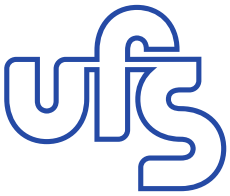


UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Aprimorando mensagens de erro geradas pela ferramenta EME

Trabalho de Conclusão de Curso

João Marcelo dos Santos Nascimento

São Cristóvão – Sergipe

2025

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

João Marcelo dos Santos Nascimento

Aprimorando mensagens de erro geradas pela ferramenta EME

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

2025

Resumo

Com o crescimento do interesse pelo desenvolvimento de software, existem cada vez mais alunos em disciplinas de programação. Esses alunos tendem a ter grandes dificuldades, mesmo com apoio dos professores de forma presencial e online, durante a prática. Por isso, os Juízes Online são ferramentas tão interessantes, já que auxiliam na prática de programação disponibilizando várias questões diferentes para os mais distintos níveis e avaliando as respostas submetidas. Mesmo com essas ferramentas, esses alunos iniciantes ainda sofrem quando precisam compreender e corrigir os erros no código-fonte submetido. Pensando nisso que a ferramenta EME foi desenvolvida por Jesus (JESUS, 2018) e, para facilitar o acesso à mesma, a ferramenta FM foi criada por Rezende (REZENDE, 2024). Porém, mesmo com essas ferramentas, os alunos iniciantes ainda sentem dificuldade em compreender os erros e entender formas de como eles podem contorná-los. Para resolver isso, o trabalho aqui proposto evoluiu estas ferramentas para que apresentem uma contextualização sobre cada classe de erro de forma que possa guiar esses estudantes para uma solução e que eles consigam compreender onde estão errando. Para a evolução da EME e FM vários passos foram necessários e resultaram em um novo conjunto de ferramentas integradas que podem ajudar ainda mais os alunos de programação.

Palavras-chave: Programação, Evolução de Software, The Huxley, Erros de programação.

Abstract

With the growing interest in software development, there are more and more students taking programming courses. These students tend to have great difficulties, even with the support of teachers in person and online, during practice. This is why Online Judges are such interesting tools, since they help in programming practice by providing several different questions for the most different levels and evaluating the submitted answers. Even with these tools, these beginner students still struggle when they need to understand and correct errors in the submitted source code. With this in mind, the EME tool was developed by Jesus (JESUS, 2018) and, to facilitate access to it, the FM tool was created by Rezende (REZENDE, 2024). However, even with these tools, beginner students still have difficulty understanding errors and understanding ways to get around them. To solve this, the work proposed here has evolved these tools so that they present a contextualization of each class of error in a way that can guide these students to a solution and help them understand where they are going wrong. For the evolution of EME and FM several steps were necessary and resulted in a new set of integrated tools that can further help programming students.

Keywords: Programming, Software Evolution, The Huxley, Programming errors.

Lista de ilustrações

Lista de quadros

Lista de abreviaturas e siglas

API	Application Progamming Interface
EAD	Ensino a Distância
EME	Explain My Error
FM	Friendly Message
DCOMP	Departamento de Computação
UFS	Universidade Federal de Sergipe
IHGSE	Instituto Histórico e Geográfico de Sergipe
TCC	Trabalho de Conclusão de Curso
HTTPS	Hyper Text Transfer Protocol Secure
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
DOM	Document Object Model
CI	Continuous Integration
CD	Continuous Delivery
AWS	Amazon Web Services
EC2	Amazon Elastic Compute Cloud
SPA	Single Page Application

Sumário

1 Introdução

Como foi apresentado em LaBerge et al. (2020), com a pandemia da COVID-19 as empresas e modelos de negócios passaram por uma aceleração no processo de digitalização e na aplicação de tecnologias nos seus processos. Isso trouxe uma grande demanda repentina para o mercado de TI e, para tentar suprí-la, vários profissionais iniciantes na área foram contratados para serem treinados. Com essa demanda, o interesse nos cursos da área de TI aumentou e o censo de 2021 INEP (2021) chegou a registrar aproximadamente 8,8% a mais de matrículas. É importante também ressaltar que a grande maioria dos alunos optaram pelo ensino a distância tendo um desbalanceamento entre o EAD e o ensino presencial, segundo informações contidas no mesmo censo INEP (2021).

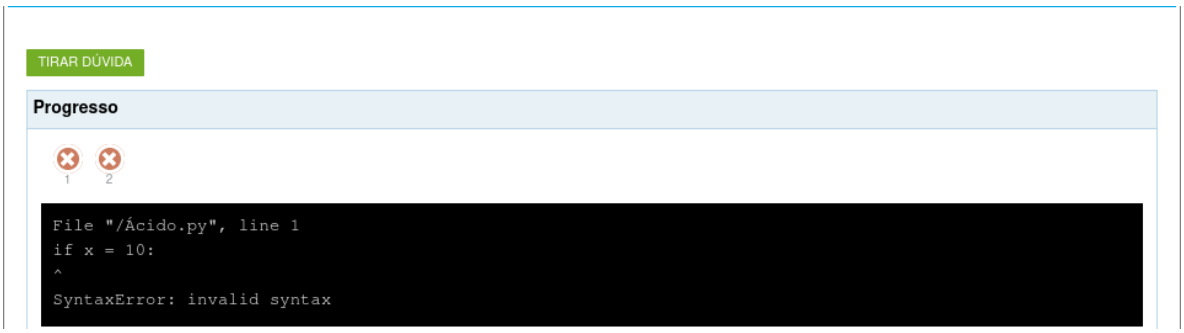
Com a crescente demanda por cursos da área de TI, um desafio em específico pode ser destacado: a dificuldade no ensino de programação (WEBER; BRUSILOVSKY; STEINLE, 2014) dada a curva de aprendizagem de novas linguagens que costuma ser crescente em iniciantes por questões como a sintaxe que pode ser rígida e a transformação da solução imaginada em lógica de programação de maneira que tudo fique bem estruturado (KELLEHER; PAUSCH, 2005).

Para ajudar os docentes no ensino de programação, existem ferramentas como os Juízes Online que funcionam como grandes repositórios de desafios onde os desenvolvedores submetem soluções em diferentes linguagens e elas são executadas em diversos casos de testes devolvendo um resultado positivo ou negativo. Essas ferramentas podem facilitar a prática do aluno e dar para ele um *feedback* mais imediato do seu progresso e também podem auxiliar o professor a monitorar a evolução dos estudantes e determinar os melhores tipos de exercícios para aplicar em diferentes turmas.

Dentre os Juízes *Online*, existe o *The Huxley* que é gratuito, produzido por brasileiros, oferece funcionalidades como a criação de turmas, estatísticas, aplicação de testes e programar diretamente na plataforma HUXLEY (2020) e é amplamente utilizado nas universidades durante as disciplinas de programação.

