

UNIVERSIDADE PAULISTA – UNIP EaD

Projeto Integrado Multidisciplinar

**Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas**

Nicolas dos Santos Lira – 2251695

SISTEMA DE RESERVA

Colégio Vencer Sempre

Ferraz de Vasconcelos, São Paulo.

2023

Nicolas dos Santos Lira – 2251695

SISTEMA DE RESERVA

Colégio Vencer Sempre

**Projeto Integrado Multidisciplinar em
Análise e Desenvolvimento de Sistemas**

**Projeto Integrado Multidisciplinar para obtenção do título de tecnólogo em Análise e
Desenvolvimento de Sistemas, apresentado à Universidade Paulista – UNIP EaD.**

Orientador (a): Profa. Ma. Gislaine Stachissini

Ferraz de Vasconcelos, São Paulo.

2023

RESUMO

Neste projeto é abordado os conceitos básicos de economia e mercado por meio da identificação de quem serão os agentes econômicos fictícios que atuaram no contexto do projeto, e como funcionará as propriedades que atribuíram valor para o desenvolvimento do *software*, o prazo para realização do mesmo e o valor que a empresa solicitará ao colégio.

Quanto ao conceito de engenharia de software, foi elaborada a coleta de requisitos funcionais, não funcionais e de negócio. A divisão das etapas do desenvolvimento utilizando de um ciclo de vida, e estudo de uma norma de qualidade que integrará todas as etapas do projeto. Já para programação orientada a objetos, foi realizado o desenvolvimento de um diagrama de classe seguindo o modelo UML, bem como a codificação de um escopo das respectivas classes do sistema.

Por fim, ao que diz respeito a Interface com usuário, foi elaborado e desenvolvido um layout para as páginas que integrariam o sistema, utilizando de conceitos de metáforas de interação, usabilidade e elegância.

Palavras chave: Modelo de Qualidade, Figma, Programação Orientada a Objetos, Design, Metáforas de interação, Requisitos, Testes, C#.

ABSTRACT

In this project, the basic concepts of economy and market are approached through the identification of who will be the fictitious economic agents that acted in the context of the project, and how the properties that attributed value to the development of the software will work, the deadline for its realization and the amount that the company will request from the college.

As for the concept of software engineering, the collection of functional, non-functional and business requirements was elaborated. The division of development stages using a life cycle, and the study of a quality standard that will integrate all stages of the project. As for object-oriented programming, the development of a class diagram following the UML model was carried out, as well as the coding of a scope of the respective classes of the system.

Finally, with regard to the User Interface, a layout was designed and developed for the pages that would integrate the system, using concepts of interaction metaphors, usability and elegance.

Keywords: Quality Model, Figma, Object Oriented Programming, Design, Interaction Metaphors, Requirements, Tests, C#.

SUMÁRIO

Conteúdo

RESUMO	2
ABSTRACT	3
SUMÁRIO.....	4
1 INTRODUÇÃO.....	6
2 CICLO DE VIDA – CASCATA (<i>waterfall</i>).....	7
3 VISÃO ECONÔMICA.....	8
4 ENGENHARIA DE SISTEMAS	9
4.1	Contextualização.
9	
4.2	Requisitos
9	
4.2.1 Requisitos Funcionais.....	9
4.2.2 Requisitos não-funcionais	10
4.2.3 Requisitos de negócio.....	10
4.3 Normas de Qualidade	10
4.3.1 MPS.BR.....	11
5 ANÁLISE	13
5.1 Resumo sobre Orientação a Objetos	14
5.2 Diagramas UML.....	17
6 DESIGN	18
6.1 Tela inicial.....	18
6.2 Barra de navegação	19
6.3 <i>Dashboard</i> Inicial.....	19
6.5 Agendamentos	21
6.6 Tela de Perfil	22

7 CODIFICAÇÃO.....	24
8 TESTES.....	26
CONCLUSÃO.....	27
REFERÊNCIAS	28
APÊNDICE A - Lista de tarefas para testes de sistema.	29
APÊNDICE B - FORMULÁRIO PARA TESTE DE SISTEMA.....	30
APÊNDICE B - FORMULÁRIO PARA TESTE DE ACEITAÇÃO DO PROTÓTIPO	31

1 INTRODUÇÃO

A busca pela qualidade na área de desenvolvimento de software é um desafio constante para as organizações que buscam se manter competitivas no mercado. Nesse contexto, a adoção de normas de qualidade se torna uma importante estratégia para garantir a qualidade do produto final e a satisfação do cliente. Entre as normas mais utilizadas no mercado brasileiro, destaca-se a MPS.BR, que tem como objetivo principal melhorar a qualidade e a produtividade do processo de software.

Além disso, para atender aos requisitos de negócio e funcionais, é necessário que o processo de desenvolvimento de software esteja alinhado com as necessidades dos clientes e das partes interessadas, garantindo que o software desenvolvido atenda às expectativas e necessidades do usuário final.

Outro aspecto importante a ser considerado é a utilização de orientação a objetos, uma abordagem que permite uma melhor modelagem do sistema, facilitando a manutenção e a evolução do software. E, por fim, a interação humano-computador (IHC) é um fator crucial para garantir a usabilidade e a experiência do usuário final com o software, o que impacta diretamente na satisfação e aceitação do produto.

Neste trabalho, serão abordados aspectos relacionados à norma de qualidade MPS.BR, aos requisitos funcionais e de negócio, à orientação a objetos e à interação humano-computador, buscando compreender a importância de cada um desses aspectos para a garantia da qualidade do software e a satisfação do usuário final.

2 CICLO DE VIDA – CASCATA (*waterfall*)

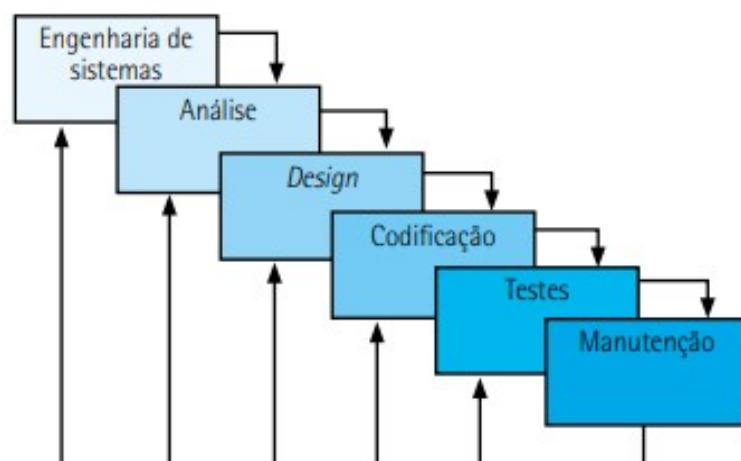
Antes de começar o desenvolver do projeto, optei por iniciar escolhendo um modelo de ciclo de vida para o desenvolvimento do sistema, com a finalidade de auxiliar nas etapas de produção. A escolha do ciclo de vida pode trazer um aumento significativo na qualidade e otimização no tempo de produção, afinal, ele auxilia o desenvolvedor ou equipe de desenvolvimento definindo os estágios do projeto e como ele será entregue.

