**UNIVERSIDADE FEDERAL DO CEARÁ**

**INSTITUTO UNIVERSIDADE VIRTUAL**

**CURSO DE GRADUAÇÃO EM SISTEMAS E MÍDIAS DIGITAIS**

**NICHOLAS TRECE ESCOSSIO**

# UMA PROPOSTA DE DESENVOLVIMENTO FRONT-END PARA FICHAS PERSONALIZADAS DE RPG ONLINE

**FORTALEZA**

**2023**

NICHOLAS TRECE ESCOSSIO

# UMA PROPOSTA DE DESENVOLVIMENTO FRONT-END PARA FICHAS PERSONALIZADAS DE RPG ONLINE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto Universidade Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Orientador: Prof. Dr. Leonardo Oliveira Moreira

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

E73p Escossio, Nicholas Trece.
Uma proposta de desenvolvimento front-end para fichas personalizadas de RPG online / Nicholas Trece Escossio. – 2023.
47 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual, Curso de Sistemas e Mídias Digitais, Fortaleza, 2023.
Orientação: Prof. Dr. Leonardo Oliveira Moreira.

1. RPG. 2. Front-End. 3. Angular. 4. Open-Source. I. Título.

CDD 302.23

NICHOLAS TRECE ESCOSSIO

# UMA PROPOSTA DE DESENVOLVIMENTO FRONT-END PARA FICHAS PERSONALIZADAS DE RPG ONLINE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto Universidade Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Aprovada em:

BANCA EXAMINADORA

---

Prof. Dr. Leonardo Oliveira Moreira (Orientador)
Universidade Federal do Ceará (UFC)

---

Prof. Me. Wellington Wagner Ferreira Sarmento
Universidade Federal do Ceará (UFC)

---

Prof. Dr. José Gilvan Rodrigues Maia
Universidade Federal do Ceará (UFC)

À minha família, por todo amor,carinho, dedicação e educação que me deram durante o meu crescimento sem me deixar faltar nada mesmo nos momentos de dificuldade e me deram o apoio necessário para seguir em frente independentemente das adversidades que surgiam em minha frente.

## AGRADECIMENTOS

Ao Prof. Dr. Leonardo Oliveira Moreira por me orientar em meu trabalho de conclusão de curso me ajudando a tornar o que eu achava impossível possível.

Ao Prof. Dr. Tobias Rafael Fernandes Neto, coordenador do Laboratório de Sistemas Motrizes (LAMOTRIZ) onde este *template* foi desenvolvido.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

Aos meus pais, minha família e meus amigos que me apoiaram e me confortaram nos momentos difíceis e me incentivaram a mesmo com as adversidades continuar até o final e me fizeram perceber que os as pequenas decisões com pequenos passos podem se tornar grandes feitos.

Aos meus pais, irmãos e sobrinhos, que nos momentos de minha ausência dedicados ao estudo superior, sempre fizeram entender que o futuro é feito a partir da constante dedicação no presente!

Agradeço à todos os professores que sempre me proporcionaram momentos de reflexão e aprendizado, amizades e um ensino impecável na construção de quem sou hoje me fazendo a cada semestre entender mais sobre a área e explorar as oportunidades que apareciam em minha frente.

Agradeço aos meus amigos José Cleanto Feitosa Arrais Neto e Pedro Henrique Pinheiro Costa pelo o companheirismo durante toda a graduação podendo agregar valores entre si e sempre dedicados a entregas os trabalhos das disciplinas com excelência.

Agradecimento à maior designer de todas, Alana Campelo, por toda a ajuda na idealização de uma interface para este projeto.

"O sonho é que leva a gente para frente. Se a gente for seguir a razão, fica aquietado, acomodado."

(Ariano Suassuna)

## RESUMO

Com o sucesso dos sistemas de RPG, surgiram diversas opções, e as fichas personalizadas tornaram-se uma realidade, permitindo que os usuários experimentem e compartilhem suas próprias fichas. Para facilitar a criação e o gerenciamento dessas fichas, especialmente à distância, surge a necessidade de uma aplicação web gratuita e de fácil utilização. O projeto propõe desenvolver uma aplicação que permita jogar sistemas de RPG existentes e testar fichas de sistemas em desenvolvimento, oferecendo uma plataforma simples, barata e acessível aos usuários. A proposta é uma aplicação de código aberto que recebe contribuições de diversos desenvolvedores para sua construção e melhoria contínua. Inspirando-se em plataformas como o Roll20, que fornecem modelos de fichas para os sistemas de RPG mais conhecidos, busca-se permitir a personalização de fichas para novos sistemas gratuitamente. Além disso, a construção de novas fichas não exige que o usuário tenha conhecimentos de programação, uma vez que existem sistemas de RPG emergentes no Brasil que não são atendidos pela ferramenta mencionada.



**Palavras-chave:** RPG. Front-End. Angular. Open-Source

## ABSTRACT

With the success of RPG systems, various options have emerged, and customized character sheets have become a reality, enabling users to experiment and share their own sheets. To facilitate the creation and management of these sheets, especially in remote settings, there is a need for a free and user-friendly web application. This project aims to develop an application that allows users to play existing RPG systems and test character sheets for systems in development, offering a simple, affordable, and accessible platform. The proposed application is an open-source project that welcomes contributions from developers for its construction and continuous improvement. Drawing inspiration from platforms like Roll20, which provide pre-made character sheet templates for well-known RPG systems, the objective is to enable free customization of character sheets for new systems. Additionally, the construction of new character sheets does not require programming knowledge, as there are emerging RPG systems in Brazil that are not supported by the mentioned tool.



**Keywords:** RPG. Front-End. Angular. Open-Source

## LISTA DE FIGURAS

## LISTA DE CÓDIGOS-FONTE

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| APIs | *Application Programming Interfaces* |
| CSS | *Cascading Style Sheets* |
| CVS | *Concurrent Versions System* |
| DOM | *Document Object Model* |
| HTML | *HyperText Markup Language* |
| JSON | *JavaScript Object Notation* |
| MDS | Metodologias de Desenvolvimento de Software |
| MVC | *Model-View-Controller* |
| MVVM | *Model-View-ViewModel* |
| RPG | *Role Playing Game* |
| RPGs | *Role Playing Games* |
| SEO | *Search Engine Optimization* |
| SPA | *Single Page Application* |
| SVN | *Subversion* |
| URL | *Uniform Resource Locator* |
| URLs | *Uniform Resource Locators* |

**SUMÁRIO**

# 1 INTRODUÇÃO

O *Role Playing Game* (RPG) é um jogo de interpretação de papéis como se fosse uma especie de teatro. Para que uma partida de RPG seja possível é necessário criar fichas nas quais dão vida e as características do papel a ser interpretado pelo jogador onde se atribui virtudes e defeitos. Além disso, para a partida acontecer também é necessário ter um mestre que vai contar a história dessa aventura e os jogadores que vão reagir a história de acordo com cada um de seus papeis para dar vida e decidir o que vão fazer e como vão fazer durante a partida. Resumidamente o RPG é uma história contada pelo mestre que muda de acordo com as decisões dos jogadores.

Todo RPG precisa de alguns elementos chaves para que possa funcionar adequadamente que são: Sistema de regras, a história contada pelo mestre e os jogadores. Além disso também existem características que não são obrigatórias, mas são comuns que é a questão da interatividade já que os jogadores fazem a história e o trabalho em grupo. Um dos sistemas de *Role Playing Games* (RPGs) mais famosos existentes atualmente e que segue essa linha é o *Dungeon and Dragons* (WIZARDS OF THE COAST LLC, 2022) que conta com um rico sistema onde com base nele são criados as personalidades dos personagens e são com essas informações do sistema que são criadas as fichas de RPG.

Fichas de RPG personalizadas são atualmente uma realidade dado que com o sucesso de vários desses sistemas e a adesão ao publico deles o único requisito limitante para o surgimento de um novo é a imaginação do criador. E daí a necessidade dessa liberdade para qualquer usuário com qualquer ideia possa experimentar e compartilhar a sua ficha personalizada para testar e jogar seu sistema. Com os diversos sistemas de RPG que andam surgindo nos últimos tempos ou até mesmo sistemas que são criados apenas para um grupo de amigos poderem jogar para compartilhar momentos surge a necessidade de que mesmo a distância eles possam criar e gerenciar essas fichas através de ferramentas *online* gratuitas e de fácil manuseio. A proposta é que por meio da aplicação o usuário possa criar uma ficha que adequá as suas necessidades sem a necessidade de pagar ou ter conhecimentos em linguagem de computador como é ofertado no Roll20.

Com isso, se vê a necessidade de criar uma aplicação web que permita tanto que jogadores especialistas possam jogar sistemas de RPGs já existentes quanto também poder realizar testes em fichas de sistemas que estão em desenvolvimento e possibilitar uma possível avaliação do estado atual de novos sistemas baseados em testes quantitativos proporcionados

pela a plataforma a ser desenvolvida e oferecendo ao usuário uma forma simples, barata e de fácil utilização ao usuário especialista. Para o desenvolvimento de uma solução computacional vão ser avaliados as vantagens e desvantagens dos três *frameworks* mais populares baseados na linguagem JavaScript para a escolha de qual seria a melhor solução. os *frameworks* são: Angular, Vue.js e React.

