ESCOLA DE ENGENHARIA DE LORENA
ENGENHARIA FÍSICA

CAMILA ABRANTES DA FONSECA BATISTA

DESENVOLVIMENTO DE UMA INTERFACE WEB INTEGRADA A UM BANCO DE DADOS PARA ATRIBUIÇÃO DE PROFESSORES A DISCIPLINAS

LORENA
2020

CAMILA ABRANTES DA FONSECA BATISTA

DESENVOLVIMENTO DE UMA INTERFACE WEB INTEGRADA A UM BANCO DE DADOS PARA ATRIBUIÇÃO DE PROFESSORES A DISCIPLINAS

Trabalho de conclusão de curso servindo como exigência parcial para obtenção do grau de bacharel em Engenharia Física na unidade Escola de Engenharia de Lorena da Universidade de São Paulo (EEL-USP).

Orientador: Prof. Dr. Luiz Tadeu Fernandes Eleno.

LORENA

2020

Ficha catalográfica elaborada pelo Sistema Automatizado
da Escola de Engenharia de Lorena,
com os dados fornecidos pelo(a) autor(a)

Batista, Camila Abrantes da Fonseca
Desenvolvimento de uma interface web integrada a um banco de dados para atribuição de professores a disciplinas / Camila Abrantes da Fonseca Batista; orientador Luiz Tadeu Fernandes Eleno. – Lorena, 2020.
79 p.

Monografia apresentada como requisito parcial para a conclusão de Graduação do Curso de Engenharia Física – Escola de Engenharia de Lorena da Universidade de São Paulo. 2020

1. Interface web. 2. Html. 3. Python. 4. Banco de dados. I. Título. II. Eleno, Luiz Tadeu Fernandes, orient.

À minha família.

AGRADECIMENTOS

A Deus pela força para perseverar.

Aos meus amigos, que sempre me acolheram e apoiaram.

À minha família, que permitiu que esse sonho da graduação fosse possível.

Ao Prof. Dr. Luiz T. F. Eleno, pela oportunidade de realizar esse projeto e pela excelente orientação.

"Não é na ciência que está a felicidade, mas na aquisição da ciência". - Edgar Allan Poe

RESUMO

O regime de trabalho de um docente envolve dedicar algumas de suas horas semanais a atividades de ensino. É função do departamento organizar o trabalho docente e ministrar as disciplinas de graduação, distribuindo os docentes entre os encargos de ensino. Esse trabalho atualmente é feito por meio da atribuição manual. O objetivo desse trabalho é facilitar o trabalho da designação de docentes, por meio de uma interface web que permite realizar a atribuição dos professores do Departamento de Materiais à disciplinas a serem ministradas. O desenvolvimento desse projeto foi dividido em cinco partes: construção de um banco de dados a partir da planilha de Atribuição de Professores do DEMAR, utilizando Google API; *front-end*, utilizando HTML e CSS; *back-end*, utilizando Python e SQL; a integração do *front-end* com o *back-end*; e a hospedagem da página na internet. Apesar de algumas funcionalidades não terem sido desenvolvidas, o resultado final é a primeira versão de uma aplicação web online que permite a realização da tarefa de atribuir professores à disciplinas, com margem para aprimoramento e otimização do projeto no futuro.

Palavras-chave: Interface Web. HTML. Python. Banco de dados.

ABSTRACT

A teacher's work schedule involves dedicating some of their weekly hours to teaching activities. It is the department's job to organize the teaching work, distributing the teachers among the teaching assignments. This work is currently done through manual assignment. The objective of this work is to facilitate the work of assigning teachers, through a web interface that allows designating Departamento de Materiais' teachers to the subjects to be taught. The development of this project was divided in five parts: construction of a database from the DEMAR Teacher Assignment spreadsheet, using Google API; front-end, using HTML and CSS; back-end, using Python and SQL; front-end/back-end integration; and hosting the webpage. Although some features have not been developed, the end result is a first version of an online web application that allows the task of assigning teachers to the subjects, with the possibility for improvement and optimization of the project in the future.

Key-words: Web interface. HTML. Python. Database.

LISTAS DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

App	Aplicativo
API	Application Programming Interface
CSS	Cascading Style Sheet
DEMAR	Departamento de Materiais
EEL	Escola de Engenharia de Lorena
GCP	Google Cloud Platform
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
PYPL	PopularitY of Programming Language
REST	Representational State Transfer
SQL	Structured Query Language
UI	User Interface
USP	Universidade de São Paulo
UX	User Experience

SUMÁRIO

INTRODUÇÃO

De acordo com o Regimento Geral da Universidade de São Paulo , o regime de trabalho de um docente envolve dividir suas horas semanais em atividades de pesquisa, de extensão de serviços à comunidade e de ensino (UNIVERSIDADE DE SÃO PAULO, 1990). Cada professor possui, portanto, um número limitado de horas semanais as quais pode se dedicar a lecionar disciplinas para o corpo discente e, de acordo com a sua formação e sua área de concurso, pode ministrar apenas alguma matérias pré-definidas.

Seguindo esse mesmo princípio, ainda de acordo com o Regimento, é função do departamento organizar o trabalho docente e ministrar as disciplinas de graduação, distribuindo os docentes entre os encargos de ensino e de extensão (UNIVERSIDADE DE SÃO PAULO, 1990). Assim, é função do Departamento de Materiais da Escola de Engenharia de Lorena dividir as 57 disciplinas do departamento - e suas respectivas turmas - entre os 39 docentes que respondem ao departamento, respeitando as horas hábeis e as áreas de atuação de cada um (ELENO, 2020). Esse trabalho atualmente é feito por meio da atribuição manual, utilizando planilhas, o que acaba sendo um processo desgastante e demorado e que, contando apenas com a ação humana, pode acabar apresentando falhas e confusões.

Dessa maneira, o objetivo desse trabalho é facilitar o trabalho da designação de docentes à disciplinas, desenvolvendo uma interface web que permite, de maneira gráfica e visual, realizar a atribuição dos professores do Departamento de Materiais às turmas das disciplinas que os mesmos são capazes de ministrar, respeitando suas condições de horas hábeis. Para criar essa plataforma, foram utilizados os conceitos explicitados a seguir.

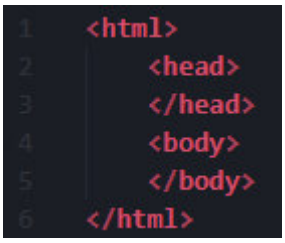
2 REVISÃO BIBLIOGRÁFICA

2.1 HTML

O *Hypertext Markup Language*, ou HTML, que traduzido para o português significa Linguagem de Marcação para Hipertexto, foi desenvolvido para escrever documentos a serem interpretados por agentes de usuário, como navegadores (SILVA, 2008). Essa invenção permitiu a criação e o compartilhamento de documentos eletrônicos e multimídia integrados em uma única página na *web* (KENNEDY; MUSCIANO, 2002). Mais tarde, criou-se também o '*hypertext linking*', que possibilitou o referenciamento automático de outros documentos localizados na rede mundial de computadores, o que facilitou a conexão global.

Um documento do tipo HTML é caracterizado por três *tags* chamadas <html>, <header> e <body> (CROWDER, 2011). A figura 1 a seguir apresenta uma estrutura básica para se criar uma página *web* com HTML.

Figura 1 - Exemplo de estrutura de documento do tipo HTML.

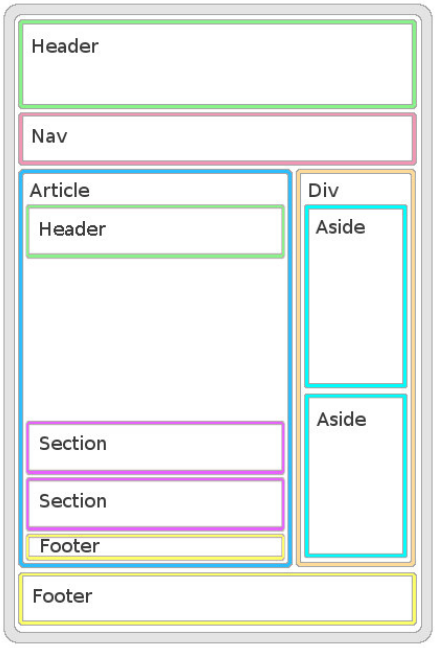


Fonte: (AUTORIA PRÓPRIA, 2020)

A primeira *tag* — <html> — indica o tipo do documento, no caso HTML, e contém as outras *tags* que compõem o documento. A *tag* <head> contém informações que não irão aparecer no corpo do documento, ou seja, que não estarão visíveis ao acessar a página na internet. Essas informações podem ser dados do autor da página, links de referência que serão utilizados para estilizar o corpo de texto, *scripts* importados de outros sites, entre outros. A única informação presente no <header> que é visível é o título do *website*, o qual aparece escrito na aba do navegador.

A tag <body>, por sua vez, é a parte do documento em que se desenvolve a parte visível do site. A figura 2 a seguir mostra como o <body> é dividido em outras *tags* e como elas se dispõem no quesito de estrutura do documento. A versão mais recente do HTML é o HTML5 e ela possui as funcionalidades de *subtags* apresentadas a seguir.

Figura 2 - Exemplo de *tags* que podem estar contidas no <body>.



Fonte: (TOMLIN VENTURES LTD, 2011)

Por fim, é importante lembrar que o HTML é uma linguagem de marcação e não uma linguagem de programação, como é erroneamente referenciado muitas vezes. A linguagem de marcação trabalha junto com linguagens de programação, como JavaScript, Python, entre outras.

2.2 CSS

CSS, ou *cascading style sheet*, é uma ferramenta para estilizar documentos HTML (SILVA, 2008). Com isso, pode-se mudar as fontes de texto, colocar backgrounds, alterar cores e espaçamentos, deixando o website mais *user-friendly*. Assim como o HTML, o CSS não é uma linguagem de programação. Nesse caso, é uma linguagem de folha de estilos.

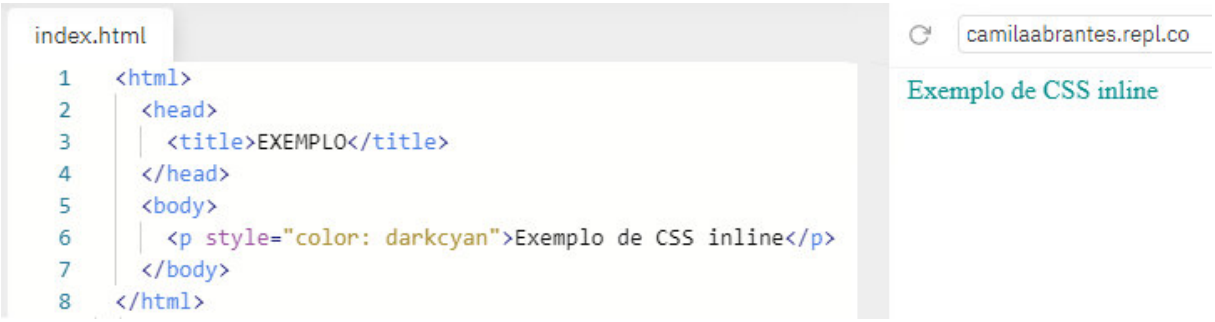
O CSS pode ser implementado ao documento HTML de três maneiras diferentes, descritas a seguir (CROWDER, 2011):

2.2.1 Inline

O método *inline* consiste em descrever o estilo do elemento diretamente na linha de código. Isso afeta apenas aquele componente local de código. Essa maneira

de estilizar o código é funcional para mudanças específicas, pois não é flexível, sendo o método mais simple. A figura 3 a seguir demonstra o uso desse método e o resultado no *website* final.

Figura 3 - Exemplo de arquivo HTML utilizando CSS *inline* para definir estilo, trocando a cor do texto, e o resultado final no *website* teste.



Fonte: (AUTORIA PRÓPRIA, 2020)

2.2.2 Tag <style>

A tag <style> é um componente do documento assim como as supracitadas <head> e <body>. No entanto, ali fica localizada toda a informação de estilização, ou seja, o CSS. Essa forma permite uma organização melhor do código, separando funcionalidade e estilo. Porém, declarar todos os estilos em um arquivo só torna-o pesado e difícil de ler, logo esse método seria o intermediário entre os três. A figura 4 a seguir demonstra o uso desse método e o resultado no website final, que se apresentou igual ao resultado da figura 3 anterior.

Figura 4 - Exemplo de arquivo HTML utilizando CSS dentro da *tag* <style> para definir estilo, trocando a cor do texto, e o resultado final no *website* teste.

Fonte: (AUTORIA PRÓPRIA, 2020)

2.2.3 Arquivo .css

Um arquivo .css separado do .html e referenciado por meio de hiperlink é o método mais indicado para se desenvolver um *website*. Dessa forma, todas as estilizações do projeto ficam separadas das funcionalidades desenvolvidas, tornando o código mais limpo, leve e fácil de processar. Uma única *stylesheet* pode ser utilizada para diversas páginas HTML, o que facilita e poupa tempo de desenvolvimento. A figura 5 a seguir demonstra o uso desse método e o resultado no *website* final, que se apresentou igual ao resultado da figura 3 e da figura 4 anteriores.

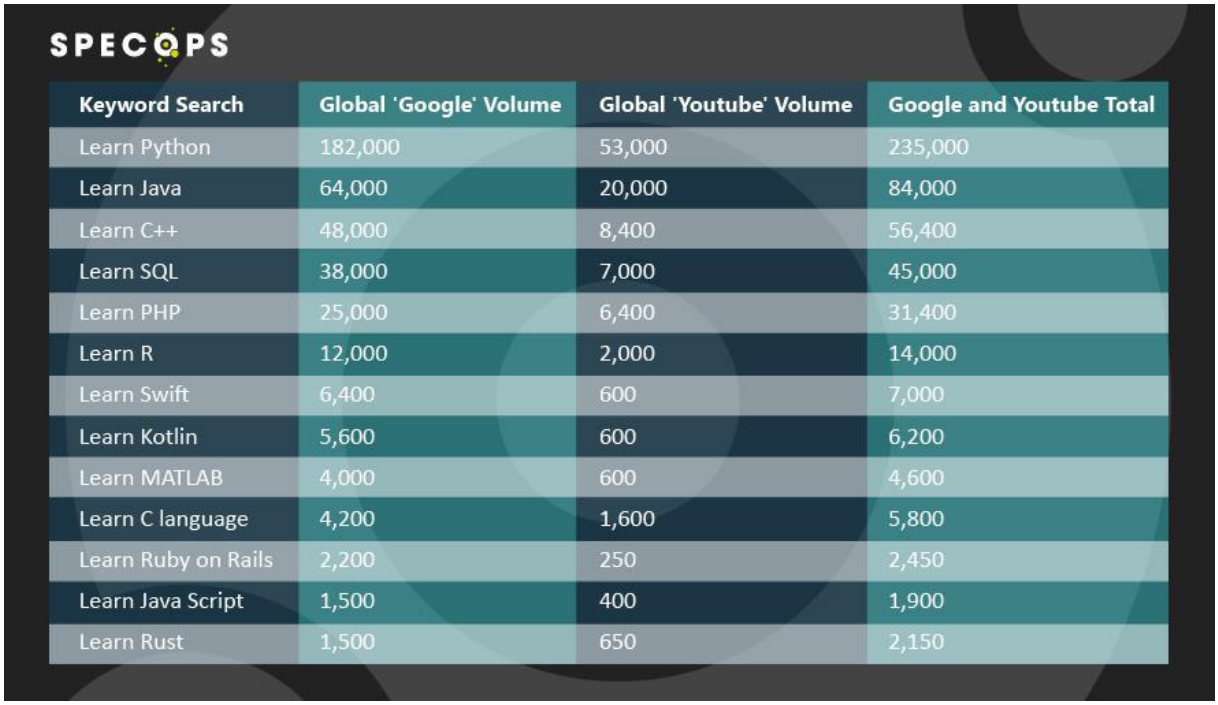
Figura 5 - Exemplo de arquivo HTML referenciando arquivo CSS por meio de hiperlink para definir estilo, trocando a cor do texto, e o resultado final no *website* teste.

