
UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

# Uma ferramenta Web para simplificar mensagens de erro de programação para estudantes iniciantes

Trabalho de Conclusão de Curso

Valmir Vinicius da Cunha Rezende


São Cristóvão – Sergipe

2024

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Valmir Vinicius da Cunha Rezende

# Uma ferramenta Web para simplificar mensagens de erro de programação para estudantes iniciantes

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): Alberto Costa Neto

São Cristóvão – Sergipe

2024

# Resumo

A prática em computação vem sendo amplamente ajudada pela utilização de Juízes Online. Estes são aplicativos que podem ser utilizadas para auxílio no desenvolvimento da prática na construção de código-fonte e lógica de programação. O Juiz Online recebe a submissão de código-fonte e realiza a execução do mesmo, submetendo-o a cada caso de teste do problema. Caso ocorra um erro, uma mensagem é retornada ao estudante, porém, essas mensagens podem ser difíceis de entender, aumentando a curva de aprendizado do estudante. Diante desse problema o projeto desenvolvido nesse trabalho tem como objetivo converter essas mensagens de difícil entendimento em mensagens simples. Como não é possível realizar esse trabalho de conversão para todos os juízes online, o The Huxley, como um dos mais populares na comunidade acadêmica brasileira, foi o foco desse trabalho. Para a realização das atividades foram realizados estudos na *API* desenvolvida por Galileu Santos de Jesus (EME), Mestre em Ciência da Computação pela Universidade Federal de Sergipe (UFS), responsável por converter as mensagens de erro em mensagens amigáveis, e na *API* do The Huxley, a fim de realizar a autenticação do usuário e acesso às submissões. Dessa maneira, a aplicação desenvolvida serve como interface web para integrar os dois ambientes, permitindo a utilização pelo estudante. A partir daí, foram elicitados requisitos do software a ser construído, com o objetivo de delimitar seu escopo, que permitiu a continuidade das atividades deste trabalho. O desenvolvimento da aplicação resultou em uma integração bem-sucedida entre a *API* EME e a *API* do TH, possibilitando a conversão de mensagens de erro complexas em mensagens de fácil compreensão. A interface web resultante proporciona aos estudantes uma experiência de utilização simples para obter as mensagens de submissões incorretas do TH convertidas para uma mensagem mais amigável.

**Palavras-chave**: Juiz online, The Huxley, Mensagens de error, Software educacional.

# Abstract

The practice in computer science has been widely assisted by the use of Online Judges. They are applications that can be used to support source code development and programming logic practice. The Online Judge receives the submission of source code and executes it, subjecting it to each test case of the problem. If an error occurs, a message is returned to the student; however, these messages can be difficult to understand, increasing the student's learning curve. Faced with this problem, the project developed in this work aims to convert these difficult-to-understand messages into simple messages. Since it is not possible to perform this conversion task for all online judges, The Huxley, as one of the most popular in the Brazilian academic community, was the focus of this work. To carry out the activities, studies were conducted on the API developed by Galileu Santos de Jesus (EME), Master in Computer Science from the Federal University of Sergipe (UFS), responsible for converting error messages into user-friendly messages, and on The Huxley's API, in order to authenticate the user and access submissions. Thus, the developed application serves as a web interface to integrate the two environments, allowing student use. From there, software requirements were elicited to delimit its scope, enabling the continuation of the activities in this work. The development of the application resulted in a successful integration between the EME API and the TH API, enabling the conversion of complex error messages into easily understandable messages. The resulting web interface provides students with a simple usage experience to convert incorrect TH submission messages into a friendlier message.

**Keywords**: Online Judge, The Huxley, Error Messages, Educational Software.

# Lista de ilustrações

# Lista de Quadros

# Lista de abreviaturas e siglas

| | |
|---|---|
| API | Application Programming Interface |
| CSV | Comma-Separated Values |
| DCOMP | Departamento de computação |
| EME | Explain my Error |
| ES | Engenharia de Software |
| ER | Engenharia de Requisitos |
| FM | Friendly Message |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| JSON | JavaScript Object Notation |
| MVC | Model-View-Controller |
| REST | Representational State Transfer |
| RF | Requisitos Funcionais |
| RNF | Requisitos Não Funcionais |
| TH | The Huxley |
| TI | Tecnologia da Informação |
| UFS | Universidade Federal de Sergipe |
| UML | Unified Modeling Language |
| XML | Extensible Markup Language |

# Sumário

# 1
# Introdução

Nos últimos anos, a área de Tecnologia da Informação (TI) ganhou bastante notoriedade, especialmente com a pandemia da COVID-19, como discutido em (GANDHI et al., 2020), que mostrou a aceleração na adoção dessas soluções pelas empresas durante a pandemia. Com a demanda em alta e a possibilidade de trabalhar em *home office*, desenvolvedores de software têm vislumbrado melhores oportunidades. Isso gerou um aumento no ingresso de estudantes na área, seja por meio de cursos online, técnicos ou de graduação, como mostra o censo do ensino superior ((INEP), 2021, pg.46), onde em 2019 aproximadamente 1 milhão e 200 mil estudantes ingressaram no ensino superior, tanto presencialmente quanto EAD, e em 2021 aproximadamente 1 milhão e 600 mil ingressaram, representando um aumento de cerca de 33%.

É sabido que a prática em desenvolvimento de software é uma atividade fundamental para a formação de pesquisadores, desenvolvedores, engenheiros de software e muitas outras carreiras na área de TI. Porém, muitas das vezes, devido à grande quantidade de alunos, os docentes acabam não conseguindo atender a todos em tempo hábil. Para lidar com isso, muitos professores têm adotado ferramentas automatizadas, como os juízes online, a exemplo do The Huxley[1].

O The Huxley é uma aplicação web que dispõe de uma grande quantidade de exercícios práticos, onde os usuário podem tentar resolvê-los e enviar suas soluções para que a ferramenta faça a análise, indicando se está correta ou incorreta.

Graças a juízes online como este, o problema de realização e acompanhamento de atividades práticas foi de certa maneira abordado. Por meio desta ferramenta, o aluno tem acesso a milhares de problemas que vão dos mais básicos até o mais complexos, visando uma grande cobertura sobre a prática no aprendizado. Porém, a sua quantidade de exercícios não foi o fator que levou à utilização de ferramentas como esta. A principal funcionalidade foi a capacidade de *feedback* rápido, ou seja, o usuário tem uma análise rápida sobre a solução construída, reduzindo

[1] The Huxley: <https://www.thehuxley.com/>

a necessidade de participação do docente.

Entretanto, os juízes online não são capazes de resolver todos os problemas. Devido à natureza do processo de aprendizado, que demanda ampla prática, é natural que os alunos enfrentem inúmeros desafios ao implementar suas soluções, resultando em erros de sintaxe e lógica, os quais podem ser difíceis de compreender e corrigir. Nesse ponto está uma das carências dos juízes online, que não conseguem prover um *feedback* que facilite o entendimento do aluno sobre o erro, já que o aluno só tem acesso à mensagem fornecida pelo interpretador ou compilador, a qual geralmente é bastante confusa e escrita na língua inglesa. Em alguns juízes online, mais direcionados para maratonas de programação, a mensagem de erro nem é apresentada, já que o aluno é informado apenas que houve um erro de compilação ou execução.

Galileu Santos de Jesus, Mestre em Ciência da Computação pela Universidade Federal de Sergipe (UFS), desenvolveu um projeto visando auxiliar alunos iniciantes na compreensão de mensagens de erro em seus programas. Em seu projeto, denominado Explain my Error (EME) (JESUS, 2018), criou uma *Application Programming Interface* (*API*) para traduzir as mensagens de erro originais, tornando-as mais compreensíveis e descritas em português.

Porém, a *API* construída em (JESUS, 2018) não é de simples acesso direto para estudantes iniciantes, já que foi idealizada para ser integrada a outras ferramentas, como juízes online. Para utiliza-la é necessário ter um conhecimento sobre envio de requisições *HTTP* e *JSON*. Esses são conceitos que, naturalmente, não se espera que um iniciante na área possua.

A *API* desenvolvida encontra-se em estado evolutivo, onde foram adicionadas, juntamente com as mensagens amigáveis retornadas, *links* de auxílio para solução do erro. Entretanto, essa funcionalidade ainda não estava completa.

Dessa maneira, este projeto teve como objetivo concluir o desenvolvimento da funcionalidade de links de auxílio, desenvolver uma aplicação web que possa ser utilizada como interface para a *API* construída em (JESUS, 2018), em conjunto com a construção de um módulo para importação das mensagens de erro em submissões no The Huxley, visando prover rapidamente um *feedback* com mensagens amigáveis, auxiliando alunos iniciantes a decifrar as mensagens de erro retornadas em suas submissões no The Huxley e descrevendo melhor seu conteúdo.

## 1.1 Objetivos

### 1.1.1 Geral

Esse trabalho teve como objetivo desenvolver uma aplicação web educacional para auxiliar os alunos iniciantes em desenvolvimento de software a decifrar mensagens de erros retornadas em suas submissões no The Huxley, através da conversão realizada pela *API* EME, descrevendo melhor seu conteúdo.

### 1.1.2 Específicos

Para alcançar o objetivo geral, em todas as etapas do projeto, foi adotada a utilização de metodologias ágeis (BECK et al., 2001), com o intuito de atingir os objetivos específicos a seguir:

- Desenvolver o Front-end de uma aplicação web que proveja uma interface simples e intuitiva para o usuário, utilizando o React[2] como tecnologia. Dessa forma, será construído um ambiente agradável para os usuários;
- Desenvolver o Back-end da aplicação web seguindo padrões, como o *Model-View-Controller* (*MVC*) e de código limpo estabelecidos por (MARTIN, 2011), além de outras boas práticas de desenvolvimento. Isso permitirá que a aplicação seja extensível e de fácil manutenção para trabalhos futuros;
- Realizar a integração com a *API REST* do The Huxley para obtenção dos dados necessários. Isso permitirá que os dados sejam obtidos de forma confiável e eficiente.
- Realizar a integração com a *API REST* do EME para conversão das mensagens de erro retornadas nas submissões.
- Realizar a extensão da *API REST* do EME para obter links de fóruns que auxiliem na correção do problema.

## 1.2 Estrutura do trabalho

Os capítulos do trabalho estão estruturados da seguinte forma:

O capítulo 1 apresentou uma contextualização do problema e do escopo do trabalho. O capítulo 2 abordará com maior profundidade os principais conceitos para fundamentar a teoria de juízes online, The Huxley, EME, desenvolvimento de software, nos quais o presente trabalho se baseia, e métodos ágeis. No capítulo 3, serão descritos os requisitos funcionais e não funcionais, a definição dos casos de uso e a arquitetura do projeto. No capítulo 4, serão apresentadas as ferramentas que foram utilizadas para o desenvolvimento. O capítulo 5 contará com detalhes referentes ao desenvolvimento da aplicação. Por fim, o capítulo 6 trará considerações finais e trabalhos futuros para este projeto.

[2] React: <https://react.dev/>

# 2

# Fundamentação Teórica

O presente capítulo abordará de forma mais detalhada os seguintes temas: Juízes online, The Huxley, EME, Desenvolvimento de software e Metodologia Ágil, considerados essenciais para o entendimento do trabalho.

Na seção 2.1, será explicado o que são Juízes online e como funcionam. A seção 2.2 trata das funcionalidades da aplicação The Huxley, tendo sido utilizados *screenshot* retirados da aplicação web (HUXLEY, 2023). Na seção 2.3, será abordada a *API* EME, apresentando detalhes de construção e funcionamento. Em seguida, na seção 2.4, será discutido o conceito de desenvolvimento de software, comentando sobre Engenharia de Software e Engenharia de Requisitos, a fim de facilitar o entendimento do processo de desenvolvimento da ferramenta, contido no capítulo 3. Por fim, na seção 2.5, será apresentado o conceito de Metodologia Ágil, que foi fundamental para a organização e construção do projeto em tempo hábil.

## 2.1 Juízes online

Segundo (KURNIA; LIM; CHEANG, 2003), a prática é indispensável para a formação profissional da área de desenvolvimento. Entretanto, desenvolver programas de computador é de difícil realização devido à dificuldade de correção e de acompanhamento ao estudante. Para avaliar uma solução proposta, vários passos tem que ser feitos. Alguns deles são testes em diversos casos, com condições diferentes, onde a solução deve apresentar êxito em todas. Graças a toda essa complexidade, surgiu a necessidade de automatizar esse processo de correção e apoio, que seria o mesmo para cada solução.

A partir dessa necessidade, no início dos anos 2000 os primeiros juízes online surgiram. Tratam-se de plataformas de aprendizagem e prática de programação que disponibilizam problemas para todos os tipos de usuários. Esses problemas podem variar em dificuldade e escopo, desde desafios simples de programação até problemas complexos de algoritmos e estruturas de

dados.

### 2.1.1 Processo de execução de um Juiz online

Juízes online utilizam um sistema de compilação ou interpretação integrado. Quando uma solução é enviada, o juiz compila o código e o executa em um ambiente controlado e seguro. Em seguida, ele executa uma série de testes automatizados no código para verificar se ele produz a saída correta para um conjunto de entradas específicas, os casos de teste, antes citados.

A avaliação da solução é feita durantes dois processos, compilação e execução, onde em cada um deles há condições a serem cumpridas para aceitação da solução enviada. Caso sejam encontradas falhas em qualquer uma dessas etapas, o juiz tem por responsabilidade fornecer *feedback* sobre os erros encontrados. Essas informações são as mensagens de retorno, que podem ser complexas para usuários iniciantes.

O The Huxley é um juiz online popular usado por estudantes, professores e profissionais de programação em todo o mundo. No entanto, há muitos outros juízes online que também possuem bastante popularidade, como o Beecrowd[1] (antigo URI Online Judge), Codeforces[2] e Uva online judge[3].

