Micael Machado Gomes

Playground Web para Aplicações Hipermídia Não-Lineares

São Luís

2022

Micael Machado Gomes

Playground Web para Aplicações Hipermídia Não-Lineares

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Curso de Ciência da Computação

Universidade Federal do Maranhão

Orientador: Prof. Dr. Carlos de Salles Soares Neto

São Luís - MA

2022

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Gomes, Micael Machado.
Playground web para aplicações hipermídia não-lineares / Micael Machado Gomes. - 2022.
64 f.

Orientador(a): Carlos de Salles Soares Neto.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, Online, 2022.

1. Ferramenta de autoria. 2. Modelo hipermídia. 3. Pesamento computacional. 4. Playground. I. Soares Neto, Carlos de Salles. II. Título.

Micael Machado Gomes

Playground Web para Aplicações Hipermídia Não-Lineares

Trabalho de conclusão de curso apresentado ao curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de bacharel em Ciência da Computação.

Trabalho aprovado em 25 de janeiro, São Luís, 2022:

Prof. Dr. Carlos de Salles Soares Neto
Orientador
Universidade Federal do Maranhão

Prof. Dr. Tiago Bonini Borchartt
Examinador
Universidade Federal do Maranhão

Prof. Me. Hedvan Fernandes Pinto
Examinador
Instituto Federal de Educação, Ciência e Tecnologia do Amazonas

---

Me. Daniel de Sousa Moraes
Examinador
Pesquisador TeleMídia/PUC-Rio

São Luís
2022

À minha família e meus amigos.

Agradecimentos

A minha querida mãe, Elenice, por sempre ter me incentivado e me apoiado em tudo.

A minha querida tia Nadir, que me acolheu como um filho em sua casa.

Ao meu irmão, Raphael Gomes, por ter sido meu mentor e amigo em tantas batalhas.

Ao meu tio Sebastião, que me estendeu a mão em momentos de dificuldade.

Aos meus quatro queridos irmãos, que estão ao meu lado desde de o início de suas vidas.

A toda minha família por sempre estarem torcendo por mim, em especial Rafaelle Gomes por ter compartilhado seu notebook comigo durante o 1º ano de curso.

A Kézia Shadai, minha companheira, por ter me dado forças e o suporte para continuar.

Ao meu pai, Antônio Miguel, por ter me ajudado no início da minha vida em São Luís.

Ao meu orientador, Carlos Salles, pelo seu comprometimento em me ajudar neste desafio.

Aos meus amigos Mateus Pinheiro, Yandson Costa, Ednara Pereira, Jéssica Costa, Paulo Ricardo, Dayvson Almeida e Pedro Thiago que fiz durante a minha graduação.

Ao corpo docente do curso de Ciência da Computação, pela qualidade demonstrada ao longo dos anos.

E a todos que me ajudaram durante esta jornada.

"Nada pode ser obtido sem uma espécie de sacrifício. É preciso oferecer algo de valor equivalente."

(Edward Elric)

Resumo

Playgrounds Web ganharam muito destaque nos últimos anos, principalmente pela praticidade em que um usuário pode construir um projeto sob uma determinada tecnologia. Eles não precisam ser instalados ou configurados em ambiente local, como uma IDE ou um compilador de uma determinada linguagem de programação. Neste trabalho, são discutidos os requisitos e especificações técnicas com base em ferramentas similares, para conceber um sistema de acordo com arquitetura serverless, permitindo a criação de recursos interativos. Esta modelagem teve como resultado o Codefab, um Playground voltado para construção de narrativas hipermídia não-lineares, estruturadas com conceitos do modelo Fábulas. Para realizar a avaliação da ferramenta foi feito um ensaio de interação seguido da aplicação de um formulário pesquisa, para analisar aspectos quantitativos e qualitativos. O estudo fez a utilização da ferramenta TAM para compreender o nível de aceitação da tecnologia proposta por parte dos participantes. Para uma abordagem prospectiva o SUS foi a melhor escolha, atribuindo um score para classificar a usabilidade. Os dados coletados mostraram que a ferramenta não é tão simples de ser utilizada em um primeiro momento. No entanto, mostraram um bom nível de aceite pelos participantes, com a usabilidade percebida classificada como ranking B.

Palavras-chaves: Playground, ferramenta de autoria, pensamento computacional, modelo hipermídia.

Abstract

Web playgrounds have gained a lot of attention in the last few years, mainly due to the practicality in which a user can build a project under a certain technology. They do not need to be installed or set up in a local environment, such as an IDE or a compiler for a particular programming language. In this work, requirements and technical specifications are discussed, based on similar tools, to design a system according to a serverless architecture, allowing the creation of interactive resources. This modeling resulted in Codefab, a Playground focused on building non-linear hypermedia narratives, structured with concepts from the Fábulas model. To carry out the evaluation of the tool, an interaction test was carried out followed by the application of a survey form, to analyze quantitative and qualitative aspects. The study used the TAM tool to understand the level of acceptance of the technology proposed by the participants. For a prospective approach, SUS was the best choice, assigning a score to classify usability. The data collected showed that the tool is not so simple to use at first. However, they showed a good level of acceptance by the participants, with the perceived usability classified as ranking B.

Keywords: Playground, authoring tool, computational thinking, hypermedia model.

Lista de ilustrações

Lista de listagens

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
AVA	Ambiente Virtual de Aprendizagem
BaaS	Backend as a Service
CJS	Common Javascript
CSS	Cascading Style Sheets
CSS-in-JS	Cascading Style Sheets in Javascript
DOM	Document Object Model
DSL	Domain-specific language
EaD	Ensino a Distância
ECA	Evento-Condição-Ação
FaaS	Function as a Service
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JS	Javascript
PaaS	Platform as a Service
REST	Representational State Transfer
SMIL	Synchronized Multimedia Integration Language
SPA	Single Page Application
SUS	System usability scale
TAM	Technology Access Measurement
UI	User Interface
UUID	Universally Unique Identifier
UX	User Experience
XML	eXtensible Markup Language

Sumário

1 Introdução

Nos últimos anos surgiram vários *Playgrounds Web* para escrita de código. Isso possibilitou que usuários de qualquer nível pudessem executar seus *scripts* e até mesmo projetos, sem a necessidade de instalar e configurar uma IDE (QUEIRÓS; PINTO; TERROSO, 2021). Como instrumento de aprendizagem, se mostrou uma abordagem bastante relevante, sendo possível diminuir as dificuldades no ensino de programação à distância. Além disso, a crise sanitária vivida em 2020 até o presente momento (EBC, 2021), potencializou a adesão em massa do EaD[1], aumentando os obstáculos no processo de acompanhamento dos alunos e dando mais destaque aos *Playgrounds Web*.

Diante disso, existem algumas opções de *Playgrounds Web* para serem aplicadas neste contexto, justamente para diminuir os obstáculos na aprendizagem de uma tecnologia ou uma linguagem de programação. O Replit (REPLIT, 2021) e *Codding Ground* (CODINGGROUND, 2021) são boas formas para exemplificar a aplicação dessas ferramentas, já que ambos possuem suporte para mais de cinquenta compiladores e interpretadores de código, sem contar os diversos plugins, hospedagem em nuvem e integrações com plataformas de versionamento de código. Além disso, possuem suporte a comandos de terminal, e um *mini-browser* embutido no layout, tudo isso para que seja possível acompanhar o que está sendo desenvolvido.

De modo similar, existem ferramentas que optam por uma abordagem mais lúdica e dinâmica do ensino, pensada para aprendizes com pouco, ou nenhum conhecimento em programação. O objetivo desse tipo de abordagem é facilitar a criação de aplicações hipermídia, e assim ensinar fundamentos importantes para o desenvolvimento do aluno. A plataforma Scratch (SCRATCH, 2021) explora mecanismos interativos para esse tipo de aprendizado, como *Drag and Drop*, blocos com sentenças condicionais e um *Preview*[2] para o projeto. O Code Combat segue na mesma linha (CODECOMBAT, 2021), porém com uma abordagem focada na construção de games, utilizando isso para estimular o aprendizado dos alunos.

Tais ferramentas atuam para simplificar e diminuir os ruídos no processo de aprendizado do aluno. Além disso, elas são de grande valor para fomentar o pensamento computacional, como por exemplo o *Code.org* (CODE.ORG, 2021), que estimula esse desenvolvimento em aprendizes de programação durante sua formação básica (FREITAS; MORAIS, 2019). Para definir Pensamento Computacional neste contexto, podemos adotar a vertente criada por Brackmann (2017), onde segundo ele, é a capacidade criativa, crítica e estratégica, fruto do pensamento humano, aplicando os conceitos da Ciência da Computação para identificar e resolver problemas, individualmente ou colaborativamente. O autor ainda aborda os quatro pilares do Pensamento Computacional, nos eixos de decomposição, abstração, identificação de padrões e algoritmos (BRACKMANN, 2017). Seguindo esta linha, podemos observar a relação existente entre os aspectos citados e as habilidades desenvolvidas durante

[1] Ensino à Distância: ensino de forma remota através de ambiente virtual de aprendizagem (AVA).
[2] termo em inglês par pré-visualização de alguma apresentação.

a utilização dos *Playgrounds*.

A decomposição no pensamento computacional é análogo à ideia empírica de decompor um objeto em unidades menores. É o equivalente de resolver uma tarefa ou construir um sistema com alto grau de complexidade, separando-o em fragmentos menores (WING, 2006) (BRACKMANN, 2017). *Playgrounds Web* aplicam essa ideia na organização dos projetos, como no *Code.org* (CODE.ORG, 2021), que fornecem ao autor a possibilidade de dividir e adicionar uma *tag* em partes menores para facilitar a visualização do que está sendo construído.

Caminhando ao lado do processo de decomposição do problema, está a abstração isolando características não pertinentes ao problema em maior ou menor grau, para que se possar compreender a ideia do todo. De forma prática, se refere a filtragem e classificação dos elementos essenciais, ignorando o que não é necessário para representar o que está se buscando resolver (BRACKMANN, 2017). Os blocos de sentenças condicionais, e a mecânica de quebra-cabeça para construir as aplicações das plataformas já citadas, são abstrações para estruturação do fluxo de execução de um programa com representação para *loops* e desvios condicionais.

Identificar padrões é muito importante para mapear o que deve ser abstraído de um elemento, e também como dividir um problema em partes menores, já que esses aspectos do pensamento computacional possuem traços homogêneos que podem ser analisados para que sejam resolvidos de forma mais eficaz (BRACKMANN, 2017). No Scratch (SCRATCH, 2021), isso é visto na criação de escopos menores, reconhecendo os blocos que pertencem ao mesmo contexto.

Algoritmos são conjuntos de instruções que seguem um tipo de ordenação para que seu objetivo seja atingido, sendo representado tanto por diagramas quanto por linguagem humana (BRACKMANN, 2017). Este é um aspecto muito importante para as ferramentas que trabalham com uma abordagem mais lúdica do aprendizado, já que sintetiza os pilares citados anteriormente. Ou seja, é o resultado da união das abstrações, da forma que o problema foi quebrado em partes menores, e da identificação das similaridades para resolver o problema com mais eficiência.

O modelo Fábulas apresenta os principiais elementos que uma narrativa contada através de um *e-book* deve possuir, sendo acessível para autores sem muito conhecimento em programação (PINTO et al., 2017). Assim como alguns *Playgrounds* citados, o modelo em questão visa permitir a produção de conteúdo multimídia envolvendo os usuários em experiências mais ricas. Os eixos do pensamento computacional apresentados e as entidades contempladas no Fábulas, convergem em vários aspectos. Por exemplo: a noção de agentes e estados contribuem para estimular a aplicação dos pilares de abstração e identificação de padrões, e a divisão da história não linear em páginas, fomenta os conceitos de decomposição e algoritmos.

Os elementos do Fábulas possuem suporte para linguagens de marcação via *tags* específicas, onde visualmente as características de cada entidade são apresentadas, e algumas delas servem para manipular objetos de mídia (imagens, sons e vídeos). O conjunto de regras para essa aplicação é definido pela linguagem de domínio específico (DSL) que aplica os conceitos do modelo Fábulas. A qualidade resultante da união dos objetos de mídia comum e os conceitos do Fábulas (embora a

qualidade individual de cada mídia seja relevante neste processo) depende de como a composição de todos os aspectos é harmoniosa, para que o resultado apresente uma experiência de qualidade (PELLAN; CONCOLATO, 2009).

Diante do contexto estabelecido, o presente trabalho se propõe a demonstrar os aspectos de construção de um *Playground* nos moldes das plataformas que foram citadas. Uma ferramenta de autoria multimídia, com uma interface inovadora, capaz de reproduzir uma história a medida em que é feita, criando recursos interativos nos moldes do Fábulas.

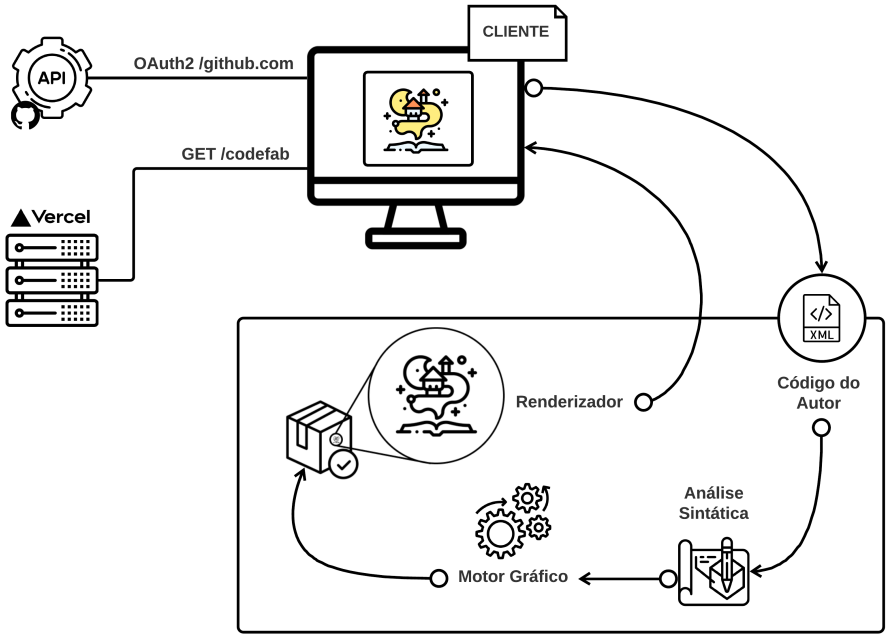
1.1 Objetivos

1.1.1 Objetivo geral

O objetivo do trabalho consiste em criar um *Playground* com intuito de facilitar a construção de aplicações hipermídias não-lineares utilizando o modelo Fábulas, tendo como nome Codefab.

A Figura 1 descreve o esboço inicial projetado para atender os conceitos trabalhados até aqui. Pensados para comportar um processo desde a análise sintática do documento de entrada, até a renderização da fábula, tudo isso apenas com os recursos disponíveis no lado cliente. Além disso, utiliza do paradigma *serverless* para agilizar a implementação do projeto no curto prazo, com uma integração de login social com o Github.

Figura 1 – Esboço da aplicação



Fonte: Autoria própria

1.1.2 Objetivos específicos

- Medir quantitativamente a usabilidade da ferramenta com o instrumento SUS (System Usability Scale), reforçando esta análise com as métricas coletadas pelo Hotjar.
- Instanciar e adequar o questionário TAM (Technology Acceptance Model) para analisar a aceitação da tecnologia.
- Avaliar quais os aspectos do pensamento computacional foram trabalhados na utilização da ferramenta.