Fonte: (AUTORIA PRÓPRIA, 2020)

2.3 Linguagens de programação

Ao longo do tempo, as linguagens de programação vem sendo cada vez mais usadas na resolução de problemas. Com o acesso cada vez mais comum e facilitado a internet e consequentemente ao conhecimento, com a ajuda de ferramentas como Google e YouTube, as pessoas passam a ter autonomia para aprenderem mais sobre seus interesses tecnológicos. Um estudo feito pela Specops Software utilizando o Google Trends mostra a média de pesquisas mensais globais sobre aprender uma linguagem de programação, como é possível observar na figura 6 a seguir (SPECOPS SOFTWARE, 2020).

Figura 6 - Média global mensal de pesquisas pelos termos envolvendo aprendizado de determinadas linguagens de programação.



SPECOPS

Keyword Search	Global 'Google' Volume	Global 'Youtube' Volume	Google and Youtube Total
Learn Python	182,000	53,000	235,000
Learn Java	64,000	20,000	84,000
Learn C++	48,000	8,400	56,400
Learn SQL	38,000	7,000	45,000
Learn PHP	25,000	6,400	31,400
Learn R	12,000	2,000	14,000
Learn Swift	6,400	600	7,000
Learn Kotlin	5,600	600	6,200
Learn MATLAB	4,000	600	4,600
Learn C language	4,200	1,600	5,800
Learn Ruby on Rails	2,200	250	2,450
Learn Java Script	1,500	400	1,900
Learn Rust	1,500	650	2,150

Fonte: (SPECOPS SOFTWARE, 2020)

Esse crescimento de interesse da população global por aprender a programar também é visto no *ranking* PYPL - *PopularitY of Programming Language* que tem o objetivo de monitorar a popularidade das linguagens de programação ao longo dos anos. Pode-se ver na figura 7 a seguir, as dez linguagens mais procuradas no Google em novembro de 2020 comparado a novembro de 2019.

Figura 7 - Popularidade das linguagens de programação em novembro de 2020 em comparação a novembro de 2019.

Worldwide, Nov 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	30.8 %	+1.8 %
2		Java	16.79 %	-2.3 %
3		JavaScript	8.37 %	+0.3 %
4		C#	6.42 %	-0.9 %
5		PHP	5.92 %	-0.2 %
6		C/C++	5.78 %	-0.2 %
7		R	4.16 %	+0.4 %
8		Objective-C	3.57 %	+1.0 %
9		Swift	2.29 %	-0.2 %
10		TypeScript	1.84 %	-0.0 %

Fonte: (CARBONELLE, 2020)

Ambas as figuras trazem como dado o quanto JavaScript, SQL e Python são tecnologias cada vez mais presentes, estudadas e utilizadas. A popularidade dessas ferramentas se dá, principalmente, por conta da facilidade de aprendizado e de utilização delas e também pela grande variedade de empregabilidade de todas em conjunto. Esses pontos foram essenciais para a escolha das tecnologias utilizadas para o desenvolvimento do projeto, as quais serão explicitadas e explicadas a seguir.

2.3.1 JavaScript

Conforme supracitado, para criar um website utiliza-se o HTML, que por sua vez não é uma linguagem de programação, mas, na verdade, de marcação. Dessa maneira, é possível montar com o HTML e com o CSS uma estrutura de um formulário, mas só essas linguagens sozinhas não são capazes de salvar os dados ali colocados

(SILVA, 2010). Nessa parte do projeto, entram as linguagens de programação como o JavaScript.

De maneira geral, o JavaScript é a linguagem de programação do desenvolvimento *web*, descrevendo o comportamento das *webpages* (FLANAGAN, 2013). Assim como o CSS, o JavaScript pode ser escrito dentro do arquivo HTML ou em um arquivo .js separado e referenciado como hiperlink.

Na figura 8 a seguir é possível ver um exemplo de como o JavaScript funciona dentro de uma estrutura HTML, sendo utilizado dentro da *tag* <script> e sendo declarado como uma função a ser obedecida. A figura 8 também mostra a página web anterior ao clique no botão '*Try it*', que obedece a função do JavaScript, e o resultado final, depois do comando dado pelo JavaScript.

Figura 8 - Exemplo da linguagem JavaScript sendo utilizada em um arquivo .html e o resultado obtido na *webpage* após a função ser executada.

Fonte: (W3SCHOOLS, 2020)

2.3.2 SQL

Structured Query Language, ou SQL, é uma linguagem de programação estruturada para a consulta em bancos de dados (BEAULIEU, 2010). Essa ferramenta é de muita utilidade, pois permite a geração, a manipulação e a visualização de grandes quantidades de dados de uma única vez e tendo uma sintaxe simples, facilitando a realização de consultas e a criação de relatórios elaborados e completos.

2.3.2.1 Banco de dados

Expandindo mais sobre o tema, um banco de dados é um agrupamento de informações que possuem relações entre si (BEAULIEU, 2010). Um exemplo prático seria um menu de restaurante, onde existem dois tipos de dados: o nome do prato e seu preço correspondente. Essa utilização cotidiana mostra que os bancos de dados estão presentes em todos os tipos de listas e associações que se possa imaginar.

Assim, é compreensível que seja mais fácil armazenar dados eletronicamente conforme a complexidade e quantidade das informações aumente. Nesse caso, o SQL apresenta mecanismos de recuperação, armazenamento e gerenciamento para diversos conjuntos de dados simultaneamente (BEAULIEU, 2008).

2.3.2.1.1 Banco de dados relacional

O banco de dados relacional é o modelo de armazenamento de dados em tabelas interligadas por uma chave relacional (BEAULIEU, 2010). Muitas vezes chamada de 'ID' ou variações, essa chave age como identificador para relacionar uma grande quantidade de tabelas e dados diferentes. Destarte, o SQL é uma linguagem de programação de banco de dados relacional.

Na figura 9 a seguir é possível ver como uma consulta no banco dados - também chamada de '*query*' - funciona. O 'SELECT' seleciona os campos do banco que serão apresentados, e o 'FROM' que mostra de qual tabela os dados serão selecionados. O uso do * após o 'SELECT' é comum e retorna todos os campos possíveis presentes na tabela.

Figura 9 - Exemplo de uma consulta, ou *query*, realizada em SQL.

```
SELECT CODIGO, TITULO, PRECO,
       LANCAMENTO, ASSUNTO, EDITORA
FROM LIVRO
```

Fonte: (COSTA, 2007)

O resultado da *query* apresentada na figura 9 anterior é representado pela figura 10 a seguir, uma tabela com os campos 'CODIGO', 'TITULO', 'PRECO', 'LANCAMENTO', 'ASSUNTO' e 'EDITORA' buscados pelo 'SELECT'.

Figura 10 - Resultado da busca em um banco de dados em SQL.

CODIGO	TITULO	PRECO	LANCAMENTO	ASSUNTO	EDITORA
1	BANCOS DE DADOS PARA A WEB	31.2	10/01/1999	B	1
2	PROGRAMANDO EM LINGUAGEM C	30	01/10/1997	P	1
3	PROGRAMANDO EM LINGUAGEM C++	111.5	01/11/1998	P	3
4	BANCOS DE DADOS NA BIOINFORMÁTICA	48		B	2
5	REDES DE COMPUTADORES	42	01/09/1996	R	2

Fonte: (COSTA, 2007)

2.3.3 Python

Python é uma linguagem de programação *open source* (LUTZ, 2011), ou seja, projetado para ser visto, utilizado e modificado abertamente pelo público (OLIPHANT, 2007), podendo ser empregada e integrada em áreas como inteligência artificial, análise de dados, desenvolvimento *web*, customização de produtos, entre outros. É também uma linguagem de 'alto-nível' (OLIPHANT, 2007), o que significa que funciona através de palavras-chave mais próximas da linguagem humana, que depois são compiladas e interpretadas, tornando a aprendizagem mais fácil e sua utilização mais intuitiva.

Além disso, Python apresenta vários outros pontos que a torna extremamente interessante, útil e valiosa. Primeiramente, é uma linguagem que roda em muitas plataformas diferentes, não precisando de um compilador específico. Ainda, é uma ferramenta que conversa com outras linguagens e tecnologias, como HTML e SQL, o que a torna ideal para desenvolvimento de projetos *web*.

Outro ponto - e também o mais importante - é que Python possui inúmeras bibliotecas e módulos, que permitem a construção de programas sofisticados e multifuncionais, conectando o código com APIs de plataformas online, por exemplo, atualizando o projeto em tempo real, tornando sua aplicação mais palpável e real. Por fim, existe uma vasta comunidade de programadores em Python que está sempre disponibilizando documentação, tirando dúvidas em fóruns e contribuindo com projetos *open source*, o que enriquece a experiência com a linguagem e facilita o caminho do aprendizado (OLIPHANT, 2007).

2.3.3.1 Bibliotecas

Conforme supracitado, Python é uma linguagem open source, portanto sua extensa biblioteca padrão - assim como o seu interpretador - estão disponíveis de maneira gratuita no site www.python.org/ (VAN ROSSUM, 2003). Dessa forma também, o site distribui bibliotecas criadas e modificadas por terceiros, que criaram programas, scripts e ferramentas para solucionar problemas e facilitar a programação para os outros programadores. Assim, as documentações das bibliotecas utilizadas nesse trabalho estão disponíveis nas documentações do próprio Python e explicitadas a seguir.

2.3.3.1.1 pandas

Uma das bibliotecas utilizadas no projeto é chamada 'pandas', que foi desenvolvida para facilitar a análise e a manipulação de dados. O principal objeto dessa ferramenta é o *'DataFrame'*, uma espécie de tabela estruturada com colunas e linhas para armazenar os dados de maneira organizada (MCKINNEY, 2012). A figura 11 a seguir apresenta um exemplo desse *'DataFrame'*.

Figura 11 - Exemplo de dados em um '*DataFrame'* utilizando a biblioteca pandas do Python.

```
>>> frame
     total_bill  tip   sex     smoker  day  time    size
1    16.99       1.01  Female  No      Sun  Dinner  2
2    10.34       1.66  Male    No      Sun  Dinner  3
3    21.01       3.5   Male    No      Sun  Dinner  3
4    23.68       3.31  Male    No      Sun  Dinner  2
5    24.59       3.61  Female  No      Sun  Dinner  4
6    25.29       4.71  Male    No      Sun  Dinner  4
7    8.77        2     Male    No      Sun  Dinner  2
8    26.88       3.12  Male    No      Sun  Dinner  4
9    15.04       1.96  Male    No      Sun  Dinner  2
10   14.78       3.23  Male    No      Sun  Dinner  2
```

Fonte: (MCKINNEY, 2012)

No contexto da interface desenvolvida nesse trabalho, a Pandas foi utilizada para armazenar os dados referentes aos docentes, suas horas hábeis e as disciplinas que os mesmos podem ministrar, após a extração dessas informações da planilha de atribuições presente no Google Cloud Platform.

2.3.3.1.2 json

JavaScript Object Notation, ou JSON, é uma sintaxe de texto independente que facilita a conversa entre as linguagens de programação (ECMA INTERNATIONAL, 2017). A partir de um conjunto de regras estruturais, essa sintaxe representa e comporta dados independente da linguagem. Essa sintaxe é muito utilizada na composição de chaves de acesso para documentos privados, o que é o caso dos documentos armazenados no Google Cloud Platform.

Portanto, para acessar a planilha online que contém os dados dos professores, é necessário o uso da biblioteca 'json' do Python. Essa ferramenta conecta e converte a sintaxe do JSON para dados interpretáveis para o Python (VAN ROSSUM, 2017). Dessa forma, a planilha on-line se torna acessível a partir do código desenvolvido.

2.3.3.1.3 Flask

O Flask é um *framework*, ou seja, um conjunto de códigos prontos voltados para o desenvolvimento web (DE SOUZA, 2019) e é uma biblioteca presente no Python. Essa ferramenta é utilizada para construir aplicações online de maneira fácil, rápida e leve. Na figura 12 a seguir, é possível ver a simplicidade do código necessário para rodar um *app* online que retorna a frase "*Hello, World!*" no *localhost* da máquina, como é mostrado também na figura 13.

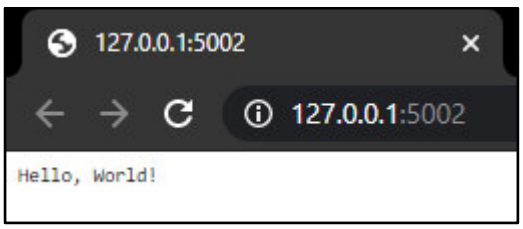
Figura 12 - Código em Python utilizando a biblioteca 'Flask' retornando "*Hello, World!*".

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Fonte: (THE PALLETS PROJECT, 2010)

Figura 13 - Página online no *localhost* da máquina retornando "*Hello, World!*".



Fonte: (AUTORIA PRÓPRIA, 2020)

Essa ferramenta se destaca dos outros *frameworks* pois admite uma customização maior para os projetos, devido a sua simplicidade e adaptação. Dentre as funções e vantagens do Flask estão (GRINBERG, 2018):

- comunicação do HTML, CSS e JavaScript com o Python, permite o desenvolvimento de um *website* com as vantagens de todas as linguagens e ferramentas envolvidas;
- suporta, lê e manipula bancos de dados, como o SQL, tornando possível a criação de páginas de registro, login ou de atribuição de dados, como é o caso desse trabalho;
- possui *templates* prontos para a criação de qualquer funcionalidade dentro do *website*, implementando tudo pronto, poupando esforços do desenvolvedor;
- facilita a criação de formulários dentro da aplicação e sua comunicação com os bancos de dados.

Dessa forma, o Flask se mostra imprescindível para o desenvolvimento de uma *website* simples, rápido e sem custo e é, assim, a base de todo esse trabalho.

2.3.3.1.4 SQLAlchemy

SQLAlchemy é uma biblioteca do Python criada para providenciar uma interface para bancos de dados relacionais como Oracle, MySQL e SQLite (COPELAND, 2008). Dessa maneira, essa ferramenta permite que se escrevam 'queries' do SQL em linguagem Python, facilitando a integração da database com o projeto. A figura 14 a seguir ilustra uma *query* utilizando o *SQLAlchemy*.

