UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Tecnologia

Arthur Guedes De Souza

PACDEV: Teste de programas por terceiros

Limeira

2022

Arthur Guedes De Souza

PACDEV: Teste de programas por terceiros

Monografia apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Guilherme Palermo Coelho

Este exemplar corresponde à versão final da monografia defendida por Arthur Guedes de Souza e orientada pelo Prof. Dr. Guilherme Palermo Coelho.

Limeira

2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Faculdade de Tecnologia
Luiz Felipe Galeffi - CRB 8/10385

So89p

Souza, Arthur Guedes de, 1998-
PACDEV : teste de programas por terceiros / Arthur Guedes de Souza. – Limeira, SP : [s.n.], 2022.

Orientador: Guilherme Palermo Coelho.
Trabalho de Conclusão de Curso (graduação) – Universidade Estadual de Campinas, Faculdade de Tecnologia.

1. Software - Testes. 2. Sistemas de computação. 3. Python (Linguagem de programação de computador). I. Coelho, Guilherme Palermo,1980-. II. Universidade Estadual de Campinas. Faculdade de Tecnologia. III. Título.

Informações adicionais, complementares

Título em outro idioma: PACDEV: third-party program testing
Palavras-chave em inglês:
Computer software - Testing
Computer systems
Python (Computer program language)
Titulação: Bacharel
Banca examinadora:
Guilherme Palermo Coelho [Orientador]
Gisele Busichia Baioco
Ulisses Martins Dias
Data de entrega do trabalho definitivo: 28-07-2022

FOLHA DE APROVAÇÃO

Abaixo se apresentam os membros da comissão julgadora desta monografia para o Título de Bacharel em Sistemas de Informação, a que se submeteu o aluno Arthur Guedes de Souza, na Faculdade de Tecnologia – FT/UNICAMP, em Limeira/SP.

Prof. Dr. Guilherme Palermo Coelho

Presidente da Comissão Julgadora

Profa. Dra. Gisele Busichia Baioco

FT/UNICAMP

Prof. Dr. Ulisses Martins Dias

FT/UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Trabalhos de Conclusão da UNICAMP.

“A informática me distanciou dos livros, não da leitura.”

(Jeziel L. Carvalho)

Agradecimentos

Agradeço aos amigos e colegas da Universidade Estadual de Campinas pelo apoio e companheirismo durante a trajetória do curso, principalmente aqueles que ouviram minha constante reclamação de largar o curso, em especial à Bianca Bonnate, Fiama Demitria, Matheus Cumpian e Matheus Bruther.

Ao meu Orientador, Prof. Dr. Guilherme Palermo Coelho, pela dedicação e atenção despendidas durante a realização deste trabalho.

À Universidade Estadual de Campinas e, em especial, a todos os docentes e funcionários da Faculdade de Tecnologia, pela atenção, conhecimento e infraestrutura disponibilizados durante o curso.

Resumo

Este trabalho tem como objetivo a construção de um sistema web que permita o gerenciamento e acompanhamento de testes de programas escritos na linguagem computacional Python. Tal sistema, denominado PACDEV, é direcionado para estudantes e desenvolvedores de software. Para um melhor entendimento do problema explorado no desenvolvimento do trabalho, foi realizada uma análise de softwares já existentes, bem como uma pesquisa de boas práticas de testes e compreensão do processo de desenvolvimento de testes em softwares. O sistema web proposto utiliza-se do GitHub *actions* para efetuar testes em aplicações: uma vez que um repositório é cadastrado em nossa aplicação, o usuário pode submeter arquivos de teste que são enviados ao repositório e inicia-se um processo de teste com estes arquivos através do GitHub *actions*. Este sistema tem como principais componentes uma interface voltada ao cadastro de projetos existentes no GitHub, mecanismos para consultar projetos cadastrados e efetuar submissões de testes a tais projetos, além de permitir a consulta ao histórico destes. Espera-se que este trabalho contribua com a formação de profissionais de TI no âmbito de testes, além de criar a possibilidade de expansão do produto a partir da inclusão de novas funcionalidades em um momento futuro.

Palavras-chave: Teste de Software. Desenvolvimento de Sistemas. Teste Unitário. GitHub actions.

Abstract

This work aims to build a web system that allows the management and monitoring of tests of programs written in Python. Such system, called PACDEV, is aimed at students and software developers. For a better understanding of the problem explored in the development of this work, an analysis of existing software was carried out, as well as a research of good testing practices and the a higher understanding of the software test development process. The proposed web system has as main components an interface aimed at registering existing projects on GitHub, mechanisms to consult registered projects and make test submissions to them, in addition to being able to consult their history. It is expected that this work will contribute to the training of IT professionals in the scope of tests, in addition to creating the possibility of expanding the product from the inclusion of new functionalities at a future time.

Keywords: Software Test. System Development. Unit test. GitHub Actions.

Lista de Figuras

Lista de Siglas e Abreviações

TI	Tecnologia da Informação
TCC	Trabalho de Conclusão de Curso
CI/CD	*Continuous Integration/ Continuous Delivery*
API	*Application Programming Interface*
MVC	*Model-View-Controller*
SO	Sistema Operacional
SQL	*Structured Query Language*
ID	Identificador

Sumário

Capítulo 1

Introdução

Atualmente, a composição do mundo é baseada no meio digital, seja no cotidiano das pessoas ou na vida profissional. Em vista disso, o mundo digital vem ganhando força, requerendo cada vez mais profissionais capacitados em tecnologia da informação e atualizações em sistemas existentes. Tais atualizações devem ser rápidas, mas não podem prejudicar a qualidade do produto a ser entregue.

Em meio a esse cenário, o qual requer que os conteúdos sejam disponibilizados em velocidades quase tão rápidas quanto produzidos, temos a tendência de fomentar a internet como o primeiro canal de publicação de conteúdo, visto que grande parte dos dados e informações geradas atualmente ficam armazenados dentro de um computador.

Partindo desse princípio, a demanda por mão de obra qualificada para o mercado de desenvolvimento de software vem apresentando números crescentes. Segundo uma pesquisa da plataforma de recrutamento GeekHunter (CNN, 2021), somente no primeiro semestre de 2021 houve um aumento entre 131% e 344% nas oportunidades para profissionais de TI (Tecnologia da Informação), variando conforme o conhecimento técnico e experiências no mercado.

Assim, garantir que profissionais disponham de meios para treinar suas habilidades de testes é essencial, uma vez que um software desenvolvido requer testes antes de ser disponibilizado para o cliente final. Além do mais, é importante que a formação de tais profissionais contemple não só a criação desses produtos e sua manutenção, mas também o desenvolvimento de testes, tanto para funcionalidades antigas quanto para novas (BECK, 2002).

Em uma tentativa de apoiar desenvolvedores a melhorarem suas habilidades de testes de software, este trabalho buscou a construção de uma ferramenta que auxilia estudantes e profissionais com diferentes tipos de experiências a desenvolverem suas habilidades para testes de software. A aplicação desenvolvida aqui tem como objetivo ser uma abstração para efetuar testes em aplicações sem a necessidade de se conhecer o código-fonte. Para o uso da aplicação, o desenvolvedor precisa apenas disponibilizar o código-fonte de sua aplicação em um repositório público do GitHub[1], ter um arquivo *Readme.md* na raiz e uma pasta chamada “PACDEV”, contendo um arquivo chamado *Teste,* com qualquer extensão, contendo o nome das funções a serem testadas. A partir dessa abstração, um usuário que anseie exercitar suas habilidades de testes ou até mesmo um desenvolvedor que queira testar sua habilidade de criar códigos sem *bug*s podem facilmente fazê-lo usando o PACDEV.

Este documento apresenta, na forma de relatório técnico, o desenvolvimento da ferramenta PACDEV. No Capítulo 2 serão apresentadas a conceituação de “teste” e as aplicações similares à desenvolvida no presente trabalho. Já no Capítulo 3 será feita uma explicação dos requisitos e uma pequena introdução sobre as tecnologias usadas. No Capítulo 4, será apresentada a ferramenta em si. Finalmente, no Capítulo 5 é apresentada a conclusão deste trabalho.

[1] GitHub. Disponível em <https://github.com>. Acesso em 01 de junho de 2021.

Capítulo 2

Contextualização do Problema

Neste capítulo, será apresentado o domínio do problema em que este trabalho se apoia, de modo que possibilite uma melhor compreensão acerca das ferramentas similares ao sistema desenvolvido.

Inicialmente, serão apresentados os tipos de testes e, em seguida, uma pesquisa de mercado, realizada com intuito de analisar os principais produtos já existentes com o propósito igual e/ou similar ao aqui apresentado, apontando as principais funcionalidades e características de cada um dos sistemas abordados neste trabalho.

2.1 Tipos de Teste de Software

Nesta seção, iremos apresentar os principais tipos de testes, assim como sua aplicabilidade. Vale ressaltar que o usuário não precisa conhecer os tipos de testes para utilizar nosso sistema web. O usuário, conforme as funcionalidades a serem testadas no código, estará sempre utilizando um dos testes citados abaixo.