Mesmo com essas ferramentas para que os estudantes possam praticar programação, ainda existem barreiras que podem impedir a evolução e aprendizado. Uma delas é as mensagens de erros que são geradas pelos compiladores e apresentadas pelos Juízes *Online*, como, por exemplo, a mensagem de erro presente na Figura 1. Essas mensagens de erros normalmente estão em inglês e apresentam uma linguagem um pouco mais técnica que pode dificultar o entendimento para iniciantes.

Figura 1 – Exemplo de erro apresentado no The Huxley



Fonte: <https://thehuxley.com>

No trabalho (JESUS, 2018) foi criada uma ferramenta (*Explain My Error*) que traduz essas mensagens de erros para auxiliar os programadores iniciantes na sua compreensão. Mas, como afirmado por Rezende, essa ferramenta não é de fácil acesso por ser uma API (REZENDE, 2024) e por esse motivo, no trabalho (REZENDE, 2024) foi criada a *Friendly Message* (FM), uma estrutura para facilitar o uso da ferramenta *Explain My Error* (EME) em conjunto com a API do The Huxley, facilitando a integração desses diferentes e complementares facilitadores do ensino de programação.

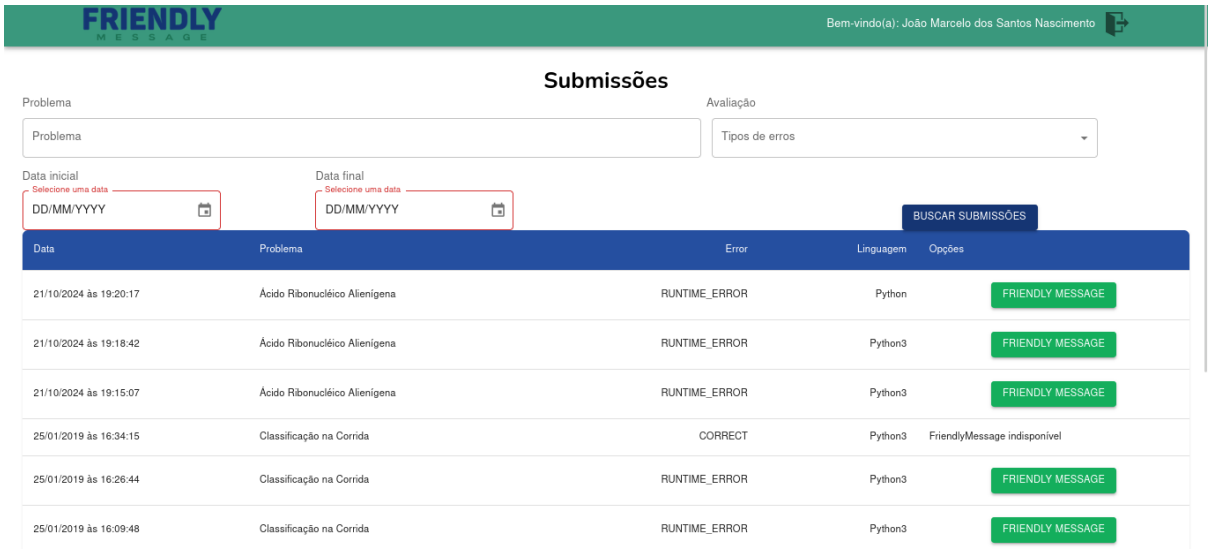
Nas Figuras 2 e 3 é possível ver um exemplo da ferramenta *Friendly Message* (FM) em execução. Ao selecionar uma submissão com erro na linguagem Python, é apresentada a versão traduzida do erro usando a API EME. Mesmo com esse conjunto de ferramentas trabalhando juntas, os alunos de programação ainda podem sentir dificuldades na compreensão do erro e em formas de contornar a situação.

1.1 Objetivos

1.1.1 Geral

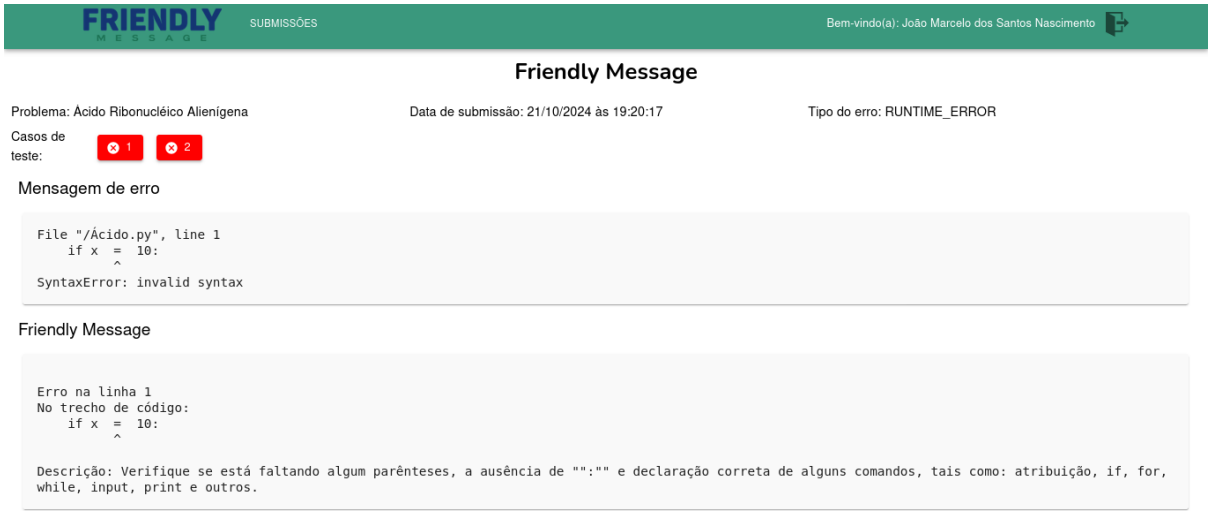
Esse trabalho teve como objetivo evoluir as integrações entre as ferramentas *Explain My Error*, *Friendly Message* e *The Huxley* de forma que para cada mensagem de erro fosse adicionado mais contexto para que os alunos consigam, de forma mais natural, encontrar onde estão errando e quais caminhos podem seguir para realizar a correção.

Figura 2 – Tela de submissões na Friendly Message



Fonte: Autor (2024)

Figura 3 – Tela de visualização da submissão na Friendly Message



Fonte: Autor (2024)

1.1.2 Específicos

Para alcançar o objetivo geral, foram criados alguns objetivos específicos para cada etapa do trabalho:

- Criar uma documentação para cada tipo de erro da linguagem Python que pode ser traduzida usando a ferramenta EME.
- Evoluir a ferramenta EME e o *Backend* da ferramenta FM para que retornem mais

informações sobre o erro traduzido.

- Evoluir o *Frontend* da ferramenta FM para apresentar de forma dinâmica e de fácil atualização uma documentação extra sobre cada mensagem de erro traduzida.

