Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Desenvolvimento de Software com Interface Gráfica em Raspberry PI 4 para controle de uma máquina de cisalhamento

Autor: Ivan Diniz Dobbin

Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF

2022

Ivan Diniz Dobbin

Desenvolvimento de Software com Interface Gráfica em Raspberry PI 4 para controle de uma máquina de cisalhamento

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF

2022

Ivan Diniz Dobbin

Desenvolvimento de Software com Interface Gráfica em Raspberry PI 4 para controle de uma máquina de cisalhamento/ Ivan Diniz Dobbin. 2022.

108 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2022.

1. Sistemas Embarcados. 2. Raspberry Pi 4. 3. Desenvolvimento de interface 4. Máquina de ensaio de cisalhamento direto I. Sampaio, Renato Coral. II. Universidade de Brasília. Faculdade Unb Gama.

CDU 02:141:005.6

Ivan Diniz Dobbin

Desenvolvimento de Software com Interface Gráfica em Raspberry PI 4 para controle de uma máquina de cisalhamento

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 28 de setembro de 2022:

Prof. Dr. Renato Coral Sampaio
Orientador

Prof. Dr. Tiago Alves da Fonseca
Convidado 1

Prof. Dr. Robinson Andrés Giraldo Zuluaga
Convidado 2

Brasília, DF
2022

Este trabalho é dedicado a todos aqueles que sonham em se tornarem cientistas um dia.

Agradecimentos

Eu agradeço a paciência e auxílio dos meus pais e irmãos durante o período de desenvolvimento do TCC. Também agradeço ao meu orientador, professor Renato, e ao professor Guillermo por me auxiliarem em diversas etapas do trabalho. Por último, agradeço o professor Robinson que me auxiliou na parte de priorização de requisitos.

"A vida não é um jogo de sorte.
Se você quer vencer, trabalhe duro."
(Ahiru no Sora, mangá de basquete)

Resumo

O solo é uma parte importante para a engenharia civil, sendo utilizado como material de construção e suporte para fundações estruturais. O seu estudo deu origem a mecânica dos solos, onde se pesquisa suas propriedades físicas e comportamentais. O presente trabalho tem como objetivo realizar um estudo sobre a aplicação de *smart retrofitting* em uma máquina de ensaio de cisalhamento direto que se encontra no Campus Darcy Ribeiro. Assim, será desenvolvido um *software* embarcado com interface gráfica, e em união com uma camada de controle, a máquina de cisalhamento direto poderá ser controlada. Requisitos foram levantados e priorizados para a criação dos protótipos e a partir desses resultados foi possível desenvolver uma arquitetura adequada que solucionasse o problema. Utilizando essa arquitetura, criou-se o *software* de interface para *Raspberry Pi 4* em uma tela sensível ao toque (*touchscreen*) e investigou-se a possibilidade de acesso remoto com base na Internet das Coisas (IoT). Por fim, foi possível alcançar os objetivos estabelecidos e responder positivamente a pergunta inicial do trabalho.

Palavras-chave: Sistemas Embarcados. Raspberry Pi 4. UX/UI. Cisalhamento. Máquina de ensaio de cisalhamento direto. Retrofit.

Abstract

Soil is an important part of civil engineering, being used as a construction and support for structural foundations. Its study gave rise to the mechanics of soils, where their physical and behavioral properties are researched. The present work aims to carry out a study on the application of smart retrofitting in a direct shear testing machine located on Campus Darcy Ribeiro. Thus, an embedded software with a graphical interface will be developed, and in union with a control layer, the direct shear machine can be controlled. Requirements have been raised and prioritized for the creation of prototypes and from these results it was possible to develop an appropriate architecture that would solve the problem. Using this architecture, a software interface was created for a Raspberry Pi 4 on a touchscreen and the possibility of remote access based on the Internet of Things (IoT) was investigated. Finally, it was possible to reach the established objectives and respond positively to the initial question of the work.

Key-words: Embedded systems. Raspberry Pi 4. UX/UI. Shear. Direct shear testing machine. Retrofit.

Lista de ilustrações

Lista de tabelas

Lista de quadros

Lista de abreviaturas e siglas

ASIC	*Application-specific Integrated Circuit*
DSP	*Digital Signal Processor*
DWS	*Digital Weight Scale*
FPGA	*Field Programmable Gate Array*
GUI	*Graphical User Interface*
IoT	*Internet of Things*
IP	*Intellectual Property*
PCB	*Printed circuit board*
PSP	*Personal Software Process*
RAM	*Random Access Memory*
RDBMS	*Relational Database Management System*
ROM	*Read Only Memory*
SBC	*Single Board Computer*
SD	*Secure Digital*
TCC	Trabalho de Conclusão de Curso
UI	*User Interface*
UX	*User Experience*
VCS	Sistema de Controle de Versões

Lista de símbolos

mm	milímetros
kPa	kilopascal
τ	letra grega tau
σ	letra grega sigma
$\sum$	somatório
GB	Gigabyte
MB	Megabyte
kB	kilobyte
ms	milissegundo

Sumário

1 Introdução

1.1 Contextualização

O solo é uma parte importante para a engenharia civil, sendo utilizado como material de construção e suporte para fundações estruturais. Seu estudo é realizado para garantir qualidade e segurança nas construções (PINTO, 2006). Pode-se definir solo da seguinte forma:

> Para fins de engenharia, solo é definido como um agregado não cimentado de grãos minerais e matéria orgânica decomposta (partículas sólidas), com líquido e gás preenchendo os espaços vazios existentes entre as partículas sólidas (SOBHAN, 2014).

A mecânica dos solos é uma área que estuda as propriedades físicas e o comportamento do solo ao ser submetido a variados tipos de tensão, segundo Pinto (2006) e Sobhan (2014). O estudo dos solos, como afirma Pinto (2006), teve desenvolvimentos marcantes em séculos passados, com os trabalhos de Charles Coulomb (1736-1806), William Rankine (1820-1872) e Henry Darcy (1803-1858).

A engenharia dos solos surge pela aplicação dos fundamentos dessa mecânica em problemas práticos (SOBHAN, 2014). Um outro termo relevante, para o presente trabalho, é engenharia geotécnica, com a seguinte definição apresentada por Sobhan (2014):

> A engenharia geotécnica é a subdisciplina da engenharia civil que estuda materiais naturais encontrados próximos à superfície da terra.

Umas das preocupações com o solo são suas condições de ruptura ou deslizamento. Na maior parte das vezes, rupturas são causadas por fenômenos de cisalhamento e os dois principais critérios para detectá-las são os critérios de Mohr e Coulomb (PINTO, 2006).

Existem variados tipos de ensaios que podem ser realizados com o solo, como: ensaio de cisalhamento direto, ensaio triaxial, ensaio de cisalhamento simples, ensaio triaxial de deformação plana e ensaio de cisalhamento anular ou *ring shear* (SOBHAN, 2014). O foco será no ensaio de cisalhamento direto (explicado com mais detalhes na Seção 2.2), pois é este o ensaio realizado pela máquina de cisalhamento do laboratório (Fig. 1).

O termo *smart retrofitting* significa melhorar a acurácia, velocidade, manutenibilidade e usabilidade de um equipamento, aplicando elementos da quarta revolução industrial, como sistemas embarcados e Internet das Coisas (IoT) (AL-MAEENI et al., 2020). Os ensaios mencionados anteriormente, necessitam, na maioria das vezes, que a medições sejam realizadas por um computador externo, ou até mesmo manualmente por uma

pessoa. Assim, a aplicação de *smart retrofitting* seria uma maneira de modernizar estas máquinas.

Este projeto faz parte de um trabalho em conjunto com Pereira (2022), em que o objetivo final é realizar o *retrofit* de uma máquina de cisalhamento (Fig. 1). O projeto de Pereira (2022) teve foco na parte de eletrônica, enquanto que o projeto do aluno Ivan Diniz (trabalho atual) terá foco no desenvolvimento de um *software* de interface para operação da máquina de cisalhamento.

1.2 Justificativa

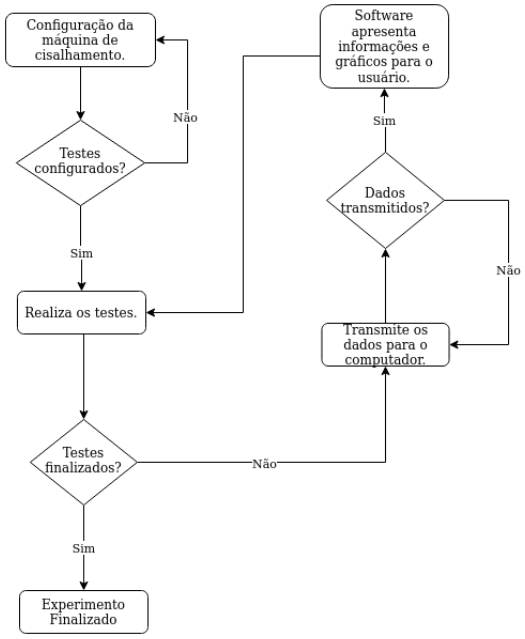
No campus Darcy Ribeiro da Universidade de Brasília, mais especificamente no Laboratório de Infraestrutura Rodoviária (INFRALAB), existe uma *Geotest Direct-Residual Shear Machine* ou Máquina de Cisalhamento Direto-Residual Geotest modelo *S2215A*, que realiza o ensaio de cisalhamento direto com variados tipos de materiais. A Figura 1 mostra uma foto dessa máquina e a Fig. 3 apresenta o sistema completo. A Figura 2 exibe de maneira simplificada o funcionamento do sistema, no qual após a realização dos testes pela máquina de ensaio de cisalhamento direto, os dados capturados são transmitidos para o computador que disponibiliza a visualização de gráficos e outras informações relevantes para um usuário, através de um *software*. Algumas dessas informações são o deslocamento em milímetros (mm) e a carga em Kilopascal (kPa).

Figura 1 – Máquina de cisalhamento

Fonte: Elaborado pelo autor, 2022

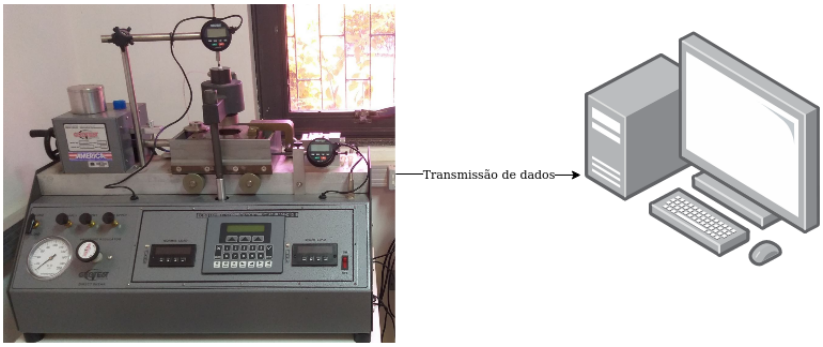
Atualmente são utilizadas 2 máquinas para o funcionamento do sistema, tornando o sistema mais complexo e, assim, dificultando o seu manuseio. Para facilitar a utilização, propõe-se a integração do sistema em apenas 1 máquina, usando-se conceitos de sistemas embarcados. De acordo com White (2012) um sistema embarcado é um sistema computacional de propósito específico, como um controle remoto ou um sistema computacional de um avião. Este tipo de sistema lida com diversas limitações, como memória, tamanho dos componentes, velocidade de processamento e energia. Um computador pessoal não é um sistema embarcado, é um sistema de propósito geral, pois com ele pode-se fazer diversas ações, desde visualizar vídeos na internet até criar *softwares*, ou seja, ele não foi criado

Figura 2 – Funcionamento básico do sistema



Fonte: Elaborado pelo autor, 2022

Figura 3 – Sistema completo



Fonte: Elaborado pelo autor, 2022

para ser utilizado com um propósito único. O tópico sistemas embarcados é explicado com mais detalhes na Seção 2.4.

1.3 Problema de Pesquisa

O trabalho proposto por Pereira (2022) sugere a utilização de uma tela sensível ao toque em conjunto com uma *Raspberry Pi* que substituirá o computador e realizará todas as operações em apenas 1 máquina. Sendo assim, para este trabalho se faz a seguinte pergunta:

- Como realizar o desenvolvimento de um software para um tela sensível ao toque em conjunto com uma *Raspberry Pi* que atenda a maioria dos requisitos necessários para o bom uso da máquina de ensaio de cisalhamento direto?

1.4 Objetivos Gerais

O objetivo geral deste trabalho é implementar um software que permita a utilização da máquina de ensaio de cisalhamento direto por meio de uma tela sensível ao toque conectada a uma *Raspberry Pi*. A Figura 4 mostra um rascunho de como ficará a máquina, na qual a presença de um computador externo não é mais necessário.

Figura 4 – Sistema com tela sensível ao toque

Fonte: Elaborado pelo autor, 2022

1.4.1 Objetivos Específicos

Os objetivos específicos do projeto são:

- Realizar o levantamento de requisitos, para melhor atender as necessidade do cliente.
- Desenvolver um protótipo para validação do design.
- Entender qual arquitetura pode ser aplicada.
- Pesquisar qual linguagem será utilizada para desenvolver o projeto.
- Pesquisar qual ferramenta será utilizada para construir o software.
- Investigar a possibilidade da inclusão de acesso remoto ao software que será construído.
- Investigar se o acesso remoto traz benefícios ao uso do equipamento.
- Investigar se as mudanças propostas trazem benefícios à situação atual.

- Implementar os requisitos levantados.
- Testar o código.

1.5 Metodologia

Neste trabalho o autor decidiu utilizar a metodologia *Scrum Solo* com *Kanban*. O *Scrum Solo* surgiu da união do *Personal Software Process (PSP)* com *Scrum*, realizando a junção de aspectos de gerenciamento que auxiliam na garantia da qualidade do produto do *PSP*, com artefatos do *Scrum* como *Sprint, Product Backlog e Sprint Backlog* que auxiliam na organização e planejamento do projeto (PAGOTTO et al., 2016). O *Kanban* foi escolhido em conjunto com *Scrum Solo* para auxiliar na visualização das tarefas e organização do trabalho. Mais detalhes sobre a metodologia escolhida podem observados no Capítulo 3.

1.6 Organização do Trabalho

- Capítulo 2 (Referencial Teórico): descreve conceitos importantes para o entendimento do trabalho.
- Capítulo 3 (Metodologia): descreve a metodologia utilizada no projeto.
- Capítulo 4 (Suporte Tecnológico): apresenta as ferramentas e tecnologias que auxiliaram e que auxiliarão no desenvolvimento do trabalho.
- Capítulo 5 (Proposta de solução): apresenta a proposta de solução do projeto
- Capítulo 6 (Resultados): discute sobre os resultados alcançados.
- Capítulo 7 (Conclusão): realiza a conclusão do trabalho, informando se os objetivos foram concluídos, as dificuldades que ocorreram e as limitações do trabalho atual.

2 Referencial Teórico

Este capítulo tem como objetivo explicar termos essenciais para o melhor entendimento do trabalho. Inicia-se explicando conceitos de resistência do solo e seus critérios de ruptura, o funcionamento da Máquina de Cisalhamento Direto-Residual Geotest modelo *S2215A*, e o ensaio de cisalhamento direto que ela realiza. Segue-se com o esclarecimento do que são sistemas embarcados, sua categorização e por fim suas limitações e dificuldades. Depois é explicado o conceito de *retrofit* e sua aplicação neste trabalho e o que é uma *Raspberry Pi 4*. Por fim é apresentado o que é *Design* de interface e como ela é afetada por sistemas embarcados e aplicação de IoT.

2.1 Resistência do solo e seus critérios de ruptura

Para melhor compreensão sobre o funcionamento da máquina de cisalhamento (Fig.3), será realizada uma explicação sobre a resistência do solo e seus critérios de ruptura.

Umas das preocupações com o solo são suas condições de ruptura ou deslizamento. Na maior parte das vezes, rupturas são causadas por fenômenos de cisalhamento, 2 exemplos dados por Pinto (2006) são:

- Sapata de fundação carregada até a ruptura.
- Escorregamento de um talude (Figura 5).

Figura 5 – Escorregamento de um talude

Fonte: (MARINHO, 2020)

Um conceito que trata sobre isso é o de resistência ao cisalhamento. Foram encontradas 2 definições:

> A resistência ao cisalhamento de uma massa de solo é a resistência interna por área específica que essa massa pode oferecer para resistir a rupturas e a deslizamentos ao longo de qualquer plano em seu interior (SOBHAN, 2014).

> A resistência ao cisalhamento de um solo pode ser definida como a máxima tensão de cisalhamento que o solo pode suportar sem sofrer ruptura, ou a tensão de cisalhamento do solo no plano em que a ruptura estiver ocorrendo (PINTO, 2006) .

Os critérios de ruptura, de acordo com Pinto (2006), podem ser definidos como: elaborações que procuram refletir as condições em que acontece a ruptura dos materiais. Os critérios de Couloumb e de Mohr são os dois critérios de ruptura que melhor descrevem o comportamento do solo.

De acordo com Pinto (2006), pode-se definir o critério de Coulomb como:

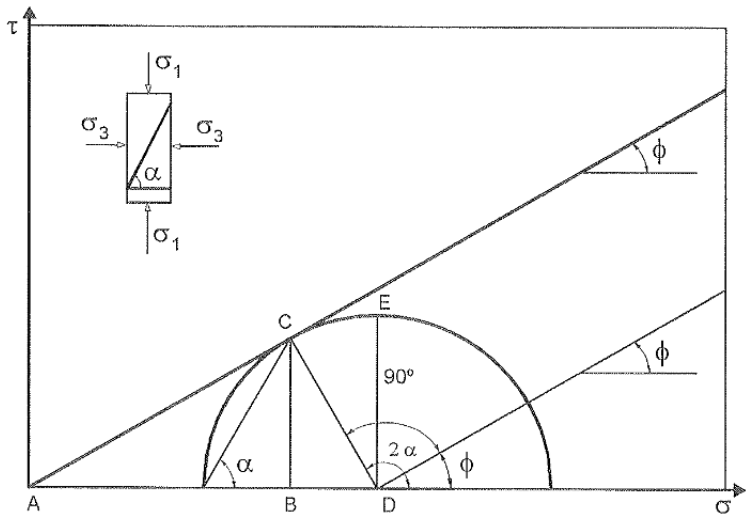
> Não há ruptura se a tensão de cisalhamento não ultrapassar um valor dado pela expressão $c + f \times \sigma$, sendo c e f constantes do material e σ a tensão normal existente no plano de cisalhamento (PINTO, 2006).

e o critério de Mohr como:

> Não há ruptura enquanto o círculo representativo do estado de tensões se encontrar no interior de uma curva, que é a envoltória dos círculos relativos a estados de ruptura, observados experimentalmente para o material (PINTO, 2006).

Como envoltórios curvos são de difícil aplicação, geralmente substitui-se as curvas por linhas retas (PINTO, 2006). Dando origem ao chamado critério de Mohr-Coulomb, apresentado na Fig. 6.

Figura 6 – Critério de Mohr-Coulomb



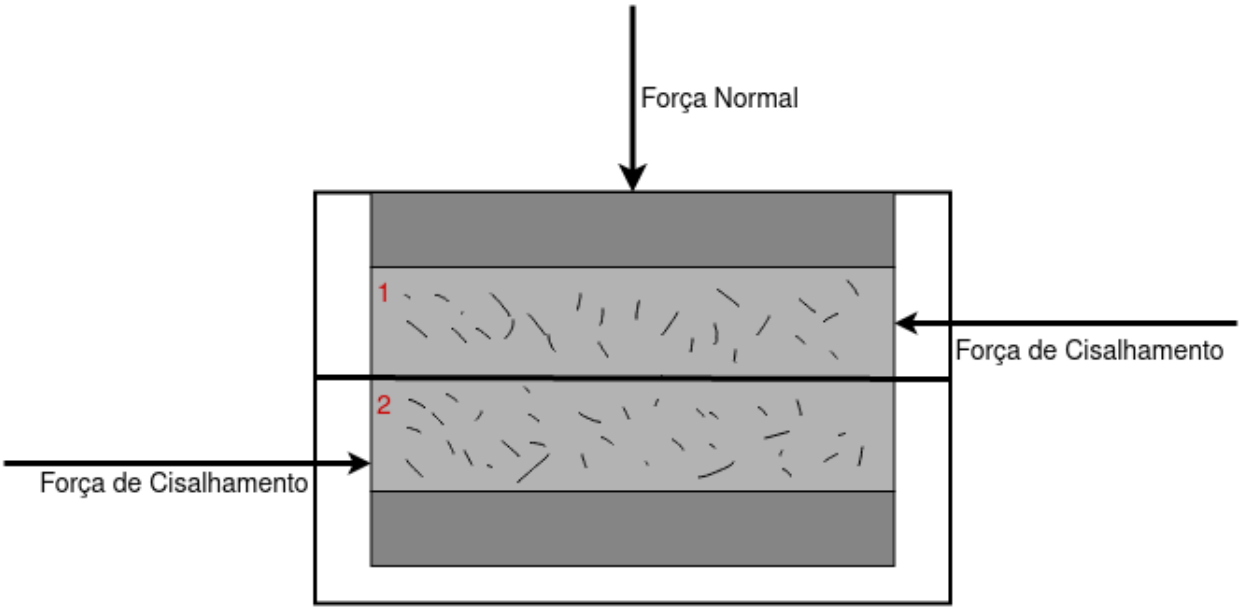
Fonte: (PINTO, 2006)

2.2 Ensaio de cisalhamento direto

O ensaio de cisalhamento tem como objetivo determinar os parâmetros da resistência ao cisalhamento (SOBHAN, 2014).