## 2.2 The Huxley

O The Huxley foi desenvolvido pelo Prof. Rodrigo Paes e seus orientandos da Universidade Federal de Alagoas, no Brasil, como uma plataforma para auxílio no desenvolvimento da prática em programação. Ele foi projetado para ser uma aplicação educacional que pode ser usada por qualquer um que deseje exercitar sua prática em desenvolvimento, seja iniciante ou não. A plataforma oferece uma ampla variedade de problemas e exercícios de programação em várias linguagens de programação, incluindo C, C++, Java, Python, Octave e Pascal.

O funcionamento do The Huxley é simples e intuitivo. Quando os usuários acessam a plataforma, eles podem escolher entre uma variedade de problemas de programação disponíveis para solucionar. Os usuários podem escolher o nível de dificuldade, o tipo de problema e a linguagem de programação para compor melhor sua aprendizagem.

O The Huxley executa uma série de casos de teste, baseados em entrada e saída, usando o código-fonte submetido pelos usuários para verificar se a solução está correta e é executada dentro de um tempo limite. Os usuários recebem uma pontuação com base na precisão e na eficiência da solução. Se o código não passar em todos os testes, os usuários podem tentar novamente e enviar uma nova solução para avaliação.

[1] Beecrowd: <https://www.beecrowd.com.br/>
[2] Codeforces: <https://codeforces.com/>
[3] Uva: <https://onlinejudge.org/>

Em um estudo realizado por (SILVA; SILVA; MARTINS, 2018), a utilização do The Huxley resultou em um maior aproveitamento por parte dos alunos na resolução dos problemas propostos. Onde foi observado sua motivação e interação na resolução do problemas, o que pode levar a um melhor aprendizado.

Em resumo, o The Huxley é um exemplo de um juiz online eficaz e popular que é usado por estudantes e profissionais de programação em todo o mundo. A aplicação oferece uma ampla variedade de problemas de programação que podem ajudar os usuários a melhorar suas habilidades em programação.

### 2.2.1 Representação de problemas no The Huxley

No The Huxley, os problemas são divididos em três categorias: Algoritmo, Múltipla escolha e Complete o código. Os problemas de "Algoritmo" tratam-se da apresentação de um problema onde o usuário deve desenvolver uma solução em código com alguma linguagem aceita e submetê-la à ferramenta para análise. Os problemas de múltipla escolha tratam-se de questões objetivas onde são feitas perguntas sobre qualquer tema desejado pelo professor que a elaborou. A terceira e última são os problemas de "Complete o código", onde basicamente um código pré-montado, em alguma linguagem, é entregue ao usuário e ele deve completar a lógica para encontrar a solução desejada.

Cada um dos problemas possui um nível de dificuldade, sendo eles: iniciante, fácil, médio, avançado e expert. Esses níveis são representados por números, variando de 1 a 5, sendo o nível 1 correspondente ao iniciante e o nível 5 ao expert.

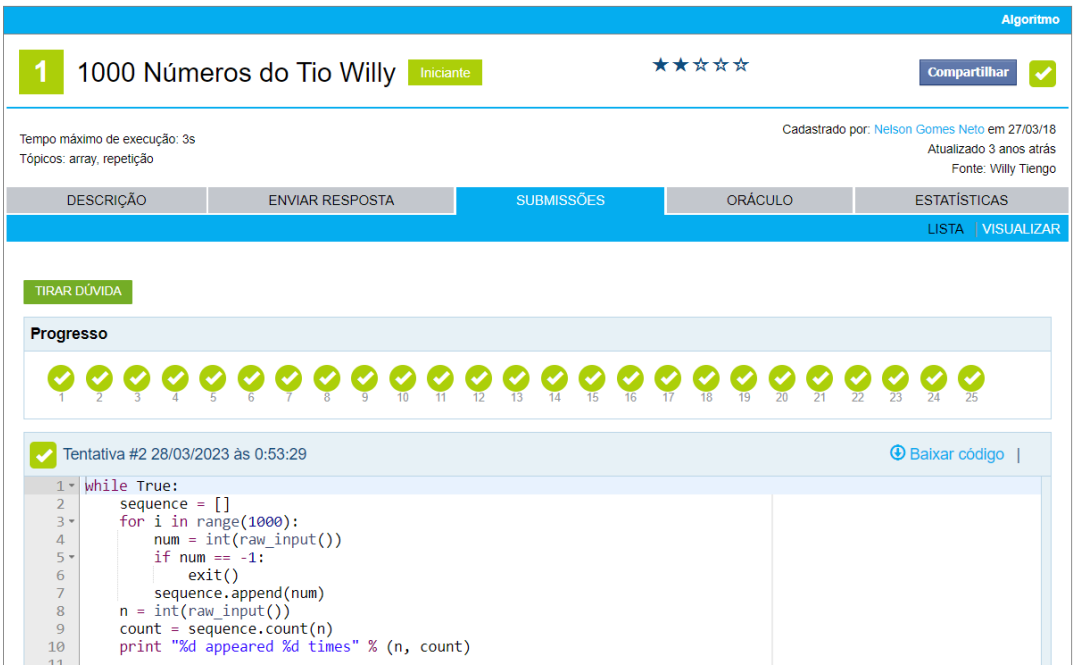
### 2.2.2 Submissões e mensagens de erro no The Huxley

Na aba de submissões, acessível ao fazer após entrar no The Huxley, é possível visualizar todas as já realizadas pelo usuário ao The Huxley. A aba conta com opções de filtros para facilitar a listagem visto que um registro é criado para cada submissão feita. Os filtros são referentes aos problemas, avaliação e intervalo de tempo.

Na listagem do registro consta a data da submissão, o problema a que ela se refere, a avaliação, que indica o sucesso ou tipo de erro resultante, a linguagem utilizada na solução e uma opção de visualização completa da submissão.

Na visualização completa da submissão temos acesso às informações do problema, como o tipo e a dificuldade. Junto a isso, a visualização também é composta por informação da execução da solução, exibindo o tempo de execução, os casos de testes feitos e o código executado, como exibido na Figura 1.

Figura 1 – Visualização de uma submissão bem-sucedida




Fonte: Autor (2023)

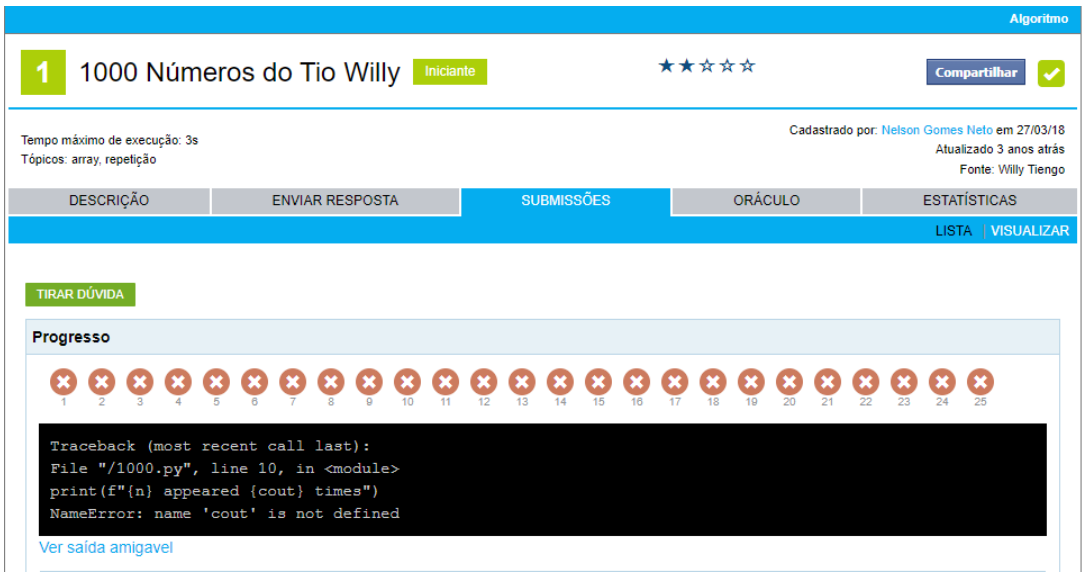
A Figura 1 exibe a visualização de uma submissão bem-sucedida, ou seja, a solução criada para o problemas conseguiu atender a todos os casos de testes. Entretanto, também é possível visualizar as submissões mal-sucedidas, ou seja, submissões cuja solução enviada não conseguiu atender a todos os casos de testes ou gerou algum outro erro no processo de compilação/execução.

Figura 2 – Visualização de uma submissão mal-sucedida


Fonte: Autor (2023)

Na exibição da submissão incorreta, ao clicar em algum caso de teste, representados pelas bolinhas vermelhas enumeradas, é possível visualizar a mensagem de erro gerada na execução do problema. Essa deve ser utilizada como guia para resolver os problemas no código enviado.

Figura 3 – Visualização de uma mensagem de erro




Fonte: Autor (2023)

Essa mensagem de erro retornada é o foco desse projeto. Facilitar o entendimento da mesma foi o objetivo da construção da *API* EME (JESUS, 2018). A possibilidade de tornar essa

*API* acessível aos usuários é um dos foco desse trabalho de conclusão de curso.

## 2.3 *API* do The Huxley

O The Huxley dispõe de uma *API* crucial, responsável por fornecer todos os dados necessários para a plataforma web do juiz online. Esta *API* permite a interação com os diversos recursos oferecidos pela plataforma educacional, tais como a obtenção de problemas e submissões. No desenvolvimento do Friendly Message, a *API* do The Huxley desempenhou um papel crucial como fonte de dados, sendo utilizada tanto para autenticação quanto para a obtenção de submissões incorretas.

Infelizmente, no momento da criação deste projeto, a documentação da *API* não está disponível publicamente, exigindo um esforço manual para entender como utilizá-la. Este processo será detalhadamente abordado na seção subsequente, intitulada "Obtenção das Rotas do The Huxley". Essa seção fornecerá informações detalhadas sobre como as rotas foram identificadas e documentadas.

### 2.3.1 Obtenção das rotas do The Huxley

Para obter os dados da *API* do The Huxley, foi necessário entender como utilizar suas rotas. Portanto, foi necessário realizar um trabalho manual de análise do envio de requisições feitas através do site do The Huxley e documentar suas rotas. Essa avaliação foi realizada utilizando a funcionalidade de "Inspecionar", presente nos navegadores, que permite analisar as requisições feitas pelo web site.

Para obtenção dessas rotas, foi analisado e compreendido o funcionamento das seguintes operações: autenticação de usuário, obtenção das informações do usuário, obtenção da lista de submissões, obtenção da lista de problemas, obtenção de uma única submissão, obtenção do código-fonte pertencente a uma submissão, obtenção de um único problema e filtragens de submissões. Com essa análise, foram obtidas as rotas descritas no quadro 1.

Quadro 1: Rotas da API do The Huxley

| **EndPoint** | **Descrição** |
|---|---|
| `www.thehuxley.com/api/login` | Rota responsável por autenticar o usuário e retornar o token de autenticação. |
| `www.thehuxley.com/api/v1/problems` | Rota para obter a lista de problemas disponíveis. |
| `www.thehuxley.com/api/v1/problems /idProblema` | Rota para obter os dados do problema com o id passado. |
| `www.thehuxley.com/api/v1/user/submissions` | Rota para obter a lista de submissões do usuário. |
| `www.thehuxley.com/api/v1/user/submissions /idSubmissao` | Rota para obter os dados da submissão com o id passado. |
| `www.thehuxley.com/api/v1/user/submissions /idSubmissao/sourcecode` | Rota para obter o código-fonte da submissão com o id passado. |
| `www.thehuxley.com/api/v1/user` | Rota para obter informações do usuário. |
| `www.thehuxley.com/api/v1/submissions/ user/idUsuario/stats` | Rota para obter estatísticas das submissões do usuário. |

Fonte: Autor (2023)

Além das rotas, foram mapeados campos de auxílio que podem ser utilizados nas rotas, como exibido no quadro 2.

Quadro 2: Campos auxiliares para as rotas da API da Friendly Message

| **Campos** | **Descrição** |
|---|---|
| `problemName` | Campo utilizado para indicar uma busca por problemas usando o nome. Exemplo: `problemName=tabuada`. O caractere % indica um espaço vazio. |
| `max` | Campo utilizado para especificar a quantidade máxima de itens a serem retornados em uma requisição. Exemplo: `max=10`. No máximo, 10 registros serão retornados. |
| `offset` | Campo utilizado para indicar a partir de qual item os resultados devem ser retornados. Exemplo: `offset=20`. Serão retornados registros a partir do índice 20. |
| `order` | Campo utilizado para indicar a ordem de classificação dos resultados (ascendente ou decrescente). Exemplo: `order=desc`. Irá retornar os dados ordenados de maneira decrescente. |
| `sort` | Campo utilizado para ordenar os resultados com base em alguma métrica, como datas. Exemplo: `sort=submissionDate`. Irá ordenar os registros pela data de submissão. |
| `evaluations` | Campo utilizado para especificar o tipo de avaliações a serem consideradas, podendo ser "RUNTIME_ERROR"ou "COMPILATION_ERROR". Exemplo: `evaluations=RUNTIME_ERROR`. Somente submissões com erro de compilação serão retornadas. |
| `submissionDateGe` | Campo utilizado para filtrar os resultados com base na data de submissão, indicando uma data de início. Exemplo: `submissionDateGe=2023-11-12T00:00:00-03:00`. Seu formato consta de `yyyy/mm/ddTHH:MM:SS:-03:00`. |
| `submissionDateLe` | Campo utilizado para filtrar os resultados com base na data de submissão, indicando uma data de término. Exemplo: `submissionDateLe=2023-11-12T00:00:00-03:00`. Seu formato consta de `yyyy/mm/ddTHH:MM:SS:-03:00`. |
| `problem` | Campo utilizado para especificar o ID de um problema específico ao buscar informações detalhadas sobre ele. Exemplo: `problem=1`. O problema de ID 1 será buscado ou utilizado como fonte de dados. |

Fonte: Autor (2023)

### 2.3.2 Autenticação no The Huxley

Outro ponto crucial a ser destacado é a necessidade de compreender o processo de autenticação da *API* do The Huxley, uma vez que é vital para garantir o acesso e a obtenção dos dados de cada usuário. Dessa forma, descreveremos o processo de autenticação abaixo.