Existem diversos modelos de ciclo de vida disponíveis no mercado, todavia para este será utilizado o modelo clássico, ou também conhecido como modelo cascata. O modelo Cascata é um dos mais antigos *life cycle model*, ele surgiu em meados de 1970 e foi por muito tempo o único modelo de aceitação geral. Este é um derivado de atividades de engenharia, que tem como finalidade estabelecer ordem no modelo de desenvolvimento, sua versão original foi aprimorada ao longo do tempo e por conta disso continua sendo muito utilizado atualmente.

Este modelo foi escolhido pela sua simplicidade de implementação e grande uso no mercado, bem como pelo fato do mesmo ser orientado a documentação, sendo essa orientação não apenas limitada a arquivos de texto. O processo de desenvolvimento com o modelo cascata é dividido em níveis, a qual tem seu início assim que o antecessor é finalizado.

Abaixo é apresentada uma imagem que ilustra as etapas do ciclo de vida.

Figura 1 – Modelo Cascata.



FONTE: Livro Texto, Engenharia de Software.

3 VISÃO ECONÔMICA

Para o desenvolvimento deste projeto, se faz a necessidade básica de mapear quem serão os principais agentes econômicos envolvidos e alguns conceitos básicos de economia. O colégio Vencer Sempre é a parte solicitante do serviço, ou seja, aquele que pagará pelo serviço que será gerado, enquanto a empresa de *software* é o produtor, que receberá por desenvolver o produto.

A empresa tem como função definir o valor de seu produto de *software*, seguindo o custo do mercado com a finalidade de não acabar ficando com prejuízos pela produção, nem superfaturando o bem fazendo com que o cliente busque outras empresas de valor mais viável. O custo que a empresa apresentará ao cliente será levemente acima da média presente no mercado, pois a mesma integra em sua produção uma norma de qualidade, o que traz um acréscimo de valor à instituição. Os envolvidos financeiros da empresa serão os desenvolvedores, avaliadores, e gerente de projetos.

Abaixo é apresentada uma tabela com algumas informações sobre os custos e prazos do desenvolvimento, além de outras informações relevantes.

Figura 2 – Propriedades e custos do desenvolvimento.

Prazo previsto para conclusão:	Dois meses.
Aplicação de norma de qualidade:	Sim.
Desenvolvimento de protótipos:	Sim.
Documentação:	Sim.
Desenvolvimento de Diagramas:	Sim.
Base de dados:	Sim.
Avaliação de protótipos:	Sim.
Roteiro de testes:	Sim.
Custo:	R\$ 1.564,90

FONTE – Arquivo pessoal.

4 ENGENHARIA DE SISTEMAS

Faz-se necessário para realização deste projeto, definir quais serão seus requisitos funcionais, não funcionais e de negócio do sistema, e visando isso, será necessária uma leitura analítica da contextualização do projeto, e após isto será elaborada uma lista para cada requisito.

4.1 Contextualização.

O caso é simples, o colégio Vencer Sempre disponibiliza equipamentos de informática e vídeo como ferramentas de apoio aos docentes para suas aulas e palestras, porém o funcionário da instituição de ensino só pode usufruir do mesmo caso realize uma solicitação antecipado aos colaboradores, que alocam o material na sala de aula apontada ou auditório.

Atualmente a reserva é realizada de forma manual, e muitos dos docentes não conseguem utilizar do benefício pela ineficiência do agendamento. A solicitação da instituição de ensino é para que nossa empresa desenvolva um *software*, com a finalidade de permitir com que os docentes realizem o agendamento de uma maneira prática e eficiente.

4.2 Requisitos

4.2.1 Requisitos Funcionais

Requisitos funcionais são aqueles que descrevem funções que o software irá realizar, e podemos definir mais objetivamente como um conjunto de entradas, processamentos e saídas de dados. Para este sistema, teremos como requisitos funcionais:

- Agendamento de equipamento.
- Login.
- Visualização de equipamentos.
- Visualização de equipamentos disponíveis.
- Cancelar agendamento.
- Gerenciamento de equipamentos.

4.2.2 Requisitos não-funcionais

Este é um requisito que se refere à utilização da aplicação ao que diz respeito a desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenção e as tecnologias envolvidas. Os requisitos não-funcionais para esse sistema são:

- Limitar acesso aos docentes e colaboradores.
- Alocar na sala dos professores, sala do coordenador e do diretor.
- Sem necessidade de treinamento.
- O sistema deverá estar disponível a todo o momento aos docentes.
- O sistema deverá rodar em sistemas operacionais Windows.
- O sistema emitirá um código de confirmação de agendamento.
- Programa será desenvolvido em C#.
- O programa será orientado a objetos.
- O sistema se comunicará com o MYSQL.
- O sistema só exibirá o nome do docente, dia, horário e a sala que o equipamento deve ser implantado para os colaboradores.

4.2.3 Requisitos de negócio

São os valores que se espera aderir com a aquisição do *software* na organização. Neste caso temos como requisitos de negócio:

- Realizar o agendamento do equipamento de forma eficiente, e dinâmica.
- Praticidade para realizar o agendamento.

4.3 Normas de Qualidade

Além da realização dos requisitos, se faz necessário escolher uma norma de qualidade para produção deste software, atualmente podemos optar por três normas de qualidade, as ISO'S (*International Organization for Standardization*), o CMMI (*Capability Maturity Model Integration*), ou o MPS.BR (Melhoria de Processamento de Software Brasileira). Como o sistema desenvolvido será utilizado por um colégio, de princípio terá será utilizado em território nacional, e apresenta uma baixa complexibilidade, a norma de qualidade selecionada para implementação deste é a MPS.br.