O ensaio de cisalhamento direto consiste em colocar o corpo de prova ou amostra do solo na caixa de cisalhamento. Inicialmente é aplicado a força normal no topo da caixa para realizar o adensamento. Após a fase de adensamento inicia-se o cisalhamento, no qual a força de cisalhamento é aplicada ao deslocar horizontalmente uma metade da caixa sobre a outra. O experimento é finalizado quando ocorre a ruptura da amostra. É importante lembrar que durante a fase de cisalhamento a amostra de solo continua sofrendo a carga da força normal (SOBHAN, 2014) e (PINTO, 2006). A Figura 7 mostra a caixa do ensaio de cisalhamento direto, onde as metades das caixa ficam representadas pelos números 1 e 2.

Figura 7 – Ensaio de cisalhamento direto



Fonte: Elaborado pelo autor, 2022

2.3 Máquina de Cisalhamento Direto-Residual Geotest modelo *S2215A*

A Máquina de Cisalhamento Direto-Residual Geotest modelo *S2215A* (Fig. 1) é responsável por realizar o ensaio de cisalhamento direto, mencionado na Seção 2.2. Essa máquina passará por um processo chamado *retrofit*, mencionado na Seção 1.2. Esta seção tem como objetivo apresentar mais detalhes sobre este maquinário.

De acordo com GEOTEST (2002) esta máquina está em conformidade com os padrões rígidos estabelecidos pelo Exército dos EUA e ASTM 03080 AASHTO T236. Ela

combina a precisão de carga morta com a conveniência e flexibilidade da carga normal aplicada pneumaticamente.

Aplicando a carga de consolidação pneumaticamente significa que praticamente qualquer tamanho de amostra, até 10.16 cm, pode ser acomodado sem alterar os conjuntos de pesos. A pressão para uma carga selecionada pode ser predefinida e a carga pode ser aplicada rapidamente, quando abre-se a válvula. Para garantir manutenção precisa da carga em toda a faixa de 8.8964 a 8896.44 Newtons, dois pistões são empregados com um razão de pressão efetiva de 1:10. Ambos os pistões são vedados com vedações de rolamento de fricção muito baixa (GEOTEST, 2002).

A carga é aplicada e medida diretamente por uma célula de carga tipo panqueca de alta precisão. Medindo a carga normal real em vez de definir a carga como feito em outras máquinas pneumáticas, todos os problemas causados pela conformidade do sistema são eliminados. O cisalhamento é feito a uma taxa constante por um controle eletrônico de velocidade, apesar da tensão da linha flutuante e mudança da força de cisalhamento (GEOTEST, 2002).

2.4 Sistemas Embarcados

Na Seção 1.2 foi descrito de maneira breve o que é um sistema embarcado. Esta Seção 2.4 busca descrever esse tipo de sistema com mais detalhes.

A definição de um sistema embarcado pode variar de autor para autor. Por exemplo, alguns autores consideram que qualquer sistema com um sistema operacional embutido não é um sistema embarcado (WHITE, 2012). Assim, de acordo com White (2012), pode-se definir um sistema embarcado da seguinte forma: um sistema embarcado é um sistema computacional que foi construído para desenvolver um propósito específico, como um câmera digital.

Uma outra maneira de definir um sistema embarcado é:

> Um sistema embarcado pode ser definido como um sistema de hardware de computador que possui software embarcado específico para realizar algumas tarefas. Um sistema embarcado pode ser um sistema independente ou fazer parte de um sistema maior (HU, 2022, tradução nossa).

Apesar dessas duas definições utilizarem termos diferentes, possuem em seu cerne o mesmo significado. Uma definição apenas completa a outra, não há contradição. Para facilitar o entendimento, uma lista de exemplos de sistemas embarcados, seria a seguinte: geladeira, micro-ondas, relógio digital, equipamento médico (como monitor de batimentos cardíacos), câmeras, rádios digitais e etc.

No geral, pode-se dividir um sistema em: embarcado - já explicado acima - ou de propósito geral.

Uma definição de sistema de propósito geral de acordo com White (2012) e Hu (2022) é: um sistema que foi construído sem um propósito específico e, assim, pode realizar diversas funções diferentes dentro de seus limites, como um computador que pode ser usado de diversas maneiras, desde programar até assistir vídeos do *Youtube*.

Além do computador, alguns outros sistemas de propósito geral seriam o celular e o *tablet*, que com a evolução da tecnologia se tornaram praticamente computadores de bolso.

2.4.1 Classificação

Existem variados tipos de sistemas embarcados, como câmeras e monitores cardíacos, mencionados na Seção 2.4. Estes equipamentos tem tamanhos, complexidades e necessidades diferentes, assim será apresentada uma categorização de sistemas embarcados.

2.4.1.1 Escala

A escala de um sistema embarcado se refere a sua complexidade e muitas vezes ao seu tamanho e uso de memória (HU, 2022). De acordo com Hu (2022) pode-se dividir a escala em 3 tamanhos:

- Pequena: sistemas desenvolvidos com microcontroladores de 8 ou 16 bits. Usa menos memória e não necessita de um sistema operacional.
- Média: sistemas desenvolvidos com um único microcontrolador ou microprocessador de 16 ou 32 bits. Crescem as complexidades de software e hardware, com a utilização de *debuggers*, elementos de tempo real e etc .
- Sofisticada: Sistemas com muitas complexidades de software e hardware, onde é comum a utilização de microprocessadores, DSPs, FPGAs, IPs, ASICs e processadores configuráveis ou escaláveis. Utilizado em sistemas de tecnologia de ponta.

2.4.1.2 Tempo Real

Um sistema de tempo real deve reagir a um evento em um período de tempo determinado (WHITE, 2012). De acordo com Hu (2022) e Almeida (2016) pode-se classificar esse sistema em 2 categorias:

- Crítico ou *Hard*: É um sistema que deve reagir a um evento com a menor latência possível, pois atrasos podem ser catastróficos. Exemplo: um marca-passo não pode

errar os tempos da batida do coração, se o batimento é de 1 em 1 segundo e a reação se atrasar em 0.1 segundos, ocorrendo no segundo 1.1 , isso pode criar batimentos irregulares e, no pior caso, prejudicar o coração de quem usa o equipamento.

- Não crítico ou *Soft*: É um sistema que pode reagir a um evento com um certo nível de latência, assim uma reação que deveria ocorrer de 1 em 1 segundo pode ocorrer em 0.9 ou 1.1 segundos, porém atrasos degradam a performance do aparelho. Um exemplo seria a televisão: no qual um atraso no envio de imagens não é algo crítico, porém pode ser desagradável aos olhos do usuário.

2.4.1.3 Tolerância a falta

Tolerância a falta é a maneira como o sistema reage ao se deparar com um erro. Um sistema tolerante a faltas é aquele que deve reagir de maneira graciosa ao se deparar com erros, tentando se recuperar deles e evitando assim a geração de mais problemas, como a parada total do equipamento (WHITE, 2012). Existem situações em que o conserto do dispositivo se torna inviável, como um rastreador de baleias, no qual existe um grande nível de dificuldade para recuperar o sistema. De acordo com White (2012) o sistema pode demonstrar o erro para o usuário de maneiras diferentes:

- Silenciosa: o dispositivo não avisa o usuário que houve algum erro, apenas o trata e segue seu funcionamento normal, fazendo com que na maioria das vezes o usuário nem perceba que um erro ocorreu.
- Não silenciosa: o dispositivo deve avisar de maneira clara que algum problema está ocorrendo, por exemplo: o sistema de navegação de uma aeronave não deve manifestar falta silenciosamente.

2.4.1.4 Integração

Integração é a conectividade ou não de um dispositivo com outros aparelhos através de alguma rede (HU, 2022). De acordo com Hu (2022) esses dispositivos podem ser classificados em:

- Dispositivo em rede: é um dispositivo conectado a outros através de uma rede e assim realiza suas funções como uma parte de um todo. Exemplos:
  - Uma rede de sensores em uma floresta que manda dados para um computador.
  - Uma câmera de vigilância que manda imagens para um monitor.
- Dispositivo independente: é um dispositivo que funciona sozinho, não há a necessidade de outros equipamentos para realizar suas funções. Exemplos: micro-ondas e máquina de lavar roupa.

O *Digital Weight Scale* (DWS) é uma comunicação *offline* (não utiliza internet) de máquina para máquina sem a interferência humana. Um exemplo: uma rede de sensores em uma floresta que manda dados para um computador. Com a evolução da internet o termo DWS evoluiu para *Internet of Things* (IoT), no qual a comunicação é feita principalmente pela internet (HU, 2022).

2.4.1.5 Mobilidade

A mobilidade é caracterizada pela maneira como o produto foi desenvolvido para se deslocar no ambiente (HU, 2022). De acordo com Hu (2022) esses produtos podem ser divididos em:

- Móvel: Um dos requisitos do produto é que ele possa ser movido facilmente pelo ambiente, como um celular ou uma câmera digital.
- Fixo: O produto tem uma posição fixa e não se move, como um micro-ondas.

2.4.2 Limitações e Dificuldades

Esta Seção tem como objetivo esclarecer algumas das limitações e dificuldades que surgem no desenvolvimento de sistemas embarcados.

2.4.2.1 Recursos

Sistemas embarcados lidam com diversas limitações, como: consumo de energia, memória (RAM), espaço de código (ROM ou flash), ciclos de processador ou velocidade e periféricos do processador. É possível realizar um certo nível de troca entre esses elementos, como diminuir o gasto de energia ao reduzir a velocidade do processador ou aumentar o gasto de memória ou espaço de código para reduzir ciclos do processador (WHITE, 2012). Exemplo: pré-computar valores de senos e cossenos e salvar os resultados em uma tabela. Isto é feito para que durante a execução do código não seja necessário gastar ciclos do processador para calcular estes valores. Assim, é necessário entender os requisitos do sistema e tentar balancear da melhor maneira possível os seus recursos para reduzir gastos e a possibilidade de problemas futuros.

2.4.2.2 Mudanças e *Bugs*

Ao longo da vida de um produto mudanças são comuns, e como dito em 2.4, um sistema embarcado é construído com propósitos determinados ou até únicos. Durante o desenvolvimento é necessário balancear as diversas restrições e descobrir quais delas podem se tornar um problema no futuro, dessa maneira desenvolver um sistema que seja adaptável a mudanças é um dos grandes desafios dessa área. Um exemplo: o espaço de

código (ROM ou flash) não tem memória suficiente para receber uma atualização do sistema (WHITE, 2012).

Uma das dificuldades de desenvolvimento nesses sistemas é descobrir onde se encontram os *bugs*. Dependendo do aparelho no qual se está trabalhando o seu código pode causar danos. Um sistema mal projetado pode permitir que o processador aqueça acima de temperaturas recomendadas e por consequência, queime o dispositivo (WHITE, 2012).

2.4.2.3 Planejamento

Com a evolução rápida das tecnologias, a necessidade de desenvolver rapidamente os produtos vem aumentando. Como consequência disso muitos desenvolvedores descartam etapas importantes do planejamento para um lançamento mais veloz do produto. Contudo, na maioria das vezes, ao tomar essa atitude, os desenvolvedores são obrigados a voltar para trás e corrigir erros, tornando o tempo de desenvolvimento mais longo do que deveria. Um bom planejamento pode reduzir o tempo total de desenvolvimento e garantir uma melhor qualidade de produto (HU, 2022).

2.5 *Retrofit*

Retrofit é o ato de atualizar uma máquina antiga com novos componentes. É realizada uma investigação dos componentes atuais da máquina para entender o que deve ser substituído. Na maioria das vezes substitui-se os componentes eletrônicos e mecanismos de controle. O software também pode ser atualizado, segundo Lima II et al. (2005). Isto acontecerá no trabalho atual. O objetivo do *retrofit* é melhorar a acurácia, velocidade, manutenibilidade e usabilidade de um equipamento. Um outro motivo é seu custo menor em relação a obtenção de um novo equipamento (AL-MAEENI et al., 2020).

A quarta revolução industrial ou indústria 4.0 é caracterizada pelo uso difundido de sensores inteligentes, sistemas embarcados, sistemas ciberfísicos (CPS) e Internet das Coisas (IoT). *Smart retrofitting* está diretamente ligado a indústria 4.0, pode-se definir *smart retrofitting* como a atualização de uma máquina adicionando elementos da indústria 4.0, como sistemas embarcados e Internet das Coisas (IoT) (AL-MAEENI et al., 2020).

O presente trabalho terá foco em *smart retrofitting*, pois será estudado a implementação de um sistema embarcado com a *Raspberry Pi* e a utilização de IoT através do acesso remoto.

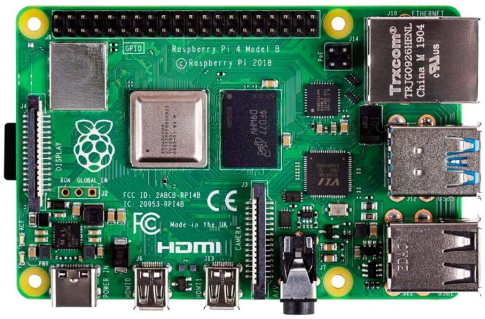
2.6 Raspberry Pi 4

Para o desenvolvimento deste projeto será utilizado uma *Raspberry Pi 4* Modelo B. Esta Seção tem como objetivo explicar de maneira sucinta o que é esse equipamento.

2.6.1 Definição e Usos

Uma *Raspberry Pi* é um computador de placa única (SBC), construído em uma placa de circuito impresso (PCB). Ele consegue desempenhar as mesmas funções de um computador, como acessar a internet e assistir vídeos, porém pode demorar mais tempo para realizá-las. Uma de suas grandes vantagens é o seu tamanho pequeno, aproximadamente do tamanho de um cartão de crédito (HALFACREE, 2019). A Figura 8 mostra uma imagem de uma *Raspberry Pi 4* Modelo B.

Figura 8 – *Raspberry Pi 4* Modelo B



Fonte: raspberryPi4

O uso deste computador tem crescido nos últimos anos na área de embarcados, sendo usado em lavadoras de louça, exploração de regiões em baixo de água, aspiradores de pó inteligentes e etc (RASPBERRY PI FOUNDATION, 2022b).

2.6.2 Especificações

O Quadro 1 mostra as especificações de uma *Raspberry Pi 4* Modelo B de acordo com (RASPBERRY PI FOUNDATION, 2022a).

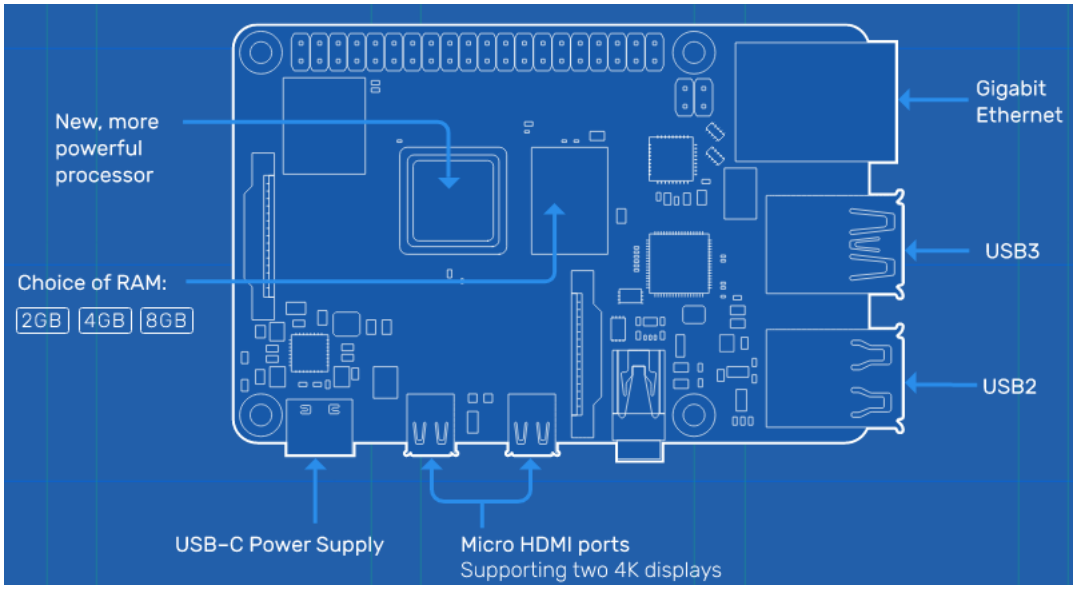
Quadro 1 – Especificações *Raspberry Pi 4* Modelo B

Número	**Especificação**
1	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
2	1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (dependendo do modelo)
3	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
4	Gigabit Ethernet
5	2 portas USB 3.0; 2 portas USB 2.0
6	Raspberry Pi standard 40 pin GPIO header (totalmente compatível com as placas anteriores)
7	2 portas micro-HDMI (suporta até 4kp60)

8	Porta de exibição MIPI DSI de 2 vias
9	Porta de câmera MIPI CSI de 2 vias
10	Áudio estéreo de 4 polos e porta de vídeo composto
11	H.265 (decodificação 4kp60), H264 (decodificação 1080p60, codificação 1080p30)
12	OpenGL ES 3.1, Vulkan 1.0
13	Slot para cartão micro-SD para carregar o sistema operacional e armazenar dados
14	5V DC via conector USB-C (mínimo 3A*)
15	5V DC via conector GPIO (mínimo 3A*)
16	Power over Ethernet (PoE) habilitado (requer PoE HAT separado)
17	Temperatura de operação: 0 – 50 graus C ambiente
Observação	* Uma fonte de alimentação de 2,5 A de boa qualidade pode ser usada se os periféricos USB *downstream* consumirem menos de 500 mA no total.

A Figura 9 mostra alguma das características apresentadas no Quadro 1.

Figura 9 – Especificação *Raspberry Pi 4* Modelo B



Fonte: (RASPBERRY PI FOUNDATION, 2022a)

2.7 *Design* de interface

Foram encontradas 2 definições para design que se complementam:

> O conceito de design compreende a concretização de uma ideia em forma de projetos ou modelos, mediante a construção e configuração resultando em um produto industrial passível de produção em série (LöBACH, 2001).

> Design é o processo de visualizar e planejar a criação de objetos, sistemas, veículos, serviços, espaços, edifícios, etc. Seu objetivo é abordar um problema ou necessidade e fornecer uma solução para ele (FARRINGTON, 2020).

Em seu cerne as 2 definições tem o mesmo significado, “planejamento da construção de uma ideia ou objeto”.

Para obter um bom design o desenvolvedor deve se preocupar com questões estéticas e em sua usabilidade. Um software pode ser extremamente bonito, porém complexo demais para ser utilizado, tornando ruim a experiência para o usuário (NORMAN, 2013).

Era comum, em tempos passados, o desenvolvedor culpar o usuário pelo mau uso do produto, alegando que se o usuário tivesse lido o manual ele não cometeria esses erros, porém até os manuais podiam ser confusos. Esse pensamento está desatualizado, o desenvolvedor deve compreender como o usuário utilizará o produto, tornando o seu uso simples e fácil e informando seus erros de maneira clara. O desenvolvedor pode cometer erros, porém é necessário aprender com eles (NORMAN, 2013).

> O problema com os projetos da maioria dos engenheiros é que eles são muito lógicos. Temos que aceitar o comportamento humano do jeito que é, não do jeito que gostaríamos que fosse (NORMAN, 2013, tradução nossa).

Norman (2013) menciona em seu livro que até portas podem ter um design ruim, as chamadas "portas de Norman". As portas podem ser puxadas, empurradas, levantadas e abaixadas. Com um design de má qualidade os usuários não saberão para qual direção a porta abre.

2.7.1 *User Experience* (UX) e *User Interface* (UI)

User Experience (UX) e *User Interface* (UI) se entrelaçam e são essenciais um para o outro, uma bela interface não consegue esconder uma navegação ruim (FARRINGTON, 2020). De acordo com Farrington (2020) pode-se definir UI e UX como:

- UI: este termo está associado a estética do produto, como o esquema de cores, tamanho de botões, fontes, design das telas e etc.
- UX: este termo está associado à experiência do usuário ao utilizar o produto. Trata da interação do usuário com os elementos do produto e a navegação que o mesmo realiza para concluir alguma atividade. O nível de dificuldade para realizar tarefas e interagir com elementos também é um fator importante.

Uma bela UI é fácil de ser notada, pois se trata de algo estético, em contrapartida uma boa UX não é facilmente detectada. Geralmente pode-se dizer que uma UX é boa quando o usuário não percebe a sua presença, ou seja, ele consegue realizar suas atividades sem empecilhos (FARRINGTON, 2020).

2.7.2 Sistemas Embarcados e IoT

O design para um sistema embarcado é diferente do de um computador. Como mencionado na Seção 2.4.2 existem limitações de hardware, então é necessário um desenvolvimento mais cuidadoso das interfaces para o usuário ter uma boa UX e UI (CHARLIER, 2015).

Ao utilizar sistemas com IoT as complicações aumentam, pois a transmissão da informação ocorre através de alguma rede, e esta pode ter atrasos, *delays*. Isso pode fazer o usuário perder o *feedback* instantâneo que existe em interações com o mundo real. Um exemplo: ligar um micro-ondas pelo aplicativo ou por um botão na máquina. Ao utilizar o aplicativo pode ocorrer um problema na rede, fazendo com que o comando de ligar demore para chegar no micro-ondas. O botão de ligar na máquina não tem esse problema pois, ao clicar nele, o *feedback* é instantâneo (CHARLIER, 2015).

Neste contexto a inter usabilidade, ou a usabilidade do sistema completo, tem um papel importante. Em sistemas IoT deve-se pensar em como melhor distribuir as funcionalidades entre os dispositivos. Tratar cada equipamento como um sistema isolado e sua própria UI irá tornar a experiência geral do sistema pior para o usuário. Assim deve-se garantir que a experiência do usuário ao utilizar o sistema completo seja tão boa ou melhor do que em um aparelho individual (CHARLIER, 2015).

2.8 Requisitos

A elicitação de requisitos é uma importante área para a engenharia de software e a falta de cuidado ao trabalhar com ela pode trazer muitos problemas no futuro. Um dos descuidos comuns é a maneira como as pessoas descobrem, documentam e modificam os requisitos (WIEGERS; BEATTY, 2013). Uma definição de requisito é:

> Os requisitos são uma especificação do que deve ser implementado. São descrições de como o sistema deve se comportar, ou de uma propriedade ou atributo do sistema. Eles podem ser uma restrição no processo de desenvolvimento do sistema (SOMMERVILLE; SAWYER, 1997, tradução nossa).