Figura 14 - *Query* escrita em *Python* utilizando o *SQLAchemy* para introduzir um *username* e uma *password* em uma tabela de usuários.

```
statement = user_table.insert(user_name='rick', password='parrot')
statement.execute()
```

Fonte: (COPELAND, 2008)

2.3.3.1.5 wtforms

Conforme supracitado, uma das vantagens do *Flask* é a facilidade de se criar formulários para o *website*, quando utilizado. Ainda assim, a *'wtforms'* é uma outra

biblioteca, e também um *microframework*, que trabalha em conjunto com outros frameworks como o *Flask*, *Django* e outros, para simplificar o processo de criação de formulários (WTFORMS, 2008). A *'wtforms'* disponibiliza templates para os formulários, tornando o desenvolvimento da aplicação mais fácil de executar.

2.4 Computação em nuvem

A computação em nuvem é um conceito que está revolucionando a maneira de desenvolver softwares (KRISHNAN; GONZALEZ, 2015). Essa ferramenta consiste no fornecimento de uma rede compartilhada - normalmente internamente entre funcionários de empresas - de recursos computacionais, incluindo servidores, bancos de dados e armazenamento, com o diferencial de ser centralizada em um único supercomputador que sempre estará ativo e que pode ser alugado por meio de empresas como Amazon, Microsoft e Google (TAURION, 2009).

A opção da computação na nuvem é vantajosa pois os dados alocados nesses supercomputadores não sofrem com queda de internet ou internet lenta, possuem um limite bem alto de armazenamento, são extremamente eficientes por serem processados por uma rede mundial de datacenters seguros e facilita e reduz os custos de backup (MICROSOFT AZURE, 2020). Além disso, outro ponto favorável dessa ferramenta é a facilitação do uso de aplicações online e em tempo real por meio de interfaces de programação de aplicações, ou APIs.

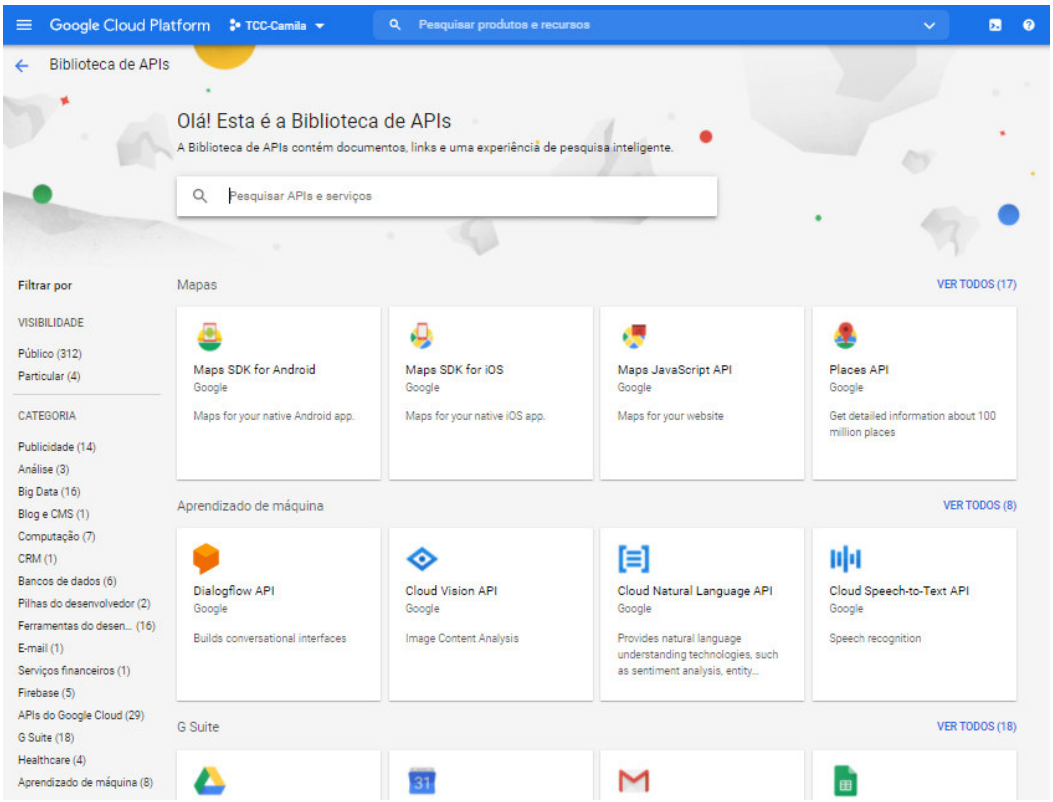
2.4.1 Google Cloud Platform e Google APIs

A Google Cloud Platform, ou GCP, é a plataforma de computação na nuvem do Google. Assim como seu buscador universalmente utilizado, a GCP é baseada em das maiores redes de computadores do mundo, compreendida por milhares de quilômetros de fibra ótica e redes avançadas de software, o que oferece desempenho rápido, consistente e escalável (KRISHNAN; GONZALEZ, 2015). Essa plataforma permite que desenvolvedores - inclusive independentes - construam seus softwares, realizem testes, monitorem suas aplicações e as integrem com outras ferramentas oferecidas pelo Google.

A GCP funciona com base nas APIs, sendo essas as principais interfaces para os produtos oferecidos pelo Google. Isso significa que para utilizar uma ferramenta do

Google em um projeto hospedado na nuvem é necessário habilitar sua API correspondente na Biblioteca de APIs usando o Console de Desenvolvedor, representados na figura 15 a seguir.

Figura 15 - Biblioteca de APIs presente no Console de Desenvolvedor do Google Cloud Platform.



Fonte: (AUTORIA PRÓPRIA, 2020)

2.5 User Interface e User Experience

O desenvolvimento de um website vai muito além de funcionalidades e tecnologias que criam bancos de dados ou geram respostas automáticas. Além da parte técnica, existe também o fator humano. Como o usuário entende a aplicação que está usando? Ele se sente confortável com a plataforma? As funcionalidades são intuitivas e fáceis de utilizar sozinho? Todas essas questões são abordadas com estudos de experiência do usuário, ou user experience (UX), para melhorar cada vez mais a interface para o usuário, ou user interface (UI).

O UX engloba todos os sentimentos, pensamentos, sensações e ações que o visitante ao site pode ter ao realizar uma atividade dentro da sua webpage (BENYON, 2019). Bons sentimentos levam a boas ações e isso pode significar, por exemplo, mais

visitas para o seu site, um maior reconhecimento de marca, aumento nas vendas e no lucro. Por conta disso, o design da UX deve ser interativo, atrativo e agradável de usar, tendo foco na pessoa utilizadora, incluindo acessibilidade.

Com isso em consideração, é desenvolvida a interface com foco no usuário. A interface é o meio por onde humanos interagem com um sistema computacional, logo é importante que a UX seja valorizada no design da UI, pois uma UI mal pensada pode acabar levando o usuário ao erro. Um exemplo disso é um aplicativo confuso de banco, que pode levar o cliente a perder dinheiro por não mostrar da maneira correta que seu saldo está negativo. Dessa forma, a escolha da ferramenta para a criação e o design da UX e da UI é de suma importância para a usabilidade e aceitação do projeto pelo usuário.

2.6 Arquitetura REST

O *Hypertext Transfer Protocol*, ou HTTP, é o principal protocolo de comunicação para sistemas Web. Para que o cliente se comunique com a webpage são necessários quatro requisições HTTP: GET, POST, PUT e DELETE (FREDERICH, 2020). O GET lê e retorna dados já existentes no sistema. O POST é utilizado para postar novos dados e o PUT para atualizá-los. Por fim, o DELETE serve para deletar dados indesejados.

As regras e princípios da arquitetura desse protocolo é chamada de *Representational State Transfer, ou* REST, que em português significa Transferência de Estado Representacional (MASSE, 2012)**.** Por sua vez, as APIs REST são a ferramenta que possibilita que um cliente faça uma requisição a uma interface e essa responda com os dados corretos, como podemos ver na figura 16 a seguir.

Figura 16 - Esquema da comunicação de um cliente com o back-end da interface por meio de uma REST API.

FONTE: (MASSE, 2012)

As interfaces que funcionam utilizando a arquitetura REST são chamadas de RESTful. Serviços web que são RESTful, utilizando APIs REST bem escritas e desenvolvidas são um diferencial para a web-page, pois a torna mais organizada e fácil de ser lida (MASSE, 2012).

3 MATERIAIS E MÉTODOS

O desenvolvimento desse projeto foi dividido em quatro partes: construção de um banco de dados a partir da planilha de Atribuição de Professores do Demar; *front-end*, sendo a parte visível da *webpage* junto com seu design e estilização; *back-end*, sendo a parte que integra o banco de dados com o front-end e realiza as requisições devolvendo os dados corretos; a integração do *front-end* com o *back-end;* e a hospedagem da página web, permitindo que a página fique online e acessível para o usuário. A elaboração de cada etapa será descrita a seguir.

3.1 Banco de dados

3.1.1 Planilha online

Para criar o banco de dados contendo os dados dos professores, suas horas hábeis e as disciplinas que podem ministrar, o primeiro passo foi transformar a planilha fornecida pelo Dr. Prof. Luiz Eleno e transformá-la em uma planilha online no serviço Google Sheets. Dessa forma, a planilha pode ser atualizada de qualquer local, pelas pessoas autorizadas, e com a atualização da planilha, atualiza-se também o banco de dados automaticamente. Para isso, o processo foi simples: apenas copiar os dados de uma e colar na outra. Houve também um processo de limpeza de dados, para melhor leitura da planilha pelas ferramentas de programação, resultando na planilha representada pela figura 17 a seguir.

Figura 17 - Captura de tela de parte da planilha de atribuição online presente no Google Sheets.

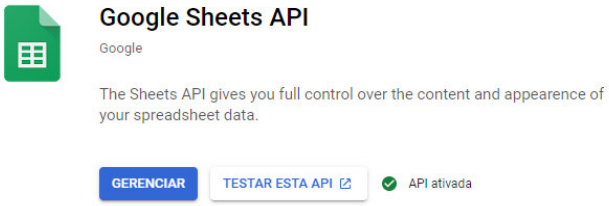
	A	B	C	D	E
1	Docente DEMAR		1o. SEMESTRE de 2020		
2	Contrato	Horas Hábeis	DISCIPLINA	C.Júpiter	A.
3	Alain L. M. Robin	40	Licença Médica		
4	Ana Márcia Barbosa da Silva Antunes	40	Introdução à Ciência dos Materiais	LOM3016	2
5	Ana Márcia Barbosa da Silva Antunes	40	MIQ	LOM3022	2
6	Ângelo Capri Neto	40	Química Analítica para Engenharia	LOQ4056	4
7	Antônio F. Sartori	40	Eletrometalurgia	LOM3008	2
8	Antônio F. Sartori	40	Física I	LOB1018	2
9	Antônio F. Sartori	40	Métodos Experimentais Física I Sartori/Durval	LOM3228	2
10	Antonio J. S. Machado	40	Química dos Materiais	LOM3241	2
11	Carlos A. M. Santos	40	Tópicos Especiais da Engenharia Física I (Optativa)	LOM3241	4
12	Carlos A. R. P. Baptista	40	Fadiga dos Materiais Metálicos (OPTATIVA)	LOM3050	2

Fonte: (AUTORIA PRÓPRIA, 2020)

3.1.2 Google Sheets API

Após essa conversão de planilha, a próxima etapa foi ativar a API do Google Sheets, no Console de Desenvolvedor presente no GCP. Para isso, foi criada uma conta de serviço na aba de ‘credenciais’ na seção de ‘APIs e serviços’. Ao criar essa conta, é gerado um arquivo .json que é a chave de permissão para integrar a planilha online com o código em Python. Após isso, a API foi ativada, conforme mostra a figura 18 a seguir.

Figura 18 - Ativação da API do Google Sheets.



Fonte: (AUTORIA PRÓPRIA, 2020)

Depois de ativar a API, a conta de serviço está pronta para ser utilizada e operar alterações e comunicações com o banco de dados e próximos passos de código. Para que ela funcione dessa maneira é necessário colocá-la como ‘Editor’ do arquivo online do Google Sheets, o que é apresentado na figura 19 a seguir:

Figura 19 - Chave de serviço do Google Sheets API com acesso de ‘Editor’ na planilha online do Google Sheets.

Fonte: (AUTORIA PRÓPRIA, 2020)

Essa etapa de configuração da API é importante, pois é ela que faz toda a comunicação do que está online com o que está no código. Sem a chave de serviço e a chave .json de permissão não é possível acessar o arquivo, o que mostra que é um procedimento muito seguro para a utilização de dados sensíveis.

3.1.3 Código em Python

Para criar o código em Python cuja finalidade é ler os dados da planilha online e transformá-la em uma DataBase, inicialmente foi necessário importar as bibliotecas ‘pandas’, ‘google.cloud’, ‘google.oauth2’, ‘googleapiclient’, ‘os’ e ‘json’. (Explicar depois). Após a importação, foram definidos os parâmetros de acesso, como o

escopo, a chave de acesso .json, as credenciais, a identificação da planilha e o alcance dos dados dentro dela.

Depois disso, criou-se uma definição de método chamada ‘read_spreadsheet()’, que tem como funções:

```
service = build('sheets','v4',credentials=credentials)
sheet = service.spreadsheets()
result_input = sheet.values().get(spreadsheetId =
SAMPLE_SPREADSHEET_ID_input, range =
SAMPLE_RANGE_NAME).execute()
values_input = result_input.get('values', [])
df = pd.DataFrame(values_input[1:],
columns=values_input[0])
df.dropna()
return df
```

O código supracitado utiliza um serviço que “constrói” uma planilha a partir das credenciais e acessa os dados da planilha com o artifício sheets.values().get(), utilizando os parâmetros previamente definidos. Os valores dos dados acessados são transformados em um “dicionário” em Python por meio do método result_input.get(‘values’, [ ]). Esse dicionário é transformado em um DataFrame utilizando o comando pd.DataFrame e depois é efetuado uma limpeza dos dados com o df.dropna(), que retira os dados vazios. Por fim, o DataFrame final é retornado como resultado do método definido.

3.1.4 Banco de dados resultante em SQL

A parte final da etapa de construção do banco de dados foi feita pelo seguinte código:

```
planilha = read_spreadsheet()
conn = sqlite3.connect('Atribuicao.db')
c = conn.cursor()
c.execute('CREATE TABLE ATRIBUICAO (Contrato, Horas
Hábeis,
DISCIPLINA, C Júpiter, A)')
conn.commit()
planilha.to_sql('ATRIBUICAO', conn, if_exists='replace',
index = False)
```

Inicialmente o código chama e executa o método read_spreadsheet(), transformando a planilha online em uma DataBase. O resultado dessa operação é o banco de dados representado na figura 20 a seguir, com campos de Contrato sendo o nome do professor, horas hábeis, disciplina, C.Júpiter sendo o código da disciplina e A. sendo os créditos referentes a disciplina em questão.

Figura 20 - *DataBase* do pandas referente a planilha de Atribuição online.