1.2 Estrutura

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, EME, FM, nos quais o presente trabalho se baseia.
- No capítulo 3, será descrita a solução proposta, a definição dos casos de uso atualizados e a arquitetura do projeto após sua evolução.
- No capítulo 4, serão apresentadas as ferramentas que foram utilizadas para o desenvolvimento.
- No capítulo 5, será abordado o processo para criar a documentação de cada classe de erro.
- No capítulo 6, teremos a apresentação do processo de desenvolvimento/evolução das ferramentas.
- Por fim, o capítulo 7 trará considerações finais e trabalhos futuros para este projeto.

2
Fundamentação

Neste capítulo serão abordados temas importantes para a compreensão desse trabalho. Na seção 2.1 aprofunda um pouco mais sobre os Juízes Online. A seção 2.2 explica sobre documentações e sua importância. E por último, a seção 2.3 fala sobre evolução de software.

2.1 Juízes Online

Juízes Online são plataformas amplamente utilizadas no ensino e aprendizado de programação, especialmente em cursos introdutórios. Essas ferramentas permitem a prática de resolução de problemas de programação, oferecendo *feedback* imediato sobre a validade das soluções apresentadas pelos estudantes.

O funcionamento geral dessas plataformas tende a seguir o padrão:

1. Apresentação de um problema: consiste na contextualização de uma questão para ser resolvida usando uma linguagem de programação.

2. Exemplos de entrada e saída: são casos de teste dados como exemplo, onde a partir de determinada entrada (fase inicial do problema) é esperado que determinada saída seja concedida pelo programa (resultado).

3. Área de desenvolvimento: normalmente essas plataformas contam com um editor de texto online com *highlight* para as linguagens de programação suportadas e o disponibilizam para que o programador escreva a solução proposta para o problema e envie para ser avaliada.

4. Avaliação da solução: o código escrito como solução pelo programador será compilado e/ou interpretado pelos servidores da plataforma e depois será executado com vários casos de testes.

5. Após todos esses passos, a plataforma apresenta para o programador um resultado que normalmente é um dos seguintes: submissão aceita, quando o código passa em todos os casos de teste; erro de compilação, se houver problemas na sintaxe ou na compilação; resposta errada, quando a saída do programa não corresponde ao esperado; tempo excedido, se o código não conceder uma resposta no tempo máximo permitido para a questão; e erro em tempo de execução, quando o programa finaliza por conta de algum erro durante sua execução como um acesso indevido a uma posição que não existe de uma lista.

2.1.1 The Huxley

Uma das plataformas mais populares é o *The Huxley* (HUXLEY, 2020), que fornece um vasto repositório de questões, categorizadas por nível de dificuldade e conceitos abordados. Essa ferramenta é utilizada em diversas instituições de ensino como suporte para a aprendizagem prática, contribuindo para o desenvolvimento de habilidades de programação e pensamento lógico dos alunos.

Com uma interface de fácil uso, os usuários conseguem escolher os problemas que querem resolver com facilidade e prática já que estão todos categorizados e nivelados na plataforma. Após enviar uma resposta, o *The Huxley* faz a execução do código fonte para cada caso de teste e avalia a submissão considerando a exatidão da resposta e o tempo de execução.

2.2 Documentações

Em (SOMMERVILLE, 2019) a documentação de software é abordada em diferentes pontos e perspectivas, sendo sempre enfatizada a sua importância para um software robusto e manutenível. Desde sistemas pequenos como uma calculadora básica até sistemas grandes que lidam com vidas como um software para uma máquina de raio x, a documentação é uma parte crucial que pode determinar o sucesso ou falha do sistema.

2.2.1 Documentações de linguagens

Linguagens de programação também são softwares e, como qualquer outro, precisam ser documentadas para difundir o conhecimento sobre a mesma e facilitar a sua manutenção.

Todos os programas de computador podem ter erros lançados pelo compilador e/ou interpretador durante seu processo de compilação e/ou interpretação. Esses erros também devem ser documentados para que o programador consiga investigar a causa e encontrar formas de contorná-lo. É justamente nessa parte da documentação que esse trabalho atua.

Olhando por um ponto de vista específico, esse trabalho pode ser considerado com uma extensão da documentação dos erros da linguagem de programação Python. Extensão essa que

tenta guiar alunos de programação iniciantes para possíveis soluções dos seus erros e possíveis causas.

2.3 Evolução de software

Como dito por (SOMMERVILLE, 2019), *softwares* precisam de atualizações constantes para atender as necessidades dos seus usuários, corrigir falhas e acompanhar mudanças do seu ambiente operacional.

Este trabalho irá realizar uma mistura de manutenção adaptativa, evolutiva e preventiva (SOMMERVILLE, 2019) nos trabalhos de Jesus (JESUS, 2018) e Rezende (REZENDE, 2024).

A manutenção adaptativa, no contexto desse trabalho, será a adaptação das ferramentas para os seus requisitos atualizados. A manutenção evolutiva, de certa forma semelhante a adaptativa, adicionará novas funcionalidades para atingir os objetivos específicos desse trabalho. Por último, a manutenção preventiva se ateve em realizar alterações para evitar problemas futuros na manutenção de cada uma das ferramentas.

O processo de evolução de software tem a mesma duração do tempo de vida do sistema que está sendo evoluído. Como dito pela lei de evolução de Lehman (LEHMAN; BELADY, 1985), os sistemas que não evoluem entram cada vez mais em desuso e tornam-se inúteis com o passar do tempo. Afirmação que torna esse trabalho ainda mais relevante para manter esses sistemas em constante atualização.

2.4 Explain my error

Jesus criou a *API EME* no projeto (JESUS, 2018) para facilitar o entendimento dos erros na linguagem Python, que foi escolhida por sua relevância na comunidade. Atualmente mais linguagens são suportadas como por exemplo as linguagens C e Java.

Na versão mais atual da *API EME*, a rota principal recebe uma requisição HTTP POST tendo como corpo um JSON com a mensagem de erro e a extensão do arquivo (linguagem que gerou o erro). Essa rota tem como resultado um JSON com a mensagem de erro traduzida e links de apoio obtidos na *API Stack Exchange*.

2.5 Friendly message

Tendo como base o trabalho de Jesus (JESUS, 2018), Rezende criou uma aplicação *Web* em seu projeto (REZENDE, 2024) que é capaz de facilitar o uso da EME com as submissões enviadas para as questões do *The Huxley*.

Com um *Single Page Application (SPA)* responsivo, que usa ferramentas como o *React* que é apresentado no capítulo 4, para os usuários visualizarem os seus dados do *The Huxley* e um servidor *Web* que é responsável por facilitar o uso das ferramentas TH e EME em conjunto, a aplicação criada por Valmir é um facilitador para estudantes iniciantes em programação e qualquer outra pessoa que queira fazer uso da EME em conjunto com o *The Huxley*.

SPA são sites de páginas únicas que funcionam como aplicações nos navegadores. Normalmente eles carregam o HTML completo apenas uma vez e todas as outras atualizações são parciais otimizando o tempo de carregamento e a experiência do usuário.