Alguns problemas relatados por Wiegers e Beatty (2013) em projetos que não tiveram sucesso no levantamento de requisitos :

- O objetivo, visão e escopo do projeto não foram claramente definidos.
- Clientes não se reuniram com os desenvolvedores para trabalhar nos requisitos, pois estavam ocupados.
- Clientes disseram que todas as funcionalidades eram críticas e assim não houve priorização.
- Os desenvolvedores encontraram trechos no qual havia ambiguidade durante o código e assim chutaram qual seria o próximo passo.
- Os clientes nunca aprovaram os requisitos.

2.8.1 Priorização

Existe um certo conflito entre clientes, desenvolvedores e priorização de requisitos. Uma dúvida dos clientes é: “porque fazer a priorização se todos os requisitos são importantes?”. Eles também têm o receio de que requisitos com baixa prioridade nunca sejam feitos. Em contrapartida os desenvolvedores acreditam que conseguem fazer tudo

e que a priorização não será útil, porém a realidade é que eles não conseguem fazer tudo (WIEGERS; BEATTY, 2013).

A priorização é uma parte essencial da elicitação de requisitos. Sabe-se que todos os requisitos têm importância, porém alguns mais que outros. Assim, a priorização permite que sejam feitos mais rapidamente os requisitos que possuem mais urgência (WIEGERS; BEATTY, 2013). Além disso, a criação de um cronograma se torna mais fácil com a informação do que precisa ser feito primeiro, contribuindo com o planejamento das *sprints* utilizadas no *Scrum Solo* (mencionado no capítulo 3). Uma definição para priorização é:

> A priorização significa alocar cuidadosamente recursos limitados para obter o máximo benefício do investimento que uma organização faz em um projeto (WIEGERS; BEATTY, 2013, tradução nossa).

Serão utilizados dois métodos, utilizados juntos, descritos por Wiegers e Beatty (2013) para construir a priorização de requisitos:

- Escala de três níveis: alto, médio, baixo.
- Priorização baseada em valor, custo e risco.

Estes dois métodos são utilizados em conjunto, pois a recomendação de Wiegers e Beatty (2013) é que a priorização baseada em valor, custo e risco não seja utilizada em requisitos com alta prioridade. Primeiro encontram-se os requisitos com mais alta prioridade no método da escala de 3 níveis, a fim de depois utilizar a priorização baseada em valor, custo e risco para os requisitos com média e baixa prioridade.

Na escala de três níveis os *stackholders* definem se um requisito é de alta, média ou baixa prioridade. Contudo é possível que ainda fiquem muitos requisitos na alta prioridade. Sendo assim, deve-se dividir os requisitos que estão na alta prioridade em extremamente alta, muito alta e alta. Apenas os requisitos classificados com prioridade extremamente alta continuarão na classificação alta e o resto irá para a média (WIEGERS; BEATTY, 2013). A Figura 10 mostra este processo.

Com a definição dos requisitos de mais alta prioridade, pode-se seguir para a segunda etapa e utilizar a priorização baseada em valor, custo e risco para determinar a ordem entre os requisitos de baixa e média prioridade. O Quadro 2, baseado nas explicações de Wiegers e Beatty (2013), mostra o significado das variáveis utilizadas nesse método.

Quadro 2 – Explicação das variáveis da priorização baseada em valor, custo e risco

Número	**Variável**	**Nome Relativo**	**Explicação**

1	Benefício Relativo	BR	É o benefício relativo que a implementação do requisito irá trazer para o cliente ou negócio. Escala de 1 a 9, na qual 1 representa que ninguém achou útil o requisito e 9 representa que o requisito é extremamente importante.
2	Penalidade Relativa	PR	É a penalidade relativa que o cliente ou empresa terá com a não implementação do requisito. Escala de 1 a 9, na qual 1 representa que a não implementação do requisito teria quase nenhum impacto e 9 representa que a não implementação pode trazer grandes problemas.
3	Valor Total	VT	$VT = (BR \times Peso1) + (PR \times Peso2)$ (2.1)
4	Valor em %	VP	$VP = \frac{(100 \times VT)}{Soma\ de\ todos\ os\ VT}$ (2.2)
5	Custo Relativo	CR	É o custo que os desenvolvedores terão para desenvolver o requisito. Escala de 1 a 9, no qual 1 representa o desenvolvimento rápido e fácil do requisito e 9 como extremamente longo , custoso e difícil.
6	Custo em %	CP	$CP = \frac{(100 \times CR)}{Soma\ de\ todos\ os\ CR}$ (2.3)
7	Risco Relativo	RR	É a probabilidade de não conseguir desenvolver uma funcionalidade. Escala de 1 a 9, no qual 1 representa que é extremamente fácil programar essa funcionalidade e 9 representa que provavelmente não é viável, podendo ser os motivos: complexidade escondida no requisito, falta de conhecimento da equipe, falta de equipamento e utilização de ferramentas desconhecidas

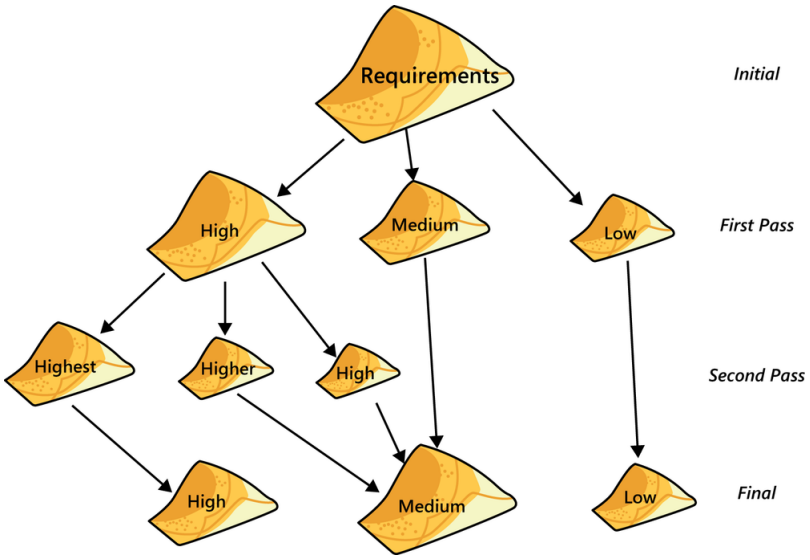
8	Risco em %	RP	$$RP = \frac{(100 \times RR)}{Soma\ de\ todos\ os\ RR} \quad (2.4)$$
9	Prioridade	P	É a ordem no qual os requisitos serão desenvolvidos. $$P = \frac{VP}{(CP \times Peso3) + (RP \times Peso4)} \quad (2.5)$$

A Tabela 1 mostra um exemplo dessa prioridade, onde os pesos relativos foram:

- Peso1 (Benefício Relativo): 2,0
- Peso2 (Penalidade Relativa): 1,0
- Peso3 (Custo Relativo): 1,0
- Peso4 (Risco Relativo): 0,5

Lembrando que cada projeto pode definir os seus próprios pesos.

Figura 10 – Priorização em uma escala de 3 níveis



Fonte: (WIEGERS; BEATTY, 2013)

Tabela 1 – Exemplo de priorização baseada em valor, custo e risco

Requisito	**Benefício Relativo**	**Penalidade Relativa**	**Valor Total**	**Valor em %**	**Custo Relativo**	**Custo em %**	**Risco Relativo**	**Risco em %**	**Prioridade**
Imprimir uma folha de dados de segurança do material	2	4	8	5,16	1	2,70	1	3,03	1,22
Status da consulta de um pedido de fornecedor	5	3	13	8,39	2	5,41	1	3,03	1,21
Gerar um relatório de inventário de estoque de produtos químicos	9	7	25	16,13	5	13,51	3	9,09	0,89
Veja o histórico de um recipiente químico específico	5	5	15	9,68	3	8,11	2	6,06	0,87
Pesquisar catálogos de fornecedores para um produto químico específico	9	8	26	16,77	3	8,11	8	24,24	0,83
Manter uma lista de produtos químicos perigosos	3	9	15	9,68	3	8,11	4	12,12	0,68
Modificar uma solicitação química pendente	4	3	11	7,10	3	8,11	2	6,06	0,64
Gerar um relatório de inventário de laboratório individual	6	2	14	9,03	4	10,81	3	9,09	0,59
Verifique o banco de dados de treinamento para registro de treinamento de produtos químicos perigosos	3	4	10	6,45	4	10,81	2	6,06	0,47
Importar estruturas químicas de ferramentas de desenho de estrutura	7	4	18	11,61	9	24,32	7	21,21	0,33
Totais	53	49	155	100	37	100	33	100	

Fonte: (WIEGERS; BEATTY, 2013)

3 Metodologia

Este capítulo tem como objetivo detalhar melhor as metodologias aplicadas pelo autor para a realização deste trabalho. Como dito na Seção 1.5, serão aplicadas as metodologias do *Kanban* e *Scrum Solo* e para melhor compreensão sobre *Scrum Solo* é necessário discorrer brevemente sobre *Scrum* e *PSP*.

3.1 *Scrum*

O *Scrum* é uma metodologia de desenvolvimento ágil para equipes que visam a melhoria contínua. Por este aspecto incremental ele auxilia em projetos onde o planejamento extensivo no início do desenvolvimento se torna inviável (SMITH, 2018). De acordo com Pagotto et al. (2016), Schwaber e Sutherland (©2020) e Smith (2018) os principais aspectos do *Scrum* são:

- *Sprint*: ciclos curtos de desenvolvimento (1 a 4 semanas).
- Reuniões diárias: reuniões curtas que auxiliam no entendimento de como as tarefas estão sendo desenvolvidas.
- *Sprint Planning*: reunião para definir quais itens do *product backlog* entrarão no *sprint backlog*.
- Revisão da *sprint*: reunião para entender o que foi feito durante a *sprint*.
- Retrospectiva da *sprint*: reunião feita para entender como as atividades foram desenvolvidas durante a *sprint* e se houveram problemas.
- *Product Backlog*: lista de tarefas que devem ser desenvolvidas ao longo do projeto, é incremental
- *Sprint Backlog*: lista de tarefas que devem ser realizadas durante a *sprint*.
- *Product Owner*: pessoa responsável por garantir que a visão de produto do cliente e *stackholders* seja transmitida corretamente para os desenvolvedores.

3.2 *Personal Software Process (PSP)*

O *Personal Software Process (PSP)* é uma metodologia de desenvolvimento de trabalho individual, seu objetivo é garantir a qualidade dos projetos ajudando os desenvolvedores a organizarem sua forma de trabalho. Este método visa melhorias contínuas do

processo individual (PAGOTTO et al., 2016). De acordo com Humphrey (©2000) alguns princípios dessa metodologia são:

- Erros descobertos mais cedo são mais baratos.
- Para engenheiros melhorarem sua qualidade de trabalho são necessários processos bem definidos.
- Os engenheiros devem se sentir responsáveis pela qualidade do produto para garantir sua qualidade.
- É mais barato prevenir erros do que corrigi-los posteriormente.

O Quadro 3 mostra os principais objetivos de acordo com Humphrey (©2000) e Pomeroy-Huff et al. (©2009).

Quadro 3 – Principais objetivos do PSP

Numeração	**Objetivo**
1	Aperfeiçoar a estimativa de esforço e prazo para as etapas de desenvolvimento do software.
2	Criar cronogramas e planejamentos que auxiliem na organização e que possam ser concluídos, evitando o excesso de atividades.
3	Garantir a qualidade do software.
4	Diminuir a quantidade de erros.

3.3 *Kanban*

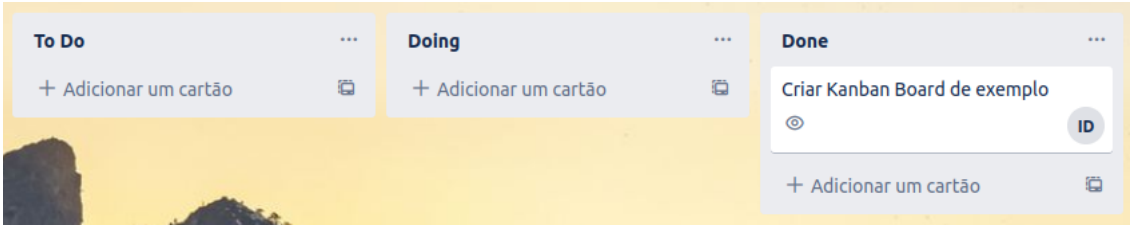
O *Kanban* é uma metodologia que tem como princípio facilitar a visualização e organização do trabalho. Um artefato essencial é o *Kanban Board*, pois é nele que se criam e se visualizam as tarefas (MELãO, 2021). De acordo com Melão (2021) as principais práticas são:

- Visualização do fluxo de trabalho, permitindo assim encontrar gargalos mais facilmente.
- Limitação da quantidade de trabalho, diminuindo a chance de sobrecarga.
- Adaptável a alterações, como a criação de uma nova coluna no *Kanban Board*.

- Fácil visualização das tarefas, melhorando a identificação de tarefas que estão paradas.

A Figura 11 demonstra um exemplo de divisão do fluxo de atividades.

Figura 11 – *Kanban Board*



Fonte: Elaborado pelo autor, 2022

3.4 *Scrum Solo*

O *Scrum Solo* é uma adaptação do *Scrum* com *PSP*. A lista abaixo descreve algumas características utilizadas de cada metodologia.

- *Scrum*: utilização de artefatos como *Sprints, Product Backlog e Sprint Backlog* que auxiliam na organização e planejamento do projeto.
- *PSP*: adição de novas etapas ao processo de desenvolvimento, como *Management*, que garantem a qualidade e velocidade na produção de software ao monitorar, planejar e controlar o desenvolvimento do produto.

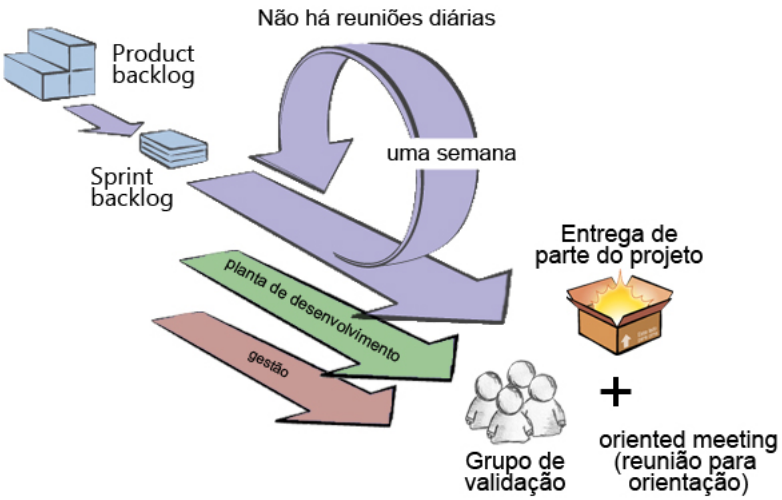
A Tabela 2 mostra de maneira mais clara as semelhanças e diferenças entre o *Scrum* e o *Scrum Solo* na utilização de artefatos.

Tabela 2 – Comparação entre *Scrum* e *Scrum Solo*

Tópico	**Scrum**	**Scrum Solo**
Trabalho em equipe	Sim	Não
Duração da *sprint*	1 a 4 semanas	1 semana
Reunião diária	Sim	Não
Sprint Planning	Sim	Sim
Revisão da *sprint*	Sim	Sim
Retrospectiva da *sprint*	Sim	Sim
Product Backlog	Sim	Sim
Sprint Backlog	Sim	Sim

Apesar do desenvolvimento ser um trabalho individual, existem ainda outras pessoas envolvidas neste TCC, como o *product owner* e o orientador do TCC, que validarão

Figura 12 – *Scrum Solo*



Fonte: (PAGOTTO et al., 2016)

as entregas feitas. Este ciclo de desenvolvimento pode ser observado na Figura 12 que demonstra o ciclo de uma *sprint*.

Essa metodologia será utilizada pelo autor para garantir uma melhor organização, qualidade e evolução dos processos durante o TCC.

4 Suporte Tecnológico

Este capítulo tem como objetivo apresentar as ferramentas que auxiliaram o desenvolvimento do trabalho.

4.1 Organização e gerenciamento do código

Essa Seção apresenta as ferramentas que foram utilizadas para auxiliar o autor em sua organização do trabalho.

4.1.1 Git

O git é um sistema de controle de versão (VCS) utilizado pela sua flexibilidade de desenvolvimento. Com ele é possível saber quem fez, quando fez e o que fez ao projeto, permite também que vários desenvolvedores alterem elementos do projeto ao mesmo tempo. Além disso é um projeto de código aberto (ATLASSIAN, 2022). Um importante artefato são as *branchs*:

- *Branchs*: É um ramo de desenvolvimento que é criado para evitar o desenvolvimento no ramo principal (produção). Isso permite que várias pessoas alterem o código ao mesmo tempo, sem que uma atrapalhe a outra (GITHUB, © 2022a).

4.1.2 Github

O Github é uma plataforma de hospedagens que permite os desenvolvedores salvarem seus projetos na internet. Ele implementa o gerenciamento de versões utilizando o Git (mencionado na Seção 4.1.1) (GITHUB, 2016). Alguns artefatos que facilitam o desenvolvimento de projetos nesta plataforma:

- *Issues*: Permite a descrição de tarefas ou problemas que devem ser resolvidos para o projeto (GITHUB, © 2022b).

- *Pull Requests*: Ao finalizar uma alteração em código, os desenvolvedores podem subir um *Pull Request* para que os outros desenvolvedores entendam as alterações que foram feitas e se tudo estiver correto permitir que a *branch* secundária se junte a uma *branch* principal (GITHUB, © 2022c).

4.1.3 Zenhub

O Zenhub é uma extensão web que auxilia na organização e produtividade de equipes ao implementar *Kanban Boards* (mencionado na Seção 3.3). De acordo com ZenHub (2022) ao utilizar essa ferramenta a produtividade de uma equipe pode aumentar em até 75%. Neste projeto o Zenhub está sendo utilizado em conjunto com o Github para organizar e facilitar a visualização das tarefas.

4.2 Desenvolvimento

Essa Seção tem como objetivo apresentar as ferramentas e tecnologias que serão utilizadas para o desenvolvimento do *software*.

4.2.1 Interface

A escolha de ferramentas e tecnologias foi realizada através de uma comparação em 2 fases, onde os critérios escolhidos foram definidos pelo autor. Os critérios da primeira fase foram:

- Se a ferramenta utiliza C, C++ ou Python: C, C++ foram escolhidos por uma questão de familiaridade do autor e seu desempenho, como mostrado na Fig. 13. O Python foi escolhido por possuir diversas bibliotecas que facilitam o desenvolvimento de *software* e sua baixa curva de aprendizado.
- Se a interface era moderna: utilizar uma ferramenta que produz uma *Graphical User Interface* (GUI) ou Interface Gráfica do Usuário ultrapassada torna pior a experiência do usuário. Mais detalhes sobre experiência do usuário podem ser encontrados na Seção 2.7.
- Compatível com *Raspberry Pi*: este é o critério principal, pois a incompatibilidade da ferramenta com a *Raspberry Pi* impossibilitaria o desenvolvimento do *software*.

A Tabela 3 mostra a comparação entre as diversas tecnologias escolhidas. Nela é possível observar que as ferramentas QT, Kyvi Python e Pyxis SDK passaram para a segunda fase por terem atingido os 3 pontos.

Na segunda fase os critérios foram:

- C, C++: a Figura 13 mostra que o desempenho com Python foi quase 94 vezes mais lento do que com C/C++. Assim, Python foi retirado dos critérios.
- Boa documentação disponível: uma ferramenta com boa documentação reduz significativamente a dificuldade do trabalho. Sua nota varia de 0 a 2. Com 0 sendo

Figura 13 – Comparação de performance de linguagens na Raspberry Pi

Table 5-1: Numeric Computation Time for 5,000,000 Iterations of the *n*-Body Algorithm on Raspbian (Jessie Minimal Image)

VALUE	TYPE	RPi 3 at 1.2 GHZ[1]	RPi 2 at 1 GHZ[2]	RPi B+ at 1 GHZ[3]	64-BIT i7 PC[4]
C/C++	Compiled	1.00 × (6.5s)	1.00 × (9.3s)	1.00 × (10.0s)	1.00 × (0.61s)
C++11	Compiled	1.06 × (6.9s)	0.69 × (6.4s)	0.70 × (7.03s)	0.95 × (0.58s)
Haskell	Compiled	1.16 × (7.6s)	1.17 × (10.8s)	1.07 × (10.8s)	1.15 × (0.70s)
Java[5]	JIT	1.52 × (9.94s)	1.45 × (13.4s)	2.29 × (23.0s)	1.36 × (0.83s)
Mono C#	JIT	2.72 × (17.8s)	2.47 × (22.9s)	3.62 × (36.4s)	2.16 × (1.32s)
Cython[6]	Compiled	2.74 × (17.9s)	2.67 × (24.8s)	2.80 × (28.0s)	1.26 × (0.77s)
Node.js[7]	JIT	2.76 × (18.1s)	6.23 × (57.7s)	50.1 × (503s)	6.54 × (3.99s)
Lua	Interpreted	20.2 × (132s)	21.2 × (197s)	25.7 × (258s)	34.3 × (20.9s)
Cython	Compiled	64.2 × (420s)	66.6 × (618s)	163 × (1633s)	58.0 × (34.4s)
Perl	Interpreted	92.6 × (601s)	81.5 × (756s)	171 × (1716s)	82.0 × (50.0s)
Python	Interpreted	94.1 × (616s)	89.9 × (834s)	198 × (1992s)	89.7 × (54.7s)
Ruby	Interpreted	147 × (962s)	140 × (1298s)	265 × (2662s)	47.4 × (28.9s)

[1] RPi 3 running at 1.2 GHz, quad core (only one core utilized), ARMv7 (rev 4 with a 32-bit Linux distribution: Linux 4.1.19-v7+) supports: half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32. Please ensure that you use a high-quality power supply that is capable of delivering at least 1.5 A.