O The Huxley possui um sistema de autenticação para o uso de seus serviços, tanto no site quanto na *API*. A autenticação requer um nome de usuário ou e-mail e senha. Para acessar os *endpoints* da *API*, é necessário autenticar-se em cada solicitação, o que nos levou a investigar o processo de autenticação do The Huxley.

Ao analisar o processo de autenticação, foi observado que, ao fornecer um nome de usuário/e-mail e senha corretos, o servidor de autenticação do The Huxley gera um *token Bearer*. Esse *token* é vital para a abordagem de autenticação. Ele contém informações sobre o usuário, como sua identidade e, em alguns casos, suas permissões.

O processo de autenticação com o *token Bearer* [4] envolve várias etapas. Após a validação das informações pela *API*, a geração do *token* é realizada. Em seguida, o *token* é enviado ao cliente como parte da resposta de autenticação. É crucial armazenar esse *token* de forma segura, uma vez que ele contém informações sensíveis sobre o usuário. O cliente, a aplicação FM, por sua vez, deve incluir o *token Bearer* em cada solicitação *HTTPS*, adicionando-o ao cabeçalho "*Authorization*"no formato "Authorization: Bearer [seu-token-aqui]". Nesse contexto, para garantir a segurança do *token*, foi adotado o uso do *sessionStorage* para armazenamento do *token* durante a sessão do navegador, e a comunicação segura via HTTPS foi implementada para proteger a transmissão do *token* entre o cliente e o servidor.

Com o levantamento de todos os *endpoints* para obter as informações necessárias e o sistema de autenticação para poder acessá-las, temos mapeado o suficiente para a construção da *API* do Friendly Message.

## 2.4 Explain My Error

A *API REST* desenvolvida por Galileu em seu trabalho de mestrado recebe o nome de "Explain My Error- EME (JESUS, 2018). Através de uma requisição *POST*, é possível enviar para essa *API* a mensagem de erro retornada pelo The Huxley, nos casos de submissão incorreta, utilizando o formato *JSON*.

As requisições mencionadas anteriormente utilizam o HTTP, que é um protocolo de comunicação amplamente utilizado na internet para transferir dados entre dispositivos. Existem diferentes tipos de requisições, como *GET*, *POST* e *PUT*, que são usadas para diferentes finalidades, como obter dados, enviar dados ou atualizar recursos existentes, respectivamente.

[4] Token Bearer: <https://apidog.com/articles/what-is-bearer-token/>

Requisições são pedidos enviados por um cliente a um servidor para que este execute alguma ação ou retorne alguma informação.

Além disso, para enviar os dados nas requisições, é possível utilizar diferentes formatos, como *XML*, *CSV* e *JSON*. No caso mencionado, a requisição enviada ao servidor utilizou o formato *JSON*, que é um formato simples e eficiente para representar dados estruturados em texto.

No retorno da requisição *POST* realizada à *API* EME, caso a mensagem enviada esteja em conformidade com o modelo esperado, como explicado em (JESUS, 2018), a mensagem de erro convertida é retornada pela requisição, caso contrário, uma mensagem informando que a mensagem de erro está fora do padrão é retornada.

### 2.4.1 Utilizando a *API*

Para exemplificar do funcionamento da *API*, utilizaremos um exercício qualquer disponível no The Huxley para o qual foi feita uma submissão incorreta. Dessa maneira conseguiremos obter sua mensagem de erro para conversão.

Para podermos realizar as requisições para a *API*, que está hospedada em localhost para essa exemplificação, utilizaremos uma ferramenta chamada *Postman*.[5]

O *Postman* é uma aplicação que auxilia o desenvolvimento de *APIs*, permitindo os desenvolvedores criar, testar e documentar suas *APIs* de forma mais eficiente. Com ela é possível realizar requisições a *APIs* sem a necessidade de uma interface de usuário, facilitando portanto o desenvolvimento.

Para testes foi criada uma solução incorreta onde a variável *"count"* foi escrita de maneira incorreta na função *"print"* do Python 3, uma das versões utilizadas pelo The Huxley. A mensagem de erro gerada pode ser vista na Figura 3.

Para realizarmos uma requisição *POST* no endereço da *API*, é necessário que o corpo da mensagem *JSON* possua os seguintes campos: "errorMsg"e "extension". Os campos fazem referência a mensagem de erro retornada pelo compilador/interpretador e a extensão que a linguagem utiliza para identificação, respectivamente.

Utilizando nossa mensagem de erro de exemplo, retornada de um código escrito em Python 3, inclui-se a mensagem de erro em uma requisição no formato *JSON*, conforme a Figura 4. Logo em seguida temos a resposta da requisição com a mensagem amigável apresentada na Figura 5.

[5] <https://www.postman.com/>

Figura 4 – Mensagem de erro contida na requisição *POST*


```
{
    "errorMsg": "Traceback (most recent call last):\nFile \"/1000.py\",
                 line 10, in <module> \nprint(f\"{n} appeared {cout}
                 times\")\nNameError: name 'cout' is not defined",
    "extension": ".py"
}
```


Fonte: Autor (2023)

Figura 5 – Retorno da requisição *POST* contendo mensagem amigável


```
{
    "data": [
        {
            "eme_msg": "\nErro na linha 10\nNo trecho de código:\nprint(f\"{n}
                       appeared {cout} times\")\nDescrição: Foi feito o uso de
                       uma variável que não foi definida ou um comando que foi
                       escrito de forma errada. Verifique a palavra 'cout'.\n",
            "stackoverflow_links": [
                null
            ]
        }
    ]
}
```


Fonte: Autor (2023)

É retornada uma mensagem *JSON* contendo um array de objetos "data" em que cada objeto é composto por "eme_msg" e "stackoverflow_links", contendo a mensagem amigável para o usuário e os links de apoio para indicar fontes de estudo visando ajudar o aluno na resolução do erro, respectivamente.

O campo referente aos links de apoio, a serem obtidos do StackOverflow, veio vazio pois trata-se de uma proposta de continuidade feita por Galileu (JESUS, 2018), que não foi implementada pelo mesmo.

Com esse exemplo pode ser notada a grande utilidade da ferramenta no processo de aprendizado, uma vez que a mensagem complexa e em inglês retornada pelo compilador foi convertida em uma mensagem amigável ao usuário e em português. Além disso, a arquitetura utilizada permite que a *API REST* seja integrada a outras ferramentas de forma fácil, que será justamente um dos focos deste trabalho de conclusão de curso.

Nesse projeto, a *API* EME sofreu um processo de extensão, no qual, além do retorno da mensagem de erro convertida em uma mensagem amigável, são também retornados links de apoio para a correção. Esses links são obtidos a partir da *API* do StackOverflow[6]. Detalhes sobre essa extensão estão descritos na seção 5.5.

[6] StackOverflow: <https://stackoverflow.com/>

## 2.5 Desenvolvimento de software

De acordo com Birrell e Ould (1988), o desenvolvimento de software é um processo pelo qual é planejado e desenvolvido um sistema computacional, ou seja, transformar a necessidade de um utilizador em um produto de software.

No desenvolvimento de software, existe a Engenharia de Software (ES). Segundo (PRESSMAN, 2016), a ES é dividida em camadas, sendo elas: Processo, Métodos e Ferramentas. A junção dessas três camadas tem como objetivo possibilitar a criação de softwares de qualidade.

De acordo com o (PRESSMAN, 2016), ainda, o Processo constitui a camada que realiza o alinhamento entre as outras, servindo como a base para o gerenciamento de projetos de software, estabelecendo métodos e ferramentas a serem aplicadas.

Os Métodos fornecem as informações técnicas para desenvolver software. Os métodos envolvem uma variedade de tarefas, como análise de requisitos, modelagem de projeto, desenvolvimento do software e testes. Na camada de Métodos da ES surgiu a Engenharia de Requisitos (ER), que se trata do processo de levantar, analisar e documentar requisitos de um software. "A engenharia de requisitos constrói uma ponte entre o projeto e a construção..."(PRESSMAN, 2016, pg.132).

As Ferramentas são instrumentos utilizados para apoiar as atividades de desenvolvimento de software. Elas incluem linguagens de programação, banco de dados, bibliotecas, *frameworks*, *IDEs*, sistemas de gerenciamento de versão, sistemas de hospedagem entre outras. Elas tem como objetivo aumentar a eficiência e a qualidade na criação de código.

## 2.6 Método Ágil

O Método ágil é um conjunto de metodologias utilizadas no desenvolvimento de software. Essas metodologias compartilham de um único objetivo: permitir um processo de desenvolvimento mais flexível e adaptativo, capaz de lidar com as incertezas e mudanças que ocorrem durante o desenvolvimento do software.

Para permitir essa flexibilização e adaptabilidade no processo, a grande maioria das metodologias constam em dividir o trabalho a ser realizado em interações de entrega. Dessa maneira, caso algum problema ou mudança ocorra, é possível fazer ajustes e correções mais rapidamente, sem afetar o prazo final da entrega do projeto. A entrega a ser feita nessas interações pode variar de acordo com a metodologia escolhida e o tamanho da interação.

Métodos Ágeis enfatizam comunicações em tempo real entre a equipe e os *stakeholders*, colaboração entre os membros da equipe, auto-organização da equipe e a entrega incremental e iterativa de funcionalidades do produto (BECK et al., 2001).

# 3

# Especificação de Requisitos, Casos de Uso e Arquitetura

Neste capítulo, apresentaremos em detalhes o projeto da ferramenta, detalhando, na seção 3.1, os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF). A seção 3.2 abordará o Diagrama de Casos de Uso, acompanhado da Descrição dos Casos de Uso. Por fim, na seção 3.3, exploraremos a arquitetura tanto do projeto quanto da aplicação.

## 3.1 Requisitos

De acordo com o (SOMMERVILLE, 2019), os requisitos de um software fazem parte da ER, comentada no capítulo 2, e são divididos em Requisitos Funcionais e os Requisitos Não Funcionais.

### 3.1.1 Requisitos Funcionais

Os requisitos funcionais (RF) representam as funções que o software deve realizar. Com base nas análises realizadas a partir de reuniões com o *stakeholder*, Dr. Alberto Costa Neto, orientador do trabalho, que é usuário do TH há 8 anos, os RF listados no Quadro 3 foram elicitados.

Quadro 3: Requisitos Funcionais

| **Id** | **Título** | **Descrição** |
|---|---|---|
| RF001 | Autenticar | Disponibilizar uma operação de autenticação para o usuário através do The Huxley. |
| RF002 | Acessar submissões incorretas | Acessar submissões incorretas do usuário no The Huxley, permitindo filtragem e/ou ordenação com as mesmas opções disponíveis pelo TH. |
| RF003 | Exibir detalhes da submissão | O sistema deve exibir as informações contidas na submissão selecionada. |
| RF004 | Exibir detalhes de um caso de teste | O sistema deve exibir a mensagem de erro original e os valores de entrada e saída referentes ao caso de teste selecionado. |
| RF005 | Apresentar mensagem de erro amigável e links de auxílio | O sistema deve apresentar ao usuário a mensagem de erro amigável juntamente aos links de auxílio. |

### 3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais (RNF), mostrados no Quadro 4 representam características que o software deve atender. Com base nas análises realizadas a partir da pesquisa com os *stakeholders* e no mercado, os seguintes RNFs foram elicitados:

Quadro 4: Requisitos Não Funcionais

| **Id** | **Título** | **Descrição** |
|---|---|---|
| RNF001 | Ambiente | O sistema deverá funcionar no ambiente Web |
| RNF002 | Disponibilidade | O sistema deverá estar disponível 24 horas por dia, 7 dias por semana, desde que o The Huxley e a *API* EME estejam online |
| RNF003 | Nível de acesso | Somente usuários do The Huxley, docentes e alunos, poderão ter acesso ao sistema |
| RNF004 | Responsividade | O sistema deve ser responsivo independentemente da tela |

Fonte: Autor (2023)

## 3.2 Casos de Uso da Aplicação

Após a identificação dos principais requisitos da aplicação foram definidos os Casos de Uso.

### 3.2.1 Descrição dos Casos de Uso em Nível de Usuário

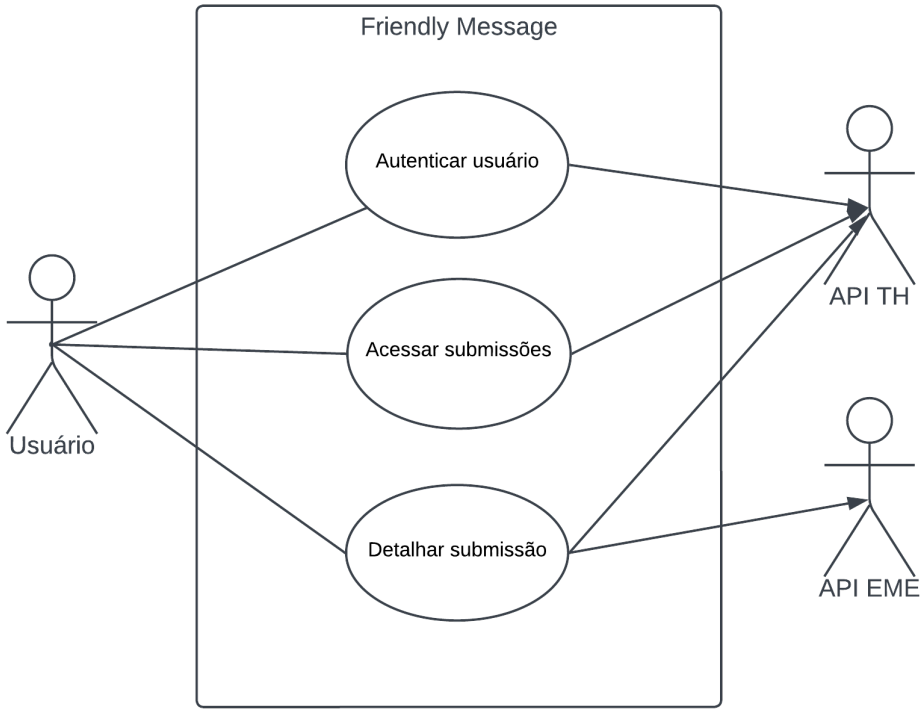
Nesta seção, tanto os atores quanto cada caso de uso do sistema serão descritos, visando promover um maior entendimento das funções da aplicação. O Quadro 5 traz o objetivo e a descrição do Caso de Uso de Autenticação. Já os Quadros 6 e 7 apresentam os Casos de Uso referentes às submissões e à mensagem amigável recebida pela *API REST* EME.

