. 2013. Citado 3 vezes nas páginas 12, 16 e 45.

SHAH, H. Node. js challenges in implementation. *Global Journal of Computer Science and Technology*, 2017. Citado na página 20.

SOBRE o Adobe PDF. 2020. <https://acrobat.adobe.com/br/pt/acrobat/about-adobe-pdf.html>. [Acesso em: 03 jan. 2020.]. Citado na página 14.

SOBRE o Node Package Manager. 2020. <https://docs.npmjs.com/about-npm/>. [Acesso em: 03 jan. 2020.]. Citado 2 vezes nas páginas 21 e 34.

SPERBERG-MCQUEEN, M. et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. [S.l.], 2008. Http://www.w3.org/TR/2008/REC-xml-20081126/. Citado na página 19.

STACK Overflow. <https://stackoverflow.com/>. [Acesso em: 03 jan. 2020.]. Citado na página 13.

STACK Overflow Trends. <https://insights.stackoverflow.com/trends?tags=>. [Acesso em: 03 jan. 2020.]. Citado na página 13.

TABULA. <https://github.com/tabulapdf/tabula>. [Acesso em: 03 jan. 2020.]. Citado na página 22.

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 20.

WARNOCK, J. P.; GESCHKE, C. P. Pdf reference: Adobe portable document format version 1.4 with cdrom. Addison-Wesley Longman Publishing Co., Inc., 2001. Citado na página 15.