UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

The Avaliator: uma ferramenta *web* para avaliação de qualidade de código-fonte de aprendizes de programação em *Python*

Trabalho de Conclusão de Curso

Erick Santos Resende

São Cristóvão – Sergipe

2024

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Erick Santos Resende

The Avaliator: uma ferramenta *web* para avaliação de qualidade de código-fonte de aprendizes de programação em *Python*

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Professor Doutor Alberto Costa Neto

São Cristóvão – Sergipe

2024

Resumo

Um dos problemas encontrados por professores no ensino da programação é a dificuldade de corrigir códigos-fonte e quantificar a qualidade dos mesmos. Por isso, em 2021 foi criada uma ferramenta que compara os códigos-fonte de professor e alunos, para em seguida determinar a sua qualidade a partir de métricas definidas. Entretanto, a ferramenta não possuia uma interface de utilização, dificultando o processo de avaliação e o tornando desgastante pois requer que sejam salvos manualmente os códigos-fonte dos alunos e do professor. Dessa forma, este projeto trata do desenvolvimento de uma aplicação *web* que avalie códigos-fonte a partir do uso dessa ferramenta, em conjunto com uma conexão com o juiz online The Huxley para acessar automaticamente os dados de exercícios e subsmissões que existem no site. A ferramenta desenvolvida faz com que os professores que já usam o The Huxley na prática das disciplinas de programação tenham um sistema de pontuação de código-fonte que os ajude a determinar os pontos fortes e fracos de cada aluno, pois a ferramenta permite detectar soluções com métricas melhores ou piores que a do professor, o qual pode analisar as razões das soluções serem tão divergentes da sua e entrar em contato diretamente com o aluno sobre aquela solução específica.

Palavras-chave: The Huxley, Avaliação de código-fonte, Software de avaliação.

Abstract

One of the challenges faced by programming educators is the difficulty of grading source code and objectively assessing its quality. In response to this, a tool was developed in 2021 that compares source code from both teachers and students, subsequently determining its quality based on predefined metrics. However, the tool lacked a user-friendly interface, complicating the evaluation process and making it cumbersome as it required manual saving of student and teacher source codes. This project focuses on the development of a web application that evaluates source code using this tool, in conjunction with a connection to the online judge The Huxley to automatically access exercise and submission data on the platform. The developed tool provides programming educators who already use The Huxley in their teaching practices with a source code scoring system, aiding them in identifying each student's strengths and weaknesses. The tool enables the detection of solutions with metrics that are better or worse than those of the teacher, allowing the analysis of reasons for such divergences and facilitating direct communication with the student regarding specific solutions.

Keywords: The Huxley, Source Code Evaluation, Evaluation Software.

Lista de ilustrações

Lista de Quadros

Lista de abreviaturas e siglas

API	Application Programming Interface
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
UFS	Universidade Federal de Sergipe
CSU	Caso de Uso
SGBD	Sistema de Gerenciamento de Banco de Dados

Sumário

1 Introdução

Com o passar dos anos, a tecnologia vem evoluindo ao redor no mundo e se tornando cada vez mais parte do nosso dia a dia, o que gera um interesse das pessoas em ingressar no meio da computação e aprender mais sobre programação. Porém, apesar do aumento no interesse, os indíces de desistências em cursos de computação também são altos. De acordo com o Mapa do Ensino Superior no Brasil (SEMESP, 2021), feito pelo Instituto SEMESP, dentre os 20 cursos de graduação das universidades privadas com o maior número de alunos, o curso de Sistemas de Informação é o que tem a maior taxa de evasão, calculada em 37.6%.

Uma parte importante dos cursos na área de computação é o ensino da programação, que deve despertar no aluno sua curiosidade em aprender mais sobre o tema. Porém, o ensino da mesma é um grande desafio para os professores, já que normalmente precisam corrigir manualmente vários códigos-fonte e analisar a qualidade dos mesmos, o que se torna um trabalho exaustivo e de pouca praticidade.

De acordo com Berssanette (BERSSANETTE; FRANCISCO; BARAN, 2018), os professores não devem se restringir mais apenas a ter conhecimentos em sua área específica, já que boa parte dos alunos tem domínio das mídias digitais para acessar conteúdos. Logo, torna-se necessário o uso de abordagens didáticas que usem as tecnologias digitais. Existem diversos aplicativos que auxiliam o ensino de várias disciplinas diariamente, o avanço da tecnologia veio com uma grande ajuda ao ambiente acadêmico.

Em função disso, muitos professores optam por utilizar os chamados "Juízes *Online*". Segundo Santos (SANTOS; RIBEIRO, 2011), a função principal dos juízes online é a avaliação de códigos-fonte que foram escritos em determinada linguagem, o que os leva a serem utilizados no ambiente acadêmico com o propósito de intensificar o ensino da programação, já que os alunos podem se sentir desafiados ao encontrar problemas e motivados ao resolvê-los. Nesse trabalho iremos dar foco ao juíz *online* conhecido como The Huxley.

O The Huxley (THE. . . , 2023) é uma plataforma de estudos para alunos de programação,

que permite resolver vários exercícios e praticar diversas linguagens de programação. Porém, essa plataforma tem algumas limitações, sendo uma delas que a análise da solução dos problemas é feita de uma forma muito objetiva, onde apenas o problema é dado como resolvido ou não, sem uma análise mais profunda da qualidade do código. Para um aluno que ainda é novato se torna necessária uma análise de diferentes métricas de código, afinal a sua solução pode não ser a melhor possível.

Segundo Martin (MARTIN, 2011), o código mal elaborado pode levar a problemas de manutenção e atrasos no desenvolvimento, o que a longo prazo pode ser crucial para levar o programador a uma produtividade quase nula. Por isso, é de extrema importância que durante o aprendizado os estudantes entendam e procurem melhorar a qualidade dos seus programas, para que no futuro não sejam prejudicados por incorporar maus hábitos de programação sem qualquer alerta.

Dada essa necessidade, Victor (VIEIRA, 2022), graduado em Ciência da Computação pela Universidade Federal de Sergipe, construiu um *software* que, de forma simplificada, recebe duas soluções para um problema, sendo uma do professor e outra do aluno, e então as compara para obter algumas métricas e definir a qualidade da solução do aluno em relação à do professor. Essa qualidade é quantificada por meio de diferentes métricas que serão explicadas no capítulo 2 desse trabalho. Porém, ainda existia espaço para melhora nessa solução, já que, na ferramenta anterior era necessário baixar os códigos-fonte dos alunos e do professor, organizar em pastas e depois rodar a aplicação para comparação via linha de comando. Todo esse processo acabava se tornando um trabalho tedioso e pouco prático. De fato, a ferramenta desenvolvida por Victor (VIEIRA, 2022) na verdade foi pensada para ser integrada, futuramente, a outras interfaces, focando inicialmente no cálculo das métricas e comparação com a solução do professor.

Assim, neste trabalho foi construída uma interface que auxilia os professores a usar esse *software* para análise de código. Dessa forma, tornou-se mais fácil selecionar os códigos-fonte e verificar os resultados, ou seja, os professores têm em mãos uma plataforma que os ajuda a analisar os códigos-fonte e identificar pontos fortes e fracos dos alunos de forma prática e rápida. Além disso, a partir de uma conexão com o próprio The Huxley, é possível utilizar todo o sistema de exercícios e turmas da plataforma, de modo que os alunos respondam os exercícios no juiz *online*, e no sistema proposto por nós estão as métricas de suas submissões, acessíveis apenas para os professores da turma.

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver a ferramenta *web* The Avaliator, que permitirá avaliar automaticamente as submissões dos alunos realizadas na plataforma The Huxley, utilizando um conjunto de métricas

para quantificar a qualidade do código-fonte.

1.1.2 Específico

Os seguintes objetivos específicos foram traçados desde o começo do desenvolvimento deste projeto:

- Desenvolver um módulo que acessa o The Huxley para extrair as informações e também o código-fonte das submissões já realizadas pelo professor e pelos alunos de uma turma.
- Criar uma *API REST* que encapsule a ferramenta já criada anteriormente (VIEIRA, 2022), visando extrair as informações e métricas acerca das submissões de forma padronizada.
- Garantir uma interface com boa usabilidade para o professor acessar e realizar as análises necessárias, auxiliando-o a identificar as submissões que requerem algum tipo de intervenção.
- Disponibilizar ao professor um mecanismo de comparação entre as métricas das suas submissões com as dos alunos, tornando a análise mais fácil.

1.2 Estrutura

De modo a ajudar na navegação pelos capítulos deste trabalho, segue a organização dos mesmos: O primeiro capítulo é a introdução, na qual foi apresentado o problema e detalha as necessidades do trabalho, bem como os objetivos gerais e específicos. O segundo capítulo apresenta as principais funcionalidades do The Huxley relacionadas ao projeto da ferramenta de análise de código-fonte proposta por Vieira (VIEIRA, 2022) e que será usada neste trabalho. O terceiro capítulo contém a especificação de requisitos, a qual descreve os requisitos funcionais e não funcionais da ferramenta, além dos casos de uso e arquitetura. O capítulo 4 aborda ferramentas que foram utilizadas durante o desenvolvimento da aplicação. O quinto capítulo detalha desenvolvimento da ferramenta, que além de detalhar a construção da mesma, também exibe imagens da aplicação. O sexto e último capítulo traz0 as considerações finais e trabalhos futuros, onde é finalizada a discussão do projeto, além de concluir com uma visão do que pode ser feito no futuro.

1.3 Metodologia

De acordo com Severino (SEVERINO, 2014), a primeira tarefa do cientista ao aplicar o seu método é a observação dos fatos, com o objetivo de coletar dados importantes e embasar o seu estudo. Nesse contexto, foi importante entender o funcionamento dos juíz online The Huxley, bem como dificuldades que os professores encontram no ensino da programação. O The Huxley

serviu de base para proporcionar os dados de turmas e problemas, aqui nessa etapa foi necessário identificar como são feitas as requisições *HTTP* no site.

Posteriormente, o estudo da ferramenta construída por Vieira (VIEIRA, 2022) é de grande importância para o desenvolvimento da aplicação, já que, além do uso de sua funcionalidade de avaliar códigos-fonte, também foi necessário transformá-la em uma *API REST* que se comunique com o sistema web.

As reuniões de orientação foram presentes durante todo o desenvolvimento do projeto, sendo um dos pontos tratados a priorização de requisitos, para determinar quais sãos as principais funcionalidades que deverão ser implementadas.

Após o entendimento dos requisitos, foi dado início ao desenvolvimento da aplicação *web*, realizando as conexões com o The Huxley e com a *API* avaliadora de código, de modo que todo o sistema funcione de forma rápida e eficiente. Logo em seguida a ferramenta foi testada e validada com o *stakeholder* e orientador do projeto, Professor Doutor Alberto Costa Neto, e os resultados avaliados para melhorar o seu funcionamento.

2

Ferramentas de análise de código-fonte

Nesse capítulo será abordado todo o embasamento teórico para o desenvolvimento do trabalho, sendo tratados dois pontos: O juiz online The Huxley e a ferramenta de análise e pontuação de código desenvolvida por Vieira (VIEIRA, 2022).

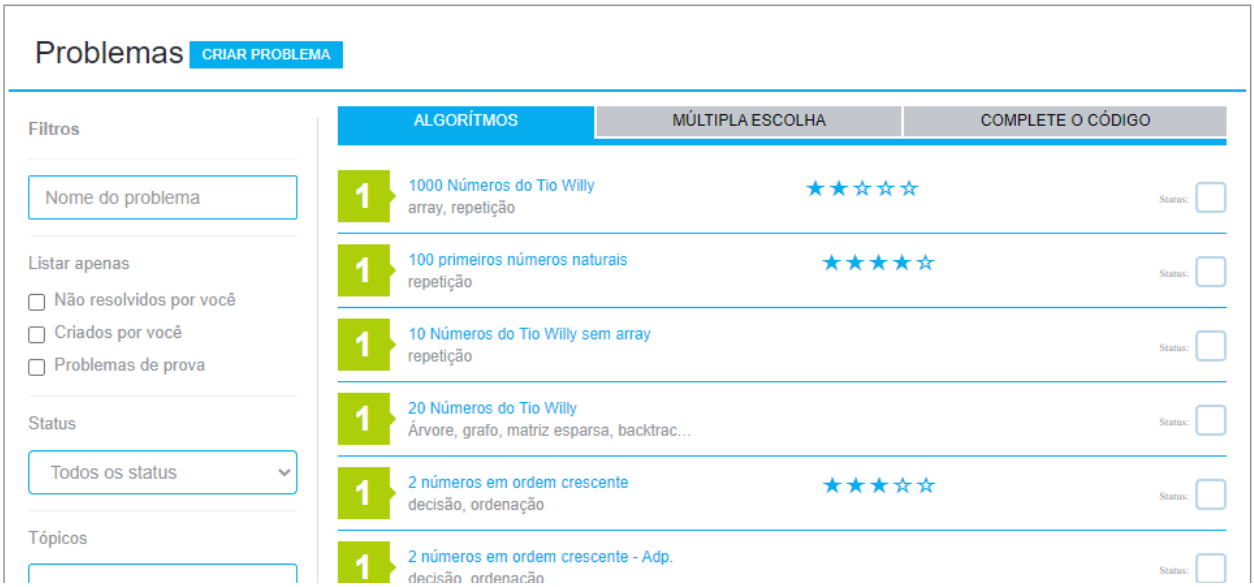
2.1 The Huxley

O The Huxley é um juiz *online* desenvolvido com o objetivo de auxiliar alunos e professores no ensino da programação. Juízes *online* são ferramentas poderosas, já que são bons avaliadores de código, o que acaba tirando esta responsabilidade repetitiva do professor. Segundo Francisco (FRANCISCO, 2016), apesar dos juízes *online* terem sido desenvolvidos com o intuito de realizar competições de programação, alguns de seus benefícios, como o *feedback* automático ao aluno e produtividade no trabalho do professor demonstram que eles também podem ser úteis no ensino das disciplinas de programação.

Na plataforma do The Huxley existe uma diversidade de problemas que os alunos podem solucionar em diferentes linguagens e testar os seus conhecimentos. Essas soluções são armazenadas no site e são acessíveis pelo professor, desde que os alunos estejam na turma do professor e as questões façam parte das atividades a turma.

A Figura 1 exibe a tela dos problemas, os quais são divididos em três tipos: Algoritmos, Múltipla Escolha e Complete o código, cada um desses problemas pode ser resolvido em diferentes linguagens.

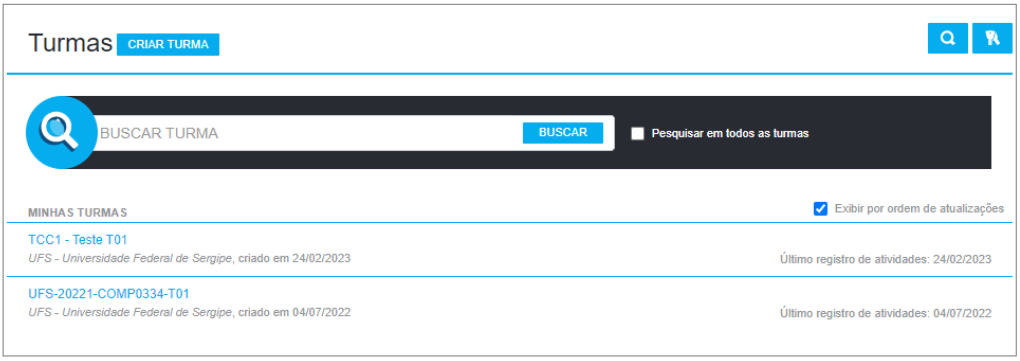
Figura 1 – Lista de problemas e seus tipos



Fonte: Autor (2023)

Visando o ensino em sala de aula, a plataforma disponibiliza aos professores a possibilidade de criar turmas para os seus alunos. Na Figura 2 é possível ver a tela referente à lista de turmas que o usuário faz parte.

Figura 2 – Lista de Turmas



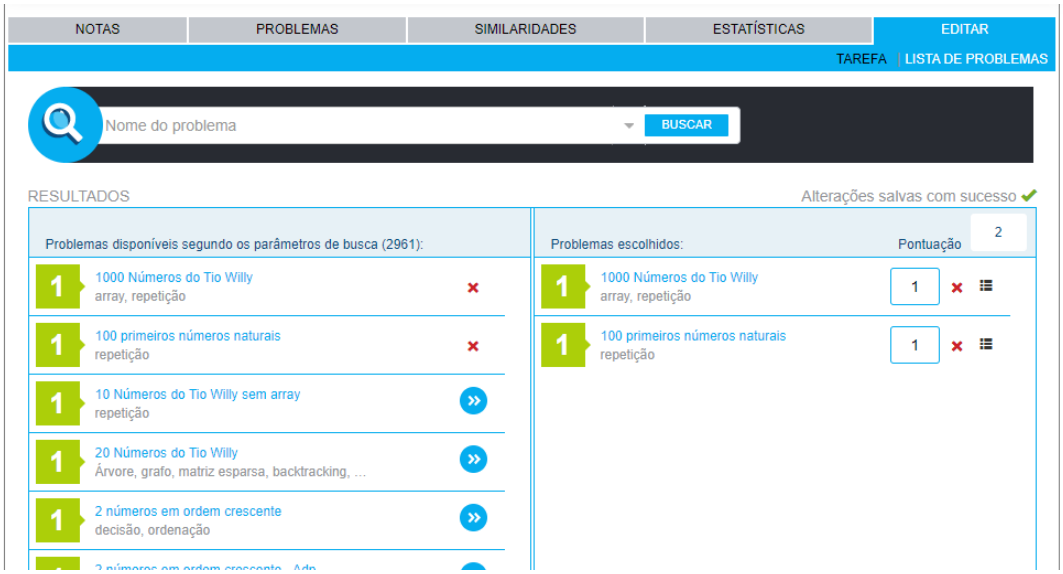
Fonte: Autor (2023)

Na tela da turma, o professor pode criar uma tarefa para que seus alunos respondam. Nesse formulário o professor deve preencher o nome, descrição e a data em que essa tarefa estará disponível, essa tela está ilustrada na Figura 3. Após a criação da tarefa, o professor deve selecionar os problemas que vão compor a mesma, a Figura 4 ilustra essa etapa.