1.2 Organização do trabalho

Além do Capítulo 1, este trabalho está organizado em mais 4 capítulos, sendo o Capítulo 2 a fundamentação teórica do trabalho, descrevendo as tecnologias, ferramentas e conceito utilizados na construção do Codefab.

O Capítulo 3 apresenta os requisitos e as especificações técnicas sobre o *Playground*. Além disso, são apresentados os diagramas comportamentais e estruturais do sistema.

Por fim, o Capítulo 4 apresenta o método proposto para realizar avaliação do Codefab discutindo os resultados. E o Capítulo 5 traz as conclusões do trabalho juntamente com os trabalhos futuros.

2 Fundamentação teórica

Neste capítulo são apresentadas as tecnologias e ferramentas utilizadas no desenvolvimento deste trabalho. Seguindo essa linha, é apresentada a justificativa e o contexto em que elas foram empregadas, para retratar a forma utilizada de alcançar o objetivo do projeto. Em seguida, as Seções 2.2 e 2.3 com uma visão geral sobre os conceitos abordados no trabalho fábulas e sua materialização através de uma biblioteca *javascript*, finalizando com um comparativo entre o modelo consolidado e o aperfeiçoamento realizado.

2.1 Tecnologias & ferramentas

As ferramentas utilizadas no desenvolvimento da aplicação e no *workflow* do projeto estão listadas na Tabela 1, cada linha presente da tabela contém o nome e a versão do item caso tenha, além de uma breve descrição.

Tabela 1 – Ferramentas e Tecnologias

Nome	Versão	Descrição
React	17.0.2	O React é um framework construído com Javascript e tem como objetivo a criação de aplicações SPAs com foco na experiência do usuário. (FACEBOOK, 2021b)
Context API	-	A Context API disponibiliza uma forma de passar dados de uma forma Global dentro da aplicação, sem a necessidade de uso constante de props na criação de componentes.(FACEBOOK, 2021a)
Konva	7.2.5	Biblioteca javascript que manipula o canvas do HTML5 para efeitos gráficos em 2D. (KONVAJS, 2021)
react-codemirror	4.0.7	Editor de texto com padrões modernos, feito para navegadores.(CODEMIRROR, 2021)
styled-components	5.2.3	Biblioteca que aplica o conceito CSS-in-JS, usada para estilizar componentes do React.js. (MADDERN, 2021)
Typescript	4.1.2	É um conjunto de ferramentas para se trabalhar com Javascript em aplicações modernas, como tipagem de dados, por exemplo. (MICROSOFT, 2021a)
Javascript	CJS	Linguagem de programação simples feita para ser intepretada em Browsers. Além de ser multi-paradigma. (MDN, 2021)

ESLint	7.32.0	Padronizador de código Javascript muito utilizado para aplicação de padrões de projetos, que venham facilitar a manutenção do código e adição de novos requisitos. (FOUNDATION, 2021)
Prettier	7.32.0	Formatador de código Javascript, serve para aplicar os padrões definidos no projeto.(PRETTIER, 2021)
react-xml-parser	1.1.8	É um analisador sintático que recebe uma entrada de dados em XML, seguindo regras específicas, para montar a estrutura que será usada no projeto.(@MATANSH, 2021)
Visual Studio Code	1.60.2	Ambiente de desenvolvimento, bastante utilizado para criação de Apps para a Web. (MICROSOFT, 2021b)
Figma	-	Ferramenta de prototipação em alto nível. (FIGMA, 2021)
Lucidchart	-	Ferramenta de modelagem para criação de diagramas estruturais e comportamentais. (LUCIDSOFTWARE, 2021)
GitHub	-	uma plataforma de hospedagem de código, além de versionar projetos.(GITHUB, 2021)
LocalStorage API	-	API nativa em Navegadores modernos, que possibilitam a persistência de dados independente de sessão.(WHATWG, 2021)
UUID	8.3.2	Indetificador único universal. (UUID, 2021)

Nos tópicos a seguir, estão as justificativas da utilização das ferramentas e tecnologias no projeto, citadas na ordem em que está organizado na Tabela 1.

- React.js - Aplicações modernas com o foco na experiência do usuário não requerem a necessidade de reloads ou avançar e voltar, tornando a navegação bem mais fluída. Além destas vantagens, a API do React será a base da criação de componentes (FACEBOOK, 2021b)
- Context API - Gerenciamento de estado global virou uma necessidade em UIs modernas, já que a forma de refletir informações nas interfaces ficou muito mais rápida e dinâmica. Na estruturação do projeto, esta dependência será a ponte entre os componentes, para assim compartilhar dados e seguir com o fluxo da ferramenta (FACEBOOK, 2021a).
- react-codemirror - Ele é utilizado para que não seja necessário construir um editor de texto, onde o autor possa digitar sua fábula, além é claro de usar o plugins e atalhos que tornem o processo mais eficiente (CODEMIRROR, 2021).

- styled-components - Foi escolhido para ser possível escrever código CSS dentro do próprio javascript, para que o estilo do componente possa ficar encapsulado. Além disso, serve também para construção das animações e leiaute da aplicação (MADDERN, 2021).

- Typescript - Serve para trabalhar com sintaxe de tipos em estruturas do javascript. Assim, ao utilizar no projeto será possível garantir a qualidade e a manutenção (MICROSOFT, 2021a) .

- Javascript - É o código objeto gerado no processo de *build* em uma versão com alta compatibilidade, resultado da transpilação do React somado ao Typescript, para que a aplicação possa ser interpretada em qualquer navegador (MDN, 2021).

- ESLint - Será a ferramenta para garantir a qualidade da manutenção do projeto através do emprego do *Design Pattern do AirBnB*. O Ambiente de desenvolvimento poderá avisar sobre os padrões empregados e até mesmo impedir a compilação quando não estiver de acordo com o está definido. (FOUNDATION, 2021)

- Prettier - Ferramenta capaz de aplicar as regras definidas no tópico anterior, formatando ajustes mais simples no processo de desenvolvimento.(PRETTIER, 2021)

- react-xml-parser - Esta ferramenta será utilizada para transformar o código de marcação escrito pelo autor em um DOM, para que seja possível trabalhar com o documento dentro ferramenta. (@MATANSH, 2021)

- Visual Studio Code - Com o uso de plugins e extensões, será o ambiente de desenvolvimento utilizado para codificar o *Playground* Web. (MICROSOFT, 2021b)

- Figma - É a ferramenta utilizada para construção dos protótipos de alta fidelidade que vão nortear o desenvolvimento. Além disso, ela será usada para elicitar e validar requisitos através dos artefatos gerados. (FIGMA, 2021)

- Lucidchart - É uma ferramenta que será utilizada no processo de modelagem da aplicação, criando os diagramas que serão consumidos no processo de desenvolvimento. (LUCIDSOFTWARE, 2021)

- GitHub - Será utilizada para controlar as versões, e hospedar o código do *Playground*. Também será parte da arquitetura do projeto, estabelecendo uma interface de comunicação entre os serviços de autenticação e publicação dos repositórios. (GITHUB, 2021)

- LocalStorage API - Neste projeto será utilizado para persistir informações através das sessões em que a ferramenta for utilizada, como por exemplo os dados obtidos na autenticação. Desta forma, criando uma política de *caching*, para diminuir o número de requisições em serviços de terceiros (WHATWG, 2021).

- UUID - É um identificador único universal gerado a partir de um pacote, ele será usado para criar uma chave única para o agente renderizado em tela, para que seja possível resgatar ou alterar alguma informação após a execução da fábula. (UUID, 2021)

2.1.1 DSL para autoria de *e-books*

Domain-Specific Language (DSL) é uma linguagem de especificação ou de programação, que através de abstrações e notações, busca resolver problemas em relação a um domínio específico(DEURSEN; KLINT; VISSER, 2000). Para o contexto de criação de *e-books*, ela se apresenta como uma proposta bastante interessante, já que abre uma gama de possibilidades para se construir documentos estruturados, consequentemente criando padrões para livros interativos. Para aplicar este conceito, existem linguagens de marcação muito relevantes, como o Extensible Markup Language (XML) (W3C, 2021), que através da organização dos dados usando *tags*[1] promove uma leitura legível para desenvolvedores e a interpretação dos dados por um sistema.

Existem trabalhos na literatura que apresentam propostas de criação de *e-books* interativos, construídos nas bases do HTML5, para que neste formato o livro tenha suporte para hiperlink, áudio, video e até 3D (CHOI, 2014). Neste cenário, os autores não precisam de base de programação para começar a construir os próprios recursos interativos e imersivos, o que promove o engajamento dos leitores no conteúdo, fornecendo uma experiencia de uso com transições e efeitos sonoros.

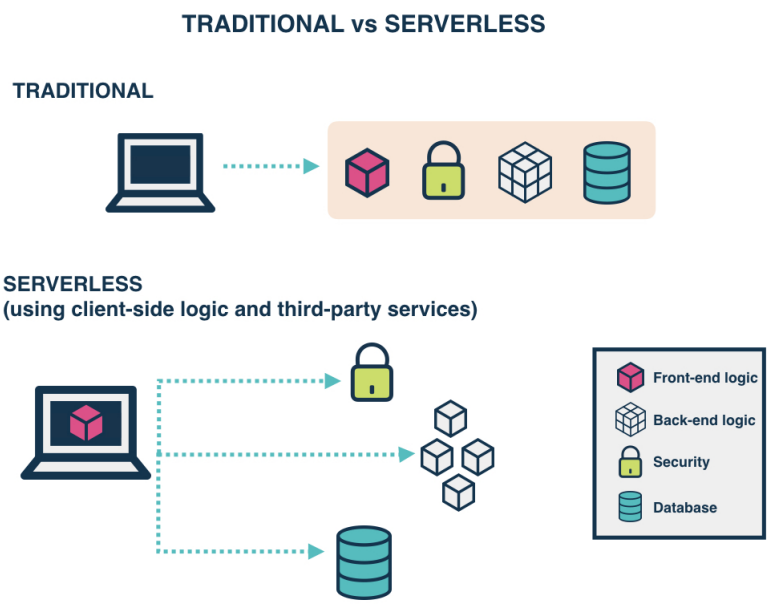
2.1.2 Arquitetura Serverless

O paradigma de desenvolvimento *serverless* é um dos vários modelos de computação em nuvem, que fornece recursos para a criação e a escalabilidade de aplicações, sem que o desenvolvedor fique responsável pela saúde e a manutenção da infraestrutura envolvida. É importante citar que os servidores estão presentes neste modelo, porém o objetivo da arquitetura é retirar as responsabilidades pertinentes a eles, deixando para o programador a missão de projetar uma aplicação que caiba neste cenário (REDHAT, 2021b).

Um ponto interessante neste modelo é a escalabilidade horizontal de sistemas que o adotam. Isso acontece porque a implementação é feita considerando o encapsulamento do projeto em uma imagem. Na prática, o código está empacotado junto com todas as suas dependências em um local controlado, o qual é conhecido como container, facilitando a replicação da aplicação para atender uma grande quantidades de requisições em um dado momento (DOCKER, 2021). Este ambiente é bastante interessantes para aplicações sem estados, que podem começar a executar a partir de qualquer momento, como por exemplo um blog ou sites que tenham requisições de maneira muito volátil.

[1] São marcadores de início e fim em uma linguagem de marcação, que serve para delimitar um contexto

Figura 2 – Comparação entre o modelo cliente-servidor clássico e o *serverless*



Fonte: (GOCD.ORG, 2021)

A Figura 2 exibe as principais diferenças entre as abordagens citadas até aqui. Na forma tradicional, podemos observar um cliente realizando uma requisição para o servidor, onde todos os aspectos envolvidos na operação são de responsabilidade do projetista do sistema. Para o modelo serverless, a figura está mostrando uma abordagem conhecida como *backend as a service* (BaaS), ou *backend* como serviço, onde pelo próprio fornecedor é possível ter acesso à serviços de autenticação, hospedagem, bancos de dados e vários outros(BACK4APP, 2021), onde o exemplo de maior relevância é Google Firebase (GOOGLE, 2021).

A arquitetura serverless pode ser dividida em várias categorias, onde o critério utilizado é abordagem para a implantação do modelo de negócio, como a que foi citada anteriormente. Por exemplo, a variante *Function as a Service* (FaaS), ou função como serviço, que é comumente a proposta mais atrelada ao modelo serverless, onde tem como objetivo aplicar a lógica de containers para execução de procedimento isolados no servidor (REDHAT, 2021a). Assim, esta abordagem permite um acesso ao banco de dados ou outros serviços, não sendo necessário a construção de uma REST API para manter a camada lógica do sistema.

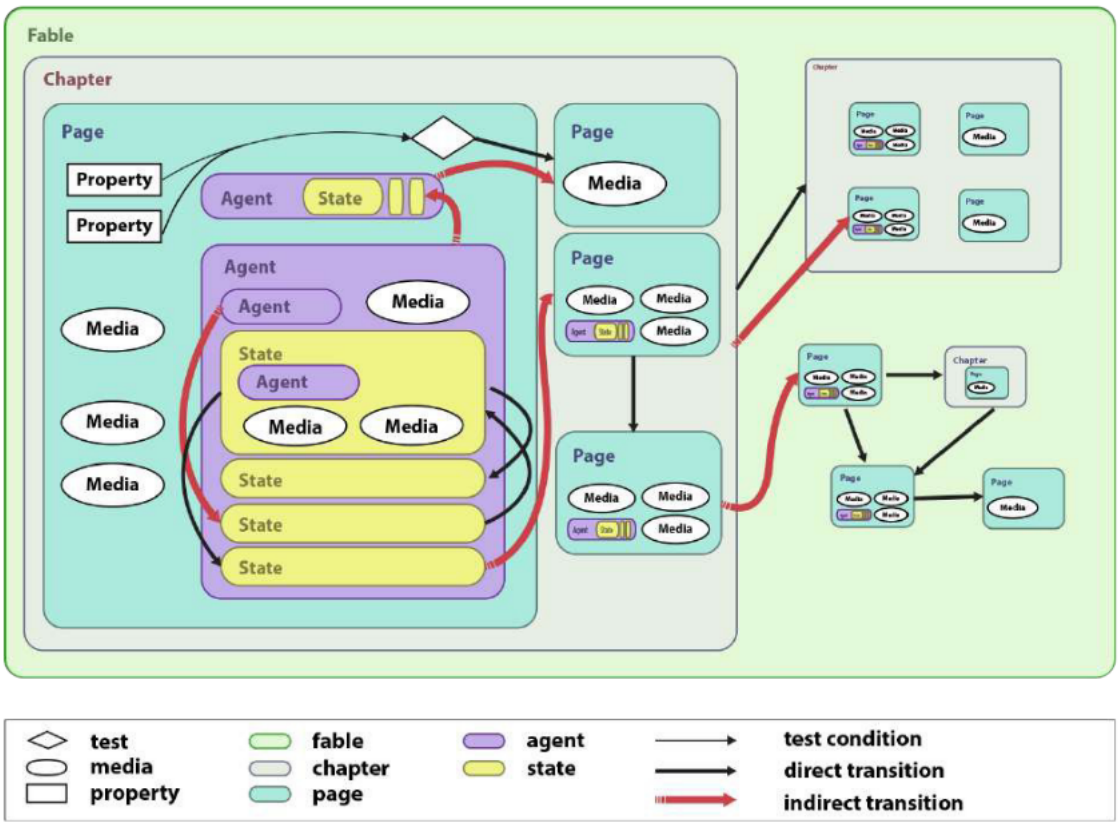
2.2 Modelo Fábulas

Como ferramenta pedagógica, o *e-book* é um recurso que se destaca pela sua experiência imersiva e dinâmica no processo de aprendizagem. Porém, existem alguns fatores limitantes para que este recurso possa ser trabalhado em sua máxima eficiência, como por exemplo, a necessidade de programação para construção de história com funcionalidades mais avançadas.