Lembrando que os testes fazem parte do processo de desenvolvimento de qualquer software, podem ser feitos pelos próprios desenvolvedores ou, em alguns casos, por pessoas especializadas na área de testes.

A construção dos testes tem como objetivo antecipar e corrigir falhas e *bugs* que poderiam aparecer para o usuário final. Apesar de parecer um processo simples, é mais comum que os desenvolvedores foquem na criação das ferramentas do que em seus testes, principalmente pela etapa parecer simples, porém seu uso é essencial para evitar "*fenômenos aleatórios e negativos"*, ou a necessidade de "*apagar um incêndio*" dentro da sua aplicação

após uma atualização que poderia ter sido evitada com sua aplicação tendo sido testada previamente.

2.1.1 Teste de Performance

Testes de Performance normalmente são aplicados para prever o comportamento da aplicação em condições de consumo e estresse específicas, como por exemplo o lançamento de um sistema que, em teoria, terá grande número de acessos, uma vez que pontos como estabilidade, tempo de resposta, escalabilidade e outros variam em um cenário real, com altos níveis de consumo (VALENTE, 2020).

2.1.2 Teste de Integração

Os testes de integração, como o nome sugere, têm por objetivo unir os diversos módulos do sistema e testá-los em conjunto. Eles são realizados após os testes de mesa e testes unitários, os quais garantem o funcionamento individual das partes (VALENTE, 2020).

2.1.3 Teste de Aceitação

Testes de aceitação possuem duas características que os distinguem de todos os testes que estudamos antes. Primeiro, são **testes manuais**, realizados pelos clientes finais do sistema. Segundo, eles não constituem exclusivamente uma atividade de verificação (como os testes anteriores), mas também uma atividade de validação do sistema. A verificação testa se fizemos o sistema corretamente, isto é, de acordo com a sua especificação e/ou requisitos. Já validação testa se fizemos o sistema correto, isto é, aquele que o cliente pediu e precisa (VALENTE, 2020).

Testes de aceitação podem ser divididos em duas fases. **Testes alfa** são realizados com alguns usuários, mas em um ambiente controlado, como a própria máquina do desenvolvedor. Se o sistema for aprovado nos testes alfa,

pode-se realizar um teste com um grupo maior de usuários e não mais em um ambiente controlado. Esses testes são chamados de **testes beta**.

2.2 Pesquisa de mercado

Na fase inicial do planejamento para o desenvolvimento deste sistema *web*, foi realizado um levantamento de mercado visando entender as características das ferramentas voltadas para desenvolvedores testarem suas habilidades, no âmbito de testes, e sua capacidade de interpretar problemas.

A análise foi realizada com os principais nomes do mercado que não se limitam a ter a solução dos problemas abordados em uma linguagem de programação específica. A análise foi feita através de consultas nas especificações técnicas de cada aplicação, disponível em seus respectivos sites. Foram analisadas somente as funcionalidades disponíveis gratuitamente em cada ferramenta, não contemplando funcionalidades no modelo pago, uma vez que este trabalho tem o propósito de ser gratuito.

2.2.1 HackerRank

A primeira aplicação é o HackerRank (HACKERRANK, 2022), uma das principais ferramentas de mercado no segmento de teste de software, feita para desenvolvedores treinarem suas habilidades em construção de código-fonte, se baseando em um enunciado. Nela, os desenvolvedores escrevem programas, geralmente com base em um problema matemático e/ou computacional definido em um determinado enunciado, com o intuito de permitir a análise do raciocínio lógico dos desenvolvedores e o tempo de desenvolvimento. Seus enunciados se referem, no geral, a problemas cotidianos que devem ser resolvidos via soluções computacionais.

Na Figura 1, vemos a tela principal do HackerRank, acessível aos desenvolvedores, com o enunciado de um problema a ser solucionado,

submissões passadas, discussões e, ao lado direito, a opção para *download* de casos simples de teste.

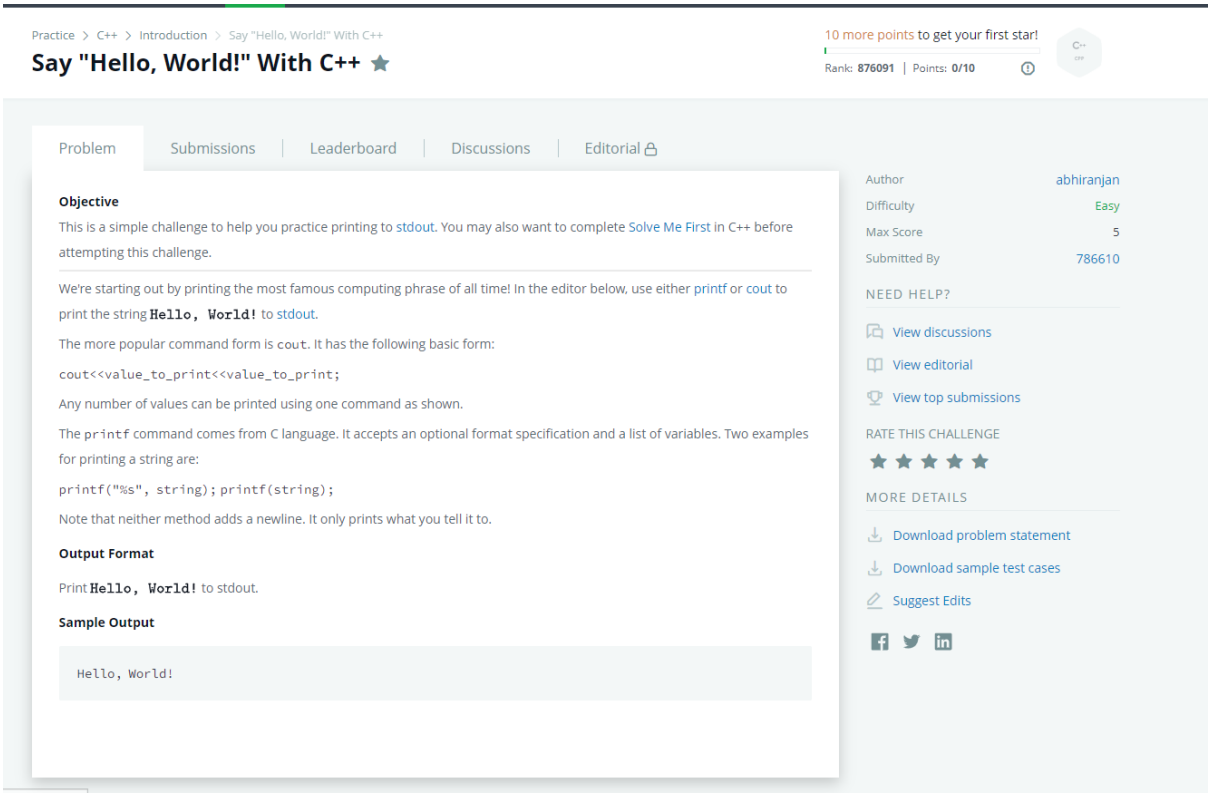

Figura 1 Tela principal do HackerRank, exibindo o projeto "Say 'Hello World!' with C++." Fonte: HackerRank, 2021.

Na Figura 2, é possível visualizar uma área para digitação do código-fonte, acoplada a um compilador *online*, que corresponde à área em que o desenvolvedor deve construir o código solucionador do problema.