Figura 3 – Tela de criação de tarefa

Fonte: Autor (2023)

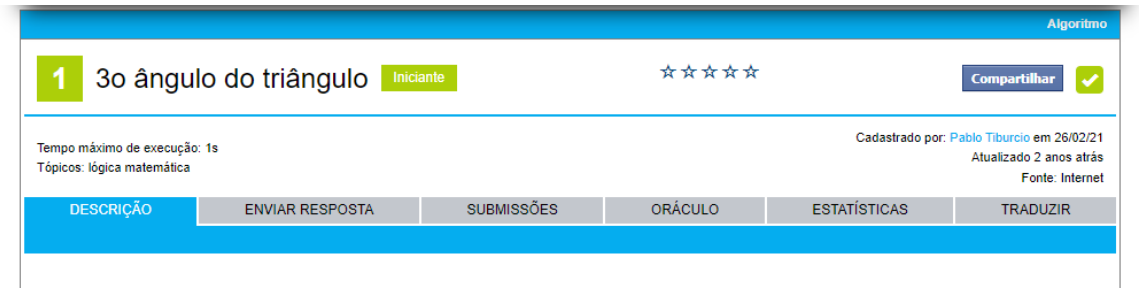
Figura 4 – Tela de seleção de problemas



Fonte: Autor (2023)

Com relação à resolução do problema, tanto o aluno quanto o professor podem solucionar, porém só as submissões dos alunos são listadas na tarefa. A Figura 5 exibe a tela do problema, onde temos várias opções oferecidas pelo The Huxley, vamos focar na aba "Envio de Respostas".

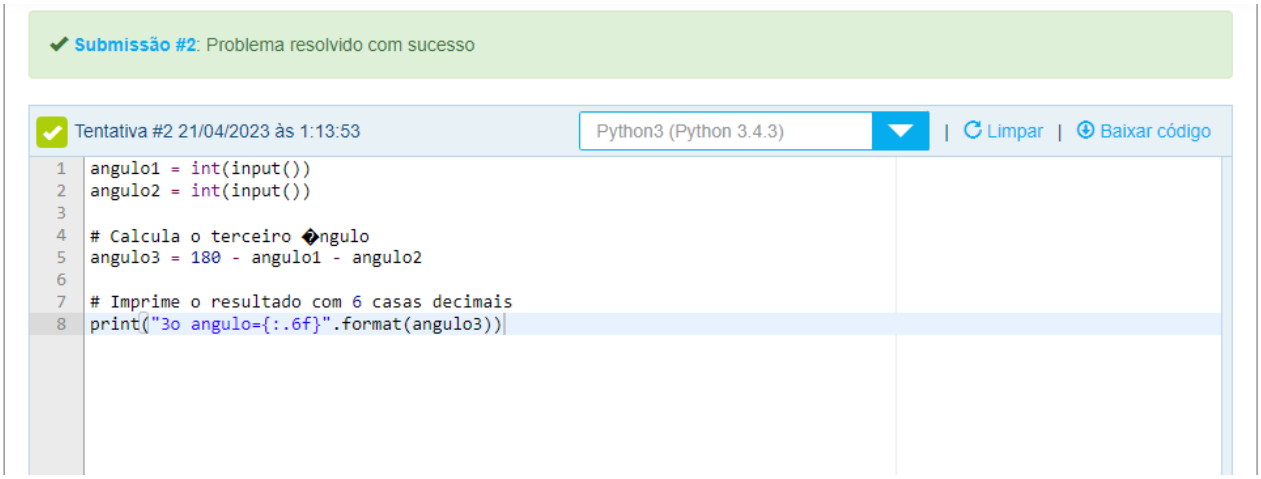
Figura 5 – Tela do problema



Fonte: Autor (2023)

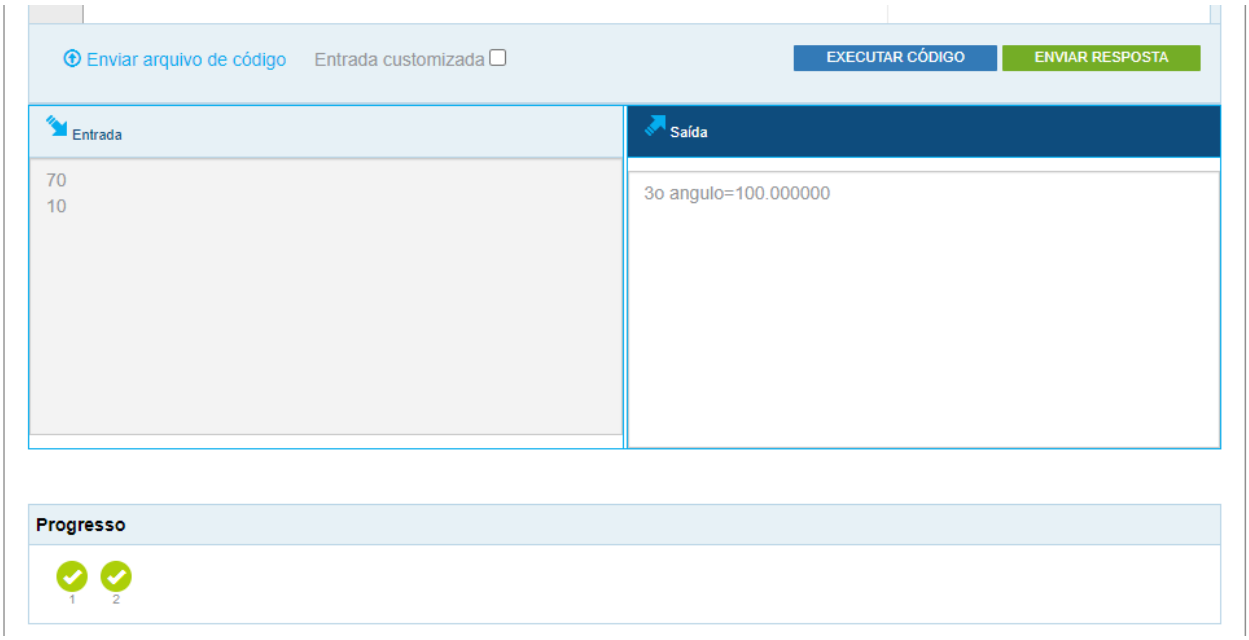
Na aba onde o usuário envia as resposta, temos disponível uma caixa de texto para escrever o código e, logo abaixo, um espaço para testar entradas e saídas. Ao clicar em "Enviar Resposta", o The Huxley testará o código-fonte submetido com os casos de teste cadastrados no site, dando um veredito. Para esse trabalho não existiu o foco em soluções incorretas, já que não faria sentido avaliar a qualidade de uma submissão ainda não concluída. As Figuras 6 e 7 ilustram as etapas descritas anteriormente.

Figura 6 – Envio de uma submissão correta e caixa de texto para envio de resposta



Fonte: Autor (2023)

Figura 7 – Entradas e saídas do problema



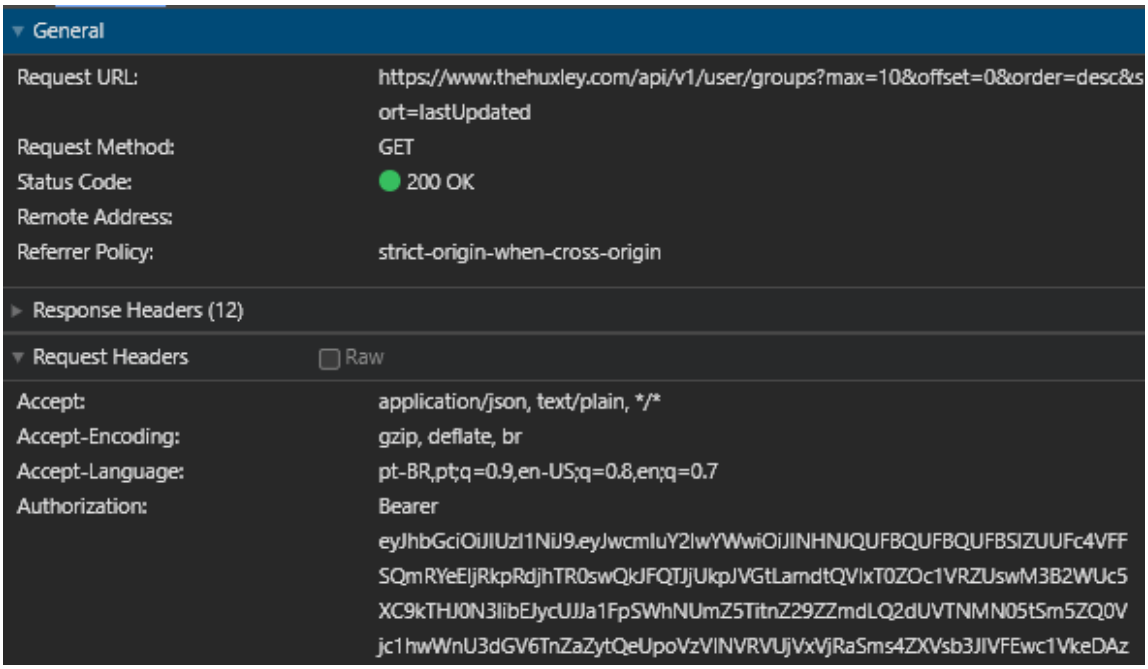
Fonte: Autor (2023)

Como é possível ver nas imagens anteriores, a plataforma não dá informações acerca da qualidade da solução. Tanto o aluno quanto o professor tem apenas a confirmação se o código resolve ou não o problema.

2.1.1 API The Huxley

A *API* do The Huxley já existe e não foi necessário fazer mudanças, porém foi necessário entender o funcionamento da mesma. Inicialmente, foram analisadas as requisições ao acessar o The Huxley e a forma como o *login* é realizado, e então em seguida foi perceptível a importância do *token*, que é recebido ao realizar essa autenticação, esse *token* é utilizado para realizar todas as demais requisições à *API*, tornando-se um elemento vital para o funcionamento do The Avaliator, a ferramenta desenvolvida durante este trabalho. A partir disso, os códigos que realizam as requisições tem estruturas semelhantes.

Figura 8 – Requisição de listagem de turmas no The Huxley



Fonte: Autor (2023)

Para a requisição de listagem de turmas que é feita no The Huxley (Figura 8), utilizamos o *token* de acesso que é enviado no cabeçalho da requisição, e em seguida recebemos um *JSON* com a lista de turmas das quais o professor faz parte.

Em seguida, foi necessário analisar as rotas que precisaríamos requisitar para alcançar o nosso objetivo (obter o código-fonte das submissões), as rotas estão detalhadas no Quadro 1.

Quadro 1: Rotas da *API* do The Huxley

EndPoint	**Descrição**
`https://www.thehuxley.com/api/login`	Rota de Autenticação: rota utilizada para obter o *token* de autenticação do usuário, é enviada uma requisição *POST* com usuário e senha no corpo dela.
`https://www.thehuxley.com/api/v1/user/groups?max=10&offset=0&order=&sort=lastUpdated`	Rota de Turmas: rota utilizada para obter a lista de turmas nas quais o usuário faz parte, é enviada uma requisição *POST* com o *token* no cabeçalho dela.
`https://www.thehuxley.com/api/v1/user/quizzes?filter=OWN&offset=0&order=desc&sort=endDate`	Rota de Tarefas: rota utilizada para obter a lista de tarefas registradas em uma turma, é enviada uma requisição *POST* com o *token* no cabeçalho e o *ID* da turma no *link* da mesma.
`https://www.thehuxley.com/api/v1/quizzes/codigoTarefa/users?max=&offset=0`	Rota de Alunos: rota utilizada para obter a lista de alunos que fazem parte da turma, é enviada uma requisição *POST* com o *token* no cabeçalho e o *ID* da turma no *link* da mesma.
`https://www.thehuxley.com/api/v1/quizzes/codigoTarefa/users/codigoUsuario/problems`	Rota de Problemas: rota utilizada para obter a lista de problemas que foram selecionadas para a tarefa em questão, é enviada uma requisição *POST* com o *token* no cabeçalho e as seguintes informações no *link*: *ID* da tarefa e *ID* do problema.
`https://www.thehuxley.com/api/v1/submissions?problem=&submissionDateLe=dataLimite&user=codigoUsuario`	Rota de Submissões: rota utilizada para obter a lista de submissões que o aluno fez em determinado problema, é enviada uma requisição *POST* com o *token* no cabeçalho e as seguintes informações no *link*: *ID* do problema, data limite da tarefa e *ID* do usuário.
`https://www.thehuxley.com/api/v1/user/problems/codigoProblema/submissions?currentPage=1&max=10&offset=0&order=desc&sort=submissionDate&totalItems=0`	Rota de Submissões do Professor: rota utilizada para obter a lista de subsmissões que o professor fez em determinado problema, é enviada uma requisição *POST* com o *token* no cabeçalho e o *ID* do problema no *link* da mesma.
`https://www.thehuxley.com/api/v1/submissions/codigoSubmissao/sourcecode`	Rota de Código-fonte: rota utilizada para obter o código-fonte da submissão em questão, é enviada uma requisição *POST* com o *token* no cabeçalho e o *ID* da submissão no *link* da mesma.

Fonte: Autor (2023)

Em cada *endpoint* existem propriedades que podem ser passadas pelo *link*, elas estão detalhadas no Quadro 2.

Quadro 2: Propriedades presentes nos *Endpoint's* da *API* do The Huxley

Campos	**Descrição**
`group`	Propriedade que irá conter o *ID* da turma que irá ser pesquisada.
`user`	Propriedade que irá conter o *ID* do usuário que irá ser pesquisado.
`currentPage`	Propriedade que irá conter o número da página que deverá ser pesquisada.
`max`	Propriedade que irá conter o número máximo de itens que será retornado na busca.
`quiz`	Propriedade que irá conter o *ID* da tarefa que irá ser pesquisada.
`offset`	Propriedade que irá conter a posição do primeiro item que será retornado na busca.
`order`	Propriedade que irá conter a ordem (crescente ou decrescente) em que deverão ser organizados os itens retornados na busca.
`sort`	Propriedade que irá conter algum método de organização dos itens (como o SubmissionDate, que organiza por data de submissão)
`evaluations`	Propriedade que irá conter o tipo de avaliação que deverá ser retornada, nesse projeto usamos como "CORRECT"ao acessar o *endpoint* da lista de submissões, pois queremos avaliar apenas as submissões corretas.
`submissionDateLe`	Propriedade que irá conter uma data de submissão final, ou seja, os resultados serão os que foram submetidos em uma data anterior a essa.
`problem`	Propriedade que irá conter o *ID* de um problema específico.

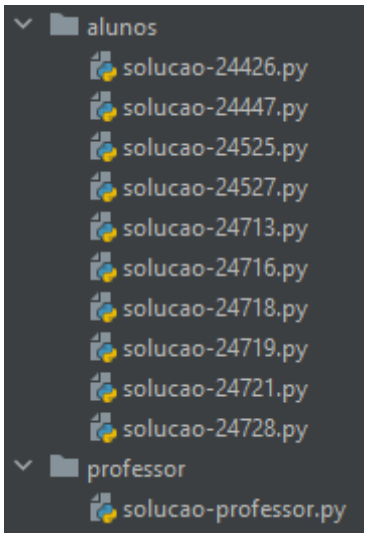
Fonte: Autor (2023)

2.2 Ferramenta de análise de código

A aplicação desenvolvida por Vieira (VIEIRA, 2022) funciona a partir da análise estática de código utilizando a ferramenta Radon (RADON. . . , 2023), que é capaz de calcular métricas de cada código-fonte, de modo que permite criar uma abordagem para comparar códigos-fonte que solucionem o mesmo problema. Para tal, é utilizada uma solução base que seria a do professor, e essa é comparada com uma ou mais soluções dos alunos.

Primeiramente, os arquivos de código-fonte das soluções são organizados em duas pastas: aluno e professor. Na pasta professor se encontra a solução que será usada como base, e na pasta alunos estão as demais soluções que serão avaliadas, é possível ver isso na Figura 9.

Figura 9 – Diretórios da ferramenta avaliadora



Fonte: Autor (2023)

Após organizar essas soluções, é feito o processo de parsing e análise da *AST (Abstract Syntax Tree)* e a ferramenta gera as pontuações utilizando algumas das métricas que o Radon (RADON..., 2023) calcula. Para gerar essas pontuações, primeiro ela acessa um arquivo de configuração nomeado de *configs.json*, nele se encontram algumas configurações de diretório, onde era usado o diretório raiz do arquivo que ia ser avaliado, e eram definidas também as limitações nas pontuações dos alunos.

Segundo a documentação da versão 4.1.0 do Radon (RADON..., 2023), que é a utilizada no projeto, são extraídas várias métricas, porém será dado foco nas que o avaliador de código se baseia para o cálculo da pontuação:

- Complexidade Ciclomática: é calculada a partir da enumeração do número de caminhos que um programa de computador pode seguir do seu começo até o final.
- Linhas de Código (LOC): trata-se da quantidade de linhas que o programa possui, quanto menor esse número, melhor é considerada a solução.
- Linhas Lógicas de Código-Fonte (SLOC): Trata-se do valor de LOC, descontando-se a quantidade de linhas em branco e de comentários.
- Linhas Lógicas de Código (LLOC): Baseia-se em SLOC, mas quando há mais de uma instrução em uma linha, computa separadamente cada instrução. Por exemplo, se uma instrução vier após o caractere : e na mesma linha de um *if* ou *else* em Python, o LLOC vai contar a instrução separadamente. Já o SLOC e LOC contam como uma única linha. Por outro lado, um *print* que tenha 3 parâmetros, sendo disposto em 3 linhas, será contado com 1 no LLOC e 3 no LOC e SLOC.