4.3.1 MPS.BR

A MPS.BR foi criado em dezembro de 2003 pela SOFTEX (Associação para Promoção de Excelência do Software Brasileiro), com o apoio do MCTIC (Ministério da Ciência, Tecnologia, Inovação e Comunicação), e financiada pelo FINEP (Financiadora de Estudos e Projetos), do SEBRAE (Serviço Brasileiro de Apoio às Micro e Pequenas Empresas, e o BID (Banco Interamericano de Desenvolvimento).

O objetivo deste é permitir que empresas de Micro, Pequena e Médio porte consigam inserir padrões de qualidade aceitos internacionalmente, e utilizarem de boas práticas e melhoria de processo já disponíveis, de uma forma mais prática e com um custo consideravelmente menor se comparado com a aquisição das outras normas citadas anteriormente. Ele é alinhado às normas internacionais ISO 12207, ISO 15504, ISO 25000, e CMMI, porém o reconhecimento como selo de qualidade da norma MPS.BR está limitada apenas ao território brasileiro.

A norma brasileira de qualidade está dividida em quatro componentes, sete níveis de maturidade e dezenove processos que estão dispostos nos níveis definidos. Os componentes são modelos de referência para desenvolvimento, aquisição, e avaliação, já os níveis de maturidade são a classificação a qual as organizações recebem de acordo com a avaliação, e por fim, os processos são atividades que a organização pratica para atingir os níveis de maturidade. Os respectivos componentes do MPS.BR são:

Guia Geral MPS de Software: Possui a descrição da estrutura dos modelos e detalha o Modelo MPS para Software, bem como os seus componentes, e as definições necessárias para sua compreensão e aplicação.

Guia Geral MPS de Serviços: Este possui a descrição da estrutura dos modelos MPS e detalha o modelo para serviços, seus componentes e as definições para seu entendimento e aplicação.

Guia Geral MPS para gestão de pessoas: Retém a descrição da estrutura dos modelos do MPS, detalha o modelo de referência para gestão de pessoas, bem como seus componentes, e as definições para sua compreensão e aplicação.

Guia de Avaliação: Por fim, este descreve o processo e o método de avaliação, os requisitos para os avaliadores líderes, avaliadores adjuntos e as instituições Avaliadoras.

O Componente de maior interesse para este trabalho é o que diz respeito ao *software* ou desenvolvimento, este modelo de referência do MPS para software tem como base técnica a NBR ISO/IEC/IEEE 12207, que tem como objetivo estabelecer uma estrutura para os processos de ciclo de vida de um software, como uma terminologia bem definida que pode ser referência para indústria de software, pois a estrutura dispõe de processos, tarefas, atividades, propósitos e resultados que deverão ser aplicados. Esta norma foi construída para ser utilizada em todo ciclo de vida, ou seja, desde o início até a descontinuidade do sistema.

Ele também possui em sua estrutura a série de normas ISO/IEC 330xx, essa série substitui e amplia o uso da ISO/IEC 15504, bem como estabelece um *framework* que tem como finalidade melhorar a qualidade dos processos, alinhando-os com os objetivos do negócio. Por fim, a versão de 2023 do modelo tem compatibilidade com o CMMI-DEV, esse tem como intuito de integrar melhores práticas que se dedicam à melhoria do desempenho de seus processos chave.

O MPS-SW define seis níveis de maturidade, que são uma mescla de processos e sua capacidade. Os processos estão divididos entre os organizacionais e os de Projeto. Os processos de projeto são utilizados para projetos de *software*, seja em desenvolvimento de um novo produto, manutenção ou evolução do mesmo. Já os processos organizacionais, são aqueles que fornecem os recursos necessários para que as expectativas e necessidades dos interessados sejam atendidas. Abaixo temos uma imagem que apresenta os processos e suas capacidades:

Figura 2 – Conjunto de processos de projeto e organizacionais.



Fonte: Guia Geral MPS de Software, Softex 2023.

Por fim, os resultados desejados destes processos estão dispostos da forma mais adequada entre os níveis de maturidade da norma, e por conta disto, nem todos estão alocados nos primeiros níveis de maturidade, e os mesmos vão evoluindo de acordo com a evolução.

Os processos são cumulativos, ou seja, à medida que a organização avança no nível de maturidade, os resultados presentes nos níveis anteriores deverão estar presentes no nível atual. Abaixo é apresentada uma figura que representa os níveis de maturidade, que se iniciam no G e finalizam no A

Figura 3 – Níveis de maturidade.



Fonte: Guia Geral MPS de Software, Softex 2023

5 ANÁLISE

Nesta fase, se faz necessário observar de uma forma mais analítica e lógica os requisitos que foram levantados acima, com a finalidade de elaborar a forma com que o sistema funcionará, ou seja, definir suas regras de negócio. A forma como os requisitos funcionais estão dispostos em lista foi a qual foram idealizados, e para iniciarmos a montar a lógica do programa, é necessário organizar a lista de requisitos com base na utilização do *software*. Abaixo está a lista ordenada por funcionalidades em seguimento lógico.

- ☐ Login.
- ☐ Visualização de equipamentos.
- ☐ Visualização de equipamentos disponíveis.
- ☐ Agendamento de equipamento.
- ☐ Cancelar agendamento.
- ☐ Gerenciamento de equipamentos.

O sistema inicia solicitando o login ao usuário, este será composto pela entrada de duas cadeias de caracteres, a primeira é um conjunto que será gerado assim que o mesmo for implementado no sistema, e será denominado como Código de Entrada, já a segunda entrada também será uma cadeia de caracteres denominada senha.

Assim que acessar o sistema, o usuário acessa seu painel de controle, onde terá uma função para pesquisa de equipamentos, um local a qual poderá verificar os agendamentos realizados, e acessar seu perfil no sistema, bem como terá os itens do banco de dados dispostos em ordem alfabética.

Ao realizar a pesquisa, o usuário terá em seu *dashboard* uma lista filtrada com base nos caracteres que serviram de entrada. Já o ambiente de verificação de agenda, se constitui de uma lista de datas em que se um equipamento estiver agendado para certo dia, estará demarcado.

No ambiente de perfil, o usuário poderá visualizar ou alterar algumas das informações ali presentes. Por fim, caso seja selecionado algum equipamento, após a pesquisa ou apenas selecionando pela lista em ordem alfabética, o usuário será levado a uma tela que mostrará uma agenda mostrando a disponibilidade do equipamento.