## 1.1 Justificativa

O projeto surgiu após a necessidade de encontrar uma ferramenta que permitisse a criação de fichas de RPG personalizadas que tivessem características a escolha do usuário. Durante essa busca foi encontrado a ferramenta Roll20 no entanto esse tipo de função é disponível apenas para usuários vip e também seria necessário ter conhecimentos em linguagens de computador para a programação da ficha. O Roll20 apesar de disponibilizar muitas outras funcionalidades que podem auxiliar em uma mesa RPG com encontros *online* o problema em ser uma ferramenta paga e com a necessidade do usuário ter conhecimentos técnicos não atende ao cenário atual de usuários especialistas que necessitam de uma ferramenta de uso simples e gratuito. Daí surgiu a ideia de criar uma aplicação onde o usuário coloque apenas os nomes e valores que a ficha deve conter e ela seja gerada automaticamente sem a necessidade do usuário possuir conhecimentos na área para tal desenvolvimento além de ser uma aplicação *open-source* e gratuita para que possa ser melhorada constantemente pela a comunidade.

[1]

## 1.2 Objetivos

### *1.2.1 Geral*

Apresentar uma solução computacional para criação de fichas personalizadas de RPG por meio de aplicações web.

[1] Link do Roll20 https://app.roll20.net/why-subscribe-to-roll20

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.1 RPG

RPG é um estilo de jogo caracterizado pela interpretação de papeis dentro de um contexto. Para isso, é necessário que existam os jogadores que interpretam os papeis ao criarem seus personagens e o mestre que narra a descrição de cenas e cenários para que os jogadores possam imaginar e fazer suas escolhas. De acordo com Pavão (1999), esse estilo de jogo surgiu por volta dos anos 70 como uma melhoria dos jogos de guerra da época e inspirados pelos livros de fantasia do Tolkien (1995) e acabou se popularizando posteriormente.

> O RPG consiste em cada jogador "interpreta" um personagem, herói ou protagonista e, conforme o jogo se desenvolve, as suas ações, através do personagem, vão fluindo e acabam por ditar o andamento da história e os rumos que ele seguirá. Para iniciar o jogo e se situar, (localização espaço temporal), os personagens de cada jogador, existe a função do mestre de jogo, a pessoa que irá também fazer a parte dos outros personagens que farão parte da história, com os quais os personagens dos outros jogadores irão interagir, seja pedindo informações,subornando, seduzindo, interrogando, fazendo parcerias ou simplesmente entrando em combate (BASSANI, 2002)

Cada jogador para interpretar seu personagem inicialmente precisa ler um livro de regras do sistema no qual ele escolheu jogar e a partir delas criar seu personagem com qualidades, defeitos, aparência, características psicológicas (modo de agir e de pensar), nome, a história por trás daquele individuo explicando sua história e suas ambições e também suas escolhas de equipamentos e atributos que definem o seu personagem. Através dessa ficha que resume o papel a ser interpretado é onde o mestre de jogo vai contar uma história na qual o jogador decide o que fazer, quando fazer e como fazer.

Geralmente as partidas são jogadas com um grupo de jogadores e um mestre que é responsável por de acordo com as decisões tomadas pelos atores a direção em que a história vai seguir. Inserido na história existem situações onde é necessária uma rolagem de dados de acordo com a caraterista a ser solicitada pelo o mestre. Por exemplo em um jogo de D&D onde o jogador quer escalar uma montanha. O mestre do jogo deve pedir uma rolagem no qual o sucesso dessa escalada depende do quão ele foi bem ao rodar o dado, quanto maior o valor do dado com mais habilidade ele realizou aquela determinada ação.

## 2.2 Jogos Digitais

Para definir jogos digitais inicialmente é preciso definir o que é um jogo. Um jogo nada mais é que uma brincadeira que possui um conjunto de regras e um objetivo que deve ser realizado para a finalização dele. Na própria pre-história já tínhamos jogos que se baseavam mais em atividades físicas como corridas e escaladas e futuramente tivemos o inicio das olimpíadas que eram jogos também relacionados a atributos físicos. De acordo com Huizinga (2019) o jogo pode ser definido como uma atividade lúdica onde através da imaginação é realizado um ato voluntario no qual uma tensão é gerada juntamente com a falta de informação de qual será o desfecho já que durante um jogo uma de suas características é a surpresa que ocorre ao final já que não se pode conhecer qual a forma na qual ele deve terminar.

Com isso, não conhecer como o jogo acaba é importante nos aspectos dos jogos porque não interfere e cabe ao jogador entender o que se passa durante o jogo e o que deve ser feito para chegar ao final dele utilizando de estratégias e informações que são ganhas através do decorrer do jogo. Existem vários tipos de jogos como: Jogos cooperativos onde os jogadores se unem para alcançar o mesmo objetivo, jogos de tabuleiro como por exemplo os WarGames dos anos 70 que se caracterizavam por um jogo de estrategia com movimentação de peças para "conquistar" setores e que posteriormente deram origem ao RPG, jogos de cartas que através de um baralho os jogadores fazem estrategias e disputam entre si etc.

Jogos digitais são, de acordo com Schuytema (2008) um tipo de jogo realizado através de dispositivos eletrônicos que exercem uma atividade lúdica que se caracterizam por ações e decisões que chegam a uma conclusão pelas atitudes tomadas que o levaram até aquele momento. Essas atitudes são restringidas por uma serie de regras que são definidas pelo o jogo no qual foram programadas utilizando linguagem de computador para suas especificações e definições.

O mundo gerado por essas regras muda de acordo com as decisões do jogador seja por alterar a narrativa do jogo para concretizar a causa e efeito que as ações do jogador tiveram dentro daquele universo e que essas atitudes levaram ao estado atual. Além de limitar as ações dentro de um conjunto de regras enquanto o objetivo de quem joga é realizar o desafio que lhe foi proposto o objetivo do software e a programação dele foi feita puramente para dificultar e atrapalhar com que o jogador consiga realizar o que deseja aplicando desafios durante sua jornada.

Com a pandemia os RPGs de mesa que já eram uma realidade no cenário virtual

passaram a ser mais frequentes. Pelo o fato de que as restrições sanitárias se aplicaram não era possível transitar normalmente e se reunir na casa de amigos ou familiares para realizar as partidas e por isso não só jogadores de RPG, mas o público mundial tanto por motivos profissionais quanto por motivos recreativos foram forçados a investir em um computador ou em um *notebook* para facilitar esse processo.

Ferramentas como Roll20 (2022) que e FireCast (2022) que já eram utilizadas para viabilizar as partidas se tornaram ainda mais populares e nesse mesmo período uma crescente de *lives* com partidas de RPGs foram surgindo. Ao mesmo tempo novos sistemas *Role Playing* iam surgindo e dando necessidade a ferramentas que possibilitassem a criação de fichas personalizadas de RPGs com fácil manuseio e acesso. O Roll20 apesar de ser uma ferramenta muito utilizada e com o objetivo de permitir que os jogadores se reúnam para jogar independente da distancia e também possui a proposta para que caso você não tenha um grupo seja possível que você ache um ele não te permite que gratuitamente e sem necessidade de programação você crie uma ficha personalizada que atenda a necessidades mais especificas de um novo sistema.

## 2.3 Desenvolvimento de jogos

No desenvolvimento da maioria dos jogos digitais da atualidade se serve de *frameworks* para o sua implementação. Isso se dá, de acordo com Pereira *et al.* (2017), pelo o fato de que desenvolver jogos do início sem funções básicas já construídas para auxiliar o desenvolvimento não seja uma tarefa trivial e é por isso que a utilização dos *frameworks* são essenciais. Eles existem para minimizar a resolução de problemas que já foram resolvidos em outros casos e permitir a reutilização de código e assim diminuindo a complexidade, aumentando a produtividade e consequentemente diminuindo o tempo e os recursos gastos na produção de um jogo e por esse motivo que a maioria dos jogos utilizam de algum *framework* em sua produção.

## 2.4 Possibilidade de frameworks para a criação de interfaces gráficas

Um *framework* de desenvolvimento é um conjunto de soluções que geralmente são necessárias na maioria das aplicações já implementadas em um só lugar. A ideia é aproveitar boa parte das soluções já criadas e que são comuns a grande parte das aplicações e torna-las reutilizáveis e de fácil acesso além de executar boa parte das tarefas de mais baixo nível sem uma necessidade real de implementação por parte do desenvolvedor e com isso permitindo que o

foco fique apenas nas novas funcionalidades a serem criadas para a aplicação em questão.

Para o desenvolvimento será escolhido um dos *frameworks* baseados em JavaScript mais populares no mercado em termos de *front-end*. Temos o Angular, Vue.js, e React. O Angular é um *framework open-source* pela a Google baseado na linguagem JavaScript e que usa o TypeScript que é um superconjunto de recursos para JavaScript criado pela a Microsoft para melhorar a padronização de código, qualidade e segurança e por isso é muito utilizado principalmente em aplicações de larga escala. Além do TypeScript, o Angular possui outras vantagens como: transformar o *HyperText Markup Language* (HTML) da página dinamicamente através de bibliotecas,possibilidade para reaproveitamento de código e uma grande e variada quantidade de *plugins*, tem suporte em diversas plataformas e uma grande comunidade.

> There are many advantages to Angular. It is supported on various platforms. Angular presents tools and design patterns to build a good project. Angular is designed using TypeScript, a superset of JavaScript. The components are decoupled. Angular is intended to be methodically tested and it supports both unit and end-to-end testing. Also, Angular is actively maintained, which means that it has a huge community and environment (Novac *et al.* (2021), p. 1)

O Vue.js é um *framework* para aplicações web criado por Evan Yo em 2014 utilizado geralmente para criar *Single Page Application* (SPA) e é bem popular pelo o fato de sua curva de aprendizado ser baixa, além de ser acessível e bastante versátil. De acordo com Novac *et al.* (2021) (2021) o Vue.js tem uma arquitetura adaptável juntamente com uma documentação detalhada e extensa além de ser simples de entender e bastante claro em suas implementações, mas não possui tantas bibliotecas ou *plugins* como o React e o Angular.