É neste cenário que surge o modelo Fábulas, que segundo o autor tem como

foco a criação de um documento guia, para que os autores possam criar seus recursos de forma declarativa (PINTO et al., 2017). Como é destacado no próprio trabalho, isso foi feito a partir de uma análise sistemática de *e-books*, ferramentas para a geração dos mesmos, e linguagens de autoria multimídia. Além da apresentação do modelo conceitual multimídia, o trabalho também apresentou um estudo de caso para demonstrar a viabilidade da aplicação em exemplos de *e-books*.

Figura 3 – Modelo conceitual de um eBook interativo como definido neste trabalho



Fonte: (PINTO et al., 2017)

Na Figura 3 é possível observar a estrutura de um *e-book* a partir da aplicação do modelo Fábulas, onde podemos observar um encadeamento das entidades que são usadas para a construção da fábula, muito semelhante com um conjunto de *tags* em um documento XML. O *Fable* encapsula toda a estrutura e serve para demostrar que o documento em questão é um história interativa. Além disso, temos o *Chapter* que é um apanhado de *Pages*, onde ambos servem para estruturar e organizar os elementos da narrativa, servindo como objetos de partida e destino para as transições da história.

Conceitualmente, o documento é formado por entidades e objetos de mídia comum, que se relacionam através de um outro modelo denominado evento-condição-ação, o ECA. Desta forma, o fluxo da história é controlado a partir das interações entre os elementos da narrativa e/ou ação direta do leitor da mesma. Esse relacionamento se dá a partir do *Agent*, que é a parte interativa do modelo, capaz de alternar entre possíveis estados e eventos no decorrer da história. O *State* é a

reapresentação desse estado contido em um agente, sendo que podem existir um ou vários, eles agrupam dentro do contexto do agente as imagens, vídeos, áudios e eventos relacionados ao mesmo.

O próprio modelo Fábulas é a principal colaboração do trabalhado citado, já que o mesmo é o resultado de uma abordagem comparativa entre vários *e-books* interativos, destacando os elementos mais importantes na construção de um *storytelling*. Como efeito direto do processo de pesquisa, é possível elencar como resultados, o mapeamento de ferramentas de autoria e linguagem multimídia para construção desses recursos, além dos requisitos de criação de *e-books* infantis a partir da visão de desenvolvedores.

Para a construção do *Playground* Web, será necessário um documento de marcação que faça referência a elementos multimídia, logo suas especificações serão nos moldes do modelo fábulas, seguindo algumas abstrações citadas nesta seção. A entidade *Chapter* não será aproveitada, apesar dela ser uma representação de um elemento de um livro físico, para efeito prático ela torna o modelo muito verboso com um degrau a mais, sem contar que pode facilmente ser substituída por uma *Page*.

2.3 FableJS

O FableJS é uma biblioteca escrita em javascript com o objetivo de criar livros interativos como uma instância do modelo Fábulas (SILVA et al., 2019). A implementação aplica os conceitos levantados na Seção 2.2, usando-os de forma prática para construção de histórias não-lineares em *e-books* para navegadores de internet.

A biblioteca tem como base o *framework* AngularJS, para poder ser possível a criação de *tags* personalizadas, capazes de encapsular estilos e eventos, sem contar a utilização dos atributos próprios. A implementação atua diretamente com tecnologias Web, já que tem como o principal documento de manipulação o HTML, utilizando as funcionalidades nativas da linguagem, como a estilização CSS de uma tag, e o suporte aos elementos multimídia.

A Tabela 2 mostra que o FableJS além de implementar as entidades apresentada no modelo fábulas, apresentou melhorias em determinados contextos. O *Board* e o *Alert* são bons exemplos dessas melhorias, eles foram criados para suprir a necessidade de caixas de diálogo com o leitor da história. Assim, não é mais necessário uma customização em CSS para cada texto que expresse uma mensagem para o usuário, sendo a diferença o contexto em que cada entidade será usada.

A *Detect* insere no modelo a possibilidade de zonas de detecção do evento de *drag and drop*[2], assim utilizando o atributo *draggable* nos elementos relacionados na tabela, é possível executar tal ação. Por último, a propriedade *transition* que tem como objetivo suavizar a transição entre *Pages* e *Chapters* durante o fluxo da história.

[2] Arrastar e soltar. Nomenclatura utilizada nas interfaces gráficas de computadores

Tabela 2 – Novos elementos e atributos

Nome	**Tipo**	**Elemento Pai**	**Atributos**
board	element	state, agent, page	set-class, font-size, color
alert	element	state, agent, page	-
draggable	attribute	-	-
detect	element	state	target, agent, change-state
trigger	element	state	agent, change-state
transition	attribute	-	-

Fonte: (SILVA et al., 2019)

O *Playground* terá suporte as melhorias apresentadas nesta seção, não necessariamente na preservação da entidade, mas no conceito em relação a experiência de uso na Web. No entanto, apesar dos novos elementos facilitarem a utilização da biblioteca, o número de entidades aumentou muito, o que pode dificultar o processo de escala do modelo para novos domínios. Além disso, a implementação tem uma forte dependência do *framework* AngularJS, o que acarreta na necessidade de experiência prévia com a ferramenta.

2.3.1 Aperfeiçoamento do Modelo

Na Listagem 1 a DSL está descrita no próprio HTML, assim o autor deve conciliar tando o código de marcação quanto o código para escrever a história. Além disso, o modelo Fábulas fica restrito as regras semânticas e sintáticas da especificação corrente do HTML, o que impede de utilizar atributos como gerador de eventos ou caminhos customizados para arquivos de mídia. Um efeito direto disso é uso exagerado de *tags* encadeadas, que apesar de ser um recurso poderoso, dificulta a leitura do código e acarreta uma DSL mais horizontal com *tags* que possuem um único atributo.

Seguindo para o Listagem 2, temos uma DSL de entrada para o playground, e a primeira impressão é que é um código mais vertical, graças ao pouco uso de *tags* encadeadas. Além disso, as *tags* que são filhas de uma *tag* `<agent />`, leva o nome do estado no próprio nome da *tag*, facilitando a busca pelo novo estado e a redução de um atributo para tal ação. Outro ponto, é o usuário não precisar declarar o caminho relativo ou absoluto de uma mídia, como é feito em um código que é executado em um ambiente local. Para que seja possível usá-la basta passar apenas o nome da mesma na propriedade que referência a fonte da mídia.

```
<fable width="1000px" height="600px" bg-sound="graphics/montagne.mp3">
  <page id="1" bg-image="graphics/BG.png" transition="fadeIn">
    <div class="title" id="title" style="position: relative; left:200px; top:50px;"
      transition="fadeIn">O pequeno cavaleiro</div>
    <img  src="graphics/char1.png" width="100" height="130" x="100" y="300"></img>
    <agent id="inicio">
      <state id="text1">
        <on-touch>
          <div style="position: absolute; left: 250px; top: 220px; font-size:30px; width:400px;">
            <p transition="fadeIn">
              Era uma vez um pequeno cavaleiro que recebeu uma carta de um amigo.
            </p>
              </div>
          <target start="inicio#text2">
        </on-touch>
      </state>
      <state id="text2">
        <on-touch>
          <div style="position: absolute; left: 250px; top: 220px; font-size:30px; width:400px;">
            <p transition="fadeIn" >
              Ele ficou muito muito muito surpreso, pois esse amigo tinha sumido a anos.
            </p>
              </div>
          <target start="inicio#text3">
        </on-touch>
      </state>
      <state id="text3">
        <on-touch>
          <div style="position: absolute; left: 250px; top: 210px; font-size:30px; width:400px;">
            <p transition="fadeIn">
              Só que as noticias na carta não eram muito boas. O seu amigo tinha contraído
              uma doença misteriosa e precisava   de ajuda para pegar a cura.
            </p>
              </div>
          <target start="inicio#text4">
        </on-touch>
      </state>
      <state id="text4">
        <on-touch>
          <img src="graphics/sign2.png" x="600" y="500" transition="tada">
          <target start="page#2">
        </on-touch>
        <div style="position: absolute; left: 250px; top: 220px; font-size:30px; width:400px;">
          <p transition="fadeIn">
            O mais rápido que pôde ele foi ao encontro desse amigo.
          </p>
            </div>
      </state>
    </agent>
    <img src="graphics/block.png" x="90"  y="421"/>
  </page>
</fable>
```

Listagem 1 – Modelo proposto para o FableJS

Além das melhorias já destacadas, nenhuma entidade precisa receber um atributo de identificação, para que seja possível criar as interações entre os elementos. Na Listagem 3, está demostrado o *Schema* gerado pela DSL do Exemplo 2, para que assim seja possível identificar os agentes, os estados, as ações e os eventos contidos na **<page />** que está ativa.

Como podemos observar no vetor de agentes, alguns deles possuem também um vetor, contendo os estados, que por sua vez podem conter um atributo de "on-touch", indicando que após uma ação de clique/toque, ele será direcionado

```
<fable width="500" height="500">
  <page background="bg.png" soundtrack="montagne.mp3">
    <agent text="O pequeno cavaleiro" x="40" y="30" />
    <agent img="cavaleiro.jpg" x="100" y="310" width="100" height="100" />
    <agent img="block.png" x="50" y="400" width="200" height="200" />
    <agent
      text="Era uma vez um pequeno cavaleiro que recebeu uma carta de um amigo."
      x="40" y="75" width="300" on-touch="story-1">
      <story-1
        text="Ele ficou muito muito muito surpreso, pois esse amigo tinha sumido a anos."
        on-touch="story-2" />
      <story-2
        text="Só que as noticias na carta não eram muito boas. O seu amigo tinha contraído
          uma doença misteriosa e precisava   de ajuda para pegar a cura."
        on-touch="story-3" />
      <story-3 text="O mais rápido que pôde ele foi ao encontro desse amigo." />
    </agent>
  </page>
</fable>
```

Listagem 2 – Modelo gerado após refinamento

para outro estado. De acordo com este exemplo, é possível observar que a DSL é realmente declarativa, já que após a execução, não será necessário um código para realizar a mudança de estado.

```json
{
  "pages": [
      {
          "background": "bg.png",
          "soundtrack": "montagne.mp3"
      }
  ],
  "agents": [
      [
          {
              "id": "652435a7-5847-4b04-ae57-62f0fb6d6e9d",
              "attributes": {},
              "states": []
          },
          {
              "id": "53958124-bc07-4d52-9b41-0e1b46fbeb35",
              "attributes": {
                  "img": "cavaleiro.png",
                  "x": "100",
                  "y": "310",
                  "width": "100",
                  "height": "100"
              },
              "states": []
          },
          {
              "id": "ddfca660-ed61-4f3f-b250-d4c7a1d11ec9",
              "attributes": {
                  "img": "block.png",
                  "x": "50",
                  "y": "400",
                  "width": "200",
                  "height": "200"
              },
              "states": []
          },
          {
              "id": "e1a5532a-4fc4-42ac-9eaf-31289fa97c0f",
              "attributes": {
                  "text": "Era uma vez um pequeno cavaleiro que recebeu uma carta de um amigo.",
                  "x": "40",
                  "y": "75",
                  "width": "300",
                  "on-touch": "story-1"
              },
              "states": [
                  {
                      "attributes": {
                          "text": "Ele ficou muito muito muito surpreso, pois esse amigo [...]",
                          "on-touch": "story-2"
                      },
                      "name": "story-1"
                  },
                  {
                      "attributes": {
                          "text": "Só que as noticias na carta não eram muito boas. O seu [...]",
                          "on-touch": "story-3"
                      },
                      "name": "story-2"
                  },
                  {
                      "attributes": {
                          "text": "O mais rápido que pôde ele foi ao encontro desse amigo."
                      },
                      "name": "story-3"
                  }
              ]
          }
      ]
  ]
}
```

Listagem 3 – Schema gerado após o processamento da DSL do exemplo 2 que será consumido pela Engine API

3 Codefab

A principal contribuição deste trabalho é a construção de um *Playground Web* com suporte ao modelo Fábulas, para que seja possível a criação de fábulas sem que haja a necessidade de configuração prévia de um ambiente por parte do usuário. O processo de modelagem, descrito neste capítulo, tem como objetivo uma ferramenta para gerar conteúdo multimídia, além de ser possível a visualização da fábula em navegadores modernos.

No que se segue, são discutidos os requisitos elicitados na Seção 3.1 para a construção da ferramenta, a modelagem completa do sistema na Seção 3.2, a apresentação da mesma após a aplicação das etapas anteriores na Seção 3.3 e, por fim, a Seção 3.4 com um relato da experiência de desenvolvimento, e métricas do projeto.

3.1 Requisitos

Nesta seção são apresentados os requisitos funcionais e não funcionais para direcionar o desenvolvimento da aplicação. Primeiramente são catalogados os requisitos funcionais seguindo a ordem da Tabela 3, cujo prefixo do código é RF. No que se segue, é feito o mesmo com os requisitos não funcionais de acordo com a Tabela 4, cujo prefixo do código é RNF.

Tabela 3 – Requisitos Funcionais

CÓDIGO	DESCRIÇÃO
RF01	O sistema deve permitir o gerenciamento dos *assets* importados pelo usuário
RF02	O sistema deve permitir que o usuário escreva uma fábula
RF03	O sistema deve permitir a edição de uma fábula escrita pelo usuário
RF04	O sistema deve renderizar a fábula escrita pelo usuário
RF05	O sistema deve salvar a fábula localmente no navegador utilizado

- **RF01: O sistema deve permitir o gerenciamento dos *assets*** - Para que a fábula tenha uma interação divertida, é interessante que o autor consiga usar ilustrações, sons e animações de sua preferência, para assim dar mais personalidade a sua história. A partir disso, é necessária a criação de um mecanismo de upload para que o usuário consiga manter os assets disponíveis para a sua fábula. Desta forma, as mídias que foram adicionadas vão ficar expostas para que o usuário consiga referencia-las durante o uso da ferramenta.

- **RF02: O sistema deve permitir que o usuário escreva uma fábula** - O principal requisito do sistema é permitir que seja possível a criação de uma fábula. Para isso, é necessário um editor de texto que seja acessível para que o autor consiga escrever um código. Este por sua vez, é escrito em uma linguagem de marcação (Ex: XML, HTML, SMIL), para que seja viável organizar as mídias de uma forma capaz de ser interpretada por um *Parser*. Para minimizar os atritos na criação do conteúdo, é interessante seguir os padrões de escrita em editores modernos como o *Sublime Text 3*[1].

- **RF03: O sistema deve permitir a edição de uma fábula escrita pelo usuário** - Este requisito é complementar ao anterior (o RF02), já que ambos os requisitos dizem respeito à forma a qual o usuário deve manter a sua fábula. Seguindo essa linha, o usuário é capaz de editar e apagar qualquer conteúdo escrito previamente, para assim, realizar alterações e correções no código que está trabalhando.