Os atores do sistema são: Usuário, *API* TH e *API* EME. O usuário pode ser qualquer utilizador do The Huxley, docente, discente ou outros. O ator *API* TH refere-se à *API* do Juiz

online The Huxley. O ator *API* EME refere-se à *API* do projeto Explain My Error, desenvolvido por Galileu em (JESUS, 2018).

A Figura 6 apresenta um diagrama dos Casos de Uso para facilitar o entendimento.

Figura 6 – Diagrama de Casos de Uso




Fonte: Autor (2023)

Quadro 5: Descrição do Caso de Uso Autenticar

| CSU01 - Autenticação | |
|---|---|
| **Objetivo** | Confirmar a identidade do usuário |
| **Descrição** | Para utilizar o sistema, o usuário deve utilizar suas informações de login (usuário/e-mail e senha) do The Huxley, desde que, antes disso, tenha feito seu cadastro no The Huxley. |

Fonte: Autor (2023)

Quadro 6: Descrição do Caso de Uso Acessar submissões

| CSU02 – Acessar submissões | |
|---|---|
| **Objetivo** | Listar e filtrar submissões |
| **Descrição** | O usuário poderá listar e filtrar suas submissões incorretas utilizando os mesmos critérios disponíveis para consultar as submissões realizadas no The Huxley. |

Fonte: Autor (2023)

Quadro 7: Descrição do Caso de Uso: Detalhar Submissões

| CSU03 – Detalhar Submissões | |
|---|---|
| **Objetivo** | Exibir casos de teste, código-fonte e mensagem de erro associadas a um caso de teste específico e sua mensagem amigável. |
| **Descrição** | O usuário, por meio da listagem de submissões, pode selecionar uma submissão específica para visualizar seus detalhes. Ao detalhar uma submissão, é possível obter a mensagem de erro retornado pelo juiz online, em cada caso de teste, bem como a mensagem amigável correspondente, juntamente com links de apoio. |

Fonte: Autor (2023)

## 3.3 Arquitetura

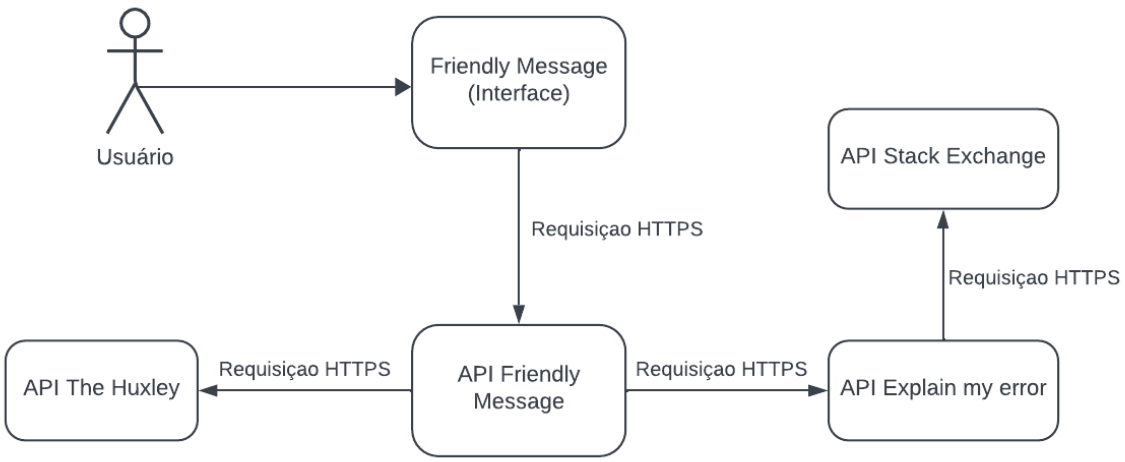
Nesta seção, será apresentada a arquitetura utilizada no sistema. A Subseção 3.3.1 abordará a arquitetura da aplicação, descrevendo como o FM se comunica com as *APIs* externas, TH e EME, enquanto a Subseção 3.3.2 focará na estrutura do projeto Friendly Message, incluindo sua interface web e *API*.

### 3.3.1 Arquitetura da Aplicação

O Friendly Message é uma aplicação que integra uma interface web para a interação do usuário e uma *API* para a obtenção de dados. A *API* do FM faz uso de duas outras *APIs*: a do The Huxley e a Explain my Error. A *API* do TH, discutida no Capítulo 2, é responsável por obter informações das submissões do usuário no The Huxley, enquanto a EME, também discutida no Capítulo 2, converte a mensagem de erro original em uma mensagem amigável e, além disso, obtém links de suporte de fóruns do *Stack Overflow* para facilitar a resolução do problema pelo estudante. A EME realiza essa obtenção de links por meio da *API* do *Stack Overflow*, a Stack Exchange.

A arquitetura escolhida para a aplicação é cliente-servidor (SOMMERVILLE, 2019, 160). Nessa abordagem, a interface web do FM age como cliente, enviando solicitações *HTTPS* à *API* do FM, que busca as informações necessárias por meio de requisições *HTTPS* às outras *APIs*, TH e EME. Essas *APIs* seguem uma arquitetura *REST* e utilizam requisições *HTTPS* como meio de comunicação, conforme visualizado na Figura 7.

Figura 7 – Diagrama de arquitetura da aplicação




Fonte: Autor (2023)

A *API* do FM atua como intermediária entre as *APIs* do TH e EME, obtendo as informações da *API* do TH e as retornando à interface, além de enviá-las para a EME realizar a conversão e obtenção dos links. A EME também possui uma dependência externa, utilizando a *API* do *Stack Overflow* para obter os links. Dessa forma, ao receber uma solicitação da *API* do FM, ela realiza a conversão da mensagem e faz uma solicitação na *API* do *Stack Overflow*, que a retorna para a EME e, em seguida, retorna para a do FM, que finalmente a retorna para a interface do usuário.

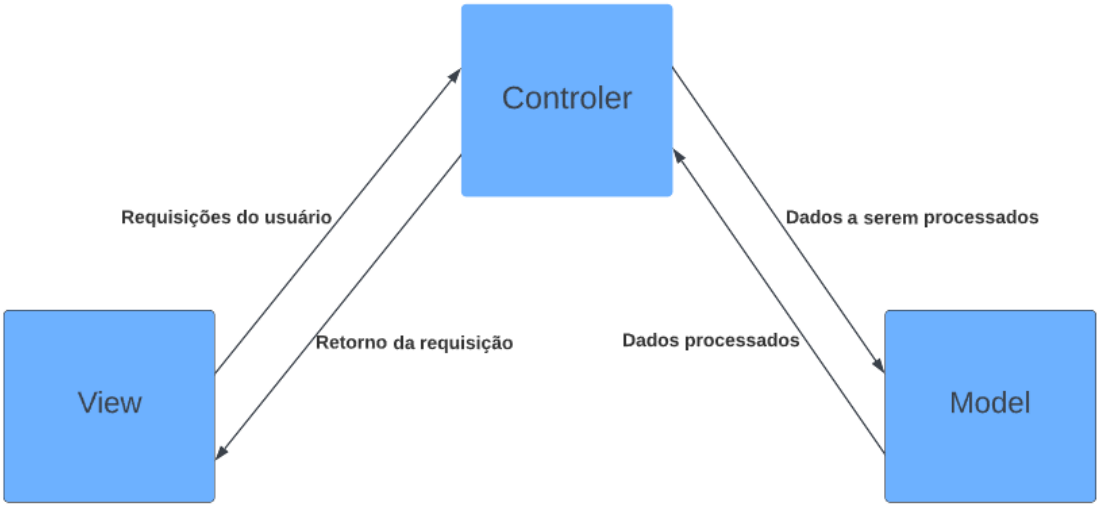
Dessa maneira, a *API* do FM atua como meio de comunicação e integração entre as duas *APIs* e a interface web.

### 3.3.2 Arquitetura do Friendly Message

O projeto Friendly Message utiliza a arquitetura *Model-View-Controller (MVC)* (SOMMERVILLE, 2019, 155). O *MVC* é uma estrutura que separa a lógica de negócios, a interface do usuário e as manipulações de dados em três camadas: *Model*, *View* e *Controller*.

A camada *Model* é responsável por representar os dados, as regras de negócios da aplicação e manipular os dados para fornecer ao *Controller*. A camada *View* é responsável pela interface do usuário, como uma página web, e também conecta o usuário ao *Controller*. O *Controller* gerencia a interação do usuário com a aplicação, realizando tratamentos/validações nos dados antes de enviá-los ao *Model*. Além disso, é responsável por receber os dados do *Model* e enviá-los para a *View*, conforme demonstrado na Figura 8.

Figura 8 – Diagrama de arquitetura em camadas do projeto




Fonte: Autor (2023)

Transpondo essa estrutura para o contexto do Friendly Message, a camada de *View* seria representada pela interface web construída, enquanto as camadas de *Model* e *Controller* estão presentes na *API* FM.

# 4
# Ferramentas

Neste capítulo, iremos elucidar o conjunto de ferramentas necessário para o desenvolvimento da aplicação. O objetivo deste capítulo é explicar cada uma delas e o motivo de sua utilização.

## 4.1 React

O React[1] é uma biblioteca JavaScript desenvolvida pelo Facebook utilizada para construir interfaces para aplicações Web e Mobile. O React surgiu como uma maneira eficiente de atualizar a interface de usuário em aplicações web dinâmicas a partir da mudança de estado da aplicação.

Além dessa atualização da interface a partir do estado da aplicação, o React também é conhecido por seu recurso de reutilização de código. Esse conceito é baseado na ideia de componentes, em que os componentes são os blocos de construção da interface. Um componente pode ser uma função ou uma classe que define como uma parte específica da interface deve ser exibida.

Essas características presentes no React, além de ser código aberto e possuir uma boa documentação disponibilizada pelo próprio Facebook, foram os motivos que levaram à sua escolha para compor as ferramentas de desenvolvimento desse projeto de conclusão de curso. Graças à sua flexibilidade e eficiência na atualização da interface pelo estado, a aplicação contará com uma interface mais composta para o usuário. Juntamente com essa grande flexibilização da interface, a nível de implementação, teremos um código composto por componentes que são extensíveis, reutilizáveis, simples de manter e testar.

[1] React: <https://react.dev/>

## 4.2 Axios

O [2] é uma biblioteca JavaScript amplamente usada para simplificar requisições *HTTPS* ou *HTTPS* em projetos web. Sua flexibilidade e simplicidade o tornam popular entre desenvolvedores que buscam eficiência em interações assíncronas com servidores.

Conhecido pela confiabilidade e facilidade de uso, o Axios suporta métodos *HTTPS* como *GET*, *POST* e *PUT*, oferecendo diversas opções para comunicação entre o frontend e o backend de uma aplicação.

Destaca-se por sua sintaxe clara e concisa, tornando o código mais legível. Além disso, possui recursos avançados como cancelamento de requisições, tratamento automático de respostas *JSON* e a capacidade de incluir cabeçalhos personalizados.

Mantido ativamente pela comunidade de desenvolvedores, o Axios recebe atualizações regulares com melhorias e correções de *bugs*. Sua documentação abrangente e a presença de uma comunidade ativa contribuem para a confiabilidade e sucesso dessa biblioteca no desenvolvimento web.

## 4.3 ASP.NET Core

O ASP.NET[3] Core é um *framework* de desenvolvimento que estende o ambiente .NET, sendo de código aberto e multiplataforma. Ele permite a criação de aplicativos web escaláveis, serviços e *APIs*, utilizando linguagens como C#, VB.NET e F#. No projeto em questão, a linguagem utilizada é o C#.

O ASP.NET Core tem como principais características sua alta performance, eficiência, alta modularidade e segurança. Outro fator que justifica sua escolha é sua ampla documentação, comunidade ativa e suporte da Microsoft. Além disso, o padrão arquitetural *Model-View-Controller (MVC)* é amplamente suportado pelo ASP.NET Core, o que o torna uma escolha favorável para o projeto. Essa integração proporciona uma estrutura organizada e eficiente para o desenvolvimento, alinhada com as boas práticas do *MVC*.

Atualizado regularmente, o ASP.NET Core recebe novos recursos e melhorias com frequência, garantindo que os desenvolvedores tenham acesso às tecnologias mais recentes e avançadas para criar aplicativos modernos e eficientes.

[2] Axios: <https://axios-http.com/docs/intro>
[3] ASP.NET Core: <https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core>

## 4.4 Microsoft Visual Studio

O Microsoft Visual Studio[4] trata-se de um ambiente integrado de desenvolvimento para criação de software em diversas plataformas, incluindo a Web.