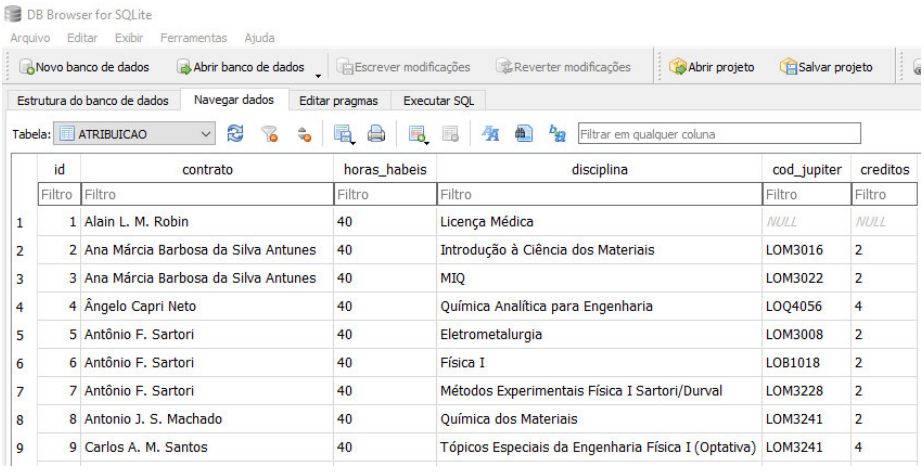
	Contrato	Horas Hábeis	DISCIPLINA	C.Júpiter	A.
0	Alain L. M. Robin	40	Licença Médica	None	None
1	Ana Márcia Barbosa da Silva Antunes	40	Introdução à Ciência dos Materiais	LOM3016	2
2	Ana Márcia Barbosa da Silva Antunes	40	MIQ	LOM3022	2
3	Ângelo Capri Neto	40	Química Analítica para Engenharia	LOQ4056	4
4	Antônio F. Sartori	40	Eletrometalurgia	LOM3008	2
..	...	...	...	...	...
70	Sebastião Ribeiro	40	Tratamento de Minérios	LOM3071	2
71	Sérgio Schneider	40	Processamento de Materiais Metálicos I	LOM3004	2
72	Sérgio Schneider	40	Processos da Indústria Mecânica	LOM3079	2
73	Sérgio Schneider	40	Processos de Materiais Metálicos I	LOM3004	2
74	Viktor Pastoukhov	40	AFASTAMENTO	None	None

Fonte: (AUTORIA PRÓPRIA, 2020)

Depois, o sqlite3.connect() cria um arquivo .db para armazenar o banco de dados em SQL. O conn.cursor() garante que o arquivo .db será salvo no mesmo local que o arquivo .py em questão. O c.execute() cria uma tabela com os mesmo campos que existem na planilha online e na DataBase. Por fim, o .to_sql() transforma a DataBase 'planilha' em um banco de dados SQL. O código completo está presente na íntegra no Apêndice A.

O arquivo 'Atribuicao.db' gerado pelo código supracitado é o banco de dados em SQL utilizado no *back-end*. Com a ajuda do programa 'DB Browser for SQLite', é possível visualizar a estrutura final, inserindo a coluna 'id', que também será necessária para facilitar a identificar os dados no *back-end*. A figura 21 a seguir é uma captura de tela referente ao arquivo.

Figura 21 - Banco de dados SQL referente a planilha de Atribuição online.



	id	contrato	horas_habeis	disciplina	cod_jupiter	creditos
	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	1	Alain L. M. Robin	40	Licença Médica	NULL	NULL
2	2	Ana Márcia Barbosa da Silva Antunes	40	Introdução à Ciência dos Materiais	LOM3016	2
3	3	Ana Márcia Barbosa da Silva Antunes	40	MIQ	LOM3022	2
4	4	Ângelo Capri Neto	40	Química Analítica para Engenharia	LOQ4056	4
5	5	Antônio F. Sartori	40	Eletrometalurgia	LOM3008	2
6	6	Antônio F. Sartori	40	Física I	LOB1018	2
7	7	Antônio F. Sartori	40	Métodos Experimentais Física I Sartori/Durval	LOM3228	2
8	8	Antonio J. S. Machado	40	Química dos Materiais	LOM3241	2
9	9	Carlos A. M. Santos	40	Tópicos Especiais da Engenharia Física I (Optativa)	LOM3241	4

Fonte: (AUTORIA PRÓPRIA, 2020)

3.2 *Front-end*

O *front-end* de uma página da web é a parte visível ao usuário, a que apresenta a estrutura dos elementos e que pode ser estilizada. Para o desenvolvimento do front-end desse projeto foram utilizados: o programa Figma para UX e UI design; HTML, para estruturação dos elementos da página; e CSS para estilização desses elementos.

3.2.1 UX e UI Design

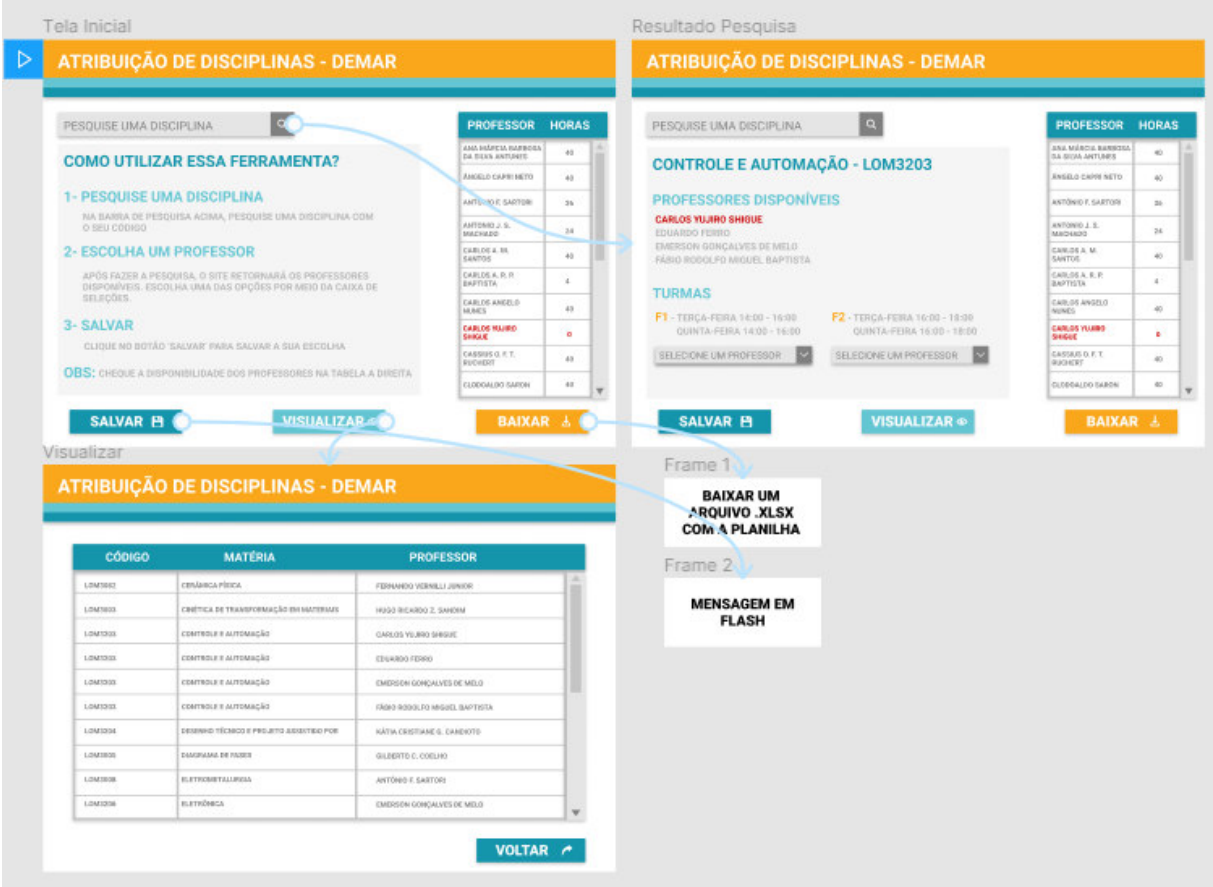
A ideia inicial da interface foi proposta como sendo uma aplicação web de uma única página, em que o chefe de departamento conseguisse ver as horas disponíveis de cada professor e, a partir de uma busca por uma determinada disciplina, fosse capaz de visualizar os professores disponíveis, as turmas disponíveis e atribuir os professores de acordo com as turmas e também respeitando as horas hábeis de cada um. Dessa forma, foi desenhada a interface da aplicação obedecendo os critérios da experiência desejada para o usuário e também de acordo com a identidade visual da USP e do seu site Jupiterweb (UNIVERSIDADE DE SÃO PAULO, 2020). A figura 22 a seguir é o protótipo da interface criado pelo Figma (FIGMA, 2020), um programa específico para UX e UI design de aplicações web.

Figura 22 - Design da interface no programa Figma.

Fonte: (AUTORIA PRÓPRIA, 2020)

Porém, ao longo do desenvolvimento do projeto, percebeu-se a necessidade da criação de duas páginas iniciais. A primeira página é a página ‘home’, contendo um tutorial de como utilizar a ferramenta. A segunda página é a página ‘visualizar’, onde o usuário pode visualizar uma tabela com as escolhas e atribuições que fez até o momento. A página antes criada foi colocado como a página ‘search’, que contém o resultado das pesquisas. Também foi definido que o botão ‘salvar’ apresenta uma mensagem de sucesso para o usuário ao ser clicado e o botão ‘baixar’ faz o *download* da planilha final de atribuições em um arquivo .xlsx, ou seja, formato do Excel. A figura 23 a seguir apresenta o protótipo final criado para ser seguido pelo desenvolvimento do projeto.

Figura 23 – Protótipo final da aplicação como um todo.



Fonte: (AUTORIA PRÓPRIA, 2020)

3.2.2 HTML

Visando construir uma estrutura semelhante a desenhada para a interface, o código HTML - presente na íntegra em Apêndice B, Apêndice C e Apêndice D - foi

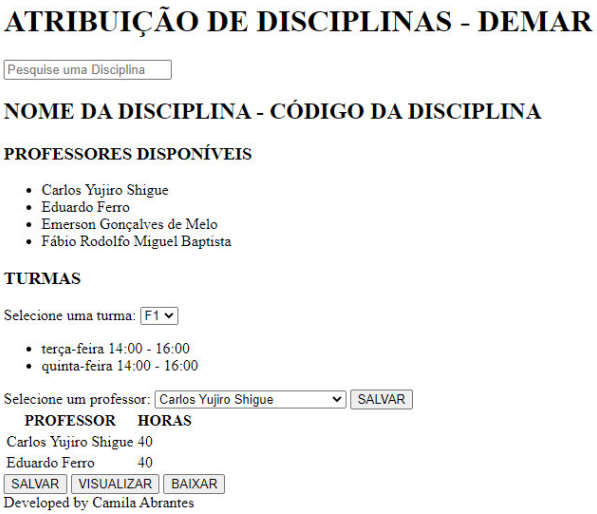
dividido em algumas *tags*, essas representando cada parte específica da página. O <body> do arquivo foi dividido entre <header>, <section> e <footer>. A *tag* <section>, por sua vez foi organizada em <header> e duas *tags* , o que torna a estruturação mais organizada e fácil de estilizar pelo CSS depois. Como os três arquivos HTML seguem a mesma estrutura, o código do arquivo 'search.html' presente no Apêndice C com uma tabela simplificada será utilizado para explicação.

Inicialmente, foi criada a estrutura do header, contendo o título do site e as faixas laranja, azul e azul claro, que são características da identidade visual online da USP. Depois, criou-se a tag <section>, contendo um <header> interno que apresenta o campo de pesquisa por disciplinas, e duas , uma contendo os botões de salvar, baixar e atualizar e outra contendo a principal parte da aplicação.

Essa contém uma tag <article class = "column left"> e uma <div class = column right>. Essas classes foram escolhidas para que as respectivas estruturas pudessem ser estilizadas com o CSS de maneira a ficarem lado a lado, porém ocupando larguras diferentes na tela. A <div class = column right> contém a tabela de professores e suas horas hábeis. A <article class = "column left"> contém toda a parte da aplicação referente ao resultado da busca feita no <header>, incluindo os professores disponíveis, as turmas disponíveis, a seleção do professor e o botão de salvar.

Por fim, a última estrutura da página é o footer, que contém os créditos do desenvolvimento. A figura 24 a seguir mostra como fica a página web desenvolvida apenas no HTML, sem a estilização com o CSS.

Figura 24 - Estrutura da página apenas em HTML.



Fonte: (AUTORIA PRÓPRIA, 2020)

3.2.3 CSS

Com a estruturação das tags no HTML pronta, a estilização por meio do CSS se torna muito mais fácil e intuitiva. O código inteiro referente a essa estilização está presente no Apêndice E. Conforme supracitado, a sintaxe do CSS é bem simples, sendo definida a partir das tags e classes em que se quer estilizar.

O primeiro passo do desenvolvimento do arquivo .css foi selecionar a fonte das letras da página, a qual foi setada com o código: html,body {font-family: Roboto; font-style: normal;}. Depois, foi setado que as seções não teriam margem externa, nem interna e que a página html ocuparia 100% da página. Após isso, foram setadas as cores de fundo das seções, as cores das letras, os alinhamentos de texto e as decorações de texto.

Um ponto importante da estilização foi a utilização das classes “right” e “left” para setar a tabela ocupando 25% do tamanho da tela e a área de informações da busca de disciplinas ocupando 70% da mesma, com ambas apresentando 1rem de margem entre si. O rem é uma unidade do CSS que padroniza os tamanhos dos elementos de acordo com o tamanho da fonte do elemento raiz do documento.

Além disso, foi colocado um código para fixar o <footer> ao final da página, para não ficar atrapalhando a leitura ao mudar o tamanho ou a resolução da tela. Isso foi realizado utilizando a class “content” na <section> e dando a ela um atributo {flex:1 0 auto;} e dando ao <footer> um atributo <margin-top:0;>.

Também, foram feitas as estilizações dos botões presentes na página, colocando atributo “:hover”, para que mudem de cor caso o cursor seja passado por cima deles, e ajustando as centralizações e margens internas e externas. Por fim, estilizou-se a tabela para que a mesma apresente uma estética mais agradável e mantenha seu tamanho fixo, utilizando um scroll para acessar todos os conteúdos da tabela.

Após todas essas modificações feitas, a página se apresentou mais parecida com o design inicial feito pelo Figma. Algumas partes se apresentam diferentes porque o primeiro protótipo foi feito a mão com formas geométricas e textos soltos, já o segundo foi desenvolvido do início apenas com HTML e CSS, o que apresenta limitações para a parte criativa. A figura 25 a seguir é uma captura de tela da página web resultante do desenvolvimento do front-end do projeto.

Figura 25 - Estrutura da página estilizada com CSS.

Fonte: (AUTORIA PRÓPRIA, 2020)

3.3 *Back-end*

O *back-end* de uma página da web é a parte que não é visível ao usuário, a que é responsável por toda a comunicação do *front-end* com o banco de dados e pelas requisições HTTP. Para o desenvolvimento do *back-end* desse projeto foram utilizados as biblioteca do Python: virtualenv, para setar um espaço específico para o projeto; Flask, para fazer a ponte com o HTML e realizar o protocolo HTTP; e SQLAlchemy, para realizar a comunicação com o banco de dados em SQL.