O React é uma biblioteca também baseada na linguagem JavaScript para criação de interfaces. Ele foi criado em 2011 pelo Facebook[1] e chamou atenção pela forma na qual conseguia gerenciar os estados de uma aplicação de forma simples e prática. Além disso, de acordo com Kaushalya e Perera (2021) ele possui estabilidade, reuso de código, alta performance e estabilidade aliado com o fato de ser um *framework* de fácil aprendizado e de rápido desenvolvimento.

Em relação as desvantagens segundo Novac *et al.* (2021) o Angular tem uma grande complexidade em sua curva de aprendizado, poucas opções de *Search Engine Optimization* (SEO), E no caso das versões do AngularJS para o Angular não se pode trocar com facilidade devido a vários problemas de compatibilidade, mas ainda assim vários sites o utilizam em seu desenvolvimento como: Rockstar Gamer, Google Cloud e YouTubeTV. O Vue.js de acordo

[1] Facebook. Disponível em: <https://www.facebook.com/>. Acesso em: 18 de outubro de 2022.

com Novac *et al.* (2021) tem uma pequena comunidade de desenvolvedores em comparação aos outros *frameworks* além disso, também foi recentemente desenvolvido e isso acaba tornando ele menos utilizado apesar de seu constante aprimoramento já que na maioria dos casos se procura ferramentas com mais tempo de uso e aprimoramento antes de utiliza-las. No React apesar de ser um *framework* bem popular e que cresceu rapidamente isso também trás malefícios. Segundo o Wickramasinghe (2022), o ambiente está sempre mudando e isso dificulta a adaptação dos programadores. Um exemplo dessa mudança é a forma de componentização que antes era feito através de classes e posteriormente foi mudado para padrão de funções computacionais. Além disso, a documentação não é tão detalhada quanto o do Angular.

## 2.5 Desenvolvimento com Angular

No Angular, um padrão de projeto comumente utilizado é o padrão *Model-View-ViewModel* (MVVM). Esse padrão é uma variação do padrão *Model-View-Controller* (MVC) e é especialmente adequado para aplicações baseadas em componentes, como as desenvolvidas com o Angular. As principais partes do padrão MVVM no Angular são:

- Model: O Model representa os dados e a lógica de negócios da aplicação. Ele geralmente consiste em classes ou interfaces que definem os objetos de dados e serviços relacionados. No Angular, o Model é geralmente implementado por meio de classes de serviço, que são responsáveis por recuperar e manipular os dados.
- View: A View é a camada de apresentação da aplicação. No Angular, a View é definida por meio de *templates* HTML que descrevem a estrutura e o layout da interface do usuário. A View é responsável por exibir os dados do Model e interagir com o usuário.
- ViewModel: O ViewModel atua como um intermediário entre o Model e a View. Ele fornece a lógica necessária para manipular os dados e responder a eventos na View. No Angular, o ViewModel é implementado pelos componentes. Os componentes recebem os dados do Model, os processam e os disponibilizam para a View por meio de propriedades e métodos. Além disso, os componentes respondem a eventos e interações do usuário, atualizando o Model conforme necessário.

O padrão MVVM no Angular também se beneficia dos recursos de *data binding* bidirecional fornecidos pelo *framework*. Isso permite que as alterações na View sejam refletidas automaticamente no ViewModel e vice-versa, facilitando a sincronização de dados entre as camadas. Além do padrão MVVM, também é comum no Angular utilizar outros padrões e

práticas recomendadas, como o padrão de componentização, injeção de dependência, separação de responsabilidades, entre outros. Esses padrões ajudam a promover uma estrutura clara e organizada para o desenvolvimento de aplicações Angular. É importante ressaltar que o Angular é um *framework* flexível, permitindo a adoção de diferentes padrões de projeto com base nas necessidades específicas da aplicação e nas preferências da equipe de desenvolvimento. No Angular existem duas abordagens no desenvolvimento de telas que utilizam formulários que são: Abordagem `template driven` e `Reactive Forms`. No `template driven` temos as seguintes características:

- Abordagem baseada em diretivas onde a lógica do formulário é manipulado através de diretivas como, `ngModel`, `ngForm` e `ngSubmit`. Essas diretivas adicionam comportamento e funcionalidades ao formulário sem a necessidade de escrever o código no componente.
- Validações diretamente no HTML: as validações são definidas diretamente no HTML da página utilizando atributos como `required`, `minLenght`, `maxLenght`, `pattern` etc. A validação é feita automaticamente pelo Angular com base nos atributos adicionados.
- Verificação automática de estado e validação: O Angular verifica automaticamente o estado de cada controle do formulário como (por exemplo: `touched`, `dirty` e `valid` e `invalid`) e exibe as mensagens de validação correspondentes enquanto o estado do formulário é atualizada conforme o usuário interage com os campos.
- Pouco controle do formulário: Com essa abordagem não conseguimos tanto controle sobre o formulário como em comparação ao `Reacitve Forms`. A lógica e a manipulação dos dados do formulário são tratadas principalmente pelo Angular impedindo implementações de recursos mais complexos e personalizados.

No Reactive Forms:

- Abordagem baseada em desenvolvimento: A lógica do formulário é manipulada principalmente pelo o código escrito no componente. É preciso criar explicitamente instâncias de classes como `FormGroup`, `FormControl` e `FormBuilder` para construir e controlar o formulário.
- Validação feita através de funções: As validações são definidas usando funções fornecidas pelo Angular, como `Validators.required`, `Validators.minLenght`, `Validators.maxLenght`, `Validatorns.pattern` e outras. Além disso também podem ser criados validadores personalizados e essas validações são aplicadas aos controles do formulário durante a criação.
- Controle total sobre o formulário: Como todo o controle é feito pelo próprio desenvolve-

dor é possível realizar validações personalizadas, adicionar ou remover dinamicamente controles e verificar o estado do formulário controlando as interações de forma mais dinâmica.

- Manipulação assíncrona: Permite a manipulação de validações e operações assíncronas com mais facilidade, como validação remota e preenchimento automático de campos onde é possível que os controles sejam validados em tempo real com base em chamadas de *Application Programming Interface* (API) ou outros operações assíncronas.

Com base nessas duas formas de implementação, já que uma ficha é basicamente um formulário e para atender à necessidade proposta neste trabalho é necessário um formulário dinâmico onde seja possível adicionar e remover dinamicamente campos de acordo com a necessidade do usuário a abordagem escolhida foi a implementação utilizando o `Reactive Forms` do angular que apesar de necessidade de uma quantidade maior de desenvolvimento é a que permite a liberdade necessária para manipular um formulário conforme a solicitação do usuário.

Também é importante mencionar o que são diretivas já que eles são utilizadas de forma frequente na comunicação entre os componentes que fazem parte de uma página.

No Angular, uma diretiva é uma construção fundamental que permite adicionar comportamento e funcionalidade personalizada a elementos do *Document Object Model* (DOM). Elas são usadas para estender e manipular o comportamento dos componentes e elementos existentes, permitindo que você crie recursos reutilizáveis e personalizados. Existem três tipos principais de diretivas no Angular:

- Componentes: Os componentes são diretivas com um *template*, que define a estrutura e a aparência do elemento e sua lógica de exibição. Eles são os blocos de construção fundamentais da interface do usuário no Angular.
- Diretivas de Atributo: As diretivas de atributo são usadas para modificar o comportamento ou a aparência de um elemento específico. Elas são aplicadas como atributos em elementos HTML existentes e podem alterar estilos, adicionar eventos, interagir com outros componentes ou executar qualquer lógica personalizada.
- Diretivas Estruturais: As diretivas estruturais são usadas para modificar a estrutura do DOM adicionando, removendo ou manipulando elementos. Elas são aplicadas como atributos especiais no elemento pai e podem controlar a renderização condicional de elementos filhos com base em expressões ou condições.

As diretivas são definidas em classes TypeScript usando a anotação `@Directive` ou `@Component` do Angular. Essas classes podem ter propriedades, métodos e ganchos de ciclo de vida que permitem controlar o comportamento da diretiva. Para usar uma diretiva em um *template* Angular, é necessário importá-la no módulo apropriado e, em seguida, referenciá-la usando seu seletor em elementos ou atributos relevantes. A diretiva será aplicada a esses elementos ou atributos, fornecendo o comportamento definido pela diretiva. Em resumo, as diretivas no Angular permitem estender e personalizar o comportamento dos elementos do DOM, fornecendo uma maneira flexível de adicionar funcionalidades personalizadas aos seus aplicativos.

Um recurso também importante no angular é o serviço (*service*). um serviço é uma classe que possui uma responsabilidade específica e fornece funcionalidades e lógica reutilizáveis para serem compartilhadas em diferentes partes do aplicativo. Os serviços são usados para centralizar a lógica de negócio, interação com *Application Programming Interfaces* (APIs), manipulação de dados, entre outras tarefas. Os serviços no Angular são projetados para promover a separação de preocupações e o princípio de responsabilidade única. Eles ajudam a manter um código mais organizado, facilitam a reutilização e a testabilidade. Suas características e usos comuns são:

- Injeção de Dependência: Os serviços são tipicamente usados com o recurso de injeção de dependência do Angular. Isso significa que eles podem ser injetados em outros componentes, diretivas ou serviços, permitindo o compartilhamento de funcionalidades entre essas partes do aplicativo. A injeção de dependência é gerenciada pelo Angular, facilitando a criação e o uso dos serviços.
- Lógica de Negócio: Os serviços são usados para encapsular a lógica de negócio do aplicativo. Eles podem conter métodos e propriedades que lidam com cálculos, manipulação de dados, interações com APIs ou serviços externos, autenticação, autorização, entre outras tarefas relacionadas ao domínio específico do aplicativo.
- Compartilhamento de Dados: Os serviços são frequentemente usados para compartilhar dados entre componentes ou outras partes do aplicativo. Eles atuam como um intermediário para armazenar e fornecer acesso a dados que podem ser necessários em várias partes do aplicativo. Isso evita a duplicação de código e garante a consistência dos dados.
- Gerenciamento de Estado: Os serviços podem ser usados para gerenciar o estado do aplicativo. Isso pode envolver a centralização do estado compartilhado, manipulação de eventos, notificações ou atualizações de estado em diferentes partes do aplicativo.