Ele contém uma ampla variedade de recursos e ferramentas que auxiliam no desenvolvimento, simplificando-o. Algumas dessas ferramentas são: depuração, recursos de autocompletar, gerenciamento de testes, instalação e gerenciamento de pacotes, integração com ferramentas de versionamento, como o Git, compatibilidade com várias linguagens, incluindo C#, C++, VB.NET, F#, Python e JavaScript, entre outras diversas funcionalidades.

Além disso, sua comunidade ativa, suporte da Microsoft e documentação são outros fatores que justificam a escolha do Visual Studio como *IDE*.

## 4.5 Bitbucket

O Bitbucket[5] é uma plataforma de hospedagem de repositórios Git, utilizada por desenvolvedores para gerenciar o código fonte de seus projetos. Ele é desenvolvido e mantido pela Atlassian, a mesma empresa por trás de outras ferramentas como Jira[6].

O Bitbucket oferece recursos para gerenciamento de repositórios, como a criação de *branches*, realização de *pull requests*, revisão de código e integração com outras ferramentas de desenvolvimento, como o Jira.

A plataforma possui planos pagos que oferecem uma maior quantidade de funcionalidades para gerenciamento dos repositórios, mas também oferece planos gratuitos com funcionalidades reduzidas.

O Bitbucket é uma ferramenta completa para hospedagem de repositórios de código, assim como o GitHub, mas ela foi escolhida por, principalmente, uma questão de continuidade, onde repositórios de projetos anteriores que visam o auxílio no aprendizado de programação, como a *API* EME, (JESUS, 2018), estão hospedados nela. Por essa questão, o BitBucket foi a ferramenta escolhida.

## 4.6 Heroku

O Heroku[7] é uma aplicação que permite hospedagem e configuração. Entre outras funções, ele simplifica a configuração da infraestrutura para a implantação das aplicações, mais conhecido como *deploy* da aplicação.

[4] Microsoft Visual Studio: <https://visualstudio.microsoft.com/>
[5] Bitbucket: <https://bitbucket.org/>
[6] Jira: <https://www.atlassian.com/software/jira>
[7] Heroku: <https://www.heroku.com/>

Além dessas características, a que mais se destaca é a capacidade de fazer *deploy* de maneira contínua a cada modificação realizada em seu sistema de controle de versionamento de código-fonte, como o BitBucket.

O Heroku é uma ferramenta paga, entretanto, disponibiliza uma conta de estudante com um valor de US$100 em créditos para utilização ao realizar um cadastro na aplicação com sua conta institucional do DCOMP/UFS. Dados esses motivos, o Heroku foi a escolha como serviço de hospedagem.

# 5

# Desenvolvimento da Ferramenta

Neste capítulo, descreveremos a construção da *API* do Friendly Message, responsável por realizar a intermediação entre o The Huxley e a *API* EME, além do processo de construção da interface web da aplicação Friendly Message e a extensão realizada na *API* EME.

## 5.1 Desenvolvimento da *API* do Friendly Message

A arquitetura da aplicação destaca a importância da *API* FM como elemento fundamental, desempenhando um papel central na obtenção de informações do The Huxley e na apresentação desses dados na interface web, bem como na transformação de mensagens de erro em mensagens amigáveis com o auxílio da *API* EME. Desta forma, a *API* FM representa o ponto de convergência entre o The Huxley, o EME e a interface web, servindo como a camada de acesso utilizada pelo usuário.

A *API* FM utiliza duas fontes de dados: a *API* do The Huxley e a *API* EME. Portanto, o desafio inicial consistiu em desenvolver uma abordagem eficaz e organizada para obter informações dessas duas *APIs*. Para resolver esse desafio, implementou-se o padrão de projeto *Repository*.

O padrão *Repository* [1] é uma abordagem de design que se concentra na separação das operações de acesso a dados e consulta do código de negócios da aplicação. Ele visa criar uma camada intermediária (o repositório) entre o código de negócios e as fontes de dados, neste caso, as *APIs* do The Huxley e do EME.

Ao implementar o padrão *Repository*, é criada uma camada que abstrai os detalhes da interação com as fontes de dados. Isso significa que o código de negócios da aplicação não precisa se preocupar com os detalhes de como fazer chamadas de *API*, autenticar, lidar com

[1] Padrão Repository: <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>

erros ou quaisquer outras questões relacionadas a chamadas à *API*. Tudo isso é tratado pelo repositório.

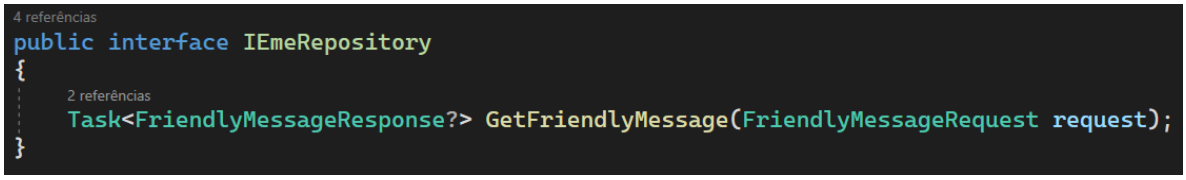
O padrão *Repository* melhora a organização e manutenção do código, encapsulando operações de dados e tornando-o mais legível e coeso. Além disso, torna a aplicação flexível, permitindo futuras mudanças na fonte de dados, como migrar de uma *API* para um banco de dados, afetando apenas a camada de repositório e minimizando o impacto nas outras partes do código.

O padrão *Repository* foi a melhor escolha para a construção de uma *API* que necessita acessar várias fontes de dados, tais como as *APIs* do The Huxley e do EME. As figuras 9 e 10 ilustram a interface do *repository* do TH e EME, respectivamente.

Figura 9 – Interface do repositório da *API* TH


Fonte: Autor (2023)

Figura 10 – Interface do repositório da *API* EME




Fonte: Autor (2023)

A implementação de cada método dos repositórios pormenoriza a comunicação com suas respectivas *APIs*. No repositório do TH todas possuem a mesma estrutura, dessa forma, será exemplificado um deles. O método `GetSubmissionsById` será utilizado.

```
1 public async Task<Submission?> GetSubmissionsById(int id, HttpRequest
      httpRequest)
2 {
3     var requestUrlSubmission = $"{UrlTheHuxley}v1/submissions/{id}";
4     var request = new HttpRequestMessage(HttpMethod.Get,
          requestUrlSubmission);
```

```
5       request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("
            application/json"));
6       if (httpRequest.Headers.TryGetValue("Authorization", out var header))
7       {
8           var authHeaderValue = header.ToString();
9           var token = authHeaderValue.Replace("Bearer ", "");
10          request.Headers.Authorization = new AuthenticationHeaderValue("
                Bearer", token);
11      }
12      var response = await _httpClient.SendAsync(request);
13      if (response.IsSuccessStatusCode)
14      {
15          var responseBody = await response.Content.ReadAsStringAsync();
16          var submission = JsonConvert.DeserializeObject<Submission>(
                responseBody);
17          return submission;
18      }
19      var errorHandled = ErrorHandlingService.HandleTHError(response.
            StatusCode);
20      throw errorHandled;
21  }
```

Listing 5.1 – Código fonte em C-Sharp

O método `GetSubmissionsById` é um dos métodos dentro do repositório que realiza a comunicação com a *API* do TH para obtenção dos dados. O método tem o propósito de obter informações específicas sobre uma submissão.

O método constrói a *URL* da solicitação, na linha 3, configura a solicitação *HTTPS* nas linhas 4 e 5, lida com a autorização se um cabeçalho de autorização estiver presente na solicitação original, da linha 6 até a 11. Em seguida, envia a solicitação de forma assíncrona, na linha 12, e lida com a resposta, deserializando-a em um objeto *`Submission`*, se a resposta for bem-sucedida, da linha 13 até 18. Em caso de erro, o serviço *`ErrorHandlingService`* é chamado para lidar com o erro com base no código de status retornado pela *API* TH, nas linhas 19 e 20.

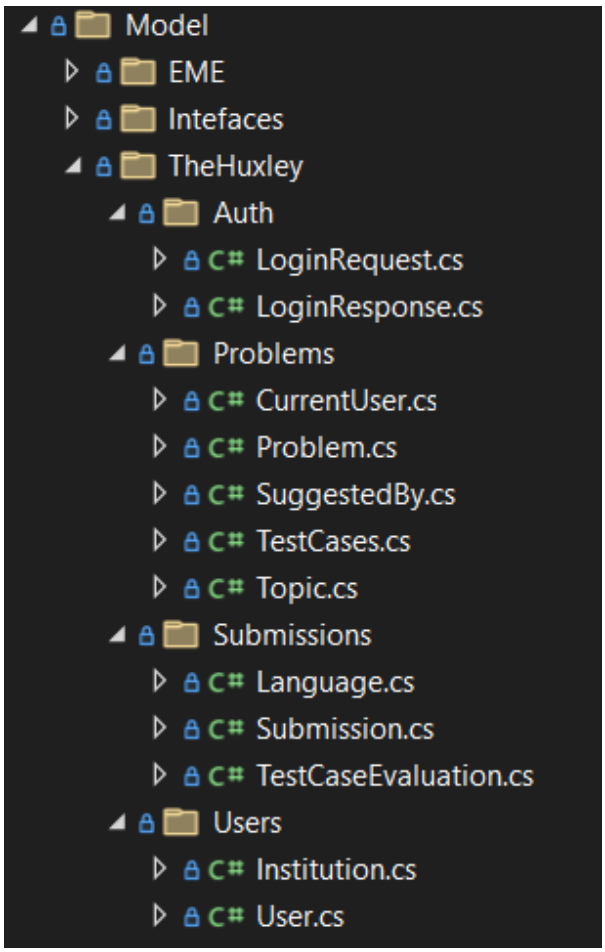
Essa estrutura é aplicada a todas as operações de acesso às *APIs* do The Huxley e do EME, mantendo um código consistente e organizado para a obtenção de informações das fontes de dados externas.

## 5.2 Camada de Modelo

Como mencionado anteriormente, a camada de modelo foi construída com base na análise dos *JSONs* retornados pelas *APIs*. Dado que o C# é uma linguagem fortemente tipada, é necessário criar classes de modelo que correspondam à estrutura do *JSON* para que seja possível

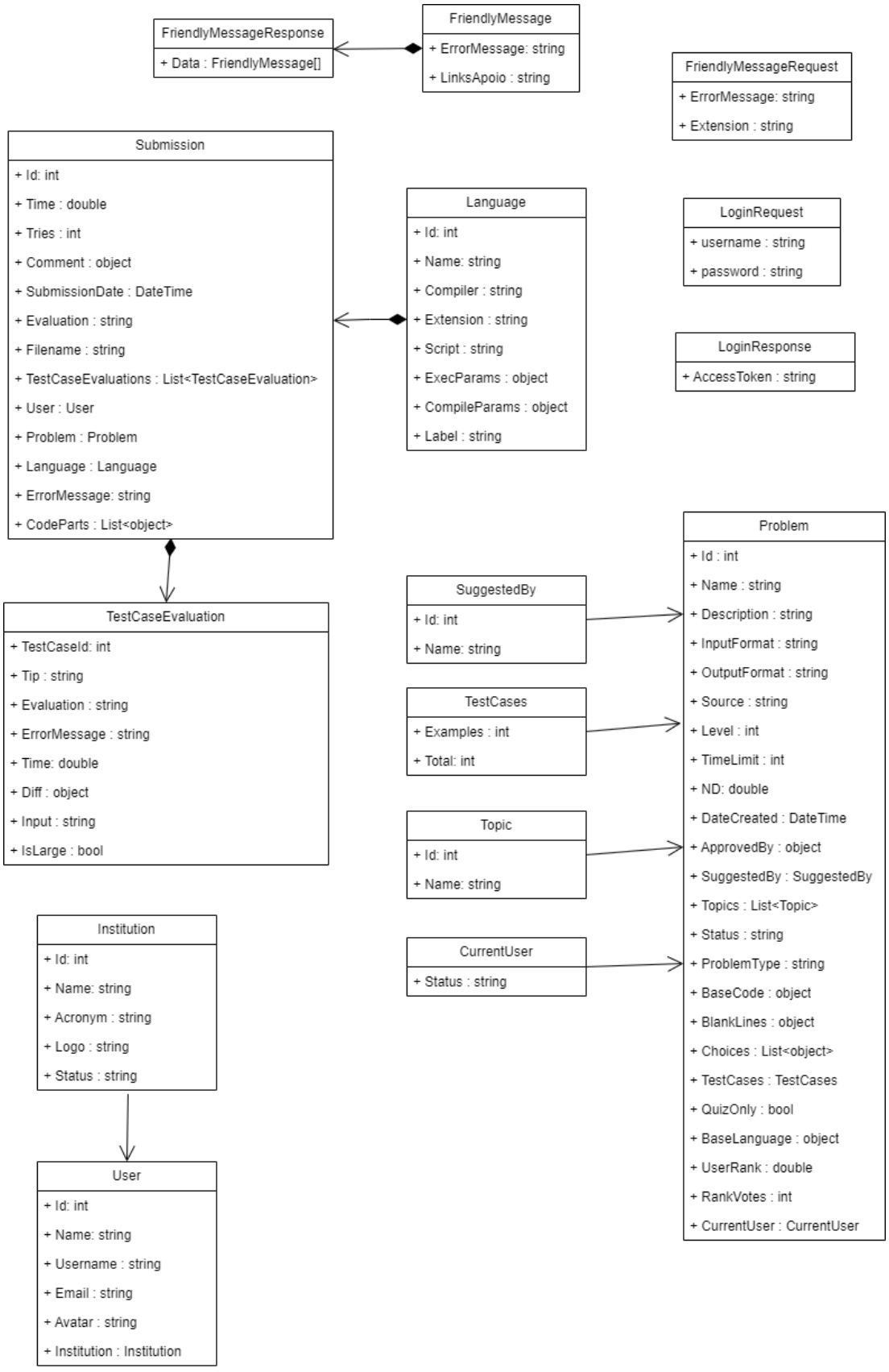
deserializá-lo em um objeto. Essa conversão permite o tratamento dos *JSONs* como objetos na linguagem, possibilitando manipulações mais flexíveis. A camada de Modelo é apresentada na Figura 11 e no diagrama de classes na Figura 12.