- **RF04: O sistema deve renderizar a fábula escrita pelo o usuário** - Para que o autor saiba o status do seu progresso em relação à fábula, é de vital interesse conseguir acessar um retrato de sua história. Isto posto, a aplicação deve conter uma tela de pré-visualização do conteúdo, para que seja possível observar as mudanças no rumo do projeto. A forma para utilizar este recurso na ferramenta é através de um botão de *Play*, onde, após o processamento do código renderiza um *Preview* com as mídias usadas.

- **RF05: O sistema deve salvar a fábula localmente no navegador utilizado** - Um dos fatores para uma boa usabilidade em um editor de texto padrão é a possibilidade em retomar a escrita de onde foi parado. Desta forma, a ferramenta deverá salvar ou carregar o código que está sendo construído pelo autor a partir de sua última alteração. A forma que este requisito fica disponível segue o padrão do mercado, onde a combinação de teclas, $Crtl + S$, é usada para executar a ação de salvar.

Após a elicitação dos requisitos funcionais, se mostrou necessária a utilização da técnica de prototipação para refinar e validar as funcionalidades levantadas. Além disso, foi usado como artefato neste processo a própria implementação em *Javascript* do Modelo Fábulas, visto que um ponto importante deste trabalho diz respeito a portabilidade para o *Playground Web*. A Figura 4 exibe o *wireframe*[2] construído a partir da Tabela 3, com o objetivo de guiar o desenvolvimento da ferramenta.

[1] https://www.sublimetext.com/3

[2] Esqueleto ou protótipo de uma versão bastante primitiva do visual de um projeto.

Figura 4 – Wireframe montado a partir dos requisitos

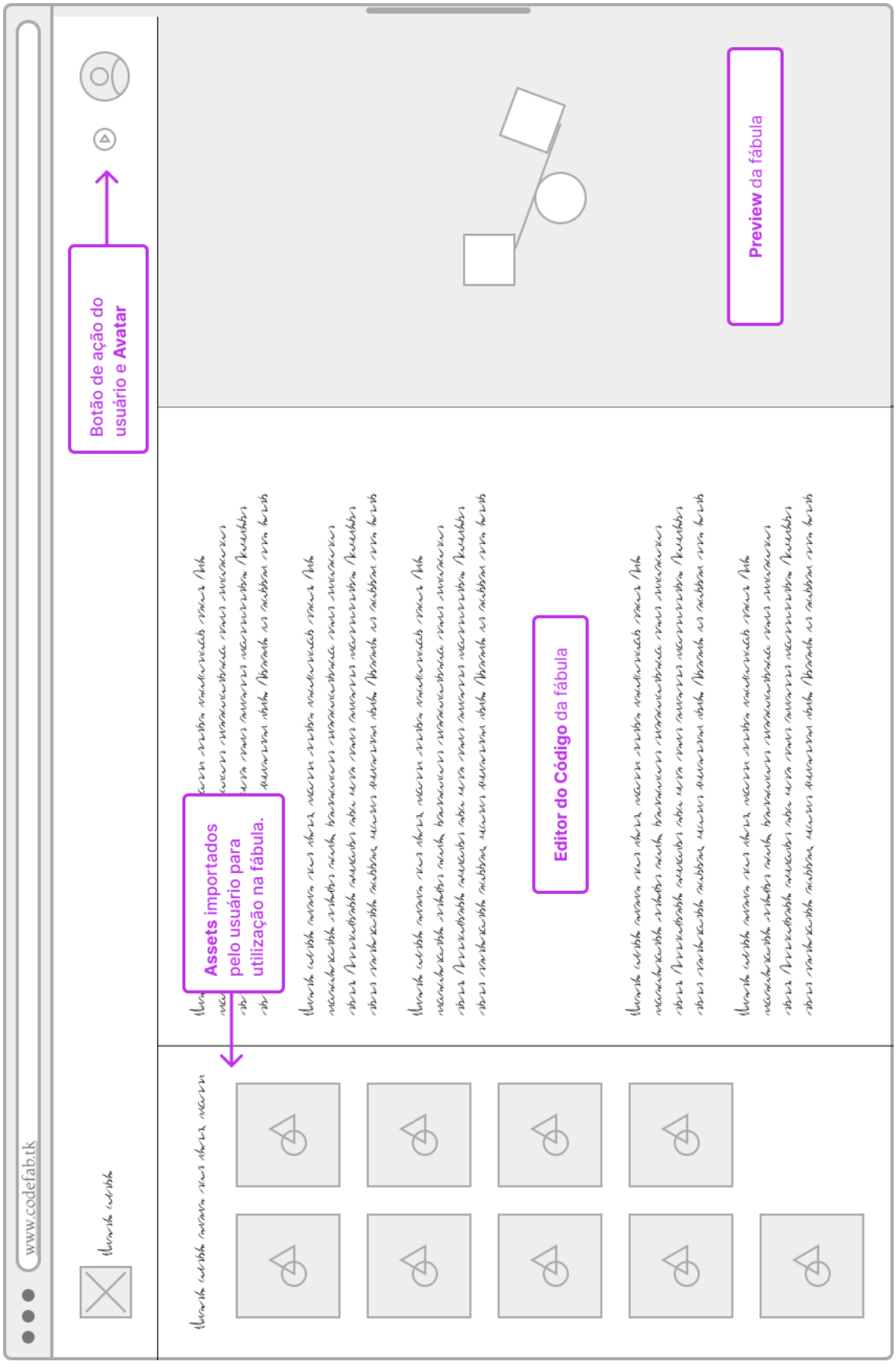


Tabela 4 – Requisitos Não Funcionais

CÓDIGO	DESCRIÇÃO
RNF01	O sistema deverá ser feito com foco na Web, sendo uma SPA
RNF02	O sistema deverá executar nos mais usados e modernos navegadores de internet
RNF03	O sistema deverá estar disponível 99% do tempo

- **RNF01: O sistema deve ser feito com foco na Web, sendo uma SPA** - Para diminuir a adesão da ferramenta, a Web se apresenta como uma boa opção. Isso porque, não se faz necessária a instalação e/ou configuração prévia de um ambiente para o uso da ferramenta. Além disso, a aplicação será construída como uma Single Page Application (SPA), para que o usuário tenha a experiência fluída, sem carregamentos e mudanças de páginas durante o uso da ferramenta. A forma de atenter este requisito é utilizando um *framework web* conhecido como React.js.
- **RNF02: O sistema deve executar nos mais usados e modernos navegadores de internet** - A forma que se dá o uso da ferramenta é através dos navegadores de internet. Desta forma, é de extrema importância a compatibilidade da aplicação com os seguintes navegadores: Google Chrome, Microsoft Edge e Mozilla Firefox, nos termos definidos na convenção de regras conhecida como *browserlist*[3]. Através do próprio *React.js* é possivel atingir os navegadores citados, já que ele mesmo usa o *Babel*[4], que é um compilador de *JavaScript*, para gerar código objeto para várias especificações utilizando *Polyfills*[5].
- **RNF03: O sistema deve estar disponível 99% do tempo** - Para atender a demanda de utilização da ferramenta, se faz necessário estar acessível a maior parte do tempo. Assim, o autor pode criar suas fábulas a qualquer horário. Desta forma, uma solução *FaaS*[6] se mostrou bastante interessante dado o contexto. Uma das opções disponíveis no mercado é a *Cloud* da *Vercel*[7], onde usando a conta gratuita já é possível fazer *deploy*[8] de uma aplicação Web.

3.2 Modelagem

3.2.1 Diagramas comportamentais

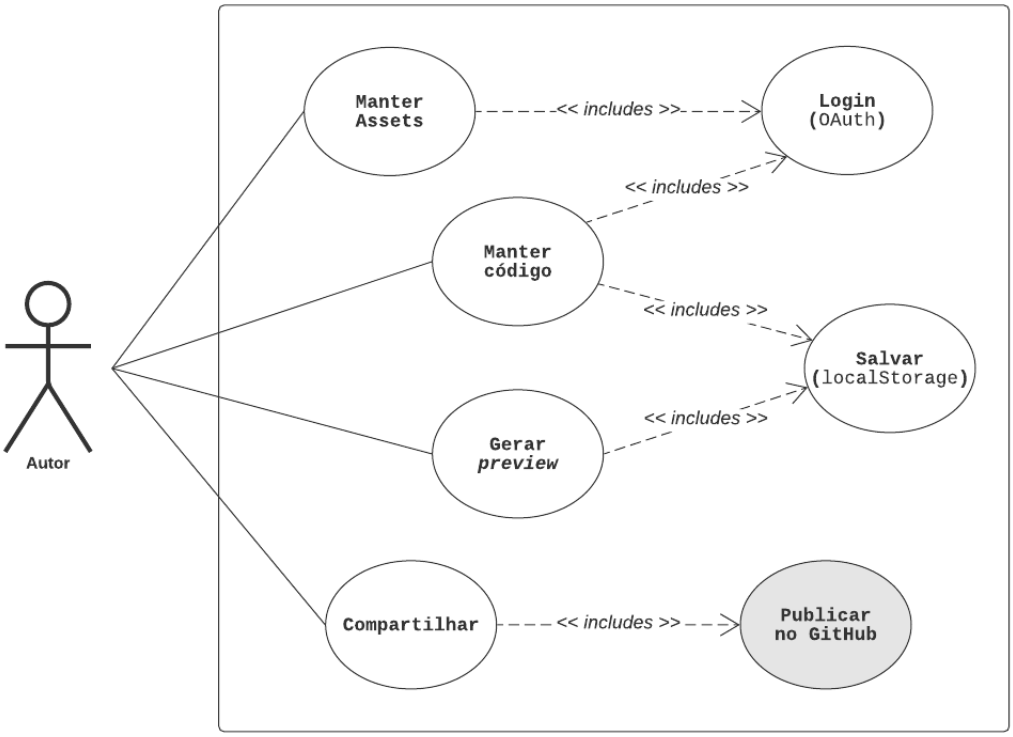
Começando pela Figura 5, temos uma visão geral de como o usuário se relaciona com as funcionalidades da ferramenta. Segundo o diagrama, podemos

[3] https://github.com/browserslist/browserslist
[4] https://babeljs.io
[5] É um pedaço de código usado para fornecer funcionalidades modernas em navegadores mais antigos que não o suportam nativamente.
[6] Uma solução Cloud Computing em que as plataformas de hardware e software de aplicações são fornecidas por terceiros, com suporte a funções em *containers* sem necessidade de uma REST API.
[7] https://vercel.com
[8] Colocar no ar alguma aplicação que teve seu desenvolvimento concluído.

observar que o autor da fábula é único ator do sistema, e por sua vez, ele pode manter ilustrações, efeitos sonoros e *sprites* para contar suas histórias. Além disso, o usuário é capaz de editar, apagar, copiar e salvar o código que foi escrito pelo editor.

Para que seja possível executar essas ações é necessário que haja uma autenticação do usuário que acessou a ferramenta. Dessa forma, o autor da fábula pode compartilhar o conteúdo que foi gerado pelo playground e realizar os casos de uso já citados. Por fim, para perceber o progresso de construção da fábula, o autor pode interagir com o resultado a partir de uma tela de pré-visualização.

Figura 5 – Caso de uso Geral



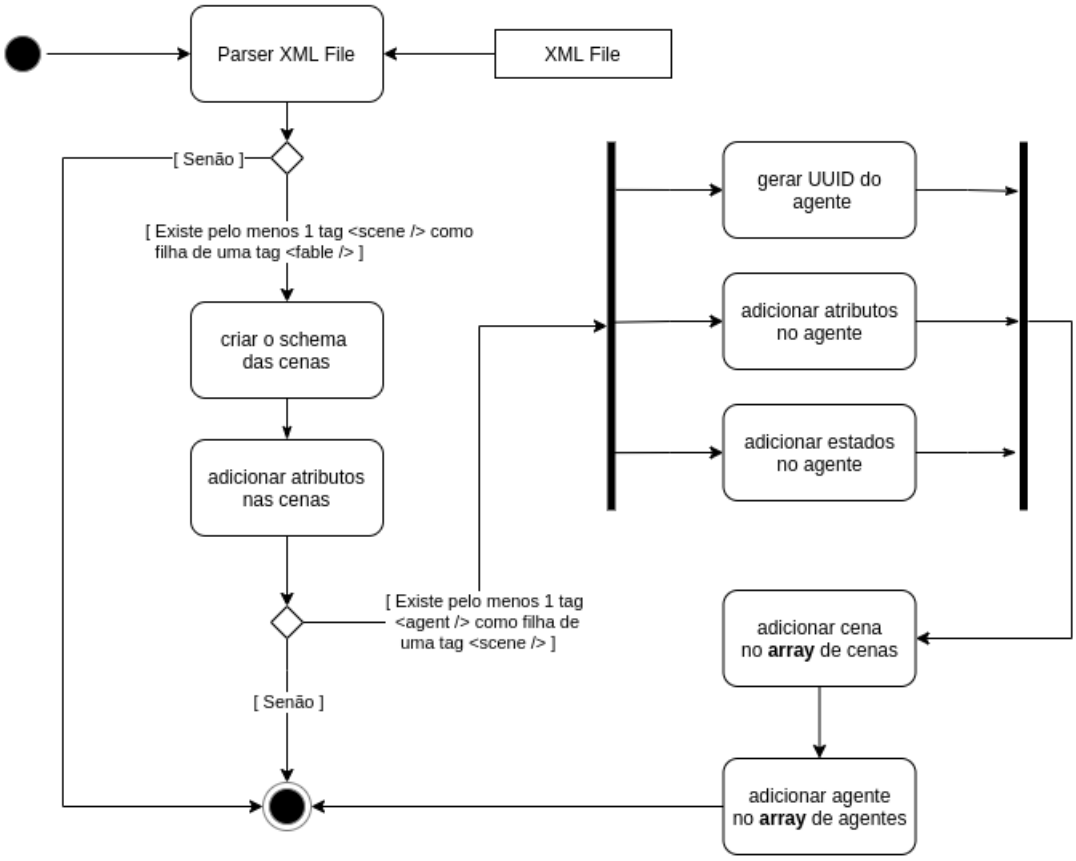
A partir do caso de uso geral apresentado, podemos ter uma compreensão do propósito da ferramenta, e uma visão genérica sobre as funcionalidades do sistema. Porém, esta perspectiva é muito alto nível e lida com muitas questões, para ajudar a compreender a complexidade dos casos de uso, segue os diagramas descrevendo as ações e desvios condicionais de alguns fluxos da ferramenta.

A Figura 6 mostra, com detalhes, o processo de montagem do ***Schema***[9], desde o modelo inserido no início do fluxo até a saída que será consumida pelo motor gráfico, no mercado de games denominado Engine. O processo se inicia a partir da entrada de um arquivo XML escrito nos moldes da linguagem de especificação do modelo Fábulas. A partir disso, é criado um modelo de documento por objetos (DOM), para que seja possível checar se a convenção está sendo aplicada.

[9] Vocabulário de dados estruturados que define entidades, ações e relacionamentos

Para que o *Schema* seja gerado, é necessária a utilização de *tags* reservadas na construção da fábula, sendo as principais: **<fable />**, **<page />** e **<agent />**. Além disso, é necessário que haja o encadeamento das *tags*, para mostrar ao Parser o contexto do emprego dos atributos de cada uma. Antes de gerar cada agente, o Parser já realiza a montagem de cada cena dentro da fábula. Esta fase leva em consideração os atributos da *tag* referentes à cena, ou seja, a imagem de background e a trilha sonora, para a *tag* refente à fábula e o título da mesma.