- Comunicação entre Componentes: Os serviços podem ser usados para facilitar a comunicação entre componentes. Eles podem atuar como um canal de comunicação, permitindo que os componentes se inscrevam, emitam eventos ou compartilhem dados entre si por meio do serviço intermediário.

Portanto, um serviço no Angular é uma classe que fornece funcionalidades reutilizáveis, lógica de negócio, compartilhamento de dados e gerenciamento de estado em um aplicativo Angular. Eles são essenciais para manter um código organizado, promover a reutilização e facilitar a estabilidade.

Outro recurso essencial principalmente em termos de navegação entre páginas é o sistema de rotas do angular. O sistema de rotas é um recurso para criar aplicativos de página única (SPA) e controlar a navegação entre as diferentes partes do aplicativo. O sistema de rotas permite que você mapeie *Uniform Resource Locators* (URLs) específicas para componentes específicos, tornando a navegação do usuário mais intuitiva e fornecendo uma experiência de usuário mais fluida.

O primeiro passo é a configuração de rotas. Primeiro é preciso configurar as rotas pelo arquivo `app-routing.module.ts`. Nesse arquivo é necessário importar o módulo `routerModule` do Angular e usar o método `forRoot` para configurar as rotas. Cada rota é definida como um objeto que mapeia um caminho de *Uniform Resource Locator* (URL) para um componente específico. A segunda parte é no *template* do componente raiz do aplicativo que geralmente é chamado de `AppComponent` onde nele é necessário adicionar o `<router-outlet></router-outlet>`. Esse elemento é onde o Angular irá renderizar os componentes correspondentes às rotas. O Angular também, através dos *links* de navegação, permite que os usuários acessem diferentes partes da aplicação. Utilizando a diretiva `routerLink` para criar *links* para as rotas configuradas anteriormente como por exemplo: `<a routerLink=‘‘/dashboard’’>Dashboard</a>` cria um *link* para a rota `/dashboard`. Além dos *links* de navegação também é possível realizar o roteamento programático usando o serviço do `Router`.

O serviço `Router` permite navegue para uma rota específica usando métodos como `navigateByUrl` ou `navigate`. O sistema de rotas no Angular também suporta parâmetros na URL. É possível definir parâmetros de rota usando “`:`” seguidos pelo nome do parâmetro no caminho da rota. Exemplo: você pode ter uma rota como `/user:id` onde `id` é um parâmetro variável que pode ser acessado no componente correspondente. Por último, é possível fazer

roteamentos aninhados onde utilizando o `<router-outlet></router-outlet>` pode-se criar hierarquias de navegação mais complexas definindo rotas filhas dentro de rotas pais aninhando-os para renderizar os componentes quanto necessário.

## 2.6 Versionamento do código

O versionamento de código é uma prática essencial no desenvolvimento de software, envolvendo o controle e gerenciamento das diferentes versões do código-fonte ao longo do tempo. Permite registrar, rastrear e documentar as alterações realizadas, como correção de *bugs*, implementação de recursos e otimizações de desempenho. Isso facilita a compreensão da evolução do projeto, identificando autores, datas e descrições de mudanças, além de possibilitar a reversão a versões anteriores, se necessário.

Facilita o trabalho em equipe, pois permite que desenvolvedores trabalhem em *branch* separadas do código principal e depois mesclá-las (*merge*) ao código principal. Também auxilia na solução de problemas, rastreando as origens de erros e revertendo para versões sem defeitos. O Git é um dos sistemas de controle de versão mais populares, sendo distribuído e permitindo o trabalho independente dos desenvolvedores, sincronizando alterações posteriormente.

Apesar de o *Subversion* (SVN) e o *Concurrent Versions System* (CVS) tenham sido sistemas populares no passado e ainda sejam usados em alguns projetos, o Git se tornou amplamente adotado devido às suas vantagens em termos de descentralização, ramificação eficiente, velocidade, flexibilidade e suporte da comunidade.

# 3 METODOLOGIA

Neste capitulo será abordado as etapas antes do desenvolvimento da aplicação juntamente com tecnologias a serem utilizadas e outros levantamentos importantes para sua produção.

## 3.1 Aspectos Metodológicos

De acordo com Oliveira e Seabra (2015) as Metodologias de Desenvolvimento de Software (MDS) foram criadas para padronizar os procedimentos na criação de um produto de software. Essas metodologias tem como objetivo de: diminuir o tempo de produção, melhorar a qualidade do produto a ser concebido e reduzir o tempo de desenvolvimento e seus custos.

Segundo Sommerville (2019) seguindo o modelo incremental é possível priorizar quais as necessidades do usuário ou cliente são mais importantes para que estas possam ser desenvolvidas primeiro. E assim, a cada iteração pode-se entregar um conjunto de funcionalidades que compõem o produto incrementando aos poucos o projeto em produção e evoluindo aos poucos no produto desejado pelo cliente ou usuário.

Neste capítulo é apresentado a metodologia que será utilizada na realização deste trabalho dividindo seu desenvolvimento através de etapas. As etapas são: levantamento de requisitos, coleta de dados e escolha do *framework* a ser utilizado e sua implementação.

## 3.2 Coleta de Dados

É dado continuidade a pesquisa exploratória Wazlawick (2009) com o objetivo de entender quais os pontos fortes e fracos de outras aplicações que possuem um objetivo semelhante e que seguem um padrão de interfaces com uma boa usabilidade. O Roll20 será um dos alvos dessa pesquisa levando em conta que segue muitos dos princípios de usabilidade comentados anteriormente.

Um dos seus pontos fracos é o seu aplicativo para dispositivos móveis, o Roll20 Companion. O aplicativo não possui uma navegação intuitiva e usuários na PlayStore relatam muitos travamentos e não contempla muitas das funcionalidade existentes em sua aplicação web. A nota do aplicativo na PlayStore é de 2.3.

Em seu modo web pelos navegadores disponíveis para computadores em geral a navegação é intuitiva e de fácil utilização. A exibição de seus elementos possui um tamanho de

fonte adequado, contraste entre fonte e o seu fundo, funções auxiliares para usuários mais leigos poderem fazer suas rolagens de dados e um modo noturno para quem desejar.

## 3.3 Escolha do *framework*

Como critérios para escolher um dos três *frameworks* citados anteriormente, escolheu-se o que:

- Tenha uma grande variedade de biblioteca e *plugins* para facilitar o desenvolvimento.
- Seja consolidado no mercado em relação ao seu tempo de uso e resolução de problemas.
- Tenha uma documentação detalhada.
- Seja uma das expertises do desenvolvedor.

Levando em consideração esses pontos e as vantagens e desvantagens de cada um dos três *frameworks* citados anteriormente neste trabalho escolheremos como opção de desenvolvimento que mais se adequará, aos critérios deste projeto, é o *framework* Angular.

## 3.4 Escolha da ferramenta de versionamento de código

GitHub, GitLab e Bitbucket são plataformas de hospedagem e gerenciamento de projetos baseadas em sistemas de controle de versão distribuídos, como o Git e o Mercurial. Embora todas as três plataformas compartilhem características semelhantes, existem diferenças significativas que podem influenciar a escolha de uma delas com base nas necessidades e preferências dos desenvolvedores.

Com base na exposição das três plataformas supracitadas, a escolhida foi o GitHub. Embora o GitLab e o Bitbucket também ofereçam recursos e funcionalidades valiosos, o GitHub é frequentemente considerado a melhor escolha devido à sua popularidade, comunidade ativa, interface intuitiva, recursos robustos, integração com outras ferramentas e serviços, além do suporte abundante disponível. Esses fatores fazem do GitHub uma plataforma altamente recomendada para hospedar, colaborar e gerenciar projetos de desenvolvimento de software além de ser uma forma de exibição para os trabalhos realizados dentro da plataforma onde é geralmente solicitado até em preenchimentos de vagas de emprego.

## 3.5 Configurando o *framework*

O trabalho de implementação começará após a configuração do ambiente desenvolvimento. Para isso, será seguido a documentação de passos iniciais de acordo com o que está descrito no Angular (2010) que solicita alguns requisitos para a instalação dele como: Node.Js e o npm package manager. Com os requisitos instalados por meio do terminal no Visual Studio Code que será o editor de código utilizado no desenvolvimento dessa aplicação instalaremos o *framework* através do comando `npm install -g @angular/cli`.

Como usuário de um sistema operacional Windows segundo a documentação do Angular (2010) é necessário um passo adicional para que ele funcione corretamente e esse passo se dá ao abrir o PowerShell do Windows e permitindo a execução de *scripts* secundários. Concluindo esse segundo passo será necessário utilizar o comando `ng new nome-do-aplicativo` para criar a pasta do produto a ser desenvolvido e para iniciar o projeto localmente a linha de comando para isso é `ng` serve.

## 3.6 Implementação utilizando o *framework*

Nesta etapa vale ressaltar que as páginas vão ser criadas utilizando linguagem HTML, *Cascading Style Sheets* (CSS) e TypeScript no desenvolvimento do *Front-End* para exibição do *layout* e lógica da página. O projeto tem como uma das suas bases um *layout* responsivo que segundo Beaird (2016) é um conceito de *design* que faz com que a aplicação web possa se adaptar nos diferentes tamanhos de tela. A construção da tela será feita seguindo uma ordem de dispositivos que será do menor para o maior ou seja, o processo de desenvolvimento começará de dispositivos móveis até o tamanho de monitores dos dispositivos de mesa conhecidos como *desktops*.