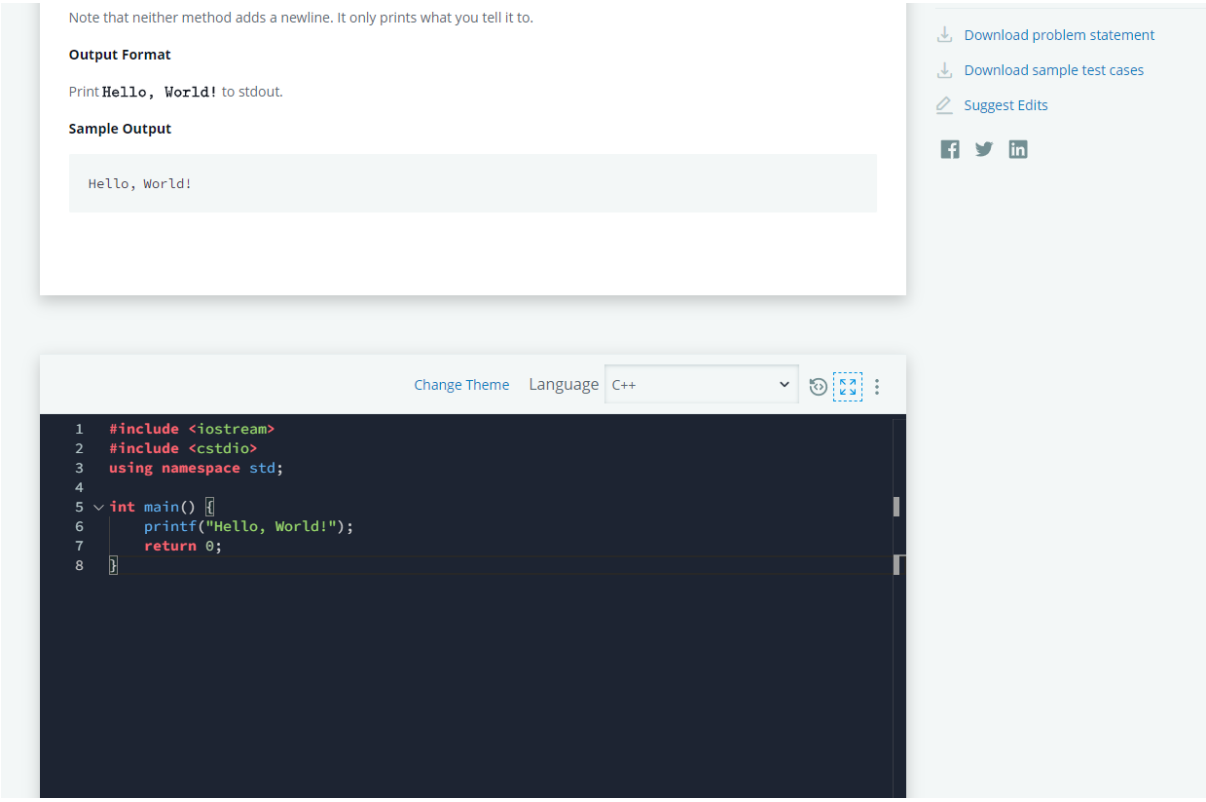

Figura 2 Tela de edição de código-fonte do HackerRank. Fonte: HackerRank, 2021

2.2.2 Leetcode

Assim como o HackerRank, a LeetCode (LEETCODE, 2022) é uma das principais ferramentas no segmento de testes online. Nela, os desenvolvedores também escrevem códigos-fonte se baseando em um determinado enunciado. Em contraposição à ferramenta anterior, os enunciados aqui são voltados ou se referem, no geral, a problemas cotidianos que devem ser resolvidos via soluções computacionais.

Na Figura 3, temos um exemplo da tela principal do LeetCode, que pode ser acessada pelos desenvolvedores: nela temos o enunciado do problema a ser solucionado, submissões passadas e discussões. A solução do problema é apenas disponibilizada na versão *premium* (paga). Em contraponto, no nosso

sistema não existe uma solução definitiva, certa ou ideal. Contudo, existe a solução do próprio desenvolvedor e dos *testes* da aplicação.

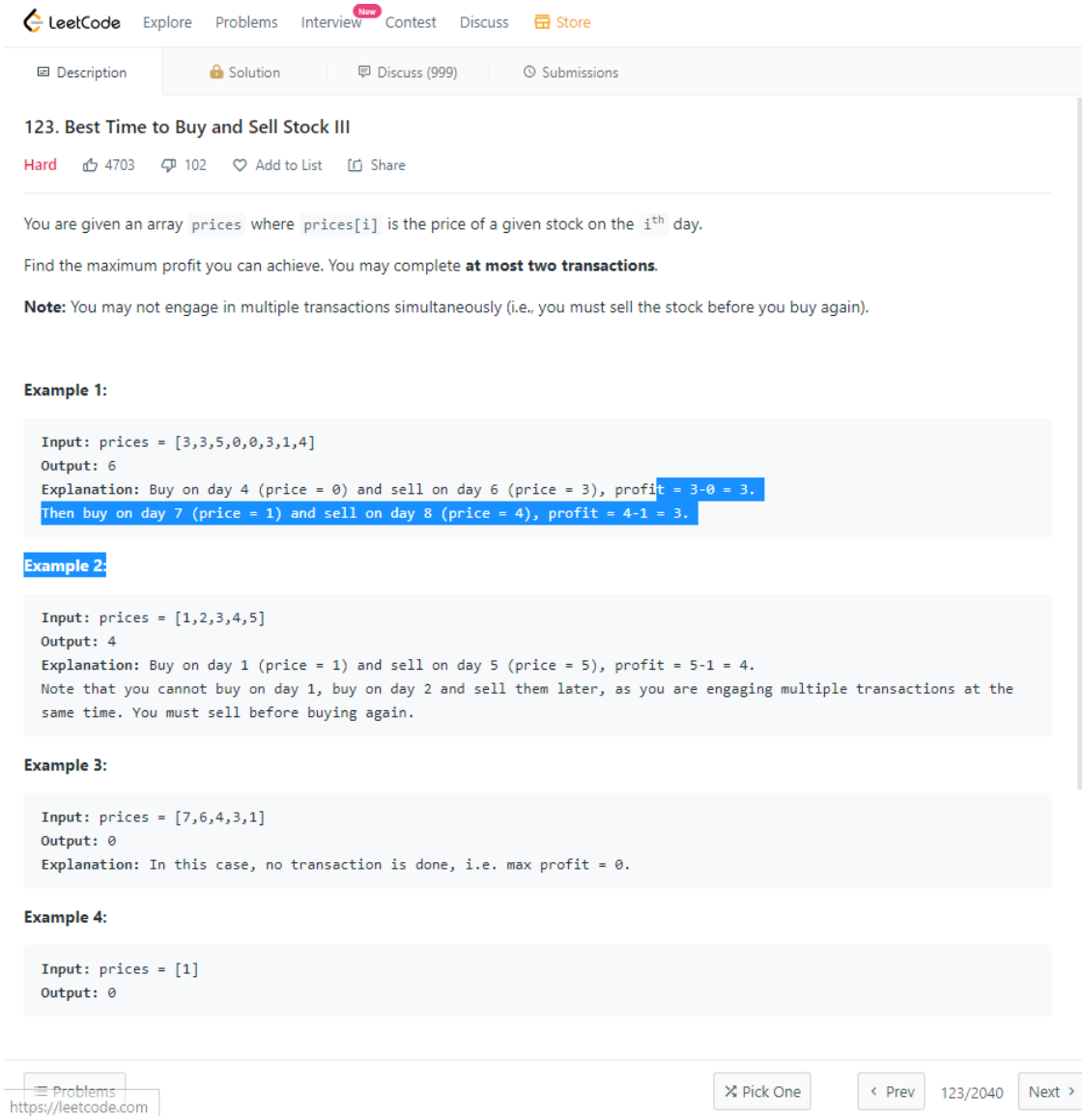

Figura 3 "Best Time to Buy and Sell Stock 3". Fonte: LeetCode, 2022.

2.3 Comparação de sistemas

A pesquisa de mercado teve como objetivo encontrar aplicações similares ao sistema aqui proposto, contudo, não se encontrou aplicações

exatamente iguais. Porém, as ferramentas apresentadas acima têm particularidades que tornam possível ao menos uma breve análise da sua estrutura em relação ao PACDEV.

As ferramentas apresentadas acima também trabalham com testes, porém com propósitos e finalidades diferentes. Elas têm como sua principal funcionalidade, a partir de um enunciado (problema), a escrita de um programa que busca solucioná-lo, sendo aprovado em testes fechados, que fornecem entradas ao programa e comparam seus resultados às saídas esperadas.

Por outro lado, o sistema aqui proposto vai em contraponto a isso, ou seja, o usuário é responsável não por escrever um programa com propósito de passar nos testes, e sim, escrever um teste para um programa já existente.

Importante ressaltar que apenas as empresas detentoras dos sistemas similares aos apresentados aqui (GeekHunter e LeetCode) possibilitam o cadastramento de enunciados com problemas, o que vai contra a proposta deste Trabalho de Conclusão de Curso (TCC), uma vez que os próprios desenvolvedores, ao atenderem determinados requisitos, poderão disponibilizar enunciados.

Capítulo 3

Requisitos

Neste capítulo, serão apresentados os requisitos do problema em que este trabalho se apoia, de modo que possibilite uma melhor compreensão em relação às funcionalidades do sistema.

Em seguida, serão apresentados os requisitos funcionais do sistema, ou seja, o que o sistema deve fornecer ao usuário, seguidos pelos requisitos não funcionais, aqueles que devem existir para um bom funcionamento da aplicação.

Por fim, serão discutidas as tecnologias empregadas assim como as justificativas para suas escolhas.

3.1 Diagrama de casos de Uso

Para ajudar na modelagem dos casos de uso de negócio do aplicativo e consequentemente dos requisitos funcionais, foi construído um diagrama de casos de uso onde todos os atores envolvidos na aplicação estão presentes e interagem com cada caso de uso possível.

Na Figura 4 podemos ver todas as ações que o usuário pode realizar no sistema.