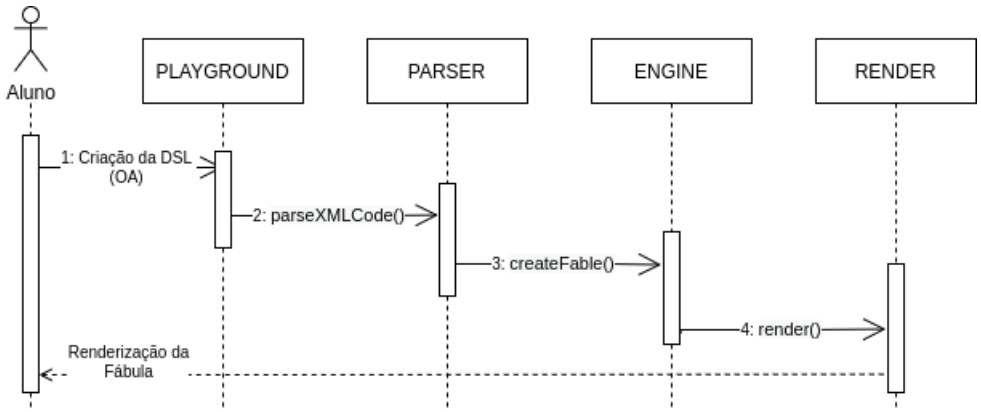
Figura 6 – Fluxo de Parser do documento



O processo de criação de cada agente se assemelha ao da cena, porém com uma diferença fundamental: o controle dos estados. Cada um deles pode alternar entre as possibilidades definidas na DSL, para serem usados a partir de ações que ocorrem durante a interação com a fábula. Para isso, é necessário um identificador único, que após a execução da fábula, é utilizado para rastrear o agente na cena e aplicar a mudança de estado. Por fim, o modelo de entrada já está processado e pronto para ser organizado em um *Schema*.

Para representar o fluxo de pré-visualização da fábula, foi utilizado o diagrama de sequência, para assim exemplificar os aspectos de ordem temporal em relação ao usuário. De acordo com a Figura 7, vemos que a partir do processo descrito anteriormente, o sistema executa a transposição do modelo. Desta forma, o *Schema* está apto para ser encaminhado para a Engine. A partir deste momento, utilizando a função *createFable()*, é passado como argumento o resultado de processamento

Figura 7 – Fluxo de renderização



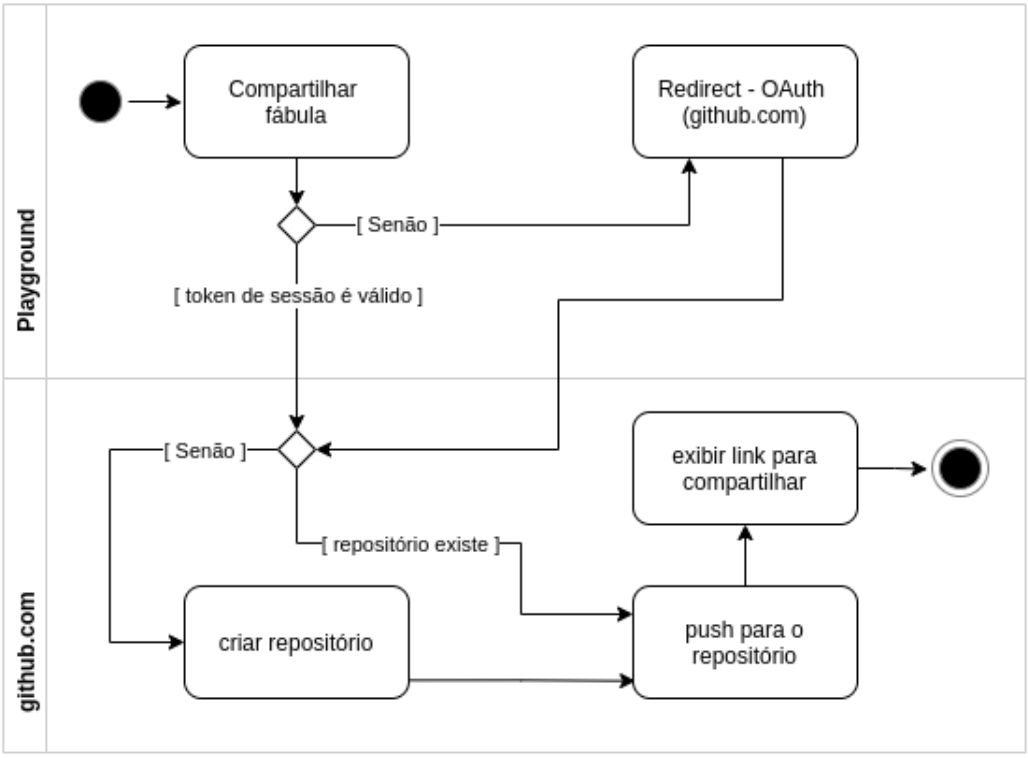
do parser, para que o motor gráfico da aplicação consiga renderizar as mídias de acordo com as marcações feitas pelo autor da fábula.

Por fim, a Figura 8 retrata o processo de publicação do conteúdo de uma forma geral. Além do código escrito pelo autor, existem os assets utlizados no processo de criação da história, que devem ser empacotados em um mesmo lugar. O processo de publicação utiliza o github para manter o projeto, já que o mesmo é utilizado para versionar código. A diferença aqui é que esta etapa é feita de forma automática pela própria ferramenta a partir de uma ação do usuário.

Após o autor da fábula iniciar o processo de publicação, é checado se o token de sessão (JWT[10]) gerado na autenticação ainda é válido. O diagrama abaixo exibe o comportamento da aplicação diante da resposta. Após a validação ou geração de um novo token, os arquivos são enviados para um repositório no domínio do *github.com*, depois deste processo é exibido o link para compartilhar o projeto.

[10] JSON Web Token

Figura 8 – Fluxo de Publicação do conteúdo



3.2.2 Diagramas estruturais

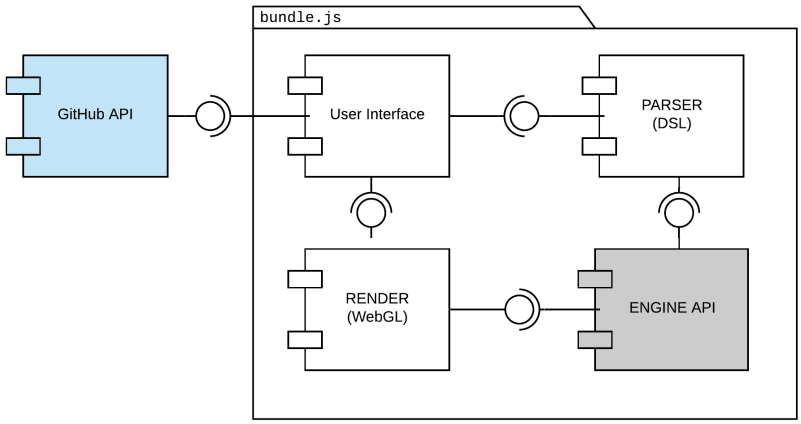
Seguindo para a estrutura do sistema, temos o diagrama de componente responsável por demonstrar a arquitetura da aplicação na Figura 9. A aplicação utiliza apenas recursos de armazenamento e processamento locais para sua execução. Desta forma, após o download dos arquivos estáticos do servidor, a ferramenta irá rodar integralmente no lado cliente.

Após o carregamento do código objeto, a aplicação se organiza em quatro módulos essenciais para seu funcionamento e uma interface de comunicação com um serviço externo. Antes de exemplificar cada um deles, é importante tratar da forma que eles são orquestrados durante a execução. Comumente aplicações que são SPAs, usam gerenciadores de estado global. Assim cada componente pode manter sua dependência e ao mesmo tempo estar ciente dos dados que estão sendo compartilhados.

O Primeiro módulo é responsável pela captura das informações de entrada do usuário, ou seja, são os componentes visíveis pelo autor da fábula. Esta parte da arquitetura recebe através do editor de texto, um arquivo de marcação, seguindo o padrão XML, onde o usuário determina a forma como o conteúdo está organizado.

Seguindo para a próxima etapa do processo, temos o *Parser* onde é realizada a tarefa de transpilação. O código de marcação é interpretado pelo motor gráfico da ferramenta, logo, este serviço é necessário para criar a estruturas que irão conter os dados de cada mídia. Na Figura 10 é possível ver a organização do dados extraídos

Figura 9 – Módulos da aplicação

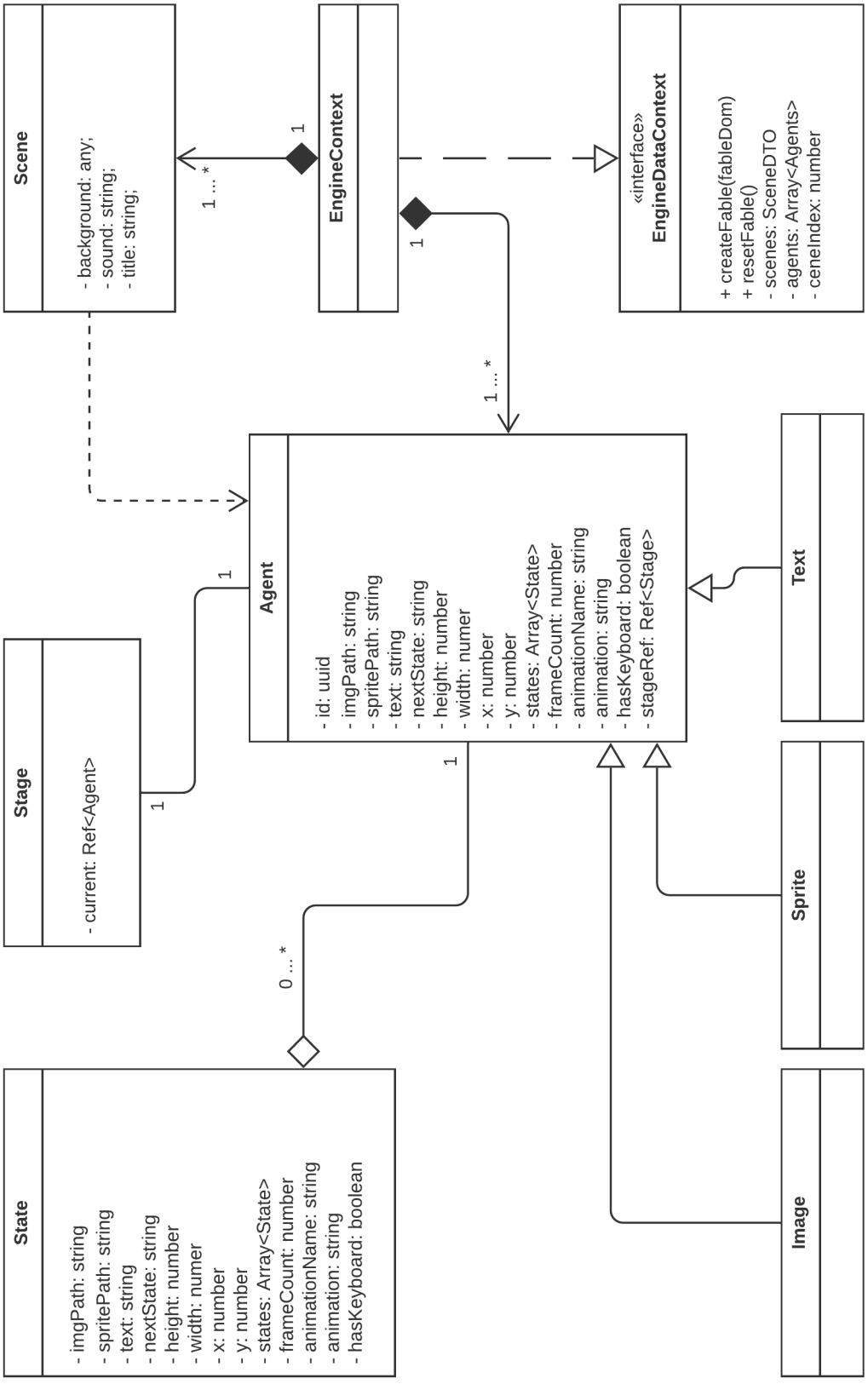


em um diagrama de classe e como estão relacionados.

Após o código objeto ser gerado, ele é passado como parâmetro para a construção visual da história. Na prática, uma fábula é um apanhado de mídias visuais e sonoras, com interações a partir dos periféricos. O método chamado para a criação da história entende a estrutura que lhe foi fornecida. Portanto, cada imagem que foi declarada na marcação é seguida de informações importantes para que o processo de renderização seja possível. Cada mídia é estruturada de forma a manter uma posição em (X, Y), uma largura e um comprimento, e também um caminho relativo para a imagem em si.

Por fim, o processo de renderização chega a sua última parte do trajeto: o desenho das mídias na tela de *Preview*. O módulo de render recebe as especificações descritas no parágrafo anterior, e desta forma temos a fábula sendo renderizada em um nó do DOM.

Figura 10 – Diagrama de Classes da estruturação dos Agentes



3.3 Apresentação da Ferramenta

Nesta seção serão apresentadas as telas da aplicação após o seu desenvolvimento. Assim, seguindo a ordem das figuras abaixo, será descrito o que foi entregue em cada uma delas.

A Figura 11 e a 12 exibem a ferramenta na visão do usuário no processo de construção de sua fábula. Podemos observar, que o design da ferramenta está no nível de uma aplicação moderna e focada na experiência de uso. Além disso, o leiaute segue o padrão de dashboards, onde a divisão é feita entre a Barra de Navegação na parte superior, uma Barra lateral à esquerda e o Editor no centro-direita para o foco principal da ferramenta. Outro ponto importante, é a localização do *Preview*, já que o mesmo não está encaixado no que foi descrito, pois ele possui uma posição flutuante na aplicação. Desta forma, o usuário, através de uma operação de *drag and drop*, pode movimenta-lo sobre as demais seções.

Figura 11 – Dashboard do usuário contendo as fábulas de sua autoria

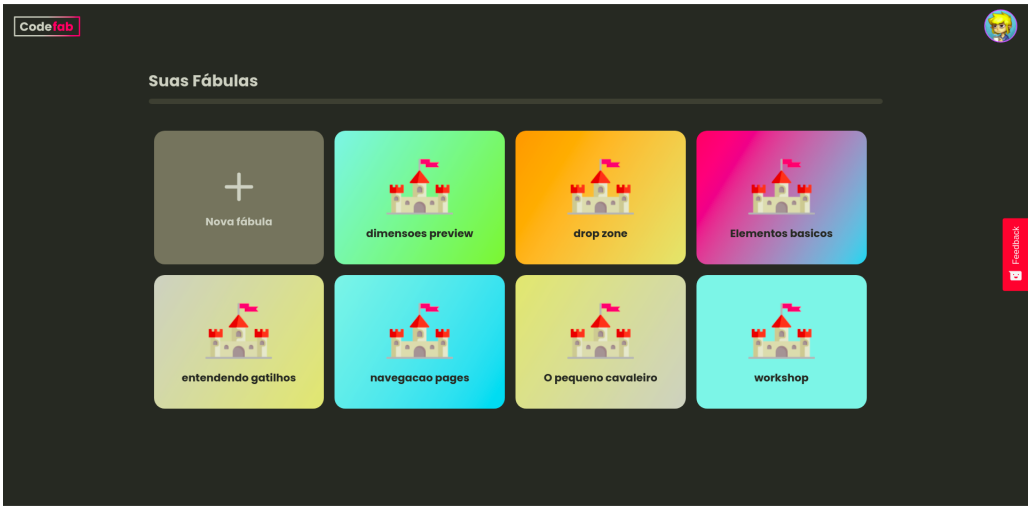


Figura 12 – Visão Geral da Ferramenta

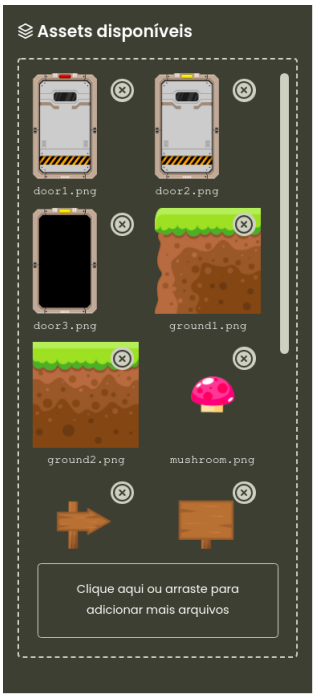
A Figura 13 mostra o componente Barra de Navegação, que é responsável por agrupar o *branding* da aplicação ao lado esquerdo, e ao direito o menu de ação

do usuário. Neste menu, é possível observar o botão de *Play*, que é responsável por iniciar o processo de renderização da fábula. Além dele, são exibidas as opções para manter a fábula, como publicar ou deletar. Também existe o suporte para o autor, disponível no botão de dúvidas, e a opção para compartilhar o que foi produzido.

