UNIVERSIDADE FEDERAL DO MARANHÃO

Curso de Ciência da Computação

Lukas Henrique Braga Dias

Estendendo a Plataforma Codefab para Colaboração e Divulgação de Fábulas Interativas.

São Luís

2025

Lukas Henrique Braga Dias

Estendendo a Plataforma Codefab para Colaboração e Divulgação de Fábulas Interativas.

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.

Orientador: Prof. Dr. Carlos de Salles Soares Neto

São Luís

2025

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

Dias, Lukas Henrique Braga.
Estendendo a Plataforma Codefab para Colaboração e Divulgação de Fábulas Interativas / Lukas Henrique Braga Dias. - 2025.
91 f.

Orientador(a): Carlos de Salles Soares Neto.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2025.

1. Narrativa Interativa. 2. Editor de Código Web. 3. Dsl. 4. Autoria Colaborativa. I. Soares Neto, Carlos de Salles. II. Título.

Lukas Henrique Braga Dias

Estendendo a Plataforma Codefab para Colaboração e Divulgação de Fábulas Interativas.

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.

Trabalho aprovado em São Luís, 8 de agosto de 2025:

Prof. Dr. Carlos de Salles Soares Neto
Orientador

Profa. Dra. Alana de Araujo Oliveira Meireles Teixeira
Examinadora

Prof. Dr. Mario Antonio Meireles Teixeira
Examinador

São Luís
2025

Agradecimentos

À minha amada mãe, pelo apoio incondicional, pela resiliência e pelos conselhos que me guiaram à conclusão deste trabalho, ocupando com sabedoria os papéis de pai e mãe ao longo de quatorze anos da minha vida.

Ao meu falecido e amado pai, que, embora não esteja mais presente, estará sempre em minha memória.

Aos meus avós, pela resiliência na construção das famílias, ato que permitiu que eu chegasse onde estou.

Às minhas tias Elcilene, Eliane, Maria das Graças, Suely, Iracy, Rosângela, Conceição e Adha, que sempre me acolheram como a um filho.

À minha família materna e paterna, que sempre me recebeu de braços abertos, garantindo que, onde quer que eu fosse, estivesse bem amparado e rodeado de pessoas que me querem bem.

Ao meu orientador, Carlos Salles, pelos conselhos e pela sabedoria transmitidos durante a graduação, e pela força em me auxiliar na conclusão deste trabalho.

Aos meus amigos da escola e da universidade: Amely Ferreira, Luan Neves, Milena Machado, Vitor França, Flávio Gibran, Geci Helen Mesquita, Letícia Bringel, João Pedro Vale, Thúlio Uchoa, Felipe Lenida, Gilson Dias, Leonardo Lindoso, Jéssica Costa, Vinícius Costa, Carlos Costa, Alfredo Tito, Lucas Terças, Micael Gomes e Weverton Trindade, agradeço pela companhia e pelas experiências que marcaram minha trajetória.

Aos mentores que tive no começo de minha carreira profissional: Antonio Carlos, Jefferson Lima, Girresse Ribeiro, Eucylay Franco, Maçuete Austriaco e Ricardo Johannsen, agradeço pela sabedoria e pelos conselhos transmitidos no início da minha carreira profissional.

Aos docentes do curso de Computação da UFMA, pela qualidade e pelo ensino prestado à minha jornada até este trabalho.

E a todos que, de alguma forma, puderam me auxiliar e garantir que eu seguisse meu caminho de forma segura, registro aqui minha sincera gratidão.

"Não pedirei que não chorem,
pois nem todas as lágrimas são um mal."

J. R. R. Tolkien, em *O Retorno do Rei* (1955)

Resumo

Editores de código baseados na Web evoluíram rapidamente para proporcionar experiências de desenvolvimento mais práticas e eficientes. Nesse contexto, este trabalho expande um ambiente de autoria de fábulas interativas, adicionando mecanismos de colaboração e compartilhamento de recursos em um ecossistema autossustentável. Além da plataforma, apresenta-se uma nova versão da linguagem de domínio específico já proposta em estudos anteriores, agora enriquecida com abstrações algorítmicas mínimas que ampliam a expressividade dos autores. A usabilidade da solução foi avaliada por meio do questionário SUS, permitindo comparar o escopo atual com versões anteriores e identificar o impacto das funcionalidades adicionais. Paralelamente, aplicou-se o TAM para aferir a aceitação da tecnologia e verificar se os usuários consideram justificado o uso da plataforma. Os resultados mostram que a ampliação de recursos reduziu a usabilidade percebida: o SUS atribuiu nota C, inferior à obtida na versão anterior, indicando que a maior abrangência funcional introduziu barreiras adicionais ao uso, embora a intenção de continuidade e a percepção de utilidade permaneçam favoráveis, segundo o TAM.
Palavras-chave: narrativa interativa, editor de código web, DSL, autoria colaborativa.

Abstract

Web-based code editors have rapidly evolved to deliver more practical and efficient development experiences. In this context, this study extends an interactive-fable authoring environment by adding collaboration and resource-sharing features, thereby creating a self-sustaining ecosystem. The work also introduces an updated domain-specific language enriched with minimal algorithmic abstractions that enhance author expressiveness. Usability was assessed with the SUS questionnaire, allowing comparison with previous versions and identification of the impact of new functionalities. In parallel, TAM was applied to gauge overall technology acceptance and to determine whether users regard the platform's use as justified. Results reveal that the expanded feature set led to a decline in perceived usability: the SUS yielded a C score, lower than that of the earlier version, indicating that broader functionality introduced additional barriers. Nevertheless, TAM findings show favorable intention to continue use and positive perceptions of usefulness, suggesting that the platform's extended capabilities remain attractive despite increased complexity.

Keywords: interactive storytelling, web-based code editor, DSL, collaborative authoring.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
AST	*Abstract Syntax Tree*
BaaS	*Backend as a Service*
CC	*Ciência da Computação*
CDN	*Content Delivery Network*
CM6	*CodeMirror 6*
DSEL	*Domain-Specific Embedded Language*
DSL	*Domain-Specific Language*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
ID	*Identifier*
IDE	*Integrated Development Environment*
LSP	*Language Server Protocol*
PHP	*Hypertext Preprocessor*
REST	*Representational State Transfer*
SPA	*Single Page Application*
SUS	*System Usability Scale*
TAM	*Technology Acceptance Model*
UI	*User Interface*
UFMA	*Universidade Federal do Maranhão*
XML	*eXtensible Markup Language*

Sumário

1 Introdução

A modernização dos editores de código e dos Ambientes de Desenvolvimento Integrados (IDEs) trouxe recursos como sugestões contextuais, realce aprimorado da sintaxe e integração com sistemas de controle de versão(Git), elevando a experiência de desenvolvimento a novos patamares. Ferramentas como (CODESANBOX, 2025) e (STACKBLITZ, 2025) ampliaram funcionalidades antes restritas a aplicações desktop, tornando-as disponíveis em soluções web e eliminando etapas manuais de configuração de ambiente.