Figura 11 – Estrutura da Camada de Modelo




Fonte: Autor (2023)

Figura 12 – Diagrama de classes da Camada de Modelo




Fonte: Autor (2023)

## 5.3 Geração de Exceções para Gerenciamento de Erros

Dentro da *API* FM, existe uma funcionalidade de conversão de erros que desempenha um papel crucial na gestão de situações inesperadas ou excepcionais durante a execução da aplicação. Isso envolve a manipulação de erros provenientes das *APIs* externas, como o The Huxley e o EME. O objetivo é fornecer mensagens de exceção mais claras e concisas aos usuários, visando aprimorar a experiência do sistema.

### 5.3.1 The Huxley - Geração de Exceções

O serviço de conversão oferece recursos para gerar exceções em resposta a erros provenientes da *API* The Huxley. Quando ocorre um erro durante a comunicação com o The Huxley, o serviço não realiza uma intervenção direta; em vez disso, gera uma nova exceção com uma mensagem mais simples. Os principais tipos de erro gerados são:

- **Erro de Autenticação:** Se o código de status indicar falha na autenticação do usuário, o serviço de conversão de erros gerará uma nova exceção `HttpRequestException` com uma mensagem apropriada para orientar o usuário sobre como resolver o problema.
- **Erro Interno no Servidor:** Em caso de erro interno no servidor do The Huxley, o serviço de conversão de erros criará uma exceção `HttpRequestException` indicando um problema de comunicação e sugerirá que o usuário tente novamente.
- **Erro de Não Encontrado:** Se a *API* The Huxley não for encontrada, o serviço de conversão de erros gerará uma exceção `HttpRequestException`, informando ao usuário que o servidor não pôde ser localizado e sugerindo uma nova tentativa.
- **Erros Genéricos:** Para outros erros não específicos, o serviço de conversão de erros gerará uma exceção `HttpRequestException` com uma mensagem genérica, indicando que ocorreu um erro não especificado ao obter dados da *API* The Huxley.

### 5.3.2 EME - Geração de Exceções

Além disso, também são oferecidos recursos para a geração de exceções em resposta a erros provenientes da *API* EME. Os erros tratados incluem:

- **Erro Interno no Servidor:** Semelhante ao processo de The Huxley, em caso de erro interno no servidor do EME, o serviço de conversão de erros gera uma exceção `HttpRequestException` indicando problemas de comunicação e sugerindo que o usuário tente novamente.
- **Erro de Não Encontrado:** Quando a *API* EME não é encontrada, o serviço de conversão de erros gera uma exceção `HttpRequestException` informando ao usuário que não foi possível encontrar o servidor e sugere uma nova tentativa.

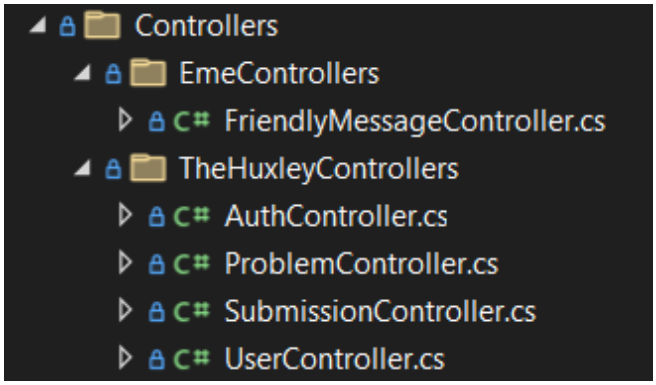
- **Erros Genéricos:** Para outros erros não específicos, o serviço de conversão de erros gera uma exceção `HttpRequestException` com uma mensagem genérica, indicando que ocorreu um erro não especificado ao obter dados da *API* EME.

Essa abordagem permite que a aplicação forneça feedback claro e compreensível aos usuários em caso de problemas, promovendo uma experiência mais intuitiva.

## 5.4 Controladores

Os controladores desempenham um papel fundamental como ponto de entrada da *API* FM, uma vez que contêm os métodos executáveis por meio de solicitações. Nessa estrutura, os controladores foram organizados conforme as classes de modelo correspondentes, como ilustrado na Figura 13.

Figura 13 – Estrutura dos Controladores da *API*




Fonte: Autor (2023)

Cada classe de modelo principal possui sua própria controladora, a qual utiliza os repositórios necessários para acessar e manipular os dados. Para a integração com a *API* do The Huxley, foram implementadas as controladoras: *AuthController*, *ProblemController*, *SubmissionController* e *UserController*. Para a integração com a *API* EME, foi criada a controladora *FriendlyMessageController*.

Com a implementação das controladoras, é possível acessar um conjunto de endpoints listados no quadro 8 para utilizar a *API* Friendly Message por meio da interface web.

Quadro 8: Rotas da *API* do Friendly Message

| **EndPoint** | **Descrição** |
|---|---|
| `api/auth` | Rota responsável por autenticar o usuário e retornar o token de autenticação. |
| `api/problems` | Rota para obter a lista de problemas disponíveis. |
| `api/problems/id` | Rota para obter detalhes de um problema específico com o ID correspondente. |
| `api/submissions` | Rota para obter a lista de submissões. |
| `api/submissions/id` | Rota para obter detalhes de uma submissão com o ID correspondente. |
| `api/user` | Rota para obter informações do usuário. |
| `api/user/stats` | Rota para obter estatísticas sobre as submissões do usuário. |
| `api/friendlyMessage` | Rota para obter mensagens amigáveis da API EME. |

Fonte: Autor (2023)

## 5.5 Atualização da *API* EME

Como já mencionado, a *API* EME é responsável por receber a mensagem de erro retornada pelo compilador/interpretador. Dessa forma, ela é um elemento fundamental para o desenvolvimento do Friendly Message, que consiste em uma interface de utilização para os estudantes, conectando suas submissões no The Huxley com a *API* EME e permitindo a conversão das mensagens de erro, auxiliando no aprendizado.

Ao iniciar o desenvolvimento do Friendly Message, buscamos criar um ambiente de auxílio completo ao estudante. Além de obter uma mensagem amigável para o erro encontrado, o estudante receberia também maneiras de apoiar a solução. A EME deveria disponibilizar, além da conversão da mensagem de erro, a possibilidade de obter links de apoio para a solução. Em outras palavras, a partir da mensagem de erro enviada, links de apoio para a resolução do problema deveriam ser retornados juntamente com a mensagem. Entretanto, conforme documentado por (JESUS, 2018), a implementação dessa funcionalidade não foi concluída, sendo sugerida como trabalho futuro.

Dessa forma, para criar um ambiente de auxílio completo, foi necessário realizar ajustes na *API* EME para finalizar a implementação da busca de links de apoio para a solução do problema. Para realizar essa implementação, foi necessário estudar e entender como a *API* funciona. Esse entendimento foi obtido através do estudo da dissertação de mestrado de Galileu (JESUS, 2018) e análise da implementação realizada.

Após essa estudo, foi percebido que a análise poderia ser realizada pela obtenção de palavras-chave na mensagem de erro. Conforme mencionado no documento, os principais erros de cada linguagem foram mapeados e foram definidas palavras-chave que os identificam. Após isso, foi criada uma classe para análise de cada possibilidade, resultando em um método de análise para cada possível erro na linguagem. Para gerenciar essa implementação, visto que

todos os métodos deveriam ser chamados, foi utilizado o padrão *Chain of Responsibility*[2] para coordenar a execução da análise pelos vários métodos.

Para ser possível encontrar links de apoio, era necessário obter as palavras-chave que identificariam os erros. Assim, no algoritmo que implementa o padrão de projeto *Chain of Responsibility*, foi adicionada uma variável para que, quando um método identificasse que existia uma palavra-chave na mensagem, essa palavra-chave fosse guardada em uma variável. Essa variável seria utilizada em um método de busca em seguida.

Com a palavra-chave obtida, deu-se início à construção do processo de busca. Para limitar a busca e trazer resultados mais certeiros, em vez de utilizar as ferramentas de busca como: Google[3] e Bing[4], foi decidido utilizar o StackOverflow para buscar fóruns que possuíssem a palavra-chave como título e que também possuíssem a linguagem como *tag* identificadora. Dessa forma, teríamos a possibilidade de retornar ao usuário um link de apoio para a solução de forma mais precisa.

Para obter os links para os fóruns do StackOverflow, foi utilizada a sua *API* pública StackExchange[5]. A *API* possui uma documentação onde foi possível identificar a maneira de realizar as requisições pelos fóruns. Assim, nesse momento, temos as palavras-chaves que identificam o erro e uma *API* que realiza a busca no maior fórum de computação do mundo.

Com essas informações, deu-se início ao processo de desenvolvimento da requisição responsável por obter os links de apoio. Como já mencionado, a implementação da funcionalidade tinha sido iniciada, mas não concluída. Portanto, o algoritmo responsável por obter a mensagem amigável já continha a chamada ao método de busca pelos links. Assim, foi necessário apenas realizar a sua implementação.

```
1  def busca_stackoverflow(error_msg, linguagem):
2      try:
3          url = "https://api.stackexchange.com/2.3/search"
4          parametros = {
5              "order": "desc",
6              "sort": "relevance",
7              "intitle": error_msg,
8              "site": "stackoverflow",
9              "tagged" : linguagem,
10             "pagesize" : 10
11         }
12         resposta = requests.get(url, params=parametros)
13         if resposta.status_code == 200:
14             data = resposta.json()
15             if "items" in data:
```

[2] Padrão Chain of Responsibility: <https://refactoring.guru/design-patterns/chain-of-responsibility>
[3] Google: <https://www.google.com/>
[4] Bing: <https://www.bing.com/>
[5] StackExchange: <https://api.stackexchange.com/>

```
16                  itens_da_linguagem = []
17                  for item in data["items"]:
18                      link = item["link"]
19                      titulo = item["title"]
20                      item_formatado = {"Forum": {"Titulo": titulo, "Link":
                            link}}
21                      itens_da_linguagem.append(item_formatado)
22
23                  itens_da_linguagem.append({"Stack overflow": "https://pt.
                        stackoverflow.com/"})
24                  return json.dumps(itens_da_linguagem)
25              else:
26                  resposta.status = 404
27                  return json.dumps({"message": "Links de apoio n\~{a}o
                        encontrados."})
28          else:
29              resposta.status = resposta.status_code
30              return json.dumps({"message": "Falha ao obter links de apoio do
                    stackoverflow."})
31      except Exception as ex:
32          resposta.status = 400
33          return json.dumps({"message": "Erro: " + str(ex)})
```

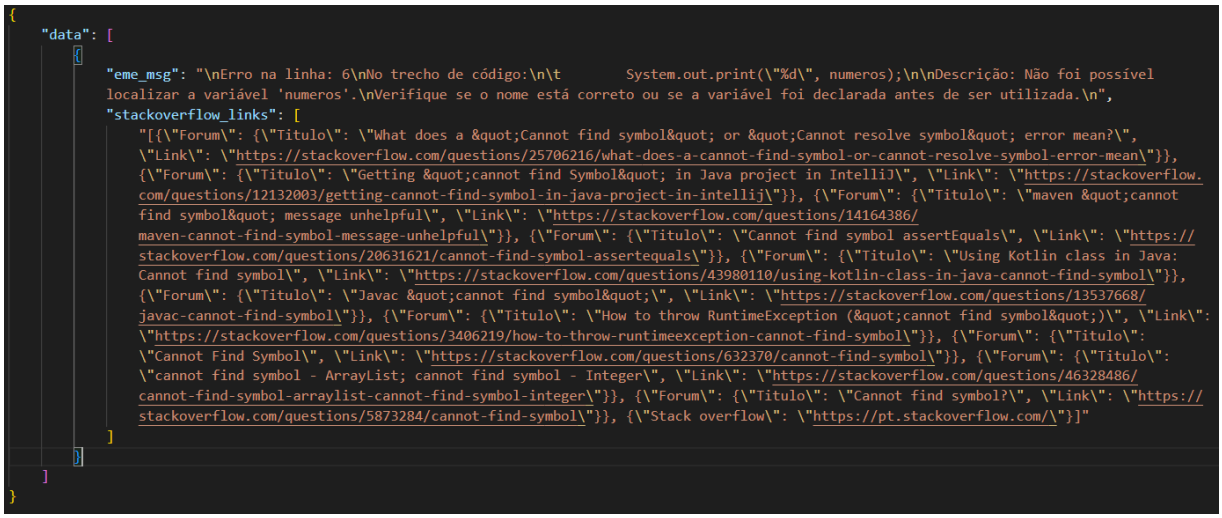
Listing 5.2 – Código fonte em Python

A função `busca_stackoverflow` recebe dois parâmetros: `error_msg` (mensagem de erro) e `linguagem` (linguagem de programação). Ela busca no Stack Overflow links de perguntas que contenham a `error_msg` no título e estão marcadas com a linguagem especificada.

O procedimento começa com a definição da *URL* da *API* do Stack Exchange e dos parâmetros da busca, incluindo a ordenação por relevância e a restrição aos títulos das perguntas, da linha 3 até a linha 11. A função então envia uma requisição *GET* com esses parâmetros para a *API*, na linha 12.

Se a resposta da *API* for bem-sucedida, os dados são convertidos de *JSON* para um formato Python, nas linhas 13 e 14. A função verifica se há itens na resposta, ou seja, se foram encontradas perguntas relacionadas à mensagem de erro, na linha 15. Se encontradas, ela processa os dados para obter o título e o link de cada pergunta. Essas informações são organizadas em um formato padronizado e devolvidas como uma resposta em *JSON*, da linha 16 até a 25, como ilustrado na Figura 14.