3.3.1 Preparação de ambiente

Para preparar o ambiente virtual do projeto utilizou-se a biblioteca virtualenv. Conforme supracitado, a virtualenv seta bibliotecas e recursos necessários para o projeto rodar, assim como deixa fixo ambiente em que a aplicação foi desenvolvida, para caso aconteça alguma atualização do Python, não existir conflito com o projeto. Os comandos realizados para essa preparação foram os seguintes:

```
pip install virtualenv
virtualenv interface
interface/Scripts/Activate.bat
```

```
cd interface
pip freeze > requirements.txt
```

O comando ‘pip install’ instala a biblioteca. O ‘virtualenv’ cria uma pasta chamada interface com o ambiente virtual configurado dentro dela. Depois, os *scripts* são ativados e é criado um documento explicitando os requisitos do projeto com o ‘pip freeze>requirements.txt’.

3.3.2 Instalação de bibliotecas

Com o ambiente virtual configurado, instala-se as bibliotecas Flask e Flask-RESTful, essa última sendo uma API de extensão da primeira supracitada, a qual facilita a realização de requisições HTTP pela aplicação. Ambas são instaladas com o ‘pip install’, assim como a biblioteca SQLAlchemy, que será utilizada para realizar *queries* dentro do banco de dados e retornar um resultado para a interface.

3.3.3 Arquivos .py

Para criar o relacionamento do banco de dados com o *front-end*, foi necessário criar um arquivo chamado ‘models.py’, disponível na íntegra no Apêndice F. Uma das funções desse código é a conexão com o banco de dados criado anteriormente, o que é realizado pelo trecho a seguir.

```
engine = create_engine('sqlite:///Atribuicao.db',
convert_unicode=True)
db_session =
scoped_session(sessionmaker(autocommit=False,
bind=engine,))
Base = declarative_base()
Base.query = db_session.query_property()
```

Para as funções desejadas da página web proposta neste projeto, foi necessário dividir o banco de dados em duas tabelas: uma chamada __tablename__ = 'ATRIBUICAO', contendo os dados obtidos pela planilha inicial e por já existir recebe o atributo __table_args__ = {'extend_existing': True} ; e outra criada pelo código ‘models.py’, chamada __tablename__ = 'NOVA_ATRIBUICAO', cuja função é receber as atribuições de professores e as matérias que os mesmos ministrarão.

Cada tabela dessas é definida por meio da definição de classes, onde o nome da tabela é definido, assim como as suas colunas e o tipo de dados que elas recebem. Além disso, esse arquivo cria e inicia o banco de dados aplicação desenvolvida. A biblioteca responsável por esses comandos é a SQLAlchemy.

Outro documento necessário para o desenvolvimento da aplicação é o app.py, presente integralmente no Apêndice G. Esse arquivo é responsável por importar as relações configuradas em 'models.py', por estabelecer os métodos HTTP e realizar as *queries* de acordo com as buscas efetuadas na interface. A biblioteca Flask configura e criar a aplicação web e a Flask-RESTful cria uma API de requisição HTTP para o aplicativo.

Para cada classe definida no 'models.py', foi definido métodos HTTP por meio da API. Um exemplo disso é o código a seguir.

```
class Contratos(Resource):
   def get(self):
       contrato = db_session.query(Contrato).distinct(
           Contrato.contrato).group_by(Contrato.contrato)
       response = [{
           'contrato': i.contrato,
           'horas_habeis': i.horas_habeis,
       } for i in contrato]
       table = json2html.convert(json=response)
       headers = {'Content-Type': 'text/html'}
       return make_response(render_template("index.html",
       table=table),headers)
```

Esse método define a resposta da aplicação para o protocolo GET, fazendo uma consulta de todos os dados contidos na tabela Contrato, previamente configurada em 'models.py' e retornando uma tabela com os nomes dos professores e as respectivas horas hábeis. Essa query é a responsável pela apresentação dos dados na tabela definida como <table> no arquivo .html.

Além disso, o comando api.add_resource() define o local em que a classe definida irá atuar. No caso do código anterior, a definição é api.add_resource(Contrato, '/'), pois estará presente na página inicial da aplicação. Por fim, é dado o comando de iniciar a aplicação no *localhost* da máquina pelo código descrito a seguir.

```
if __name__ == "__main__":
```

```
app.run(debug=True)
```

3.3.4 JavaScript e mensagem Flash

Para realizar o download da tabela de atribuição ao clicar no botão ‘baixar’, foi implementando no arquivo ‘visualizar.html’ o seguinte código em JavaScript.

```
<script>
    $("#btnExport").click(function (e) {
        window.open('data:application/vnd.ms-excel,'
      + $('#dvData').html());
        e.preventDefault();
    });
</script>
```

Esse código define que ao clilcar no botão com id = btnExport, a tabela definida com id = dvData será transformada para .xlsx e baixada como um arquivo Excel.

Por fim, para o site retornar uma mensagem flash de sucesso, o seguinte código foi implementado aos botões ‘salvar’.

```
<a href="/"><button onClick="alert('Seleção salva com
sucesso!')">SALVAR</button></a>
```

Esse código retorna uma mensagem com o clique do botão e manda o usuário de volta a página de início.

3.4 Integração *front-end*/*back-end*/banco de dados

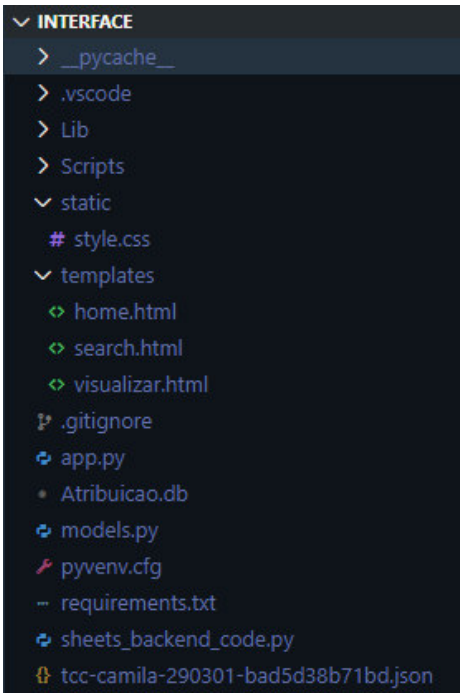
Depois de desenvolvidos os códigos para obtenção do banco de dados, para o *front-end* e para o *back-end* é necessário definir como funciona a comunicação entre esses códigos. Com essa finalidade, é utilizada a biblioteca Flask, que inicia a aplicação, apresenta a estrutura e a estilização, faz as requisições HTTP e devolve os resultados pedidos pelo usuário.

Primeiramente, é feita a integração do app com o *front-end*. Para isso, é necessário que os arquivos index.html é colocado em uma pasta chamada “templates” e o arquivo style.css é colocado em outra pasta chamada “static”. Essa estrutura é obrigatória, pois é uma especificação do Flask.

Após alocar os arquivos em suas devidas pastas, o arquivo .html passa a receber alguns atributos do Flask. Para referenciar o arquivo de estilização, é utilizado o comando <link rel= "stylesheet" type= "text/css" href="{{ url_for('static',filename='style.css') }}">, que procura o style.css na pasta static, conforme supracitado. Além disso, são colocados os itens interiores da tag <head>, com exceção da tag <meta>, dentro das marcações {% block head %}{% endblock %}. A tag <title> recebe {% block title %}{% endblock %} em seu interior e a tag <body> recebe {% block body %} {% endblock %}. Por fim, a tag <table> inicial é substituída pela sintaxe {{ table| safe }}, onde será colocado o resultado do protocolo GET da classe Contratos supracitada.

A estrutura final da pasta 'interface' onde foi desenvolvido o projeto é a apresentada na figura 26 a seguir. A pastas '__pycache__', 'Lib\site-packages', 'Scripts' e os arquivos '.gitignore', 'pyvenv.cfg' e 'requirements.txt' são os documentos gerados pela preparação de ambiente virtual, anteriormente explicadas. O arquivo .json é a chave de acesso do Google API. Os arquivos 'sheets_backend_code.py', 'home.html', 'search.html', 'visualizar.html', 'style.css', 'models.py' e 'app.py' estão presentes respectivamente em Apêndice A, Apêndice B, Apêndice C, Apêndice D e Apêndice E, Apêndice F e Apêndice G.

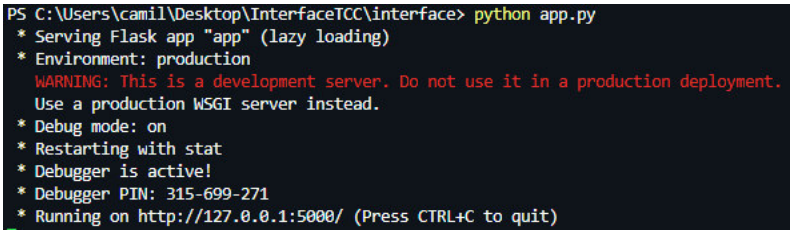
Figura 26 - Árvore de arquivos da pasta de desenvolvimento da página web.



Fonte: (AUTORIA PRÓPRIA, 2020)

Para ativar a aplicação web é dado o comando “python app.py” na linha de comando do editor de código. A resposta da máquina é a hospedagem da página no *localhost* do computador, na porta 5000, que pode ser acessado pelo endereço http://127.0.0.1:5000/. É importante salientar que esse ambiente criado é para desenvolvimento e não para implementação, ou *deploy*. As mensagens devolvidas pela máquina ao rodar o aplicativo estão presentes na figura 27 a seguir.

Figura 27 - Iniciação do arquivo ‘app.py’ no *localhost* do computador



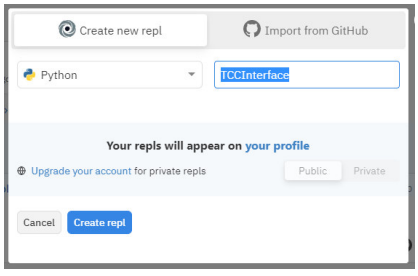
Fonte: (AUTORIA PRÓPRIA, 2020)

3.5 Hospedagem

Para que seja possível o acesso a um site online, é necessário que esse site seja hospedado na internet, e não no localhost da máquina como foi feito. Normalmente, contrata-se um serviço de hospedagem com empresas que oferecem suporte técnico, serviços de gerenciamento, backup, entre outras ferramentas. Porém, um serviço desses custa caro, então para efeito de realização do projeto, existem alguns dispositivos que visam ajudar o desenvolvedor a colocar um site simples no ar para testes.

A principal ferramenta utilizada para isso nesse projeto é o site Repl.it que permite a criação de repositórios online, onde pode-se programar direto no navegador e visualizar o resultado do desenvolvimento automaticamente, o que é ótimo para depuração de erros. Para utilizá-lo, é necessário criar uma conta e um repositório, selecionando a linguagem principal do projeto e dando um nome para o mesmo. A figura 28 a seguir representa essa tela de configuração inicial.

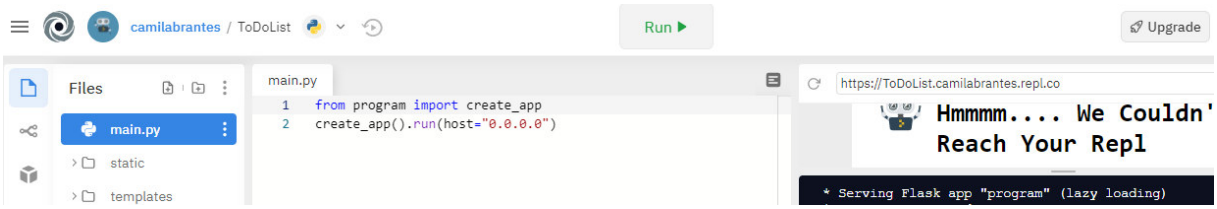
Figura 28- Tela de configuração inicial do projeto pelo Repl.it.



Fonte: (AUTORIA PRÓPRIA, 2020)

Como a aplicação é iniciada com o arquivo ‘app.py’, a linguagem Python é escolhida como principal. Agora, esse arquivo torna-se ‘main.py’, pois é o padrão definido pelo site. Para ativar a aplicação, é necessário apenas clicar no botão ‘Run’. A figura 29 a seguir é uma captura de tela do ambiente de programação do Repl.it.

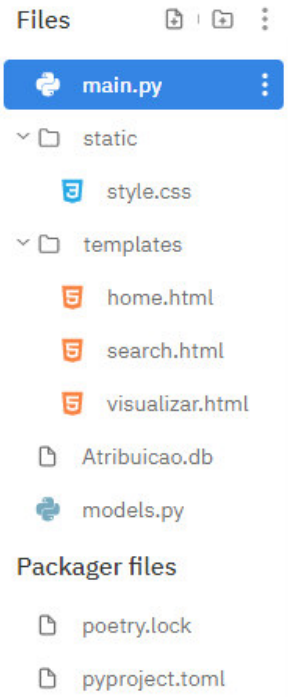
Figura 29 - Exemplo de ambiente de programação do Repl.it.



Fonte: (AUTORIA PRÓPRIA, 2020)

No caso desse ambiente, ao rodar o ‘main.py’, o próprio Repl.it instala as dependências e cria um ambiente virtual específico. Dessa maneira, não existe a necessidade dos comandos do virtualenv. Assim, para colocar a página no ar é necessário colocar o código do ‘app.py’ em ‘main.py’ e subir manualmente os outros arquivos, de maneira que a árvore de arquivos final é a apresentada pela figura 30 a seguir.

Figura 30 - Árvore de arquivos no Repl.it.



Fonte: (AUTORIA PRÓPRIA, 2020)

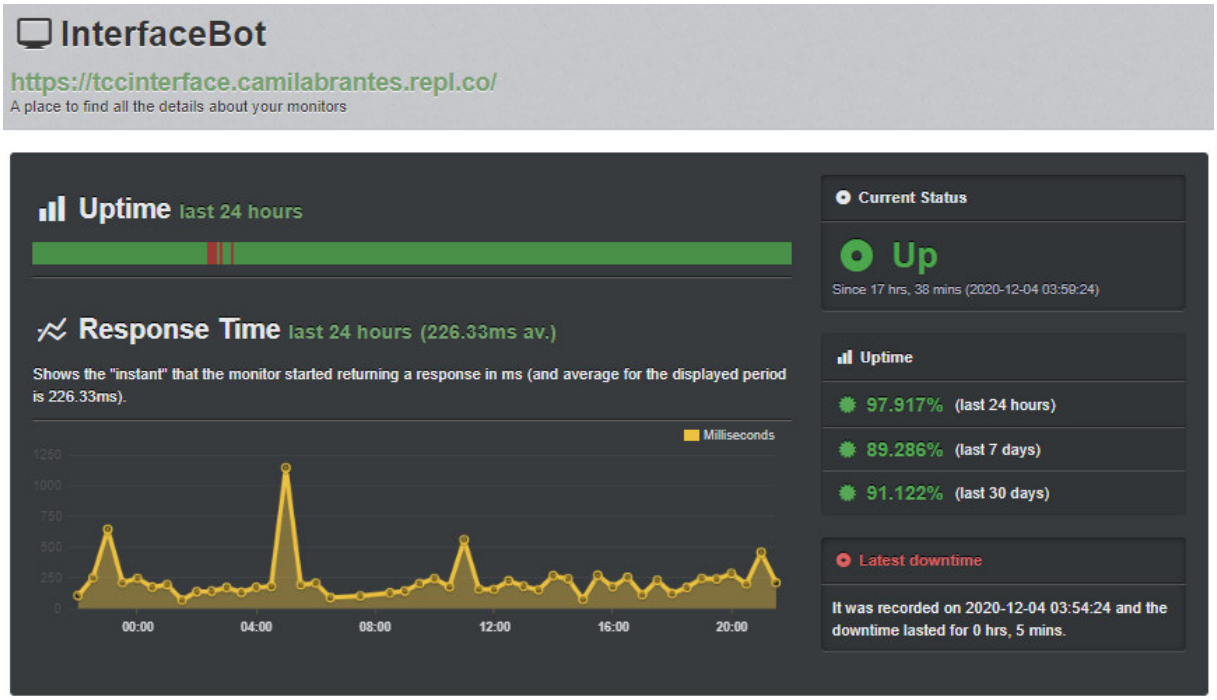
No caso, os arquivos presentes em ‘Packager files’ são os arquivos instalados com a ativação do código. Depois dessa configuração, a aplicação pode ser

inicializada e ser acessada na internet por quem tiver o link de acesso, no caso: https://TCCinterface.camilabrantes.repl.co.