Para entender melhor o que cada métrica significa, segue um exemplo de código em Python e as suas métricas correspondentes, baseadas no sistema de contagem elaborado por Park (PARK, 1992).

```
1  '''
2  Exemplo de leitura de x e y
3  -> x e y devem ser não negativos
4  '''
5  while True:
6     x = int(input())
7     y = int(input())
8
9     # x e y positivos (encerra)
10    if (x >= 0 and
11        y >= 0): break
12    # x e/ou y negativos (continua)
13    else: print('algum é negativo')
14 print('Soma de x e y = ', x+y)
```

Nesse código a contagem de LOC é 14, sendo seis linhas de comentário, outra linha em branco e sete linhas de código. Em relação ao LLOC, há 9 linhas ao todo, que se referem as instruções *while*, atribuição(=), *if*, *and*, *break*, *else* e *print*. Na sequência, em relação ao SLOC, a contagem é de 7 linhas, que são as linhas de código desconsiderando os comentários e as linhas em branco. Essas métricas são normalmente utilizadas para medir o esforço que será necessário para desenvolver o *sofware*.

No arquivo de configuração citado anteriormente, são definidos limites, relativos às métricas do código-fonte do professor, para os valores citados anteriormente, e caso a pontuação do aluno ultrapasse algum desses valores, é alertado para o usuário na planilha de alerta de resultados, que é um dos 3 arquivos distintos criados para organizar as métricas.

Utilizando esses valores limite, a ferramenta calcula as métricas do código utilizando a biblioteca Radon (RADON. . . , 2023), para então gerar os arquivos de resultado e concluir a avaliação. A última métrica calculada é a de *Final Score*, a qual a ferramenta utiliza tanto as métricas do Radon (RADON. . . , 2023) quando os valores limite para calcular, é considerado que o código do professor possui valor igual a 100 e então as métricas do código-fonte do professor são comparadas com as do código-fonte do aluno, a partir do momento que a pontuação do aluno se inicie com 100, são realizados incrementos e decrementos correlacionados ao valor das métricas do aluno em comparação com as do professor, para então obter o resultado final, o método de pontuação está melhor detalhado em (VIEIRA, 2022).

O decremento e o incremento da pontuação são feitos da seguinte forma.

- Primeiro cada métrica do aluno é comparada com a do professor, e então, caso o valor seja diferente, o cálculo é feito. Se o aluno obtiver uma pontuação melhor, será um cálculo de incremento, caso contrário, será um cálculo de decremento.
- Para a complexidade ciclomática, caso seja uma situação de incremento, a pontuação inicial será incrementada em um cálculo multiplicando o fator de incremento (valor fixo definido no arquivo de configurações) pela diferença, já para o decremento o cálculo é feito com o fator de decremento.
- Considerando as demais métricas, os fatores de incremento e decremento são definidos a partir de um cálculo percentual feito utilizando a taxa de incremento e de decremento, valores definidos no arquivo de configurações.
- Esse arquivo de configurações citado anteriormente contém informações como: diretórios dos arquivos e valores fixos para os cálculos, é possível ver mais informações em (VIEIRA, 2022).

Segue o código que lida com essa parte.

```
def __calculate_score(self, solution, diff_in_points, is_raw_metrics=False):
    increase_factor = self.configs['increase_factor']
    decrease_factor = self.configs['decrease_factor']

    if is_raw_metrics:
        increase_factor *= (self.configs['rate_increase_to_raw_metrics'] / 100)
        decrease_factor *= (self.configs['rate_decrease_to_raw_metrics'] / 100)

    if diff_in_points > 0:
        solution.score -= diff_in_points * decrease_factor
    else:
        solution.score += -diff_in_points * increase_factor
```

Código 2.1 – Código fonte em Python, retirado de (VIEIRA, 2022)

No primeiro arquivo (Figura 10), se encontram os dados resultantes do cálculo feito, no Quadro 3 estão detalhadas as colunas dessa planilha.

	A	B	C	D	E	F	G	H	I	J	K	L
1	PROBLEM	SOLUTION	IS_TEACHER	CYCLOMATIC_COMPLEXITY	EXCEEDED LIMIT CC	LINES OF CODE	EXCEEDED LIMIT LOC	LOGICAL LINES OF CODE	EXCEEDED LIMIT LLOC	SOURCE LINES OF CODE	LIMIT SLOC	FINAL SCORE
2	76-Correcao-de-Provas	solucao-24426	NO	10	NO	36	NO	32	NO	32	NO	93.7
3	76-Correcao-de-Provas	solucao-24447	NO	9	NO	30	NO	28	NO	28	NO	100.0
4	76-Correcao-de-Provas	solucao-24525	NO	10	NO	32	NO	32	NO	32	NO	94.5
5	76-Correcao-de-Provas	solucao-24527	NO	10	NO	38	NO	38	NO	38	NO	87.9
6	76-Correcao-de-Provas	solucao-24713	NO	12	NO	46	NO	37	NO	37	NO	79.45
7	76-Correcao-de-Provas	solucao-24716	NO	10	NO	34	NO	30	NO	30	NO	95.4
8	76-Correcao-de-Provas	solucao-24718	NO	8	NO	29	NO	26	NO	26	NO	104.0
9	76-Correcao-de-Provas	solucao-24719	NO	10	NO	34	NO	30	NO	30	NO	95.4
10	76-Correcao-de-Provas	solucao-24721	NO	11	NO	53	NO	41	NO	41	NO	84.7
11	76-Correcao-de-Provas	solucao-24728	NO	7	NO	33	NO	27	NO	27	NO	104.8
12	76-Correcao-de-Provas	solucao-professor	YES	8	-	39	-	31	-	31	-	-
13												

Figura 10 – Planilha com as métricas das submissões

Fonte: Autor (2023)

Quadro 3: Modelo de resultado

PROBLEM	Nome do problema
SOLUTION	Nome do arquivo de solução
IS_TEACHER	Se a submissão é do professor ou não
CYCLOMATIC_COMPLEXITY	Valor (inteiro) da complexidade ciclomática
EXCEEDED LIMIT CC	Se excedeu o limite estabelecido de complexidade ciclomática
LINES OF CODE	Valor (inteiro) da quantidade de linhas de código
EXCEEDED LIMIT LOC	Se excedeu o limite da quantidade de linhas de código
LOGICAL LINES OF CODE	Valor (inteiro) da quantidade de linhas lógicas de código
EXCEEDED LIMIT LLOC	Se excedeu o limite da quantidade de linhas lógicas de código
SOURCE LINES OF CODE	Valor (inteiro) da quantidade de linhas de código-fonte
LIMIT SLOC	Se excedeu o limite da quantidade de linhas de código-fonte
FINAL SCORE	Pontuação final obtida após o processamento das métricas

Fonte: (VIEIRA, 2022)

No segundo arquivo (Figura 11) se encontra um texto que detalha o resultado das pontuações e as compara com a solução base.

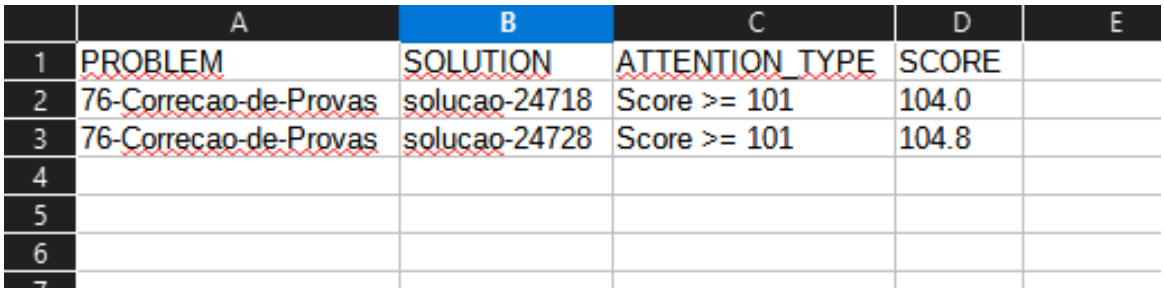
Figura 11 – Arquivo de texto com as métricas das submissões

```
--------------- Resultados para o problema: `76-Correcao-de-Provas`. ---------------
        Resultado para a solução base:
                Complexidade ciclomática: 8 pontos.
                Linhas de código: 39
                Linhas lógicas de código: 31
                Linhas de código fonte: 31
        Resultado para as submissões dos alunos:
                Submissão: solucao-24426
                        Complexidade ciclomática: 10 pontos contra 8 da solução base.
                        Linhas de código: 36; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 32; Diferença com a submissão base: 1;
                        Linhas de código fonte: 32; Diferença com a submissão base: 1;
                Submissão: solucao-24447
                        Complexidade ciclomática: 9 pontos contra 8 da solução base.
                        Linhas de código: 30; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 28; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas de código fonte: 28; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                Submissão: solucao-24525
                        Complexidade ciclomática: 10 pontos contra 8 da solução base.
                        Linhas de código: 32; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 32; Diferença com a submissão base: 1;
                        Linhas de código fonte: 32; Diferença com a submissão base: 1;
                Submissão: solucao-24527
                        Complexidade ciclomática: 10 pontos contra 8 da solução base.
                        Linhas de código: 38; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 38; Diferença com a submissão base: 7;
                        Linhas de código fonte: 38; Diferença com a submissão base: 7;
                Submissão: solucao-24713
                        Complexidade ciclomática: 12 pontos contra 8 da solução base.
                        Linhas de código: 46; Diferença com a submissão base: 7;
                        Linhas lógicas de código: 37; Diferença com a submissão base: 6;
                        Linhas de código fonte: 37; Diferença com a submissão base: 6;
                Submissão: solucao-24716
                        Complexidade ciclomática: 10 pontos contra 8 da solução base.
                        Linhas de código: 34; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 30; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas de código fonte: 30; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                Submissão: solucao-24718
                        Complexidade ciclomática: 8 pontos, igual a solução base
                        Linhas de código: 29; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 26; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas de código fonte: 26; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                Submissão: solucao-24719
                        Complexidade ciclomática: 10 pontos contra 8 da solução base.
                        Linhas de código: 34; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 30; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas de código fonte: 30; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                Submissão: solucao-24721
                        Complexidade ciclomática: 11 pontos contra 8 da solução base.
                        Linhas de código: 53; Diferença com a submissão base: 14;
                        Linhas lógicas de código: 41; Quantidade igual de linhas com a submissão base
                        Linhas de código fonte: 41; Quantidade igual de linhas com a submissão base
                Submissão: solucao-24728
                        Complexidade ciclomática: 7 pontos contra 8 da solução base
                        Linhas de código: 33; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas lógicas de código: 27; Menos linhas que a submissão base e não ultrapassou o limite de alerta
                        Linhas de código fonte: 27; Menos linhas que a submissão base e não ultrapassou o limite de alerta
```

Fonte: Autor (2023)

No terceiro arquivo se encontra mais uma planilha (Figura 12), porém agora apenas como alerta de soluções que precisam de mais atenção do professor. Vieira (VIEIRA, 2022) define dois termos importantes, o *score to great solution*, que indica a pontuação necessária para que uma solução seja considerada ótima, e o *score to not so good solution*, que indica a pontuação para que esse código-fonte seja considerado abaixo do esperado, o Quadro 4 exibe as colunas da planilha.

Figura 12 – Planilha com as pontuações das submissões que necessitam de atenção do professor



	A	B	C	D	E
1	PROBLEM	SOLUTION	ATTENTION_TYPE	SCORE	
2	76-Correcao-de-Provas	solucao-24718	Score >= 101	104.0	
3	76-Correcao-de-Provas	solucao-24728	Score >= 101	104.8	
4					
5					
6					

Fonte: Autor (2023)

Quadro 4: Modelo de resultado de alerta

PROBLEM	Nome do problema
SOLUTION	Nome do arquivo de solução
ATTENTION_TYPE	Motivo da submissão estar neste arquivo (pontuação maior ou menor que o limiar)
SCORE	Pontuação

Fonte: (VIEIRA, 2022)

3
Ferramentas

Este capítulo apresenta as ferramentas que foram utilizadas durante o desenvolvimento do projeto. Cada seção descreve uma ferramenta e sua aplicação no projeto.

3.1 ASP.NET *Core*

Para o back-end da solução, foi utilizado o *framework* de código aberto ASP.NET Core[1], que propõe um desenvolvimento *web* de maneira contemporânea e eficaz. É o sucessor do ASP.NET, sendo um *framework* totalmente modular e que pode ser codificado em qualquer *IDE*.

Os principais motivos para utilizar esse *framework* são a sua característica de ser multiplataforma, além de fazer parte do ambiente .NET que, dentre os seus pontos fortes, tem a possibilidade de reutilização de código-fonte e um conjunto abrangente de ferramentas para desenvolvimento de aplicativos em diversas plataformas.

3.2 React

Para desenvolver o *front-end* da aplicação foi utilizado o *framework* chamado React[2], o qual é comumente usado para criar interfaces de usuário para aplicativos *web*. Uma das principais vantagens do React é a sua flexibilidade e facilidade de uso, além do que se trata de uma biblioteca de código aberto, o que resulta em uma comunidade empenhada em melhorar a ferramenta.

Além do mais, no React é possível reutilizar componentes, ou seja, trechos de código que podem ser aproveitados em diferentes partes do programa. Segundo Camargos (CAMARGOS et al., 2019), o React possui uma curva de aprendizado menor que outros *frameworks* de JavaScript,

[1] https://dotnet.microsoft.com/en-us/apps/aspnet
[2] https://react.dev/

e o seu poder de reaproveitamento de código contribui para agilizar o desenvolvimento de projetos.

3.3 Heroku

O Heroku[3] é uma plataforma de hospedagem na nuvem que permitiu que a aplicação fosse disponibilizada para acesso geral. Os códigos são armazenados em contêineres chamados *dynos*, e existe um limite de recursos disponíveis para o usuário com base no plano em que ele pertence.

O principal motivo para usar essa plataforma é a sua facilidade de *deploy*, sendo possível realizá-lo a partir de um repositório no Github [4].

3.4 Visual Studio Code

O Visual Studio Code[5] é uma *IDE* multiplataforma que, tendo sido utilizado para codificar a aplicação do The Avaliator, foi desenvolvido pela Microsoft e inclui diversas extensões que ajudam a desenvolver código-fonte, além de suporte para depuração e Git incorporado.

Usando esse *software* foi possível desenvolver em React e ASP.NET Core sem a necessidade de trocar de ambiente. A existência de extensões específicas agilizou a escrita do código-fonte e facilitou o desenvolvimento. Como é uma ferramenta que possui integração com Git, também foi possível controlar o histórico de versões da aplicação.

3.5 PyCharm

O PyCharm[6] é um ambiente de desenvolvimento de programas em Python, que foi utilizado para desenvolver a *API* da ferramenta avaliadora de código, devido à sua facilidade e ao suporte da *IDE* para a linguagem. Também foi a ferramenta utilizada por Vieira (VIEIRA, 2022) para o desenvolvimento da ferramenta avaliadora de código.

3.6 PostgreSQL

O PostgreSQL[7] foi o Sistema de Gerenciamento de Banco de Dados Relacional utilizado para manter todos os dados necessários para a aplicação, devido à sua facilidade de uso, além de ser gratuito e de código aberto.

[3] https://www.heroku.com/
[4] https://github.com/
[5] https://code.visualstudio.com/
[6] https://www.jetbrains.com/pt-br/pycharm/
[7] https://www.postgresql.org/

Esse SGBD possui suporte para múltiplas plataformas e uma comunidade ativa, o que traz benefícios principalmente na etapa de resolução de problemas.

3.7 GitHub

O GitHub[8] foi a plataforma utilizada para hospedar os arquivos do projeto, uma das principais vantagens apresentadas é o controle de versão de código, já que faz uso do Git. O conhecimento prévio da plataforma e a facilidade de uso foram os principais motivos para a escolha.

A plataforma possui diversas funcionalidades úteis, como o sistema de lista de tarefas e *Projects Boards*, por meio do quais é possível gerenciar o projeto com mais praticidade. Além disso, com pacote de estudante, é possível obter diversos benefícios utilizando o *e-mail* do DCOMP, inclusive o plano profissional.

[8] https://github.com/

4

Especificação de Requisitos

Neste capítulo serão abordados os requisitos da aplicação. Na Seção 4.1 são descritos os requisitos funcionais e não funcionais. da aplicação. Em seguida, na Seção 4.2, são detalhados os casos de uso. A Seção 4.3 aborda o diagrama de arquitetura da aplicação. Finalmente, na Seção 4.4 discute-se alguns detalhes sobre a API REST que será desenvolvida.

4.1 Requisitos

4.1.1 Requisitos Funcionais

Os requisitos funcionais se referem às funções que a aplicação deve ter, as quais estão detalhadas no Quadro 5.

Quadro 5: Requisitos Funcionais

Id	**Título**	**Descrição**
RF001	Autenticação de usuários	Autenticação de usuário através do The Huxley.
RF002	Manter informações do The Huxley	Manter no sistema dados de turmas, problemas e submissões realizadas.
RF003	Avaliar submissões	Mostrar no sistema dados sobre as pontuações obtidas através da comparação entre o código do professor com o do aluno feita pela *API* de avaliação de código.
RF004	Filtrar submissões	Possibilidade de filtrar as submissões de acordo com a sua pontuação, situadas em três categorias: pontuação baixa, pontuação ok e pontuação alta.

Fonte: Autor (2023)

4.1.2 Requisitos Não Funcionais

Os requisitos não funcionais são aqueles que tratam da forma que o *software* vai realizar as suas funcionalidades, conforme detalhado no Quadro 6.