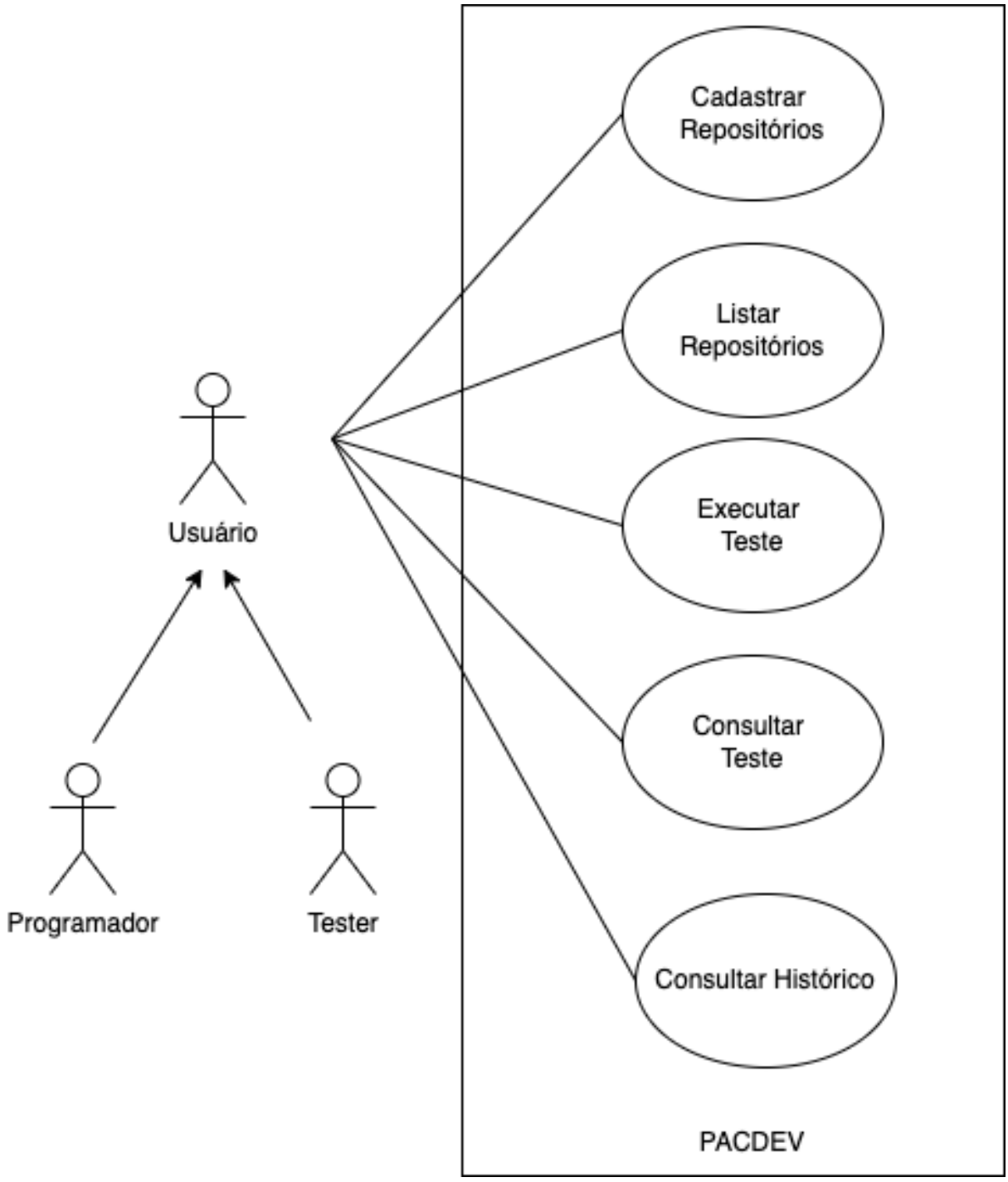

Figura 4 Diagrama de Casos de Uso.

3.2 Requisitos do sistema

A seguir são apresentados os requisitos do sistema. O levantamento dos requisitos foi realizado para garantir que a implementação do projeto realmente solucionasse o problema. Isso foi feito através de análise de sistemas similares e discussões entre orientando e orientador.

Estes requisitos são divididos em requisitos funcionais e não funcionais, conforme exposto a seguir:

3.2.1 Requisitos funcionais

- O sistema deve informar ao usuário sobre os testes sendo executados internamente, assim como resgatar os resultados deles.
- O *website* deve possuir uma *homepage* com objetivo de demonstrar os requisitos para cadastro de um repositório no sistema.
- O sistema deve exibir histórico dos testes enviados.
- O sistema deve possuir um menu que permita o cadastro de projetos já armazenados no GitHub.
- O sistema deve possuir funcionalidade para submissão de testes.

3.2.2 Requisitos não-funcionais

- O sistema deve possuir um sistema de integração seguro com o GitHub, armazenando as credenciais de acesso aos repositórios de forma segura.

- O desempenho do sistema deve ser fluido, a fim de minimizar problemas de *time-out*[2] e maximizar o tempo de retorno dos testes.

- O repositório cadastrado deve fornecer um arquivo de exemplo para os testes.

A seguir, temos uma descrição mais detalhada sobre a escolha das tecnologias e a justificativa para seu uso.

3.3 Tecnologias Empregadas

A seguir são apresentadas as tecnologias empregadas no desenvolvimento do sistema, assim como as respectivas justificativas para seu uso, vale ressaltar que Flask, Python, Angular foram escolhidas pela familiaridade do autor.

3.3.1 Git

Git é uma ferramenta de software usada para registrar o histórico de edições de qualquer tipo de arquivo. Por exemplo: o usuário pode escrever um livro utilizando o Git, o que permite que a escrita se desenvolva aos poucos.

O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para dar suporte ao desenvolvimento do Kernel Linux, mas foi adotado por muitos outros projetos. Cada diretório de trabalho do Git é um repositório com um histórico completo, e o sistema permite o acompanhamento total das revisões, independentemente de acesso a uma rede ou a um servidor central (CHACON; STRAUB, 2014).

[2] Tempo que uma aplicação fica sem responder.

Essa tecnologia foi escolhida por ser o sistema de versionamento mais recomendado para programas, sendo este a base do projeto, onde é os desenvolvedores armazenam os projetos cadastrados.

3.2.2 GitHub Actions

GitHub *Actions* é uma ferramenta voltada para automatizar o desenvolvimento de fluxos de trabalho no GitHub. Essa ferramenta também é utilizada para executar fluxos de execução dentro de um repositório, após adição de um arquivo dentro do repositório. No caso, o GitHub *Actions,* permite realizar ações de CI/CD (*Continuous Integration*/*Continuous Delivery*), sendo estes, respectivamente, ações de teste de um código e a atualização dele dentro do sistema, assim como combinar ações, testes e afins (MICROSOFT, 2022).

Essa tecnologia foi escolhida devido ao seu uso focado em testes, além de sua integração com o git. E ser capaz de executar scripts em python e testes

3.3.3 Container

Um container é considerado como uma forma mais leve e uma boa alternativa para a criação de uma máquina virtual. Um container por si só encapsula uma aplicação com o seu próprio sistema operacional de forma mínima, juntamente com as regras de negócio e a lógica do sistema (DEAL, 2019).

Precisávamos garantir que nosso sistema tivesse um meio seguro de armazenamento de credenciais e execução dos arquivos submetidos. Sendo o container um ambiente isolado, conseguimos garantir esse cenário. Essas credenciais são as utilizadas para acessar os repositórios cadastrados

3.3.4 DynamoDb

Amazon DynamoDB é uma aplicação de banco de dados NoSQL proprietário, totalmente gerenciada, que oferece suporte a estruturas de dados de documentos e valores-chave. O *DynamoDB* é oferecido pela Amazon.com como parte do portfólio da *Amazon Web Services* (AWS, 2022).

Precisávamos de um banco de dados não relacional, este foi escolhido devido poder manipular facilmente os Ids dos documentos. Foi avaliado que um banco de dados relacional não era necessário, uma vez que não existiria relacionamento entre os projetos e outras coisas.

3.3.5 Angular

Angular é uma plataforma de aplicações web de código-fonte aberto e *front-end* baseado em TypeScript, desenvolvido pela Equipe Angular do Google e por uma comunidade de indivíduos e corporações. Angular é uma reescrita completa do AngularJS (SESHADRI, 2018).

3.3.6 Python

Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada e objetos, funcional, de tipagem dinâmica e forte. (Guido van Rossum, 2020)

3.3.7 Flask-Blueprint

Flask é um framework de aplicação web muito popular que deixa quase todas as decisões de design e arquitetura para o desenvolvedor. O Flask Blueprint, ou Blueprint abreviado, tem como objetivo auxiliar na documentação e a estruturação do seu aplicativo Flask, agrupando suas funcionalidades em componentes reutilizáveis para a documentação (VAN ROSSUM, 2020).

Capítulo 4

Desenvolvimento da aplicação

Neste capítulo, é explicada a aplicação em si, os dados que são armazenados pela aplicação e suas relações, e a configuração do banco de dados. Também são explicadas as validações de valores inseridos pelo usuário na aplicação.