Para realizar o agendamento, o docente necessita apenas entrar com o local a qual o equipamento será utilizado dentro da unidade escolar, e para o cancelamento de um agendamento será necessária uma confirmação.

Após desenvolver o modelo mental lógico de como o software funciona, é uma boa prática criar diagramas para colocar de uma forma ainda mais documental como o *software* funcionará, e quais serão os tipos de entrada, saída e processos que o mesmo realizará com a finalidade de ter êxito em suas funcionalidades.

Este projeto será estruturado em Orientação a Objetos, que é uma das vantagens de utilizar da linguagem de programação C#, e por conta dessa estrutura, os diagramas serão desenvolvidos nos padrões da UML (*Unified Modeling Language*), que são utilizados para realização de notações orientadas a objeto, que são um conjunto de diagramas que auxiliam a documentar de uma forma clara o funcionamento do sistema orientado a objetos.

5.1 Resumo sobre Orientação a Objetos

Para melhor compreensão dos diagramas que serão apresentados posteriormente, bem como melhor entendimento dos escopos de código do capítulo sete, será apresentado os conceitos de Objeto, classes, heranças e polimorfismo, que são a base para metodologia de desenvolvimento orientada a objetos.

Começamos pelo conceito de Objeto, este que pode ser descrito como entidade discreta, identidade própria e escopo bem definido, a qual agrega em sua estrutura o estado e seu comportamento. O **Estado** é um conjunto de descrições de espaço com nome e tipo, ou seja, e estes levam a identidade do objeto mantendo seus valores desse tipo. Já o **Comportamento** é representado por funções, ou operações, que são o comportamento do objeto, ou seja, o que o objeto pode fazer, como por exemplo, realizar uma funcionalidade em uma determinada interação. Os objetos interagem por meio de uma troca de informações, e esta é conhecida como **mensagem**, e normalmente é realizada por meio de operações.

Já Classe pode ser descrita de uma forma simples e precisa, como uma estrutura para objetos, sendo que ela agrega em sua estrutura os atributos e operações, bem como relacionamentos e comportamento dos objetos que dela derivam. Na figura abaixo é representado o diagrama de uma classe:

Figura 4 – Classe Carro.

Carro
- Marca : string - Modelo : string - Cor : string
+ Ligar() : bool + Acelerar() : int + Parar() : int + Desligar() : bool

FONTE: AstahUML.

Esta é uma classe simples que possui apenas 3 atributos e 4 operações, e ela possibilita exemplificar o conceito de classes e objetos da maneira mais simples. A estrutura lógica da classe na figura acima, está separada em 3 sessões, a primeira nomeia a classe, nos permitindo realizar a identificação da mesma, já a segunda contém seus atributos, que caso preenchidos serão os estados do objeto, enquanto a terceira parte possui as funções da classe, ou comportamento do objeto.

Nas figuras abaixo é apresentado à construção de dois objetos com base na estrutura desta classe, e nestes é possível notar que os objetos se diferem por conta dos estados a qual foram inseridos em seus atributos.

Figura 5 – Objeto Carro.

CARRO	
MARCA : String	"Fiat"
MODELO : String	"Uno Mille Way"
COR : String	"Preto"

FONTE: Arquivo Pessoal.

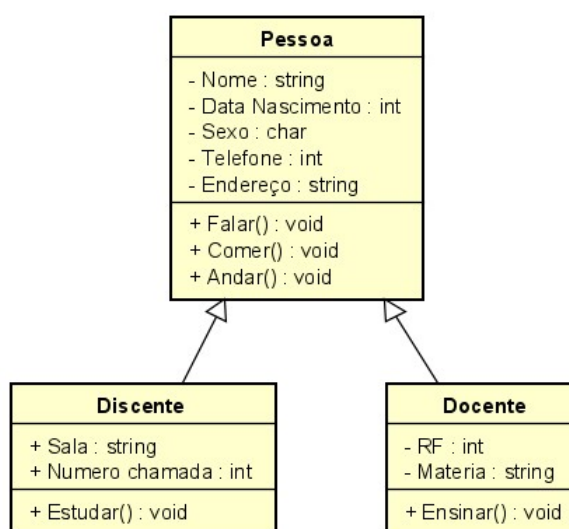
CARRO	
MARCA : String	"Chevrolet"
MODELO : String	"Celta"
COR : String	"Prata"

FONTE: Arquivo Pessoal.

Figura 6 – Objeto Carro.

O conceito de **herança** também é algo simples, todavia, ela demanda que o conceito da estrutura lógica de uma classe esteja bem desenvolvido. A herança consiste em possibilitar que uma classe adquira os atributos e comportamentos de outra classe, abaixo é apresentado um diagrama que ilustra o funcionamento de herança:

Figura 7 – Herança.



FONTE: AstahUML.

O diagrama da imagem acima apresenta três classes, a Pessoa, o Docente e o Discente, e nesta, as classes Docentes e Discentes estão herdando a superclasse Pessoa. Com esse exemplo é possível seguir a lógica simples por trás do conceito de herança, sabemos que o docente e discente possuem alguns atributos e funções divergentes, entretanto, os dois possuem a mesma base, ou seja, ambas são pessoas, e por conta disso, podemos utilizar a superclasse para definir os estados e comportamentos padrões que ambos possuem.

Por fim, **polimorfismo** é um conceito que diz que objetos diferentes apresentam comportamentos diferentes realizando a mesma ação. De início esse conceito aparenta ser muito complexo para se entender, porém quando você o compreende, por uso contínuo ou ampla pesquisa, percebe que a complexidade deste é baixa.