# 4 RPG - FRONT-END

Este capítulo tem como objetivo, minuciosamente, abordar o processo de desenvolvimento do projeto intitulado “RPG - Front-End”, o qual, tomando como ponto de partida as informações previamente delineadas no capítulo referente à Capítulo 3 e serpa dividido nas seguintes etapas: a) levantamento de requisitos, b) delimitação de Requisitos e c) desenvolvimento da aplicação;

## 4.1 Levantamento de requisitos

Inicialmente é feito o levantamento para identificar os requisitos que serão necessários para o desenvolvimento do projeto e para futuramente separar seu desenvolvimento utilizando alguma metodologia ágil para dividir os casos de uso e determinar o escopo da implementação. Na etapa de levantamento dos requisitos iremos listar quais serão os requisitos para a aplicação e verificar como aplicar esses requisitos de forma que se tenha uma boa usabilidade e desenvolvimento.

Barbosa (2010 apud NIELSEN, 1994) definem o critério de usabilidade como um conjunto de fatores que qualificam quão bem uma pessoa pode interagir com um sistema interativo. Esses critérios estão relacionados com a facilidade e o esforço necessários para os suários aprenderem e utilizarem um sistema. Desse modo a usabilidade endereça principalmente a capacidade cognitiva, perceptiva e motora dos usuários empregada durante a interação. Os fatores de usabilidade por ele considerados são:

- Facilidade de aprendizado.
- Facilidade de recordação.
- Eficiência.
- Satisfação do usuário.

Os requisitos relacionados ao desenvolvimento, principalmente reutilização de *layouts* e integração, são:

- Criar um conjunto de *layouts* pre-definidos que podem ser alterados com o clicar de um botão.
- Possibilitar a criação de fichas com atributos definidos pelo o usuário sem a necessidade de criar *scripts*.
- Criar a tela de *login* da aplicação.

- Integração com API para *login*.
- Criar tela para listagem de sistemas.
- Integração com API para listagem de sistemas.
- Criar tela para a criação do sistema.
- Integração com API para cadastro de sistema.
- Criar tela para a criação da ficha.
- Integração com API para listagem de fichas de um usuário especifico.
- Criar *layout* responsivo.

## 4.2 Delimitação de Requisitos

Com o objetivo de entregar um protótipo mínimo aceitável foi realizado um processo de priorização e delimitação do escopo de acordo com o levantamento de requisitos realizado na seção 4.1. Com isso, foi definido quais seriam os principais pontos de desenvolvimento para a apresentação do protótipo.

Inicialmente, estabelecemos as restrições quanto aos requisitos que poderiam ser negligenciados, a fim de assegurar a natureza abrangente do projeto, levando em consideração a proposta de fornecer uma aplicação que possa auxiliar o usuário a criar sua ficha personalizada de RPG.

Com isso, os requisitos de página de *login*, seleção de sistemas listagem de fichas do usuário e integrações de APIs no geral foram desconsideradas no protótipo mínimo aceitável.

A priorização dos requisitos foi direcionada a tela de criar sistema juntamente com sua a aplicação de sua lógica atrelado com um *layout* responsivo substituindo as comunicações com as integrações de APIs por um armazenamento no `localstorage`.

## 4.3 Desenvolvimento da aplicação

Determinado as prioridades e as tecnologias a serem utilizadas demos inicio ao desenvolvimento da aplicação que foi dividida em:

a) sistema de rotas e definição de nome dos componentes;
b) criação dos serviços necessários;
c) desenvolvimento da página de criar sistema; e
d) desenvolvimento da página de criar ficha;

### *4.3.1 Sistema de rotas e definição de nome dos componentes*

O primeiro passo na criação do projeto foi definir o sistema de rotas da aplicação juntamente com os nomes das páginas.

Nesse projeto Angular, o sistema de rotas é configurado usando o módulo `RouterModule` do pacote `@angular/router`. Esse módulo é importado juntamente com o `Routes`, que é uma interface do Angular para definir as rotas.

O `routes` é um *array* de objetos que define as rotas disponíveis na aplicação. Cada objeto no *array* representa uma rota e contém informações como o caminho e o módulo a ser carregado quando a rota for acessada.

A primeira rota tem um caminho vazio (`path: ''`) e redireciona para a rota 'criar-sistema' usando o `redirectTo`. O `pathMatch: 'full'` indica que a rota vazia só será redirecionada se a URL corresponder exatamente a um caminho vazio. Isso é feito Para caso o usuário digite uma rota inexistente possa retornar para a primeira rota do inicio da aplicação

A segunda rota tem o caminho `criar-sistema` e utiliza o `loadChildren` para carregar o módulo `CriarSistemaModule` de forma assíncrona. Isso significa que o módulo será carregado sob demanda quando essa rota for acessada.

A terceira rota tem o caminho `criar-ficha` e também utiliza o `loadChildren` para carregar o módulo `CriarFichaModule` de forma assíncrona.

Por fim, o módulo `AppRoutingModule` é definido como um módulo Angular usando o `@NgModule`. Ele importa o `RouterModule` com as rotas definidas no *array* `routes` usando `forRoot(routes)`. O método `forRoot` é usado quando as rotas são configuradas no módulo raiz da aplicação. O `AppRoutingModule` também exporta o `RouterModule`, permitindo que outros módulos da aplicação possam importar o sistema de rotas.

Com essa configuração, quando a aplicação Angular for iniciada e as rotas forem acessadas no navegador, o Angular irá corresponder a URL com os caminhos definidos e carregar os respectivos módulos de forma assíncrona, se necessário.

### *4.3.2 Criação dos serviços necessários*

Uma etapa importante que será utilizada como forma de armazenamento de dados na aplicação é o *service*.Para o projeto criamos o LocalStorageService. A função principal desse serviço é fornecer métodos para salvar, obter e remover objetos do armazenamento local do

navegador usando o *localStorage* e torna-lo reutilizável para qualquer página do projeto. e os métodos criados são:

- `saveObject(key: string, object: any): void`: Este método recebe uma chave (*key*) e um objeto (*object*). Ele converte o objeto em uma string *JavaScript Object Notation* (JSON) usando `JSON.stringify` e, em seguida, armazena a string JSON no armazenamento local com a chave fornecida usando `localStorage.setItem`. A implementação pode ser vista no Código-fonte 1.

Código-fonte 1 – `saveObject`

```
1       public saveObject(key: string, object: any): void
            {
2         const objectString = JSON.stringify(object);
3         localStorage.setItem(key, objectString);
4       }
```

- `getObject(key: string): any`: Este método recebe uma chave (*key*) e retorna o objeto associado a essa chave no armazenamento local. Ele recupera a string JSON armazenada usando `localStorage.getItem` com base na chave fornecida e, em seguida, a converte de volta para um objeto usando `JSON.parse`. Se a chave não existir ou se ocorrer algum erro na conversão, o método retorna `null`. A implementação pode ser vista no Código-fonte 2.

Código-fonte 2 – `getObject`.

```
1       public getObject(key: string): any {
2         const objectString = localStorage.getItem(key);
3         if (objectString) {
4           return JSON.parse(objectString);
5         }
6         return null;
7       }
```

- `removeObject(key: string): void`: Este método recebe uma chave (*key*) e remove o objeto associado a essa chave do armazenamento local usando `localStorage.removeItem`.

A implementação pode ser vista no Código-fonte 3.

Código-fonte 3 – `removeObject`.

```
1       public removeObject(key: string): void {
2         localStorage.removeItem(key);
3       }
```

O serviço `LocalStorageService` é projetado para facilitar a manipulação de dados no armazenamento local do navegador. Ele encapsula a lógica necessária para serializar e desserializar objetos JSON e interagir com a API `localStorage`. Ao usar esse serviço, os componentes e serviços podem salvar, recuperar e remover objetos do armazenamento local de forma fácil e consistente.

### *4.3.3 Desenvolvimento da página de criar sistema*

A primeira página da aplicação é a página de criar sistema que tem o objetivo de permitir ao usuário a criação de uma ficha dinâmica que atenda as necessidades do sistema que será criado. As funcionalidades da página funcionam da seguinte forma:

- O formulário é definido usando a diretiva `[formGroup]="form"`, que vincula o formulário ao `FormGroup` chamado `form`. Dentro desse `FormGroup` existem três seções que é a de cabeçalho, de sessões e de cálculos.
- O cabeçalho do formulário contém os campos `nomeSistema`, `linkSistema` e `criadorSistema`, que são controlados pelo `FormGroup` `cabecalho`.
- Cada campo de entrada usa a diretiva `formControlName` para vincular os valores aos respectivos controles do `FormGroup`.
- Para cada campo de entrada, há uma *tag* `<small>` com a classe “error-message” que exibe a mensagem “Campo Obrigatório” caso o campo seja inválido.
- O formulário possui uma seção chamada “sessoes” que é um `FormArray`. Essa seção permite adicionar várias sessões ao sistema. A primeira sessão é definida inicialmente, mas é possível adicionar mais sessões clicando no botão “Adicionar nova seção”. O botão “-” permite remover uma sessão adicional, exceto a primeira.
- Dentro de cada sessão, há um campo de entrada para o nome da sessão, controlado pelo `FormGroup` da sessão atual.