Ambientes de desenvolvimento na nuvem (Cloud Development Environments) consolidam essa evolução ao oferecer infraestrutura, escalabilidade automática e integração contínua, permitindo que equipes economizem tempo e recursos financeiros. Esses ambientes reduzem barreiras de entrada ao adotarem a metodologia *plug-and-play*, incentivando o início imediato da produção sem atrasos causados pela configuração do ambiente. Além disso, a disponibilização de modelos de projeto pré-configurados otimiza o reaproveitamento de iniciativas semelhantes, como exemplificado pela plataforma (CODESANDBOX, 2025).

Paralelamente, plataformas de programação visual como o Scratch demonstram que abstrações lúdicas tornam o aprendizado de conceitos de pensamento computacional mais acessível a iniciantes (SCRATCH, 2025; WING, 2006). A combinação de práticas consolidadas do mercado para otimização da experiência de desenvolvimento com ambientes interativos projeta-se como horizonte promissor para a construção de plataformas de autoria visual.

Em (GOMES, 2022) apresenta-se um *playground* web que uniu esses aspectos, edição em tempo real, abstrações de narrativa e simplicidade de uso, seguindo diretrizes iniciais propostas por (SILVA, 2020) e fundamentadas no modelo Fábulas por (PINTO, 2017). A partir da primeira linguagem de domínio específico sugerida em (SILVA, 2020), desenvolveu-se uma versão simplificada em (GOMES, 2022) e, neste trabalho, propõe-se uma nova DSL que estende as anteriores, fornecendo elementos adicionais e ferramentas de apoio.

Para ampliar a aplicação do modelo Fábulas em contextos educacionais, é preciso consolidar um ambiente de autoria que reúna edição visual integrada, colaboração social nativa e sintaxe de baixa complexidade. Ao refinar e expandir os marcos alcançados por estudos anteriores, pretende-se tornar a plataforma atraente aos autores, reduzindo barreiras iniciais e estimulando a adoção em práticas de narrativa digital.

Portanto, este trabalho apresenta uma nova iteração a esta linha de pesquisa aprimorada por (GOMES, 2022), agora com interações sociais incorporadas (antes

terceirizadas a plataformas como (PADLET, 2025)) e uma linguagem de domínio específico expandida e dedicada à criação de narrativas visuais. Além disso, essa versão inclui abstrações algorítmicas mínimas para ampliar o leque de funcionalidades disponíveis, editor com recursos de *lint* e *autocomplete* para incentivar a produtividade, e documentação interativa que oferece explicações contextuais sobre a própria DSL.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é projetar, implementar e avaliar uma nova versão do CodeFab que incorpore funcionalidades sociais nativas e suporte a uma linguagem de domínio específico própria, visando ampliar a colaboração entre autores e a expressividade na criação de fábulas interativas.

1.1.2 Objetivos Específicos

- Refatorar a arquitetura da aplicação e a interface de usuário, visando maior coesão, escalabilidade e integração entre módulos.
- Projetar e formalizar uma linguagem de domínio específico (DSL) dedicada ao CodeFab, incorporando extensões que ampliem a expressividade e facilitem a autoria de narrativas interativas.
- Implementar funcionalidades sociais nativas que promovam colaboração, compartilhamento e interação entre usuários diretamente na plataforma.
- Avaliar o impacto das novas funcionalidades na usabilidade e na aceitação da plataforma, utilizando instrumentos validados (SUS e TAM), identificando pontos de melhoria.

1.2 Estrutura da Monografia

O Capítulo 2 apresenta um histórico sintético dos estudos que originaram o CodeFab. O Capítulo 3 descreve a arquitetura e as funcionalidades da versão 2.0 da plataforma. O Capítulo 4 detalha o projeto da FableML e suas extensões ao modelo Fábulas. Em seguida, o Capítulo 5 relata um ensaio de interação conduzido com usuários, enquanto o Capítulo 6 discute os resultados obtidos. Por fim, o Capítulo 7 apresenta as conclusões e sugestões para trabalhos futuros.

2 Histórico do Projeto CodeFab

A evolução do CodeFab articula uma sequência de estudos que, entre 2016 e 2022, transformou um levantamento preliminar de requisitos em um ambiente completo de autoria de narrativas digitais. Cada pesquisa ampliou as conclusões da anterior, refinando tanto os fundamentos conceituais quanto as soluções práticas.

2.1 Padrões de Interação para Ferramentas de Autoria de Storytellings Infantis

Em (CRUZ, 2016) se investigou aplicativos de livros infantis para identificar padrões de interação relevantes à criação de e-books. Utilizando grupos focais e *card sorting*, a autora organizou categorias como Ação, Gráficos, Áudio e Navegação, delineando requisitos de usabilidade para autores e leitores.

Embora o estudo incluísse categorias fora do escopo de uma linguagem de autoria, o conjunto de requisitos serviu como ponto de partida para formalizações posteriores. A sistematização desses padrões estabeleceu um repertório que orientou a construção de um modelo conceitual dedicado à autoria de narrativas interativas.

2.2 Autoria de *e-books* Interativos: Modelo Conceitual Fábulas e Requisitos

Em (PINTO, 2017) se converteu o levantamento de (CRUZ, 2016) em um modelo declarativo denominado Fábulas, estruturado em Fables, Chapters e Pages. O autor incorporou o paradigma Evento–Condição–Ação por meio de Agentes e Propriedades, separando a intenção autoral da lógica programática e tornando a criação de histórias interativas acessível a não programadores.

A viabilidade do modelo foi verificada em navegadores web e na linguagem SceneSync (BUSSON et al., 2016). O resultado estabeleceu bases sólidas para implementações futuras.

2.3 FableJS: Biblioteca para Criação de Histórias Interativas

Em (SILVA, 2020) se implementou o FableJS com Angular e tecnologias web, materializando o modelo Fábulas em uma biblioteca prática. O trabalho introduziu

elementos adicionais, como *Board*, *Alert* e *Draggable*, reduzindo a verbosidade da linguagem declarativa e ampliando a expressividade da autoria.

Avaliações com estudantes demonstraram ganho de autoconfiança no aprendizado de programação, confirmando a pertinência pedagógica do FableJS. A biblioteca comprovou a aplicabilidade do modelo Fábulas em ambiente web e preparou o terreno para um *playground* de autoria em tempo real.

2.4 Playground Web para Aplicações Hipermídia Não-Lineares

Em (GOMES, 2022) foi introduzido o CodeFab, *playground* web, ambiente com foco no desenvolvimento de narrativas e com suporte a uma linguagem própria. O autor simplificou a DSL derivada do FableJS, diminuiu o aninhamento de tags e incorporou funcionalidades como autocompletar, realce de sintaxe e *drag and drop*.

A principal inovação reside na pré-visualização em tempo real, que executa a narrativa conforme é construída, favorecendo a experiência de desenvolvimento drasticamente. O CodeFab facilitou ainda mais a autoria de narrativas interativas, oferecendo um ambiente acessível para iniciantes e consolidando a linha de pesquisas iniciada por (SILVA, 2020).