Um exemplo a qual me ajudou a fixar e compreender o polimorfismo deu-se no livro “Java, como programar 10ª Edição” de DEITEL, onde o mesmo apresenta o seguinte exemplo:

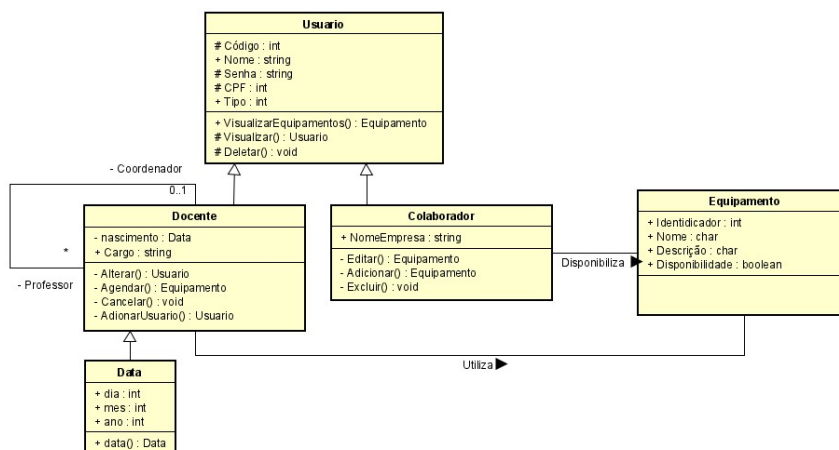
“Para realizar um estudo biológico, foi desenvolvido um sistema que simula a movimentação dos animais de classe Peixe, Anfíbio e Pássaros, e cada uma destas herda a classe Animal, que possui o método mover(). Para realizar a simulação, o software envia para o objeto o comando mover para cada um dos objetos, entretanto, cada uma das classes responder a solicitação de uma maneira única: O peixe nada um metro em linha reta, o anfíbio pula um metro e meio para frente, já o pássaro pode voar três metros. Isso é possível, pois cada objeto sabe “fazer a coisa certa”, ou seja, faz o que é apropriado para o seu tipo de objeto em resposta ao mesmo método, e esse é o conceito-chave do polimorfismo. A mesma mensagem enviada a uma variedade de objetos tem muitas formas de resultados.” [DEITEL; DEITEL, 2017, p.312]

Os conceitos presentes na orientação a objetos podem ser fortalecidos por meio da prática, tanto desenvolvendo diagramas ou então desenvolvendo sistemas aplicando o conceito, quanto realizando a leitura de livros e documentos que abordam o assunto.

5.2 Diagramas UML

Com o conceito de Orientação a objetos e compreendendo o que é UML, na figura abaixo temos o diagrama de classe e outro de caso de uso, ambos foram elaborando utilizando o *software* AstahUML e estão disponíveis no arquivo compacto a qual este se encontra, dentro da pasta Documentação.

Figura 8 - Diagrama de Classes do sistema.



FONTE: AstahUML.

O diagrama de classes do projeto contém uma superclasse chamada “**Usuário**” que tem como finalidade conter os atributos e métodos em comum das classes “**Docente**” e “**Colaborador**”, sendo eles o **Código** único de cada perfil, o **Nome** do usuário, a **Senha** para acesso ao sistema, e o **Tipo** de usuário para delimitar se ele é “**Docente**” ou “**Colaborador**”.

A classe “**Docente**” diz respeito aos professores e coordenadores da instituição de ensino, e têm como parâmetros apenas a **Data de nascimento** do educador, e o seu respectivo **cargo** na instituição de ensino. Já a classe “**Colaborador**” possui a estrutura do colaborador que disponibiliza o equipamento para instituição, tendo como único atributo o **Nome da empresa** a qual o mesmo atua. Podemos analisar que ambas as classes herdeiras possuem atributos e métodos específicos tais quais atribuem seus estados únicos frente à superclasse usuário.

Por fim, temos a classe equipamento, que contém como atributos as especificações básicas para encontrar o mesmo no sistema, e não há necessidade de se atribuir funções para a classe, pois as funções que o sistema solicita são realizadas unicamente pelos usuários e seus herdeiros. Os atributos dessa classe são o **Identificador**, o **Nome** do equipamento, a **Descrição** das características do mesmo, e sua **Disponibilidade** referente ao dia.

6 DESIGN

Para realizar o desenvolvimento do protótipo deste projeto, foi utilizada a plataforma Figma, disponível para navegadores e desktop. Escolhi este ambiente por conta da facilidade de sua utilização, gratuidade, funcionalidades, funções, melhor alinhamento de itens, sua fantástica função de interatividade e a possibilidade de utilizar extensões que auxiliam no desenvolvimento da UX/UI, que para este foi utilizada a biblioteca de ícones do Google.

6.1 Tela inicial

Assim que acessar o sistema, o usuário visualiza a interface de *login*, neste ele encontrará textos, uma imagem ilustrativa, dois campos de formulário e um botão de confirmação. Os campos de formulários apresentam ícones com metáforas de interação, sendo elas um desenho que representa o usuário, e o outro um cadeado representando a senha. Assim que o usuário realizar o login, preenchendo os campos da tela anterior com valores

válidos e enviando o mesmo, ele será direcionado para a próxima página do programa (Imagem 4 e 5).

Imagem 1 – Tela de login.



FONTE: Figma.

6.2 Barra de navegação

Este componente está presente em todas as interfaces que o usuário se depara após realizar o login, e será por meio dela que será feita a navegação pelas interfaces do programa. Esta é composta unicamente por ícones que atribuem metáforas de interatividade, e cada usuário (Docente e colaborador) possui uma barra de navegação própria por conta da diversificação dos métodos que realizam.

Imagem 2 – Navegação Docente



FONTE: Figma.

Imagem 3 – Navegação Colaborador



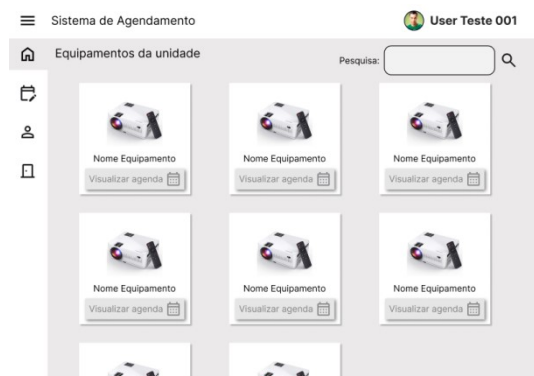
FONTE: Figma

6.3 Dashboard Inicial

Ao adentrar no sistema com o login e senha corretos, o usuário acessa de princípio a página abaixo, nomeada como *home*. Nela ele tem acesso a alguns elementos, sendo os principais a barra de navegação e os *cards* contendo os equipamentos disponíveis para

unidade escolar, esta é a interface a qual o docente realizará o agendamento de um equipamento e o colaborador terá acesso a exclusão e alteração das propriedades da máquina.

Imagem 4– Home Docente



FONTE: Figma.

Imagem 5 – Home Colaborador



FONTE: Figma.

6.4 Pop-Up agendar equipamento

Assim que o usuário clicar no botão de visualizar agenda presente no *card* da interface *home*, será apresentado na tela um *pop-up* contendo o nome do equipamento, sua descrição e um calendário contendo sua disponibilidade por data.