- Abaixo do campo de nome da sessão, há um vetor de campos de entrada para os atributos da sessão. Os campos são controlados por outro `FormArray` chamado “atributos”. É possível adicionar mais campos de atributos clicando no botão “Adicionar novo atributo” e remover campos adicionais clicando no botão “-”.
- Após a seção de sessões, há uma seção chamada `calculos` que também é um `FormArray`. Essa seção permite adicionar várias campos que vão ser utilizados para calcular campos que de acordo com o manual criado representem uma função na ficha como por exemplo o valor atribuído a vida no *Dungeon and Dragons* (WIZARDS OF THE COAST LLC, 2022) que se joga um dado e soma um dos atributos a esse dado e assim é obtido o valor de vida. É possível adicionar mais espaços de campos desse tipo clicando no botão “Adicionar novo atributo” e remover adicionais clicando no botão “-”.
- O formulário possui um botão “Salvar” que, quando clicado, chama a função `submit()`. Se o formulário for válido, ele salva os dados do formulário em um serviço chamado `localStorageService` e redireciona para a página `criar-ficha`.
- O componente também possui várias funções auxiliares, como `controlFormArraySessoes`, `controlFormArrayCalculos`, `controlFormArrayAtributos`, `addSection`, `addCalcFormContro` `removeSection`, `newSectionFor`, `newCalcForm`, `addFormControlAtributos`, `removeFormContr` `isFormControlInvalid`, `displayErrorMessage`, `redirect`, entre outras, que são usadas para manipular o formulário e realizar validações.

Em resumo, o componente representa um formulário de criação de sistema com a capacidade de adicionar sessões e fórmulas de cálculo dinamicamente. Ele valida os campos obrigatórios e salva os dados do formulário quando o botão “Salvar” é clicado. O Código-fonte **??** apresenta mais detalhes em relação a lógica da página. ver Apêndice **??** O resultado da página em uma aplicação *mobile* pode ser visualizado nas Figuras 1 , 2 e 3.

O resultado da página em uma aplicação *desktop* pode ser visualizado nas Figuras 4 e 5.

### *4.3.4 Desenvolvimento da página de criar ficha*

A próxima página é a de criar ficha que é um formulário gerado de acordo com a ficha desenvolvida no momento da criação do sistema junto com seu nome, *link* e nome do criador. A página funciona da seguinte forma:

- Estrutura HTML: consiste em uma `<div>` principal com a classe `page-criar-ficha`.

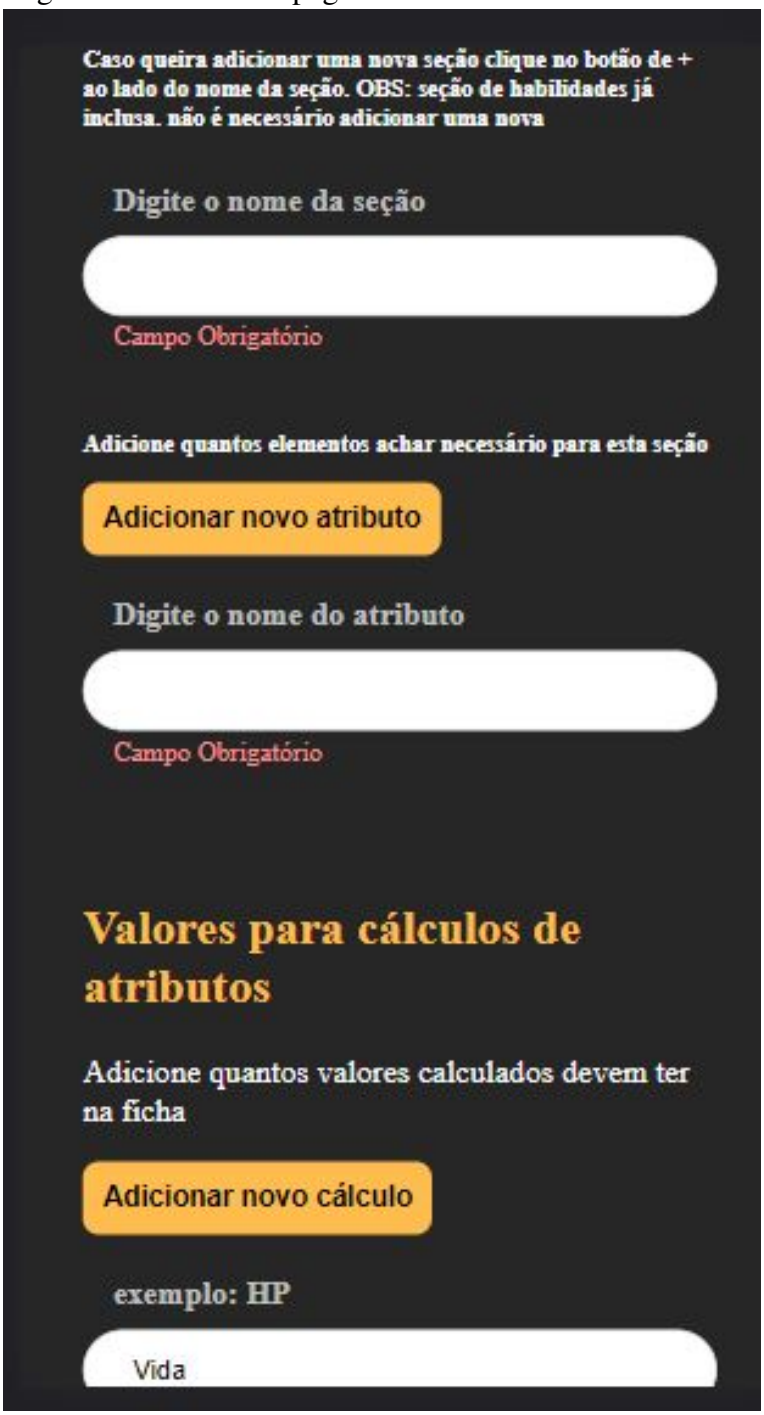
Figura 1 – Parte 1 da página na visão *mobile*.


Fonte: Elaboração Própria.

Dentro dessa `<div>`, há um formulário representado pelo elemento `<form>`. O formulário é composto por várias seções, cada uma com sua própria estrutura de elementos HTML.

- Cabeçalho: A primeira seção do formulário é representada pelo elemento `<section>` com a classe `input-wrapper ficha-header`. Essa seção contém campos de entrada relacionados ao cabeçalho da ficha de personagem, como nome do jogador, nome do personagem, nome do sistema, *link* do sistema, criador do sistema, alinhamento e idade.

Figura 2 – Parte 2 da página na visão *mobile*.

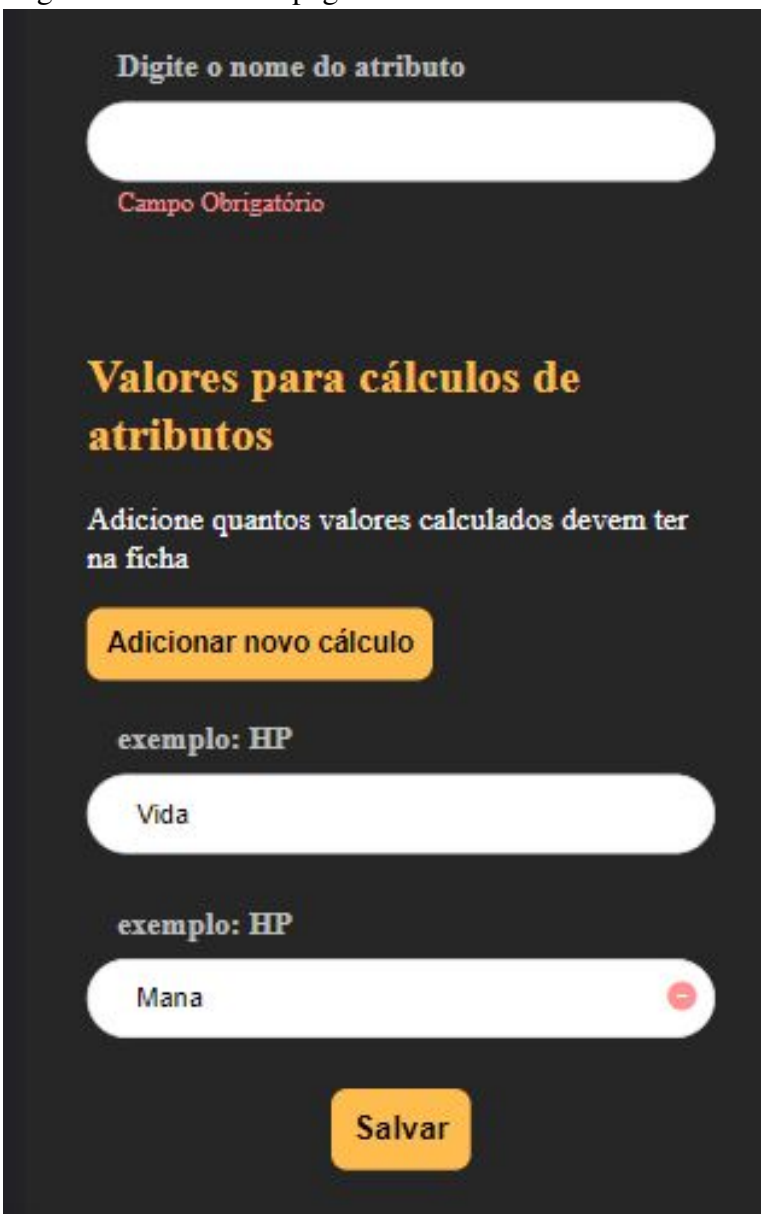



Fonte: Elaboração Própria.

Cada campo de entrada é representado pelo elemento `<input>`, com seus respectivos rótulos representados pelo elemento `<label>`. Além disso, há também elementos `<small>` que exibem mensagens de erro caso algum campo seja inválido.