Figura 14 – Retorno *API* EME com links de apoio



```
{
    "data": [
        {
            "eme_msg": "\nErro na linha: 6\nNo trecho de código:\n\t        System.out.print(\"%d\", numeros);\n\nDescrição: Não foi possível
            localizar a variável 'numeros'.\nVerifique se o nome está correto ou se a variável foi declarada antes de ser utilizada.\n",
            "stackoverflow_links": [
                "[{\"Forum\": {\"Titulo\": \"What does a &quot;Cannot find symbol&quot; or &quot;Cannot resolve symbol&quot; error mean?\",
                \"Link\": \"https://stackoverflow.com/questions/25706216/what-does-a-cannot-find-symbol-or-cannot-resolve-symbol-error-mean\"}},
                {\"Forum\": {\"Titulo\": \"Getting &quot;cannot find Symbol&quot; in Java project in IntelliJ\", \"Link\": \"https://stackoverflow.
                com/questions/12132003/getting-cannot-find-symbol-in-java-project-in-intellij\"}}, {\"Forum\": {\"Titulo\": \"maven &quot;cannot
                find symbol&quot; message unhelpful\", \"Link\": \"https://stackoverflow.com/questions/14164386/
                maven-cannot-find-symbol-message-unhelpful\"}}, {\"Forum\": {\"Titulo\": \"Cannot find symbol assertEquals\", \"Link\": \"https://
                stackoverflow.com/questions/20631621/cannot-find-symbol-assertequals\"}}, {\"Forum\": {\"Titulo\": \"Using Kotlin class in Java:
                Cannot find symbol\", \"Link\": \"https://stackoverflow.com/questions/43980110/using-kotlin-class-in-java-cannot-find-symbol\"}},
                {\"Forum\": {\"Titulo\": \"Javac &quot;cannot find symbol&quot;\", \"Link\": \"https://stackoverflow.com/questions/13537668/
                javac-cannot-find-symbol\"}}, {\"Forum\": {\"Titulo\": \"How to throw RuntimeException (&quot;cannot find symbol&quot;)\", \"Link\":
                \"https://stackoverflow.com/questions/3406219/how-to-throw-runtimeexception-cannot-find-symbol\"}}, {\"Forum\": {\"Titulo\":
                \"Cannot Find Symbol\", \"Link\": \"https://stackoverflow.com/questions/632370/cannot-find-symbol\"}}, {\"Forum\": {\"Titulo\":
                \"cannot find symbol - ArrayList; cannot find symbol - Integer\", \"Link\": \"https://stackoverflow.com/questions/46328486/
                cannot-find-symbol-arraylist-cannot-find-symbol-integer\"}}, {\"Forum\": {\"Titulo\": \"Cannot find symbol?\", \"Link\": \"https://
                stackoverflow.com/questions/5873284/cannot-find-symbol\"}}, {\"Stack overflow\": \"https://pt.stackoverflow.com/\"}]"
            ]
        }
    ]
}
```


Fonte: Autor (2023)

Se a busca não retornar itens, a função retorna uma mensagem indicando que não foram encontrados links de apoio para a mensagem de erro fornecida, nas linhas 26 e 27. Se a requisição à *API* falhar por algum motivo, a função retorna uma mensagem indicando que houve uma falha ao tentar obter os links de apoio do Stack Overflow, nas linhas 29 e 30. Em caso de exceções (por exemplo, erros de execução), a função retorna uma mensagem de erro indicando a natureza da exceção ocorrida, na linha 32.

Essencialmente, essa função é uma interface para acessar a *API* do Stack Exchange e buscar links de perguntas relacionadas a uma mensagem de erro e uma linguagem de programação específica no Stack Overflow, proporcionando assim apoio aos usuários na resolução de problemas de programação.

## 5.6 Construção da interface web

Nesta seção, será apresentado o processo de construção da interface web da aplicação Friendly Message. A interface web desempenha um papel crucial, servindo como a principal ponte de comunicação entre os usuários e a aplicação. Ela foi desenvolvida utilizando a biblioteca React[6] com JavaScript[7], conhecida por sua capacidade de criar interfaces de usuário dinâmicas e interativas.

A escolha do React como a base para nossa interface foi motivada por sua ampla aceitação na indústria de desenvolvimento de aplicativos web. O React simplifica o desenvolvimento de interfaces de usuário escaláveis e de alto desempenho, permitindo-nos fornecer uma experiência agradável aos usuários da Friendly Message.

[6] React: <https://legacy.reactjs.org/docs/getting-started.html>
[7] JavaScript: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

Dessa forma, demos início à construção dos componentes que estarão presentes nas telas da aplicação. Os componentes foram construídos visando a reutilização, um dos principais motivos que levaram à escolha do React. Dessa forma, a maioria deles possui o comportamento geral esperado, que pode ser modificado conforme a necessidade.

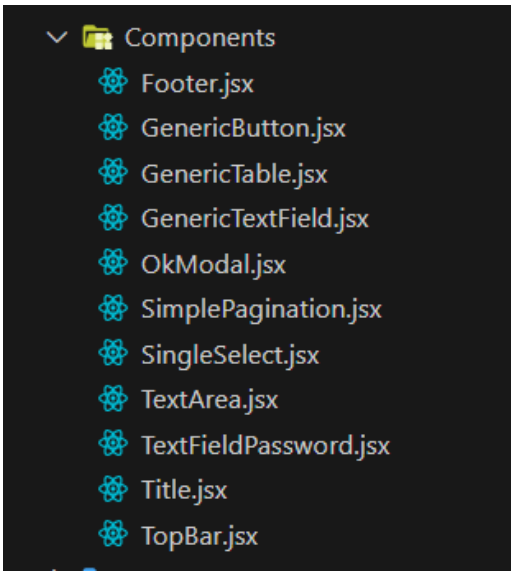
Para exemplificar isso, observe o código abaixo. Ele trata-se de um componente de botão genérico da interface, que possui o comando ao ser clicado, a cor e o texto customizáveis. Esse componente é utilizado em diversas partes da aplicação, como na tela de login e na listagem de submissões incorretas.

```
import React from 'react';
import Button from '@mui/material/Button/Button';

function GenericButton(props) {
  const { command, text, color } = props;
  return (
    <Button
      variant="contained"
      style={{ backgroundColor: color }}
      onClick={() => {
        command();
      }}
    >
      {text}
    </Button>
  );
}

export default GenericButton;
```

Listing 5.3 – Código fonte em JavaScript

O componente possui propriedades que são passadas em sua utilização. Essas propriedades tornam possível a flexibilidade de utilização do componente. Elas controlam a funcionalidade executada ao clique com a propriedade *'command'*, o texto exibido no botão com a propriedade *'text'* e a cor com a propriedade *'color'*. Essas propriedades tornam o componente flexível, sendo possível reutilizá-lo em várias partes do sistema de maneiras diferentes e sem a necessidade de duplicação de código. Todos os componentes implementados e utilizados seguem essa abordagem de possuir propriedades configuráveis por quem irá utilizá-los. Na Figura 15 estão os componentes implementados para a interface.

Figura 15 – Serviços de comunicação da interface web




Fonte: Autor (2023)

Nas seções a seguir, será abordado a integração com a *API* Friendly Message e as telas da aplicação, que utilizam os componentes.

### 5.6.1 Integração com a *API* do Friendly Message

A integração com a *API* é uma etapa fundamental na construção da interface da Friendly Message, pois é nesse ponto que ocorre a comunicação entre a aplicação e o servidor. Para realizar essa integração, foi adotada a biblioteca Axios[8].

O Axios é uma biblioteca JavaScript que facilita a realização de requisições *HTTPS* de forma assíncrona. Ele desempenha um papel crucial na obtenção e envio de dados entre a aplicação e a *API* do Friendly Message. Essa integração é vital para garantir que as informações sejam atualizadas em tempo real e que os usuários tenham acesso às funcionalidades da aplicação de maneira eficiente.

O código apresentado abaixo faz parte do aplicativo Friendly Message e tem como objetivo fazer solicitações *HTTPS* para a *API*. Essas solicitações são realizadas usando a biblioteca Axios. No exemplo apresentado, o foco está na interação com as submissões da *API* por meio da classe *'Submission'*.

```
1 import axios from 'axios';
2 class Submission {
3   constructor(token) {
4     this.token = token;
5     this.url = `${process.env.REACT_APP_URLBACK}/api/submissions`;
```

[8] Axios: <https://axios-http.com/docs/intro>

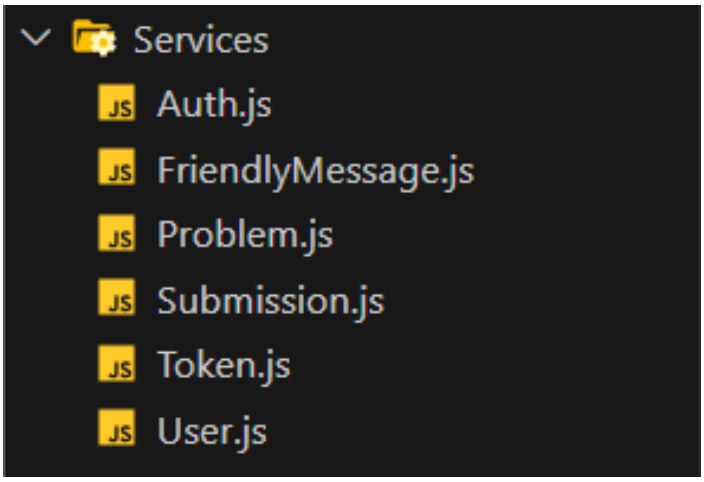
```
6       this.config = {
7         headers: {
8           'Content-Type': 'application/json',
9           'Accept': 'application/json',
10          'Authorization': `Bearer ${this.token}`
11        }
12      };
13    }
14
15    async obterTodos() {
16      const requestUrl = `${this.url}`;
17      return axios.get(requestUrl, this.config);
18    }
19  }
```

Listing 5.4 – Código fonte em React

No código acima, para realizar uma solicitação na *API* do Friendly Message, três propriedades essenciais são definidas na classe *'Submission'*. A primeira é *'token'*, na linha 4, que armazena o *token* de autorização necessário para acessar a *API* de forma segura. A propriedade *'url'* contém a *URL* base da *API*, para onde as solicitações serão direcionadas, na linha 5. A propriedade 'config' contém as configurações das solicitações, incluindo o tipo de conteúdo (*JSON*), a aceitação do formato *JSON* e a inclusão do *token* de autorização por meio do cabeçalho *'Authorization'*, da linha 6 até a 13.

O método 'obterTodos', da linha 15 até a 18, é utilizado para buscar todas as submissões na *API* FM. Ele constrói a URL de requisição combinando a URL base da *API* com a rota específica das submissões, na linha 16. Em seguida, utiliza o Axios para fazer uma solicitação *GET* para essa URL, na linha 17, incluindo as configurações definidas na propriedade 'config'. Como resultado, a função retornará os dados das submissões a partir da *API* FM. Essa estrutura é aplicada a todas as operações de comunicação com a *API* do Friendly Message, realizadas pelas classes na camada de serviço da aplicação, ilustradas na Figura 16.

Figura 16 – Serviços de comunicação da interface web



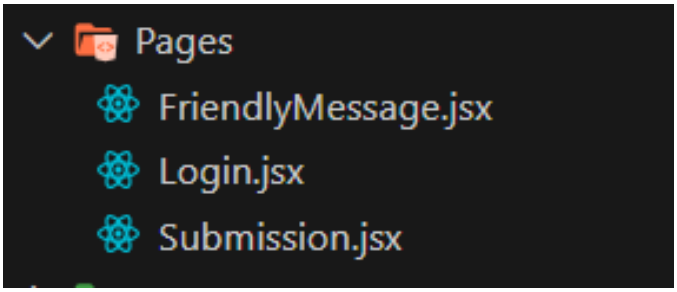

Fonte: Autor (2023)

### 5.6.2 Telas da aplicação

Nesta seção, serão apresentadas as telas desenvolvidas para a aplicação Friendly Message. A interface de usuário é um dos aspectos mais importantes de qualquer aplicativo, e essas telas foram criadas com o objetivo de fornecer uma visão geral das funcionalidades do sistema. As telas foram projetadas visando a simplicidade e praticidade de utilização.

Foram implementadas três páginas, sendo que cada uma utiliza um conjunto de componentes e serviços para sua construção. Cada página será descrita nas seções a seguir.

Figura 17 – Páginas da interface web

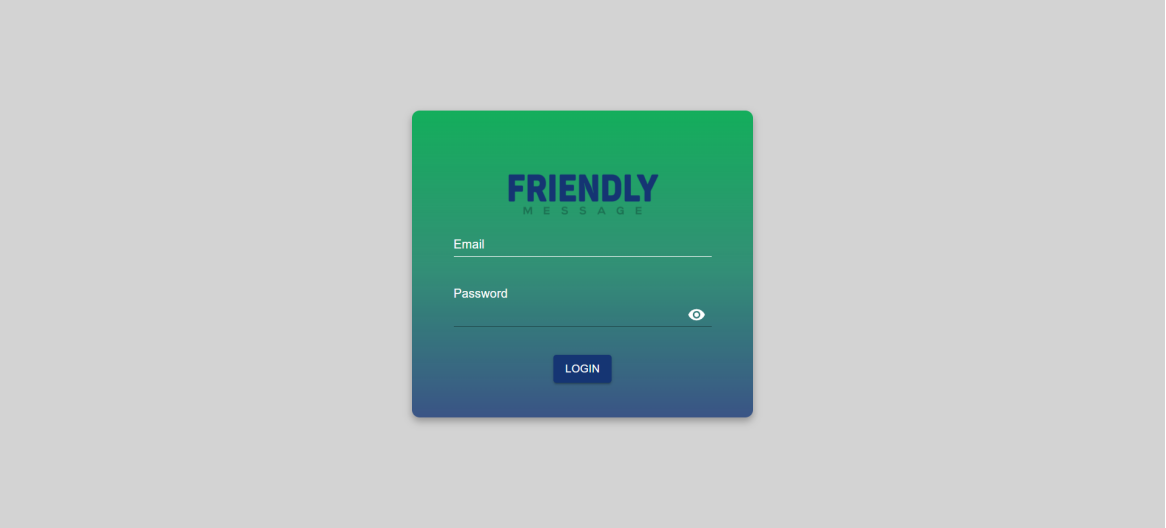



Fonte: Autor (2023)

#### 5.6.2.1 Tela de login

A tela de login, Figura 18 será a primeira tela vista pelo usuário ao acessar a aplicação. Ela é responsável por coletar as informações do usuário e realizar a autenticação do mesmo. Nessa tela, temos o nome da aplicação, campos para o preenchimento do *username* e senha do usuário no The Huxley, e um botão para realizar a autenticação.