Figura 13 – Barra de navegação da Ferramenta

A Figura 14 exibe a Barra lateral da aplicação que é usada para a exibição dos *assets* mantidos pelo usuário. Além disso, é possível notar um mini-preview da mídia, para facilitar a utilização da mesma por parte do autor. Após o upload, fica disponível a função para remove-la do projeto.

Figura 14 – Coluna de Assets da Ferramenta



Na Figura 15, temos o principal requisito do *Playground* sendo atendido, o local responsável pela entrada de código da fábula: o Editor de texto. Podemos observar, que ele segue um padrão bem moderno de design, lembrando o *Sublime Text 3*. Além disso, ele possui atalhos de teclado como o $Ctrl + S$, e o *Code Folding* que permite compactar blocos de código definidos em uma *tag* XML, para assim facilitar a leitura geral da fábula. Outra funcionalidade bastante interessante é o auto complete para código de marcação, ou seja, com o atalho $Ctrl + Space$ é possível listar as *tags* que possuem suporte na ferramenta. Além disso, após a declaração de uma *tag*, a utilização do atalho ajuda a exibir os atributos aceitos naquele contexto.

Por fim, na Figura 16 pode-se ver a tela de pré-visualização da aplicação, onde é possível o autor acompanhar e interagir com o estado mais recente da fábula em construção. As figuras abaixo estão exibindo os agentes em cena com uma *box* e uma placa com o ID ao redor deles, para demostrar o *tracking* de cada um após

Figura 15 – Editor de Código da Ferramenta

a execução da fábula. Além disso, no quadro da direita é possível observar que a porta mudou de estado e seu identificador continuou o mesmo.

Figura 16 – Preview de fábulas da Ferramenta

3.4 Conclusão

Com a construção desta ferramenta foi possível observar que os *Frameworks* da comunidade Javascript vem se tornando cada vez mais relevantes nos cenários de experiência do usuário, já que aliado aos conceitos de SPA's, vem transformando as aplicações Web em sistemas mais escaláveis e robustos. Aliado ao Typescript, o React.js mostrou um grande poder em criar componentes acessíveis a partir de qualquer navegador ou plataforma, sem a necessidade de inúmeras configurações prévias para colocar no ar um sistema.

Outro ponto para ser destacado é a vasta contribuição de plugins e pacotes da comunidade via projetos *Open Source*[11]. Desta forma, é possível a criação

[11] Código fonte disponibilizado e licenciado com uma licença de código aberto

de ferramentas cada vez mais poderosas, alicerçadas nos pilares da comunidade, gerando valor com muito mais velocidade. Vale destacar também as plataformas com opções gratuitas para fazer o *deploy* de um App Web. A Vercel é uma empresa que presta serviço na modalidade *PaaS*, onde é possível com uma conta grátis colocar um sistema online. Além dela, existe a *Freenom*, um provedor de domínios, onde é possível comprar ou adquiri-los de forma gratuita.

Por fim, foi possível atender os requisitos elicitados a partir dos artefatos e da experiência adquirida nas interações com os *stakeholders*. A ferramenta foi inteiramente construída sobre o *framework React.js* com o suporte à Typescript. Ela conta com 4.185 linhas de código, 6 componentes e 5 páginas, isso tudo no tamanho total de 53.1 MB, sem contar as dependências dos *node modules*. Além de tudo isso, sua versão comprimida possui um bundle de 19.63 MB, o que diminui a dificuldade de instalação em ambiente de produção.

4 Avaliação

Para avaliar o Codefab na sua proposta enquanto *Playground*, foi realizado um estudo qualitativo no qual os voluntários da pesquisa construíram uma fábula de tema livre. Para que isso fosse possível sem tutoria, houve um momento para realização de um tutorial da ferramenta, abordando conceitos e funcionalidades. Desta forma, o principal foco é avaliar a capacidade de criação de uma história interativa multimídia não-linear (fábula), utilizando o Playground Codefab para tal tarefa, além de avaliar os aspectos definidos nos objetivos específicos.

No que se segue, a Seção 4.1 faz o detalhamento do contexto e das etapas da metodologia utilizada, na Seção 4.2 é apresentado os resultados do experimento.

4.1 Metodologia de Avaliação

O conceito de usabilidade está relacionado com a facilidade de uso em relação a utilização da interface, assim como a satisfação, fruto deste processo (BARBOSA; SILVA, 2010). Para avaliar esta relação no *Playground* de forma prospectiva, se fez necessário o desenvolvimento de um questionário que aplica instrumentos para realizar esse tipo de análise.

Um dos instrumento citados é o System Usability Scale (SUS), que tem como mecânica a aplicação de 10 questões adaptáveis ao contexto da aplicação, sobre a percepção do usuário em relação a facilidade de uso do sistema (BROOKE, 1995). Ao final de sua aplicação é gerado uma pontuação, o *SUS Score*, onde com ele é possível classificar a ferramenta em uma faixa de usabilidade percebida. Esta classificação leva em consideração estudos realizados em diversas ferramentas, as notas são letras (de A até F - maior para o menor ranking), onde o valor mais alto representa a possibilidade do sistema ser indicado para outras pessoas (SAURO, 2021).

Apesar do foco do instrumento em medir a usabilidade da interface de um sistema, ele também pode ser usado para medir a capacidade de aprendizado da ferramenta. Este conceito se refere a facilidade com que os usuários completam suas tarefas em seu primeiro contato com a interface, além da capacidade de aperfeiçoamento a medida que utilizam o sistema(JOYCE, 2021). Para isso, é destacado dois itens do questionário que dizem respeito a esta dimensão, assim aproveitando o instrumento para trabalhar os aspectos de usabilidade e a capacidade de aprendizado (LEWIS; SAURO, 2009).

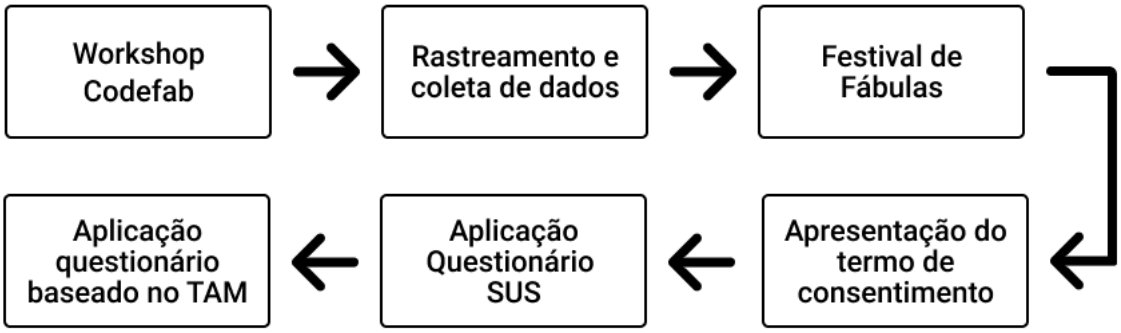
Para uma abordagem qualitativa existem modelos teóricos para compreender a aceitação de uma tecnologia e como isso impacta a percepção do usuário em relação ao sistema. É neste contexto que se apresenta o Technology Acceptance Model (TAM), uma ferramenta criada para este cenário, onde seu objetivo é encontrar fatores relacionados a satisfação do usuário (VENKATESH; DAVIS, 2000). Este instrumento, está fundamentado sobre dois pilares, a utilidade e a facilidade percebida de uso, onde é definido o grau em que o usuário entende que o sistema aumenta

seu desempenho no trabalho, e o nível em que ele acredita que o uso do sistema será livre de esforço (DIAS et al., 2011).

Para mapear de forma descritiva os quatro pilares do pensamento computacional que estão presentes no modelo Fábulas, foi proposto um ensaio de interação, para analisar a relação de cada participante com a interface. Para unir forças com esta análise, de forma quantitativa, a ferramenta Hotjar (HOTJAR, 2021) se mostra uma solução interessante, já que é possível capturar métricas de uso do sistema através do próprio *Playground*. Além disso, para organizar e controlar as fábulas, o Padlet (PADLET, 2021a) possui um Mural para este tipo de proposta, sem contar as funcionalidades de interação social por meio de comentários e curtidas.

Diante disto, o processo utilizado para realizar a avaliação da ferramenta foi formado por dois grandes momentos: o primeiro é o ensaio de interação, que para ganhar um aspecto lúdico de acordo com a temática, foi chamada de Festival de Fábulas. O segundo é a aplicação do questionário de pesquisa, contemplando os instrumentos abordados, além de duas questões de resposta livre sobre pontos positivos e negativos sobre o Codefab. Para as respostas do SUS e do TAM, foi adotado a escala Likert de cinto pontos(VAGIAS, 2006), tudo isso aplicado remotamente via *Google Forms*. A Figura 17 ilustra a ordem das etapas que estão contidas nestes momentos.

Figura 17 – Etapas da avaliação



Fonte: Autoria própria

- **Workshop Codefab:** O objetivo desta etapa é fazer a apresentação da ferramenta (editor, menu do usuário, botões de ação e manter o projeto), além de passar os conceitos fundamentais para estruturação do documento de marcação.
- **Rastreamento e coleta de dados:** Durante o período de tempo entre a primeira e a segunda etapa, será coletado os dados de tracking de click e scroll do mouse, além da gravação de tela e dados sobre a sessão.
- **Festival de Fábulas:** O objetivo desta fase é convidar todos os participantes para apresentar os resultados que obtiveram. Além disso, gerar uma interação social entre os voluntários, comentando e interagindo com os demais projetos.
- **Apresentação do termo de consentimento:** A apresentação do termo enfatiza o contexto e objetivo do estudo. A partir disso, o voluntário está ciente da

coleta dos seus dados no processo de pesquisa, podendo consentir ou não sobre a utilização dos mesmos.

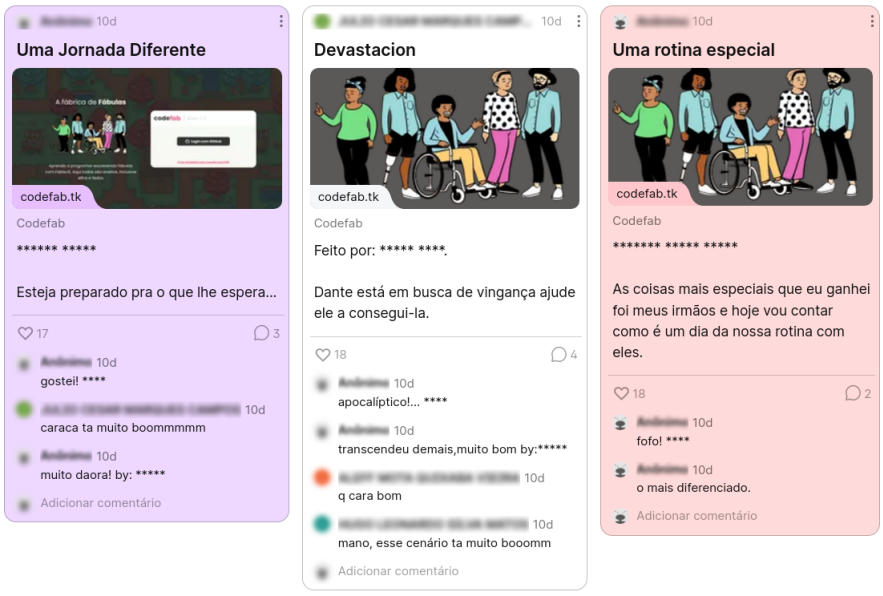
- **Aplicação questionário SUS:** O objetivo desta etapa é coletar as percepções sobre a usabilidade do Playground, aplicando a ferramenta SUS via formulário de pesquisa.
- **Aplicação questionário baseado no TAM:** Seguindo a etapa anterior, esta fase busca colher informações sobre a utilidade e facilidade percebida de uso com a aplicação de um formulário baseado no TAM.

4.2 Resultados e Discussões

Para este experimento foram convidados estudantes universitários do curso de Ciência da Computação (UFMA), regularmente matriculados na disciplina de Algoritmos I. Durante o Festival de Fábulas, foram contabilizadas trinta e quatro participações, porém apenas um voluntário não participou das pesquisas, totalizando trinta e três respostas. Dos que participaram, 60,5% possuíam idades entre 18 e 19 anos, e os demais variando de 20 a 29 anos, destes 81,8% são do sexo masculino e 18,2% feminino.

O andamento do Festival de Fábulas ocorreu como o esperado, sem o acontecimento de problemas técnicos que inviabilizassem as inspeções, de modo que as apresentações pudessem acontecer de forma ininterrupta. Todos os participantes cumpriram o desafio de construir um fábula de tema livre com várias páginas não-lineares.

Figura 18 – Fábulas do Mural



Fonte: Autoria própria

A Figura 18 mostra exemplos das interações ocorridas no Mural do Padlet do Codefab (PADLET, 2021b) ao longo do Festival. Durante a apresentação o autor era convidado a compartilhar suas impressões sobre a experiência de utilizar o *Playground*. No entanto, esta não era a única forma de levantar informações de uso do sistema, já que o Hotjar (HOTJAR, 2021) estava integrado na ferramenta e registrou 221 sessões na plataforma.

Através dos dados capturados, foi possível observar que a duração média de cada sessão foi 50 minutos e 27 segundos, tendo 82 minutos e 28 segundos como desvio padrão, o que mostra o alto grau de dispersão do volume de tempo gasto na ferramenta. Além disso, foi registrado os contadores de cliques e de navegação entre as páginas, com valores de 41.319 e 1.378 respectivamente. Isso demonstra o engajamento da ferramenta entre os usuários, dado o grande volume de ações capturadas e o tempo gasto dentro da plataforma.