2.5.1 Especificação de requisitos

O requisitos funcionais e não funcionais foram especificados em (REZENDE, 2024), e estão presentes nos Quadros 1 e 2:

Quadro 1 – 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.

Fonte: (REZENDE, 2024)

Quadro 2 – 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: (REZENDE, 2024)

2.5.2 Casos de uso

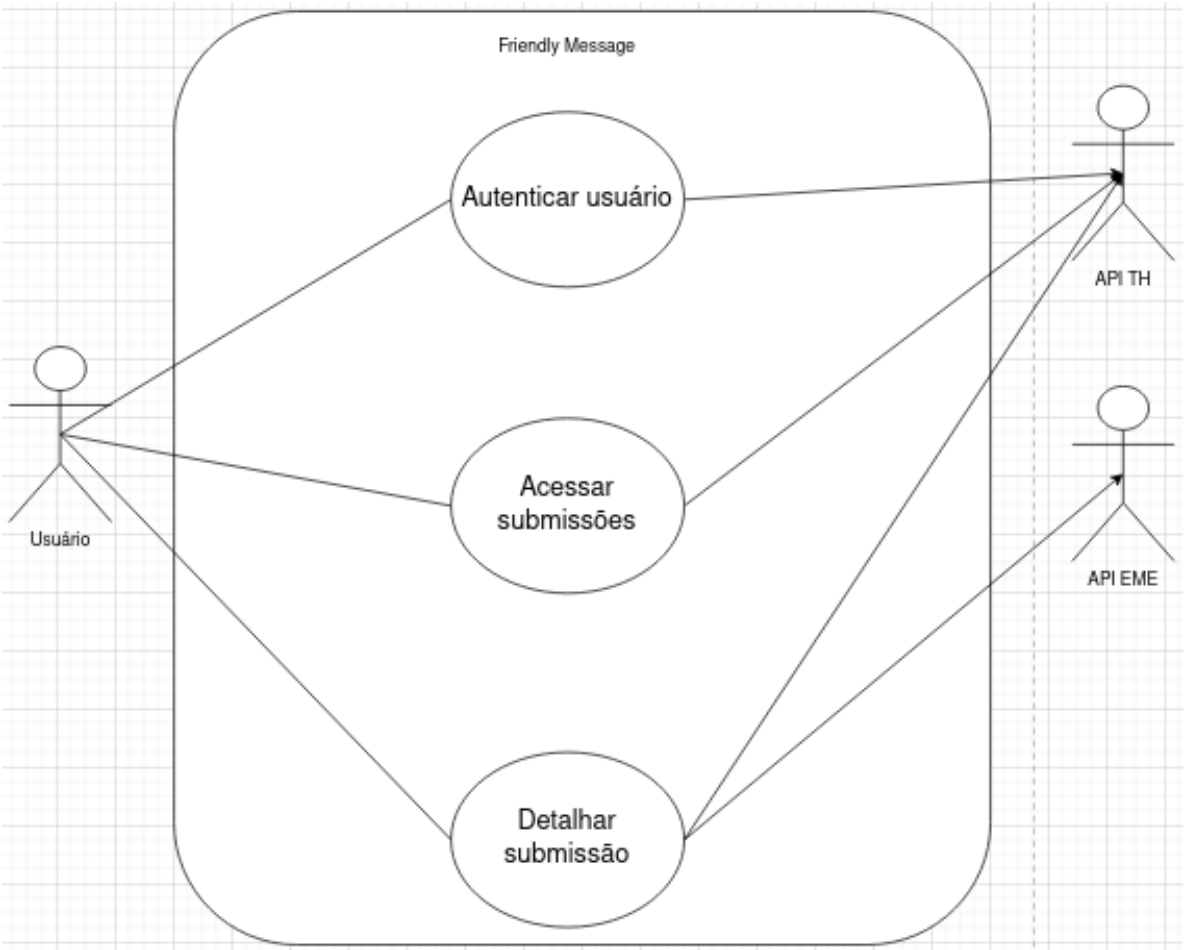
Assim como os requisitos, os casos de uso foram descritos em (REZENDE, 2024) e serão apresentados mais adiante nessa seção. O diagrama de casos de uso é exibido na Figura 4.

Como descrito em (REZENDE, 2024), o sistema tem o caso de uso de autenticação apresentado no Quadro 3, o caso de uso referente às submissões presente no Quadro 4 e o caso de uso referente às mensagens amigáveis presente no Quadro 6.

Os atores do sistema foram descritos em (REZENDE, 2024) como sendo:

- Usuário: qualquer utilizador do *The Huxley*.
- *API* TH: *API* do Juiz Online *The Huxley*.
- *API* EME: *API* do projeto desenvolvida em (JESUS, 2018) por Jesus, *Explain My Error*.

Figura 4 – Diagrama de casos de uso



Fonte: (REZENDE, 2024)

Quadro 3 – Descrição do Caso de Uso Autenticar Usuário

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: (REZENDE, 2024)

Quadro 4 – 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.

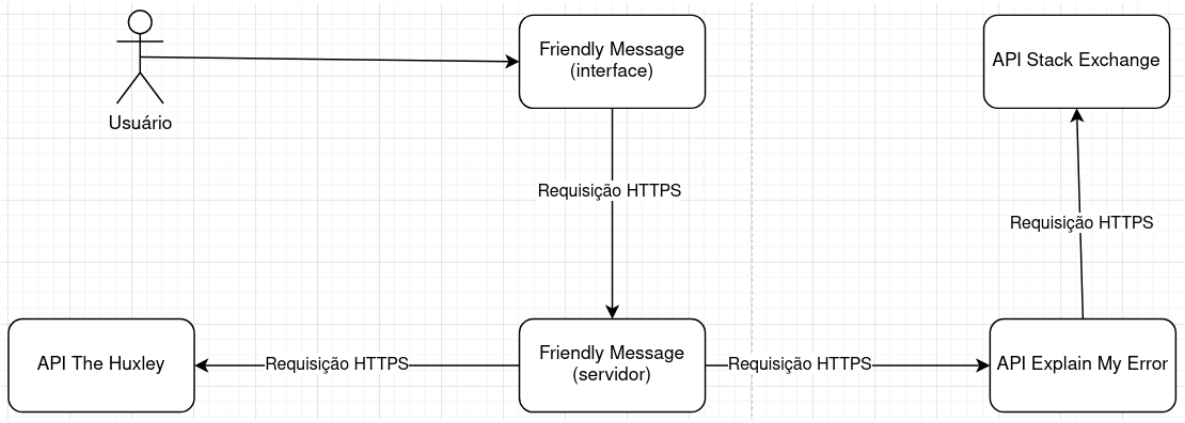
Fonte: (REZENDE, 2024)

2.5.3 Arquitetura

Como descrito em (REZENDE, 2024), a arquitetura usada nas aplicações segue o padrão cliente-servidor (SOMMERVILLE, 2019, 160) onde a interface *web* do FM representa o cliente que envia requisições HTTPS para o servidor FM que por sua vez realiza as requisições HTTPS para a API TH e a API EME.