[2] RPi 2 overclocked at 1 GHz, quad core (only one core utilized), ARMv7 (rev 5) supports: half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm. Note: Overclocking your RPi may reduce its lifespan.

[3] RPi B+ overclocked at 1 GHz, single core, ARMv6 (rev 7 v6) supports: half thumb fastmult vfp edsp java tls.

[4] Windows 8.1 PC running a 64-bit Debian Jessie VirtualBox VM that was allocated 3 threads (of 12) on an Intel i7-5820K @ 3.3 GHz, with the VM allocated 16 GB of RAM. Only one thread is used.

[5] You can use `sudo apt install oracle-java8-jdk` to install the Oracle JDK on the Raspberry Pi platform.

[6] This Cython test involved modifying the Python source code to optimize it. It is not simply the compilation of raw Python code. The second Cython test represents the simple compilation of raw Python source code.

[7] Node.js (`node -v`) is version v5.10.1 and it supports the ARM NEON accelerator processor. NEON is available on the RPi 2/3 (ARMv7) but not on the RPi B+ (ARMv6), which contributes to the poor performance of Node.js on the RPi B+ of 50.1× the baseline. See the feature titled "LAMP and MEAN" in Chapter 12 for instructions on how to install the latest version of Node.js on the RPi.

Fonte: (MOLLOY, 2016)

documentação de baixa qualidade, 1 sendo documentação de qualidade média e 2 sendo documentação de alta qualidade.

A Tabela 4 mostra o resultado da comparação na fase 2. O resultado foi a escolha da ferramenta QT por possuir uma boa documentação e permitir o uso da linguagem C++.

Tabela 3 – Comparação das ferramentas fase 1

Ferramentas/ Critérios	**C,C++ ou Python**	**Interface Moderna**	**Compatível com Raspberry Pi**	**Resultado 1ª fase**
Tkinter	1	0	1	2
Qt	1	1	1	3
Swing	0	0	1	1
wxWidgets	1	0	1	2
guizero library	1	0	1	2
Kyvi Python.	1	1	1	3
Electron.	0	1	1	2
GTK	1	0	1	2
TK	1	0	1	2
Pyxis SDK	1	1	1	3

Tabela 4 – Comparação das ferramentas na fase 2

Ferramentas/ Critérios	**C,C++**	**Boa documentação disponível**	**Resultado 2ª Fase**
Qt	1	2	3
Kyvi Python.	0	1	1
Pyxis SDK	1	0	1

4.2.1.1 QT

O QT é uma ferramenta que permite a produção de interfaces gráficas modernas e , na maioria das vezes, possui bom desempenho pela utilização de C++. Esta ferramenta é compatível com múltiplas plataformas, como Linux, Windows, Android e IOS. Assim, é possível realizar o seu desenvolvimento em diversos dispositivos, como computadores pessoais, sistemas embarcados e dispositivos móveis (celulares, por exemplo) (QT, 2022).

Complementando essa definição, pode-se dizer que:

> Qt não é uma linguagem de programação por si só. É um framework escrito em C++. Um pré-processador, o MOC (Meta-Object Compiler), é usado para estender a linguagem C++ com recursos como sinais e slots. Antes da etapa de compilação, o MOC analisa os arquivos de origem escritos em C++ estendido para Qt e gera fontes C++ compatíveis com o padrão a partir deles. Assim, a própria estrutura e os aplicativos/bibliotecas que a utilizam podem ser compilados por qualquer compilador C++ compatível com os padrões, como Clang, GCC, ICC, MinGW e MSVC (QT, 2022, tradução nossa).

4.2.1.2 *Benchmark*

Para realizar uma comparação de desempenho mais próxima do contexto do trabalho, foi realizado um *benchmark* entre QT em C++ e QT em Python. A função executada no *benchmark* consistia das seguintes etapas:

1. Antes do *benchmark* iniciar, cria-se o componente *QLineEdit* (simples campo de texto editável);

2. Simula digitação no teclado com a palavra “abc”;

3. Verifica se a palavra digitada corresponde a palavra esperada (abc);

4. Limpa o campo de texto (*QLineEdit*) e

5. Volta a etapa 2 (realiza apenas n vezes).

Essa comparação pode ser observada na Tabela 5.

Tabela 5 – Comparação entre QT em C++ e QT em Python

Número	**QT C++ (ms)**	**QT Python (ms)**
1	0.2	0.3036011
2	0.2	0.3168317
3	0.2	0.3045274
4	0.2	0.3406811
5	0.2	0.3193592

Assim, pode-se concluir que não houve uma diferença de desempenho significativa para este teste e que o desenvolvimento em Python não afetaria o tempo de renderização dos componentes e, também não afetaria o tempo de resposta da interação do usuário com os componentes na tela. Abaixo se encontram pedaços dos códigos em C++ e Python.

Pedaço do código em C++

```
void c_Case::testLine()
{
        QLineEdit lineEdit;
        QBENCHMARK {
                QTest::keyClicks(&lineEdit,"abc");
                QCOMPARE(lineEdit.text(), QString("abc"));
                lineEdit.clear();
        }
}
```

```
Pedaço do código em Python
def line_edit_test(app, qtbot):
        qtbot.keyClicks(app.line_edit, 'abc')
        result = app.line_edit.text() == "abc"
        app.line_edit.clear()
        return  result


def test_benchmark_line_edit(benchmark,app,qtbot):
        result = benchmark.pedantic(line_edit_test,args=(app,qtbot),
                iterations=2000,rounds=51)
        assert  result == True
```

4.2.2 Banco de Dados

De acordo com Shields (2019), um banco de dados pode ser definido como uma coleção de dados organizada de tal maneira que facilite e aumente sua velocidade de consulta em um computador.

Um banco de dados relacional é um banco de dados onde se procura facilitar o relacionamento entre diversas tabelas. Esses relacionamentos acontecem por meio de chaves primárias (identificador único de uma tabela) e chaves estrangeiras (identificador da tabela A dentro de uma tabela B, indicando o relacionamento) (SHIELDS, 2019).

A Figura 14 mostra uma rascunho dos relacionamentos encontrados no sistema. Pode-se descrever esses relacionamentos da seguinte forma:

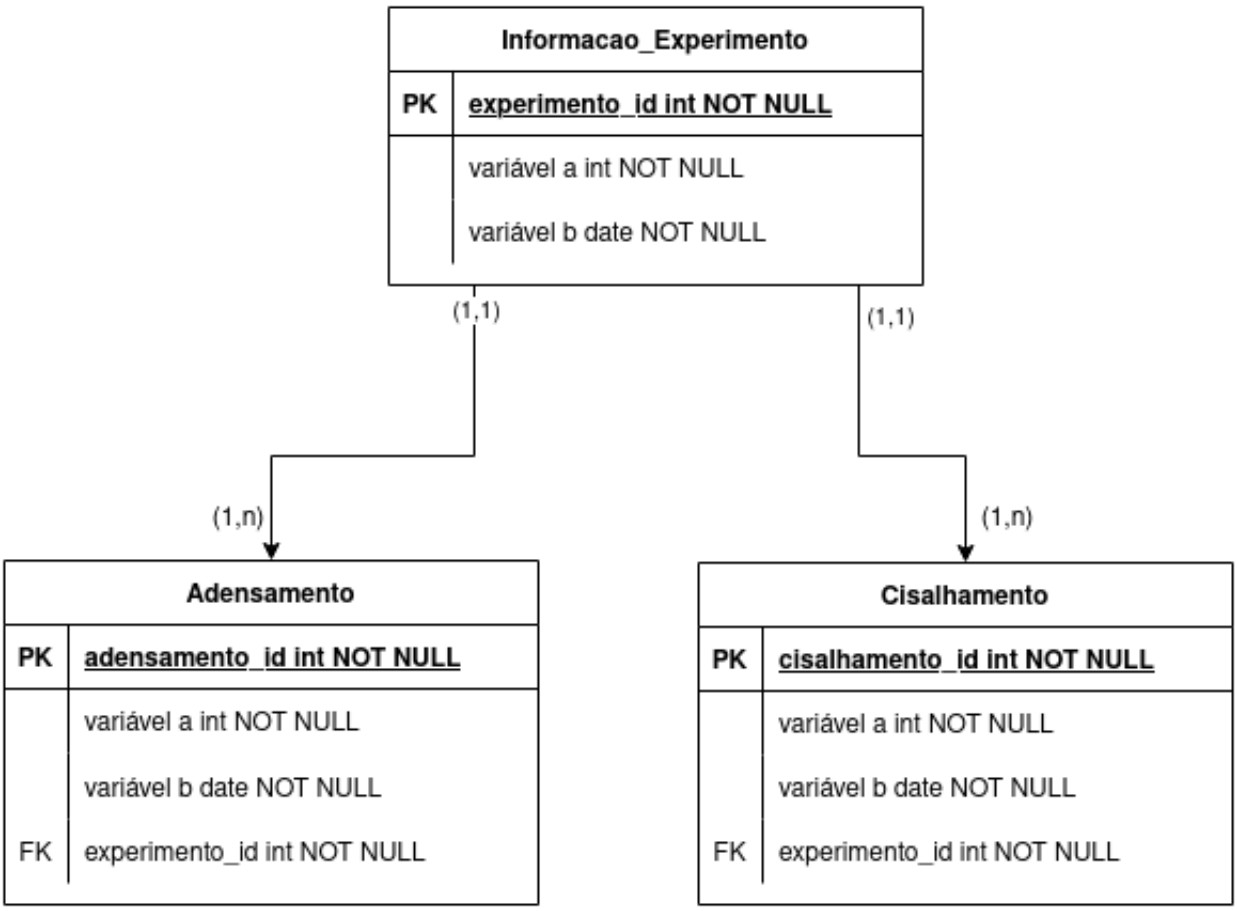
- Uma Informacao_Experimento possui entre 1 a n Adensamentos e 1 Adensamento só pode pertencer a uma Informacao_Experimento.
- Uma Informacao_Experimento possui entre 1 a n Cisalhamentos e 1 Cisalhamento só pode pertencer a uma Informacao_Experimento.

4.2.2.1 Comparação entre sistemas gerenciadores de banco de dados relacional (RDBMS)

Para descobrir o *Relational Database Management System* (RDBMS), ou sistema gerenciador de banco de dados mais adequado será utilizado 1 critério:

- Memória: a *Raspberry Pi* como qualquer outro sistema embarcado tem limitações de memória (mencionado na Seção 2.4.2.1). A *Raspberry Pi* possuída pela autor possui apenas 4GB de RAM.

Figura 14 – Rascunho do relacionamento entre as entidades do sistema



Fonte: Elaborado pelo autor, 2022

Foi realizada uma comparação entre MySQL e SQlite para verificar qual gerenciador de banco de dados melhor se aplica ao projeto. De acordo com Pomponio (2021) o uso de memória do SQLite é de aproximamente 250 kilobytes (kB), enquanto que o do MySQL é de aproximadamente 600 megabytes (MB). A Equação 4.1 demonstra a diferença de tamanho entre eles e a Equação 4.2 mostra o espaço em memória ocupado em uma *Raspberry Pi* com 4GB de RAM.

$$diferenca\_tamanho = \frac{memoriaMySql}{memoriaSQLite} \tag{4.1}$$

$$diferenca\_tamanho = \frac{600 * 1024kB}{250kB} = 2457,6\ vezes$$

$$ocupacao\_memoria = \frac{memoriaFerramenta \times 100}{memoriaRaspberry\ Pi} \tag{4.2}$$

Para MySQL:

$$ocupacao\_memoria = \frac{620MB \times 100}{4 \times 1024MB} \approx 15,1\%\ de\ ocupacao\ da\ memoria$$

Para SQLite:

$$ocupacao\_memoria = \frac{250kB \times 100}{4 \times 1024 \times 1024kB} \approx 0,0059\% \; de \; ocupacao \; da \; memoria$$

Pelo uso muito menor de RAM, foi decidido que seria utilizado o SQLite.

4.2.2.2 SQLite

Pode-se definir SQLite como:

> SQLite é uma biblioteca de linguagem C que implementa um mecanismo de banco de dados SQL pequeno, rápido, autônomo, de alta confiabilidade e cheio de recursos (SQLITE, 2022, tradução nossa).

Diferentemente de outros gerenciadores de banco de dados, o SQLite escreve e lê diretamente em arquivos e não possui um servidor ativo. Esses fatores reduzem sua complexidade e ocupação de memória (SQLITE, 2022).

4.2.3 Documentação

A documentação é uma das partes importantes do desenvolvimento de um código. Neste projeto a documentação do código foi gerada utilizando a ferramenta Doxygen. De acordo com Doxygen (© 2022), pode-se definir esta ferramenta como um gerador automático de documentação para código, com especialidade em C++, mas que também suporta C, C#, PHP, Java, Python entre outras linguagens.

De acordo com Doxygen (© 2022), esta ferramenta pode auxiliar em 3 atividades:

- Geração de documentação em HTML e/ou manual de referência em Latex.
- Configuração para extração de estruturas do código de arquivos de origem não documentados.
- Criação de uma documentação normal.

5 Proposta de solução

Este capítulo tem como objetivo apresentar a proposta de solução que resolverá o problema levantado na Seção 1.3. Será mostrado o passo a passo de como foi realizado esse processo e como se chegou à conclusão final. Este capítulo será dividido nas seguintes seções:

- Requisitos: levantamento e priorização de requisitos para garantir que o software atenda às exigências do usuário;
- Design: realização do protótipo e definição do esquema de cores;
- Arquitetura: investigação de quais arquiteturas possam ser utilizadas;
- Banco de dados: apresenta a organização do banco de dados.
- Comandos: esclarece os comandos utilizados na comunicação com a camada de controle.
- Classificação do sistema: classifica o sistema com base na Seção 2.4;

5.1 Requisitos

Esta seção tem como objetivo levantar e priorizar os requisitos do projeto com base na Seção 2.8.

5.1.1 Requisitos funcionais e não funcionais

Com o intuito de reduzir a probabilidade de erros ocorrerem, foram criadas as Tabelas 6, 7, 8, 9 e 10.

Os requisitos funcionais foram divididos em 3 categorias:

- Requisitos funcionais do *frontend* (alta prioridade: Tabela 6 e média e baixa prioridade: Tabela 7).
- Requisitos funcionais do *backend* (Tabela 8).
- Requisitos funcionais do banco de dados (Tabela 9).

Os requisitos funcionais do *frontend* foram divididos nas categorias:

- Histórico: Requisitos relacionados a disponibilização de um histórico de experimentos.
- Configuração do sistema: Requisitos relacionados a configurações do sistema, incluindo configurações da máquina, portas e do próprio sistema.
- Criação de um novo experimento: Requisitos relacionados a criação de um novo experimento.
- Controle e visualização do experimento: Requisitos relacionados a visualização de informações de um experimento e o controle do mesmo.
- Armazenamento: Requisitos relacionados ao armazenamento do dispositivo.

Tabela 6 – Requisitos funcionais do *frontend software* de alta prioridade

Número	**Categoria**	**Descrição**	**Priorização**	**Observações**
1	Configuração do sistema	O usuário deverá ser capaz de testar a conexão das portas.	alta	
2	Criação de um novo experimento	O usuário deverá ser capaz de iniciar o experimento.	alta	
3	Controle e visualização do experimento	O usuário deverá ser capaz de interromper o experimento.	alta	
4	Controle e visualização do experimento	O usuário deverá ser capaz de passar da fase de adensamento para a fase de cisalhamento e da fase de cisalhamento para o encerramento do experimento.	alta	
5	Controle e visualização do experimento	O usuário deverá ser capaz de configurar variáveis quando o processo de adensamento for finalizado, para poder iniciar o de cisalhamento.	alta	Variáveis como a distância e a velocidade.

Tabela 7 – Requisitos funcionais do *frontend software* de média e baixa prioridade

Número	**Categoria**	**Descrição**	**Priorização**	**Observações**

1	Configuração	O usuário deverá ser capaz de visualizar a configuração das portas.	1,287	
2	Controle e visualização do experimento	O usuário deverá ser capaz de exportar os dados das tabelas	1,179	É necessário conexão com a internet.
3	Histórico	O usuário deverá ser capaz de pesquisar por um experimento específico.	0,641	
4	Configuração do sistema	O usuário deverá ser capaz de visualizar informações do sistema.	0,641	
5	Configuração do sistema	O usuário deverá ser capaz de visualizar informações da máquina.	0,641	
6	Configuração do sistema	O usuário deverá ser capaz de editar as as informações da máquina.	0,641	
7	Configuração do sistema	O usuário deverá ser capaz de configurar a conexão das portas.	0,584	
8	Controle e visualização do experimento	O usuário deverá ser capaz de visualizar gráficos sobre o adensamento e cisalhamento.	0,574	
9	Controle e visualização do experimento	O usuário deverá ser capaz de visualizar os resultados finais.	0,561	
10	Histórico	O usuário deverá ser capaz acessar dados de um experimento pelo histórico.	0,459	O usuário poderá visualizar os gráficos e tabelas gerados durante o experimento.
11	Controle e visualização do experimento	O usuário deverá ser capaz de visualizar tabelas sobre o adensamento e cisalhamento.	0,459	

12	Controle e visualização do experimento	O usuário deverá ser capaz de visualizar as informações do experimento durante sua ocorrência.	0,458	
13	Histórico	O usuário deverá ser capaz de visualizar o histórico de experimentos.	0,306	
14	Configuração do sistema	O usuário deverá ser capaz de editar as as informações do sistema.	0,275	
15	Criação de um novo experimento	O usuário deverá ser capaz de visualizar o cálculo resultante das informações preenchidas.	0,268	
16	Armazenamento	Avisar o usuário quando a capacidade de armazenamento do dispositivo estiver baixa.	0,230	
17	Criação de um novo experimento	O usuário deverá ser capaz de preencher informações do experimento.	0,184	
18	Controle e visualização do experimento	O usuário deverá ser capaz de preencher variáveis ao final do experimento.	0,183	

Os requisitos funcionais do *backend* foram divididos nas seguintes categorias:

- Configuração: requisitos relacionados ao envio e recebimento de informações das configurações do sistema entre o *frontend*, camada de controle, *backend* e banco de dados.
- Histórico: requisitos relacionados ao envio e recebimento de dados de experimentos anteriores entre entre o *frontend*, *backend* e banco de dados.
- Experimento: requisitos relacionados ao envio e recebimento de dados do experimento atual entre entre o *frontend*, *backend*, camada de controle e banco de dados.
- Armazenamento: requisitos relacionados ao envio e recebimento de dados sobre a capacidade de armazenamento do dispositivo entre entre o *frontend*, *backend* e banco de dados.

Tabela 8 – Requisitos funcionais do *backend* do *software*

Número	**Categoria**	**Descrição**	**Observações**
1	Configuração	Estabelecer um canal de comunicação com a camada de controle.	Essa integração foi realizada utilizando *sockets* locais. Permitindo assim, a separação do códigos da interface do código da camada de controle.
2	Configuração	Enviar configuração das portas para a camada de controle para validação.	
3	Configuração	Enviar configuração das portas para o *frontend*.	
4	Configuração	Enviar configuração das portas para o banco de dados.	
5	Configuração	Receber configuração das portas do banco de dados.	
6	Configuração	Enviar informações do sistema/máquina para o banco de dados.	
7	Configuração	Receber informações do sistema/máquina do banco de dados.	
8	Configuração	Receber informações sobre alterações no sistema/máquina feitas pelo *frontend*.	
9	Histórico	Receber dados de experimentos anteriores do banco de dados.	
10	Histórico	Enviar dados de experimentos anteriores para o *frontend*.	
11	Experimento	Enviar criação de um novo experimento para o banco de dados e camada de controle.	Com isso a entidade no banco de dados é criada e a máquina começa o experimento.

12	Experimento	Enviar informações da criação do experimento para o banco de dados.	
13	Experimento	Enviar comandos do usuário para a camada de controle.	Os comandos podem ser: controle do envio de dados feito pela camada de controle, movimentação do carrinho com velocidade determinada, liberação da baixa pressão, liberação do outro tipo de pressão, ajuste de variáveis para o início da fase de cisalhamento e finalização do experimento. Mais detalhes na Seção 5.5.2.
14	Experimento	Receber dados do adensamento da camada de controle.	
15	Experimento	Enviar dados do adensamento para o banco de dados.	
16	Experimento	Enviar dados do adensamento para o *frontend*.	
17	Experimento	Receber dados do cisalhamento da camada de controle.	
18	Experimento	Enviar dados do cisalhamento para o banco de dados.	
19	Experimento	Enviar dados do cisalhamento para o *frontend*.	
20	Armazenamento	Receber mensagem sobre a capacidade de armazenamento do dispositivo do banco de dados.	

21	Armazenamento	Enviar mensagem sobre a capacidade de armazenamento do dispositivo para o *frontend*.	

Os requisitos funcionais do banco de dados foram divididos nas seguintes categorias:

- Experimento: requisitos relacionados ao recebimento, envio e salvamento dos dados de experimentos.

- Configuração: requisitos relacionados ao recebimento e salvamento dos dados das configurações.

- Armazenamento: requisitos relacionados ao envio da capacidade de armazenamento do dispositivo.

Tabela 9 – Requisitos funcionais do banco de dados do *software*

Número	Categoria	Descrição	Observações
1	Experimento	Receber dados do experimento do *backend*.	
2	Experimento	Salvar dados do experimento.	
3	Experimento	Enviar dados de algum experimento anterior para o *backend*.	
4	Configuração	Receber configurações do sistema/máquina/portas do *backend*.	
5	Configuração	Salvar configurações do sistema/máquina/portas.	
6	Armazenamento	Enviar mensagem quando o armazenamento do dispositivo estiver cheio para o *backend*.	