4.1 Tipos de usuário

Existem dois tipos de usuário no sistema PACDEV: o usuário padrão, que insere e executa os testes, e o usuário dono da aplicação. O usuário padrão executa os testes na aplicação, ao passo que o usuário dono da aplicação fica responsável por alimentar o sistema com projetos e afins. Esta explicação aqui tem como objetivo apenas facilitar a interpretação das observações no texto sobre quem é o agente das ações.

4.2 API da aplicação

A API foi desenvolvida com base nos conceitos de orientação a objetos e o padrão *Model-View-Controller* (MVC). Todo o código-fonte está disponível para visualização no repositório[3] do GitHub.

[3] PACDEV. Disponível em < https://github.com/arthurghz/PACDEV>. Acesso em 01 de junho de 2021.

Foi utilizado o padrão MVC, que tem como objetivo segregar a estrutura lógica do projeto, permitindo que as funcionalidades sejam isoladas e feitas de forma individual (MVC, 2021).

A camada *model* é feita para definição dos dados, sendo que, neste sistema, seu estado será mantido através do *DynamoDB.*

A camada *view* apresenta os modelos em um formato customizado para o usuário, podendo existir diferenças entre as formas de apresentação do modelo, como por exemplo a customização da data para exibição.

A camada *controller* recebe as entradas do usuário e inicia todos os *scripts*. Nela, podemos fazer a validação, ou seja, uma segunda filtragem da entrada dos dados, sendo a primeira filtragem feita nos *inputs* do usuário.

Um fato importante é que a aplicação executa a criação de um container sempre que um arquivo é submetido, este container é criado contendo apenas o arquivo e o executável da linguagem Python, executa o script Python e verifica se o retorno dele é diferente de 0. Este processo tem como objetivo evitar que arquivos maliciosos sejam enviados para o projeto.

4.3 Rotas de API

Rotas de API são *endpoints* de uma requisição, que atendem requisições web e que são constituídas em duas partes: um método HTTP e um caminho de recurso, por exemplo, GET /projects. É possível definir métodos HTTP específicos para a rota.

Na Figura 5, temos um exemplo de rota da aplicação. Para utilização desta rota, é necessário enviar uma requisição do tipo GET para a rota *“projects”*. Se tudo for enviado conforme previsto e as funções desta rota forem executadas sem intercorrências, será retornada uma mensagem de sucesso com “Status 200”, assim como uma lista de repositórios cadastrados. Caso não

haja projetos cadastrados, ele retornará “404”. A documentação completa da aplicação desenvolvida neste trabalho pode ser encontrada no site do Swagger[4] ou executando a aplicação e acessando seu path raiz.

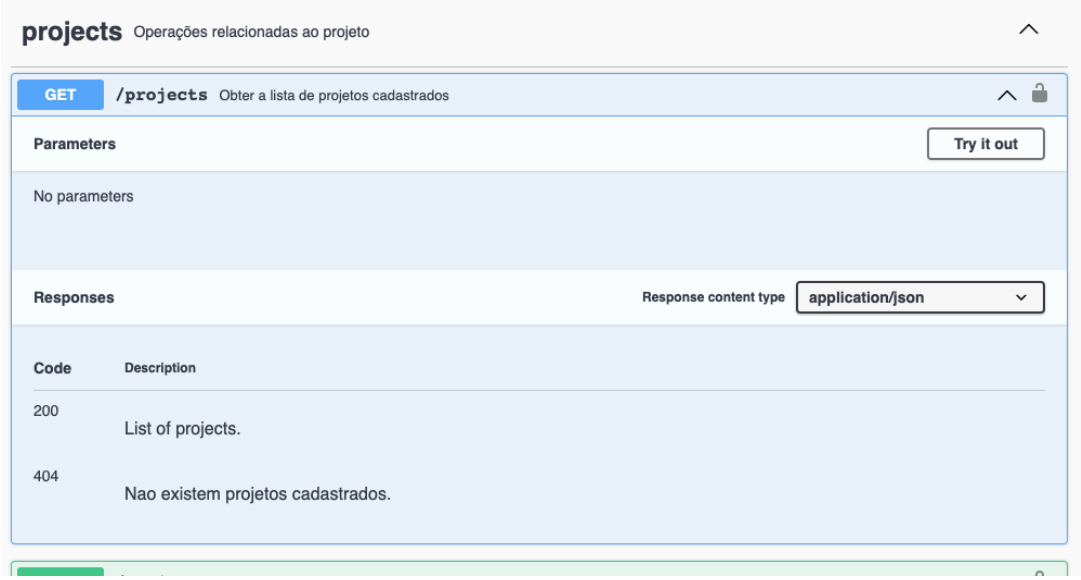

Figura 5 Exemplo de uma rota da API que lista todos os projetos.

4.4 Documentação das rotas

As rotas foram documentadas através do próprio código-fonte utilizando o *framework* Flask-blueprint[5]. Para documentar as rotas faz-se o uso de anotações escritas dentro do código. Por exemplo, ao utilizar anotação @api.response(200, "List of projects), introduz-se na documentação do código que uma das possíveis response daquela rota é o status code 200 com a mensagem "List of projects", como ilustrado no trecho de código-fonte da Figura 6.

[4] PACDEV. Disponível em < http://pacdev.com.br>. Acesso em 01 de julho de 2022.

[5] Flask-blueprint. Disponível em < https://realpython.com/flask-blueprint >. Acesso em 01 de junho de 2021.

```python
@api.route('')
class Projects(Resource):

    @api.response(200, 'List of projects.')
    @api.response(404, 'Nao existem projetos cadastrados.')
    def get(self):
        """Obter a lista de projetos cadastrados"""
        return get_projects()

    @api.response(201, 'Foi cadastrado um novo projeto')
    @api.response(409, 'Projeto já cadastrado ')
    @api.expect(_Project, validate=True)
    @api.doc('Cadastrar novo projeto')
    def post(self):
        """Cadastrar  novo projeto """
        return save_new_project(data=request.json)
```

Figura 6 Exemplo da implementação do Blueprint no código.

4.4.1 Projects - operações relacionadas ao projeto

Aqui serão mostradas as rotas relacionadas ao projeto, a escolha delas se deu através dos requisitos selecionados e os métodos necessários para o projeto.

- GET - /projects - Obter a lista de projetos cadastrados
- POST - /projects - Cadastrar novo projeto
- DELETE - /projects/{github_user}/{repo_name} – Deletar um projeto
- GET - /projects/{github_user}/{repo_name} – Pegar informacao do projeto

4.4.2 Submission - operações relacionadas a submissão do projeto

Partindo do princípio que todo uma das funcionalidades core do sistema é submeter código fonte e também obter arquivos de exemplo para tal, tivemos a necessidade de criação desta rota.

- GET - /submission/{github user}/{reponame} - Obter arquivo de exemplo do repositório
- POST - /submission/{github user}/{reponame} - Criar uma submissão

4.4.3 History - operações relacionadas ao histórico

Uma vez que o projeto é submetido, é deveras importante que possamos coletar todo o histórico de testes de um repositório e um ID especifico, partindo disso nasceu a rota “history”.

- GET - /history/{github user}/{reponame} - Obter o histórico completo do repositório
- GET - /history/{github user}/{reponame}/{id} - Obter o status de execução do {id}
- GET - /history/{github user}/{reponame}/{id}/log/app - Obter log do teste da aplicação
- GET - /history/{github user}/{reponame}/{id}/log/user - Obter log do do teste do usuário

4.5 Implementação do banco de dados

No PACDEV foi utilizado o DynamoDB como banco de dados, uma vez que é um banco no-sql e de fácil implementação, se conectando mais facilmente com nosso servidor, hospedado no mesmo serviço (Amazon WS). Dentro da nossa aplicação, existe um arquivo de configuração, onde definimos as credenciais necessárias para conexão. Após a conexão, não é necessário a criação de nenhum esquema SQL, uma vez que o banco não é relacional e os dados inseridos definem seu modelo.

Cada projeto é salvo em um único documento, composto pelo nome do usuário do GitHub, nome do repositório e a definição de ID do documento. A Figura 7 apresenta um exemplo deste documento, em que seu ID é "arthurghz/pacdev".

Figura 7 Exemplo do conteúdo de um documento da aplicação.

4.7 Primeiro contato com o sistema

Quando um usuário entra na aplicação, irá se deparar com a tela de instruções sobre como utilizar o sistema, mostrada na Figura 8. Além disso, são exibidos todos os requisitos exigidos para cadastramento de um projeto (detalhados em sessão subsequente deste trabalho), assim como a estrutura de arquivos que o usuário deve seguir.