O último passo da metodologia para garantir que o site fique no ar e possa ser acessado a qualquer momento é ativar um robô programado para acessar o site de cinco em cinco minutos. Isso é necessário, pois, como o site Repl.it é feito para facilitar testes, o página cai depois de trinta minutos sem acessos. Esse robô, então, resolverá esse problema e manterá o site sempre online.

Esse robô é configurado online pelo site UptimeRobot, onde é possível adicionar scripts monitores que acessam a página web selecionada de acordo com o tempo configurado. O tipo de monitoramento escolhido por esse projeto é o HTTP, que aciona o método GET descrito no código 'main.py'. A figura 31 a seguir mostra a dashboard de monitoramento do robô configurado.

Figura 31 – Dashboard de monitoramento do site UptimeRobot.



Fonte: (AUTORIA PRÓPRIA, 2020)

4 RESULTADOS E DISCUSSÃO

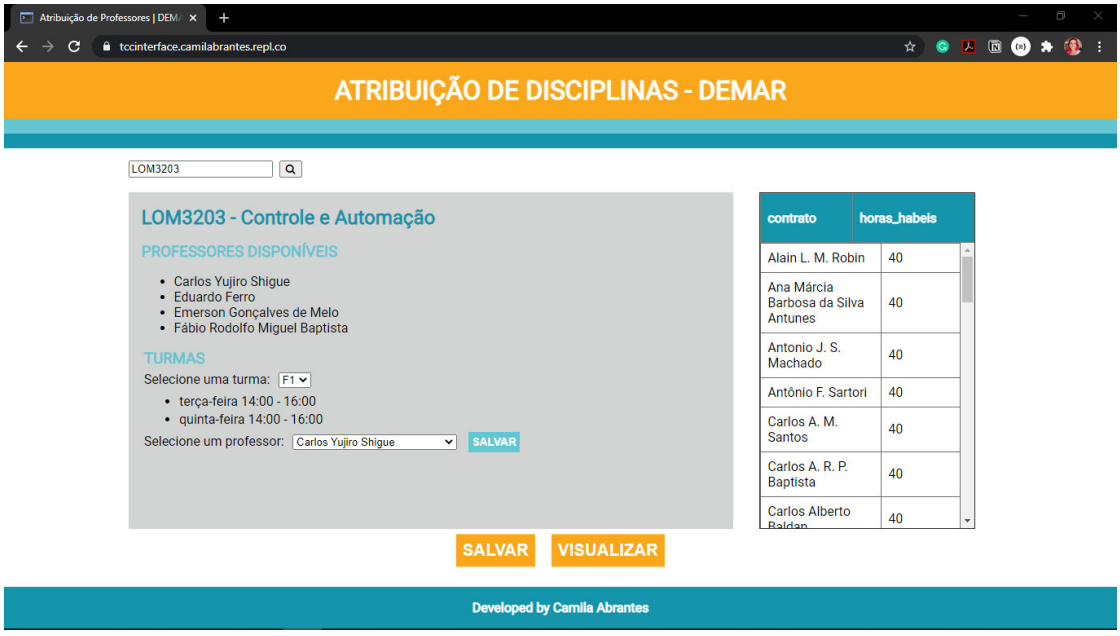
Após seguir todos os passos propostos pela metodologia, o resultado final é website disponível online no endereço https://TCCinterface.camilabrantes.repl.co para acesso a qualquer momento. Esse website contém três telas, sendo elas: página inicial, contendo a tabela de disponibilidade de professores e o tutorial de uso de

ferramenta, apresentada na figura 32 a seguir; página de pesquisa, contendo o resultado da busca por meio do código da disciplina e também a tabela de disponibilidade de professores, apresentada na figura 33, e dando a opção de salvar a seleção mostrando uma mensagem de sucesso, apresentada na figura 34; e a página de visualização, apresentando a tabela de atribuição final dos professores, apresentada na figura 35, e a opção de baixar tal tabela em arquivo .xlsx, apresentada na figura 36 a seguir.

Figura 32 - Captura de tela da página inicial.

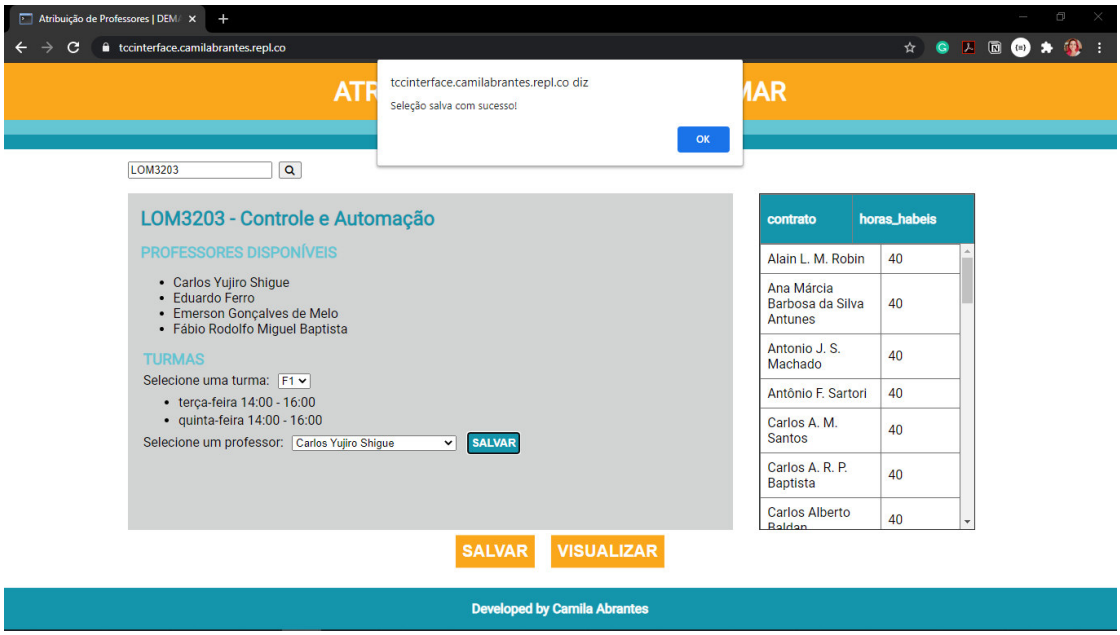
Fonte: (AUTORIA PRÓPRIA, 2020)

Figura 33 - Captura de tela da página de busca.



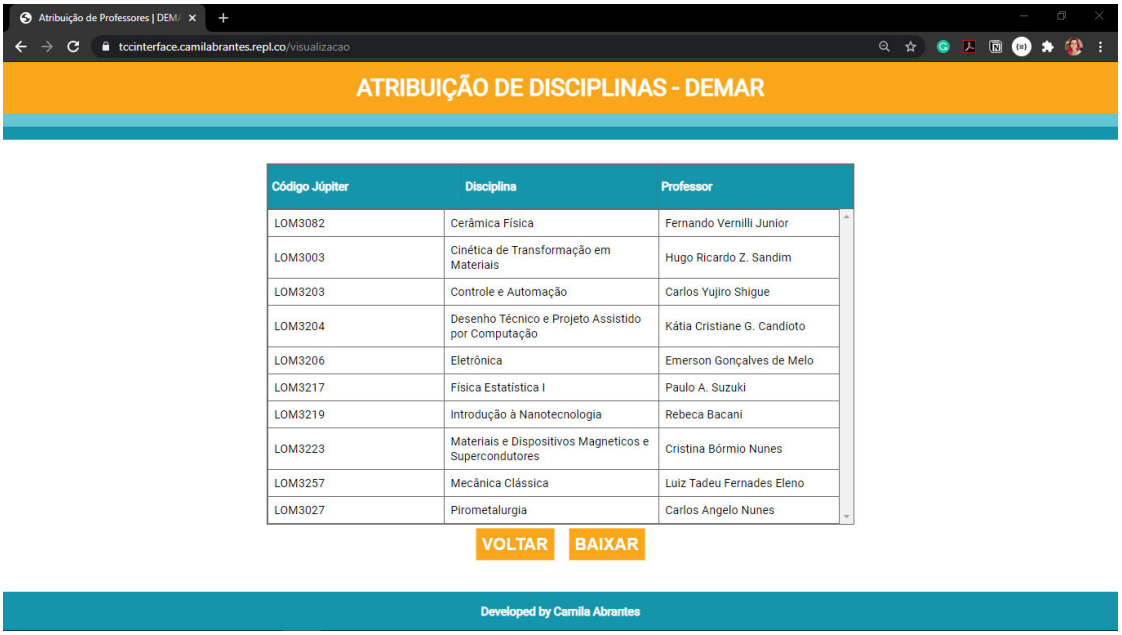
Fonte: (AUTORIA PRÓPRIA, 2020)

Figura 34 - Captura de tela da mensagem flash.



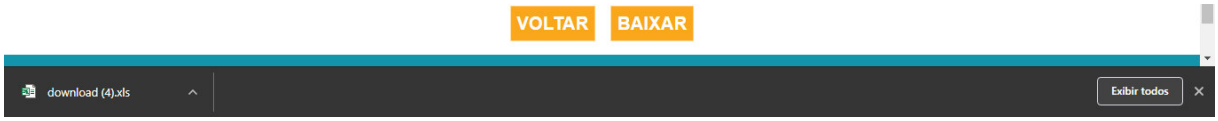
Fonte: (AUTORIA PRÓPRIA, 2020)

Figura 35 - Captura de tela da página de visualização.



Fonte: (AUTORIA PRÓPRIA, 2020)

Figura 36 - Captura de tela do *download* realizado pelo botão ‘baixar’.



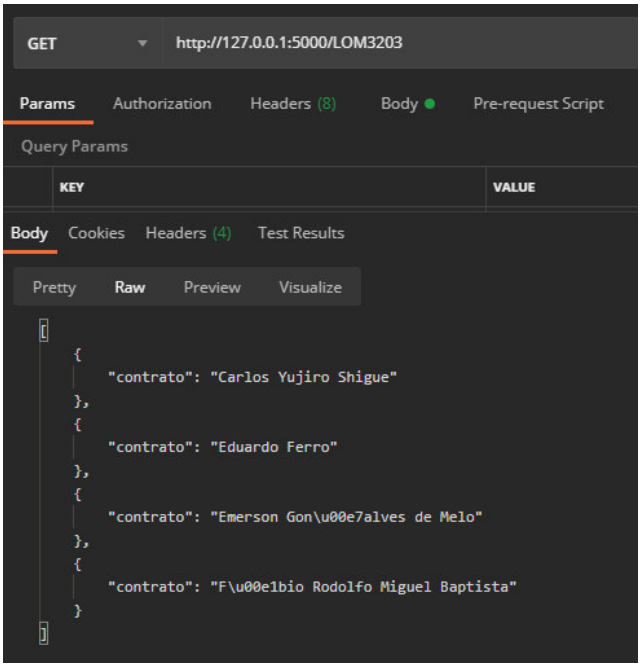
Fonte: (AUTORIA PRÓPRIA, 2020)

Como é possível ver pelas imagens anteriores, o protocolo GET que leva os dados dos nomes dos professores e suas horas hábeis para a tabela na página inicial está funcionando. A integração com o front-end também foi realizada com sucesso,

apresentando-se de acordo com o protótipo da junção apenas dos arquivos .html e .css. Além disso, o protocolo POST que retorna a página de busca ao clicar no botão de pesquisa está funcionando, assim como os botões ‘visualizar’ e ‘voltar’ também estão retornando as páginas corretas. O botão ‘baixar’ está retornando o download de uma planilha do Excel e os botões ‘salvar’ estão retornando mensagens de sucesso, conforme o planejado.

A parte lógica do back-end também foi realizada. Isso pode ser visualizado com o programa Postman. Essa ferramenta serve para documentar as requisições HTTP feitas por uma API, sendo possível operar as requisições por lá. A primeira requisição GET da class Contratos está funcionando, conforme é visto na captura de tela do site. A figura 37 a seguir mostra o resultado da requisição GET da class Disciplinas, que consulta os professores disponíveis para uma disciplina, a partir do código júpiter da mesma.

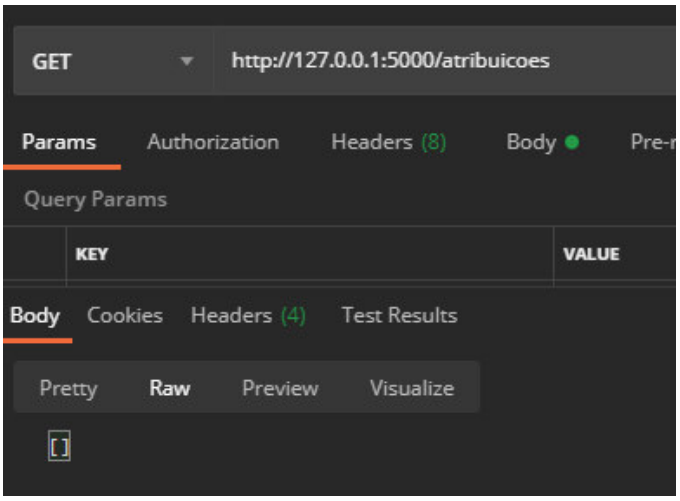
Figura 37 - Resultado da requisição GET da class Disciplinas.



Fonte: (AUTORIA PRÓPRIA, 2020)

A figura 38 a seguir mostra o resultado da requisição GET da class Atribuicoes, que retorna a lista dos professores cadastrados para ministrar uma disciplina, sem ter nada cadastrado nessa etapa.