Quadro 6: Requisitos Não Funcionais

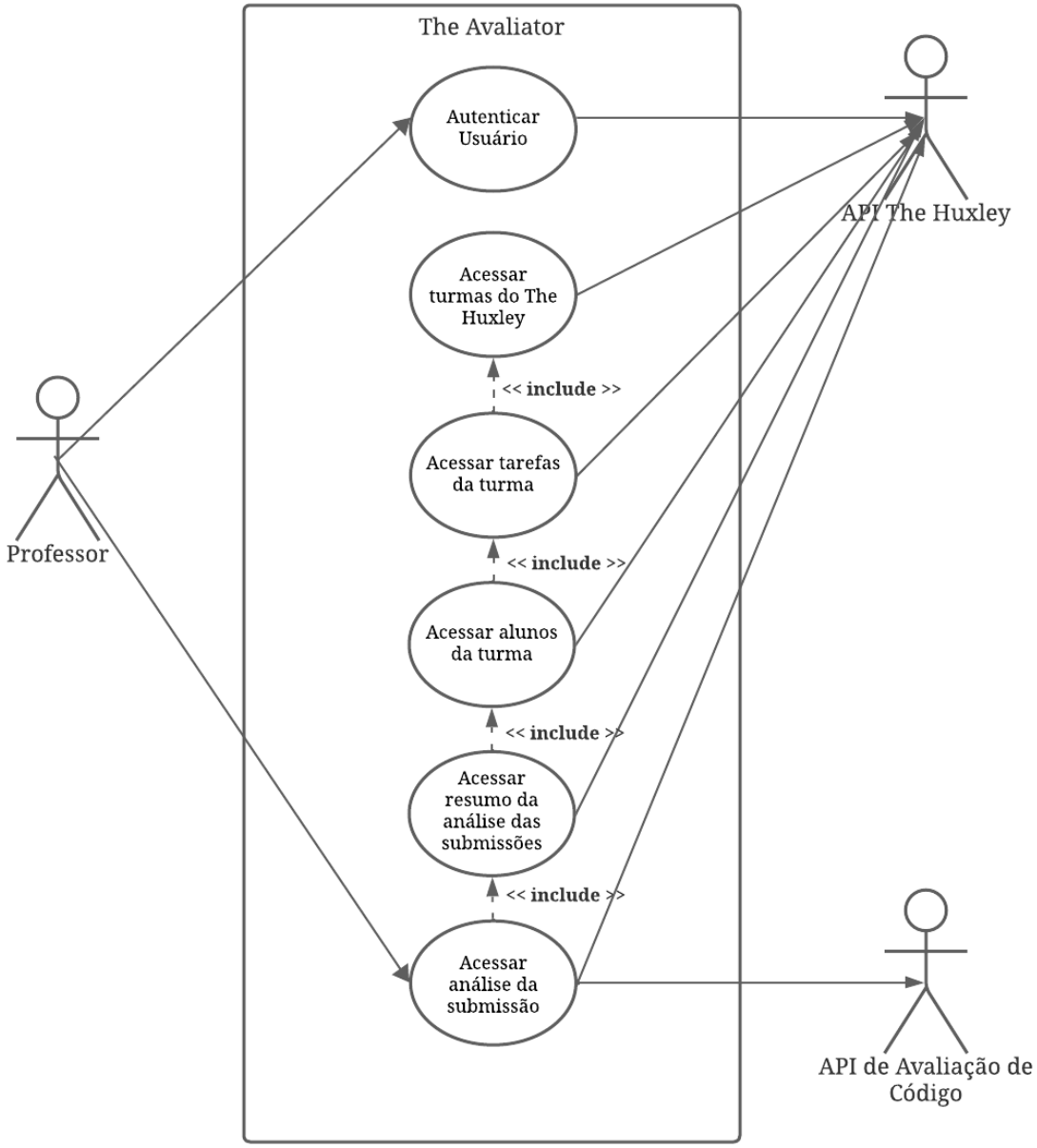
Id	**Título**	**Descrição**
RNF001	Ambiente *Web*	A aplicação deverá funcionar no ambiente *web*.
RNF002	Disponibilidade do The Avaliator	O ambiente deverá estar disponível 24 horas por dia contanto que o The Huxley esteja online.
RNF003	Responsividade	O ambiente deve ser responsivo para todas as telas que podem o acessar.
RNF004	Praticidade de uso	O ambiente deve ser prático e fácil de usar de forma a ajudar os usuários.

Fonte: Autor (2023)

4.2 Casos de Uso

O diagrama de casos de uso da aplicação (Figura 13) dá uma visão da relação entre as funcionalidades do sistema e dos atores que interagem com o mesmo. Nos quadros 7, 8, 9, 10, 11 e 12 encontram-se as descrições de cada caso de uso, sendo o primeiro relacionado à autenticação, e os seguintes relacionados aos problemas cadastrados no The Huxley. Enquanto isso, nas figuras 14, 15, 16, 17, 18 e 19, é possível visualizar os diagramas de sequência relacionados a cada caso de uso.

Figura 13 – Diagrama de Casos de Uso



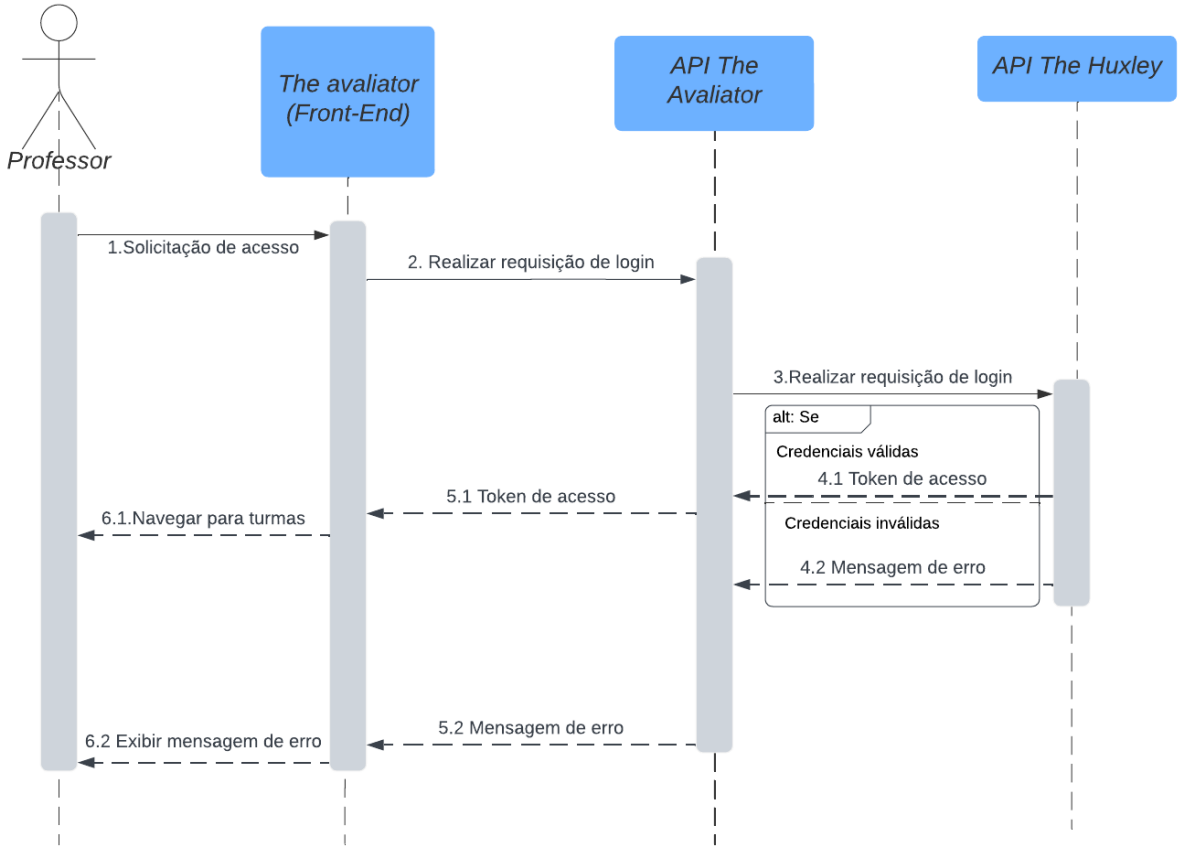
Fonte: Autor (2023)

Quadro 7: Descrição do Caso de Uso Autenticar Usuário

CSU01 – Autenticar Usuário	
Objetivo	Permitir ao usuário que faça o login no sistema.
Descrição	O usuário deve conseguir fazer o login através da autenticação pelo The Huxley.

Fonte: Autor (2023)

Figura 14 – Diagrama de sequência - Autenticar Usuário



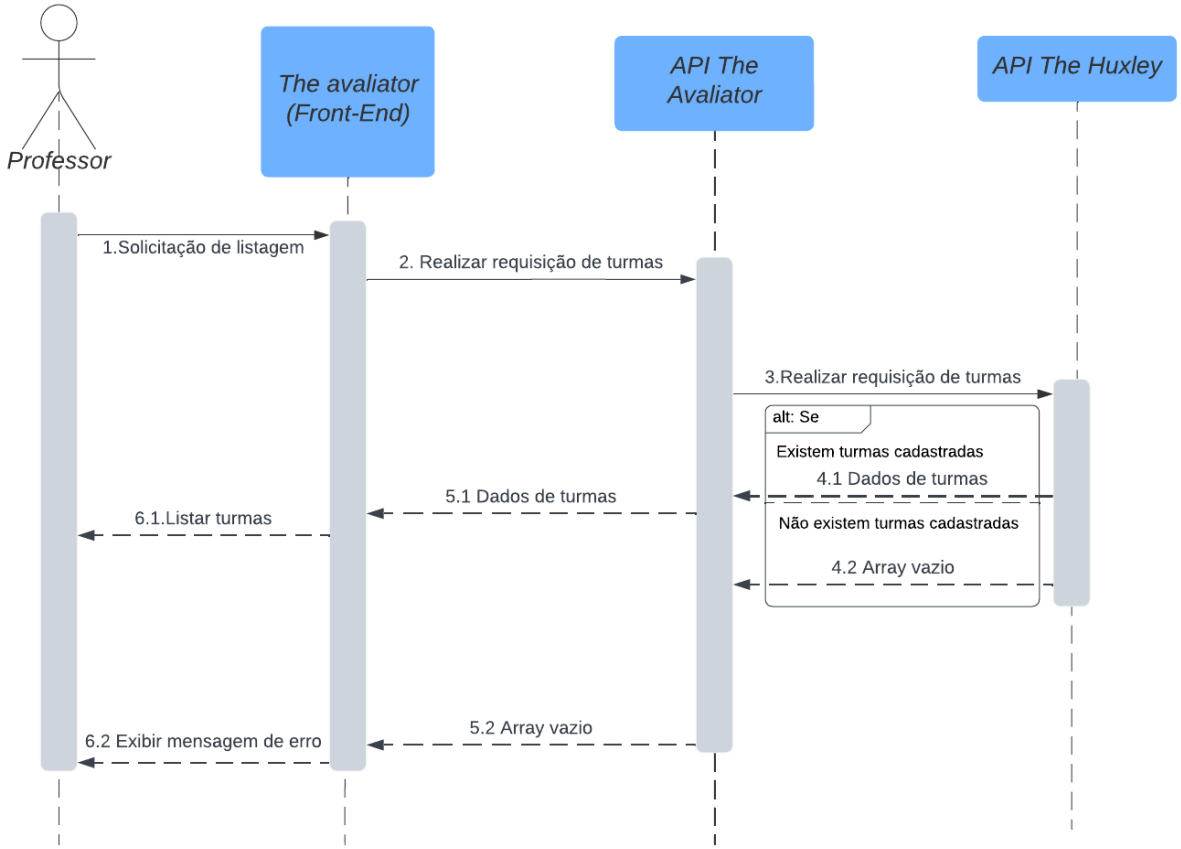
Fonte: Autor (2023)

Quadro 8: Descrição do Caso de Uso Acessar turmas do The Huxley

CSU02 – Acessar turmas do The Huxley	
Objetivo	Permitir ao usuário acessar as turmas.
Descrição	O usuário deve conseguir acessar as turmas com as quais ele tem vínculo no The Huxley.

Fonte: Autor (2023)

Figura 15 – Diagrama de sequência - Acessar turmas do The Huxley



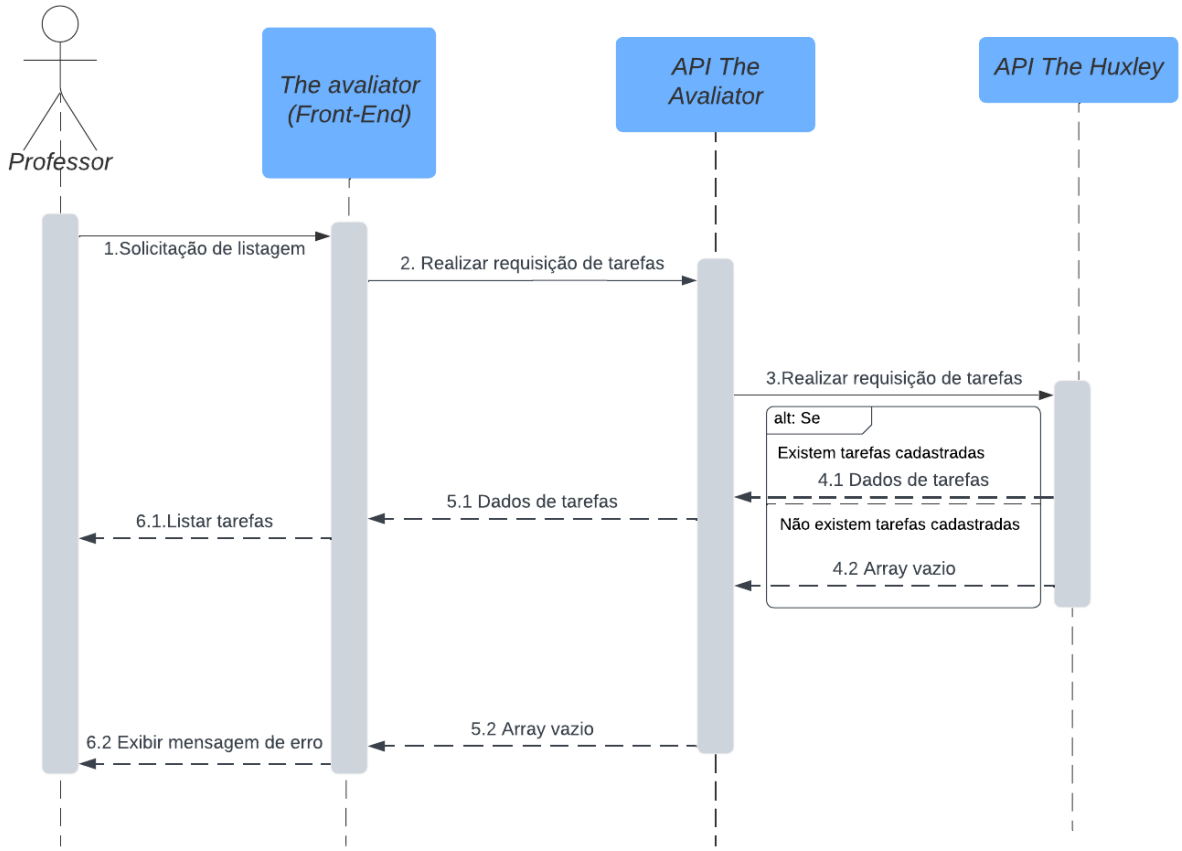
Fonte: Autor (2023)

Quadro 9: Descrição do Caso de Uso Acessar tarefas da turma

CSU03 – Acessar tarefas da turma	
Objetivo	Permitir ao usuário acessar as tarefas.
Descrição	O usuário deve conseguir acessar as tarefas das turmas com as quais ele tem vínculo no The Huxley.

Fonte: Autor (2023)

Figura 16 – Diagrama de sequência - Acessar tarefas da turma



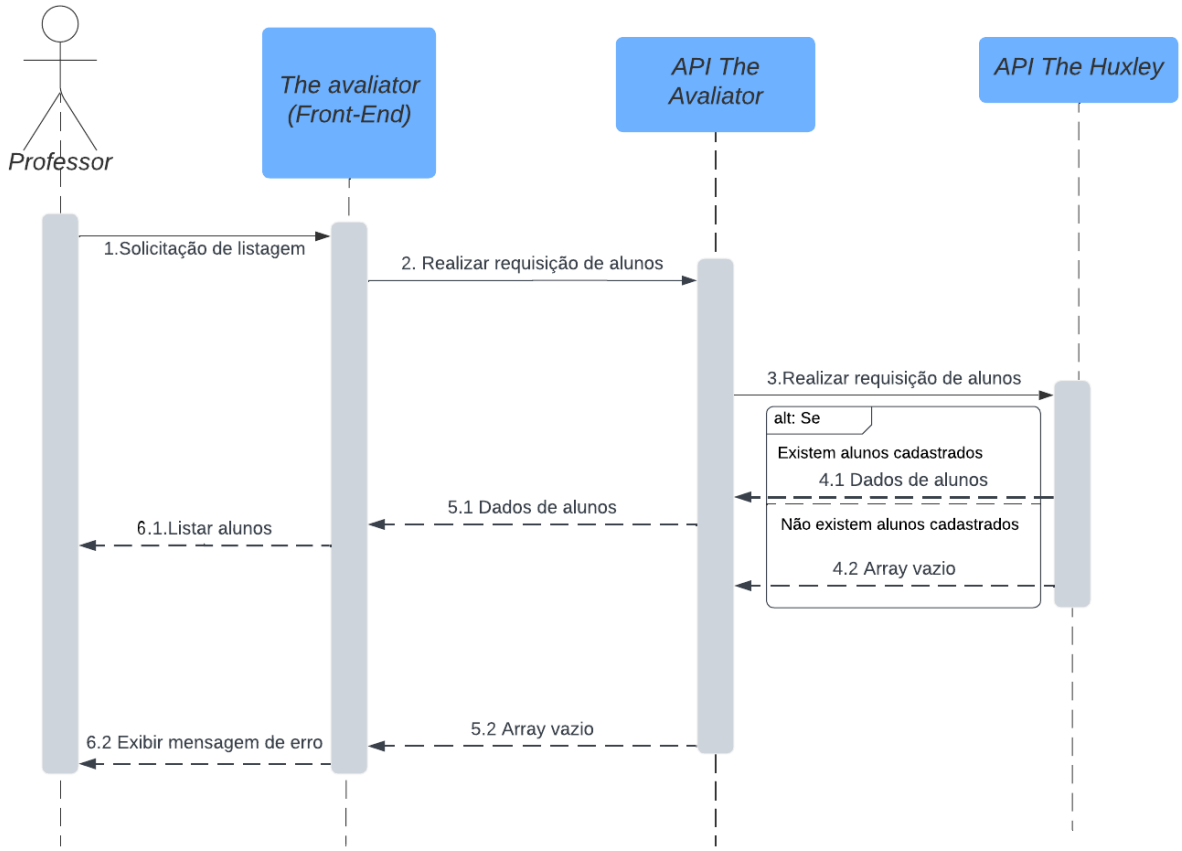
Fonte: Autor (2023)

Quadro 10: Descrição do Caso de Uso Acessar alunos da turma

CSU04 - Acessar alunos da turma	
Objetivo	Permitir ao usuário acessar a lista dos alunos.
Descrição	O usuário deve conseguir acessar a lista de alunos que fazem parte da sua turma no The Huxley.