Tabela 10 – Requisitos não funcionais de *software*

Número	**Descrição**	**Observações**
01	A aplicação poderá ser acessada remotamente.	O acesso remoto reduz a dependência da presença física no laboratório para acompanhar o experimento e aumenta as chances de visualizar se falhas ocorreram. Como alguns experimentos duram vários dias, era necessário ir ao laboratório checar se algum problema ocorreu ou se o experimento continua funcionando normalmente.
02	A aplicação deverá salvar os dados de experimentos em um banco de dados local.	Para visualizar rapidamente os experimentos anteriores é necessário guardar os dados em um local de rápido e fácil acesso, uma opção para isso é utilizar um banco de dados local.
03	A aplicação deverá mostrar para o usuário o desenvolvimento dos gráficos e tabelas durante o experimento.	Essa visualização permite que o usuário realize suas análises durante o experimento e assim não precise esperar que o experimento seja concluído. Lembrando que alguns experimentos podem durar dias.
04	A aplicação deverá possuir uma interface com boa usabilidade para facilitar seu manuseio.	Um dos objetivos do *retrofit* é facilitar o uso de um equipamento, assim um software com péssima usabilidade quebraria esse objetivo.

05	A aplicação deverá salvar os dados dos experimentos em um banco de dados na nuvem, para evitar a perda de informações caso ocorra um problema no equipamento, ou a memória esteja ficando sem espaço.	É importante ter um serviço nuvem funcionando como backup, pois a perda de informações pode ser catastrófica. Por exemplo: a perda de 5 anos de informações de experimentos, pois o sistema operacional do equipamento se corrompeu.
06	A aplicação deverá reduzir a quantidade de *delays* no acesso remoto, que existem na visualização de informações e na realização de comandos.	Utilizar um sistema que contém grandes *delays* afeta negativamente a experiência do usuário(UX).
07	A *Raspberry Pi* deve estar conectada à internet.	O segundo requisito da Tabela 7 é sobre exportação de dados. Para isso ser possível é necessário que a *Raspberry Pi* esteja conectada à internet.

5.2 *Design*

Decidiu-se criar um protótipo para auxiliar na validação de requisitos. Por ser um design de uma tela sensível, foi utilizado o livro do autor Clark (2015) para guiar o processo. Este livro descrevia sobre os variados movimentos que a mão pode realizar, como toque e arrastar, e também sobre o problema em utilizar botões em telas sensíveis ao toque.

Seguindo estas indicações, a quantidade de botões foi reduzida significativamente, porém não foi possível eliminá-los.

A Figura 15 mostra uma página do protótipo de alta fidelidade, que pode ser acessado pelo link: https://www.figma.com/proto/Omi3ilXJxwYgrQCI1ZsBIr/Prot%C3%B3tipo-TCC1-Vers%C3%A3o-3?node-id=205%3A1541&starting-point-node-id=205%3A1541

A Figura 16 mostra o padrão de cores utilizado. A cor #2986A4 63, recebeu este nome pois utiliza a codificação hexadecimal e tem 63% de transparência. Por esta razão

Figura 15 – Página do histórico no protótipo

Fonte: Elaborado pelo autor, 2022

existem duas cores #2986A4.

Figura 16 – Padrão de cores

Fonte: Elaborado pelo autor, 2022

5.3 Arquitetura

Esta Seção tem como objetivo apresentar os diagramas construídos que compõe a arquitetura completa.

5.3.1 Diagrama de blocos

A Figura 17 representa o sistema completo, suas conexões e o que será realizado neste trabalho. Assim segue abaixo uma lista dos componentes e suas principais funções.

- *Frontend* local: Permite a utilização do software através de uma interface de boa qualidade. Acessado diretamente na tela sensível ao toque, como demonstrado na Fig. 28. Será utilizado o QT em sua implementação, como mencionado na Seção 4.2.1.1.

- *Frontend* remoto: Permite a utilização do software através de uma interface de boa qualidade. Pode ser acessado através da internet.

- *Backend*: Conecta os diversos componentes, evitando a conexão direta entre eles e por consequência reduzindo suas complexidades. Isso torna o sistema menos acoplado.

- Máquina de ensaio de cisalhamento direto: É a máquina de ensaio de cisalhamento direto, responsável por realizar o experimento.

- Banco de dados local: Salva localmente os dados dos experimentos, para permitir o acesso a informação posteriormente. Será usado o SQLite em sua implementação, como mencionado na Seção 4.2.2.

- Banco de dados na nuvem: É um *backup* do banco de dados local. Assim, garante que mesmo que um problema ocorra com a máquina física, os dados não serão perdidos.

- Camada de controle: Camada de comunicação intermediária entre o *Backend* e a máquina de ensaio de cisalhamento direto. Parte realizada pelo autor (PEREIRA, 2022).

A Figura 18 representa o componente do *Backend* com mais detalhes. É mostrado as diversas funções que ele realiza, como a consulta e o salvamento de dados em um banco de dados.

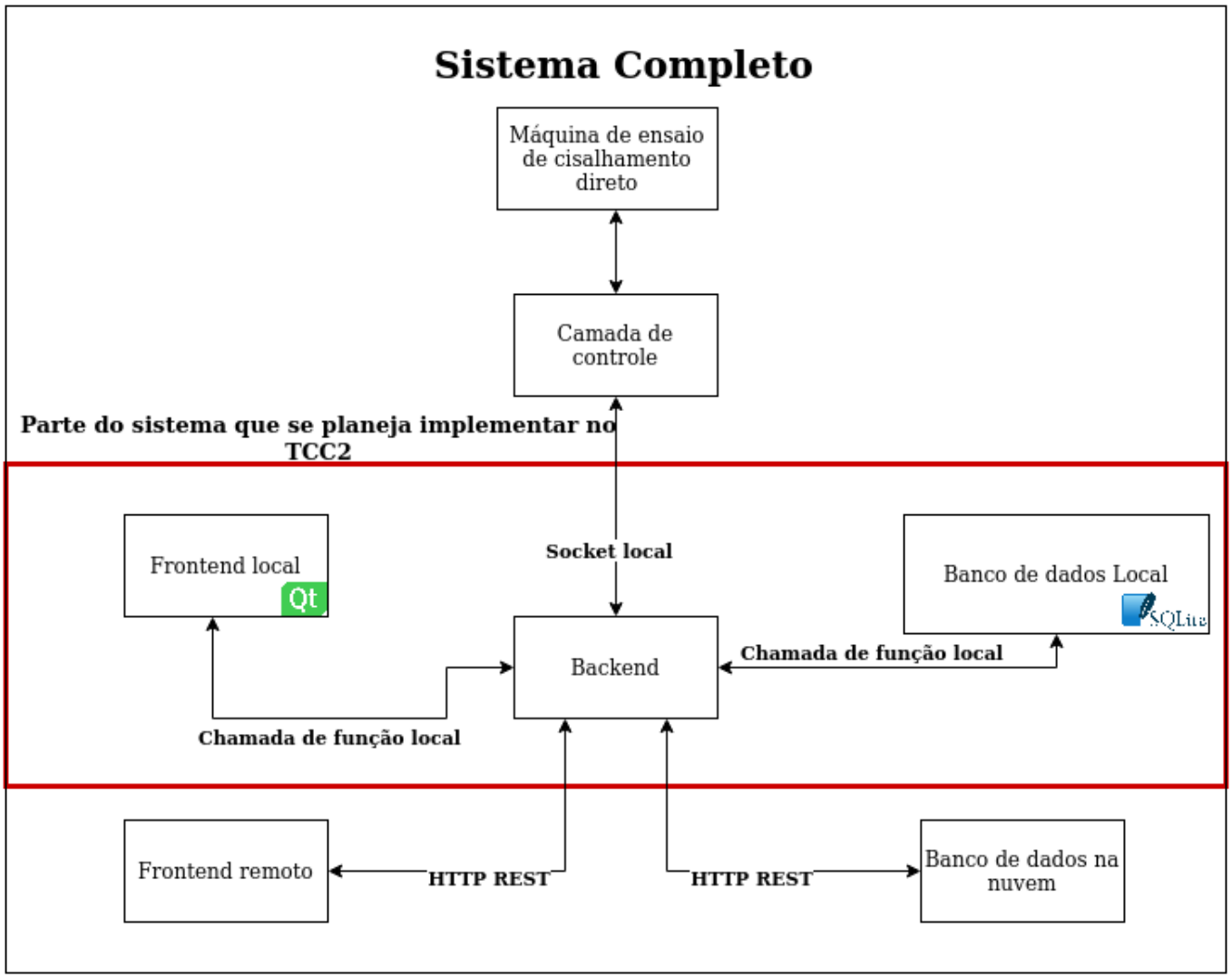
A Figura 19 representa os componentes Banco de dados local e Banco de dados remoto. Os 2 bancos tem as mesmas funções, porém têm sentidos diferentes no sistema como explicado na lista anterior.

As Figuras 20 e 21 representam, respectivamente, as principais funções dos componentes *Frontend* local e *Frontend* remoto. A diferença entre eles é que o *Frontend* remoto adiciona uma função de *login* e cadastro de usuários. Garantindo assim o acesso restrito à máquina de ensaio de cisalhamento direto.

5.3.2 Organograma de hierarquia de controle

A Figura 22 representa a visualização da hierarquia de controle do *Frontend* Local. As 3 funções “Criar novos experimentos”, “Configurar o sistema” e “Visualizar experimentos” foram decompostas, permitindo assim um melhor entendimento do sistema.

Figura 17 – Diagrama de blocos do sistema completo



Fonte: Elaborado pelo autor, 2022

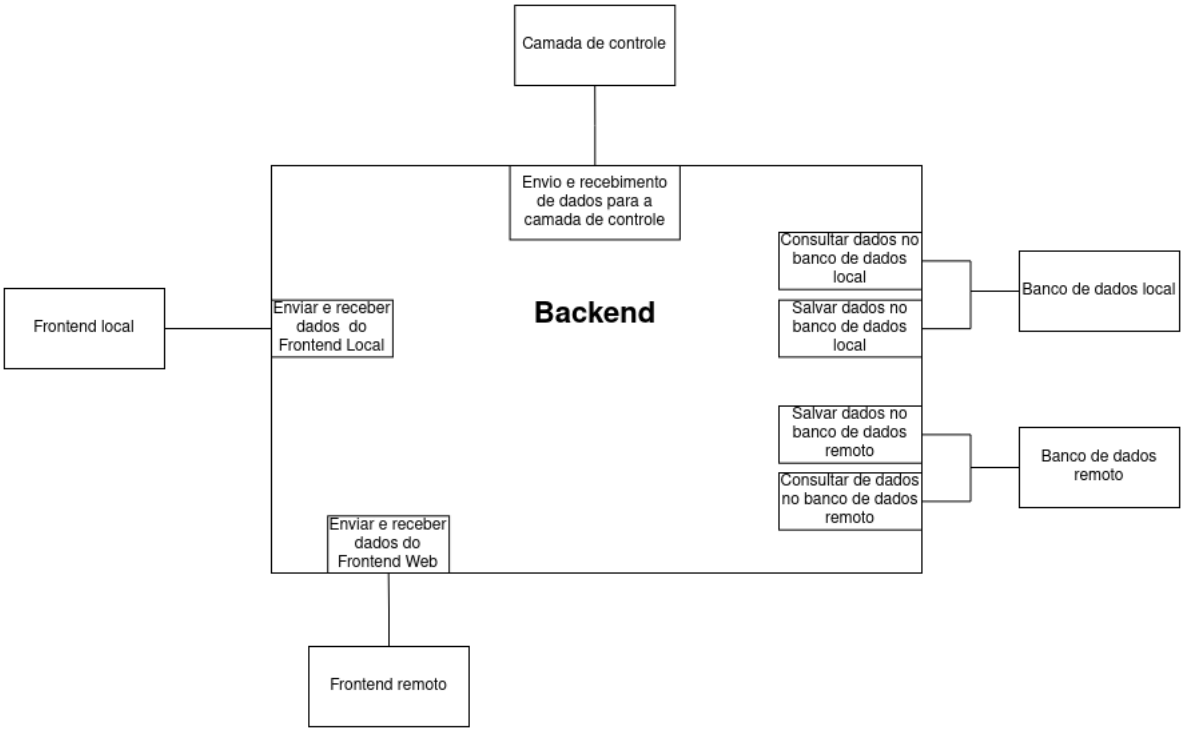
A Figura 23 representa a visualização da hierarquia de controle do *Frontend* remoto. Mostrando a adição da função de login e cadastro de usuários, mas mantendo ainda todas as funções do componente *Frontend* local.

A Figura 24 demonstra de maneira mais clara a conexão entre as funções representadas na Fig. 18.

5.3.3 Fluxograma

Para entender melhor o fluxo de criação e execução de um experimento, foi criado um fluxograma que pode ser observado na Fig. 25.

Figura 18 – Diagrama de blocos do *backend*

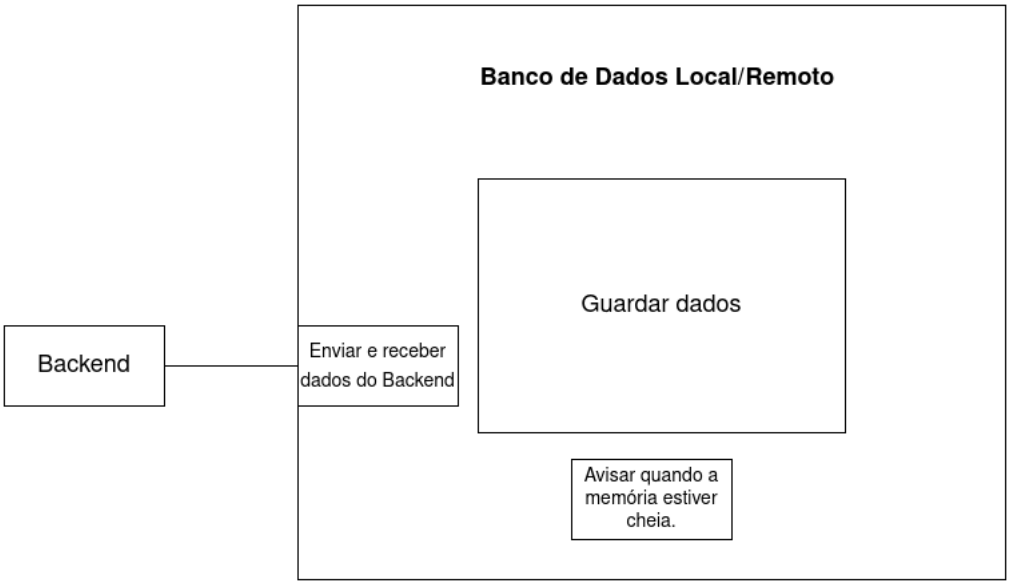


Fonte: Elaborado pelo autor, 2022

5.3.4 Diagrama de classes

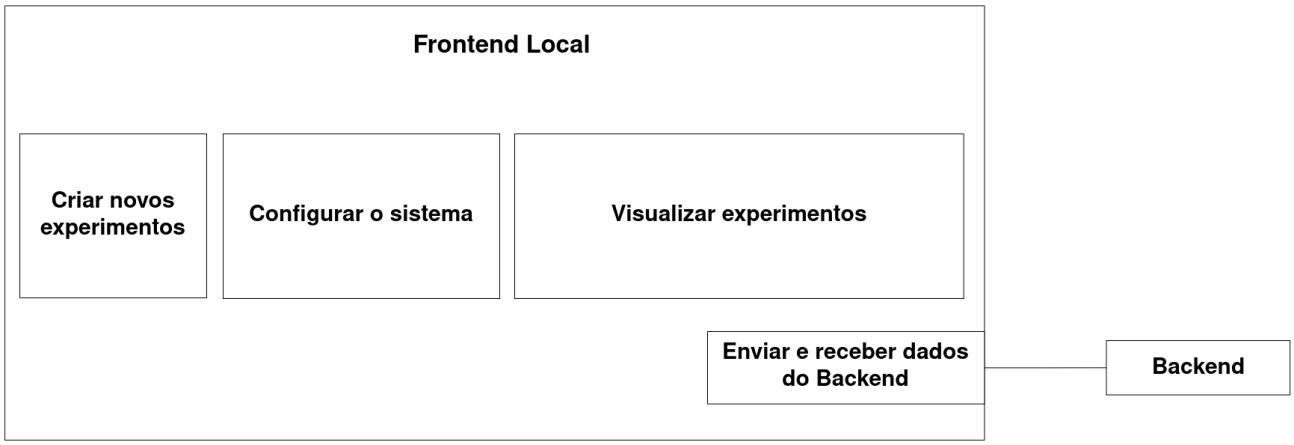
Para auxiliar no entendimento do código fonte, foi criado um diagrama de classes, representado na Fig. 26. Nesta representação, ocorreu uma redução dos atributos e métodos para permitir a visualização em apenas uma página.

Figura 19 – Diagrama de blocos do banco de dados local/remoto



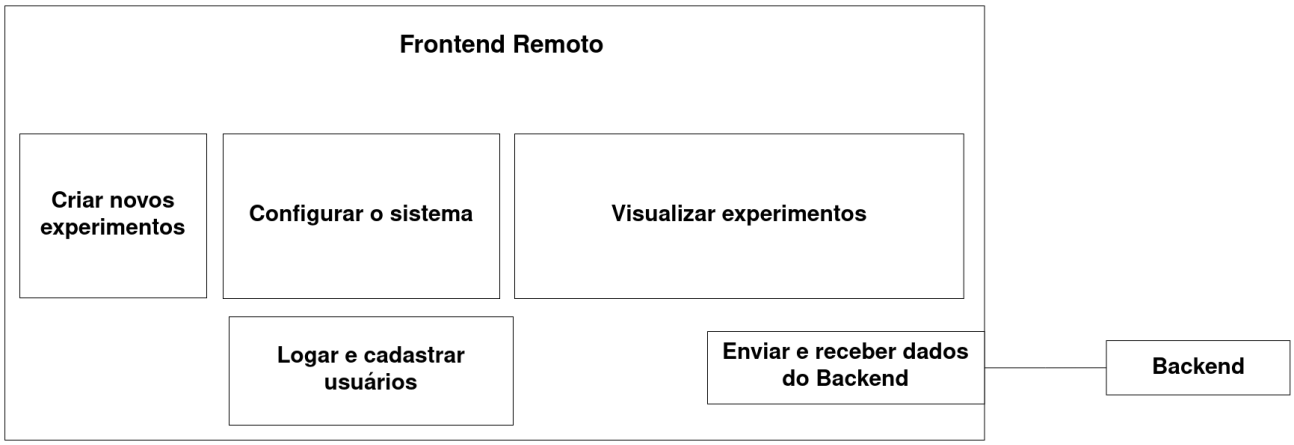
Fonte: Elaborado pelo autor, 2022

Figura 20 – Diagrama de blocos do *frontend* local



Fonte: Elaborado pelo autor, 2022

Figura 21 – Diagrama de blocos do *frontend* remoto



Fonte: Elaborado pelo autor, 2022

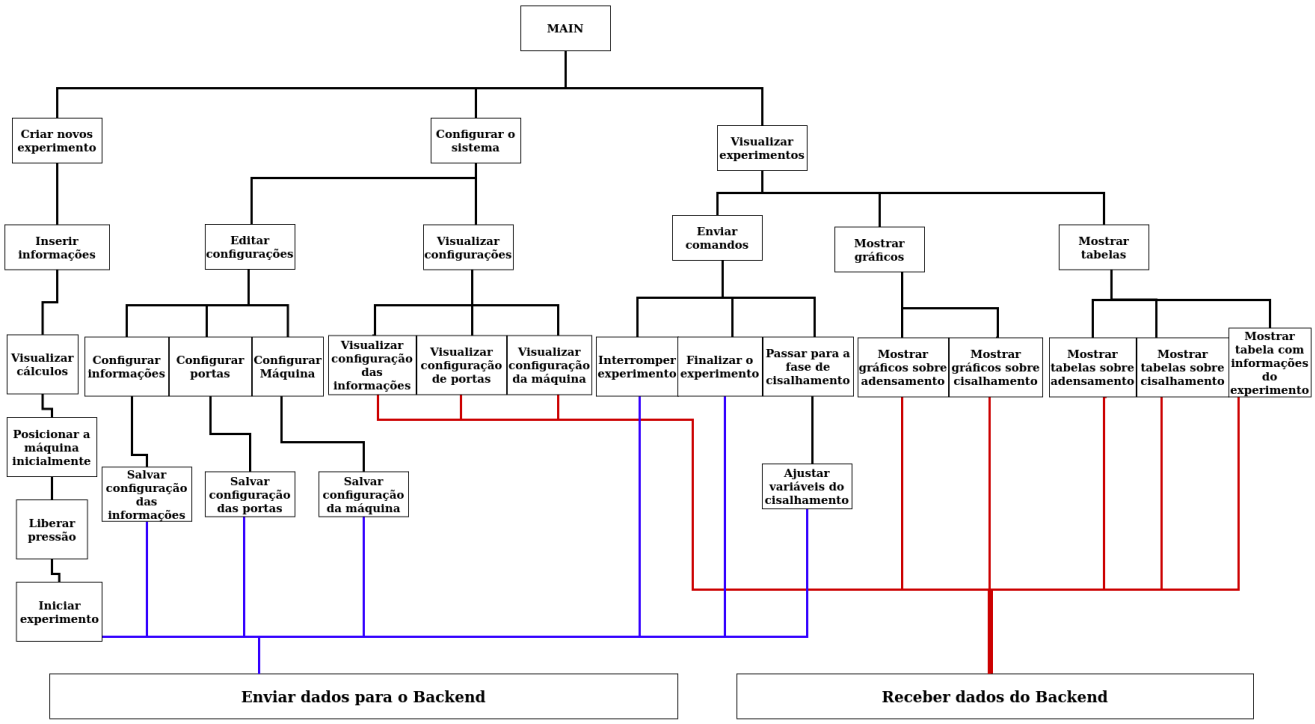
5.4 Banco de dados

Para apresentar a organização do banco de dados foi criado um diagrama de entidade relacionamento, representado na Fig. 27.