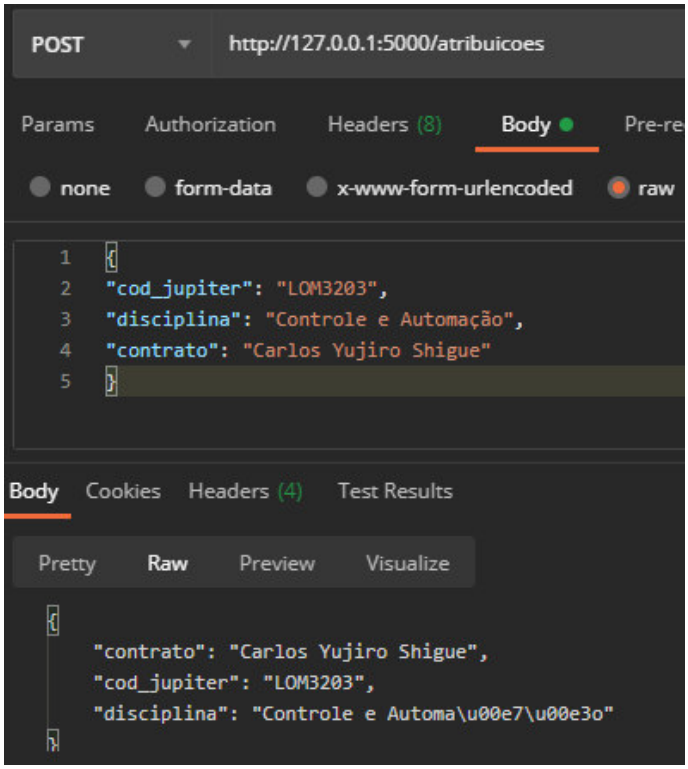
Figura 38 - Resultado inicial da requisição GET da class Atribuicoes.



Fonte: (AUTORIA PRÓPRIA, 2020)

Como visto, essa primeira requisição GET dessa class retorna uma lista vazia, pois nenhum professor foi cadastrado. A figura 39 a seguir mostra o resultado da requisição POST da class Atribuicoes, que cadastra um professor para uma determinada disciplina e seu respectivo código júpiter.

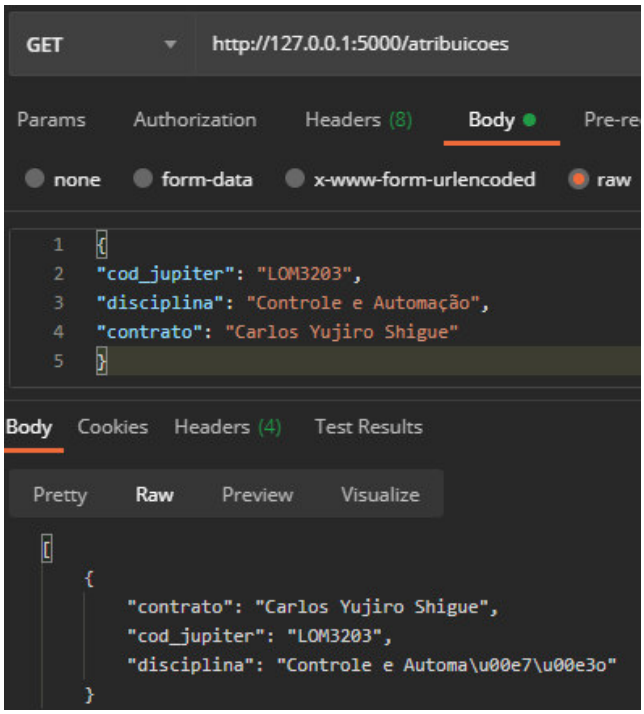
Figura 39 - Resultado da requisição POST da class Atribuicoes.



Fonte: (AUTORIA PRÓPRIA, 2020)

Por fim, a figura 40 mostra o resultado de uma segunda requisição GET feita depois do cadastro de um professor pelo método POST.

Figura 40 - Resultado final da requisição GET da class Atribuicoes.



Fonte: (AUTORIA PRÓPRIA, 2020)

5 CONCLUSÃO

O tema proposto para o projeto salientou uma necessidade importante do Departamento de Materiais da Escola de Engenharia de Lorena. A automatização do processo de organizar os docentes e as disciplinas traz muitos benefícios como: redução de tempo envolvido nessa tarefa atualmente realizada manualmente, que pode ser utilizado para atividades acadêmicas mais desafiadoras; visualização agradável dos processos, não sendo mais necessária uma planilha e programação de fórmulas; e disponibilidade online dos dados, permitindo o acesso e a atualização dos dados de onde o usuário estiver.

O desenvolvimento desse trabalho possibilitou o entendimento das várias etapas que compõem o desenvolvimento de um site. Partindo da leitura de documentos online e utilização do Google Cloud Platform, uma habilidade muito valorizada pelo mercado de trabalho, passando para a criação de uma database em Python e a transformação da mesma para SQL. Entendendo e arquitetando estruturas em HTML, estilizando-as com CSS e integrando-as com o *back-end* por meio do Python. Criando requisições HTTP para consulta do banco de dados construído,

hospedando o site e mantendo-o online. Finalizando com um *website* disponível a qualquer momento em um endereço na internet.

Algumas funcionalidades não foram implementadas, como a importação das turmas das disciplinas e outras requisições HTTP que foram definidas no código ‘app.py’, que funcionam com o site online, porém não foram integradas com o front-end. Para que sejam desenvolvidas essas integrações, será necessário um maior entendimento do funcionamento do Flask e do Flask-RESTful em conjunto com o arquivo HTML. Outra oportunidade para isso seria o uso mais avançado do JavaScript, que também deverá ser mais estudado e aprofundado para realizar sua implementação nesse projeto.

Contudo, o desenvolvimento de uma página web sempre possui espaço para otimização, melhoria e novos atributos. Dessa maneira, o projeto será continuado até que seja finalizado e entregue para o uso do chefe de departamento, já que mostra-se uma ferramenta de grande utilidade.

REFERÊNCIAS

BEAULIEU, Alan. **Aprendendo SQL:** *dominando os fundamentos de SQL*. Sebastopol: O'Reilly. 2010. Disponível em: <https://books.google.com.br/books?id=63iYDwAAQBAJ&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 16 de novembro 2020.

BENYON, David. **Designing User Experience:** *a guide to HCI, UX and interaction design*. 4. ed. Harlow, Uk: Pearson, 2019. Disponível em: <https://books.google.com.br/books?id=MXqFDwAAQBAJ&printsec=frontcover&hl=pt-BR&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false>. Acesso em 17 de novembro de 2020.

CARBONNELLE, Pierre. **PYPL PopularitY of Programming Language**. 2020. Disponível em: <http://pypl.github.io/PYPL.html>. Acesso em: 03 de novembro de 2020.

COPELAND, Rick. **Essential SQLAlchemy**. Sebastopol: O'Reilly. 2008. Disponível em: <https://books.google.com.br/books?id=septpU7dELIC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 05 de novembro de 2020.

COSTA, Rogério Luís de C.. **SQL**: *guia prático*. 2. ed. Rio de Janeiro: Brasport. 2007. Disponível em: <https://books.google.com.br/books?id=3Lxv-q6-S3MC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 16 de novembro de 2020.

CROWDER, David A.. **Construindo Web Sites para Leigos**. 4. ed. Rio de Janeiro: Alta Books. 2011.

DE SOUZA, Ivan. Rock Content. **Framework:** *descubra o que é, para que serve e por que você precisa de um para o seu site.* 2019. Disponível em: <https://rockcontent.com/br/blog/framework/>. Acesso em: 05 de novembro de 2020.

ECMA INTERNATIONAL. **The JSON Data Interchange Syntax**. 2017. Disponível em: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>. Acesso em: 04 de novembro de 2020.

ELENO, Luiz T. F.. **Demar :: EEL :: USP :: Departamento de Engenharia de Materiais - DEMAR**. 2020. Disponível em: <http://www.demar.eel.usp.br/>. Acesso em: 03 de novembro de 2020

FIGMA. **A Free, Online UI Design Tool For Teams | Figma**. 2020. Disponível em: <https://www.figma.com/ui-design-tool/>

FLANAGAN, David. **JavaScript:** *o guia definitivo*. 6. ed. Sebastopol: O'Reilly. 2013. Disponível em:

<https://books.google.com.br/books?id=zWNyDgAAQBAJ&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 17 de novembro 2020.

FREDRICH, Todd. RestApiTutorial.Com. **Using HTTP Methods for RESTful Services**. Disponível em: <https://www.restapitutorial.com/lessons/httpmethods.html>. Acesso em: 22 de novembro de 2020.

GRINBERG, Miguel. **Flask Web Development**. 2. ed. Sebastopol: O'Reilly. 2018. Disponível em: <https://books.google.com.br/books?id=cVlPDwAAQBAJ&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 05 de novembro de 2020.

KENNEDY, Bill; MUSCIANO, Chuck. **HTML & XHTML:** *The Definitive Guide*. 5. ed. Sebastopol: O'Reilly. 2002. Disponível em: <https://books.google.com.br/books?id=O5Vpwv0wTYUC&printsec=frontcover&hl=pt-BR&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false>. Acesso em: 11 de novembro de 2020.

KRISHNAN, S. P. T.; GONZALEZ, Jose L. Ugia. **Building Your Next Big Thing with Google Cloud Platform**: *a guide for developers and enterprise architects.* New York: Apress. 2015. 371 p.

LUTZ, Mark. **Programming Python**. 4. ed. Sebastopol: O'Reilly. 2011. 1584 p.

MASSE, Mark. **REST API Design Rulebook**: *designing consistent RESTful web service interfaces*. Sebastopol: O'Reilly, 2012. Disponível em: <https://books.google.com.br/books?id=eABpzyTcJNIC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 22 de novembro 2020.

MCKINNEY, Wes. **Python for Data Analysis**. Sebastopol: O'Reilly. 2013. 452 p.

MICROSOFT AZURE. **O que é computação em nuvem?**: *um guia para iniciantes*. 2020. Disponível em: <https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing/>. Acesso em: 17 de novembro de 2020.

OLIPHANT, Travis E.. **Python for Scientific Computing.** Computing in Science & Engineering. 2007. 9(3), 10–20. doi:10.1109/mcse.2007.58

SILVA, Maurício Samy. **Construindo Sites com CSS e (X)HTML:** *sites controlados por folhas de estilo em cascata.* São Paulo: Novatec. 2008. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=BxKWAwAAQBAJ&oi=fnd&pg=PA10&#v=onepage&q&f=false>. Acesso em: 11 de novembro de 2020.

SILVA, Maurício Samy. **JavaScript:** *guia do programador.* São Paulo: Novatec. 2010. Disponível em: <https://books.google.com.br/books?id=BB9WDQAAQBAJ&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 16 de novembro de 2020.

SPECOPS SOFTWARE. **The most searched for programming languages around the world**. 28 de outubro de 2020. Disponível em: <https://specopssoft.com/blog/most-searched-for-programming-languages/>. Acesso em: 03 de novembro 2020.

STONE, Debbie; JARRETT, Caroline; WOODROFFE, Mark; MINOCHA, Shailey. **User Interface Design and Evaluation**. San Francisco: Elsevier, 2005. Disponível em:<https://books.google.com.br/books?id=VvSoyqPBPbMC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 17 de novembro de 2020.

TAURION, Cezar. **Cloud Computing - Computação em Nuvem**: *transformando o mundo da tecnologia da informação*. Rio de Janeiro: Brasport, 2009. Disponível em: <https://books.google.com.br/books?id=mvir2X-A2mcC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 17 de novembro de 2020.

THE PALLETS PROJECT. **Quickstart**. 2010. Disponível em: <https://flask.palletsprojects.com/en/1.1.x/quickstart/>. Acesso em: 05 de novembro de 2020.

TOMLIN VENTURES LTD. **The HTML5 Page Structure:** *How To Structure Pages Using The New HTML5 Elements*. 2011. Disponível em: <http://www.basewebmaster.com/html/html5-page-structure.php>. Acesso em: 11 de novembro de 2020.

UNIVERSIDADE DE SÃO PAULO. **Jupiterweb**. 2020. Disponível em: <https://uspdigital.usp.br/jupiterweb/>. Acesso em: 22 de novembro de 2020.

UNIVERSIDADE DE SÃO PAULO. **REGIMENTO GERAL DA UNIVERSIDADE DE SÃO PAULO**, estabelecido pela resolução nº 3745, de 19 de outubro de 1990. Disponível em: <http://www.leginf.usp.br/wp-content/uploads/Regimento-Geral-da-USP.pdf>. Acesso em : 03 de novembro de 2020.

VAN ROSSUM, Guido. **An Introduction to Python**. Bristol: Network Theory Limited. 2003. 115 p.

VAN ROSSUM, Guido. **The Python Library Reference**. Release 3.6.0. Python Software Foundation. 2017. 1888 p. Disponível em: <https://scicomp.ethz.ch/public/manual/Python/3.6.0/library.pdf>. Acesso em: 04 de novembro de 2020.

WTFORMS. **WTForms**. 2008. Disponível em: <https://wtforms.readthedocs.io/en/2.3.x/>. Acesso em: 05 de novembro de 2020.

W3SCHOOLS. **Tryit Editor v3.6**. 2020. Disponível em: <https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_head>. Acesso em: 17 de novembro de 2020.

APÊNDICE A - CÓDIGO INICIAL

```
import pandas as pd
from google.oauth2 import service_account
from google.cloud import storage
from googleapiclient import discovery
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow,Flow
from google.auth.transport.requests import Request
from datetime import datetime
from flask import Flask, request, render_template, session,
redirect
import numpy as np
import sqlite3
import pickle
import os
import json

SCOPES =
["https://www.googleapis.com/auth/content",'https://spreadshee
ts.google.com/feeds','https://www.googleapis.com/auth/drive']
SERVICE_ACCOUNT_FILE = 'tcc-camila-290301-bad5d38b71bd.json'
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]=SERVICE_ACCOUNT_F
ILE
credentials =
service_account.Credentials.from_service_account_file(SERVICE_
ACCOUNT_FILE, scopes= SCOPES)
SAMPLE_SPREADSHEET_ID_input =
'1izntdDfUwW4hSl3fQOG0D_31hQJ8Ne3LVuc15C8V56o'
SAMPLE_RANGE_NAME = 'A2:E'

def read_spreadsheet():
    service = build('sheets','v4',credentials=credentials)
    sheet = service.spreadsheets()
    result_input =
    sheet.values().get(spreadsheetId=SAMPLE_SPREADSHEET_ID_inpu
    t,range=SAMPLE_RANGE_NAME).execute()
```

```
    values_input = result_input.get('values', [])
    df=pd.DataFrame(values_input[1:], columns=values_input[0])
    df.dropna()
    return df

planilha = read_spreadsheet()
print(planilha)

conn = sqlite3.connect('Atribuicao.db')
c = conn.cursor()
c.execute('CREATE TABLE ATRIBUICAO (Contrato, Horas Hábeis,
DISCIPLINA, C Júpiter, A)')
conn.commit()

planilha.to_sql('ATRIBUICAO', conn, if_exists='replace', index
= False)
```