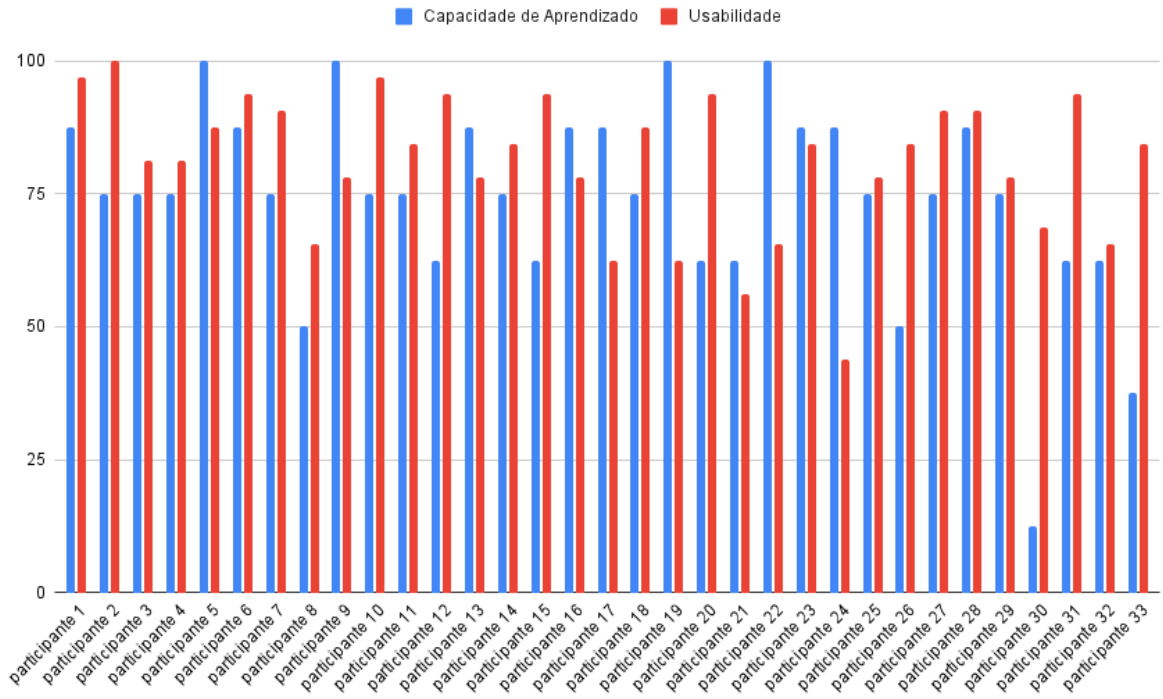
O Codefab também recebeu acesso de vários navegadores de internet diferentes, para o Google Chrome foram registradas 147 sessões , para o Microsoft Edge foram 34, já o Opera obteve 22 e o Mozilla Firefox apenas 5. Estes dados são importantes, já que mostram que os requisitos não funcionais levantados na Seção 3.1 foram atendidos, como por exemplo o RNF02 exibido na Tabela 4.

A análise dos dados obtidos pela aplicação do instrumento SUS, no geral apresentaram resultados satisfatórios. O *Playground* obteve um SUS Score médio de 79,70, onde o valor máximo é 100, o que classifica a ferramenta como ranking B (SAURO, 2021), demostrando um bom aceite no aspecto de utilidade percebida pelos usuários. Além desta pontuação, foram obtidos resultados referentes à usabilidade e a capacidade de aprendizado, onde o Codefab recebeu 81,06 e 74,24 respectivamente, em uma escala onde o valor máximo é 100. Isso demonstra que o *Playground*, apesar de uma boa avaliação de usabilidade no geral, não é tão simples de utilizar um primeiro momento, sendo necessárias mais repetições para facilitar esta tarefa ao decorrer do tempo.

A Figura 19 explora a avaliação dos aspectos trabalhados em relação a cada participante. Podemos observar que a capacidade de aprendizado são as maiores flutuações em relação aos participantes, onde é possível acompanhar resultados com a nota máxima (100) e ao mesmo tempo avaliações com uma nota inferior a 25, demostrando a facilidade de alguns em contra ponto a dificuldade de outros no uso da ferramenta em um primeiro momento. O aspecto de usabilidade é mais constante em relação a cada participante, neste item é possível observar que nenhum participante fez uma avaliação muito menor que 25.

Antes de seguir para a análise do instrumento TAM, é importante averiguar os dados que foram coletados, principalmente por ser uma abordagem qualitativa. Para estimar a confiabilidade e consistência das respostas, foi utilizado a técnica do coeficiente alfa de Cronbach. No geral, o valor obtido foi de aproximadamente 0,797, considerando apenas o pilar de utilidade percebida 0,63, e o coeficiente calculado para facilidade percebida foi de 0,81. Para valores maiores que 0,7 é considerado que as sentenças estão em um nível aceitável, já valores maiores que 0,6 são questionáveis (GLIEM; GLIEM, 2003).

Figura 19 – Gráfico com os resultados de Usabilidade e Capacidade de Aprendizado da avaliação de cada participante



Fonte: Autoria própria

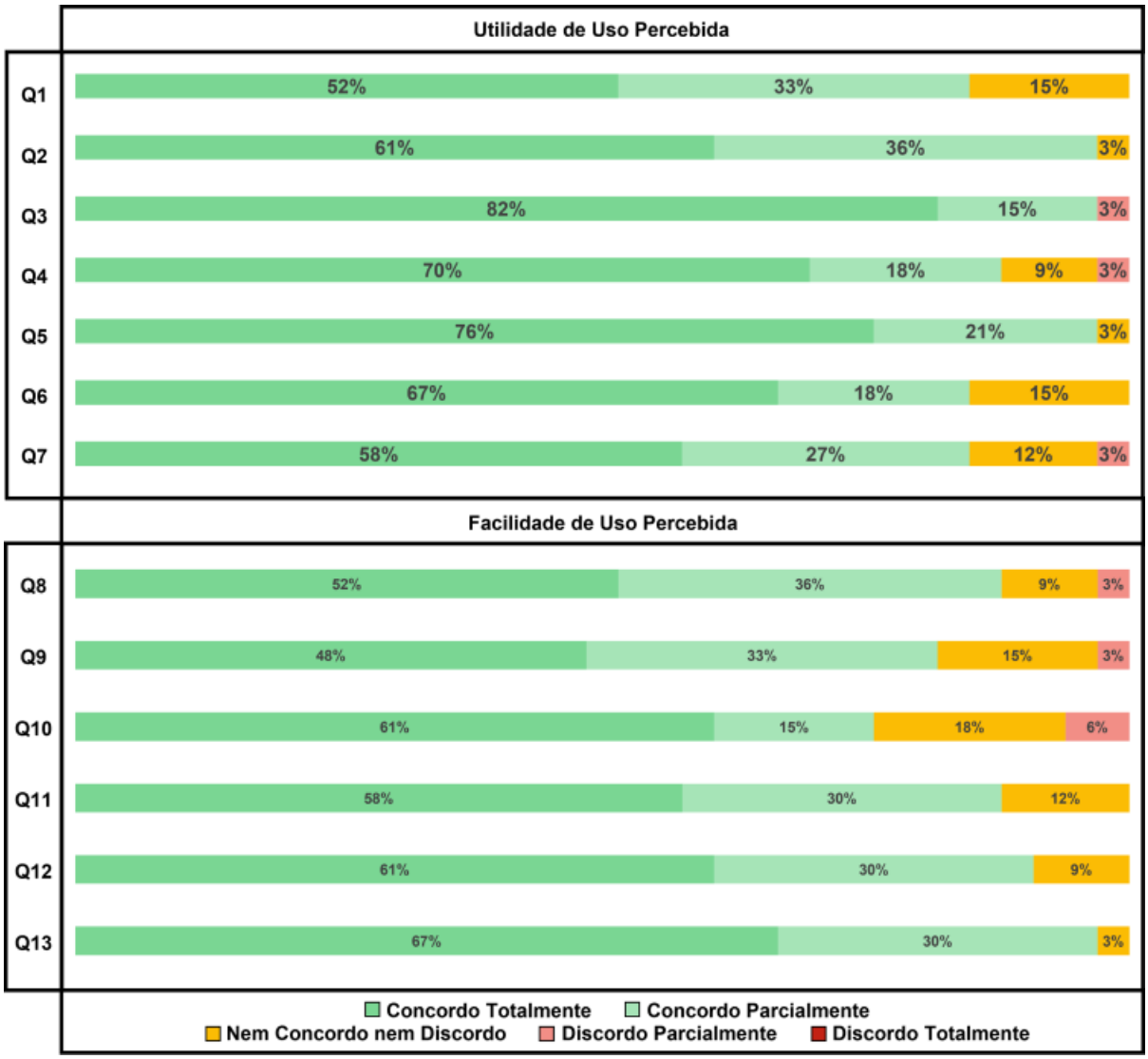
Na Figura 20 está apresentado os resultados para cada sentença baseada no TAM, agrupadas nos eixos de utilidade e facilidade percebida. Podemos observar que em nenhum item houve uma discordância absoluta, o que ressalta as pontuações satisfatórias obtidas pelo SUS. A sentença com o maior percentual foi (Q3) , que diz "Eu acredito que a ferramenta é útil.", revelando o quão a ferramenta foi bem avaliada neste aspecto.

As sentenças sobre facilidade percebida (Q8; Q13), que dizem respectivamente, "Aprender a usar o Codefab foi fácil" e "Na minha opinião, a ferramenta é fácil de usar.", mostram que a ferramenta não é tão simples. Se compararmos com a capacidade de aprendizado medida no SUS, fica claro que o *playground* não é fácil de usar em um primeiro momento, visto que exige uma adaptação do usuário no manuseio da interface.

No item (Q2), onde fala que "Os conceitos utilizados na construção de uma fábula (agentes, estados, pages, objetos de mídia) são úteis para ensinar conceitos de programação." não há discordantes. Isso demonstra a propensão da ferramenta para ser usada neste tipo de aprendizado. Para reforçar esta análise, a Tabela 5 exibe os resultados coletados no ensaio de interação, aqui chamado de Festival de Fábulas. A tabela faz uma relação com os conceitos do modelo Fábulas que aplicam os pilares de pensamento computacional.

Por fim, os usuários foram convidados a compartilhar suas impressões sobre

Figura 20 – Avaliação dos participantes nos eixos de utilidade percebida e facilidade percebida



Fonte: Autoria própria

pontos positivos e negativos sobre o Codefab. As contribuições se mostraram bastante relevantes, principalmente nos *feedbacks* de melhorias técnicas, como (i) a possibilidade de renomear a fábula, ou (ii) a limitação de *MIME types*[1] de aúdio suportados na plataforma. Além disso, a (iii) resolução do *Preview* foi considerada baixa, e (iv) a navegação do código de marcação poderia melhorar, para facilitar a navegação em projetos com muitas *Pages*.

No aspecto positivo, a (i) aparência do Codefab foi bem elogiada, já que ela facilitou as edições no documento. Aliado a isso, o item (Q6), "Funcionalidades de editores modernos (recolher e expandir regiões de código, autocompletar, highlight) aceleram a construção da fábula", demonstra que 85% concorda e não há discordantes neste ponto. Outros comentários (ii) citam como a ferramenta é bastante didática

[1] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basics_of_HTTP/MIME_types

Tabela 5 – Eixo aplicado em conceitos do Fábulas

Nome	Uso / Fábula	Eixo do Pensamento computacional
agentes	100%	Abstração; Identificação de padrões;
estados	92,86%	Abstração; Identificação de padrões;
drag and drop	78,57%	Abstração; Identificação de padrões;
multi-pages	100%	Decomposição; Algoritmos;
gatilhos	67,86%	Abstração; Identificação de padrões;
história não linear	100%	Algoritmos;

e ressaltam a importância do suporte ao usuário via guia de ajuda, sem contar (iii) da relevância do Playground para ensinar iniciantes em programação. Como bônus, tivemos opiniões sobre (iv) o incentivo ao uso de plataformas de versionamento de código, o item (Q7) "Acredito que a ferramenta incentiva a utilização de plataformas de versionamento de código (Github)", demonstra a paridade neste aspecto, onde apenas 3% discordam.

Em linhas gerais os instrumentos utilizados no processo de avaliação, tiveram resultados convergentes. Visto que, tanto a análise de facilidade percebida do TAM, quanto a capacidade de aprendizado calculada pelo SUS, mostraram que o Codefab não é uma ferramenta muito simples para se utilizar em um primeiro momento. Apesar disso, o *Playground* está bem avaliado em questões de usabilidade, com o aspecto de utilidade percebida com muita relevância, sendo indicado pelos próprios voluntários para o ensino de programação para iniciantes.

5 Conclusão

O *Playground* Codefab tinha como objetivo a criação de um ambiente para construção de narrativas interativas hipermídia não-linear. A abordagem escolhida foi a aplicação do modelo conceitual Fábulas, porém apenas de forma declarativa, utilizando um documento estruturado. Além disso, promover uma experiência durante o uso da ferramenta semelhante aos que já estão no mercado, com *Preview*, gerenciamento de assets, editor de texto embutido e outras funcionalidades.

Para avaliar a proposta deste trabalho, foram pensado dois momentos com finalidades distintas, para capturar mais de um ponto de vista da aplicação. O ensaio de interação, foi pensado para compreender a relação usuário-interface, além de coletar dados estatísticos sobre o Codefab. Fruto desta inspeção, foi possível relacionar quais eixos do pensamento computacional foram alcançados durante a construção de uma história. A pesquisa trouxe dados tanto quantitativos quanto qualitativos sobre os aspectos de usabilidade, capacidade de aprendizado, utilidade e facilidade de uso percebida.

Os resultados dos instrumentos aplicados na pesquisa mostraram a percepção do usuário em relação à interface. Através desses dados, foi possível observar que os participantes apresentaram dificuldades em relação aos primeiros contatos com a ferramenta. Esta leitura, demonstra que usuários de ferramentas com um aspecto muito técnico podem sofrer com dificuldades ao longo da sua adaptação. Contudo, isso é esperado e se trata de um *trade-off*[1] para ser possível obter resultados cada vez mais expressivos, tornando este tipo de ferramenta cada vez mais complexa.

Diante desta realidade, a usabilidade da ferramenta apresentou uma boa pontuação na análise quantitativa, juntamente com um bom percentual de respostas concordantes nas sentenças para medir a aceitação do Codefab. Como mencionado, a ferramenta atingiu boas avaliações nos percentuais de utilidade e facilidade de uso percebida, o que demonstra na prática que os participantes entenderam que a ferramenta poderia ser útil em seu objetivo. Isto pode ser reforçado pelas respostas de tema livre, onde os próprios voluntários pontuaram o emprego do *Playground* para trabalhar os fundamentos de programação.

Para ter uma compreensão do ensino de conceitos sobre programação ao longo da utilização da ferramentas, foi necessário entender a relação entre os pilares do pensamento computacional estabelecidos neste trabalho e os artifícios para construção de uma fábula. É importante ressaltar que, o presente trabalho tratou esta linha de medição do aprendizado de forma empírica, de acordo com as definições realizadas aqui, e não tratando de aspectos pedagógicos. Desta forma, foi possível estabelecer a relação dos participantes que trabalharam ao menos um dos pilares do pensamento computacional, assim, o *Playground* ajudou a aplicar os conceitos de ciência da computação durante a execução dos projetos.

No aspecto técnico, o Codefab comportou com bastante êxito todas as sessões

[1] refere-se, geralmente, a perder em um aspecto, ganhando em troca outro aspecto.

ativas e concorrentes em um dado momento, sem o aparecimento de gargalos de performance, influência direta do paradigma de desenvolvimento adotado. Isto não significa que a ferramenta estava livre de possíveis melhorias, pois de acordo com *Jakob's Law* (NIELSEN, 2021), um usuário de internet passa mais tempo em outros sites, assim ele espera que a ferramenta funcione de maneira similar ou melhor ao que ele está habituado. Desta forma, não só os critérios de avaliação foram pautados em outros *Playgrounds*, como os apontamentos feitos se referiam a tarefas que eram possíveis de serem feitas em plataformas similares.