Fonte: Autor (2023)

Figura 17 – Diagrama de sequência - Acessar alunos da turma



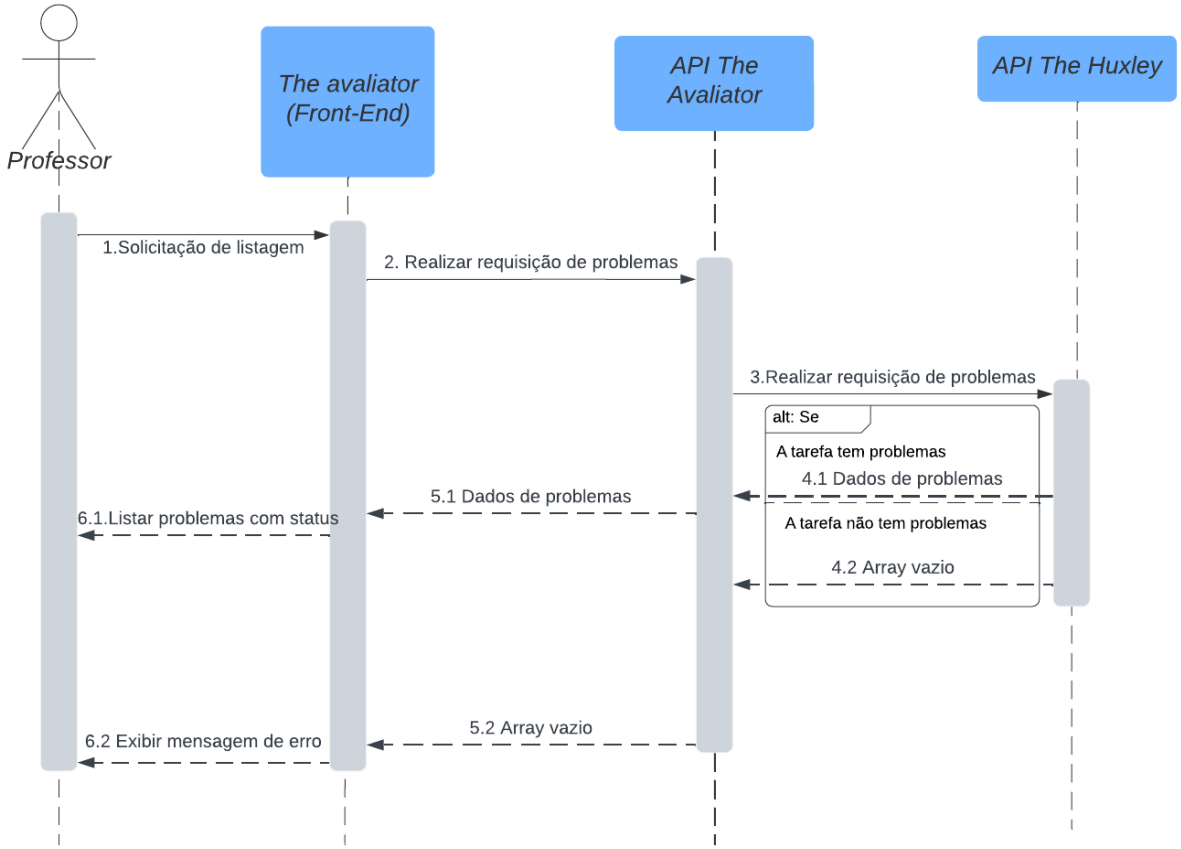
Fonte: Autor (2023)

Quadro 11: Descrição do Caso de Uso Acessar resumo da análise das submissões

CSU05 – Acessar resumo da análise das submissões	
Objetivo	Permitir ao usuário acessar um resumo da análise das submissões.
Descrição	O usuário deve conseguir acessar o resumo gerado da pontuação das submissões que foram realizadas no The Huxley.

Fonte: Autor (2023)

Figura 18 – Diagrama de sequência - Acessar resumo da análise das submissões



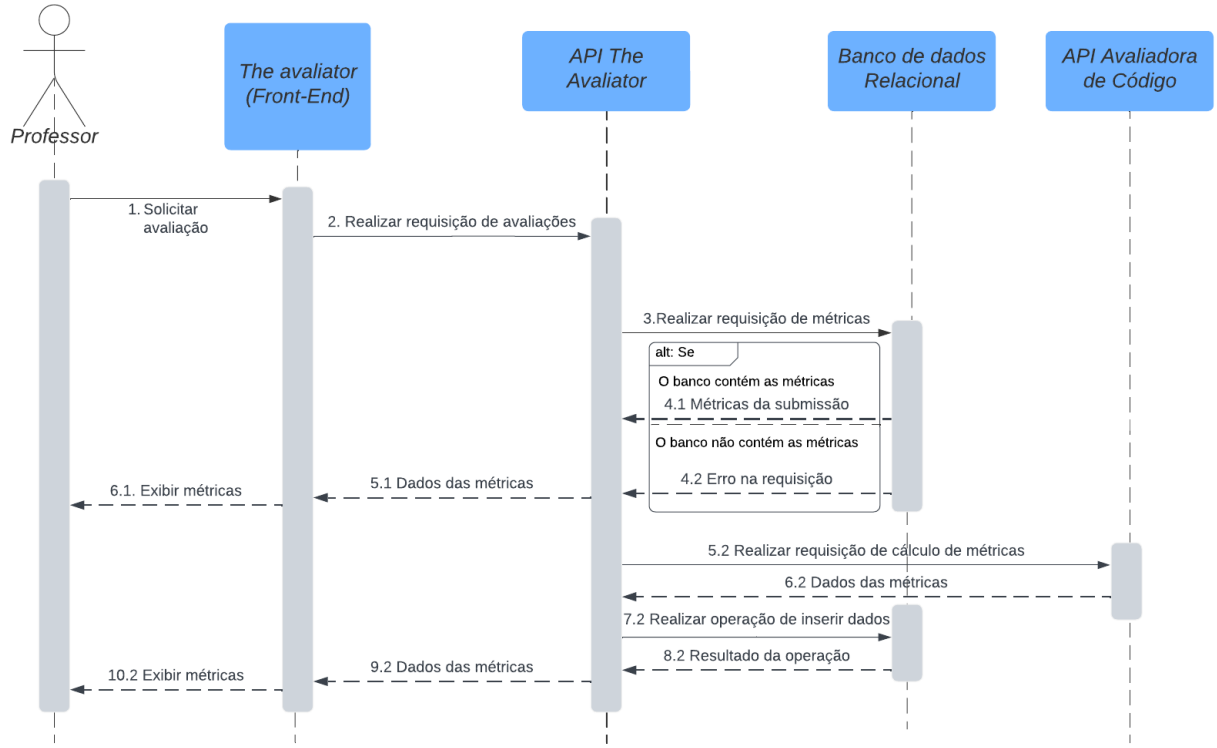
Fonte: Autor (2023)

Quadro 12: Descrição do Caso de Uso Acessar análise da submissão

CSU06 – Acessar análise da submissão	
Objetivo	Permitir ao usuário visualizar as métricas da submissão.
Descrição	O usuário deve conseguir acessar as métricas e o código-fonte da submissão que foi realizada no The Huxley.

Fonte: Autor (2023)

Figura 19 – Diagrama de sequência - Acessar análise da submissão

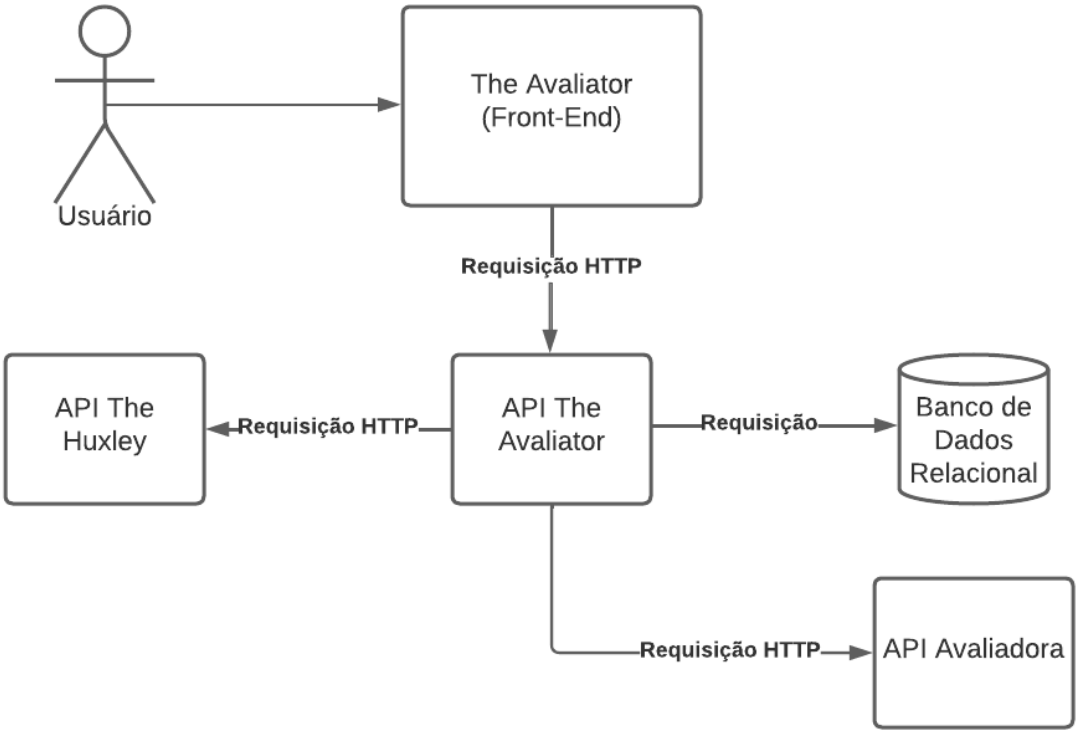


Fonte: Autor (2023)

4.3 Arquitetura

Na Figura 20, é possível visualizar o diagrama de arquitetura da aplicação, que ilustra os componentes do sistema e como eles interagem entre si durante o seu funcionamento.

Figura 20 – Diagrama de Arquitetura



Fonte: Autor (2023)

De início, o usuário irá se comunicar com o Front-End da aplicação, essa interface web se comunica diretamente com a API The Avaliator, que funciona como um mediador entre a comunicação do Front-End com três partes diferentes da aplicação: a API The Huxley, responsável por devolver os dados do site do The Huxley, a API Avaliadora, que foi desenvolvida e, a partir da ferramenta avaliadora, e retornará as métricas obtidas após a comparação do código, e o Banco de Dados Relacional, que armazenará os dados de métricas para comparações que já foram feitas anteriormente, evitando o recálculo e tornando a aplicação mais rápida e eficiente.

4.4 Desenvolvimento da API Avaliadora

Para o desenvolvimento da aplicação *web*, foi necessário criar uma fachada (GAMMA et al., 2000, 179) para a ferramenta de pontuação de código (VIEIRA, 2022) em uma *API REST* que recebe os códigos-fonte e devolva as métricas correspondentes, denominada de *API* Avaliadora. Como a ferramenta já estava desenvolvida em Python, foi utilizada essa mesma linguagem para a *API* Avaliadora de modo a facilitar o desenvolvimento e a integração. Após desenvolver a *API*, a plataforma criada consegue realizar requisições e obter de forma simples e rápida os resultados. Com relação à obtenção dos resultados, foi necessário também converter os mesmos para o formato *JSON*, o que se resolveu com o uso de bibliotecas de Python.

5

Desenvolvimento do Projeto

Neste capítulo estão especificadas as etapas de desenvolvimento do projeto. Na Seção 5.1 está descrita a etapa de transformar a ferramenta avaliadora (VIEIRA, 2022) em uma *API REST*. Em seguida, na Seção 5.2, está detalhado o *Back-End* da aplicação. A Seção 5.3 contém os detalhes do desenvolvimento do *Front-End* da aplicação.

5.1 Aprimoramentos na Ferramenta de Avaliação

Para que a ferramenta avaliadora alcançasse o seu objetivo de uso, era necessário realizar algumas atualizações importantes no seu código. Não foram feitas mudanças na parte das métricas de avaliação da *API*, apenas a mesma foi adaptada para se adequar ao projeto do The Avaliator. Cada alteração e a sua justificativa estão explicadas nas próximas sub-seções.

5.1.1 Resultados

Os resultados das avaliações eram anteriormente preenchidos em 3 arquivos diferentes, sendo duas planilhas e um arquivo de texto. No nosso contexto, foi necessário adicionar uma funcionalidade que transforme essa planilha de resultados em um arquivo *JSON*, para que ele possa ser enviado como resposta a uma requisição feita na *API*. No código abaixo consta a função que realiza essa transformação. A função itera sobre as linhas do arquivo de planilha, e então, para cada linha ela cria um dicionário *JSON*, onde as chaves são o nome da coluna e os valores são os da linha atual.

```
1 def build_json(self):
2         data = []
3         file_path_csv = os.path.join(self.path, 'result.csv')
4 
5         with open(file_path_csv, 'r') as csvfile:
```

```
            csvreader = csv.DictReader(csvfile, delimiter=';')  # Use o
                ponto-e-v rgula como delimitador
            for row in csvreader:
                # Montar o dicion rio para o JSON
                json_entry = {
                    "PROBLEM": row["PROBLEM"],
                    "SOLUTION": row["SOLUTION"],
                    "IS_TEACHER": row["IS_TEACHER"],
                    "CYCLOMATIC_COMPLEXITY": row["CYCLOMATIC_COMPLEXITY"],
                    "EXCEEDED_LIMIT_CC": row["EXCEEDED LIMIT CC"],
                    "LINES_OF_CODE": row["LINES OF CODE"],
                    "EXCEEDED_LIMIT_LOC": row["EXCEEDED LIMIT LOC"],
                    "LOGICAL_LINES_OF_CODE": row["LOGICAL LINES OF CODE"],
                    "EXCEEDED_LIMIT_LLOC": row["EXCEEDED LIMIT LLOC"],
                    "SOURCE_LINES_OF_CODE": row["SOURCE LINES OF CODE"],
                    "LIMIT_SLOC": row["LIMIT SLOC"],
                    "FINAL_SCORE": row["FINAL SCORE"]
                }
                data.append(json_entry)

        file_path = os.path.join(self.path, self.name)

        with open(file_path, 'w', encoding='UTF-8') as jsonfile:
            json.dump(data, jsonfile, indent=4)

        print(f"CSV file converted to JSON file successfully.")
```

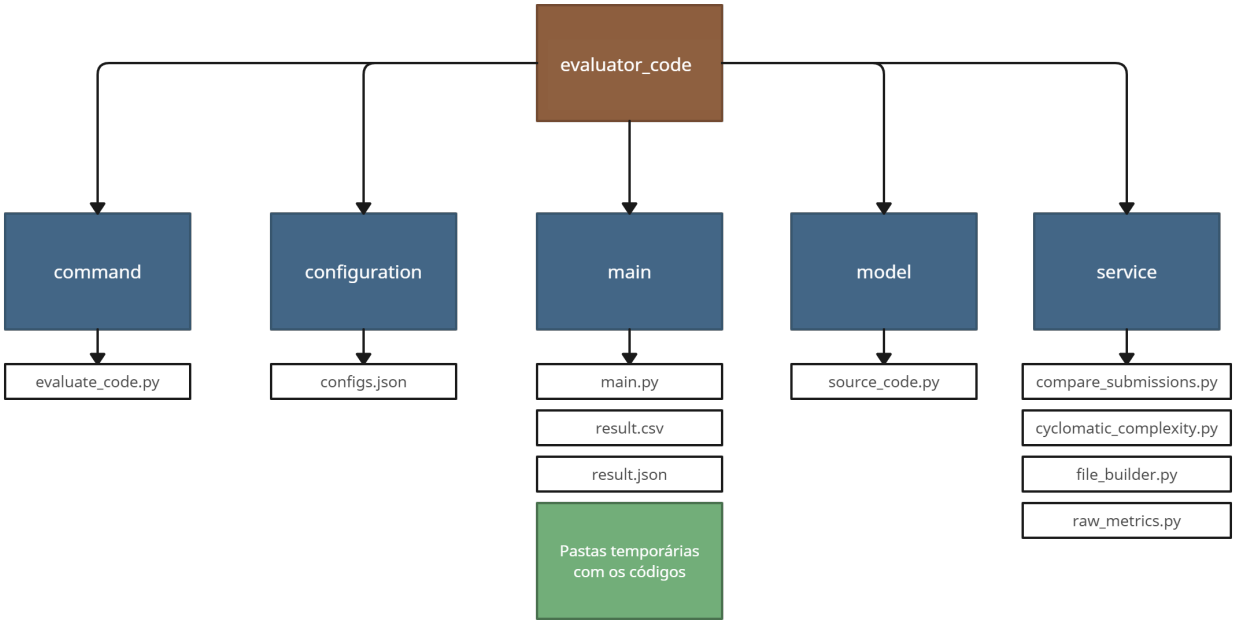
Código 5.1 – Função que cria o arquivo *JSON*

5.1.2 Diretórios

Anteriormente, a ferramenta tinha uma arquitetura com duas pastas distintas, sendo uma de documentos e a outra que contém toda a parte funcional do código.

Dessa forma, foi realizada uma alteração onde os resultados vão direto para a pasta main, onde se encontra o arquivo *main.py* que contém toda a parte de código para rodar a *API* e está detalhado na próxima seção. Dessa forma a pasta docs não tem mais utilidade no projeto. Essa mudança beneficiou a aplicação na questão da facilidade de lidar com os diretórios, uma vez que ficou mais fácil para pegar os dados do arquivo *JSON* e enviar via resposta da requisição na *API*. Na Figura 21 é possível ver como ficaram organizados os diretórios da aplicação.