Uma representação da arquitetura completa dos sistemas pode ser visualizada na Figura 5. O Usuário acessa por meio de um navegador a aplicação FM Interface que faz requisições HTTPS para a aplicação FM Servidor.

Figura 5 – Diagrama de arquitetura da aplicação



Fonte: Autor (2024)

3
Solução proposta

Neste capítulo abordaremos as especificações atualizadas do projeto. A seção 3.1 detalha os requisitos atualizados. A seção 3.2 apresenta os casos de uso atualizados. A seção 3.3 contém a arquitetura atualizada dos sistemas em conjunto.

3.1 Especificação de requisitos atualizados

A maioria dos requisitos funcionais e não funcionais foram mantidos da forma como foram especificados em (REZENDE, 2024). A única diferença se dá para o requisito RF005 que precisou ser alterado para ficar de acordo com o que é esperado para esse trabalho e está presente no Quadro 5:

Quadro 5 – Requisitos Funcionais atualizados

Id	**Título**	**Descrição**
RF005	Apresentar mensagem de erro amigável, links de auxílio e contextualizações	O sistema deve apresentar ao usuário a mensagem de erro amigável juntamente aos links de auxílio e informações extras sobre a classe de erro que forneçam mais contexto para o usuário e o auxiliem no entendimento e solução do problema.

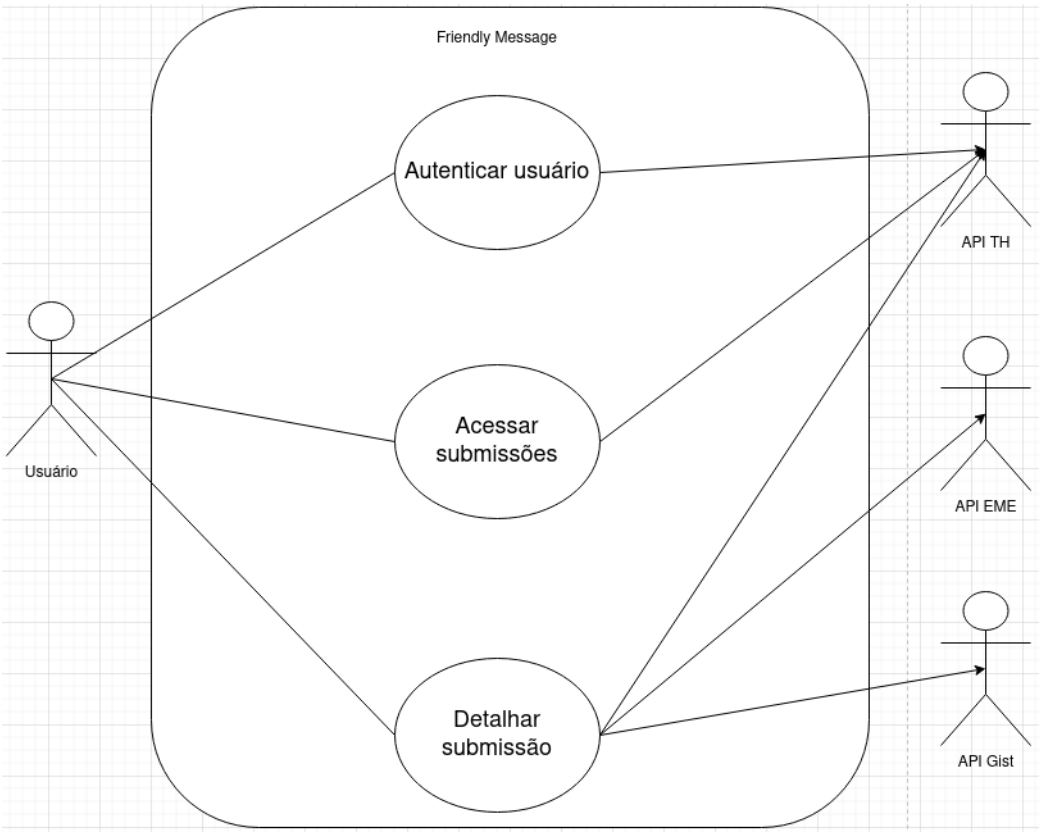
Fonte: Autor (2024)

3.2 Casos de uso

A maioria dos casos de uso se mantiveram sem alterações com poucas exceções que serão apresentadas na seção 3.2.1 e com o diagrama de casos de uso atualizado sendo exibido na Figura 6.

Com a evolução dos sistemas, um novo ator foi adicionado e denominado *API Gist* que se refere à *API* do GitHub onde foram mantidas as documentações dos tipos de erros.

Figura 6 – Diagrama de casos de uso atualizado



Fonte: Autor (2024)

3.2.1 Atualização dos casos de uso

O caso de uso referente as mensagens amigáveis foi o único que sofreu alterações para ficar de acordo com as necessidades da nova versão das aplicações e é apresentado no Quadro 6.

Quadro 6 – Descrição do Caso de Uso: Detalhar Submissões (atualizado)

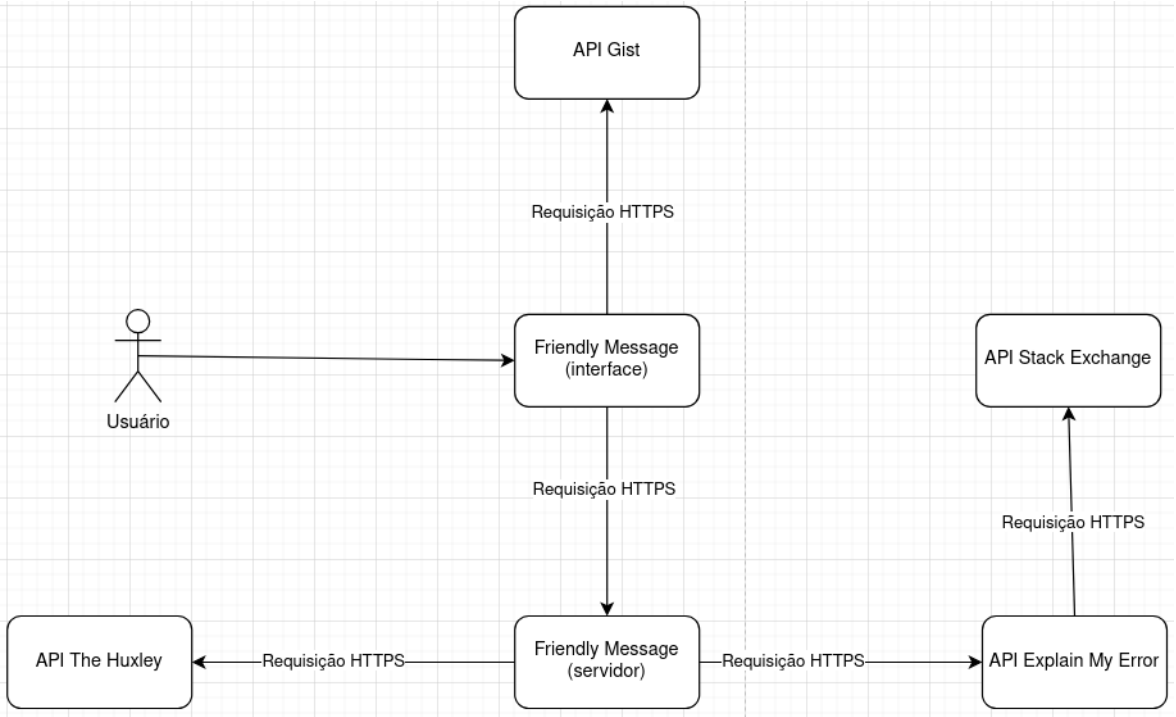
CSU03 – Detalhar Submissões	
Objetivo	Exibir casos de teste, código-fonte e mensagem de erro associadas a um caso de teste específico e sua mensagem amigável em conjunto com informações extras.
Descrição	O usuário, por meio da listagem de submissões, pode selecionar uma submissão específica para visualizar seus detalhes. Ao detalhar uma submissão, é possível obter a mensagem de erro retornado pelo juiz online, em cada caso de teste, em conjunto com a mensagem traduzida pela *API* EME e o contexto com informações extras sobre a classe de erro e formas de solucionar ele.