2.5 Trabalho Atual

Prosseguindo a linha de pesquisa delineada nos estudos anteriores, o presente trabalho tem como objetivo consolidar e ampliar as contribuições de (GOMES, 2022). A nova versão do CodeFab pretende estender a plataforma já estabelecida com recursos de interação social e colaboração, ao mesmo tempo em que reintroduz funcionalidades concebidas por (SILVA, 2020) que se perderam ao longo das iterações, tais como elementos declarativos adicionais e suporte ampliado a interações multimídia. Aliada a isso, foi nomeada e formulada uma linguagem de domínio específico exclusiva para ser utilizada na aplicação, estruturada nos fundamentos estabelecidos em trabalhos anteriores, mas também estendida, buscando novas funcionalidades.

Com isso, busca-se dar continuidade natural à evolução do modelo Fábulas, unificando a criação de aplicações, formalização conceitual e construção de bibliotecas e linguagens em uma única plataforma. A iniciativa visa não apenas recuperar recursos essenciais, mas também oferecer um ambiente mais robusto e coerente para autores interessados em narrativas digitais interativas.

3 CodeFab 2.0

A iteração do CodeFab apresentada neste trabalho é chamada de CodeFab 2.0. Ela configura-se como um ambiente unificado de autoria, divulgação e colaboração de narrativas visuais. No trabalho de (SILVA, 2020), inexistia qualquer estrutura social ou de plataforma, pois o FableJS era compilado localmente, limitando a circulação das produções. Posteriormente, (GOMES, 2022) disponibilizou um *playground* web com autenticação via GitHub, o que fomentou a formação de uma comunidade. Contudo, a exposição e a busca dessas fábulas continuavam dependentes do serviço externo (PADLET, 2025), gerando dispersão de dados e dificuldades de curadoria. A plataforma se encontra disponível em (DIAS, 2025).

Este trabalho consolida os avanços desses estudos e introduz novos pilares técnicos. A aplicação foi reescrita assim como em (GOMES, 2022) em arquitetura *serverless*, empregando desta vez o Next.js e *route-handlers*. Complementarmente, o Supabase passou a armazenar metadados essenciais como turmas, murais, comentários e reações, viabilizando funcionalidades sociais e preservando o controle sobre as informações em uma base própria. Dessa forma, reúne-se o caráter de *playground* introduzido por (GOMES, 2022) com uma camada social nativa, centralizada e alinhada às necessidades propostas.

Além disso, visando ampliar as possibilidades de autoria, elaborou-se uma versão experimental de linguagem de domínio específico(DSL) voltada à criação de narrativas visuais. A proposta retoma os fundamentos implementados por (GOMES, 2022), mas introduz extensões e novas funcionalidades que respondem às demandas observadas neste estudo. O detalhamento da DSL, sua sintaxe e seus mecanismos de abstração são apresentados no Capítulo 4.

Na sequência, apresentam-se os requisitos funcionais e não funcionais da plataforma na Seção 3.1. O texto passa, então, à discussão da arquitetura adotada e das tecnologias que sustentam essa solução, esclarecendo a organização do sistema e seus principais componentes. Também são exibidos diagramas, como o modelo entidade-relacionamento e os diagramas comportamentais, que oferecem uma visão da estrutura de dados e dos fluxos internos do sistema. Por fim, o capítulo inclui as telas resultantes da interface, acompanhadas de descrições concisas das funcionalidades implementadas.

3.1 Requisitos Funcionais e Não Funcionais

Tabela 1 – Requisitos funcionais

Código	Descrição
RF-01	O sistema deve permitir que o usuário se autentique exclusivamente com sua conta GitHub.
RF-02	O sistema deve permitir que o usuário crie, leia, atualize e remova fábulas.
RF-03	O sistema deve permitir que o usuário poste uma fábula em um mural de turma somente se fornecer a senha correta dessa turma.
RF-04	O sistema deve permitir que o usuário remova uma fábula de turmas nas quais ela foi publicada.
RF-05	O sistema deve permitir que o usuário publique a mesma fábula em mais de uma turma simultaneamente.
RF-06	O sistema deve permitir que o usuário navegue no código e execute fábulas de outros usuários hospedadas em repositórios públicos.
RF-07	O sistema deve permitir que o usuário faça upload, liste e exclua assets (imagens, áudio, vídeo) vinculados à sua fábula.
RF-08	O sistema deve permitir que o usuário crie e edite arquivos `.xml` adicionais (páginas) para compor a fábula.
RF-09	O sistema deve permitir que o usuário renomeie qualquer arquivo da fábula, exceto o arquivo principal `fable.xml`.
RF-10	O sistema deve exibir uma prévia ao clicar em um asset (imagem, áudio ou vídeo).
RF-11	O sistema deve exibir uma visão em árvore da estrutura de diretórios da fábula.
RF-12	O sistema deve permitir que o usuário comente, curta e reaja a comentários em fábulas.
RF-13	O sistema deve fornecer suporte a autocomplete e linting da nova DSL em tempo real na IDE embutida.
RF-14	O sistema deve salvar automaticamente o código a cada 5 s sem interromper a edição.
RF-15	O sistema deve exibir fábulas não publicadas em nenhuma turma em seção independente no painel principal.
RF-16	O sistema deve exibir um modo teatro para visualização de todas as fábulas de uma turma em sequência.
RF-17	O sistema deve exibir um seletor de páginas no modo editor da fábula, permitindo alternar entre múltiplas páginas de forma rápida e intuitiva.

- **RF-01: Autenticação via GitHub** — O sistema deve permitir que o usuário se autentique exclusivamente com sua conta GitHub.

  Para garantir segurança e usabilidade, é necessário integrar o fluxo OAuth do GitHub de forma transparente: ao acionar o login, o usuário é redirecionado ao provedor, concede as permissões necessárias(*public_repo, read user, delete_repo*), descritas em (GITHUB, 2025b) e retorna autenticado, sem necessidade de cadastro adicional.

Essa abordagem elimina o gerenciamento de senhas próprias e assegura que apenas contas GitHub válidas acessem os recursos da plataforma.

- **RF-02: CRUD de Fábulas** — O sistema deve permitir que o usuário crie, leia, atualize e remova fábulas.
  - *Criação*: formulário de nova fábula com validação de campos obrigatórios (nome e descrição) e seleção de categoria.
  - *Leitura*: listagem paginada com filtros por autor, categoria ou turma.
  - *Atualização*: edição no editor XML, com *preview* em tempo real.
  - *Remoção*: confirmação de exclusão exibida em modal.

- **RF-03: Publicação em Turmas** — O sistema deve permitir que o usuário publique uma fábula em um mural de turma somente após fornecer a senha correta.

  Ao inserir a senha, o sistema valida a informação no servidor e associa a fábula ao mural correspondente. Em caso de falha, exibe uma mensagem clara de "senha incorreta". Essa verificação server-side impede a exposição de murais privados e garante controle colaborativo, mesmo que qualquer usuário autenticado possa visualizar fábulas publicadas.

- **RF-04: Remoção de Fábula de Turma** — O sistema deve permitir que o usuário remova uma fábula das turmas nas quais ela foi publicada.

  A interface exibe todas as turmas associadas à fábula com um botão de "remover". Ao confirmar, o vínculo é excluído sem apagar a fábula do repositório, mantendo a opção de republicação posterior.