Figura 21 – Diagrama de diretórios da ferramenta avaliadora



Fonte: Autor (2023)

Outra mudança necessária foi em relação à função que vai no diretório dos códigos-fonte que serão comparados, já que anteriormente a mesma era feita via caminho do diretório, o que se torna inviável em uma *API*. Dessa forma foi utilizada a biblioteca *os*, do Python, para pegar os caminhos dos diretórios de forma automática, assim como no exemplo abaixo.

```
current_directory = os.getcwd()
```

Código 5.2 – Trecho do código em Python que pega o diretório atual

Esse trecho de código foi retirado do arquivo *evaluate_code.py* que se encontra na pasta command, o qual é o responsável por realizar a maioria dos cálculos e não foi alterado nesse quesito.

5.1.3 Construção da *API*

Agora vamos ao arquivo main.py, onde foi construída a *API* com o uso da biblioteca FastAPI do Python, que foi a escolhida devido à sua facilidade de uso. Esse arquivo é responsável por receber, processar e fornecer as respostas às requisições, dessa forma se tornando um elemento essencial para o funcionamento da ferramenta. O funcionamento da *API* é bem simples, ela recebe requisições apenas do tipo *POST* na rota */avaliarsubmissoes*, onde o *body* da requisição deve ter um *JSON* do seguinte modelo ilustrado no código abaixo.

```
{
  "id": 4059,
  "alunos": [
```

```
    {
      "id": 2770922,
      "codigo": "def posicao_inicial(s,vo,a,t):\n    so = s - (vo * t) - (a
          *t**2)/2\n    so = '%.2f' % so\n    return(so) #tirei a identacao\
          ns,vo,a,t = map(float,input().split())\nprint('Posicao:' +
          posicao_inicial(s,vo,a,t) + 'm')"
    }
  ],
  "professor": "S,Vo,a,t = input().split()\nS = float(S)\nVo = float(Vo)\na
      = float(a)\nt = float(t)\ndef posicao_inicial(S,Vo,a,t): #Professor
     eu n li que precisava definir fun??o, ent?o fiz s? com a f?rmula ;-;\n
       So = (Vo*t) + (t*t*a/2)\n  So1 = S - So\n  return So1\nprint(\"
     Posicao:\",\"%.2f\" % posicao_inicial(S,Vo,a,t),\"m\",sep=\"\")\n  ",
  "problema": "Posi   o inicial no MRUV"
}
```

Código 5.3 – Exemplo de modelo de *JSON* para requisição na *API* Avaliadora

Desta forma, o primeiro *ID* é o referente ao problema, e em seguida vem um agrupamento de alunos, onde cada aluno vai ter o seu *ID* único e o seu código que será avaliado e comparado com o do professor. O próximo elemento é o código do professor, e após isso temos o nome do problema. Todos esses dados são obtidos do The Huxley, o método de obtenção está explicado na seção 5.2.

Continuando, ao receber a requisição, a *API* irá atribuir cada elemento do *JSON* a uma variável diferente e processar as submissões dos alunos.

```
async def process_submission(data):
    try:
        id = data['id']
        alunos = data['alunos']
        professor_code = data['professor']
        problema = data['problema']

        alunos_ids = [aluno['id'] for aluno in alunos]
        alunos_codes = [aluno['codigo'] for aluno in alunos]

        folder_path = f"{id}"
        print(folder_path)
        os.makedirs(os.path.join(folder_path, "alunos"), exist_ok=True)
        os.makedirs(os.path.join(folder_path, "professor"), exist_ok=True)

        for aluno in alunos:
            aluno_id = aluno['id']
            aluno_code = aluno['codigo']
            aluno_filename = f"{aluno_id}.py"
            aluno_filepath = os.path.join(folder_path, "alunos",
                aluno_filename)
```

```
21              with open(aluno_filepath, 'w') as aluno_file:
22                  aluno_file.write(aluno_code)
23
24          with open(os.path.join(folder_path, "professor", "professor.py"), '
                w') as professor_file:
25              professor_file.write(professor_code)
26
27          current_directory = os.getcwd()
28          # current_directory = "."
29          full_folder_path = os.path.join(current_directory, folder_path)
30          main_flow(folder_path, full_folder_path)
31
32          with open(os.path.join(full_folder_path, folder_path), 'r') as
                json_file:
33              dados = json.load(json_file)
34
35          shutil.rmtree(folder_path)
36
37          return dados
38
39      except Exception as e:
40          error_message = str(e)
41          traceback_info = traceback.format_exc()
42          shutil.rmtree(folder_path)
43
44          response = {
45              "error": error_message,
46              "traceback": traceback_info
47          }
48          raise HTTPException(status_code=500, detail=response)
```

Código 5.4 – Código-fonte em Python

É criada uma pasta de alunos e dentro dela estão os arquivos de código Python de cada um deles. Esses códigos posteriormente serão calculados com o código do professor que foi salvo no arquivo *professor.py*, dessa forma, os diretórios estão definidos e a *API* chama a função *main_flow*, passando como argumento os diretórios dos códigos dos alunos e do professor. Após a função realizar todos os seus comandos, os diretórios são apagados e há um tratamento de exceções, que registra as informações sobre os erros e retorna uma resposta de erro *HTTP* com código 500.

Para iniciar o servidor FastAPI é utilizada outra biblioteca do Python chamada "uvicorn", que foi escolhida devido ao seu suporte a processamento assíncrono.

```
1 @app.post('/avaliarsubmissoes')
2 async def process_data(data: dict):
3     results = await asyncio.gather(process_submission(data))
```

```
    return results[0]


if __name__ == '__main__':
    import uvicorn
    port = int(os.environ.get("PORT", 5000))

    uvicorn.run(app, host="0.0.0.0", port=port)
```

Código 5.5 – Código-fonte em Python

Dessa forma, a *API* funciona apenas recebendo requisições com os dados, calculando as pontuações e as devolvendo em forma de resposta à requisição *HTTP*.

5.2 *API* The Avaliator

Nessa seção iremos detalhar o desenvolvimento da *API* do The Avaliator, que é responsável por fazer o intermédio entre os sistemas de *Front-End*, *API* Avaliadora e *API* The Huxley.

5.2.1 Construção da *API* do The Avaliator

A *API* do The Avaliator é a responsável por realizar a comunicação entre a interface *web*, o banco de dados, a API do The Huxley e a *API* avaliadora de código. Logo, o objetivo foi criar algo simples e organizado que consiga intermediar todas essas camadas.

A arquitetura escolhida foi a *MVC (Model-View-Controller)*, no contexto do ASP.NET Core. Essa arquitetura é frequentemente utilizada para a construção de aplicações *web*, já que ela facilita a manutenção do código e separa bem cada parte dele.

A camada de modelo irá representar a lógica de negócio da aplicação, a qual irá responder as requisições da camada de controle para realizar operações acerca dos dados. Abaixo segue um exemplo de uma classe implementada na camada de modelo.

```
    public class AvaliacaoAlunos
    {

        public int IdAvaliacao { get; set; }
        public string IdSubmissaoProf { get; set; }
        public int IdTurma { get; set; }
        public int IdTarefa { get; set; }

        [JsonProperty("PROBLEM")]
        public string Problem { get; set; }

        [JsonProperty("SOLUTION")]
```

```
13          public string Solution { get; set; }
14
15          [JsonProperty("IS_TEACHER")]
16          public string IsTeacher { get; set; }
17
18          [JsonProperty("CYCLOMATIC_COMPLEXITY")]
19          public string CyclomaticComplexity { get; set; }
20
21          [JsonProperty("EXCEEDED_LIMIT_CC")]
22          public string ExceededLimitCC { get; set; }
23
24          [JsonProperty("LINES_OF_CODE")]
25          public string LinesOfCode { get; set; }
26
27          [JsonProperty("EXCEEDED_LIMIT_LOC")]
28          public string ExceededLimitLOC { get; set; }
29
30          [JsonProperty("LOGICAL_LINES_OF_CODE")]
31          public string LogicalLinesOfCode { get; set; }
32
33          [JsonProperty("EXCEEDED_LIMIT_LLOC")]
34          public string ExceededLimitLLOC { get; set; }
35
36          [JsonProperty("SOURCE_LINES_OF_CODE")]
37          public string SourceLinesOfCode { get; set; }
38
39          [JsonProperty("LIMIT_SLOC")]
40          public string LimitSLOC { get; set; }
41
42          [JsonProperty("FINAL_SCORE")]
43          public string FinalScore { get; set; }
44
45          public AvaliacaoProfessor? avaliacaoProfessor { get; set; }
46
47      }
```

Código 5.6 – Código fonte em C-Sharp

Todas as classes da camada seguem o padrão da classe *AvaliacaoAlunos*, a qual contém a estrutura dos dados relacionados às pontuações dos alunos, dessa forma servindo uma representação organizada para tais informações.

No exemplo citado, a classe não apenas encapsula os dados como também a semântica e as regras associadas. Ao implementar essa camada de modelo, temos a separação da lógica de negócio da parte de manipulação de dados, o que deixa o código mais organizado e de fácil manutenção. Ao todo, temos 12 classes implementadas na camada, que contém várias regras de negócio diferentes, como autenticação, dados de turmas e dados de pontuações das submissões.

Em seguida, temos a camada de controle, que é a responsável por lidar com as requisições do usuário, interagir com a camada de modelo para obter dados e retornar as respostas adequadas para o cliente. Segue um exemplo de *controller* implementado na camada na listagem de código abaixo.

```
1  namespace TheAvaliatorAPI.Controllers
2  {
3      [ApiController]
4      [Route("api")]
5      public class LoginController : ControllerBase
6      {
7          private readonly ILogger<LoginController> _logger;
8          private readonly HttpClient _httpClient;
9
10         public LoginController(ILogger<LoginController> logger, HttpClient
               httpClient)
11         {
12             _logger = logger;
13             _httpClient=httpClient;
14         }
15
16         [HttpPost("auth")]
17         public async Task<IActionResult> Login([FromBody] LoginRequest
               request)
18         {
19             try
20             {
21                 string apiUrl = "https://www.thehuxley.com/api/login";
22
23                 var jsonContent = new StringContent(JsonConvert.
                       SerializeObject(request),
24                                                     Encoding.UTF8,
25                                                     "application/json");
26
27                 var response = await _httpClient.PostAsync(apiUrl,
                       jsonContent);
28
29                 if (response.IsSuccessStatusCode)
30                 {
31                     var responseBody = await response.Content.
                           ReadAsStringAsync();
32                     var data = JsonConvert.DeserializeObject<LoginResponse
                           >(responseBody);
33
34                     return Ok(new { data?.Access_token });
35                 }
36                 else
```

```
37                  {
38                      return BadRequest("Failed to authenticate user");
39                  }
40              }
41              catch (Exception ex)
42              {
43                  _logger.LogError(ex, "An error occurred during login");
44                  return StatusCode(500, "An error occurred during login");
45              }
46          }
47      }
48  }
```

Código 5.7 – Código fonte em C-Sharp

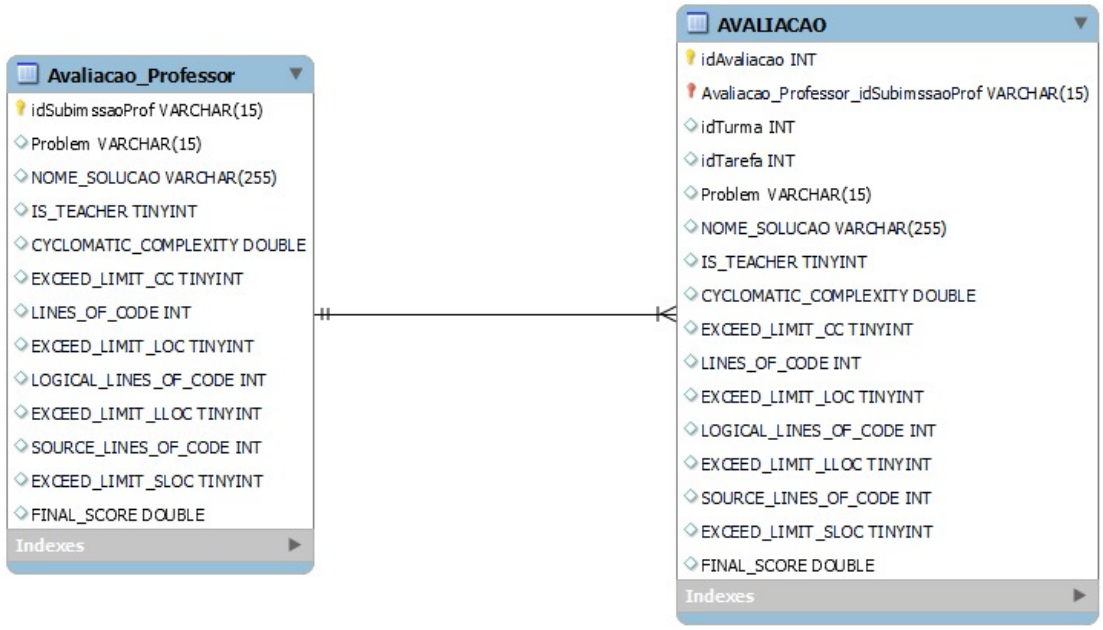
Nesse exemplo acima, temos o *LoginController*, responsável por lidar com as requisições de *login* feitas pelo cliente. Nesse ponto, ele irá interpretar os parâmetros recebidos, e realizar a conversão de dados. Além disso, também existe uma etapa em que, se necessário, o *controller* lida com as exceções.

Todos os demais *controllers* também seguem esse padrão e cada um tem uma função no código, já que eles que se comunicam com a *API* do The Huxley por meio das rotas da mesma. Além das requisições de *login*, também são feitas requisições para todas as demais rotas que foram detalhadas na sessão sobre a *API* do The Huxley, onde cada *controller* é responsável por uma dessas rotas.

No contexto do *login*, o *controller* recebe na requisição os dados de *username* e *password*, interpreta esses dados e então realiza uma requisição na *API* do The Huxley para obter o *token* de autorização do usuário. Além do *controller* de *login*, existem também os de avaliação, turmas, problemas, submissões e usuários.

O próximo passo necessário foi a construção e modelagem do banco de dados da aplicação. Para isso foi escolhido um banco de dados relacional, devido ao conhecimento prévio acerca do PostgreSQL. Em seguida, foi feita a modelagem do banco, que, possui apenas duas tabelas que possuem relação entre si na coluna de *ID* da submissão do professor, sendo uma relação de 1 para muitos (1:n), ou seja, temos uma submissão do professor para várias avaliações de alunos. A Figura 22 exibe o modelo de dados da aplicação.

Figura 22 – Modelo de dados da aplicação



Fonte: Autor (2023)

A primeira tabela se refere aos dados da avaliação do professor, os campos escritos com caixa-alta se referem às informações de avaliação que a *API* Avaliadora irá retornar, os quais já foram detalhados no Quadro 3. Segue uma explicação dos demais itens.

- idSubmissaoProf: *ID* da submissão do professor no The Huxley, coluna utilizado para diferenciar cada professor.
- Problem: *ID* do problema no The Huxley, coluna utilizado para diferenciar cada problema.

A segunda tabela se refere aos dados da avaliação do aluno, os campos escritos com caixa-alta também se referem aos dados que a *API* Avaliadora irá retornar nas métricas, assim segue uma explicação dos demais itens.

- idAvaliacao: *ID* da avaliação gerado automaticamente pelo banco de dados, usado para diferenciar as avaliações.
- Problem: *ID* do problema no The Huxley, coluna utilizado para diferenciar cada problema.
- idTurma: *ID* da turma no The Huxley, coluna utilizado para diferenciar cada turma.
- idTarefa: *ID* da tarefa no The Huxley, coluna utilizado para diferenciar cada tarefa.