Imagem 6 – Card equipamento.



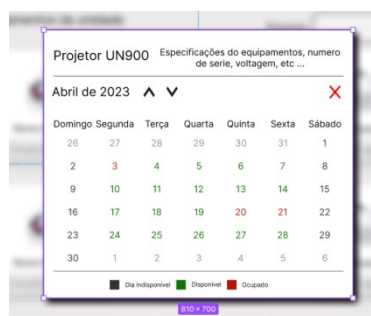
FONTE: Figma.

Imagem 7 – Botão agendar.



FONTE: Figma

Imagem 8 – Agenda equipamento

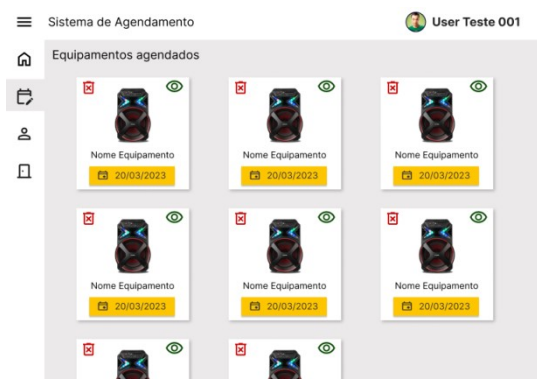


FONTE: Figma.

6.5 Agendamentos

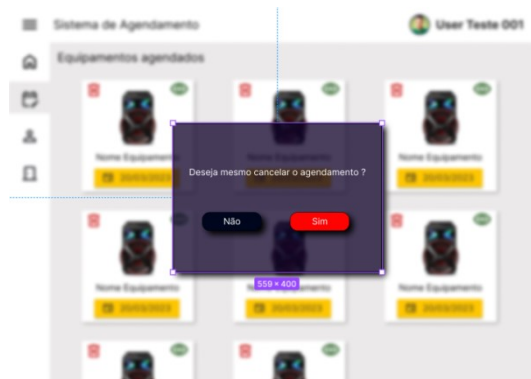
Na interface de agendamentos acessada por meio da barra de navegação, o usuário encontrará os equipamentos que foram agendados com sua respectiva data. Nos *cards* se encontram além da data de agendamento, duas metáforas de interação e a imagem da máquina, a primeira metáfora é o olho verde presente na parte superior à esquerda do contêiner, onde ao clicar o usuário terá acesso a agenda do respectivo equipamento. Já a segunda metáfora é a lixeira vermelha onde o usuário poderá realizar o cancelamento do agendamento.

Imagem 9 – Agenda



FONTE: Figma.

Imagem 10 – Alerta de confirmação



FONTE: Figma.

6.6 Tela de Perfil

Nesta o usuário pode ver os dados a qual estão vinculados ao seu código, bem como alterar seis deles no caso de um docente, e cinco no caso de ser um colaborador.

Imagem 11 – Perfil Docente



FONTE: Figma.

Imagem 12 – Perfil Colaborador

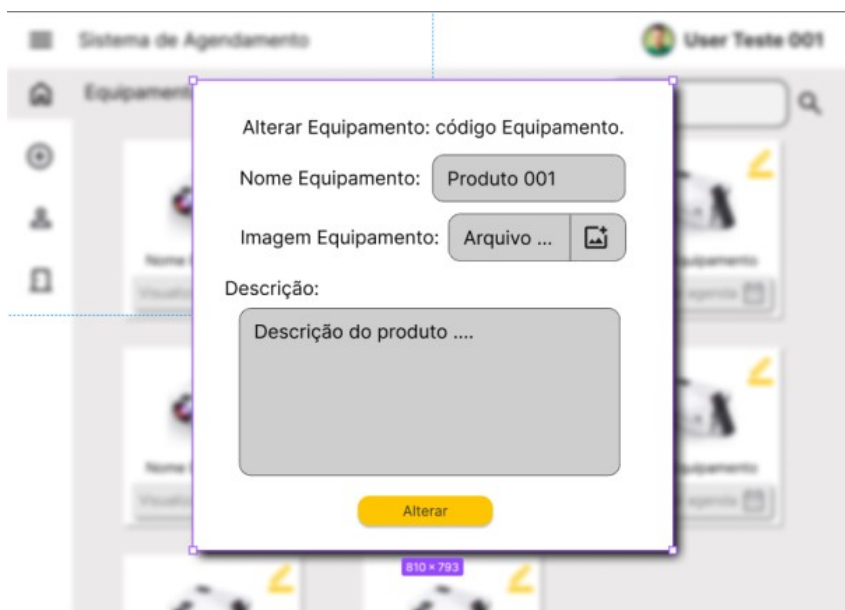


FONTE: Figma.

6.7 Telas específicas do colaborador

Dentro do sistema existem funções a qual apenas o colaborador tem acesso, é por conta disso o mesmo tem acesso a duas interfaces únicas para criação e outra para alteração dos equipamentos, já a exclusão do produto é realizada na tela *home* como pode ser visto na imagem 5.

Já para acessar a tela de alteração, o mesmo necessita clicar no lápis amarelo que fica no lado oposto da lixeira vermelha, na interface *Home*, e por fim, para acessar a tela para adicionar um novo equipamento, se faz necessário apenas selecionar o símbolo de soma presente na barra de navegação.

Imagem 13 – Tela de alteração**FONTE: Figma.****Imagem 14 – Tela de criação****FONTE: Figma.**

7 CODIFICAÇÃO

Como explicitado anteriormente, este projeto tem sua estrutura orientada a objetos que é um dos atributos da linguagem de programação C#. Esta é uma linguagem de programação que herda muitos conceitos das suas antecessoras C e C++, todavia se diverge pela adição de funcionalidade e a estrutura orientada a objeto.

Nas imagens abaixo, temos os códigos de criação das classes presente no diagrama de classe apresentado no capítulo 5, além dos conceitos de Herança, classe e polimorfismo presente no código.

Imagem 15 – Classe Usuário

```

13 public class Usuario
14 {
15     int codigo;
16     string nome;
17     string senha;
18     int tipo;
19
20     public Equipamentos VisualizarEquipamento(int identificador)...
25
26     public Usuario Visualizar(int codigo)...
31
32     public Usuario Deletar(int codigo)...
37
38 }

```

FONTE: Arquivo Pessoal.