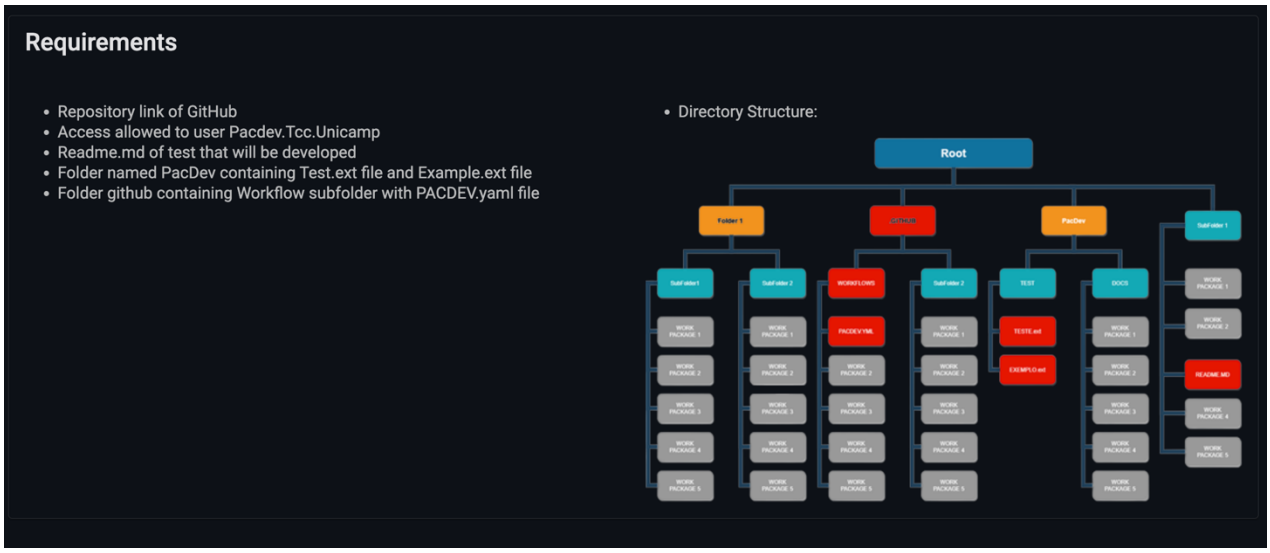

Figura 8 Página inicial do sistema PACDEV.

4.8 Funcionalidades

Nesta seção é explicada toda a lista de funcionalidades da aplicação, os dados que as rotas requerem e demais configurações, assim como seu comportamento.

4.8.1 Cadastro de Código

Essa funcionalidade não possui pré-requisito de acesso, ou seja, é permitido a qualquer desenvolvedor o cadastramento de uma aplicação (Figura 9). Porém, nem todas as aplicações são aceitas.

As informações necessárias para cadastro do código-fonte são:

- Nome do repositório no GitHub;
- Nome do usuário do repositório;
- Acesso liberado no repositório para o usuário Pacdev.Tcc.Unicamp;
- *Readme.md* contendo o *Enunciado no Teste* a ser desenvolvido para tal aplicação;
- Pasta denominada PacDev com os arquivos chamados *Teste.py* - arquivo responsável pelo teste da aplicação e *Exemplo.py* - Arquivo a ser enviado para o usuário como exemplo, contendo um explicativo das funções e alguns exemplos*;*
- Pasta *github* contendo um diretório *Workflow* com o arquivo PACDEV.yaml - Arquivo responsável por definir a ordem dos testes.

Uma vez que a aplicação é cadastrada no PACDEV, é realizada uma verificação para saber se o repositório no GitHub atende à estrutura de pastas apresentada na Figura 10, sendo as cores em vermelho as pastas e arquivos essenciais.

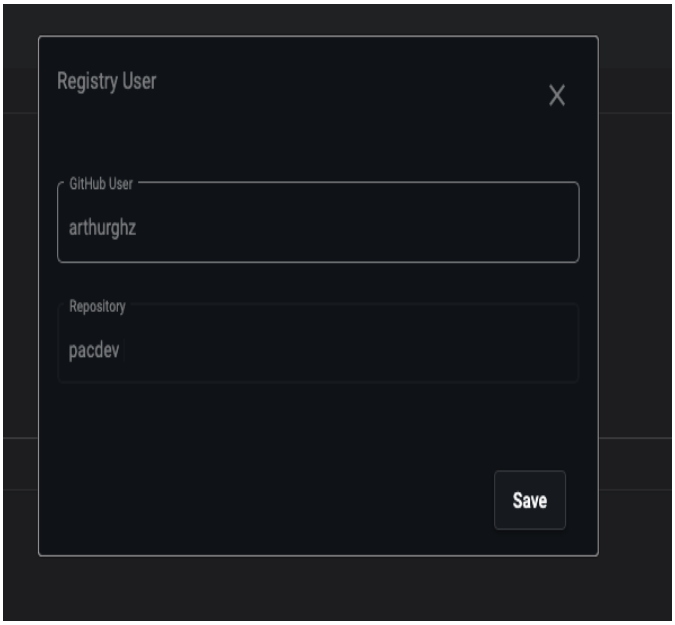

Figura 9 cadastramento de uma aplicação.

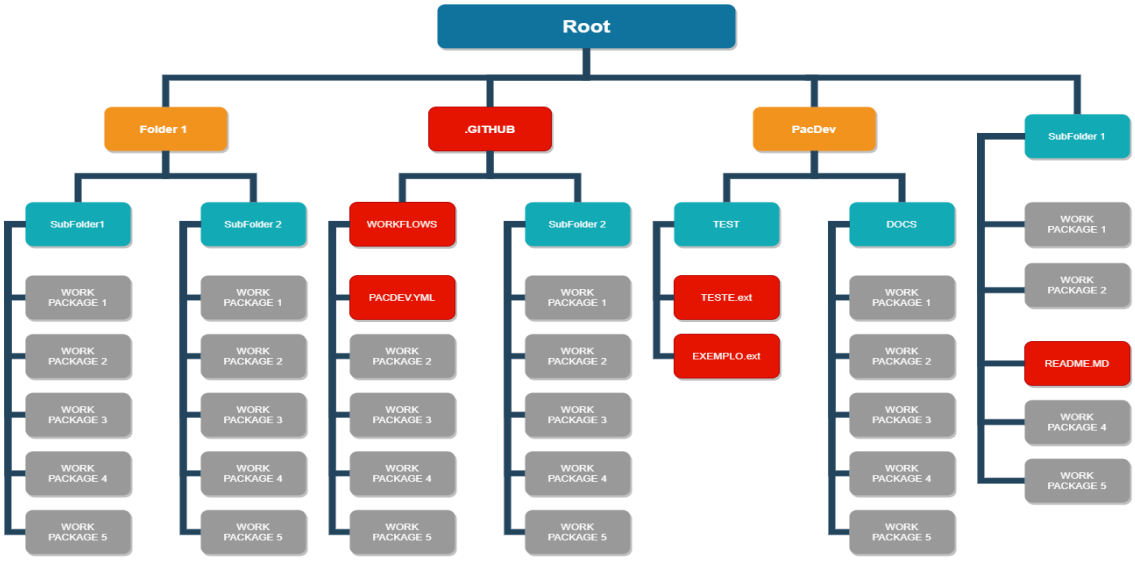

Figura 10 Exemplo de estrutura de diretórios e arquivos exigidos pelo PACDEV. Fonte: Pacdev.

Em seguida, com os diretórios validados, a aplicação estará vinculada ao PacDev com sucesso.

4.8.2 Histórico de Testes

O histórico de submissões (Figura 11) é outro fator importante dentro do software. Uma vez que o usuário queira ver o histórico de testes executados em uma aplicação, esse histórico é resgatado diretamente do GitHub, exibindo o ID de cada job (execução) e, quando o usuário seleciona um job ID, ele é direcionado para a respectiva tela (Figura 12).

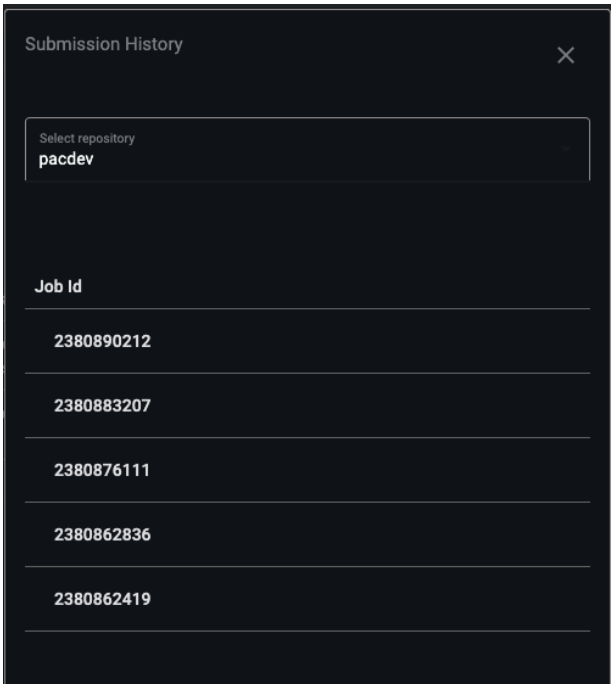

Figura 11 Histórico de submissões.

4.8.3 Visualização da execução de um teste

No resultado do teste (Figura 12 e Figura 13), é possível ver com detalhes a hora de execução, o nome do arquivo executado, o nome da função escrita no teste, dos parâmetros executados no teste e se o teste foi efetuado com sucesso ou não.