Sendo assim, com a adição do banco de dados, a partir do momento em que a *API* do The Avaliator receber uma requisição de cálculo de pontuações, ela primeiro irá realizar uma consulta

no banco para verificar se essa avaliação já foi feita anteriormente, dessa forma, economizando tempo de processamento da aplicação.

```
1  private async Task<ActionResult> PostAvaliarUmaSubmissao(Problema request,
       int idTurma, int idTarefa, string IdSubmissaoProf)
2          {
3              try
4              {
5                  var avaliacao = _repositorioAluno.Obter(p => p.Solution ==
                       request.Alunos![0].Id.ToString() && p.IdSubmissaoProf ==
                        IdSubmissaoProf)
6                      .Join(
7                          _repositorioProfessor.ObterTodos(),
8                          aluno => aluno.IdSubmissaoProf,
9                          professor => professor.IdSubmissaoProf,
10                         (aluno, professor) => new
11                         {
12                             Aluno = aluno,
13                             Professor = professor
14                         });
15
16                 if (!avaliacao.Any())
17                 {
18                     List<AvaliacaoAlunos> response = await RequisicaoApi(
                           request);
19
20                     AvaliacaoProfessor avaliacaoprofessor = new
                           AvaliacaoProfessor
21                     {
22                         IdSubmissaoProf = IdSubmissaoProf,
23                         Problem = response[1].Problem,
24                         Solution = response[1].Solution,
25                         IsTeacher = response[1].IsTeacher,
26                         CyclomaticComplexity = response[1].
                               CyclomaticComplexity,
27                         ExceededLimitCC = response[1].ExceededLimitCC,
28                         LinesOfCode = response[1].LinesOfCode,
29                         ExceededLimitLOC = response[1].ExceededLimitLOC,
30                         LogicalLinesOfCode = response[1].LogicalLinesOfCode
                               ,
31                         ExceededLimitLLOC = response[1].ExceededLimitLLOC,
32                         SourceLinesOfCode = response[1].SourceLinesOfCode,
33                         LimitSLOC = response[1].LimitSLOC,
34                         FinalScore = response[1].FinalScore,
35                     };
36                     var avaliacaoProfessor = _repositorioProfessor.Obter(p
                           => p.IdSubmissaoProf == IdSubmissaoProf);
37
```

```
38                      if (!avaliacaoProfessor.Any()){
39                          Console.WriteLine("
                                ----------------------------------------");
40                          Console.WriteLine(avaliacaoprofessor.
                                IdSubmissaoProf);
41                          Console.WriteLine("
                                ----------------------------------------");
42                          _repositorioProfessor.Adicionar(avaliacaoprofessor)
                                ;
43                      }
44
45                      AvaliacaoAlunos avaliacaoAluno = new AvaliacaoAlunos
46                      {
47                          IdTurma = idTurma,
48                          IdTarefa = idTarefa,
49                          IdSubmissaoProf = IdSubmissaoProf,
50                          Problem = response[0].Problem,
51                          Solution = response[0].Solution,
52                          IsTeacher = response[0].IsTeacher,
53                          CyclomaticComplexity = response[0].
                                CyclomaticComplexity,
54                          ExceededLimitCC = response[0].ExceededLimitCC,
55                          LinesOfCode = response[0].LinesOfCode,
56                          ExceededLimitLOC = response[0].ExceededLimitLOC,
57                          LogicalLinesOfCode = response[0].LogicalLinesOfCode
                                ,
58                          ExceededLimitLLOC = response[0].ExceededLimitLLOC,
59                          SourceLinesOfCode = response[0].SourceLinesOfCode,
60                          LimitSLOC = response[0].LimitSLOC,
61                          FinalScore = response[0].FinalScore,
62                      };
63                      _repositorioAluno.Adicionar(avaliacaoAluno);
64
65                      return Ok(response);
66                  }
67
68                  var responseHttp = avaliacao.ToList();
69
70                  responseHttp[0].Aluno.avaliacaoProfessor.AvaliacaoAlunos =
                        null;
71                  AvaliacaoProfessor professor = responseHttp[0].Aluno.
                        avaliacaoProfessor!;
72                  responseHttp[0].Aluno.avaliacaoProfessor = null;
73                  AvaliacaoAlunos aluno = responseHttp[0].Aluno;
74
75                  return Ok(new List<object> { aluno, professor });
76              }
```

```
77              catch (Exception ex)
78              {
79                  _logger.LogError(ex, "An error occurred during avaliate");
80                  return StatusCode(500, "An error occurred during avaliate")
                        ;
81              }
82          }
```

Código 5.8 – Código fonte em C-Sharp

No código acima, o sistema irá verificar se a avaliação já existe no banco de dados, conferindo o *ID* da submissão do aluno e o *ID* da submissão do professor já constam em alguma linha da tabela do banco de dados. Caso já exista, ele irá devolver os dados dessa linha. Caso contrário, ele irá realizar a requisição de avaliação à *API* Avaliadora e então adicionar os dados no banco de dados, para em seguida retornar os dados como resposta à requisição feita.

5.2.2 Construção da interface *web* do The Avaliator

Para a construção da interface gráfica, foi utilizado o *framework* React para JavaScript, devido à sua capacidade de criar interfaces de alto desempenho e com fácil componentização. Essa é uma camada importantíssima da aplicação, pois é com ela que o usuário se comunica diretamente e realiza as consultas necessárias.

Para a construção das telas, foi utiliada a biblioteca *MUI - Material UI*, que é amplamente conhecida entre os usuários de React por trazer diversos componentes e elementos gráficos que podem ser utilizados de forma fácil e prática na construção do site.

Segue um exemplo de utilização de componentes na tela de *login* do The Avaliator.

```
import Box from '@mui/system/Box';
import Styles from './css/Login.module.css';
import BasicInput from './components/BasicInput';
import LoginButton from './components/LoginButton';
import Snackbar from '@mui/material/Snackbar';
import Alert from '@mui/material/Alert';
import AlertTitle from '@mui/material/AlertTitle';
import Auth from '../services/Auth';
import IconButton from '@mui/material/IconButton';
import CloseIcon from '@mui/icons-material/Close';

function Login() {

  ...

  return (
    <div style={{backgroundColor: "#404040",}}>
      <Container style={{
```

```
19          width: "100vw",
20          height: "100vh",
21
22          display: "flex",
23          flexDirection: "row",
24          justifyContent: "center",
25          alignItems: "center",
26        }}>
27
28          <Snackbar open={errorNotification}>
29            <Alert severity="error">
30              <AlertTitle>Falha na autentica  o </AlertTitle>
31              Usu rio ou senha incorretos
32              <IconButton
33                size="medium"
34                aria-label="close"
35                color="inherit"
36                onClick={handleClose}
37              >
38                <CloseIcon fontSize="small" />
39              </IconButton>
40            </Alert>
41          </Snackbar>
42
43
44          <Box style={{
45            width: "500px",
46            height: "336px",
47            background: "#243856",
48          }}>
49            <h3 className="title">The Avaliator</h3>
50            <p>Preencha com as suas credenciais no The Huxley</p>
51            <BasicInput text="Nome de usu rio" type="text" captureUsername
                ={(username) => setUsername(username)} />
52            <BasicInput text="Senha" type="password" captureUsername={(
                password) => setPassword(password)} />
53            <LoginButton text="ENTRAR" action={handleLogin} />
54          </Box>
55        </Container>
56        </div>
57    );
58  }
59
60  export default Login;
```

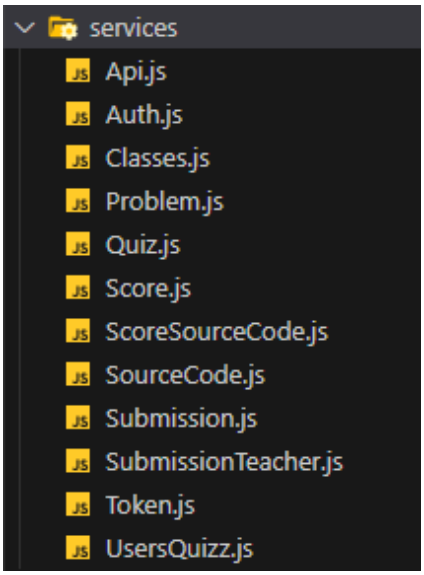
Código 5.9 – Código fonte em JavaScript

Dessa forma, basta instalar a biblioteca e importar os elementos no arquivo que desejar

usá-los, o que facilita e agiliza a codificação e montagem das telas. Além dos componentes do *MUI*, também foram criados outros componentes para utilizar nas telas, como botões, *cards* e itens de lista. A estratégia de componentização foi adotada visando a reutilização dos mesmos em outras partes do código de maneira mais fácil e escalável.

Além da parte visual, também foi utilizado o JavaScript para realizar outros serviços importantes para a aplicação. A Figura 23 exibe esses serviços.

Figura 23 – Lista de serviços da aplicação



Fonte: Autor (2023)

Dessa lista, o serviço base é o *Api.js*, ele serve como um serviço de apoio para os demais serviços e contém apenas o *link* base utilizando a biblioteca Axios para requisitar à *API* do The Avaliator, conforme mostrado na Listagem 5.2.2.

```
1 import axios from "axios";
2
3 const api = axios.create({
4     baseURL:"https://bancotheavaliator-316d4dc95349.herokuapp.com/api"
5 })
6
7 export default api;
```

Código 5.10 – Código fonte em JavaScript

A partir disso, os demais serviços irão importar essa *API* e realizar as operações necessárias, segue abaixo um exemplo do serviço que obtém a lista de tarefas de uma turma.

```
1 import api from './Api';
2
3 // Pega a lista de tarefas e utiliza o token e o ID da turma
```

```
4
5  class Quiz {
6    constructor(token,idturma) {
7      this.token = token;
8      this.url = `/quiz/${idturma}`;
9      this.config = {
10       headers: {
11         'Content-Type': 'application/json',
12         'Accept': 'application/json',
13         'Authorization': `Bearer ${this.token}`
14       }
15     };
16   }
17
18   async getQuiz(){
19     const response = await api.get(
20         this.url,
21         this.config
22       );
23       return response.data;
24   }
25 }
26
27 export default Quiz;
```

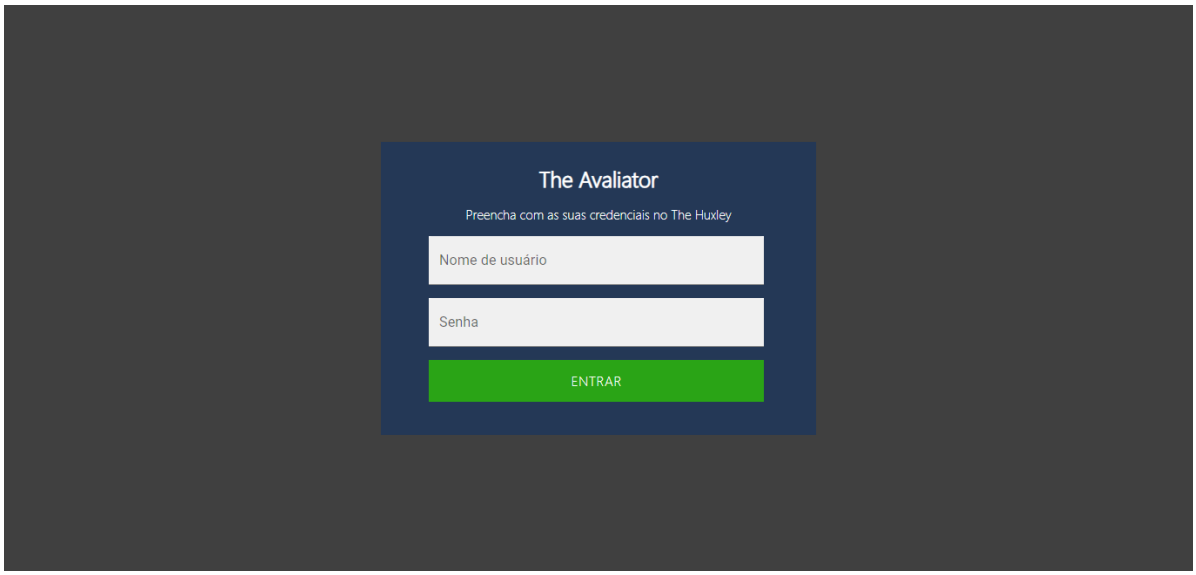
Código 5.11 – Código fonte em JavaScript

A maioria dos demais serviços seguem essa mesma estrutura, com um construtor para o cabeçalho, onde fica o *token* de autenticação (necessário para realizar as requisições à *API* do The Huxley), e um método que realiza a requisição, nesse caso o *GetQuiz*. A única exceção para essa estrutura é o serviço de obtenção de pontuação para as submissões, que não irá lidar com o token de autenticação, pois requisita apenas da *API* Avaliadora.

5.2.2.1 Tela de Login

A primeira tela (Figura 24) construída foi a de *login*, onde o usuário deve realizar a autenticação utilizando o nome de usuário ou *e-mail* e senha do The Huxley.

Figura 24 – Tela de *login* da aplicação



Fonte: Autor (2023)

5.2.2.2 Tela de Turmas

Após realizar o *login*, o professor irá ter acesso a uma tela onde estão listadas as turmas das quais ele faz parte. Também é possível realizar o *logout* através de um botão no canto superior direito da página. A Figura 25 exibe a tela.

Figura 25 – Tela de listagem de turmas da aplicação

Fonte: Autor (2023)

Para acessar uma turma, o professor deve clicar na seta que fica ao lado do nome da turma correspondente. Para a construção dessa página, foi utilizado o serviço *Classes.js*, que,

utilizando o *token* de autenticação do professor, requisita a lista de turmas das quais o professor faz parte e então usamos o resultado dessa requisição para exibir as informações na tela.

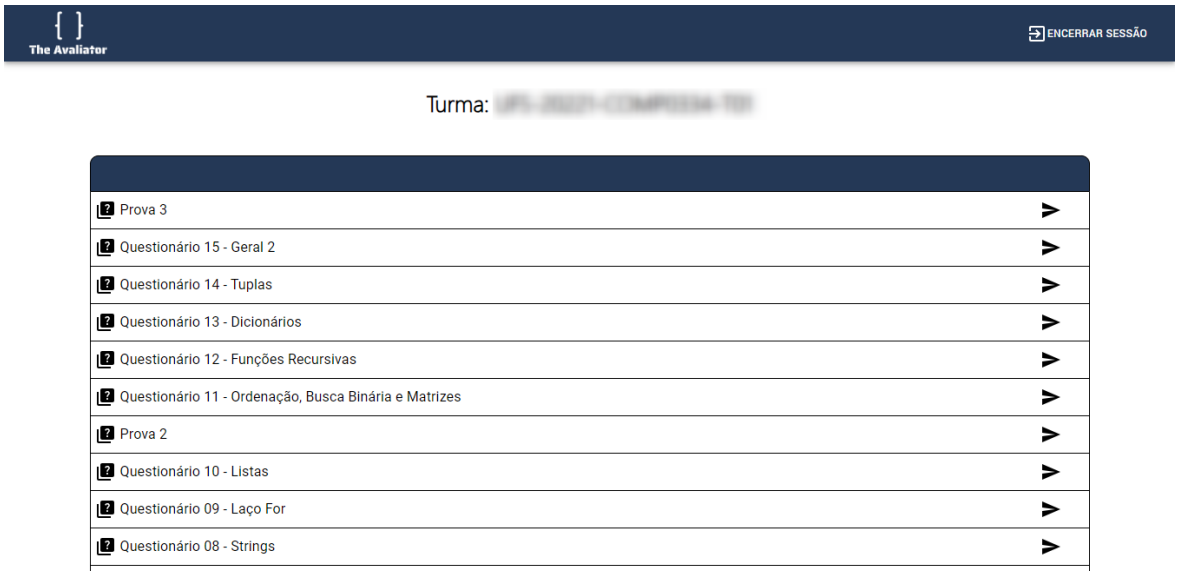
```
...

useEffect(() => {
    if (!sessionStorage.getItem('token')) {
      navigate("/");
    } else {
      classes.getClasses()
        .then((data) => {
          setClassesData(data);
        });
    }
  }, []);

  ...

            <div>
              {classesData.map(({ id, name}) => (
                <ListItem key={id} component="div" disablePadding
                    secondaryAction={<ListItemButton component="a" onClick
                    ={() => acessarTurma(id, name, token)}>
                <SendIcon/>
              </ListItemButton>} style={{ padding: "5px",
                   borderBottom: "1px solid black", borderLeft: "1px solid
                      black", borderRight: "1px solid black",
                }}>
                  <FolderIcon/>
                  <ListItemText primary={name} style={{marginLeft: "5px"
                      }}/>
                </ListItem>
              ))}
            </div>

    ...
```

Código 5.12 – Código fonte em JavaScript

5.2.3 Tela de Tarefas

Ao acessar uma turma, o professor tem acesso então à lista de tarefas dessa turma (Figura 26.

Figura 26 – Tela de listagem de tarefas da aplicação



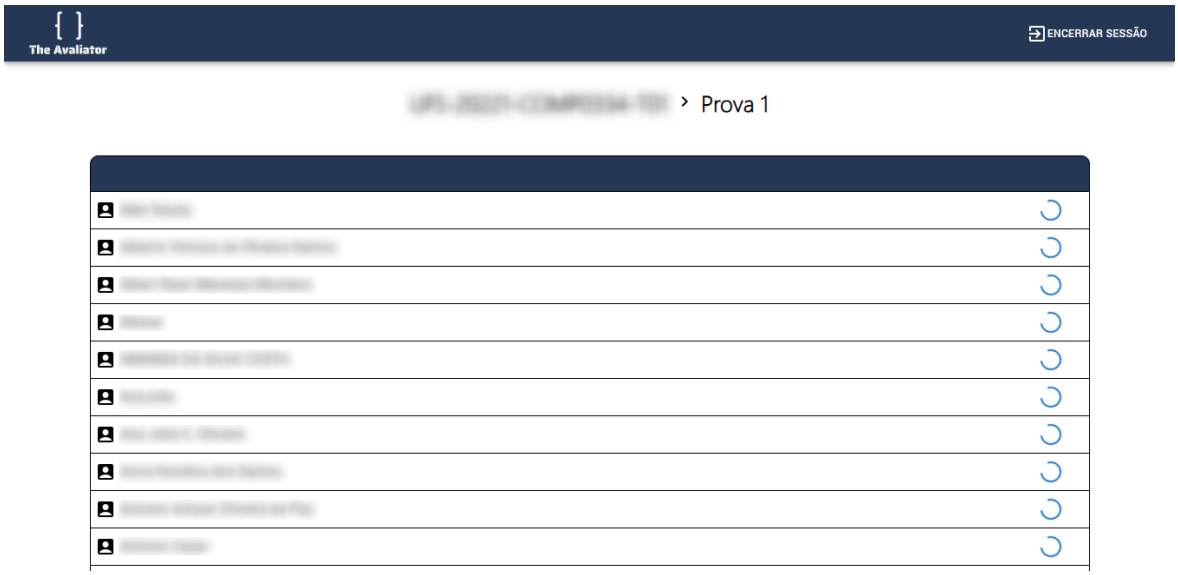
Fonte: Autor (2023)

Para a obtenção dos dados, foi utilizado o serviço *Quiz.js*, que requisita a rota de obtenção da lista de tarefas da *API* do The Huxley, ela segue a mesma estrutura do código anterior para exibir os dados.

5.2.3.1 Tela de Usuários

Ao acessar uma tarefa, o professor obtém então a lista de alunos da turma. Existem etapas diferentes durante o carregamento dessa página. A primeira é quando a aplicação irá iniciar o cálculo das pontuações de cada submissão dos alunos nessa tarefa, então no lugar da seta para acessar determinado aluno que existia nas outras telas, o sistema irá exibir um ícone de carregamento, conforme mostrado na Figura 27.

Figura 27 – Tela de resumo da análise das submissões (1ª Etapa)



Fonte: Autor (2023)

Enquanto esse carregamento está sendo feito, o sistema irá realizar várias requisições para obter essas pontuações. A ordem dessas requisições irá seguir a necessidade de informações, por exemplo: para obter a lista de alunos de uma turma, precisamos do *ID* da turma, então será necessário requisitar a lista de turmas antes para obter essa informação, no atual momento, essa lista já foi requisitada em uma tela anterior e a informação foi repassada via *SessionStorage*, que é um recurso da *API* de Armazenamento *Web* utilizado para armazenar informações na sessão local do usuário.