5.1 Contribuições

A principal contribuição do trabalho são as especificações técnicas trabalhadas a partir dos requisitos discutidos na Seção 3.1. Uma solução bastante escalável construída sob o modelo serverless, contando com um bom desempenho e agilidade na operação. Os resultados coletados revelam a qualidade do que foi construído, servindo de guia para a construção sistemas neste molde.

5.2 Trabalhos futuros

O *Playground* apresentado aqui deve continuar a sofrer alterações em vários aspectos técnicos, onde essas melhorias vão de acordo com os resultados levantados na Seção 4 para cada ponto discutido. Porém, a evolução da ferramenta não é apenas em relação a usabilidade e a aceitação da mesma, visto que para melhorar o desempenho ou atender novos requisitos é necessário contemplar pontos fora da ótica do usuário. Aplicar a arquitetura Hexagonal tem como propósito dividir o código em camadas, isolando a lógica da aplicação e criando interfaces de comunicação, conhecida como *Ports and Adapters*.

Desta forma, se permite construir integrações com serviços de terceiros, que utilizam vários tipos diferentes de protocolos, mantendo a qualidade e a escalabilidade do projeto. A partir disso, a proposta é realizar a integração com a ferramenta Discord (DISCORD, 2021), para ser possível produzir recursos enquanto se comunica via chats e canais de comunidade, além de estender a publicação para outras plataformas de versionamento de código. Além disso, outra possibilidade bastante relevante é a capacidade de criação de fábulas de forma colaborativa, implementado via *sockets*.

A integração com ambientes virtuais de aprendizagem (AVA) se mostra muito relevante dado o contexto de ensino de programação através do EaD, contemplados neste trabalho. Esta proposta pode ser aplicada no cenário onde o professor cria os recurso interativos como atividade para os alunos, fazendo uso da tecnologia H5P (JOUBEL, 2021), ou onde o próprio Codefab está incorporado a interface do AVA, utilizando o protocolo LTI (GLOBAL, 2021) para realizar a comunicação entre o *Playground* e o ambiente de ensino.

Referências

BACK4APP. *Backend as a Service – O que é?* 2021. Disponível em: <https://blog.back4app.com/pt/backend-as-a-service>. Accesso em: 2021-11-02. Citado na página 21.

BARBOSA, S.; SILVA, B. *Interação humano-computador*. [S.l.]: Elsevier Brasil, 2010. Citado na página 42.

BRACKMANN, C. P. Desenvolvimento do pensamento computacional através de atividades desplugadas na educação básica. 2017. Citado 2 vezes nas páginas 13 e 14.

BROOKE, J. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, v. 189, 11 1995. Citado na página 42.

CHOI, J. An html5-based interactive e-book reader. *International Journal of Software Engineering and Its Applications*, 02 2014. Citado na página 20.

CODECOMBAT. *CodeCombat*. 2021. Disponível em: <https://codecombat.com>. Accesso em: 2021-10-22. Citado na página 13.

CODEMIRROR. *Codemirror*. 2021. Disponível em: <https://codemirror.net>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 17 e 18.

CODE.ORG. *Code.org*. 2021. Disponível em: <https://code.org>. Accesso em: 2021-10-22. Citado 2 vezes nas páginas 13 e 14.

CODINGGROUND. *CodingGround*. 2021. Disponível em: <https://www.tutorialspoint.com/codingground.htm>. Accesso em: 2021-10-22. Citado na página 13.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, ACM New York, NY, USA, v. 35, n. 6, p. 26–36, 2000. Citado na página 20.

DIAS, G. A.; SILVA, P. M. da; JR, J. B. D.; ALMEIDA, J. R. de. Technology acceptance model (tam): avaliando a aceitaçao tecnológica do open journal systems (ojs). *Informação & Sociedade*, Universidade Federal da Paraíba-Programa de Pós-Graduação em Ciência da . . . , v. 21, n. 2, 2011. Citado na página 43.

DISCORD. *Discord*. 2021. Disponível em: <https://discord.com>. Accesso em: 2021-11-02. Citado na página 50.

DOCKER. *Use containers to Build, Share and Run your applications*. 2021. Disponível em: <https://www.docker.com/resources/what-container>. Accesso em: 2021-11-02. Citado na página 20.

EBC. *Organização Mundial da Saúde declara pandemia de coronavírus*. 2021. Disponível em: <https://agenciabrasil.ebc.com.br/geral/noticia/2020-03/organizacao-mundial-da-saude-declara-pandemia-de-coronavirus>. Accesso em: 2021-10-24. Citado na página 13.

FACEBOOK. *Context API*. 2021. Disponível em: <https://reactjs.org/docs/context.html>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 17 e 18.

FACEBOOK. *React.js*. 2021. Disponível em: <https://pt-br.reactjs.org>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 17 e 18.

FIGMA. *Figma*. 2021. Disponível em: <https://www.figma.com>. Accesso em: 2021-10-21. Citado 2 vezes nas páginas 18 e 19.

FOUNDATION, O. *ESLint*. 2021. Disponível em: <https://eslint.org>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 18 e 19.

FREITAS, M.; MORAIS, P. Possibilidade de desenvolvimento do pensamento computacional por meio do code. org: aplicado ao ensino fundamental (anos iniciais). In: SBC. *Anais do XXV Workshop de Informática na Escola*. [S.l.], 2019. p. 1219–1223. Citado na página 13.

GITHUB. *GitHub*. 2021. Disponível em: <https://github.com>. Accesso em: 2021-10-21. Citado 2 vezes nas páginas 18 e 19.

GLIEM, J. A.; GLIEM, R. R. Calculating, interpreting, and reporting cronbach's alpha reliability coefficient for likert-type scales. In: MIDWEST RESEARCH-TO-PRACTICE CONFERENCE IN ADULT, CONTINUING, AND COMMUNITY . . . . [S.l.], 2003. Citado na página 45.

GLOBAL, I. *IMS LTI® 1.3 and LTI Advantage*. 2021. Disponível em: <https://www.imsglobal.org/activity/learning-tools-interoperability>. Accesso em: 2021-11-02. Citado na página 50.

GOCD.ORG. *What is serverless?* 2021. Disponível em: <https://www.gocd.org/2017/06/26/serverless-architecture-continuous-delivery>. Accesso em: 2021-11-02. Citado na página 21.

GOOGLE. *Acelere e escalone o desenvolvimento de aplicativos sem gerenciar a infraestrutura*. 2021. Disponível em: <https://firebase.google.com/products-build?hl=pt>. Accesso em: 2021-11-02. Citado na página 21.

HOTJAR. *Hotjar*. 2021. Disponível em: <https://www.hotjar.com>. Accesso em: 2021-11-04. Citado 2 vezes nas páginas 43 e 45.

JOUBEL. *CREATE, SHARE AND REUSE INTERACTIVE HTML5 CONTENT IN YOUR BROWSER*. 2021. Disponível em: <https://h5p.org>. Accesso em: 2021-11-02. Citado na página 50.

JOYCE, A. *How to Measure Learnability of a User Interface*. 2021. Disponível em: <https://www.nngroup.com/articles/measure-learnability>. Accesso em: 2021-11-03. Citado na página 42.

KONVAJS. *Konva.js*. 2021. Disponível em: <https://konvajs.org>. Accesso em: 2021-11-02. Citado na página 17.

LEWIS, J. R.; SAURO, J. The factor structure of the system usability scale. In: SPRINGER. *International conference on human centered design*. [S.l.], 2009. p. 94–103. Citado na página 42.

LUCIDSOFTWARE. *Lucidchart*. 2021. Disponível em: <https://lucid.co/pt>. Accesso em: 2021-10-21. Citado 2 vezes nas páginas 18 e 19.

MADDERN, G. *styled-components*. 2021. Disponível em: <https://styled-components.com>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 17 e 19.

@MATANSH. *react-xml-parser*. 2021. Disponível em: <https://www.npmjs.com/package/react-xml-parser>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 18 e 19.

MDN. *Javascript Lang*. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Accesso em: 2021-10-21. Citado 2 vezes nas páginas 17 e 19.

MICROSOFT. *TypeScrip Lang*. 2021. Disponível em: <https://www.typescriptlang.org>. Accesso em: 2021-10-21. Citado 2 vezes nas páginas 17 e 19.

MICROSOFT. *Visual Studio Code*. 2021. Disponível em: <https://code.visualstudio.com>. Accesso em: 2021-10-21. Citado 2 vezes nas páginas 18 e 19.

NIELSEN, J. *Jakob's Law of Internet User Experience*. 2021. Disponível em: <https://www.nngroup.com/videos/jakobs-law-internet-ux>. Accesso em: 2021-11-03. Citado na página 50.

PADLET. *Padlet*. 2021. Disponível em: <https://padlet.com>. Accesso em: 2021-11-04. Citado na página 43.

PADLET. *Padlet do Codefab*. 2021. Disponível em: <https://padlet.com/carlossalles/tdq368lpmgdpsus6>. Accesso em: 2022-01-09. Citado na página 45.

PELLAN, B.; CONCOLATO, C. Authoring of scalable multimedia documents. *Multimedia Tools and Applications*, Springer, v. 43, n. 3, p. 225–252, 2009. Citado na página 15.

PINTO, H. F. et al. Autoria de e-books interativos: modelo conceitual fábulas e requisitos. Universidade Federal do Maranhão, 2017. Citado 2 vezes nas páginas 14 e 22.

PRETTIER. *Prettier*. 2021. Disponível em: <https://prettier.io>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 18 e 19.

QUEIRÓS, R.; PINTO, M.; TERROSO, T. User experience evaluation in a code playground (short paper). In: SCHLOSS DAGSTUHL-LEIBNIZ-ZENTRUM FÜR INFORMATIK. *Second International Computer Programming Education Conference (ICPEC 2021)*. [S.l.], 2021. Citado na página 13.

REDHAT. *O que é função como serviço (FaaS)?* 2021. Disponível em: <https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-faas>. Accesso em: 2021-11-02. Citado na página 21.

REDHAT. *O que é serverless?* 2021. Disponível em: <https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-serverless>. Accesso em: 2021-11-02. Citado na página 20.

REPLIT. *Replit in-browser IDE*. 2021. Disponível em: <https://replit.com>. Accesso em: 2021-10-22. Citado na página 13.

SAURO, J. *Measuring Usability with the System Usability Scale (SUS)*. 2021. Disponível em: <https://measuringu.com/sus>. Accesso em: 2021-10-24. Citado 2 vezes nas páginas 42 e 45.

SCRATCH. *Scratch*. 2021. Disponível em: <https://scratch.mit.edu>. Accesso em: 2021-10-22. Citado 2 vezes nas páginas 13 e 14.

SILVA, A. T.; SOUZA, W. M. de; MORAES, D. de S.; AZEVEDO, R. G.; NETO, C. d. S. S. Improving the authoring of web-based interactive e-books with fablejs. In: SBC. *Anais da VII Escola Regional de Computação do Ceará, Maranhão e Piauí*. [S.l.], 2019. p. 182–189. Citado 2 vezes nas páginas 23 e 24.

UUID. *UUID*. 2021. Disponível em: <https://www.npmjs.com/package/uuid>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 18 e 20.

VAGIAS, W. M. Likert-type scale response anchors. *Clemson International Institute for Tourism & Research Development, Department of Parks, Recreation and Tourism Management. Clemson University*, 2006. Citado na página 43.

VENKATESH, V.; DAVIS, F. D. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management science*, INFORMS, v. 46, n. 2, p. 186–204, 2000. Citado na página 42.

W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2021. Disponível em: <https://www.w3.org/TR/xml>. Accesso em: 2021-11-02. Citado na página 20.

WHATWG. *localstorage API*. 2021. Disponível em: <https://html.spec.whatwg.org/multipage/webstorage.html>. Accesso em: 2021-11-02. Citado 2 vezes nas páginas 18 e 19.

WING, J. M. Computational thinking. *Communications of the ACM*, ACM New York, NY, USA, v. 49, n. 3, p. 33–35, 2006. Citado na página 14.

APÊNDICE A – Questionário de pesquisa

Questionário de avaliação do playground Codefab

Termo de Consentimento Livre e Esclarecido

Responsável pela pesquisa: A pesquisa está sendo desenvolvida por Micael Machado Gomes sob orientação do Prof. Dr. Carlos de Salles Soares Neto.

Justificativa e Objetivos da Pesquisa: Esta pesquisa tem por objetivo analisar o aprendizado dos alunos a partir do uso do Playground CodeFab. A finalidade é analisar a usabilidade da ferramenta e quantificar o nível de pensamento computacional adquirido durante a construção da história.

Procedimentos e Métodos: Os participantes da pesquisa são alunos de graduação do curso de Ciência da Computação da UFMA. Estes discentes receberão um questionário via "Google Forms", ao qual responderão de forma voluntária às perguntas contidas no questionário.

Resultados e Benefícios esperados: Os dados serão obtidos por meio do "Google Forms" e pelo Playground Codefab, não havendo exposição direta ou confronto durante o processo. Os resultados desse estudo serão utilizados na monografia da discente responsável por esta pesquisa, podendo também ser divulgados em eventos e/ou periódicos científicos da área especializados.
Será assegurado o sigilo da identidade dos respondentes ao questionário durante a análise e qualquer divulgação do estudo que for realizada.

*Obrigatório

Informações sobre você

1. Qual a sua conta do Github? (link do perfil) *

2. Qual o seu nome completo? *

3. Qual sua idade? *

4. Com qual gênero você mais se identifica?

Marcar apenas uma oval.

- ◯ Feminino
- ◯ Masculino
- ◯ Prefiro não dizer
- ◯ Outro: _____

Formulário de Pesquisa

5. Eu acho que gostaria de usar esse sistema com frequência. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

6. Eu acho o sistema desnecessariamente complexo. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

7. Eu achei o sistema fácil de usar. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

8. Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

9. Eu acho que as várias funções do sistema estão muito bem integradas. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

10. Eu acho que o sistema apresenta muita inconsistência. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

11. Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

12. Eu achei o sistema atrapalhado de usar. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

13. Eu me senti confiante ao usar o sistema. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

14. Eu precisei aprender várias coisas novas antes de conseguir usar o sistema. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

Formulário de Pesquisa

15. Acredito que a ferramenta acelera o processo de construção de fábulas *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

16. Os conceitos utilizados na construção de uma fábula (agentes, estados, pages, objetos de mídia) são úteis para ensinar conceitos de programação. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

17. Eu acredito que a ferramenta é útil. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

18. A guia de ajuda é útil para encontrar informações sobre a ferramenta *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

19. Os assets (objetos de mídia) disponibilizados na galeria são úteis para construir fábulas *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

20. Funcionalidades de editores modernos (recolher e expandir regiões de código, autocompletar, highlight) aceleram a construção da fábula *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

21. Acredito que a ferramenta incentiva a utilização de plataformas de versionamento de código (Github) *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

22. Aprender a usar o Codefab foi fácil *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

23. Construir uma fábula dentro da ferramenta é um processo simples e compreensível *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

24. Os botões de ação no editor são claros e fáceis de encontrar *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

25. Os recursos de navegação da ferramenta são simples e compreensíveis *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

26. Importar um asset (objeto de mídia) é simples e compreensível. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

27. Na minha opinião, a ferramenta é fácil de usar. *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

28. Cite pontos positivos da ferramenta, se existirem.

29. Cite pontos negativos da ferramenta, se existirem