Os relacionamentos são classificados da seguinte maneira:

- Uma EXPERIMENT_TABLE pode ter de 0 a N DENSIFICATION_TABLE e uma DENSIFICATION_TABLE pode pertencer a apenas uma EXPERIMENT_TABLE.
- Uma EXPERIMENT_TABLE pode ter de 0 a N SHEAR_TABLE e uma SHEAR_TABLE pode pertencer a apenas uma EXPERIMENT_TABLE.
- Uma EXPERIMENT_TABLE pode ter de 0 ou uma FINAL_VARIABLES_TABLE e uma FINAL_VARIABLES_TABLE pode pertencer a apenas uma EXPERIMENT_TABLE.

Figura 22 – Hierarquia do *frontend* local



Fonte: Elaborado pelo autor, 2022

- Uma EXPERIMENT_TABLE pode ter de 0 ou uma SAMPLE_TABLE e uma SAMPLE_TABLE pode pertencer a apenas uma EXPERIMENT_TABLE.

5.5 Funcionamento do sistema completo

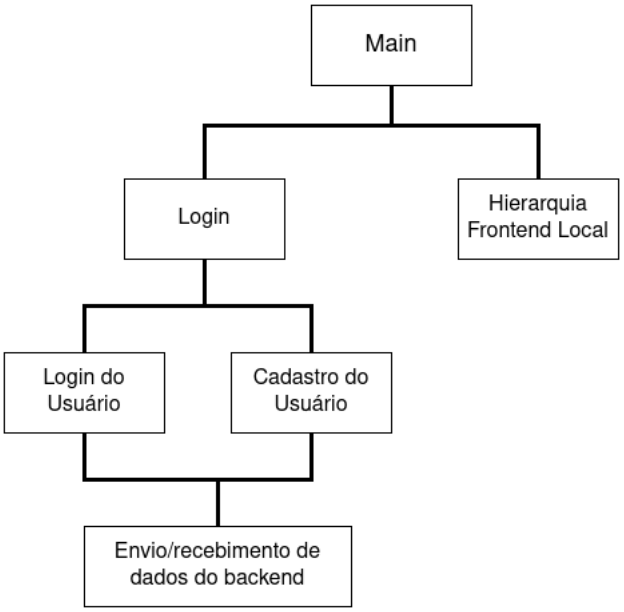
Este tópico tem como função esclarecer o funcionamento completo do sistema, visto na Fig. 17. Será explicado o funcionamento geral da camada de interface (parte em vermelho da Fig. 17), o funcionamento geral da camada de controle, como elas se comunicam e por fim o funcionamento geral do sistema.

Como mencionado na Seção 1.4, o objetivo da camada de interface é permitir a utilização da máquina de ensaio de cisalhamento direto por meio de uma tela sensível ao toque conectada a uma *Raspberry Pi*. Porém a comunicação não ocorre diretamente com a máquina, existe uma camada intermediária chamada camada de controle.

De acordo com Pereira (2022), a camada de controle é responsável por controlar os sensores e atuadores da máquina de cisalhamento direto, através de portas seriais ou por entradas usb (*universal serial bus*). Ela também é responsável por realizar de forma autônoma o monitoramento, controle, e envio dos dados para o software de alto nível (camada de interface), garantindo total sincronismo nos resultados.

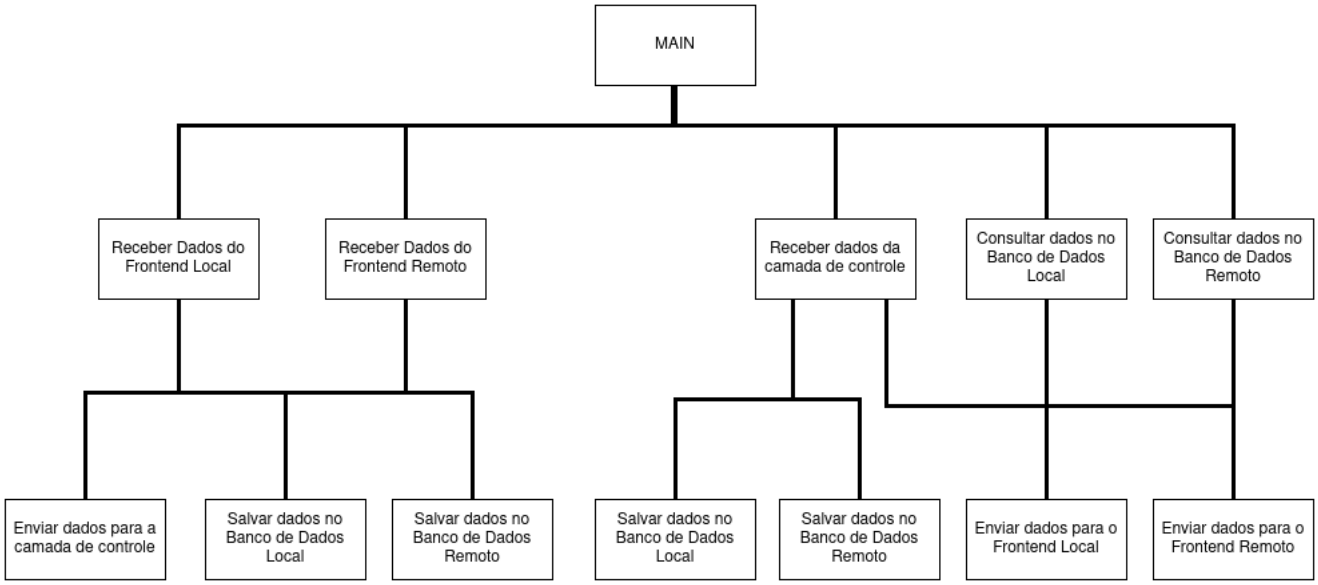
A comunicação entre essas duas camadas (interface e controle) ocorre por meio de

Figura 23 – Hierarquia do *frontend* remoto



Fonte: Elaborado pelo autor, 2022

Figura 24 – Hierarquia do *backend*



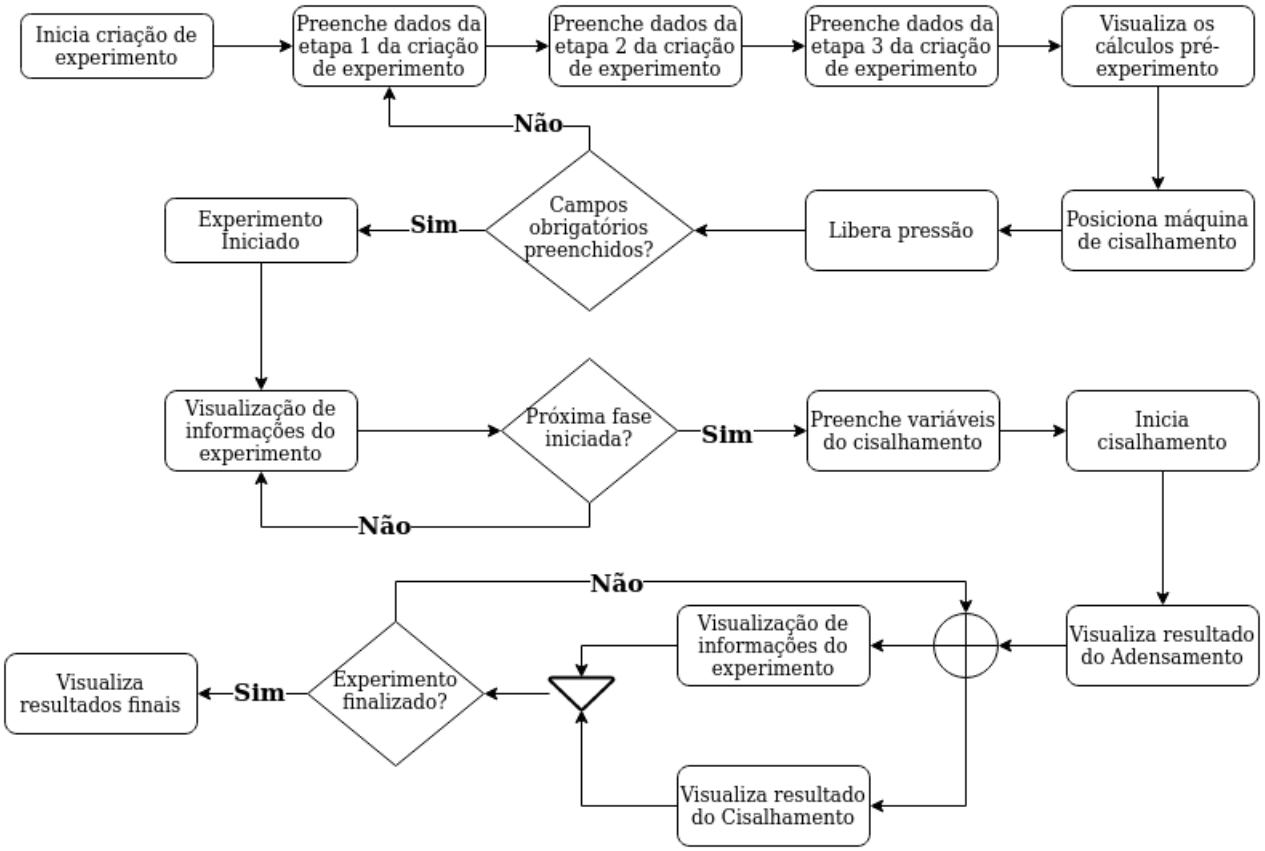
Fonte: Elaborado pelo autor, 2022

sockets locais. Esta escolha foi feita, pois este método têm garantias no envio e recebimento de dados, reduzindo a necessidade de desenvolver métodos que verifiquem a integridade da mensagem.

Assim, para realizar o controle da máquina, a camada de interface envia um comando (Seção 5.5.2) para a camada de controle (por meio dos sockets locais) e a camada de controle realiza as operações devidas com a máquina de cisalhamento.

A camada de controle também envia dados (como deslocamento vertical, carga

Figura 25 – Fluxograma da criação e execução de um experimento



Fonte: Elaborado pelo autor, 2022

vertical e etc), desta maneira foi necessário a criação de 2 sockets locais. O primeiro responsável por enviar os comandos da camada de interface para a camada de controle e o segundo, responsável por enviar os dados da camada de controle para a camada de interface.

Na *Raspberry Pi* este processo é possível com criação a de 2 processos separados, o processo da camada de interface e o processo da camada de controle.

Com este sistema implementado, o usuário conseguirá manusear a máquina de cisalhamento utilizando apenas o *software* da camada de interface, que poderá ser visto na tela sensível ao toque.

5.5.1 Estratégia de *polling*

Na Seção 5.5, foi explicado que a comunicação é realizada por meio de sockets locais. Esta seção irá elucidar a estratégia de comunicação utilizada pelos sockets.

A estratégia escolhida foi a de *polling*. A camada de interface e a camada de controle criam servidores que realizam, frequentemente, operações de *read* nos seus respectivos arquivos de socket, para verificar se um novo dado foi enviado.

5.5.2 Comandos

A comunicação entre a camada de controle e o *backend* é realizada através de sockets locais e para tornar a comunicação efetiva, foram criados 5 comandos:

- Comando 0: Controla o envio de dados feito pela camada de controle. Os 2 parâmetros utilizados são: ativado e o período de amostragem. Para o *backend* começar a receber dados, é necessário o envio da variável "ativado" com o valor verdadeiro e selecionado o período de amostragem na variável "período de amostragem. Para parar de receber dados, envie a variável "ativado" com o valor falso.
- Comando 1: Movimenta o carrinho para a direita ou esquerda com uma velocidade constante durante o posicionamento inicial. O parâmetro utilizado é a variável "velocidade" que determinará a velocidade e o sentido de movimentação do carrinho.
- Comando 2: Libera a baixa pressão, que é utilizada antes do início do experimento.
- Comando 3: Libera a alta pressão, que é utilizada no início do experimento.
- Comando 4: Configura a distância e velocidade, utilizadas na fase de cisalhamento. Os 2 parâmetros utilizados são "distância" e "velocidade".
- Comando 5: Encerramento do experimento. Neste comando a máquina encerra todas as operações anteriores, como movimentação e envio de dados.

5.6 Classificação

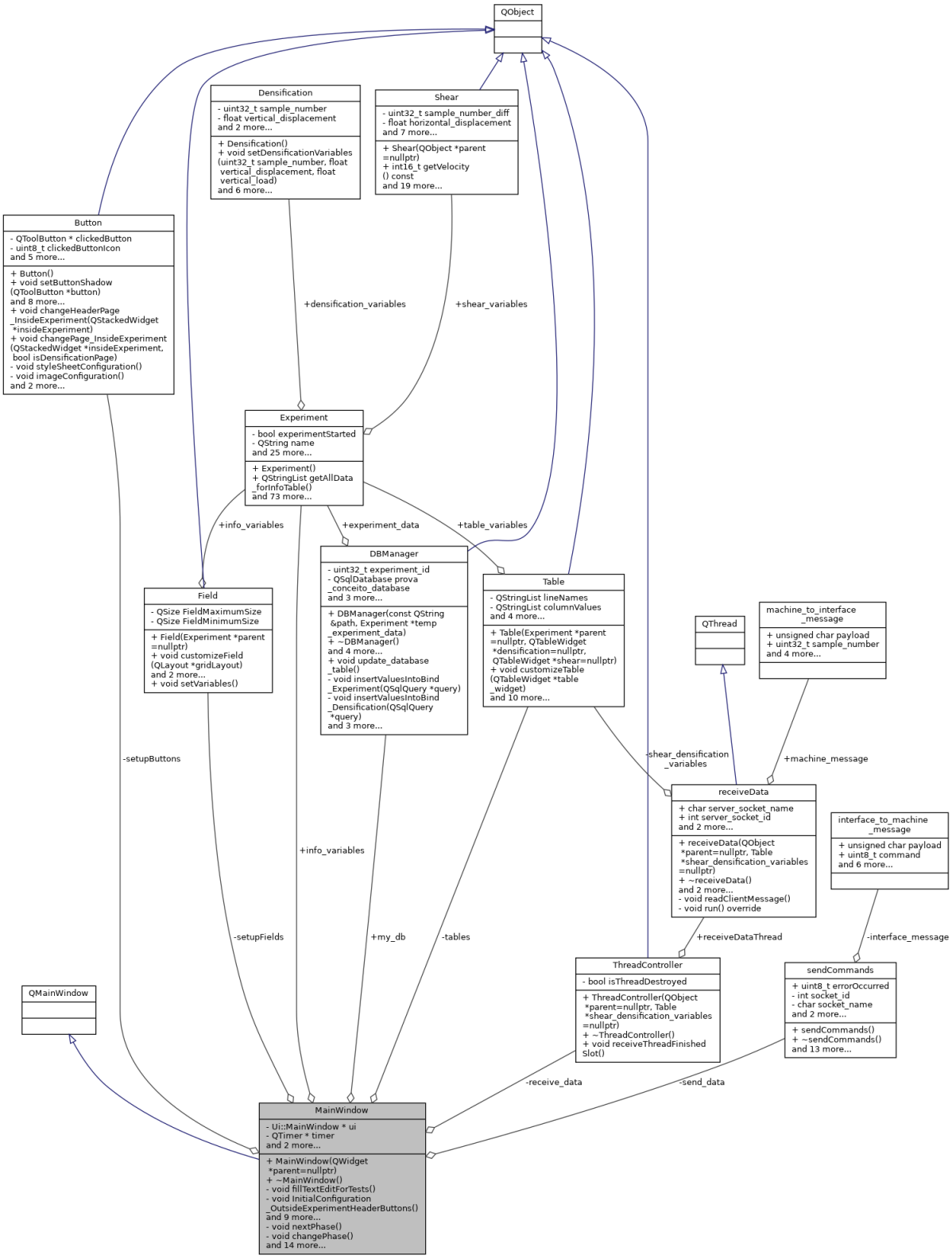
Na Seção 2.4.1 foi apresentada as diversas classificações que um sistema embarcado pode ter. Desta maneira, a Tabela 11 demonstra a classificação deste projeto.

Tabela 11 – Classificação do sistema

Categoria	**Resultado**	**Observações**
Escala	Sofisticada	A *Raspberry Pi 4* integrada a máquina de ensaio de cisalhamento direto torna o sistema de classificação sofisticada. A *Raspberry Pi 4* suporta arquitetura de 64 bits, então ela sozinha poderia ser classificada como sistema de escala sofisticada.
Tempo Real	Não crítico	O sistema irá mostrar os dados dos sensores para o usuário a cada 1 segundo aproximadamente, porém é possível haver latência sem que o sistema seja prejudicado.

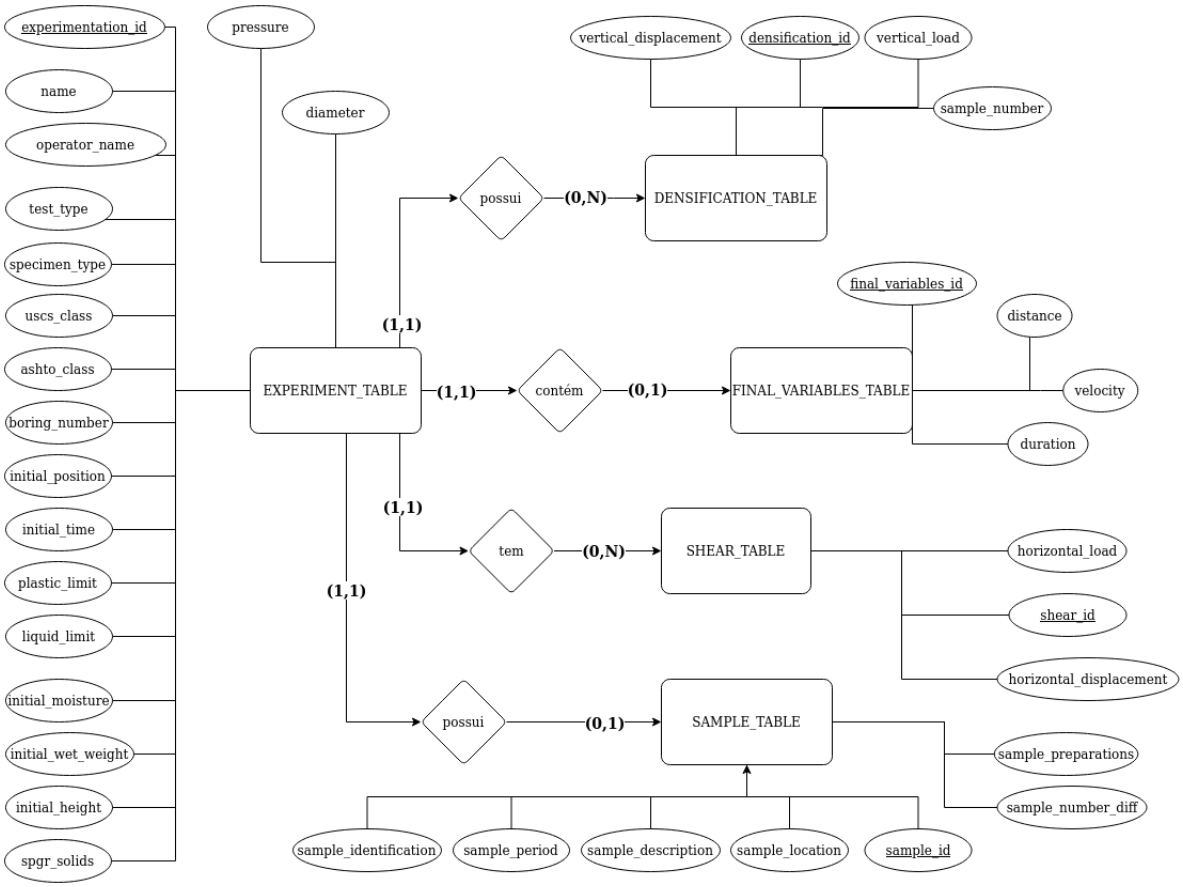
Tolerância a falha	Não silenciosa	O sistema deve avisar na tela que algum erro ocorreu, pois esse erro pode tornar inválido o resultado do experimento.
Integração	Independente	A principal parte do sistema se encontra na máquina de ensaio de cisalhamento direto, mostrado na Fig. 28. Esta parte do sistema consegue funcionar independentemente da parte de acesso remoto. A adição do acesso remoto é um extra. Assim, o sistema pode ser considerado independente.
Mobilidade	Fixo	A *Raspberry Pi 4* ficará fixa na máquina de ensaio de cisalhamento direto, como mostrado na Fig. 28. Está máquina não tem mobilidade, ela fica parada no laboratório. Assim, o sistema pode ser considerado de mobilidade fixa.

Figura 26 – Diagrama de classes



Fonte: Elaborado pelo autor, 2022

Figura 27 – Diagrama de entidade e relaciomento



Fonte: Elaborado pelo autor, 2022

6 Resultados

Este capítulo tem como objetivo apresentar os resultados obtidos pelo autor no TCC.

6.1 Máquina de cisalhamento

O trabalho realizado por (PEREIRA, 2022) teve sucesso em trocar o painel utilizado anteriormente e inserir a tela sensível ao toque. A Figura 28 mostra como ficou a máquina, na qual a presença de um computador externo não é mais necessário.

Figura 28 – Sistema com tela sensível ao toque

Fonte: Elaborado pelo autor, 2022

6.2 Protótipo e requisitos

Para entender o que deveria ser feito no sistema, foram levantados requisitos (Seção 5.1.1) e criado um protótipo (Seção 5.2). Após o levantamento inicial de requisitos, o protótipo foi criado com o intuito de ajudar na validação deles. Durante essa validação alguns problemas foram levantados:

- Navegação não está boa.
- Não é útil criar perfis de experimento, pois cada experimento terá seus próprios parâmetros.
- Faltou entendimento do processo completo do experimento, pois as variáveis de distância e velocidade devem ser configuradas no final da fase de adensamento.