APÊNDICE B - ARQUIVO HOME.HTML

```
<!DOCTYPE html>
<html>
<style>
    tbody {height: 350px;}
</style>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Atribuição de Professores | DEMAR </title>
    <link href="https://fonts.googleapis.com/css2?family=Robot
    o&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/
    ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" type="text/css" href="{{ url_for('s
    tatic',filename='style.css') }}">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/
    2.1.3/jquery.min.js"></script>
</head>

<body>
    <header>
        <h1>ATRIBUIÇÃO DE DISCIPLINAS - DEMAR</h1>
        <div class=faixa1>
            <p></p>
        </div>
        <div class=faixa2>
            <p></p>
        </div>
    </header>
    <section class="content">
        <header class="busca_disciplina">
```

```
        <form class="example" method="post" action="">
            <input class="form-
            control" type="text" placeholder="Pesquise uma
            Disciplina" aria-label="Search">
            <button type="submit"><i class="fa fa-
            search"></i></button>
        </form>
</header>
<div class="row">
  <article class="column left">
    <header>
      <h2>COMO UTILIZAR ESSA FERRAMENTA?</h2>
    </header>
    <br>
    <h3>1- Pesquise uma disciplina</h3>
    <br>
    <p>Na barra de pesquisa acima, pesquise uma disciplina
    com o seu código.</p>
    <br>
    <h3>2- Escolha um professor</h3>
    <br>
    <p>Após fazer a pesquisa, o site retornará os professo
    res disponíveis. Escolha uma das opções por meio da cai
    xa de seleções.</p>
    <br>
    <h3>3- Salvar!</h3>
    <br>
    <p>Clique no botão 'salvar' para salvar a escolha.</p>
    <br>
    <p><b>OBS:</b>  Cheque a disponibilidade dos professor
    es na tabela a direita</p>
  </article>
   <div class="column right">
      {{ table | safe }}
```

```
      </div>
    </div>
    <div class="container">
      <div class="center">
        <a href=#><button class="myButton" onClick="alert('Sel
        eção salva com sucesso!')">SALVAR</button></a>
        <a href=visualizacao><button class="myButton">VISUALIZ
        AR</button></a>
      </div>
    </div>
  </section>

  <footer>
    Developed by Camila Abrantes
  </footer>

</body>
</html>
```

APÊNDICE C - ARQUIVO SEARCH.HTML

```
<!DOCTYPE html>
<html>
<style>
  tbody {height: 350px;}
</style>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Atribuição de Professores | DEMAR </title>
  <link href="https://fonts.googleapis.com/css2?family=Roboto&
  display=swap" rel="stylesheet">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/aj
  ax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
  <link rel="stylesheet" type="text/css" href="{{ url_for('sta
  tic',filename='style.css') }}">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.
  1.1/jquery.min.js">
  </script>
</head>

<body>
  <header>
    <h1>ATRIBUIÇÃO DE DISCIPLINAS - DEMAR</h1>
    <div class=faixa1>
      <p></p>
    </div>
    <div class=faixa2>
      <p></p>
    </div>
  </header>
```

```
<section class="content">
  <header class="busca_disciplina">
    <form class="example" method="post" action="">
      <input class="form-
      control" type="text" placeholder="Pesquise uma Discipli
      na" aria-label="Search">
      <button type="submit"><i class="fa fa-
      search"></i></button>
    </form>
  </header>
  <div class="row">
    <article class="column left">
      <header>
        <h2>LOM3203 - Controle e Automação</h2>
      </header>
      <div class = "profes_disp">
        <h3>PROFESSORES DISPONÍVEIS</h3>
        <ul>
          <li>Carlos Yujiro Shigue</li>
          <li>Eduardo Ferro</li>
          <li>Emerson Gonçalves de Melo</li>
          <li>Fábio Rodolfo Miguel Baptista</li>
        </ul>
      </div>
      <div class = "turmas_disp">
        <h3>TURMAS</h3>
        <label for="turmas">Selecione uma turma:</label>
          <select name="turmas" id="turmas">
            <option value="F1">F1</option>
            <option value="F2">F2</option>
          </select>
        <ul>
          <li>terça-feira 14:00 - 16:00</li>
          <li>quinta-feira 14:00 - 16:00</li>
```

```
            </ul>
            <label for="professores">Selecione um professor:
            </label>
                <select name="professores" id="professores">
                  <option value="Carlos Yujiro Shigue">Carlos Yu
                  jiro Shigue</option>
                  <option value="Eduardo Ferro">Eduardo Ferro</o
                  ption>
                  <option value="Emerson Gonçalves de Melo">Emer
                  son Gonçalves de Melo</option>
                  <option value="Fábio Rodolfo Miguel Baptista">
                  Fábio Rodolfo Miguel Baptista</option>
                </select>
              <a href="/"><button onClick="alert('Seleção salva
              com sucesso!')">SALVAR</button></a>
          </div>
        </article>
        <div class="column right">
        {{ table | safe }}
        </div>
      </div>
      <div class="container">
        <div class="center">
          <a href="/"><button class="myButton" onClick="alert('S
          eleção salva com sucesso!')">SALVAR</button></a>
          <a href=visualizacao><button class="myButton">VISUALIZ
          AR</button></a>
      </div>
      </section>
    <footer>
      Developed by Camila Abrantes
    </footer>
</body>
</html>
```

APÊNDICE D – ARQUIVO VISUALIZAR.HTML

```
<!DOCTYPE html>
<html>
<style>
    section.content {padding: 1rem;}
    table.demo {
        height: 100px;
        border: 1px solid #6A6E6E;
        border-collapse: collapse;
        padding: 5px;
        margin-left: auto;
        margin-right: auto;
        margin-top: 1rem;
        text-align: center;
    }
    table.demo body {
        height: 350px !important;
        overflow-y: scroll;
    }
    table.demo thead th,
    tbody td {
        width: 10%; //or what you want
        float: left;
        border: 1px solid #6A6E6E;
        padding: 5px;
    }
    table.demo tr {
        width: 100%;
    }
    table.demo th {
        background: #1495AB;
    }
</style>
```

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Atribuição de Professores | DEMAR </title>
    <link href="https://fonts.googleapis.com/css2?family=Robot
    o&display=swap" rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="{{ url_for('s
    tatic',filename='style.css') }}">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/
    3.5.1/jquery.min.js"></script>
</head>

<body>
    <header>
        <h1>ATRIBUIÇÃO DE DISCIPLINAS - DEMAR</h1>
        <div class=faixa1>
            <p></p>
        </div>
        <div class=faixa2>
            <p></p>
        </div>
    </header>
    <section class="content">
        <table class="demo" id="dvData">
            <thead>
                <tr>
                    <th>Código Júpiter</th>
                    <th>Disciplina</th>
                    <th>Professor</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>LOM3082</td>
```

```
    <td>Cerâmica Física</td>
    <td>Fernando Vernilli Junior</td>
</tr>
<tr>
    <td>LOM3003</td>
    <td>Cinética de Transformação em Materiais
    </td>
    <td>Hugo Ricardo Z. Sandim</td>
</tr>
<tr>
    <td>LOM3203</td>
    <td>Controle e Automação</td>
    <td>Carlos Yujiro Shigue</td>
</tr>
<tr>
    <td>LOM3204</td>
    <td>Desenho Técnico e Projeto Assistido po
    r Computação</td>
    <td>Kátia Cristiane G. Candioto&amp;nbsp;</td>
</tr>
<tr>
    <td>LOM3206</td>
    <td>Eletrônica</td>
    <td>Emerson Gonçalves de Melo</td>
</tr>
<tr>
    <td>LOM3217</td>
    <td>Física Estatística I</td>
    <td>Paulo A. Suzuki</td>
</tr>
<tr>
    <td>LOM3219</td>
    <td>Introdução à Nanotecnologia</td>
    <td>Rebeca Bacani</td>
```

```
            </tr>
            <tr>
                <td>LOM3223</td>
                <td>Materiais e Dispositivos Magneticos e
                Supercondutores</td>
                <td>Cristina Bórmio Nunes</td>
            </tr>
            <tr>
                <td>LOM3257</td>
                <td>Mecânica Clássica</td>
                <td>Luiz Tadeu Fernades Eleno</td>
            </tr>
            <tr>
                <td>LOM3027</td>
                <td>Pirometalurgia</td>
                <td>Carlos Angelo Nunes</td>
            </tr>
            <tbody>
    </table>
    <div class="container">
        <div class="center">
            <a href="/"><button class="myButton">VOLTAR</b
            utton></a>
            <a href="/"><button class="myButton" id="btnEx
            port">BAIXAR</button></a>
        </div>
    </div>
</section>
<footer>
Developed by Camila Abrantes
</footer>
<script>
    $("#btnExport").click(function (e) {
```

```
            window.open('data:application/vnd.ms-
            excel,' + $('#dvData').html());
            e.preventDefault();
        });
    </script>
</body>
</html>
```

APÊNDICE E - ARQUIVO STYLES.CSS

```
html,body{
font-family: Roboto;
font-style: normal;
}

header, section, article, footer, p,main,h1,h2,h3,h4 {
margin:0;
padding: 0;
}

html{
height: 100%;
}

body{
background:#fff;
margin:0 auto;
margin-bottom:0;
height: 100vh;
display: flex;
flex-direction: column;
}

body>header h1{
background: #FAA71B;
color:#fff;
left: 0px;
padding:1rem 0;
text-align: center;
margin-top:0;
width:100%;
}
```

```
.faixa1 {
width: 100%;
height: 18px;
left: 0px;
background: #64C6D3;
}

.faixa2 {
width: 100%;
height: 18px;
left: 0px;
background: #1495AB;
}

section{
display: block;
width:80%;
padding: 0 1rem 1rem 1rem;
margin: auto;
margin-bottom:0;
}

section>header {
padding:0  0 1rem 0;
font-size:2rem;
}

article{
background: rgba(106, 110, 110, 0.30);
padding: 1rem;
}

article>header h2{
```

```
color: #1495AB;
font-weight: 800;
margin-bottom:1rem;
}

article button {
background-color:#64C6D3;
border:0;
display:inline-block;
cursor:pointer;
color:#ffffff;
font-size:1vw;
font-weight:bold;
padding:0.3rem;
margin-top:0.3rem;
text-decoration:none;
}

article button:hover {
background-color:#1495AB;
}

.profes_disp h3, .turmas_disp h3 {
color: #64C6D3;
font-weight: bold;
}

* {
box-sizing: border-box;
}

.row {
display: flex;
}
```

```
.column {
float: left;
vertical-align: top;
}

.left {
width: 70%;
margin-right: 1rem;
}

.right {
width: 25%;
margin-left: 1rem;
margin-right: 0;
}

.row:after {
content: "";
display: table;
clear: both;
}

.turmas_disp * {
margin: 0.2rem;
}

table {
border-spacing: 1;
border-collapse: collapse;
background: white;
overflow-y: scroll;
max-width: 800px;
width: 100%;
```

```
margin: 0 auto;
position: relative;
}

thead, tbody{
display: block;
}

tbody{
overflow-y: scroll;
}

th, td {
padding: 5px;
width:20%;
}

table * {
position: relative;
}

td{
border: 1px solid rgba(106, 110, 110);
}

table td, table th {
padding: 8px;
}

table thead tr {
height: 60px;
background: #1495AB;
font-size: 16px;
color:#FFFFFF;
```

```
width:100%;
}

table tbody tr {
border-bottom: 1px solid #E3F1D5;
}

table tbody tr:last-child {
border: 0;
}

table td, table th {
text-align: left;
}

table td.l, table th.l {
text-align: right;
}

table td.c, table th.c {
text-align: center;
}

table td.r, table th.r {
text-align: center;
}

.container {
height: 50px;
position: relative;
}

.center {
margin: 0;
```

```
position: absolute;
top: 50%;
left: 50%;
-ms-transform: translate(-50%, -50%);
transform: translate(-50%, -50%);}

.center button {
margin-left: 0.5rem;
margin-right: 0.5rem;
}

.myButton {
background-color:#FAA71B;
border:0;
display:inline-block;
cursor:pointer;
color:#ffffff;
font-size:1.5vw;
font-weight:bold;
padding:0.5rem;
margin-top:0.2rem;
text-decoration:none;
}

.myButton:hover {
background-color:#64C6D3;
}

body>footer{
background: #1495AB;
color:#fff;
font-weight: bold;
font-size: 1u;
padding:1rem 0;
```

```
text-align: center;
margin-top:0;
}

.content{
flex:1 0 auto;}
```

APÊNDICE F - ARQUIVO MODELS.PY

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import scoped_session, sessionmaker
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite:///Atribuicao.db', convert_unic
ode=True)
db_session = scoped_session(sessionmaker(autocommit=False,
                                         bind=engine,))
Base = declarative_base()
Base.query = db_session.query_property()

class Contrato(Base):
    __tablename__ = 'ATRIBUICAO'
    __table_args__ = {'extend_existing': True}
    id = Column(Integer, primary_key = True)
    contrato = Column(String(250), nullable=False)
    horas_habeis = Column(Integer, nullable=False)

class Disciplina(Base):
    __tablename__ = 'ATRIBUICAO'
    __table_args__ = {'extend_existing': True}
    id = Column(Integer, primary_key = True)
    cod_jupiter = Column(String(250), nullable=False)
    disciplina = Column(String(250), nullable=False)
    contrato = Column(String(250), nullable=False)

class Atribuicao(Base):
    __tablename__ = 'NOVA_ATRIBUICAO'
    cod_jupiter = Column(String(250), primary_key = True)
    disciplina = Column(String(250), nullable=False)
    contrato = Column(String(250), nullable=False)
    def save(self):
```

```
        db_session.add(self)
        db_session.commit()


def init_db():
    Base.metadata.create_all(bind=engine)

if __name__ == '__main__':
    init_db()
```

APÊNDICE G - ARQUIVO APP.PY

```
from flask import Flask, request, render_template, make_respon
se
from flask_restful import Resource, Api
from json2html import *
from models import Contrato, Disciplina, Atribuicao, db_sessio
n

app = Flask(__name__)
api = Api(app)


class Contratos(Resource):
    def get(self):
        contrato = db_session.query(Contrato).distinct(
            Contrato.contrato).group_by(Contrato.contrato)
        response = [{
            'contrato': i.contrato,
            'horas_habeis': i.horas_habeis,
        } for i in contrato]
        table = json2html.convert(json=response)
        headers = {'Content-Type': 'text/html'}
        return make_response(
            render_template("home.html", table=table), headers
)

    def post(self):
        contrato = db_session.query(Contrato).distinct(
            Contrato.contrato).group_by(Contrato.contrato)
        response = [{
            'contrato': i.contrato,
            'horas_habeis': i.horas_habeis,
        } for i in contrato]
```

```
        table = json2html.convert(json=response)
        headers = {'Content-Type': 'text/html'}
        return make_response(
            render_template("search.html", table=table), heade
rs)


class Visualizar(Resource):
    def get(self):
        headers = {'Content-Type': 'text/html'}
        return make_response(render_template("visualizar.html"
), headers)


class Disciplinas(Resource):
    def get(self, cod_jupiter):
        cod_jupiter = db_session.query(Disciplina).filter_by(
            cod_jupiter=cod_jupiter)
        response = [{'contrato': i.contrato} for i in cod_jupi
ter]
        return response


class Atribuicoes(Resource):
    def get(self):
        cod_jupiter = db_session.query(Atribuicao).all()
        response = [{
            'contrato': i.contrato,
            'cod_jupiter': i.cod_jupiter,
            'disciplina': i.disciplina
        } for i in cod_jupiter]
        return response

    def post(self):
        dados = request.json
```

```
        atribuicoes = Atribuicao(
            contrato=dados['contrato'],
            cod_jupiter=dados['cod_jupiter'],
            disciplina=dados['disciplina'])
        atribuicoes.save()
        response = {
            'contrato': atribuicoes.contrato,
            'cod_jupiter': atribuicoes.cod_jupiter,
            'disciplina': atribuicoes.disciplina
        }
        return response


api.add_resource(Contratos, '/')
api.add_resource(Visualizar, '/visualizacao')
api.add_resource(Disciplinas, '/<string:cod_jupiter>')
api.add_resource(Atribuicoes, '/atribuicoes')

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```