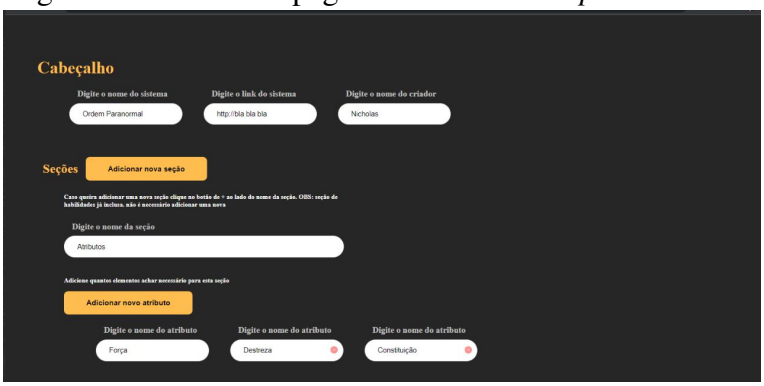
- Seções e atributos: A segunda seção do formulário é representada pelo elemento `<section>` com a classe `sections`. Essa seção contém uma lista de seções adicionais, cada uma representada por um elemento `<div>` com a classe `section-item`. Cada seção possui

Figura 3 – Parte 3 da página na visão *mobile*.



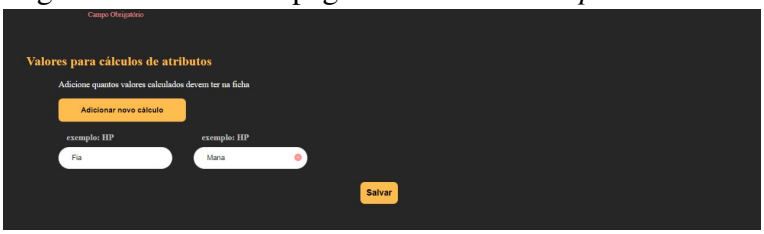

Fonte: Elaboração Própria.

Figura 4 – Parte 1 da página na visão *desktop*.




Fonte: Elaboração Própria.

Figura 5 – Parte 2 da página na visão *desktop*.




Fonte: Elaboração Própria.

um cabeçalho representado pelo elemento `<h2>`, que exibe o nome da seção com base em seu índice. Dentro de cada seção, há uma lista de atributos representada pelo elemento `<div>` com a classe `input-wrapper`. Cada atributo é representado por um elemento `<div>` com a classe `input-container`. Assim como no cabeçalho, cada atributo possui um rótulo representado pelo elemento `<label>`, e há também elementos `<small>` para exibir mensagens de erro.

- Cálculos: A terceira seção do formulário é representada pelo elemento `<section>` com a classe `calculos`. Essa seção contém uma lista de campos de entrada para valores de cálculos de atributos. Assim como nas seções anteriores, cada campo de entrada é representado pelo elemento `<input>`, com seus respectivos rótulos representados pelo elemento `<label>`. Também há elementos `<small>` para exibir mensagens de erro.
- Habilidades: A quarta seção do formulário é representada novamente pelo elemento `<section>` com a classe `calculos`. Essa seção é específica para a entrada de habilidades do personagem. Há um cabeçalho representado pelo elemento `<h2>`, seguido por um parágrafo de instruções e um botão “Adicionar” para adicionar novas habilidades. Cada habilidade é representada por um elemento `<div>` com a classe `input-container`. Cada habilidade possui um rótulo representado pelo elemento `<label>`, seguido por um campo de entrada de texto representado pelo elemento `<input>` para o nome da habilidade e um campo de entrada de texto multilinha (`textarea`) para a descrição da habilidade. Também há elementos `<small>` para exibir mensagens de erro. Além do botão de para “Adicionar” ao lado de cada uma das habilidades geradas dinamicamente a cada clique do “Adicionar” quando o *array* de lista das habilidades for maior que uma habilidade um botão para remoção de habilidades vai aparecer na tela. Diferente da tela anterior, essa tela possui a sessão de habilidades pelo o fato de que é comum na maioria dos RPGs ter uma etapa para a listagem de habilidades.
- Botão de salvar: Fora do formulário, há um elemento `<div>` com a classe `button-container`

que contém um botão “Salvar” representado pelo elemento <button>.A estrutura HTML foi projetada de forma a organizar os campos de entrada e exibir mensagens de erro quando necessário. Os elementos <formGroupName>, <formArrayName> e <formControlName> são diretivas do Angular que permitem associar os campos de entrada aos controles do formulário definidos no componente TypeScript.

Em relação ao código do componente em TypeScript da página temos:

- Importações: O componente importa várias classes e serviços do Angular necessários para sua funcionalidade, como Component, OnInit, FormBuilder, FormGroup, Validators, ActivatedRoute, Router e LocalStorageService. Essas importações permitem o uso de recursos do Angular, como criação de componentes, validação de formulários e navegação entre rotas.
- Classe do componente: O componente CriarFichaComponent é definido com o decorador @Component e implementa a interface OnInit. Isso significa que ele é um componente Angular e deve implementar o método ngOnInit.
- Propriedades: O componente possui as seguintes propriedades:
  - form: Representa o FormGroup que contém todos os controles do formulário.
  - sessoes: Armazena as sessões obtidas do serviço LocalStorageService.
  - calculos: Armazena os cálculos obtidos do serviço LocalStorageService.
- Método ngOnInit(): O método ngOnInit é executado quando o componente é inicializado. Ele é responsável por criar e configurar o FormGroup form, definindo os controles e validadores para cada campo de entrada.
- Criação do FormGroup: O FormGroup form é criado usando o FormBuilder. Ele possui os seguintes controles:
  - cabecalho: Um FormGroup que contém os controles relacionados ao cabeçalho da ficha.
  - sessoes: Um FormArray vazio que será preenchido posteriormente de acordo com os dados recebidos da página anterior.
  - calculos: Um FormArray vazio que será preenchido posteriormente de acordo com os dados recebidos da página anterior.
  - habilidades: Um FormArray com um FormGroup inicial contendo os controles para a primeira habilidade.
- Preenchimento dos valores iniciais: O método patchValue é usado para preencher os valo-

res iniciais do formulário com base nos dados obtidos do serviço `LocalStorageService`. Os valores são copiados para o `FormGroup` `form` usando o método `getObject`.

- Preenchimento das seções e atributos: O componente itera sobre as sessões obtidas e adiciona os controles de atributos correspondentes. Para cada sessão, é chamado o método `controlFormArraySessoes.push` para adicionar um novo `FormGroup` ao `FormArray` `sessoes`. Em seguida, o método `addFormControlAtributos` é chamado para adicionar os controles de atributos correspondentes à sessão. Isso é feito chamando o método `controlFormArrayAtributos` com o índice da sessão.
- Preenchimento dos cálculos: O componente itera sobre os cálculos obtidos e adiciona os controles correspondentes ao `FormArray` `calculos`. É chamado o método `controlFormArrayCalculos.push` para adicionar um novo `FormControl` ao `FormArray` `calculos`.

A estrutura HTML define a aparência e a estrutura do formulário, enquanto o componente TypeScript fornece a lógica para manipular e validar os dados do formulário. Ao preencher o formulário e clicar no botão “Salvar”, a lógica adicional pode ser implementada para enviar os dados para um servidor ou realizar outras ações necessárias. O Código-fonte **??** apresenta isso de forma mais detalhada e técnica.

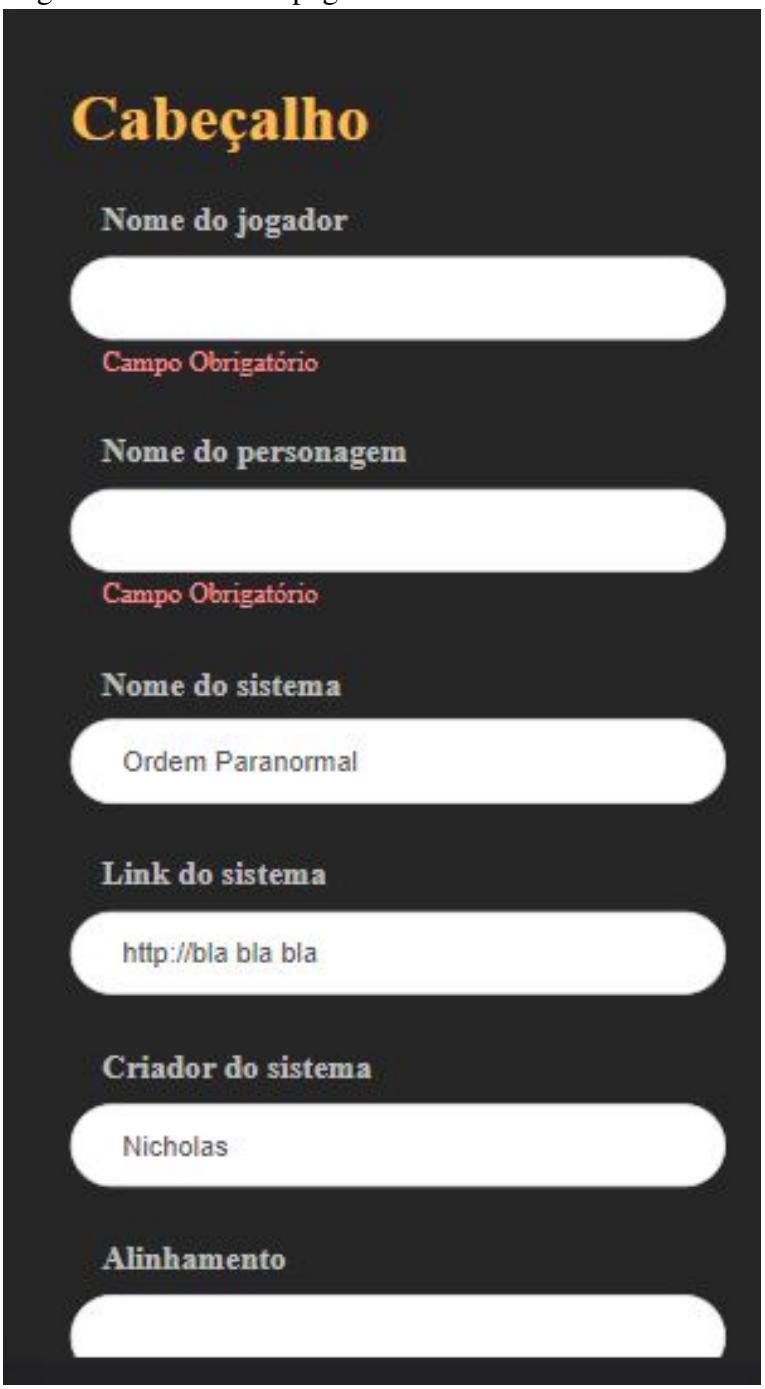
ver Apêndice **??**

O resultado da página em uma aplicação *mobile* pode ser visualizado nas Figuras 6 , 7, 8 e 9.