- **RF-05: Multi-publicação em Turmas** — O sistema deve permitir que o usuário publique a mesma fábula em mais de uma turma simultaneamente.

  No formulário de publicação, o autor pode selecionar novas turmas e, após a confirmação da senha, o servidor associa a fábula às turmas escolhidas.

- **RF-06: Visualização de Código de Outros Usuários** — O sistema deve permitir que o usuário navegue no código e execute fábulas de outros usuários hospedadas em repositórios públicos.

  Para cada fábula postada, o editor exibe a árvore de arquivos, o código-fonte e o *preview* correspondente. O motor de renderização carrega o XML e os *assets*, favorecendo o aprendizado por meio do exemplo.

- **RF-07: Gerenciamento de Assets** — O sistema deve permitir que o usuário faça *upload*, listagem e exclusão de *assets* (imagem, áudio, vídeo) vinculados à sua fábula.

O componente de gerenciamento de *assets* deve exibir barras de progresso, validar tipo e tamanho dos arquivos, atualizar a lista em tempo real e vincular as URLs diretamente ao XML da fábula.

- **RF-08: Criação e Edição de Pages** — O sistema deve permitir que o usuário crie e edite arquivos *.xml* adicionais (pages) para compor a fábula.

  Cada nova página deve aparecer na árvore de arquivos com nome editável e, ao ser selecionada, abrir um editor de texto com um *template* inicial. Alterações devem ser refletidas no *preview* sem recarregar a página inteira.

- **RF-09: Renomear Arquivos** — O sistema deve permitir que o usuário renomeie qualquer arquivo da fábula, exceto o arquivo principal *fable.xml*.

  Ao clicar duas vezes no nome do arquivo, inicia-se a renomeação. Ao pressionar Enter ou clicar fora do elemento, o sistema atualiza todas as referências internas no XML e no JSON de metadados, garantindo integridade e evitando links quebrados.

- **RF-10: Preview de Asset** — O sistema deve exibir uma prévia ao clicar em um *asset* (imagem, áudio ou vídeo).

  Este requisito complementa o RF-07, permitindo que o usuário visualize detalhadamente os arquivos enviados.

- **RF-11: Árvore de Arquivos** — O sistema deve exibir uma visão em árvore da estrutura de diretórios da fábula.

  A árvore deve permitir expandir e colapsar pastas, além de indicar o status de salvamento (ex: opacidade reduzida para itens em carregamento ou edição).

- **RF-12: Comentários e Reações** — O sistema deve permitir que o usuário comente, curta e reaja a comentários em fábulas.

  Cada fábula pode receber comentários, e qualquer comentário pode receber reações (curtidas, emojis), com contadores atualizados em tempo real.

- **RF-13: *Autocomplete* e *Linting* da DSL** — O sistema deve fornecer suporte a *autocomplete* e *linting* da DSL em tempo real na IDE embutida.

  Utilizando a biblioteca CodeMirror, o editor deve sugerir tags, atributos e exibir erros de sintaxe ou semântica à medida que o usuário digita.

- **RF-14: Salvamento Automático** — O sistema deve salvar automaticamente o código a cada 1 minuto, sem interromper a edição.

  O mecanismo de *autosave* opera em *background*, mantendo um histórico de versões e exibindo indicadores visuais discretos. Como cada alteração é tratada como um *commit* no repositório, deve-se atualizar apenas o novo SHA referente ao arquivo alterado, com sincronização imediata do conteúdo.

- **RF-15: Visualização de Fábulas Não Publicadas** — O sistema deve exibir fábulas não publicadas em nenhuma turma em uma seção separada no painel principal.

  Nessa seção, cada fábula deve apresentar título, descrição, data de criação e categoria, com opções de filtragem e ordenação para facilitar o acesso rápido aos projetos individuais.

- **RF-16: Modo Teatro** — O sistema deve exibir um modo teatro para visualização de todas as fábulas de uma turma em sequência.

  O objetivo do modo teatro é proporcionar transições mais simples entre fábulas, otimizando a experiência em ambientes educacionais e expositivos, com foco na usabilidade. A interface deve incluir uma barra de ferramentas e atalhos de teclado configurados para facilitar a navegação.

- **RF-17: Seletor de Páginas no Editor** — O sistema deve exibir um seletor de páginas no modo editor da fábula, permitindo alternar entre múltiplas páginas de forma rápida e intuitiva.

  Essa funcionalidade é essencial para depuração e navegação fluida durante o processo de escrita, evitando recarregamentos e garantindo visibilidade integral da estrutura narrativa da fábula.

Considerando a natureza prática do projeto e o foco em validação funcional, optou-se por pular etapas de prototipação estática e desenvolver diretamente o MVP da interface, utilizando a própria aplicação como meio de teste e refinamento.

- **RNF01: O sistema deve ser feito com foco na Web, sendo uma SPA** — Por questões de acessibilidade e facilidade de uso, a plataforma foi concebida como uma aplicação Web do tipo SPA[1], eliminando etapas de configuração para o usuário final.

- **RNF02: O sistema deve executar nos mais usados e modernos navegadores de internet** — Considerando que a única forma de acesso ao sistema se dá via navegador, é imprescindível garantir compatibilidade ampla com os navegadores mais populares[2].

- **RNF03: O sistema deverá estar disponível 99% do tempo** — A meta de 99% de disponibilidade será alcançada com a utilização da Vercel[3] como provedora

[1] Single Page Application — Aplicação de Página Única. Trata-se de uma aplicação que carrega uma única página HTML e atualiza dinamicamente seu conteúdo conforme a navegação, sem recarregamento completo.

[2] Como Google Chrome, Microsoft Edge, Mozilla Firefox, entre outros. A compatibilidade pode ser assegurada por meio da ferramenta *browserslist* — <https://github.com/browserslist/browserslist>

[3] Plataforma de hospedagem voltada a aplicações Web modernas com integração nativa ao Next.js(VERCEL. . . ,).

de hospedagem, aproveitando sua arquitetura FaaS[4] e recursos otimizados para aplicações desenvolvidas com Next.js[5].

A sinergia entre React e Next.js permite que o mesmo conjunto de componentes seja renderizado tanto no cliente quanto no servidor, propiciando melhor desempenho e otimização para mecanismos de busca. Além disso, a familiaridade do mercado com React facilita a manutenção futura do projeto e a incorporação de novos colaboradores, garantindo continuidade ao desenvolvimento da plataforma.

3.2 Tecnologias do Frontend

O desenvolvimento da interface do CodeFab fundamenta-se na biblioteca (REACT, 2025), adotada por sua abordagem declarativa, componentização e ampla difusão na comunidade de desenvolvimento front-end. Essa escolha estabelece a base para o Next.js, descrito na Seção 3.3.1, pois o *framework* estende o React com um sistema de roteamento baseado em arquivos e opções avançadas de renderização. A utilização de *hooks* e contextos reduz a complexidade de gerenciamento de estado e favorece a reutilização de componentes, elementos essenciais em uma plataforma que demanda editor de código, painéis de navegação e *players* integrados.