Caso o usuário queira ver o teste completo da aplicação, assim como seus arquivos de *log*, resultado da construção do *container* e afins, ele pode

efetuar o *download* de um arquivo compactado com todos os arquivos de *log*, e abri-lo localmente

O arquivo, baixado em formato zip, conterá os arquivos apresentados na (Figura 14). Este arquivo contém as etapas necessárias até a execução do teste, sendo a Etapa 1 a configuração inicial do ambiente e criação do *container*, a Etapa 2 a coleta dos arquivos do repositório a Etapa 3 o *download* dos arquivos para um *branch* alternativo em que o teste é executado, e as etapas 4 e 5 a instalação do Python e suas dependências. Por fim, a Etapa 6 trata do resultado do teste propriamente dito, assim como seu *log* mais completo.

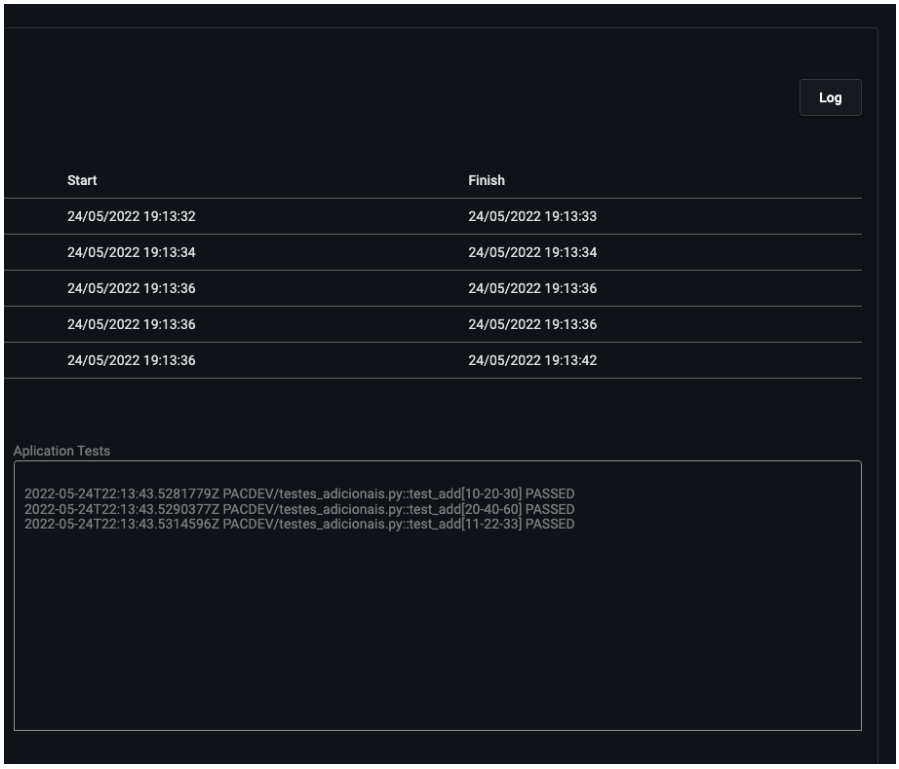

Figura 12 Lado direito da tela de visualização de testes do PACDEV.

4.8.4 Visualização de explicação do repositório

O usuário que irá executar um teste precisa de uma explicação básica sobre o funcionamento da aplicação a ser avaliada. Para isso, é mandatório que o dono da aplicação escreva um arquivo texto explicativo (*readme*) sobre

o funcionamento desta, como ilustrado na Figura 15. Este *readme*, deve orientar o usuário sobre o nome das funções contidas na aplicação e seu comportamento esperado. Além disso, o usuário desenvolvedor dos testes pode recorrer também aos comentários contidos no arquivo de exemplo de testes que deve ser incluído junto à aplicação a ser avaliada, como ilustrado na Figura 16.

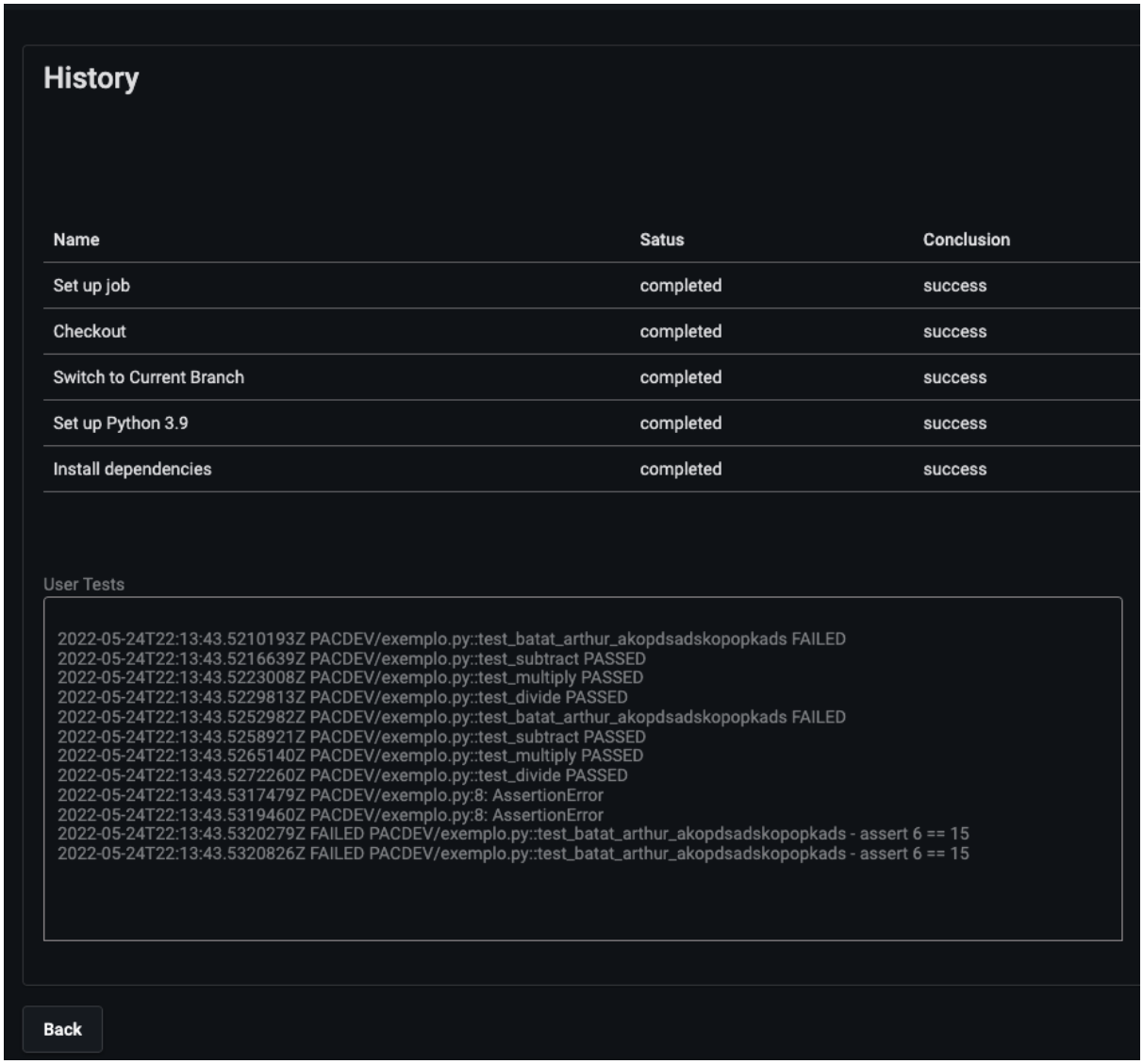

Figura 13 Lado direito da tela de visualização de testes do PACDEV.

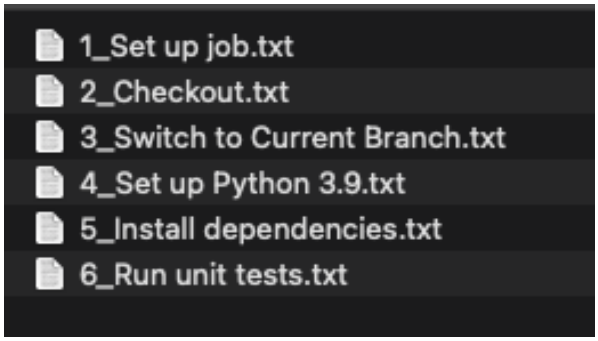

Figura 14 Conteúdo do arquivo de LOG que pode ser baixado pelo usuário.