Fonte: Autor (2024)

3.3 Evolução na arquitetura

De modo geral, toda a arquitetura se mantém como descrita em (REZENDE, 2024) com a adição da API Gist que mantém as documentações das classes de erros. Essa nova parte da arquitetura funciona em conjunto com as outras partes: A *Friendly Message Interface* faz a requisição para a *Friendly Message Servidor* que por sua vez faz as devidas requisições para a *API Explain My Error* e para a *API* EME obtendo as informações sobre a submissão e a mensagem de erro amigável, após receber essas informações a *Friendly Message Interface* realiza uma requisição para a *API* Gist obtendo a contextualização para a classe de erro da submissão. Todo esse processo está resumido na Figura 7 que contém a arquitetura atualizada da aplicação.

Figura 7 – Diagrama de arquitetura da aplicação atualizado



Fonte: Autor (2024)

4
Ferramentas

Neste capítulo serão apresentadas as tecnologias que foram usadas para alcançar os objetivos deste trabalho. Sendo elas o **React**, **ASP.NET core**, **Docker**, **Bitbucket**, **AWS**, **GitHub Gist** e **Python**, cada uma será descrita em uma seção.

O sistema completo contém duas aplicações *back-end*, uma responsável pela tradução e explicação do erro (API Explain My Error) e a outra responsável pela comunicação com o *The Huxley* (*Friendly Message* Servidor), e uma aplicação *front-end* para apresentação das informações para o usuário (*Friendly Message Interface*).

4.1 ECMAScript

O ECMA é a associação de fabricantes que realiza a padronização de dados e linguagens de programação. Ele surgiu a partir de uma necessidade que foi notada por grandes empresas da Europa (ECMA, 2020). Esta associação é a responsável pelo ECMAScript.

O ECMAScript serve como um modelo de padronização para uma linguagem, ou seja, ele é a especificação formal da linguagem para que todos que forem implementá-la o façam de maneira padrão. Foi a partir desse ponto, sua especificação pelo ECMAScript, que o JavaScript dominou a internet.

4.1.1 JavaScript

O JavaScript foi criado na década de 90 por Brendan Eich, quando ainda trabalhava para a Netscape. Como uma linguagem de *script* para dar dinamismo às páginas *Web*. Mas foi a partir de 1997, após a sua especificação pelo ECMA (ECMA, 2019), que o JavaScript ganhou notoriedade e cresceu como a linguagem oficial dos navegadores *Web*.

4.1.2 React

React é uma biblioteca JavaScript desenvolvida pelo Facebook, utilizada para a construção de interfaces de usuário. Desde seu lançamento em 2013, o React tem ganhado cada vez mais espaço no mercado de desenvolvimento e conta com uma comunidade grande de desenvolvedores.

O sucesso do React se deve, em grande parte, à sua flexibilidade, eficiência e à vasta comunidade de desenvolvedores que suportam e ampliam suas capacidades. É uma ferramenta poderosa para o desenvolvimento moderno de interfaces, adotada por empresas de todos os tamanhos ao redor do mundo (W3SCHOOLS, 2023; NETWORK, 2023).

Por esses motivos o React foi usado para construção da aplicação *front-end*.

4.2 ASP.NET core

ASP.NET Core é um framework open-source e multiplataforma desenvolvido pela Microsoft, utilizado para a construção de aplicações web. O ASP.NET Core foi lançado em 2016 como uma evolução do ASP.NET tradicional, redesenhado para ser mais leve, modular e otimizado para desempenho, permitindo o desenvolvimento de aplicações que podem ser executadas no Windows, macOS e Linux (MICROSOFT, 2024).

A aplicação *back-end* que é responsável por fazer a comunicação com o *The Huxley* foi criada usando o ASP.NET core.

4.3 Docker

Docker é uma plataforma open-source que permite aos desenvolvedores automatizar a implantação de aplicações em ambientes isolados e padronizados que são chamados de *containers*. Essa ferramenta revolucionou a maneira como as aplicações são distribuídas e executadas, promovendo a portabilidade e consistência entre diferentes ambientes, sendo útil desde o ambiente de desenvolvimento até o de produção (DOCKER, 2024b).

Docker também simplificou o processo de *continuous integration* (CI)/ *continuous delivery* (CD), já que os containers garantem que o ambiente de execução da aplicação seja o mesmo desde o desenvolvimento local até a produção. Garantindo que as dependências e configurações sejam consistentes em todas as fases do ciclo de desenvolvimento (DOCKER, 2024a).

As duas aplicações *back-end* usam o Docker para facilitar a execução, manutenção e evolução delas em diferentes máquinas e por diferentes pessoas.

4.4 Bitbucket

Bitbucket é uma plataforma de hospedagem de repositórios Git muito utilizada para controle de versão e colaboração em equipes de desenvolvimento de software. O Bitbucket oferece recursos como pipelines de CI e CD, suporte a *pull requests* e revisão de código (ATLASSIAN, 2024).

Todos os repositórios remotos estão hospedados no Bitbucket que oferece uma interface amigável e flexível.

4.5 Amazon Web Services

A *Amazon Web Services* é uma plataforma de computação em nuvem que oferece diversos tipos de serviços como computação, armazenamento, redes e inteligência artificial. A AWS é renomada por sua escalabilidade, flexibilidade e disponibilidade global, sendo assim usada por empresas de pequeno a grande porte para hospedar suas aplicações, armazenar dados e executar cargas de trabalho críticas com alta segurança e performance (AMAZON, 2024).