3.2.1 Tailwind CSS

A interface gráfica do CodeFab emprega o Tailwind CSS, um *framework* utilitário que disponibiliza classes pré-configuradas e organizadas de maneira atômica(TAILWIND, 2025). Essa estratégia promove melhor experiência de desenvolvimento, maior consistência do código e reduz o tempo gasto na manutenção de arquivos CSS tradicionais. Segundo (ROOK, 2023), a abordagem atômica do Tailwind acelera o desenvolvimento ao limitar-se a classes primitivas, embora permita a criação de utilitários personalizados quando necessário. A integração com o Next.js requer configuração mínima, e o processo de compilação remove automaticamente as classes não utilizadas, resultando em um pacote de estilos enxuto que contribui para tempos de carregamento menores.

3.2.2 Zustand

O gerenciamento de estado global é realizado com a biblioteca Zustand, escolhida por sua sintaxe enxuta, baixa complexidade e capacidade de funcionar dentro e fora do ciclo de vida do React(ZUSTAND, 2025). De acordo com (GESTEVES, 2024), o Zustand

[4] Function as a Service, modelo de computação em nuvem baseado na execução sob demanda de funções, dispensando o gerenciamento de servidores.

[5] Framework baseado em React.js utilizado para construir aplicações Web com renderização híbrida (cliente e servidor).

fornece uma visão focada em simplicidade e performance, permitindo abordagens bem menos verbosas que bibliotecas similares como MobX e Redux. No contexto atual, a biblioteca armazena informações essenciais, como conteúdo em edição, preferências de usuário e dados da DSL, mantendo as mutações centralizadas e previsíveis, o que simplifica a depuração e melhora a coesão da aplicação.

3.3 Arquitetura *Serverless* e Infraestrutura de Hospedagem

O termo *serverless* define uma estratégia de computação em nuvem na qual o provedor se responsabiliza por toda a camada de infraestrutura, cabendo ao desenvolvedor apenas a implementação da lógica de negócio (REDHAT, 2025). No CodeFab, tal arquitetura já havia sido aplicada anteriormente, porém sem a presença do *framework* descrito no item 3.3.1. O *framework* mantém integração nativa com a plataforma Vercel. Essa sinergia permitiu que o ciclo de integração e entrega contínua fosse configurado com simplicidade, dispensando ajustes manuais de escalonamento, monitoramento ou provisionamento de servidores. Dessa forma, a escolha do Next.js resultou, de modo lógico, na adoção automática de uma infraestrutura *serverless*, conferindo robustez à esteira de implantação sem adicionar complexidade operacional.

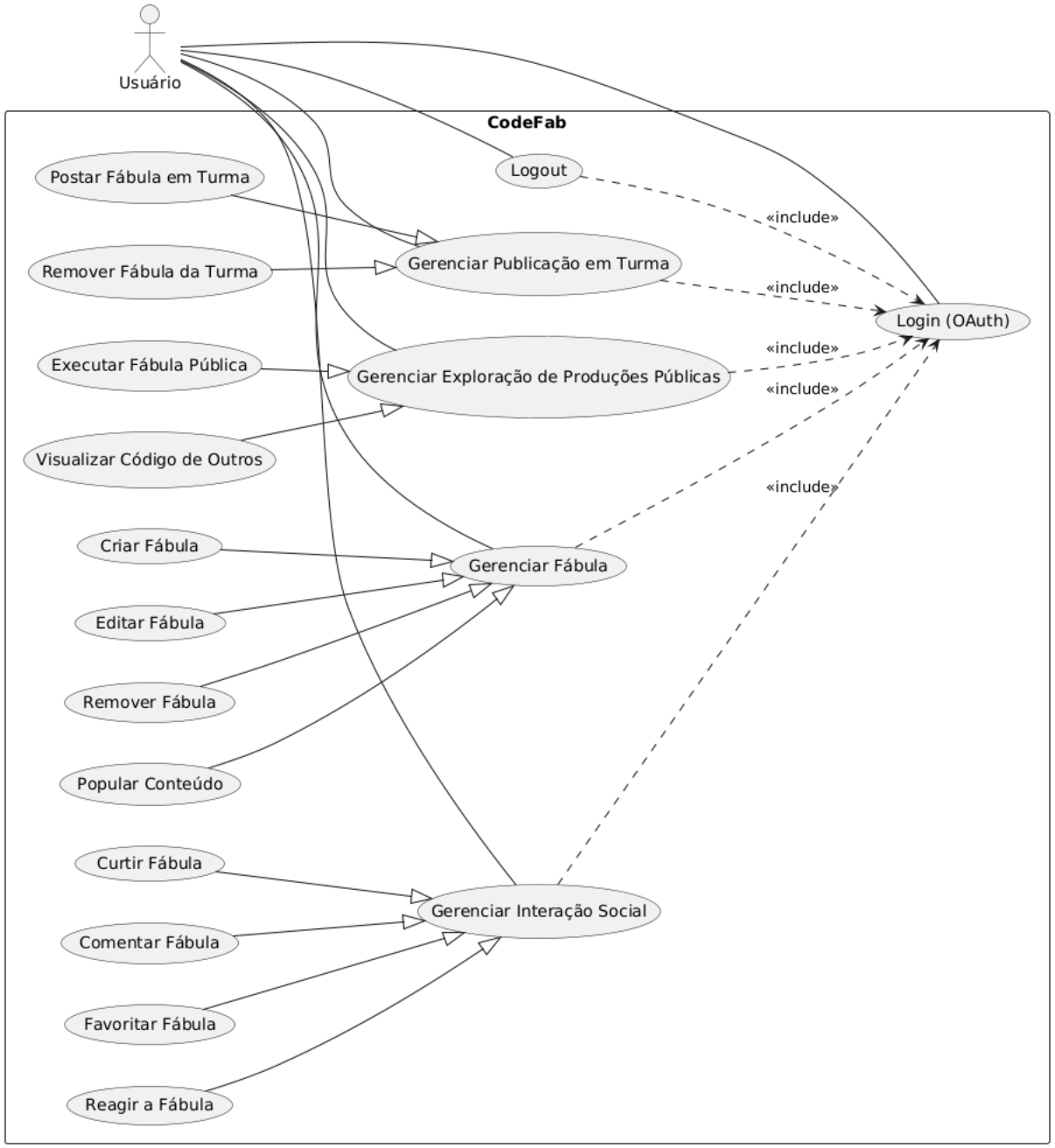
3.3.1 Emprego do Next.js

O Next.js, mantido pela Vercel, combina React com roteamento baseado em arquivos e admite múltiplas estratégias de renderização, abrangendo geração estática e renderização no momento da requisição (VERCEL, 2025). Três fatores orientaram sua adoção no CodeFab. Em primeiro lugar, o ecossistema apresenta alto grau de maturidade, sustentado por documentação abrangente e comunidade ativa. Em segundo, a difusão do *framework* no mercado facilita a manutenção futura por novos colaboradores e assegura conformidade com práticas consolidadas. Por fim, o suporte nativo a rotas de API incorporadas ao front-end reproduz a abordagem já explorada por Gomes (GOMES, 2022). Essa combinação de características proporcionou um ambiente de desenvolvimento coeso, com um fluxo de integração e entrega contínua robusto e de fácil configuração, alinhado às exigências de disponibilidade e desempenho do CodeFab.