Figura 18 – Tela de login



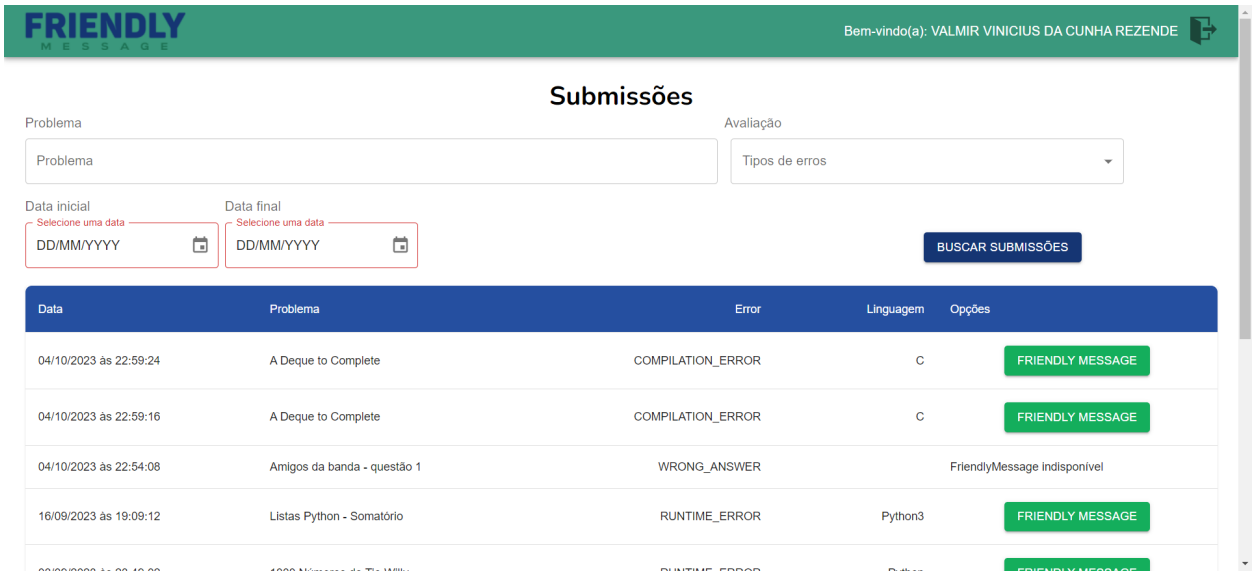

Fonte: Autor (2023)

#### 5.6.2.2 Tela de submissões incorretas

A tela de submissões incorretas (Figura 19), será a segunda tela vista pelo usuário ao acessar a aplicação e fazer login com as informações do The Huxley. Ela é responsável por listar todas as submissões incorretas realizadas pelo usuário no The Huxley. Nessa tela, temos opções de filtragem para as submissões, considerando o problema, data e tipo de erro. Esses filtros serão acumulativos, ou seja, caso um problema seja selecionado e uma data for definida, a listagem representará as submissões do problema selecionado na data informada.

Na listagem, além das informações de filtragem já mencionadas, temos o botão "Friendly Message", que tem como função exibir a próxima tela com as informações da submissão.

Figura 19 – Tela de submissões incorretas




Fonte: Autor (2023)

#### 5.6.2.3 Tela de detalhes sobre a submissão

A tela de detalhes, Figura 20 sobre a submissão é a terceira tela apresentada no sistema. Ela é acessível ao clicar na opção "Friendly Message" em uma das submissões da tela de submissões incorretas. As informações contidas nessa tela são referentes à submissão selecionada anteriormente.

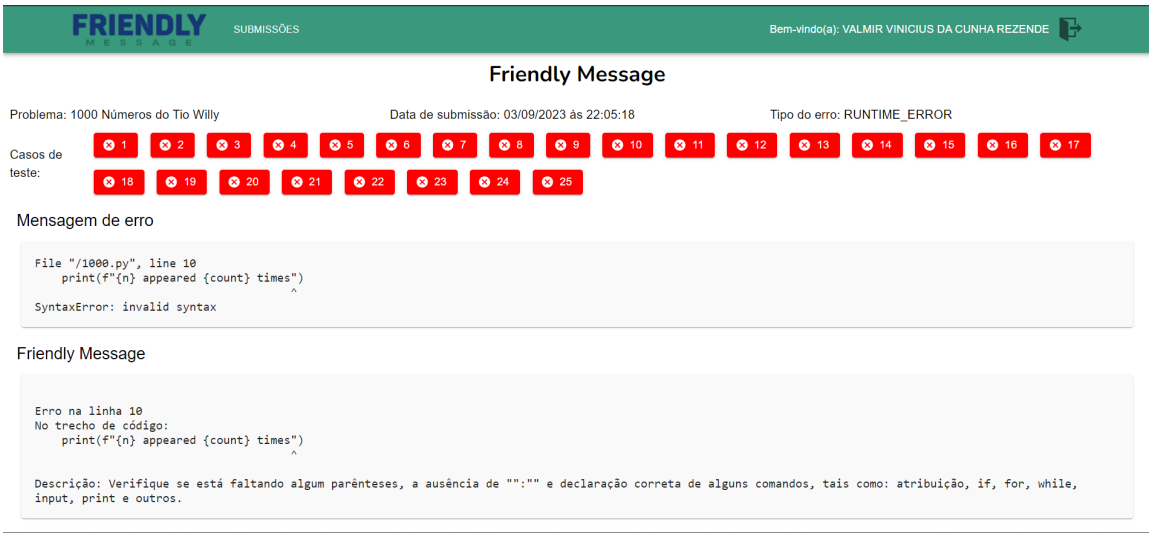
Nessa tela, informações como o nome do problema, a data de submissão, a linguagem utilizada, o tipo de erro e os casos de teste estão disponíveis. Os casos de teste, assim como no The Huxley, serão enumerados e terão uma informação visual de cor, em que o vermelho indica falha no cumprimento do caso e o verde, sucesso.

Figura 20 – Tela da submissão


Fonte: Autor (2023)

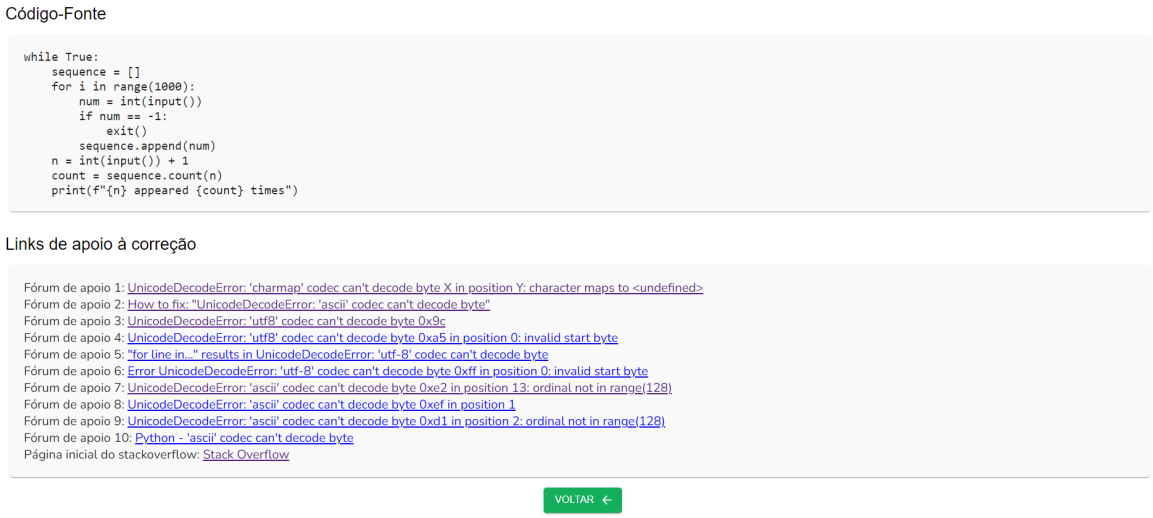
Ao clicar em um caso de uso em que houve falha (ícone vermelho), a mensagem de erro original do juiz será carregada no campo "Mensagem de erro", a mensagem amigável no campo "Friendly Message", o código-fonte no campo "Código-Fonte", enquanto os links de apoio para correção, serão carregados no campo "Links de apoio à correção". Como ilustrado nas Figuras 21 e 22, respectivamente.

Figura 21 – Tela da submissão com caso de teste selecionado




Fonte: Autor (2023)

Figura 22 – Tela da submissão com mensagem amigável e links de apoio




Fonte: Autor (2023)

# 6

# Considerações Finais e Trabalhos Futuros

Na área educacional, os estudantes tendem a se beneficiar cada vez mais com a criação de softwares educacionais. O Friendly Message vem como uma ferramenta auxiliar educacional para estudantes de programação resolverem os erros presentes no código-fonte, bem como desenvolver a capacidade analítica para identificação e reconhecimento de erros. Com a conversão para o português e a escrita amigável, o estudante começa a fazer uma associação à medida que mais erros são apresentados, criando uma base para a compreensão de próximos erros, utilizando a mensagem de origem.

Durante a realização do projeto, foram realizadas atividades fundamentais para sua concretização: levantamento da extensão a ser feita na *API* Explain my Error, elaboração dos requisitos funcionais e não funcionais da aplicação Friendly Message, definição da arquitetura a ser utilizada, definição das ferramentas de desenvolvimento a serem usadas, implementação da extensão realizada na *API* EME, estudo sobre o funcionamento da *API* TH, implementação da *API* FM e sua interface web.

A aplicação foi construída visando proporcionar uma experiência agradável e simples ao usuário, a fim de facilitar sua utilização. Paralelamente, busca oferecer qualidade em sua implementação e a possibilidade de extensão, como explicado no capítulo 3, na definição das ferramentas utilizadas. Dessa maneira, tem-se como trabalhos futuros, sendo alguns deles a criação de páginas na web que explicam as possíveis causas de erro de forma bem mais profunda, a extensão da aplicação para execução independente do The Huxley, onde um sistema de autenticação próprio pode ser criado, e a conversão de mensagens disponibilizadas a um grupo de estudantes com erros em problemas fora do ambiente do The Huxley, sejam eles em outro juiz online ou obtidos a partir da execução de programas isoladamente. Finalmente, pretende-se realizar um experimento controlado com alunos de graduação para avaliar a aplicação no contexto de aprendizes de programação.

# Referências


BECK, K. et al. Manifesto for agile software development. 2001. Citado 2 vezes nas páginas 11 e 22.

BIRRELL, N. D.; OULD, M. A. *A practical handbook for software development*. [S.l.]: Cambridge University Press, 1988. Citado na página 22.

GANDHI, S. et al. The impact of covid-19 on the technology industry. *McKinsey & Company*, 2020. Disponível em: <https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/the-impact-of-covid-19-on-the-technology-industry>. Citado na página 9.

HUXLEY, T. *Juiz Online The Huxley*. 2023. Disponível em: <https://www.thehuxley.com/>. Acesso em: 24 abr. 2023. Citado na página 12.

(INEP), I. N. de Estudos e P. E. A. T. *Censo da Educação Superior de 2021: Apresentação dos resultados*. 2021. Disponível em: <https://download.inep.gov.br/educacao_superior/censo_superior/documentos/2021/apresentacao_censo_da_educacao_superior_2021.pdf>. Citado na página 9.

JESUS, G. S. d. *Uma abordagem para auxiliar a correção de erros de programadores iniciantes*. 157 p. Dissertação (Mestrado em Ciência da Computação) — Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Sergipe, São Cristóvão, 2018. Citado 8 vezes nas páginas 10, 16, 19, 20, 21, 25, 31 e 40.

KURNIA, A.; LIM, A.; CHEANG, B. Online judge. *Computers & Education*, v. 41, p. 121–131, 08 2003. Citado na página 12.

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 11.

PRESSMAN, R. S. *Engenharia de Software: uma abordagem profissional*. 8. ed. AMGH, 2016. ISBN 9788581430744. Disponível em: <https://www.amazon.com.br/Engenharia-Software-Uma-Abordagem-Profissional/dp/8580555337>. Citado na página 22.

SILVA, C. A. F. d.; SILVA, L. D. d.; MARTINS, J. C. D. Aplicação do the huxley no ensino de programação para alunos do curso técnico em informática para internet. In: SBC. *Proceedings of SBGames*. [S.l.], 2018. p. 1309–1312. Citado na página 14.

SOMMERVILLE, I. *Engenharia De Software (Em Portuguese do Brasil)*. 10. ed. Pearson Universidades, 2019. ISBN 8543024978. Disponível em: <https://www.amazon.com.br/Engenharia-Software-Ian-Sommerville/dp/8543024978/ref=tmm_pap_swatch_0?_encoding=UTF8&qid=&sr=>. Citado 3 vezes nas páginas 23, 26 e 27.