Um dos principais serviços da AWS é o *Amazon Elastic Compute Cloud*, que permite a criação e gerenciamento de máquinas virtuais de maneira escalável e com custo reduzido principalmente em comparação com modelos *on premise*. O EC2 foi usado para hospedar todas as aplicações em conjunto com os serviços de rede da AWS.

4.6 GitHub Gist

O GitHub Gist é uma ferramenta disponibilizada pela plataforma GitHub que permite o compartilhamento de trechos de código, scripts, ou qualquer tipo de texto formatado. Uma das maiores vantagens do Gist é a possibilidade de criar versões públicas e privadas de arquivos simples ou conjuntos de arquivos, permitindo que desenvolvedores colaborem e compartilhem arquivos de forma rápida e eficiente (GITHUB, 2024).

Os Gists podem ser integrados diretamente em aplicações web. Para desenvolvedores que buscam uma maneira fácil de compartilhar pequenas partes de seus projetos ou resolver problemas específicos, o Gist pode se tornar uma ferramenta essencial.

O GitHub Gist foi usado quase que como uma *Content Delivery Network* - CDN, que são servidores que possuem como objetivo otimizar o tempo de entrega de conteúdos distribuídos. Para facilitar a manutenção da documentação dos erros e tornando elas independentes do código fonte das aplicações, foram armazenadas no GitHub Gist como arquivos no formato Markdown.

4.7 Python

Python é uma linguagem de programação de alto nível, que é utilizada em diversas áreas do desenvolvimento de software. Criada por Guido van Rossum e lançada em 1991, Python é conhecida por sua sintaxe simples, o que a torna uma escolha popular para iniciantes e profissionais (ROSSUM, 1991).

A linguagem é frequentemente recomendada para aprendizado de programação devido à sua simplicidade e suporte extensivo. A Python Software Foundation (PSF) desempenha um papel crucial na manutenção e promoção da linguagem, oferecendo recursos e suporte para a comunidade (FOUNDATION, 2024).

A aplicação *back-end* que é responsável pela explicação dos erros foi criada usando o Python.

5

Documentação dos erros

Neste capítulo será apresentado o processo e ferramentas utilizadas para criar a documentação dos erros. As ferramentas utilizadas foram algumas ferramentas de busca, o Chat GPT, a documentação oficial da linguagem e terminal para execução e testes dos erros.

5.1 Processo de criação

O processo de criação das documentações dos erros seguiu alguns passos para chegar na versão atual, sendo eles descritos na lista abaixo.

1. API EME: A dissertação (JESUS, 2018) de mestrado de Jesus foi usada como base para definir quais erros da linguagem Python seriam documentados. De lá foram obtidos os erros e sua classificação inicial, como por exemplo o tipo de erro SyntaxError.

2. Documentação oficial: Para cada erro obtido da dissertação, foi utilizada a documentação da linguagem Python (ROSSUM, 1991) para entender mais sobre o que o erro representa, formas de replicar e resolver ele.

3. Ferramentas de busca: Em alguns casos, a documentação oficial não foi suficiente para compreender o erro e por conta disso foram usadas ferramentas como o Google e o DuckDuckGo (DUCKDUCKGO,) para encontrar outras fontes que pudessem fornecer mais detalhes. Um dos erros em que foi necessário realizar esse processo foi o do tipo UnicodeDecodeError.

4. Chat GPT: Para quase todos os tipos de erros, o Chat GPT foi utilizado para gerar exemplos que pudessem replicar o problema e formas de resolver o mesmo.

5. Interpretador do Python: Para cada exemplo gerado e forma de solucionar o problema, o *script* foi testado usando o ambiente do console do Python 3 para tentar garantir que a execução e lançamento de exceções aconteceriam de acordo com o esperado.

6. Versão inicial da documentação: Foi criada uma documentação inicial usando o Google Docs para avaliação e correções.

7. Versão atual da documentação: A versão atual da documentação foi transformada em um arquivo *markdown* para cada tipo de erro e os arquivos estão hospedados no GitHub Gist.

6

Desenvolvimento

Nesse capítulos será descrito o processo de desenvolvimento e evolução dos sistemas para atingir os objetos do trabalho. A seção 6.1 apresenta de forma geral os passos tomados para evoluir os sistemas. A seção 6.2 aprofunda um pouco mais nos pontos mais importantes do processo de evolução. A seção 6.3 apresenta os resultados finais.

6.1 Etapas para a evolução do software

Para atingir os objetivos desse trabalho foi necessário primeiro entender a arquitetura e funcionamento de todos os projetos envolvidos. Os trabalhos de Galileu (JESUS, 2018) e Valmir (REZENDE, 2024) ajudaram a contextualizar tudo que estava pronto e como funcionava mas foi necessário um esforço extra para conseguir aprofundar o conhecimento sobre as ferramentas.

O processo de aprofundamento se deu na ordem cronológica em que as ferramentas foram desenvolvidas. Iniciando pela EME, que é a base de todo o projeto, onde as traduções são realizadas e chegando até a FM que é responsável por integrar o *The Huxley* com a EME e apresentar isso para o usuário final. Cada ferramenta apresentou diferentes peculiaridades e desafios.

Após ter domínio sobre as aplicações, a evolução ocorreu de forma gradual. O processo evolutivo das ferramentas foi inverso ao processo de aprofundamento:

- Primeiro a FM foi evoluída para se conectar com a *API* Gist e apresentar uma seção nova e dinâmica, na página de detalhes da submissão, contendo um arquivo *Markdown* com os detalhes sobre a classe de erro.
- Segundo a EME foi evoluída em conjunto com o servidor da FM para retornarem mais informações sobre as traduções do erro para que fosse possível que a aplicação cliente

conseguisse decidir sozinha qual seção dinâmica que deveria ser carregada na página de detalhes da submissão.

6.2 Desafios enfrentados

Um dos maiores desafios se deu para conseguir executar o projeto EME. Por esse motivo e visando sua manutenção futura, foi decidido que o Docker deveria ser configurado para simplificar o processo de execução e implantação da ferramenta. Após essa configuração, o processo na EME foi de atualizar a função principal da tradução de erros na linguagem Python para que retornasse o tipo de erro que foi considerado durante a tradução da mensagem recebida.

Um outro desafio foi encontrar uma forma de tornar a documentação dinâmica, de fácil manutenção e independente do código fonte das aplicações para não ficar *hard code*. No final, foi escolhido o Github Gist para armazenar arquivos *Markdown* contendo a documentação de cada um dos tipos de erros. Essa foi a opção escolhida já que é simples conceder acesso para que diferentes pessoas possam colaborar nessas documentações, é mantido um certo nível de controle de versões e o Github fornece uma *API* simples para obter esses arquivos, assim todos os requisitos foram atendidos.