Figura 15 Exemplo do arquivo README desenvolvido pelo usuário proprietário da aplicação a ser avaliada.

```python
import pytest

from myapp.mymodule.funcoes import *


@pytest.mark.easy_operation
def test_add():
    #  Essa funcao deve ser usada para soma,
    #recebe 2 parametros (int,int)
    # e tem como retorno o resultado da soma
    assert add(4, 8) == 12

@pytest.mark.easy_operation
def test_subtract():
    #  Essa funcao deve ser usada para subtracao,
    #recebe 2 parametros (int,int)
    #e tem como retorno o resultado da subtracao
    assert subtract(3, 6) == -3

@pytest.mark.difficult_operation
def test_multiply():
    #  Essa funcao deve ser usada para multiplicao,
    #recebe 2 parametros (int,int) e
    #tem como retorno o resultado da multiplicao deles
    assert multiply(4, 5) == 20

@pytest.mark.difficult_operation
def test_divide():
    #  Essa funcao deve ser usada para divisao,
    #recebe 2 parametros (int,int)
    # e tem como retorno o resultado da soma
    assert divide(56, 8) == 7
```

Figura 16 Arquivo de exemplo de submissão de código de testes.

4.8.5 Submissão de Código

Aqui é onde o sistema PACDEV entrega seu real valor. Cada aplicação cadastrada no sistema disponibiliza para o usuário um arquivo de exemplo, ilustrado na Figura 17, que pode ser acessado ao clicar no botão "*Download Example test*". Além disso, há um arquivo de *Teste* onde o desenvolvedor deve escrever o código de testes a ser submetido no sistema. Após submetido o arquivo de teste, o usuário será redirecionado para a tela de *Log*.

Após a submissão do arquivo pelo usuário, a aplicação executa a criação de um *container* para validar o arquivo e, à *posteriori*, executa uma adição do arquivo no repositório GitHub do usuário, deixando que o GitHub *actions* se encarregue de todo o processo de execução do teste. Na Figura 18, é possível verificar o trecho de código responsável pela adição do arquivo no GitHub: temos uma verificação se o repositório está cadastrado em nosso sistema na quarta linha representada na imagem, caso não exista, ele retorna uma mensagem de "Repositório nao cadastrado" e o *status code* 404, caso o repositório esteja cadastrado segue-se o fluxo normal, sendo executados comandos do GIT dentro do próprio repositório, o que demonstra a necessidade de acesso concedido ao PACDEV.

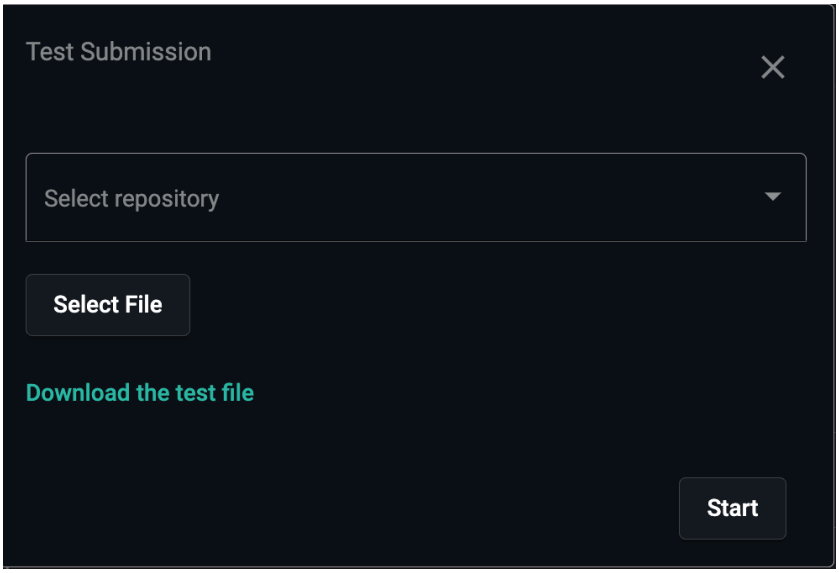

Figura 17 Tela de submissão de códigos de teste.

```python
def create_submssion(github_user,repo_name,file):
    try:
        repo_path= f'{REPOSITORY_PATH}/{github_user}/{repo_name}/PACDEV'
        if not os.path.isdir(repo_path) :
            return {'message': "Repositorio nao encontrado internamente" }, 404
        file.save(os.path.join(repo_path, secure_filename(file.filename)))
        repo = Repo(f"{REPOSITORY_PATH}/{github_user}/{repo_name}/.git")
        repo.git.add(update=True)
        repo.index.commit("Submissao Efetuada com sucesso")
        origin = repo.remote(name='origin')
        origin.push()
        last_id = get_history(github_user,repo_name)
        return {"Status": "Sucess",
                "last_commit": str(repo.head.commit),
                "id_workflow": last_id[0]}, 200
    except Exception as e:
        return {'message': str(e) }, 500
```

Figura 18 Código responsável pela submissão do arquivo no GitHub.

Capítulo 5

Conclusões

No processo de desenvolvimento de um software existem diversas etapas para mensurar sua qualidade. Algumas dessas etapas são pouco visíveis e processuais, sendo comum e passível de mensuração a quantidade de testes submetidos a aplicação e quantos testes diferentes existem para cada funcionalidade. Este processo de escrita de testes se dá geralmente no começo do processo de desenvolvimento. Neste contexto, é recomendado que se escrevam testes unitários para a aplicação e, posteriormente, testes de integração, a fim de verificar como as funcionalidades se comunicam entre si. Para simplificar este problema, o presente trabalho propôs a criação da aplicação PACDEV, permitindo que pessoas com conhecimento em linguagens de programação executem testes em aplicações de terceiros, com o intuito de incrementar a quantidade de testes.

A aplicação foi desenvolvida usando alguns conteúdos abordados durante a graduação do autor, como Python e/ou boas práticas de teste. Python assim como muitas linguagens de programação, aceita a escrita de testes voltados para as aplicaões escritas usando a própria linguagem, tendo como seu principal objetivo permitir que um usuário que não é o autor do código, introduza testes em um código já pré-existente.

Diante do que foi supracitado, os resultados obtidos atingiram os objetivos propostos neste projeto. Dessa forma, foi criada uma aplicação gratuita que permite que usuários com acesso à internet executem testes em aplicações (sem o conhecimento prévio de código), além de possibilitar que os desenvolvedores donos da aplicação coletem estes testes através do histórico de seu GitHub e melhorem sua aplicação com os testes.

Além do mais, durante o projeto foram utilizadas tecnologias gratuitas e o código-fonte gerado está disponível no GitHub. Dessa forma, existe a possibilidade de desenvolvimento de novas funções para que a aplicação se adeque a novas necessidades de processos e testes.

No mais, para auxiliar desenvolvedores que venham a utilizar o que foi gerado neste projeto, tudo o que foi produzido está documentado no repositório e protótipo citados ao longo deste documento, incluindo a arquitetura, infraestrutura e detalhes do código da aplicação.

Referências

AWS, Produtos. Amazon DynamoDB serviço de banco de dados NoSQL rápido e flexível para performance abaixo de 10 milissegundos em qualquer escala. 2022. Disponível em: <https://aws.amazon.com/pt/dynamodb/>. Acesso em: 25 de maio de 2022.

BECK, K. Test Driven Development. By Example (Addison-Wesley Signature): Addison-Wesley Longman, Amsterdam, 2002.

CHACON, Scott; STRAUB, Ben. Pro Git. Everything you need to know about Git. 2. ed., 2014. Disponível em: <https://git-scm.com/book/en/v2>. Acesso em: 25 de maio de 2022.

CNN Brasil [homepage na internet] Procura-se: busca por profissionais de TI com menos experiência cresce no 1º sem. Disponível em: <https://www.cnnbrasil.com.br/business/procura-se-busca-por-profissionais-de-ti-com-menos-experiencia-cresce-no-1-sem/>. Acesso em: 16 de outubro de 2021.

DEAL. O ambiente devops e a conteinerização são passos importantes para melhorar a eficiência das entregas e deploy das aplicações. 2019. Disponível em: <https://www.deal.com.br/blog/o-ambiente-devops-e-a-containerizacao/>. Acesso em: 25 de maio de 2022.

HACKERRANK [homepage na internet]. Disponível em < https://www.hackerrank.com>. Acesso em: 01 de junho de 2021.

LEETCODE [homepage na internet]. Disponível em <https://leetcode.com>. Acesso em: 01 de junho de 2021.

MICROSOFT. GitHub. 2022. Disponível em: <https://github.com>. Acesso em: 16 de outubro de 2021.

MVC. [MVC architecture]. Disponível em: < https://subscription.packtpub.com/book/application_development/9781785889196/10/ch10lvl1sec88/mvcarchitecture#:~:text=MVC%20is%20a%20widely%20used,data%2C%20and%20update%20the%20view>. Acesso em: 10 de dezembro de 2021.

SESHADRI, Shyam. Angular: Up and Running: Learning Angular, Step by Step. 2 ed. Sebastopol: O'Reilly Media, 2018.

VALENTE, Marco Tulio. Engenharia de Software Moderna. Princípios e Práticas para Desenvolvimento de Software com Produtividade, 1 ed., 2020. Disponível em: <https://engsoftmoderna.info/cap8.html>. Acesso em: 25 de maio 2022.

VAN ROSSUM, Guido. Learning Python: Crash Course Tutorial, ed . 2022.