3.4 Modelagem do Sistema

Na Figura 1, visualizamos como o sistema centraliza a autoria das histórias no usuário autenticado, que pode criar, modificar e compartilhar fábulas com facilidade usando o editor visual da plataforma e os recursos integrados ao GitHub.

Figura 1 – Diagrama de caso de usos do Codefab



Fonte: Elaborado pelo Autor

A partir do caso de uso geral, é viável formar uma ideia inicial de como o sistema opera e quais funcionalidades ele abrange. No entanto, dada a limitação dessa abordagem em representar cenários mais complexos, optou-se pela criação de diagramas auxiliares que ilustram ações detalhadas e caminhos condicionais da aplicação.

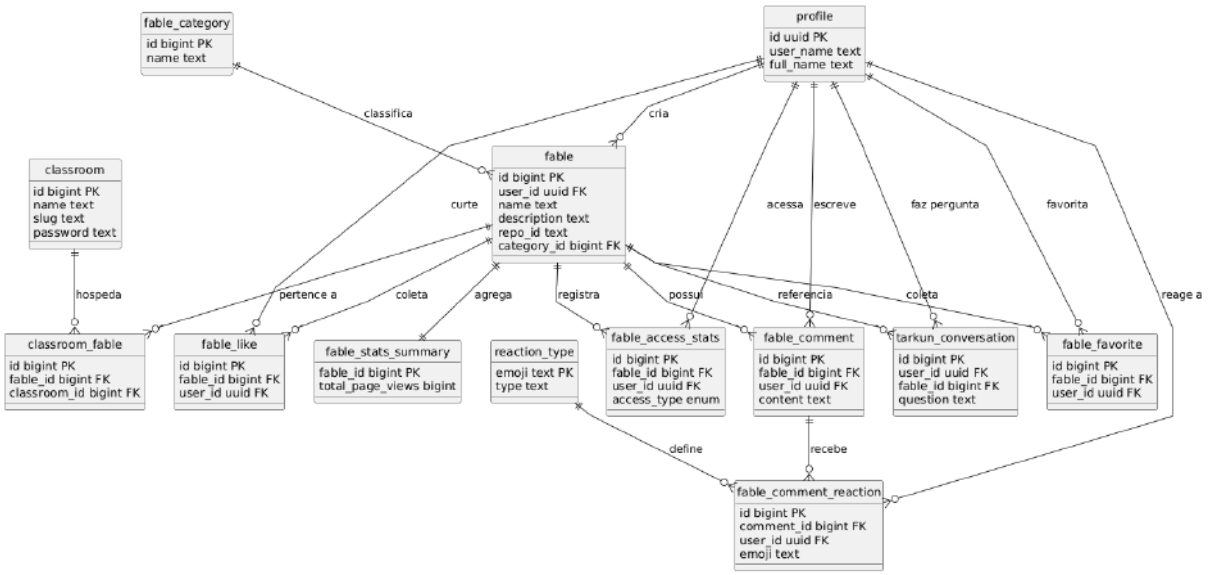
3.4.1 Modelagem de Dados

O sistema adota o PostgreSQL como gerenciador de banco de dados, disponibilizado por meio do Supabase, classificado como *Backend as a Service* (BaaS), o que simplifica a infraestrutura de dados e complexidades de manutenção(SUPABASE, 2025a).

O módulo Supabase Auth oferece autenticação baseada em OAuth e integra-se diretamente a provedores como o GitHub, solução análoga à empregada por (GOMES, 2022). As credenciais ficam armazenadas na tabela *users* e são replicadas para a tabela *profile* por gatilhos, preservando informações sensíveis em um esquema inacessível aos usuários comuns e expondo à aplicação apenas os dados necessários à interface.

A modelagem relacional organiza-se em torno da entidade *fable*, cuja posição central pode ser observada na Figura 2. Esse registro agrupa os metadados da narrativa interativa e estabelece relacionamentos com categorias, turmas, comentários, curtidas e estatísticas de acesso. As associações entre fábulas e turmas são mantidas em *classroom_fable*. Estatísticas agregadas ficam em *fable_stats_summary*, enquanto o histórico detalhado de cada acesso é registrado em *fable_access_stats*, recurso que auxilia a análise de visualizações em modo editor ou modo player.

Figura 2 – Banco de dados da aplicação



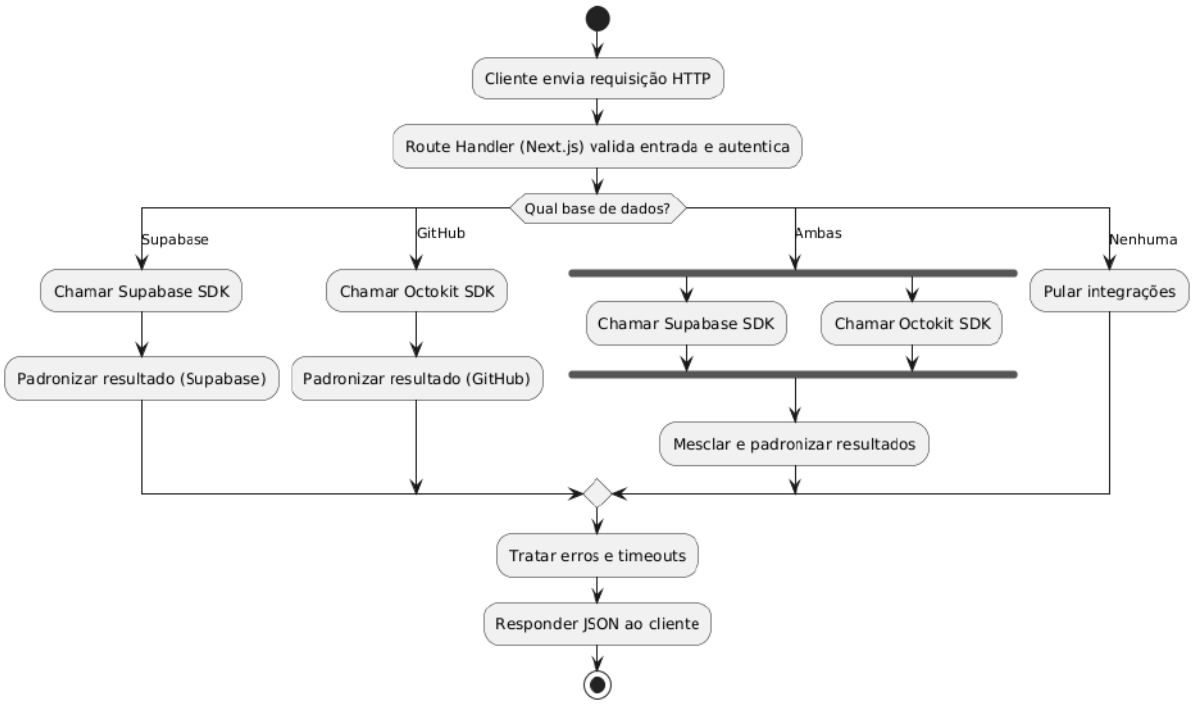
Fonte: Elaborado pelo Autor

3.4.2 Modelagem Comportamental

Esta seção apresenta diagramas UML que descrevem os comportamentos mais complexos do sistema. Inicia-se com diagramas de sequência e fluxogramas que representam, de forma genérica, todas as chamadas HTTP realizadas pela aplicação, possibilitando

compreender a estrutura dessas requisições. Em seguida detalha-se o fluxo de cadastro e de publicação de fábulas. Esses processos exigem coordenação entre duas fontes de dados, pois utilizam simultaneamente a API do GitHub e o banco relacional mantido no Supabase. Os demais comportamentos da plataforma são mais simples, uma vez que interagem com apenas uma base de dados por operação. A Figura 3 ilustra ambos os cenários, comparando o fluxo que se comunica com as duas bases com aquele restrito ao Supabase ou ao GitHub.