Assim, levando isso em consideração, foi criada uma segunda versão do protótipo e a tabela de requisitos funcionais foi atualizada. A versão 1 do protótipo pode ser encontrada no link https://www.figma.com/proto/utPMFjNwjgXFNLhxabE43z/Prot%C3%B3tipo-TCC1?node-id=2%3A3&scaling=scale-down&page-id =0%3A1&starting-point-node-id=2%3A3 e a versão 2 pode ser encontrada no link https://www.figma.com/proto/yOuSDHTD1lltbPFiL55dUg/Prot%C3%B3tipo-TCC1-Vers%C3%A3o-2?node-id=205%3A1541&scaling=scale-down&page-id=0%3A1&starting-point-node-id=205%3A1541.

Uma das mudanças pode ser observada nas Fig. 29 e Fig.30.

Figura 29 – Página configuração das portas versão 1

Fonte: Elaborado pelo autor, 2022

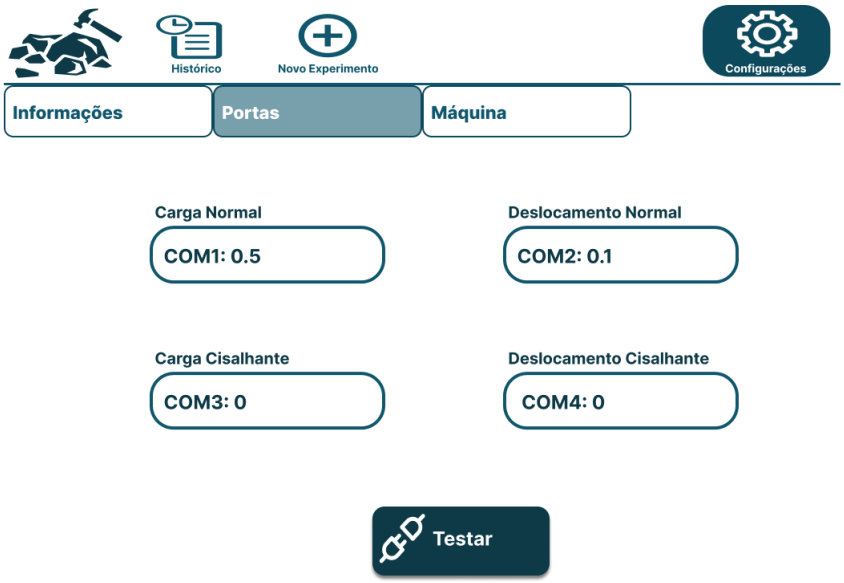
Durante o TCC2, foi observado que o segundo protótipo ainda não resolvia todos os problemas, pois novas questões foram levantadas, como:

- Posicionamento deveria ocorrer no início do experimento e não mais durante a configuração de variáveis da fase de cisalhamento;
- O usuário deve ficar preso dentro do experimento durante sua execução;
- Os comandos de "cancelar, iniciar cisalhamento e finalizar" devem aparecer em todas as páginas, e
- A informação sobre qual fase o experimento está, deve aparecer em todas as telas.

Desta maneira, foi criada a terceira versão do protótipo, disponível no link: https://www.figma.com/proto/Omi3ilXJxwYgrQCI1ZsBIr/Prot%C3%B3 tipo-TCC1-Vers%C3%A3o-3?node-id=205%3A1541&starting-point-node-id=205%3A1541.

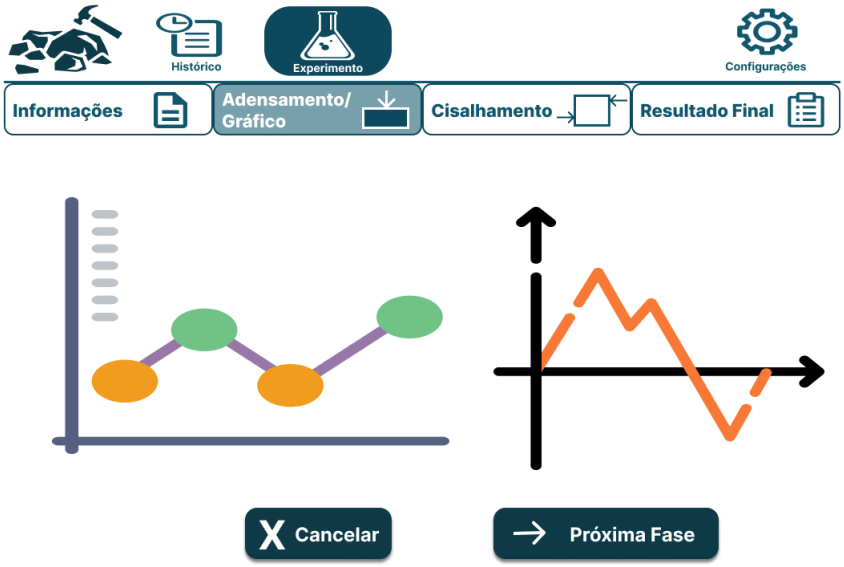
Uma dessas mudanças pode ser observada nas Fig. 31 e 32.

Figura 30 – Página configuração das portas versão 2



Fonte: Elaborado pelo autor, 2022

Figura 31 – Página de gráficos do adensamento versão 2

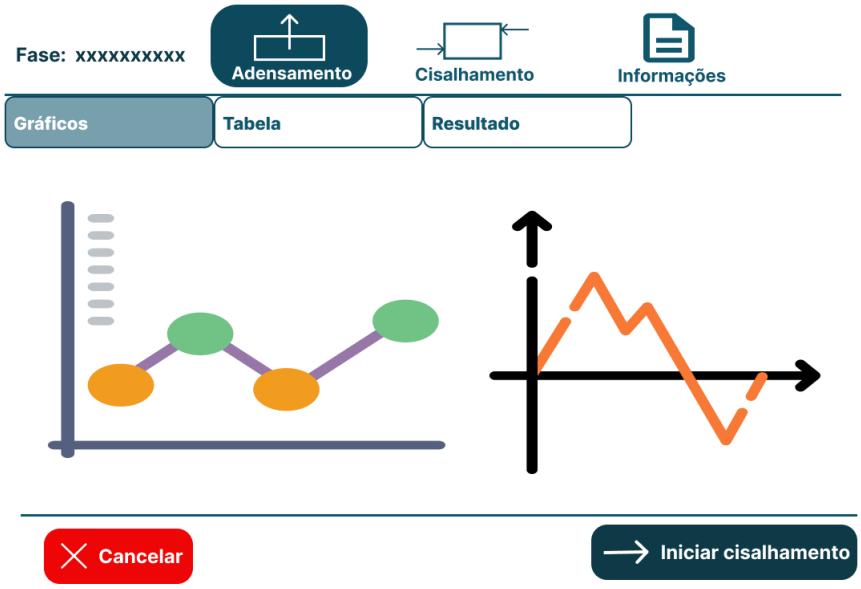


Fonte: Elaborado pelo autor, 2022

6.2.1 Priorização de requisitos

Na Seção 2.8.1 foi discutido as maneiras nas quais os requisitos iriam ser priorizados. Durante o TCC1 foi possível realizar a primeira etapa da priorização em 3 níveis (Fig. 10), ou seja os requisitos foram divididos em alta, baixa e média prioridade, porém os requisitos de alta prioridade não foram redivididos entre alta, muito alta e extremamente alta prioridade. No TCC2, a priorização foi concluída, e os resultados podem ser observados nas Tabelas 6 e 7 (quanto maior o número maior a prioridade).

Figura 32 – Página de gráficos do adensamento versão 3



Fonte: Elaborado pelo autor, 2022

6.2.2 Requisitos implementados

Com a priorização de requisitos concluída, foi iniciado o desenvolvimento do código, Seção 6.3. Os requisitos cumpridos durante esse desenvolvimento podem ser observados na Tabela 12:

Tabela 12 – Classificação do sistema

Tabela de origem	**Número dos Requisitos Cumpridos**
Requisitos funcionais do frontend software de alta prioridade (Tab. 6).	2,3,4,5
Requisitos funcionais do frontend software de média e baixa prioridade (Tab. 7).	2,8,9,11,12,15,17
Requisitos funcionais do backend do software (Tab. 8).	1,11,12,13,14,15,16,17,18,19
Requisitos funcionais do banco de dados do software (Tab. 9).	1,2
Requisitos não funcionais de software (Tab. 10).	2,3,4,7

Com estes requisitos concluídos, pode-se resumir o resultado da seguinte maneira:

- O usuário consegue criar um experimento (preenchendo os campos necessários);
- O usuário consegue ver dados das tabelas e dos gráficos de adensamento e cisalhamento sendo gerados;

- O usuário consegue visualizar os resultados das fases de adensamento e cisalhamento;
- O usuário consegue passar da fase de adensamento para a fase de cisalhamento;
- O usuário consegue finalizar ou cancelar o experimento;
- Os dados necessários para a visualização do experimento no histórico, são salvos no banco de dados. Observação: Não foi implementada a página de histórico e,
- O usuário consegue exportar dados das tabelas para um serviço na nuvem, atualmente o Github.

6.3 Código

Utilizando como base os artefatos do Capítulo 5, diagramas, protótipo e etc, foi realizada a implementação do código. A Figura 26 mostra a organização das classes e pode-se observar, que a maioria delas se relacionam com a MainWindow, pois esta é a classe principal da interface. No total foram criadas 13 classes e 2 *structs*. A Tabela 13 categoriza essas estruturas.

Tabela 13 – Classificação do sistema

Categoria	**Estruturas**
Interface	Button,Field, MainWindow, Table e Charts
Comunicação	receiveData, sendCommands, ThreadController, interface_to_machine_message e machine_to_interface_message
Experimento	Densification, Experiment e Shear
Banco de dados	DBManager
Exportação	exportData

Cada categoria pode ser descrita da seguinte maneira:

- Interface: estruturas responsáveis pela interface, responsáveis pela estilização e comportamento dos elementos presentes na tela. Está atrelada ao *frontend* local, apresentado na Fig. 20.
- Comunicação: estruturas responsáveis pela comunicação com a camada de controle, como o envio e recebimento de dados. Está atrelada ao *backend*, apresentado na Fig. 18 .
- Experimento: estruturas responsáveis por guardar, temporariamente, e calcular as variáveis do experimento que serão utilizadas pela aplicação. Está atrelada ao *backend*, apresentado na Fig. 18.

- Banco de dados: responsável por salvar os dados, necessários, em um banco de dados para a visualização do experimento na página de histórico. Está atrelada ao banco de dados local, apresentado na Fig.19.

- Exportação: responsável por exportar os dados de uma tabela escolhida para um servidor na nuvem. Atualmente este servidor é o github. Está atrelado ao *backend*, apresentado na Fig. 18.

As Figuras 33, 34, 35 e 36 mostram o estado do software atual.

O código pode ser acessado pelo link https://github.com/darmsDD/backupCodigoTCC2.

6.3.1 Uso da aplicação

Esta seção descreve como o usuário deve interagir com a aplicação construída. O fluxo escolhido teve como base a explicação da Seção 2.2 e o fluxograma na Fig. 25.

Na primeira etapa o usuário deve preencher as informações necessárias para o experimento, representada nas Fig. 33a, 33b e 33c. Após este preenchimento dos dados, será possível visualizar o resultado de alguns cálculos realizados, apresentado na Fig. 33d.

Na próxima etapa, o usuário deve movimentar o carrinho para a posição inicial desejada (Fig. 33e) e liberar a baixa pressão para o início do experimento (Fig. 33f e 34a). A Figura 34b mostra que não é possível realizar o posicionamento depois que foi liberado a baixa pressão. Para posicionar novamente o carrinho é necessário desligar a baixa pressão.

Ao iniciar o experimento, será apresentado os gráficos da fase de adensamento sendo gerados (Fig. 35a). Neste ponto, a aplicação contém 3 páginas principais, a página de adensamento, a página de cisalhamento e a página com as informações preenchidas e geradas na etapa de criação do experimento (Fig. 35d). Na parte inferior de todas as páginas poderá-se visualizar os comandos disponíveis no momento, sendo eles "'Cancelar"' (cancela o experimento) e "'Iniciar cisalhamento"' (inicia a fase de cisalhamento) ou "'Finalizar"' (finaliza o experimento).

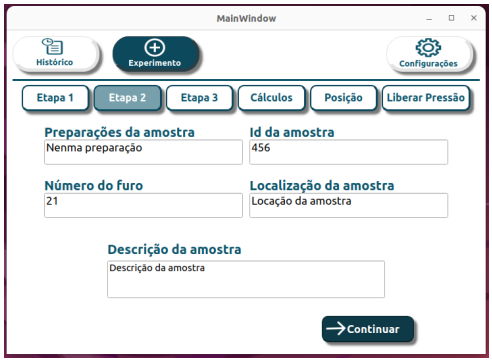
As páginas de adensamento e cisalhamento possuem 3 sub-páginas, Gráficos (Fig. 35a e 36c), Tabela (Fig. 35b e 36b) e Resultado (Fig. 35c e 36d). Ao clicar nas páginas de Gráficos e Tabela, poderá-se visualizar a chegada de novos dados, enviados pela camada de controle. Clicando-se em resultados, será mostrado os resultados atuais.

O fluxo do experimento, apresentado na Seção 2.2, explica que, inicialmente, se realiza o adensamento da amostra e posteriormente o cisalhamento. Assim, para mudar de fase na aplicação, é necessário clicar no botão "'Iniciar cisalhamento"' e preencher os

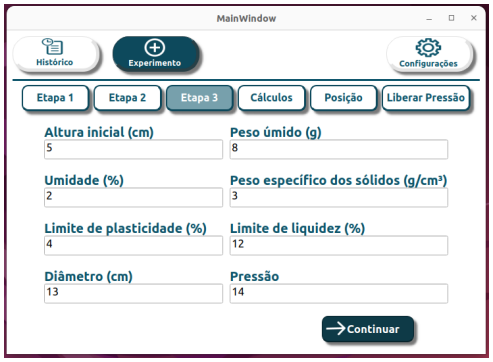
Figura 33 – Páginas de criação de experimentos

(a) Etapa 1

(b) Etapa 2

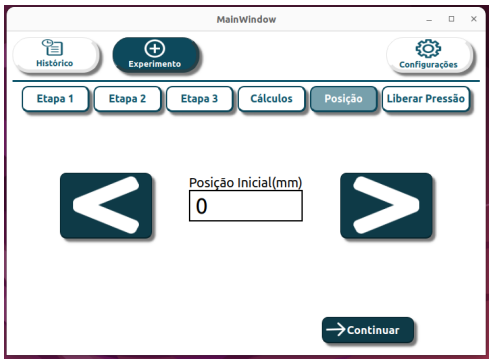


(c) Etapa 3



(d) Cálculos

(e) Posicionamento



(f) Liberar pressão

Fonte: Elaborado pelo autor, 2022

dados de velocidade e distância (Fig. 36a). Com a mudança de fase, os gráficos e tabelas de cisalhamento começaram a ser gerados.

Por fim, quando o usuário determinar que o experimento se encerrou, ele deve clicar no botão “‘Finalizar”’.

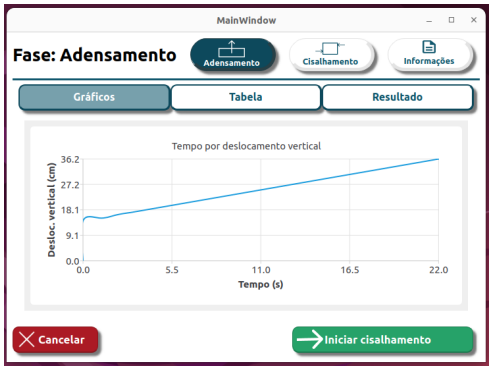
Figura 34 – Páginas de criação do experimento 2

(a) Parar a pressão (b) Posicionamento bloqueado

Fonte: Elaborado pelo autor, 2022

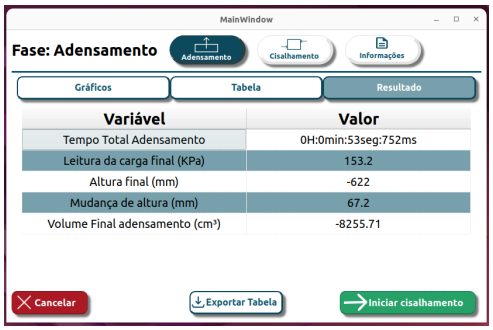
Figura 35 – Páginas de adensamento e informações

(a) Gráfico



(b) Tabela

(c) Resultado



(d) Informações

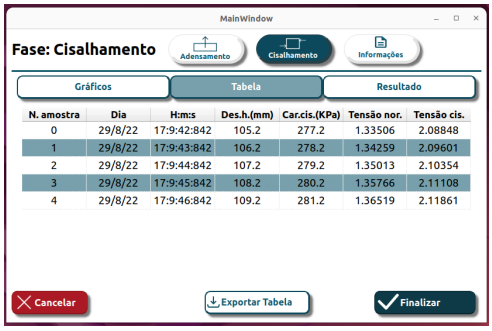
6.3.2 Testes

Esta seção descreve os testes realizados com a camada de controle simulada e os testes com a camada de controle real.

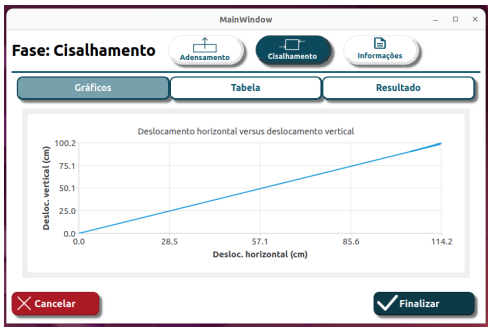
Figura 36 – Páginas de cisalhamento

(a) Inicia a fase de cisalhamento

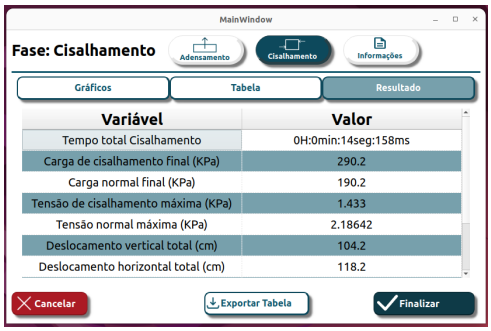
(b) Tabela



(c) Gráfico



(d) Resultado



6.3.2.1 Testes simulando a camada de controle

Foram realizados testes envolvendo apenas a camada de comunicação, garantindo que os dados estejam sendo transmitidos de maneira correta. Assim, 3 classes foram criadas, machineClient (responsável por simular o envio de dados da camada de controle), machineServer (responsável por simular o recebimento de dados pela camada de controle) e tst_sockettests.cpp (responsável por realizar os testes). Esse teste tinha o intuito de descobrir, qual o período de amostragem mínimo que garantisse uma comunicação sem falhas.

Foram utilizados 3 períodos de amostragem, 1ms , 3ms e 5ms. Em cada teste foi realizado 10000 envios de dados da camada de controle para o *backend* e outros 10000 envios do *backend* para a camada de controle. Este teste foi repetido 3 vezes para cada período de amostragem. Com 1 ms houve 59992 sucessos e 8 falhas, com 3 ms foram 59999 sucessos e 1 falha e por fim com 5 ms teve 60000 sucessos e 0 falhas.

As falhas foram insignificativas, mesmo com 1ms, ocorreu um valor máximo de 0.013% de erro (8 falhas/60000 tentativas). Contudo, a interface não consegue realizar suas atualizações com esta velocidade. Assim, 2 soluções são possíveis, aumentar o período de amostragem para 100ms ou utilizar uma estratégia de *buffers* e realizar as atualizações 1 vez a cada 20 períodos de amostragem, ou seja , a cada 100ms.

O período de amostragem padrão, durante o experimento, é de 1 segundo. Porém durante o posicionamento inicial, um período de amostragem menor proverá ao usuário uma melhor sensação de controle da movimentação do carrinho. Assim, este teste procurou descobrir qual seria o menor período que não afetasse a visualização dos componentes na tela.

Por fim, existem algumas diferenças desses testes para a aplicação real:

1. A aplicação real recebe e envia dados paralelamente, enquanto o teste realizou essas atividades de maneira sequencial.

2. A quantidade de envios de dados pela interface, é menor na aplicação real. A maior parte desses envios ocorre durante o posicionamento inicial, mas dificilmente passará de 1000, enquanto nos testes foram realizados 10000 envios.

O código de testes pode ser acessado pelo link https://github.com/darmsDD/backupCodigoTCC2.

6.3.2.2 Testes com a camada de controle real

Durante as visitas ao laboratório, foi possível realizar alguns testes de comunicação entre a camada *backend* e a camada de controle. Estes testes tiveram sucesso no envio de dados do *backend* para a camada de controle e da camada de controle para o *backend*.

6.3.3 Automatizações

Essa seção explica as 2 automatizações de código que foram criadas. A primeira sendo sobre a compilação e execução do código, e a segunda sendo sobre o início automático do programa ao ligar a *Raspberry Pi 4*.

6.3.3.1 Compilação e execução

A execução do código envolve 2 partes: executar o código da interface e executar o código da camada de controle paralelamente. Para facilitar as simulações foi criado um arquivo chamado “machine.c” que simula o comportamento da camada de controle.

Assim, criou-se um arquivo Makefile, que compila e executa a interface e o código machine.c. Este arquivo também remove a pasta de build e arquivos antigos do socket, para a nova execução do programa. Porém, era necesssário uma execução em paralelo dos programas, e portanto, foi criado um arquivo de *script*. Com a geração desses arquivos, o programa pode ser executado com apenas 1 comando “./compile.sh”.

Estas configurações podem funcionar com o código da camada de controle real, basta alterar a parte do Makefile que compila o código "machine.c", para compilar o código da camada de controle.

6.3.3.2 Início automático

Esta função foi desenvolvida para o caso de uma queda de energia causar o desligamento da *Raspberry Pi 4*. Como este é um *software* embarcado, o usuário deve ter acesso, apenas, ao programa da interface. Assim, para reduzir a possibilidade de um futuro problema, implementou-se o início automático do programa da interface e do teclado virtual.