6.3 Resultado das alterações

A Figura 8 mostra a tela de detalhes de uma submissão atualizada que apresenta a documentação dinâmica sobre cada classe de erro Python, sendo o exemplo apresentado a documentção para o tipo de erro SyntaxError. Após carregar a tela de detalhes e escolher uma submissão com erro, o cliente FM faz uma requisição para a *API* Gist solicitando a documentação para o tipo de erro que a aplicação EME identificou. Após receber o arquivo *Markdown*, o cliente FM converte ele para ser renderizado como *HTML* em uma seção nomeada como "Contextualização do erro".

Figura 8 – Resultado final das alterações

Fonte: Autor (2024)

7

Conclusão

Em todo o trabalho de evolução das ferramentas, o foco foi mantê-las o mais manutenível possível e que os alunos iniciantes em programação possam fazer uso dessa aplicação para facilitar o seu processo de estudo.

Esse trabalho teve como objetivo criar uma documentação extra sobre os erros na linguagem Python e evoluir as ferramentas criadas em (JESUS, 2018) e (REZENDE, 2024). Por isso, o Github Gist foi utilizado como uma CDN para manter a documentação extra que foi criada sobre erros e as aplicações EME e FM foram evoluídas para apresentarem estas informações extras.

Como trabalho futuro, é importante atestar o resultado da ferramenta por meio de um experimento controlado que avalie turmas que a usem e compare com turmas que não usem. Com isso, será possível obter uma avaliação sobre a ferramenta e identificar onde ela deve ser melhorada.

Além disso, como o trabalho focou apenas na documentação dos erros da linguagem Python, seria interessante abranger as outras linguagens que a EME suporta fazendo com que todas possuam acesso a essa nova seção da tela de detalhes da submissão. Como as documentações são independentes do código, torna menos complicado adicionar esse suporte já que seria necessário apenas criar o conteúdo e adicionar na implementação da linguagem a informação de qual classe de erro foi usada durante a tradução da mensagem.

Como a EME não depende de uma plataforma específica, outra evolução que pode ser feita é realizar a integração com outras plataformas de juiz online além do *The Huxley*.

Para realizar essas evoluções em trabalhos futuros, algumas informações podem ser necessárias, tais como:

1. Acesso ao repositório de código-fonte do EME[1].

2. Acesso aos repositórios do *frontend*[2] e do *backend* e [3] do FM.

3. Acesso ao Gist com a documentação dos erros[4].

[1] https://bitbucket.org/explain-my-error/eme-api/src/master
[2] https://bitbucket.org/explain-my-error/friendly-message-client/src/master
[3] https://bitbucket.org/explain-my-error/friendly-message-api/src/main
[4] https://gist.github.com/explainmyerror/779d4ec54d04252e11ff9bc046fb3d5f

Referências

AMAZON. *Amazon Web Services Documentation*. 2024. Disponível em: <https://aws.amazon.com/documentation>. Acesso em: 24 ago. 2024. Citado na página 24.

ATLASSIAN. *Bitbucket Data Center documentation*. 2024. Disponível em: <https://confluence.atlassian.com/bitbucketserver>. Acesso em: 24 ago. 2024. Citado na página 24.

DOCKER. *Configure CI/CD for your Node.js application*. 2024. Disponível em: <https://docs.docker.com/language/nodejs/configure-ci-cd>. Acesso em: 23 ago. 2024. Citado na página 23.

DOCKER. *What is Docker?* 2024. Disponível em: <https://www.docker.com/what-docker>. Acesso em: 23 ago. 2024. Citado na página 23.

DUCKDUCKGO. *DuckDuckGo Search Engine*. Disponível em: <https://duckduckgo.com>. Acesso em: 24 jan. 2025. Citado na página 26.

ECMA. *ECMA-262, 10th edition, June 2019 ECMAScript® 2019 Language Specification*. 2019. Disponível em: <https://www.ecma-international.org/ecma-262/10.0/index.html>. Acesso em: 10 ago. 2024. Citado na página 22.

ECMA. *history*. 2020. Disponível em: <https://ecma-international.org/about-ecma/history>. Acesso em: 10 ago. 2024. Citado na página 22.

FOUNDATION, P. S. *Mission*. 2024. Disponível em: <https://www.python.org/psf/mission/>. Acesso em: 25 ago. 2024. Citado na página 25.

GITHUB. *Creating gists*. 2024. Disponível em: <https://docs.github.com/en/get-started/writing-on-github/editing-and-sharing-content-with-gists/creating-gists>. Acesso em: 24 ago. 2024. Citado na página 24.

HUXLEY. *The Huxley*. 2020. Disponível em: <https://www.thehuxley.com/>. Acesso em: 02 de dezembro de 2023. Citado 2 vezes nas páginas 9 e 14.

INEP. *Censo da educação superior*. 2021. Disponível em: <https://download.inep.gov.br/educacao_superior/censo_superior/documentos/2021/apresentacao_censo_da_educacao_superior_2021.pdf>. Acesso em: 02 de dezembro de 2023. 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 2, 3, 10, 15, 17, 26, 28 e 31.

KELLEHER, C.; PAUSCH, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 37, n. 2, p. 83–137, 2005. Citado na página 9.

LABERGE, L. et al. *How COVID-19 has pushed companies over the technology tipping point—and transformed business forever*. 2020. Disponível em: <https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/

how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever>
Acesso em: 02 de dezembro de 2023. Citado na página 9.

LEHMAN, M. M.; BELADY, L. A. *Program Evolution: Processes of Software Change*. London: Academic Press, 1985. Citado na página 15.

MICROSOFT. *Overview of ASP.NET Core*. 2024. Disponível em: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core>. Acesso em: 23 ago. 2024. Citado na página 23.

NETWORK, M. D. *Introduction to React*. 2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started>. Acesso em: 10 ago. 2024. Citado na página 23.

REZENDE, V. V. d. C. *Uma ferramenta Web para simplificar mensagens de erro de programação para estudantes iniciantes*. Sergipe, Brasil: [s.n.], 2024. TCC (Graduação) – Universidade Federal de Sergipe. Orientador: Dr. Alberto Costa Neto. Citado 11 vezes nas páginas 2, 3, 10, 15, 16, 17, 18, 19, 21, 28 e 31.

ROSSUM, G. van. *Python Programming Language*. Python Software Foundation, 1991. Disponível em: <https://www.python.org/doc/>. Acesso em: 25 ago. 2024. Citado 2 vezes nas páginas 25 e 26.

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 14, 15 e 18.

W3SCHOOLS. *React Tutorial*. 2023. Disponível em: <https://www.w3schools.com/react/>. Acesso em: 10 ago. 2024. Citado na página 23.

WEBER, G.; BRUSILOVSKY, M.; STEINLE, F. Elm-pe: An intelligent learning environment for programming, 1996. *Disponivelem< http://www. psychologie. uni-trier. de*, v. 8000, 2014. Citado na página 9.