Figura 3 – Fluxograma de chamadas HTTP dentro do Codefab



Fonte: Elaborado pelo Autor

3.4.2.1 Login via Github

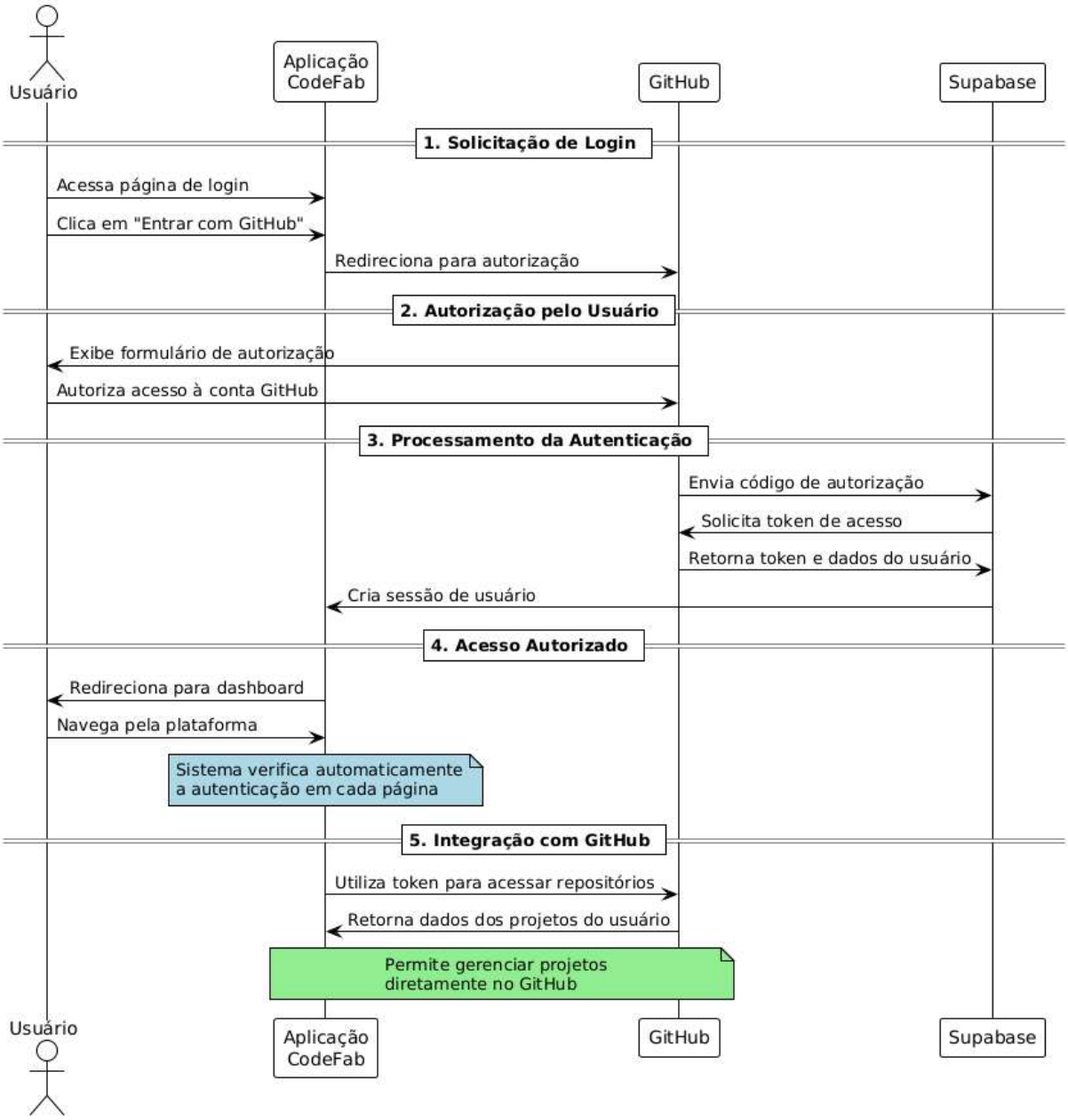
O protocolo de login adotado é o OAuth, padrão que possibilita a uma aplicação obter acesso limitado a recursos protegidos em nome do usuário, sem que as credenciais sejam compartilhadas com o serviço solicitante (HARDT, 2012). Assim, o usuário concede permissão ao sistema enquanto suas informações de autenticação permanecem sob responsabilidade exclusiva do provedor. Com essa autorização, a aplicação consegue criar e manipular repositórios no GitHub em nome do usuário, tratando cada fábula como um repositório Git convencional e, consequentemente, possibilitando versionamento, colaboração e rastreamento de alterações de maneira transparente.

A Figura 4 mostra o fluxo de autenticação implementado com o GitHub, baseado nesse mesmo protocolo OAuth. Embora o GitHub forneça autenticação e armazenamento de código, algumas funcionalidades sociais e metadados necessários ao CodeFab não são

contemplados por sua API. Por esse motivo, integra-se um banco de dados próprio, mantido no Supabase, para gerir turmas, murais e demais relacionamentos.

Conforme descrito em (GOMES, 2022), o processo de autenticação inicial baseou-se na documentação oficial do GitHub. Na solução apresentada, o Supabase atua como intermediador (SUPABASE, 2025b), centralizando tanto a autenticação quanto o armazenamento de metadados. As credenciais de usuário são armazenadas na base de dados, em substituição ao uso do local storage do navegador.

Figura 4 – Diagrama de sequência do fluxo de autenticação utilizando Supabase Auth.