O resultado da página em uma aplicação *desktop* pode ser visualizado nas Figuras 10 e 11.

Portanto, com as páginas criadas acima é possível criar um sistema através de formulários dinâmicos e criar uma ficha para preenchimento do usuário. Além disso, o versionamento do projeto foi utilizado no GitHub para rastreabilidade de *commits* e para facilitar para outros desenvolvedores interessados poderem atuar também. O link para o repositório se encontra aqui: https://github.com/nicholastre/tcc

Figura 6 – Parte 1 da página na visão *mobile*.




Fonte: Elaboração Própria.

Figura 7 – Parte 2 da página na visão *mobile*.


Fonte: Elaboração Própria.

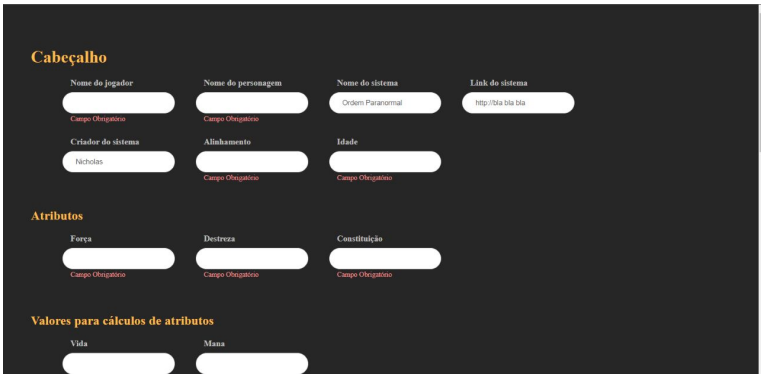
Figura 8 – Parte 3 da página na visão *mobile*.


Fonte: Elaboração Própria.

Figura 9 – Parte 4 da página na visão *mobile*.


Fonte: Elaboração Própria.

Figura 10 – Parte 1 da página na visão *desktop*.




Fonte: Elaboração Própria.

Figura 11 – Parte 2 da página na visão *desktop*.


Fonte: Elaboração Própria.

# 5 CONCLUSÃO

Este trabalho teve início como um modesto empreendimento pessoal motivado pela necessidade de narrar uma partida de RPG de um sistema brasileiro recém criado, que não contava com ferramentas de organização de fichas disponíveis em outros softwares. Os softwares existentes que permitiam a criação de uma ficha primeiramente eram pagos e por fim necessitavam de conhecimento de linguagens de programação para a criação da ficha.

Observando o esforço colaborativo de diversos membros da comunidade de jogadores de Ordem Paranormal RPG na criação de aplicativos específicos para esse sistema, surgiu a ideia de desenvolver uma proposta de aplicação *front-end* que por meio de instruções fosse possível criar uma ficha personalizada de quaisquer sistemas existentes sem a necessidade de conhecimentos específicos de programação, após meses de discussões e sugestões sobre possíveis soluções. O objetivo principal era a criação dessa aplicação que pudesse ser continuada e melhorada por diversos desenvolvedores com a proposta de entregar a usuários especialistas um produto gratuito e funcional.

O processo de desenvolvimento foi essencial primeiramente no estudo de outros *frameworks* de desenvolvimento existentes que levaram a escolha do Angular para a aplicação deste projeto. Além de conhecer um pouco melhor sobre as arquiteturas do React e do Vue.js também foi perceptível o valor agregado em conhecimento em relação ao tipo de modelo usado no desenvolvimento utilizando Angular, as diferentes formas de aplicação dos formulários que o *framework* disponibiliza. Como nomenclatura de funções foi adotado o Wikipédia (2023) e para como boa prática de desenvolvimento foi criado um *service* que permite adições e remoções que simularam a aplicação de APIs no `localStorage`.

Dentre as dificuldades do Angular a principal encontrada foi a criação de um componente reutilizável para a aplicação dos campos de dados do usuário na página na qual a componentização não foi implementada neste sentido no projeto citado. Apesar das dificuldades, o resultado foi um projeto estruturado, escalável, inovador e com nome de funções e classes intuitivas já seria um produto *open-source*.

O projeto possui limitações quanto a sua atuação, mas é um produto com potencial e que tem margem para vários outros trabalhos futuros como integração com API, páginas e integração com API de cadastro de usuários salas para jogos de RPGs.

# REFERÊNCIAS

ANGULAR. **Angular documentation**. 2010. Disponível em: <https://angular.io/docs>. Acessado em: 15 de novembro de 2022.

BARBOSA, S. D. J. **Interação Humano-Computador**. [S.l.]: Elsevier, 2010.

BASSANI, P. B. S. Produção de jogos interativos: um espaço de construção hipertextual coletiva. **Revista Tecnologia e Tendência**, v. 1, n. 2, p. 45–50, jul./dez. 2002. ISSN 2357-8610. Disponível em: <https://periodicos.feevale.br/seer/index.php/revistatecnologiaetendencias/article/view/1374>.

BEAIRD, J. **Princípios Do Web Design Maravilhoso**. 3a.. ed. [S.l.]: Alta Books, 2016. 220 p. ISBN 978-8576089827.

FIRECAST, R. **RPG FireCast**. 2022. Disponível em: <https://firecast.app/pt_br/>. Acesso em: 06 de julho de 2023.

HUIZINGA, J. **Homo ludens: O jogo como elemento da cultura**. 1a.. ed. [S.l.]: Perspectiva, 2019. 304 p. ISBN 978-8527311571.

KAUSHALYA, T.; PERERA, I. Framework to migrate angularjs based legacy web application to react component architecture. In: **2021 Moratuwa Engineering Research Conference (MERCon)**. Moratuwa, Sri Lanka: IEEE, 2021. p. 693–698. ISSN 2691-364X. Disponível em: <https://ieeexplore.ieee.org/document/9525659>.

NIELSEN, J. **Interação Humano-Computador**. [S.l.]: Morgan Kaufmann Publisher, 1994.

NOVAC, O. C.; MADAR, D. E.; NOVAC, C. M.; BUJDOSó, G.; OPROESCU, M.; GAL, T. Comparative study of some applications made in the angular and vue.js frameworks. In: **2021 16th International Conference on Engineering of Modern Electric Systems (EMES)**. Oradea, Romania: IEEE, 2021. p. 1–4. Disponível em: <https://ieeexplore.ieee.org/document/9525659>.

OLIVEIRA, F. G.; SEABRA, J. M. P. METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE: UMA ANÁLISE NO DESENVOLVIMENTO DE SISTEMAS NA WEB. **Periódico Científico Tecnologias em Projeção**, v. 6, n. 1, p. 20–34, 2015. ISSN 2178-6267.

PAVÃO, A. A AVENTURA DA LEITURA E DA ESCRITA ENTRE MESTRES DE Roleplaying Games (RPG). **ANPED**, p. 1–16, 1999. Disponível em: <https://anped.org.br/sites/default/files/gt_10_02.pdf>.

PEREIRA, A. L. K.; RODRIGUES, A. R.; AMARAL, E. C.; SABINO, E.; MUNIZ, M. S. d. A.; ABE, N. FRAMEWORKS PARA DESENVOLVIMENTOS DE JOGOS: UMA ABORDAGEM VANTAJOSA NO DESENVOLVIMENTO DE JOGOS ELETRÔNICOS. **Revista Gestão em Foco**, v. 9, p. 575–586, 2017. Disponível em: <https://portal.unisepe.com.br/unifia/wp-content/uploads/sites/10001/2018/06/058_frameworks.pdf>.

ROLL20. **Roll20**. 2022. Disponível em: <https://roll20.net/welcome>. Acesso em: 06 de junho de 2023.

SCHUYTEMA, P. **Design de games: Uma abordagem prática**. 1a.. ed. [S.l.]: Cengage Learning, 2008. 472 p. ISBN 978-8522106158.

SOMMERVILLE, I. **Engenharia De Software**. 10a. ed. [S.l.]: Pearson Universidades, 2019. ISBN 9788543024974.

TOLKIEN, C. **The War of the Jewels**. [S.l.]: HarperCollins Publishers Ltd, 1995. 496 p. ISBN 978-0261103245.

WAZLAWICK, R. S. **Metodologia de Pesquisa para Ciência da Computação**. 1a. ed. Rio de Janeiro: Elsevier Editora, 2009. ISBN 978-85-352-3522-7.

WICKRAMASINGHE, S. **Vue vs React: Qual Você Deve Usar?** 2022. Disponível em: <https://kinsta.com/pt/blog/vue-vs-react/>. Acessado em: 14 de novembro de 2022.

WIKIPÉDIA. **CamelCase**. 2023. Disponível em: <https://pt.wikipedia.org/wiki/CamelCase>. Acesso em: 06 de julho de 2023.

WIZARDS OF THE COAST LLC. **Dungeon and Dragons**. 2022. Disponível em: <https://dnd.wizards.com/pt-BR/>. Acesso em: 06 de julho de 2023.