Imagem 16 – Classe Docente.

```

40 4 referências
41 public class Docente : Usuario
42 {
43     Data nascimento;
44     string cargo;
45
46     0 referências
47     private Usuario Alterar(int codigo, string nome, string senha, int tipo, Data nascimento, string cargo)...
50
51     0 referências
52     private Equipamentos Agendar(int identificador, int codigo)...
57
58     0 referências
59     private void Cancelar(int identificador, int codigo)...
63     0 referências
64     private void Adicionar(string nome, string senha, int tipo)...
76 }

```

FONTE: Arquivo Pessoal.

Imagem 17 – Classe Colaborador.

```

77 0 referências
78 public class Colaborador : Usuario
79 {
80     string nomeEmpresa;
81
82     0 referências
83     public Equipamentos Editar(int Identificador, string nome, string descricao)...
84     0 referências
85     public Equipamentos Adicionar(string nome, string descricao)...
86     0 referências
87     public void Excluir(int identificador)...
88
89 }

```

FONTE: Arquivo Pessoal.

Imagem 18 – Classe Equipamento.

```

10 15 referências
11 public class Equipamentos
12 {
13     public int identificador;
14     public string nome;
15     public string descricao;
16     public string disponibilidade;
17
18     5 referências
19     public Equipamentos(int identificador, string nome, string descricao, string disponibilidade)...
20
21 }

```

FONTE: Arquivo Pessoal.

Imagem 19 – Classe Data.

```

9 5 referências
10 public class Data
11 {
12     int dia;
13     int mes;
14     int ano;
15
16     0 referências
17     public static Data data(int dia, int mes, int ano)...
18
19 }

```

FONTE: Arquivo Pessoal.

Podemos identificar a utilização de herança e polimorfismo nas imagens 16 e 17, onde as classes docente e colaborador herdam os atributos e métodos da superclasse usuário. Já o conceito de polimorfismo é empregado nesta mesmas classes por meio do método adicionar, que mesmo contendo o mesmo nome possuem funções divergentes.

8 TESTES

Seguindo os padrões da norma MPS.BR, após a conclusão do projeto serão realizadas as avaliações padrões que são exigidas pela norma, entretanto durante o desenvolvimento deste se fez a utilização de diversos testes unitários e de integração.

O teste unitário é realizado pelo desenvolvedor com a finalidade de garantir a plena funcionalidade do código produzido, enquanto o teste de integração é realizado para garantir que o código continue funcionando plenamente quando integrado com os outros componentes desenvolvidos.

Para validar que o software atenda todos os requisitos funcionais que foram estabelecidos, pode se realizar um teste com base em uma lista de tarefas que tem como conteúdo todos os requisitos funcionais que foram estabelecidos na etapa de levantamento de requisitos.

Já para a validação dos requisitos de negócio, após a introdução do sistema no seu ambiente final, pode ser disponibilizado um questionário para que os usuários finais respondam, com o intuito de averiguar se os requisitos de negócio estão de acordo com o desejado pelo cliente.

Por fim, no anexo deste existem os modelos de questionários e roteiros de teste que seriam utilizados após o desenvolvimento do produto e sua integração.

CONCLUSÃO

Atribuir qualidade a um *software* não é uma tarefa complicada, e como visto no desenrolar deste projeto, podemos adquirir tal qualidade seguindo um modelo de ciclo de vida para delimitar e otimizar as etapas do desenvolvimento, bem como aplicar uma norma de qualidade que apoiará as etapas do desenvolvimento do produto, bem como certificará o *software* assim que for finalizado, e realizar testes nas etapas para certificar o pleno funcionamento e averiguar a satisfação do cliente sobre o produto entregue.

Vimos também sobre o conceito programação orientada objeto (POO) bem como do diagrama de classes seguindo os padrões UML, estes abordados pelo fato que a linguagem utilizada para idealizar o escopo do sistema foi a C# da Microsoft, linguagem essa que tem como suas principais características a utilização dos conceitos provindos o POO de classe, objeto, herança e polimorfismo, bem como a integração com as outras tecnologias do framework .NET.

Também foi desenvolvido um design conceitual para a interface do sistema solicitado, e nesta foi aplicada o da interação humano-computador, visando que o *software* seja usual, de fácil aprendizado e visualmente agradável, e para isso, muito se usou de metáforas de interação. Por fim, para o desenvolver deste foi realizado alguns testes conceituais que estão disponíveis em anexo, bem como a aplicação de um teste que qualidade do protótipo desenvolvido que também está presente em anexo deste trabalho com as respectivas respostas.

REFERÊNCIAS

Paula Filho, Wilson Pádua. **Engenharia de Software: Fundamentos, Métodos e Padrões**. 3ª Edição. Rio de Janeiro: LTC, 2012.

Costa, Ivanir. **Engenharia de software**. / Ivanir Costa. – São Paulo: Editora Sol, 2014.

Ribeiro, André Luiz. **Engenharia de Software II**. / André Luiz Ribeiro. – São Paulo: Editora Sol, 2015.

Deitel, Paul; Deitel, Harvey. **Java: Como programar**. Tradução Edson Furmankiewicz. São Paulo: Person Education do Brasil, 2017.

Isotani, Seiji; V.Rocha, Rafaela. **Modelos de Processo de Software**. Disponível em: https://edisciplinas.usp.br/pluginfile.php/2939594/mod_resource/content/1/Aula02_ModelosProcessos.pdf. Acesso em: 21 de Março de 2023.

SOFTEX (org). **MPS.BR – MELHORIA DE PROCESSO DO SOFTWARE BRASILEIRO**: Guia Geral MPS de Software. Disponível em: https://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2012-c-ISBN-1.pdf. Acesso em: 23 de Março de 2023.

TRTPR. **CONCEITO: Requisitos não-funcionais**. Disponível em: https://www.trt9.jus.br/pds/pdstr9/guidances/concepts/supporting_requirements_B2C4D610.html. Acesso em: 15 de março de 2023.

APÊNDICE A - Lista de tarefas para testes de sistema.

Planilha disponível na pasta “Testes”.

Imagem 01 - Teste de sistema Docente.