A implementação atual funciona, porém ela realiza a compilação completa do código da interface todas as vezes que a *Raspberry Pi* é ligada. Como este processo é demorado, uma possível melhoria futura seria verificar a existência prévia de uma compilação ou *build* e caso exista, apagar apenas os arquivos de socket e executar novamente o código.

Mais detalhes sobre como foi realizada essa automatização podem ser encontrados lendo o README.md, acessível pelo link https://github.com/darmsDD/backupCodigoTCC2.

6.4 Banco de dados

Com o banco de dados construído, pode-se realizar uma análise mais precisa sobre o seu gasto de espaço de armazenamento. Foram realizados 5 experimentos, passando de fase em momentos diferentes e tendo diferentes durações. A Tabela 14 apresenta esses resultados cumulativos (banco de dados não foi apagado entre cada experimento).

Tabela 14 – Espaço de armazenamento ocupado pelo banco de dados

Experimento	**Amostra**	**Variáveis/ finais**	**Adensamento**	**Cisalhamento**	**Armazenamento**
0	0	0	0	0	24.576 bytes
1	1	1	7084	2322	294.912 bytes
2	2	2	13154	5248	557.056 bytes
3	3	3	18057	8626	794.624 bytes
4	4	4	21076	10614	942.080 bytes
5	5	5	24130	11557	1.064.960 bytes

Os seguintes parâmetros serão utilizados para simular o gasto de espaço de armazenamento real da *Raspberry Pi 4*:

- Período de amostragem de 1 segundo (inicialmente apenas adensamento e posteriormente cisalhamento também)

- Cisalhamento inicia-se na metade da duração do experimento. Então se o experimento durou 24 horas, a fase de cisalhamento iniciou-se às 12 horas.
- Experimento dura em média 2 dias.

Tabela 15 – Variáveis utilizadas para calcular o espaço de armazenamento ocupado pelo banco de dados

Variável	**Valor**
periodo_amostragem	1 segundo
quantidade_dias	2

Calcula-se a duração em segundos de 1 experimento (Eq. 6.1), para posteriomente calcular a quantidade de amostras de adensamento (Eq.6.2) e de cisalhamento (Eq. 6.3).

$$duracao\_segundos = quantidade\_dias \times (60 \times 60 \times 24) \tag{6.1}$$

$$duracao\_segundos = 2 \times 86400 = 172800 \ segundos$$

$$amostras\_adensamento = periodo\_amostragem \times duracao\_segundos \tag{6.2}$$

$$amostras\_adensamento = 1 \times 172800 = 172800 \ amostras \ de \ adensamento$$

$$quantidade\_amostras\_cisalhamento = periodo\_amostragem \times \frac{duracao\_segundos}{2} \tag{6.3}$$

$$quantidade\_amostras\_cisalhamento = 1 \times 86400 = 86400 \ amostras \ de \ cisalhamento$$

Para realizar uma análise mais próxima da realidade, serão utilizados os dados da quinta linha da Tab. 14. Essa escolha foi feita, pois nesta ocasião a quantidade de densificações e de cisalhamentos se aproximou da proporção 1/2.

A Equação 6.4 calcula a quantidade de bytes utilizados em 1 experimento. Foi necessário retirar o armazenamento inicial, pois este é o gasto inicial para se criar o banco de dados com as tabelas e o arquivo.

$$Armazenamento[4] = arm$$

$$Armazenamento[0] = arm\_inicial$$

$$bytes\_por\_experimento = \frac{(arm - arm\_inicial) \times amostras\_adensamento}{Adensamento[4]} \tag{6.4}$$

$$bytes\_por\_experimento = \frac{(942080 - 24576) \times 172800}{21076} \approx 7522523\ bytes\ por\ experimento$$

A quantidade de experimentos que o espaço de armazenamento suporta é apresentada na Eq. 6.5. Verifica-se quantos experimentos cabem em 20GB (suposto espaço livre no cartão de memória da *Raspberry Pi 4*) e adiciona-se o gasto inicial que foi removido anteriormente.

$$quantidade\_experimento = \frac{20 \times 2^{30}}{bytes\_por\_experimento + Armazenamento[0]} \tag{6.5}$$

$$quantidade\_experimento = \frac{21474836480}{7522523 + 24576} \approx 2845\ experimentos$$

6.5 Documentação

A documentação foi gerada utilizando a ferramenta Doxygen, mencionado na Seção 4.2.3, porém não foi possível comentar o código inteiro. Apenas as classes Button, DBManager, Densification, Field, Charts, exportData, receiveData, ThreadController e as estruturas interface_to_machine_message e machine_to_interface_message tiveram suas funções e atributos bem documentados.

Esta documentação pode ser acessada pelo link https://github.com/darmsDD/backupCodigoTCC2 na pasta Documentação. Caso deseje gerar novamente a documentação, navegue até o arquivo Doxyfile e execute no terminal o comando “doxygen Doxyfile”.

6.6 Dificuldades

Houveram algumas dificuldades durante o desenvolvimento do código. As 2 maiores sendo:

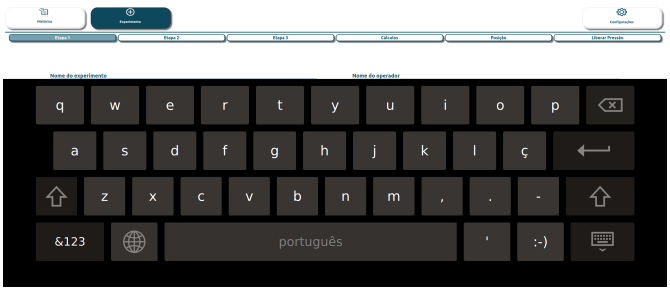
1. Foi gasto um tempo maior do que o planejado para aprender a se utilizar a ferramenta QT.

2. O teclado não aparecia automaticamente ao clicar em um campo de texto. Assim, foram gastas, aproximadamente, 2 semanas para resolver este problema.

A primeira dificuldade ocorreu, pois a ferramenta QT possuía mais complexidades do que se pensou inicialmente, assim foi necessário um tempo maior de estudo.

A segunda ocorreu, pois inicialmente, pensou-se que o teclado apareceria automaticamente ao clicar em um campo de texto, porém ele não aparece. Assim, gastou-se um certo tempo na adição de um teclado virtual ao software. Porém, ele estava ruim. Sempre era ocupado mais da metade da tela e os campos eram cobertos, demonstrado na Fig. 37.

Figura 37 – Teclado do QT



Fonte: Elaborado pelo autor, 2022

Tentou-se várias maneiras para estilizar este teclado, mas nenhuma teve sucesso. Desta maneira, decidiu-se instalar um teclado virtual para a *Raspberry Pi 4* e três *softwares* diferentes foram testados: Matchbox, Florence e por fim Onboard. Foram utilizados 3 critérios: teclado embaixo da tela, teclado não cobre os campos e teclado aparece automaticamente ao clicar em um campo de texto.

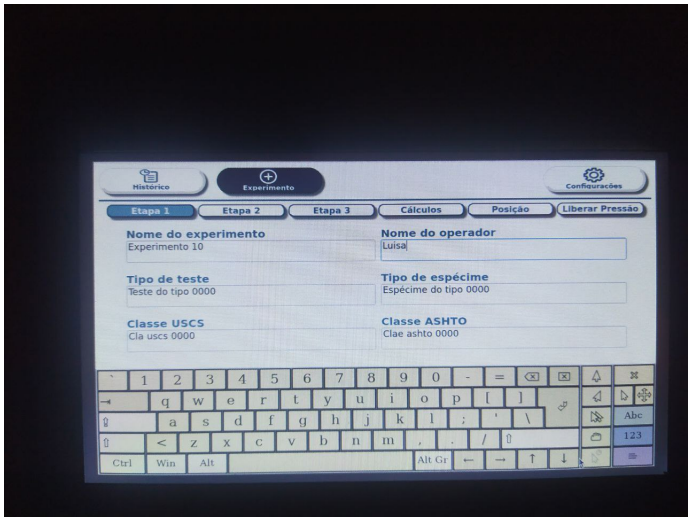
O *software* Matchbox teve alguns problemas: teclado sempre ficava na parte superior e não detectava automaticamente quando um campo de texto era clicado.

Não foi possível instalar o *software* Florence. Ele era compatível com versões anteriores da *Rasberry Pi*, mas deixou de ser com a atual.

O *software* Onboard cumpriu os 3 critérios, o teclado sobe automaticamente ao clicar em um campo de texto (depois de configurado corretamente), evita cobrir os campos da aplicação e fica em baixo da tela na maior parte do tempo. Porém, ele não consegue aparecer na frente de aplicações QT *fullscreen*.

No final, foi escolhido o software *Onboard* por melhor cumprir os 3 critérios. A Figura 38 mostra ele dentro da aplicação.

Figura 38 – Teclado do Onboard



Fonte: Elaborado pelo autor, 2022

O problema resultante desta escolha, foi que a aplicação não consegue ser *fullscreen*. Ela consegue ficar em um modo que cobre a maior parte da tela, mas ainda é possível clicar na barra de navegação superior da *Raspberry Pi*. Este problema foi mitigado, escondendo a barra de navegação, porém ainda é necessário encontrar uma melhor solução para este problema.

6.7 Projeto

Acredita-se que as mudanças propostas trarão resultados positivos para o projeto por causa dos seguintes aspectos:

- Segurança: salvar os dados em um banco de dados, facilita o seu acesso posteriormente e permite (no caso do SQLite) que seja feita uma cópia dos dados apenas duplicando o arquivo do banco de dados. Essa cópia pode então ser salva em outro dispositivo. Como a implementação do banco de dados remoto ficou fora do escopo do TCC, sugere-se que o usuário de vez em quando salve o arquivo do banco de dados em outro dispositivo.

- Acessibilidade: apesar da implementação do acesso remoto não estar no escopo do desenvolvimento do TCC, acredita-se que essa implementação futura reduzirá a necessidade de acompanhar o experimento todo presencialmente.

- Usabilidade: com apenas 1 equipamento (Fig. 28) em vez de 2 (Fig. 3), a usabilidade do equipamento se torna mais fácil pois tudo se realizará pela tela de toque sensível. Anteriormente iniciava-se o experimento na máquina de ensaio de cisalhamento direto e separadamente iniciava-se o programa no computador para visualizar os

dados. Também foi realizado um protótipo que visou tornar a usabilidade do projeto a mais simples possível depois de validação com usuários.

7 Conclusão

No início deste trabalho, Seção 1.3, foi feita a seguinte pergunta: “Como realizar o desenvolvimento de um software para um tela sensível ao toque em conjunto com uma *Raspberry Pi* que atenda a maioria dos requisitos necessários para o bom uso da máquina de ensaio de cisalhamento direto? ” e estabelecido alguns objetivos na Seção 1.4.

Durante este trabalho foram apresentados os fundamentos teóricos (Capítulo 2) que levaram as decisões futuras, as metodologias aplicadas (Capítulo 3), as ferramentas que auxiliaram o desenvolvimento (Capítulo 4), as decisões tomadas (Capítulo 5), e por fim os resultados dessas decisões (Capítulo 6).

Foi possível alcançar todos os objetivos estabelecidos e este documento apresentou argumentos suficientes para que a pergunta seja respondida de maneira positiva, ou seja, é possível realizar o desenvolvimento de um software para um tela sensível ao toque em conjunto com uma Raspberry Pi que atenda a maioria dos requisitos necessários para o bom uso da máquina de ensaio de cisalhamento direto.

As propostas estabelecidas guiaram o desenvolvimento do código, reduzindo a quantidade de problemas futuros. Porém, apesar da importância de um bom planejamento ainda é necessário aprender a lidar com imprevistos. Os dois maiores imprevistos (Seção 6.6) que postegaram a conclusão do trabalho de acordo com o cronograma inicial foram:

1. Foi gasto um tempo maior do que o planejado para aprender a se utilizar a ferramenta QT.
2. O teclado não aparecia automaticamente ao clicar em um campo de texto. Assim, foram gastas, aproximadamente, 2 semanas para resolver este problema.

O trabalho atual têm algumas limitações, sendo elas: a falta de testes de interface, a falta de testes de estresse, documentação (gerada pelo Doxygen) incompleta, Classe de Experimentos muito grande (deve ser quebrada em outras 2 ou 3 classes) e poucos testes de comunicação com a camada de controle real (Seção 6.3.2.2). Trabalhos futuros devem resolver estas limitações e realizar a implementação das funcionalidades de acesso remoto que não foram escopo deste trabalho. Os testes com a camada de controle real e os testes de estresse devem ser, especialmente, a prioridade, pois os experimentos são de longa duração (2 dias por exemplo), e assim erros inesperados podem ocorrer.

Por fim, espera-se que este trabalho tenha sido claro em sua proposta e que trabalhos futuros o utilizem como base.

Referências

AL-MAEENI, S. S. H. et al. Smart retrofitting of machine tools in the context of industry 4.0. *Procedia CIRP*, [S.l.], n. 88, p. 369–374, jun. 2020. Disponível em: <https://www.researchgate.net/publication/334036306_Smart_Retrofitting_of_Machine_Tools_in_the_Context_of_Industry_40>. Acesso em: 15 mar. 2022. Citado 2 vezes nas páginas 29 e 42.

ALMEIDA, R. D. *Programação de sistemas embarcados*: desenvolvendo software para microcontroladores em linguagem c. São Paulo: Grupo GEN, 2016. E-book (354 p.). Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595156371/>. Acesso em: 15 mar. 2022. Citado na página 39.

ATLASSIAN. *O que é Git*. Atlassian, 2022. Disponível em: <https://www.atlassian.com/br/git/tutorials/what-is-git>. Acesso em: 26 mar. 2022. Citado na página 57.

CHARLIER, C. R. *User experience design for the Internet of Things*. 1. ed. Sebastopol, CA: O'Reilly Media, 2015. 44 p. Citado 2 vezes nas páginas 46 e 47.

CLARK, J. *Designing for touch*. Nova Iorque: A Book Apart, 2015. 171 p. ISBN 978-1-9375572-9-4. Citado na página 73.

DOXYGEN. *Doxygen*: Generate documentation from source code. © 2022. Disponível em: <https://doxygen.nl/>. Acesso em: 22 set. 2022. Citado na página 64.

FARRINGTON, T. *UX design 2020*: the ultimate begginer's guide to user experience. [S.l.: Theo Farrington], 2020. E-book (155 p.) Edição Kindle. Citado 2 vezes nas páginas 45 e 46.

GEOTEST INSTRUMENT CORP. *DESCRIPTION AND INSTRUCTIONS FOR GEOTEST DIRECT-RESIDUAL SHEAR MACHINE MODELS S2213A, S2215A & S2216*. Evanston, 2002. Citado 2 vezes nas páginas 37 e 38.

GITHUB. *What is GitHub?* [San Francisco: Github], 2016. 1 vídeo (4 min.). Disponível em: <https://www.youtube.com/watch?v=w3jLJU7DT5E>. Acesso em: 26 mar. 2022. Citado na página 57.

GITHUB. *About branches*. GitHub, © 2022. GitHub Docs. Disponível em: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches>. Acesso em: 26 mar. 2022. Citado na página 57.

GITHUB. *About issues*. GitHub, © 2022. GitHub Docs. Disponível em: <https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues>. Acesso em: 26 mar. 2022. Citado na página 57.

GITHUB. *About pull requests*. GitHub, © 2022. GitHub Docs. Disponível em: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>. Acesso em: 26 mar. 2022. Citado na página 57.

HALFACREE, G. *The offical Raspberry Pi beginner's guide*: how to use your new computer. 3. ed. Cambrigde, UK: Raspberry Pi Press, 2019. 248 p. ISBN 978-1-912047-58-1. Citado na página 43.

HU, R. *Embedded systems architecture and software design*: software design principles, considerations, design concepts, and building blocks. English edition. [Seattle, WA]: Kindle Direct Publishing, 2022. E-book (354 p.) Edição Kindle. Citado 5 vezes nas páginas 38, 39, 40, 41 e 42.

HUMPHREY, W. S. *The personal software process sm (PSP SM)*. Pittsburgh,PA: Carnegie-Mellon University, ©2000. x, 41 p. Relatório CMU/SEI-2000-TR-022 ESC-TR-2000-022. Disponível em: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283>. Acesso em: 27 fev. 2022. Citado na página 54.

Lima II, E. J. et al. *Sensoring for retrofitting of an industrial robot*: information control problems in manufacturing. Oxford: Elsevier, 2005. v. 1. 545–550 p. ISBN 0-08-044249-8. Citado na página 42.

LöBACH, B. *Design industrial*: bases para a configuração dos produtos industriais. 1. ed. São Paulo: Edgard Blücher, 2001. 208 p. ISBN 85-212-0288-1. Citado na página 45.

MARINHO, F. *Estabilidade de taludes*: deslizamentos de terra. Guia de engenharia, 2020. Disponível em: <https://www.guiadaengenharia.com/estabilidade-taludes-deslizamentos/>. Acesso em: 02 abr. 2022. Citado na página 35.

MELãO, A. *Kanban muito além de um quadro com adesivos*: uma visão de agilidade e eficácia para iniciantes. [S.l.: Antonio Melão], 2021. E-book (94 p.) Edição Kindle. Citado na página 54.

MOLLOY, D. *Exploring Raspberry Pi*: interfacing to the real world with embedded linux. Indianapolis: John Wiley & Sons, 2016. 160–162 p. ISBN 978-1-119-18868-1. Citado na página 59.

NORMAN, D. *The design of everyday things*. Revised and expanded edition. Nova Iorque: Basic Books, 2013. 369 p. ISBN 978-0-465-00394-5. Citado 2 vezes nas páginas 45 e 46.

PAGOTTO, T. et al. *Scrum solo*: software process for individual development. In: *IBERIAN CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGIES (CISTI), 11. 2016, Gran Canaria*. [Piscataway, NJ]: IEEE, 2016. p. 1–6. Citado 4 vezes nas páginas 33, 53, 54 e 56.

PEREIRA, M. F. M. *Sistema embarcado para aquisição de dados e controle em ensaios de cisalhamento direto*. 135 p. Monografia (Bacharelado em engenharia eletrônica) — Universidade de Brasília, Brasília, setembro 2022. Citado 5 vezes nas páginas 30, 31, 75, 79 e 87.

PINTO, C. de S. *Curso básico de mecânica dos solos*. 3. ed. São Paulo: Oficina de Textos, 2006. 367 p. ISBN 978-85-86238-51-2. Citado 4 vezes nas páginas 29, 35, 36 e 37.

POMEROY-HUFF, M. et al. The Personal Software Process SM (PSP SM) body of knowledge, version 2.0. Pittsburgh, PA: Carnagie Mellon University, ©2009. Ebook (xii, 78 p.). Relatório especial CMU/SEI-2009-SR-018. Disponível em: <https://www.researchgate.net/publication/235116544_The_Personal_Software_ProcessSM_PSPSM_Body_of_Knowledge_Version_20>. Acesso em: 27 fev. 2022. Citado na página 54.

POMPONIO, A. *MySQL vs SQLite*: features, use cases, and performance comparison. Open Logic, 2021. Disponível em: <https://www.openlogic.com/blog/mysql-vs-sqlite>. Acesso em: 23 abr. 2022. Citado na página 63.

QT. *About Qt*. Qt, 2022. Revision as of 10:31, 23 March 2022 by StefanSequality. Disponível em: <https://wiki.qt.io/index.php?title=About_Qt&oldid=39143>. Acesso em: 23 abr. 2022. Citado na página 60.

RASPBERRY PI FOUNDATION. *Raspberry Pi 4 tech specs*. Raspberry Pi, 2022. Disponível em: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>. Acesso em: 22 mar. 2022. Citado 2 vezes nas páginas 43 e 45.

RASPBERRY PI FOUNDATION. *Raspberry Pi integrator programme*. Raspberry Pi, 2022. Disponível em: <https://www.raspberrypi.com/for-industry/integrator-programme/>. Acesso em: 22 mar. 2022. Citado na página 43.

SCHWABER, K.; SUTHERLAND, J. The scrum guide: the definitive guide to scrum: the rules of the game. [S.l.]: Ken Schwaber and Jeff Sutherland, ©2020. 13 p. Disponível em: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>. Acesso em: 25 fev. 2022. Citado na página 53.

SHIELDS, W. *SQL quickstart guide*: the simplified beginner's guide to managing, analyzing, and manipulating data with sql. 1 english. ed. Nova Iorque: ClydeBank Media LLC, 2019. E-book (348 p.). Citado na página 62.

SMITH, H. *Scrum*: the ultimate beginner's guide to learn and master scrum agile framework. English edition. [S.l.]: CreateSpace Independent Publishing Platform, 2018. E-book (48 p.) Edição Kindle. ISBN 1721770178. Citado na página 53.

SOBHAN, B. M. D. *Fundamentos de engenharia geotécnica*. São Paulo: Cengage Learning, 2014. 632 p. Tradução da 8ª edição norte-americana. ISBN 978-85-221-1824-3. Citado 3 vezes nas páginas 29, 36 e 37.

SOMMERVILLE, I.; SAWYER, P. *Requirements engineering*: a good practice guide. 1. ed. England: John Wiley & Sons Ltd, 1997. 404 p. ISBN 0471974447. Citado na página 47.

SQLITE. *About SQLite*. SQLite, 2022. Disponível em: <https://www.sqlite.org/about.html>. Acesso em: 23 abr. 2022. Citado na página 64.

WHITE, E. *Making embedded systems*. California: O'Reilly, 2012. E-book (506 p.) Edição Kindle. ISBN 978-1-449-30214-6. Citado 6 vezes nas páginas 30, 38, 39, 40, 41 e 42.

WIEGERS, K. E.; BEATTY, J. *Software requirements*. 3rd. ed. England: Microsoft Press, 2013. 672 p. ISBN 0735679665. Citado 4 vezes nas páginas 47, 48, 51 e 52.

ZENHUB. *Productivity management for software teams*: connect code to strategy. ZenHub, 2022. Disponível em: <https://www.zenhub.com/>. Acesso em: 26 mar. 2022. Citado na página 58.