Primeiro, o sistema irá utilizar o serviço *Problem.js* para obter a lista de problemas. Aqui o sistema também utiliza o serviço *Submission.js* para obter a última submissão correta do professor em Python3 para cada um dos problemas.

```
async function obterProblemas(userData) {
    if (userData.length === 0) {
      return;
    }
    const problems = new Problem(token, idtarefa, userData[0].id);
    const data = await problems.getProblems();

    const jsonArray = data.problems;

    const arraySubmissoesProfessor = await obterSubmissoesProfessor(
       jsonArray);
    if (arraySubmissoesProfessor[0] === "erro"){
      for (const [index, { id }] of userData.entries()) {
        toggleErrorTeacher(id);
        toggleLoading(id);
```

```
15        }
16      }
17      else{
18        for (const [index, { id }] of userData.entries()) {
19          obterPontuacoesCadaAluno(index, id, data, arraySubmissoesProfessor)
20        }
21      }
22    }
```

Código 5.13 – Código fonte em JavaScript

O sistema verifica se o professor possui submissões corretas no *array*. Em caso positivo, dará continuidade, caso contrário, apresentará uma mensagem de erro na tela. Ao acessar a função *obterPontuacoesCadaAluno* (Linha 1), o sistema então utilizará o serviço *Submission.js* para obter a lista de submissões que o aluno realizou, porém, iremos considerar apenas a última submissão correta do aluno em Python3, o mesmo foi feito também para o professor anteriormente. Dessa forma. o sistema terá à disposição os dois códigos que serão utilizados para obter as pontuações.

```
1  async function obterPontuacoesCadaAluno(index, idusuario, data,
       arraySubmissoesProfessor) {
2    const submissoesAlunoAtual = await obterSubmissaoPorAluno(index,
       idusuario, data, arraySubmissoesProfessor);
3
4    console.log(arraySubmissoesProfessor);
5    if (submissoesAlunoAtual.length > 0) {
6      if (arraySubmissoesProfessor.length === 0){
7        toggleErrorTeacher(idusuario);
8        toggleLoading(idusuario);
9      }
10     else {
11
12       const pontuacao = await obterPontuacoes(index, token,
           submissoesAlunoAtual, arraySubmissoesProfessor, data.problems,
           "teste");
13       if (pontuacao.length === 0){
14         toggleErrorCalculo(idusuario);
15         toggleLoading(idusuario);
16       }
17       try {
18         for (let i = 0; i < pontuacao.length; i++) {
19           const pontuacaonumero = parseFloat(pontuacao[i][0].finalScore
               );
20           if (pontuacaonumero < limiteinferior) {
21             toggleWarning(idusuario);
22             console.log(pontuacaonumero);
23             setSubmissoesRuins((prev) => [...prev, parseInt(pontuacao[i
```

```
                    ][0].solution)]);
24              } else if (pontuacaonumero > limitesuperior) {
25                toggleGoodWarning(idusuario);
26                console.log(pontuacaonumero);
27                setSubmissoesBoas((prev) => [...prev, parseInt(pontuacao[i
                      ][0].solution)]);
28              }
29              else if (pontuacaonumero > limiteinferior && pontuacaonumero
                    < 103.0) {
30                console.log(pontuacaonumero);
31                setSubmissoesNormais((prev) => [...prev, parseInt(pontuacao
                      [i][0].solution)]);
32              }
33              if (i === pontuacao.length - 1) {
34                toggleLoading(idusuario);
35              }
36            }
37          } catch {
38            console.error("nao foi possivel ler a pontuacao")
39            console.error(submissoesAlunoAtual);
40          }
41        }
42      }
43    else {
44      toggleLoading(idusuario);
45      toggleError(idusuario);
46    }
47  }
```

Código 5.14 – Código fonte em JavaScript

O sistema irá utilizar dois serviços diferentes, o *SourceCode.js* e o *ScoreSourceCode.js*, o primeiro obtém o código-fonte de uma determinada submissão, e o segundo envia as informações necessárias (*ID* da turma, *ID* da submissão, código da submissão do aluno, código da submissão do professor e *ID* da submissão do professor) para obter a pontuação e demais métricas de determinada pontuação.

Com as pontuações em mãos, o sistema então filtra cada uma delas em uma categoria diferente de acordo com o seu valor.

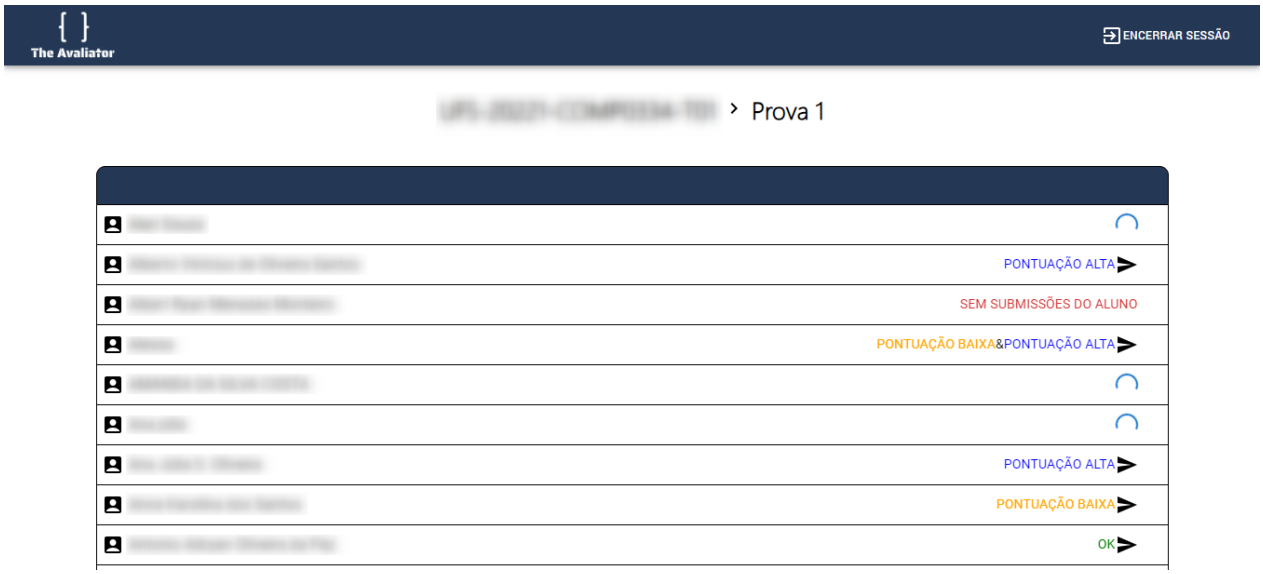
- Pontuação baixa: se a pontuação do aluno for menor que 88.0, o sistema então adiciona o *ID* dessa submissão em um *array* específico para que um aviso de pontuação baixa seja exibido na tela.
- Pontuação normal: se a pontuação do aluno se encaixar entre 88.0 e 103.0 pontos, o sistema então adiciona o *ID* dessa submissão em um array específico para que um aviso de

pontuação normal seja exibido na tela (nesse caso, é exibido um *OK* na tela).

- Pontuação alta: se a pontuação do aluno for maior que 103.0, o sistema então adiciona o *ID* dessa submissão em um array específico para que um aviso de pontuação alta seja exibido na tela.

Todos esses avisos são atualizados por aluno específico e aparecem no lugar do ícone de carregamento, chegando então a segunda etapa da nossa tela. A ordem de prioridade segue sendo pontuações altas e baixas, e após isso uma pontuação *"OK"*, caso o aluno tenha tanto pontuações altas como baixas, é exibido um aviso duplo, é possível ver a tela na Figura 28.

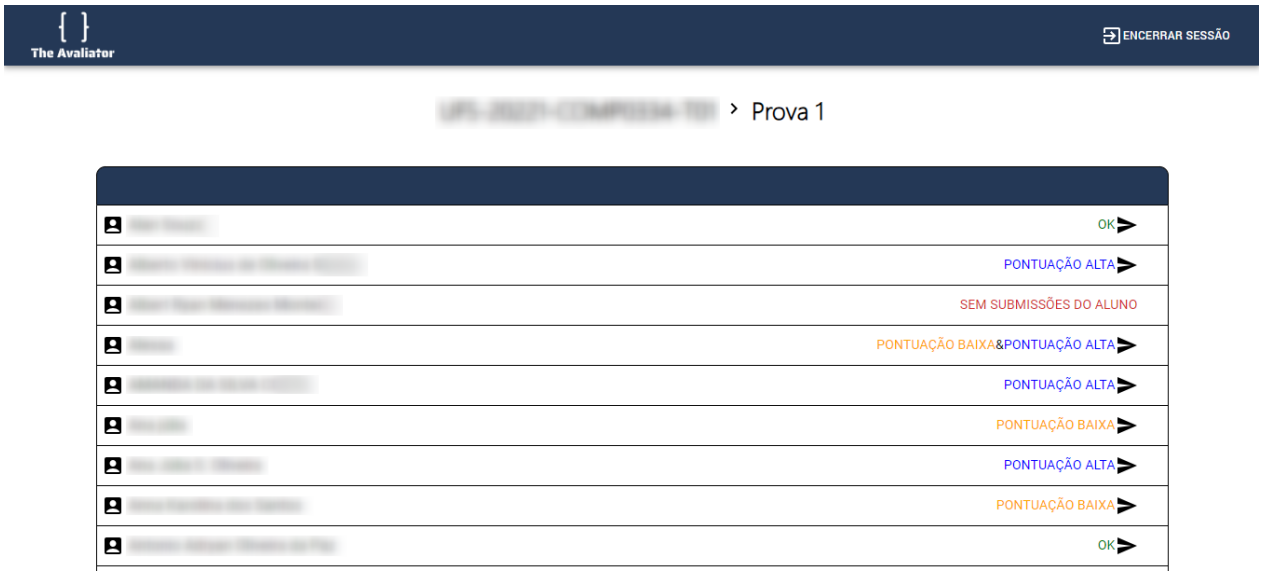
Figura 28 – Tela de resumo da análise das submissões (2ª Etapa)



Fonte: Autor (2023)

A categoria a qual cada pontuação pertence é salva no *SessionStorage* para que essa informação seja reutilizada na próxima tela. A última etapa dessa tela é quando todas as pontuações dos alunos estão carregadas (Figura 29).

Figura 29 – Tela de resumo da análise das submissões (3ª Etapa)



Fonte: Autor (2023)

A qualquer momento quando uma pontuação de um aluno for carregada, o professor pode clicar na seta que aparece ao lado do seu nome e a próxima tela irá abrir em uma nova aba (dessa forma o cálculo das pontuações não é interrompido).

Temos algumas possibilidades de mensagens de *status* que irão aparecer após o carregamento.

- Sem submissões do professor: quando o professor não realizou nenhuma submissão correta para os problemas da tarefa.
- Sem submissões do aluno: quando o aluno não realizou nenhuma submissão correta para os problemas da tarefa.
- Erro no cálculo das submissões: quando ocorre algum erro no cálculo das métricas das submissões.
- *OK*: quando a pontuação do aluno é considerada normal.
- Pontuação alta: quando a pontuação do aluno é considerada acima do esperado.
- Pontuação baixa: quando a pontuação do aluno é considerada abaixo do esperado.

5.2.3.2 Tela de problemas

Ao clicar em um aluno, o professor então irá ter acesso, em uma nova guia, à lista de problemas (Figura 30) da tarefa com um status do valor da pontuação do aluno nesse problema.

Figura 30 – Tela de listagem de problemas da aplicação

Fonte: Autor (2023)

Ao clicar em um desses problemas, uma nova guia é aberta também para que o professor analise o código e as métricas.

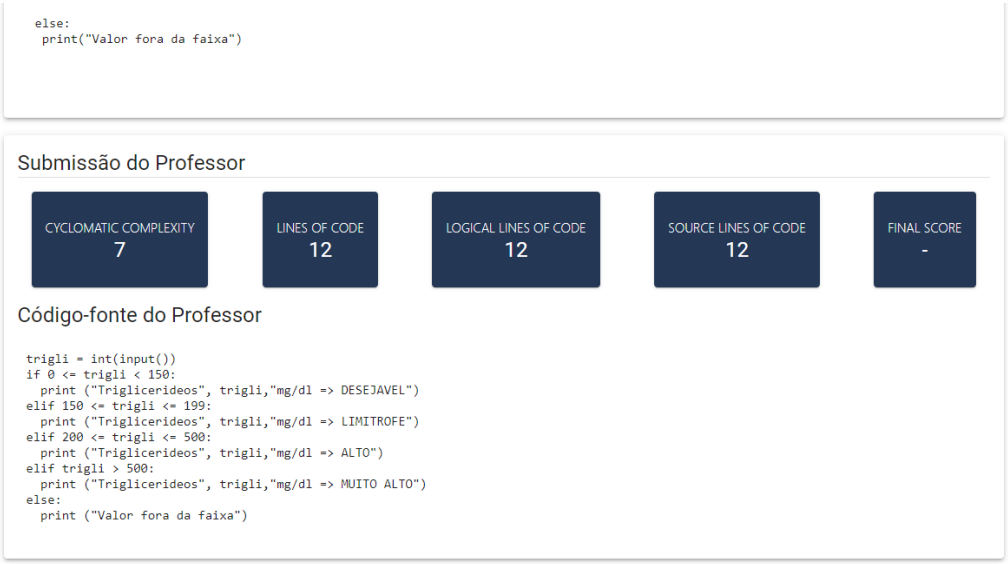
5.2.3.3 Tela de Pontuações

Ao chegar na tela de pontuações (Figura 31 e Figura 32), o professor então tem acesso ao código-fonte da submissão do aluno com as suas métricas, e também tem acesso ao seu próprio código com as suas métricas (menos a pontuação, que para o professor sempre é fixada em 100.0)

Figura 31 – Tela de análise da submissão (1ª parte)

Fonte: Autor (2023)

Figura 32 – Tela de análise da submissão (2ª parte)



Fonte: Autor (2023)

Essa tela proporciona ao professor a possibilidade de analisar os dois códigos e compará-los, dessa forma tornando mais fácil a identificação dos problemas que tornaram a pontuação baixa, ou dos pontos positivos que a fizeram subir.

6

Considerações Finais e Trabalhos Futuros

O uso da tecnologia no ensino da programação tende a beneficiar os professores e estudantes, sendo importante garantir a praticidade no ensino de forma a agilizar a realização e correção de tarefas.

A construção do sistema *web* para avaliação de submissões do The Huxley propõe atender as necessidades dos professores em relação à análise de desempenho dos alunos. Porém, a ferramenta por si só não solucionará todos os problemas, já que é necessário que o professor faça um planejamento adequado para se beneficiar ao máximo da mesma, como o uso de metodologias de ensino que se adequem ao contexto.

As reuniões de orientação do projeto foram bem produtivas, e, após meses de trabalho, os objetivos traçados anteriormente foram alcançados. A aplicação *web* produzida já está sendo testada em turmas de Programação Imperativa na Universidade Federal de Sergipe e o *feedback* tem sido positivo.

Como trabalhos futuros, uma possibilidade é a implementação da plataforma do The Avaliator no site do The Huxley, assim aumentando ainda mais a sua praticidade de uso, de modo que o professor não precise acessar outra plataforma para acessar as pontuações de cada submissão. Outra possibilidade é a expansão da *API* Avaliadora para que suporte novas linguagens, além de Python, tornando assim mais amplas as possibilidades. Além disso, um conjunto adicional de métricas pode ser considerado na análise do código-fonte. Finalmente, poderia ser gerado o resultado da análise por tarefa em formato de relatório, contendo apenas as submissões que requerem atenção do professor.

Referências

BERSSANETTE, J. H.; FRANCISCO, A. C. d.; BARAN, L. R. Espaços ampliados apoiados por tecnologias digitais para o ensino de programação de computadores. *Edição Especial: Congresso Internacional de Educação da Uniamérica*, v. 12, n. 25, 2018. Citado na página 9.

CAMARGOS, J. G. C. d. et al. Uma análise comparativa entre os frameworks javascript angular e react. *Computação e Sociedade - Universidade FUMEC*, v. 1, n. 1, 2019. Citado na página 28.

FRANCISCO, R. E. *Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte*. 115 p. — Universidade Federal de Goiás, Goiânia/GO, 2016. Citado na página 13.

GAMMA, E. et al. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. 1ª impressão. ed. Porto Alegre: Bookman, 2000. ISBN 9788573076103. Disponível em: <https://www.amazon.com.br/PadrÃµes-Projetos-SoluÃ§Ãµes-ReutilizÃ¡veis-Orientados/dp/8573076100/>. Citado na página 40.

MARTIN, R. C. *Código limpo: habilidades práticas do Agile software*. Alta Books, 2011. ISBN 9788576082675. Disponível em: <https://www.amazon.com.br/dp/8576082675/>. Citado na página 10.

PARK, R. E. *Software Size Measurement: A Framework for Counting Source Statements*. [S.l.: s.n.], 1992. Citado na página 22.

RADON 4.1.0 documentation. 2023. Disponível em: <https://radon.readthedocs.io/en/latest/index.html>. Citado 3 vezes nas páginas 20, 21 e 22.

SANTOS, J. C. d. S.; RIBEIRO, A. d. R. L. Uma proposta de um juiz online didático para o ensino de programação. *II ENINED - Encontro Nacional de Informática e Educação*, 2011. Citado na página 9.

SEMESP, I. *Mapa do Ensino Superior no Brasil*. 2021. Disponível em: <https://www.semesp.org.br/wp-content/uploads/2021/06/Mapa-do-Ensino-Superior-Completo.pdf>. Citado na página 9.

SEVERINO, A. J. *Metodologia do Trabalho Científico*. [S.l.]: Cortez Editora, 2014. Citado na página 11.

THE Huxley. 2023. Disponível em: <https://www.thehuxley.com/>. Citado na página 9.

VIEIRA, V. S. *Uma abordagem para pontuação de exercícios de programação baseada em análise estática de código*. 54 p. — Universidade Federal de Sergipe, São Cristóvão/SE, 2022. Citado 13 vezes nas páginas 10, 11, 12, 13, 20, 22, 23, 25, 26, 27, 29, 40 e 41.