Tarefas		0/12 concluído
✓	Tarefa	
<input type="checkbox"/>	Login Docente.	
<input type="checkbox"/>	Visualizar Equipamentos da instituição.	
<input type="checkbox"/>	Entrar na agenda de um equipamento.	
<input type="checkbox"/>	Visualizar datas disponível, ocupadas e indisponível.	
<input type="checkbox"/>	Realizar agendamento em uma data disponível.	
<input type="checkbox"/>	Verificar se a data do agendamento está marcada como ocupada.	
<input type="checkbox"/>	Ir para área de agendamentos.	
<input type="checkbox"/>	Verificar se o equipamento agendado está na página.	
<input type="checkbox"/>	Visualizar agenda do equipamento pela página de agendamentos.	
<input type="checkbox"/>	Cancelar agendamento do equipamento.	
<input type="checkbox"/>	Ir para página de perfil.	
<input type="checkbox"/>	Sair do sistema.	

FONTE: Planilhas Google.

Imagem 02 - Teste de sistema Colaborador.

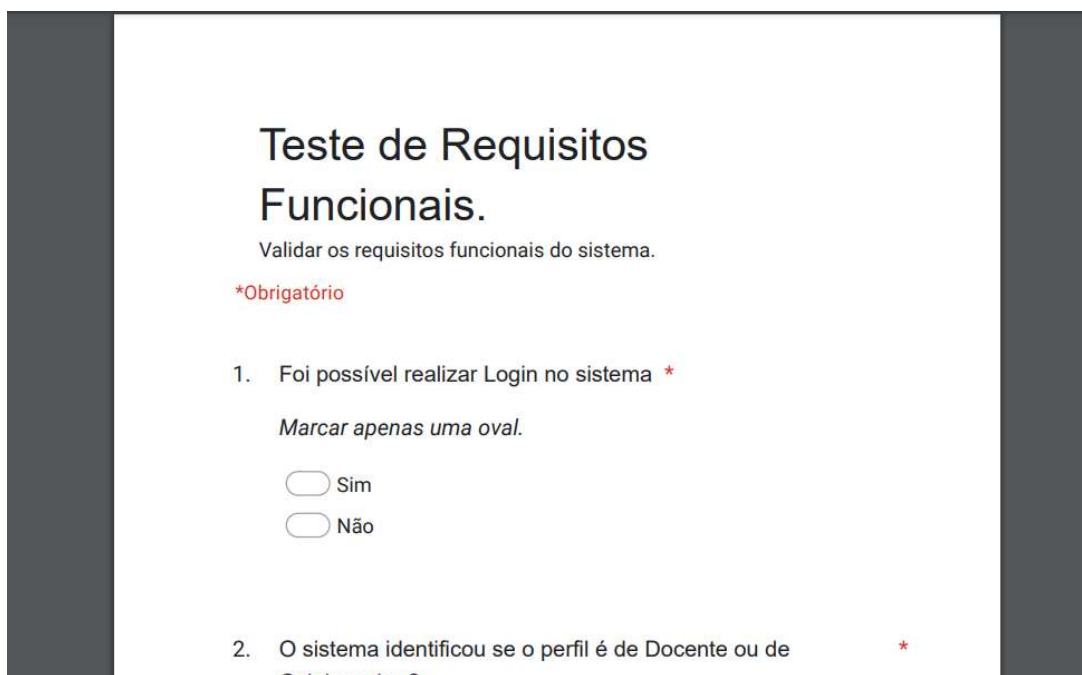
Tarefas		0/9 concluído
✓	Tarefa	
<input type="checkbox"/>	Login Colaborador.	
<input type="checkbox"/>	Visualizar Equipamentos.	
<input type="checkbox"/>	Entrar na agenda de um equipamento.	
<input type="checkbox"/>	Modificar propriedades de um equipamento.	
<input type="checkbox"/>	Deletar um equipamento..	
<input type="checkbox"/>	Adicionar um equipamento.	
<input type="checkbox"/>	Verificar se o equipamento incluído está na página de equipamentos.	
<input type="checkbox"/>	Ir para página de perfil.	
<input type="checkbox"/>	Sair do sistema.	

FONTE: Planilhas Google.

APÊNDICE B - FORMULÁRIO PARA TESTE DE SISTEMA

Formulário disponível em PDF na pasta “Testes”.

IMAGEM 01 - Formulário para teste de sistema.



The image shows a Google Form titled "Teste de Requisitos Funcionais." with the subtitle "Validar os requisitos funcionais do sistema." Below the subtitle, there is a red asterisk and the word "Obrigatório" in red. The form contains two questions. Question 1 is "Foi possível realizar Login no sistema" with a red asterisk. Below it, there is a instruction "Marcar apenas uma oval." and two radio button options: "Sim" and "Não". Question 2 is "O sistema identificou se o perfil é de Docente ou de" followed by a red asterisk. The text "Colaborador?" is partially visible below question 2.

Teste de Requisitos Funcionais.

Validar os requisitos funcionais do sistema.

***Obrigatório**

1. Foi possível realizar Login no sistema *

Marcar apenas uma oval.

☐ Sim

☐ Não

2. O sistema identificou se o perfil é de Docente ou de *
Colaborador?

FONTE: Google Forms.

APÊNDICE C - FORMULÁRIO PARA TESTE DE ACEITAÇÃO DO PROTÓTIPO

Formulário disponível em PDF na pasta “Testes”.

IMAGEM 01 - Formulário para teste de aceitação do protótipo.

The image shows a Google Form titled "Formulário de Qualidade do Protótipo". The form is designed to evaluate the quality of a prototype. It includes a description of the form's purpose and a list of questions. The first question is "1. Acho que gostaria de usar este sistema frequentemente. *", which is marked as mandatory. Below the question, there is a instruction "Marcar apenas uma oval." and a label "Discordo Plenamente." followed by three radio button options labeled 1, 2, and 3. The form is displayed on a white background with dark grey sidebars.

Formulário de Qualidade do Protótipo

Formulário desenvolvido para avaliar a qualidade do protótipo de telas desenvolvido no figma, que tem como finalidade averiguar se o mesmo é agradável visualmente e de fácil compreensão.

***Obrigatório**

1. Acho que gostaria de usar este sistema frequentemente. *

Marcar apenas uma oval.

Discordo Plenamente.

1 ☐

2 ☐

3 ☐

FONTE: Google Forms.

APÊNDICE D – Layout.

Na pasta “Design” existe dois arquivos, um deles é o PDF com todas as interfaces, e o outro um arquivo do figma.

Aqui um link para acessar pela WEB o Layout do projeto pelo Figma:

<https://www.figma.com/file/21AhIEqITCfyVuZMw0FJO9/Projeto-Integrado-V?node-id=0%3A1&t=Ejm5GqX7440YcXIN-1>