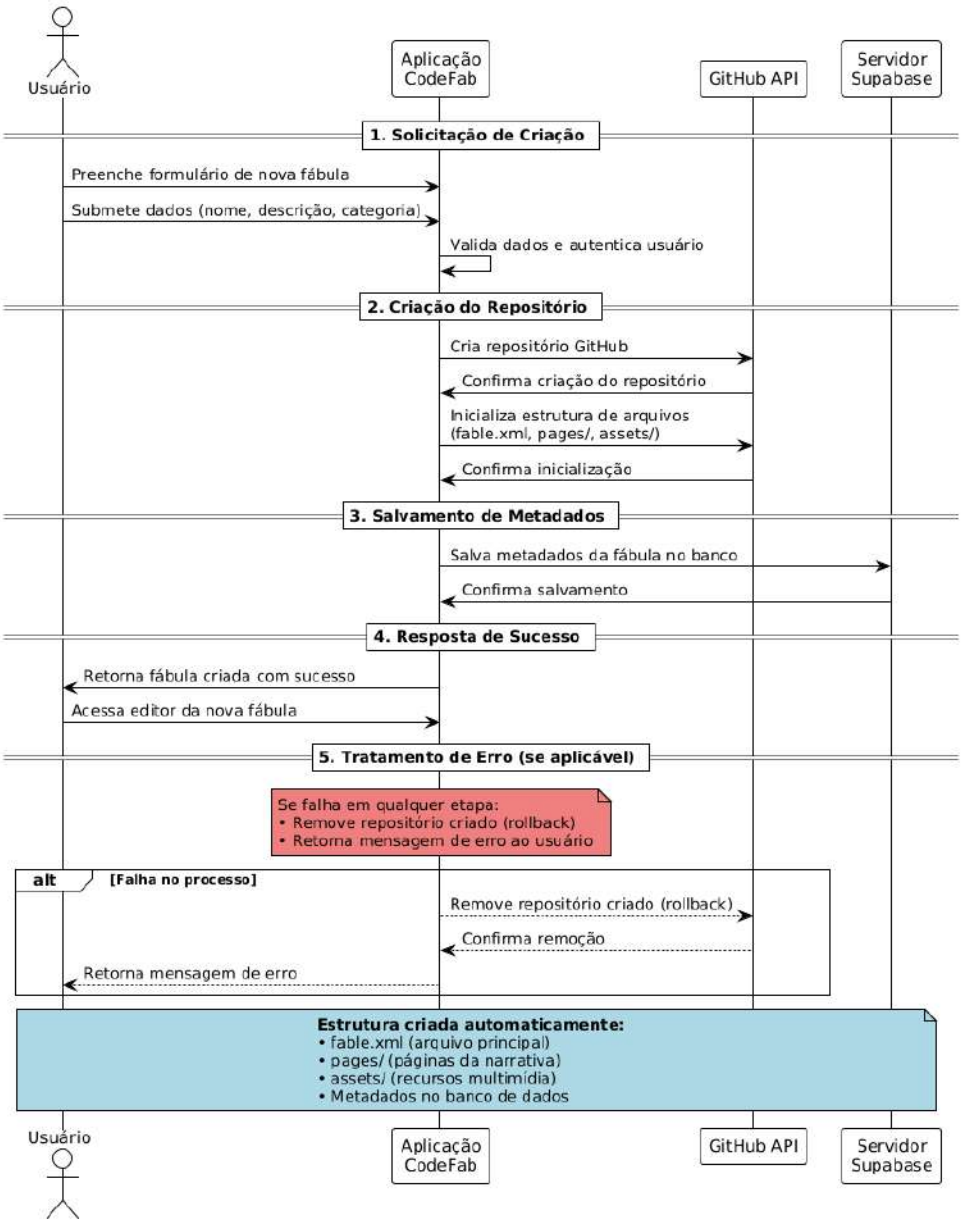
Fonte: Elaborado pelo autor

3.4.2.2 Criação de uma Fábula no CodeFab

A Figura 5 ilustra os passos para a criação de uma fábula na plataforma. Na versão anterior descrita em (GOMES, 2022), todo o fluxo de publicação era realizado exclusivamente por meio das APIs do GitHub. Nesta nova abordagem, os metadados são armazenados em um banco de dados próprio, viabilizando relacionamentos e, por consequência, funcionalidades sociais, tais como curtidas, comentários e reações.

Além da criação do repositório, a plataforma organiza automaticamente a estrutura do projeto e retorna uma instância da fábula ao cliente. Em seguida, o usuário é redirecionado ao editor de código para dar início ao desenvolvimento.

Figura 5 – Diagrama de sequência para criação de uma fábula dentro do Codefab.

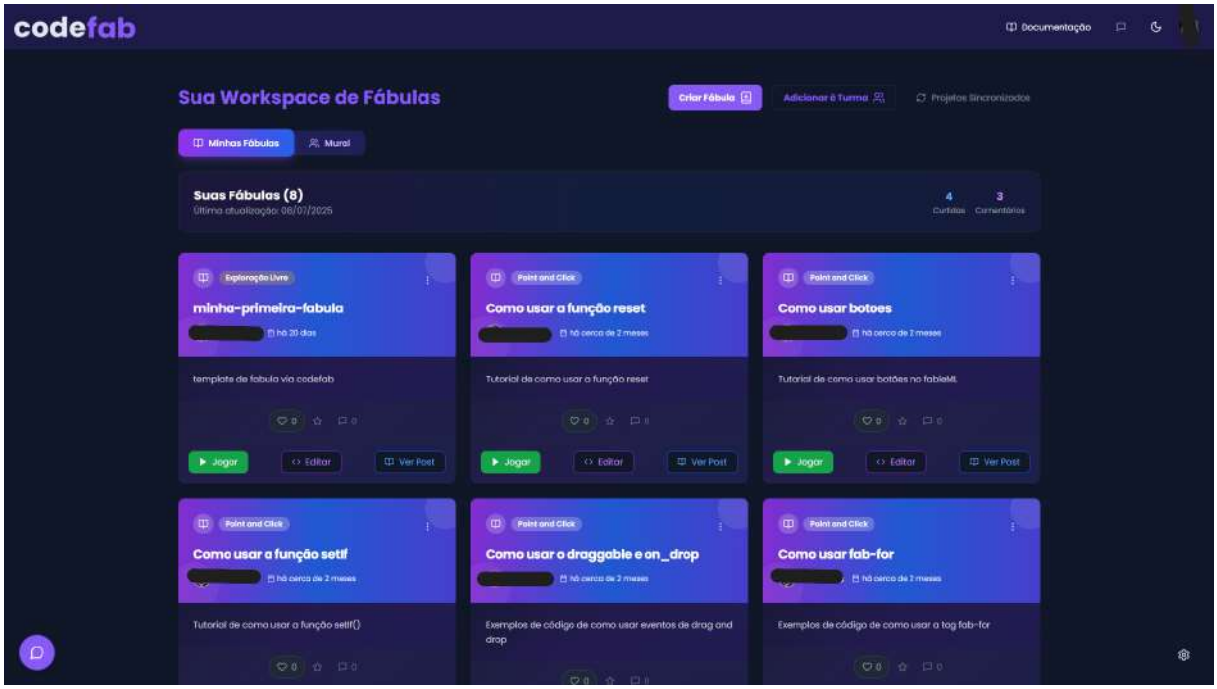


Fonte: Elaborado pelo autor

3.5 Demonstração da Plataforma

O painel principal do CodeFab, exibido na Figura 6, constitui o ponto de partida para cada usuário. Nesse *dashboard* é possível listar as fábulas de autoria própria, consultar turmas já criadas, gerar novas fábulas e vinculá-las às turmas existentes.

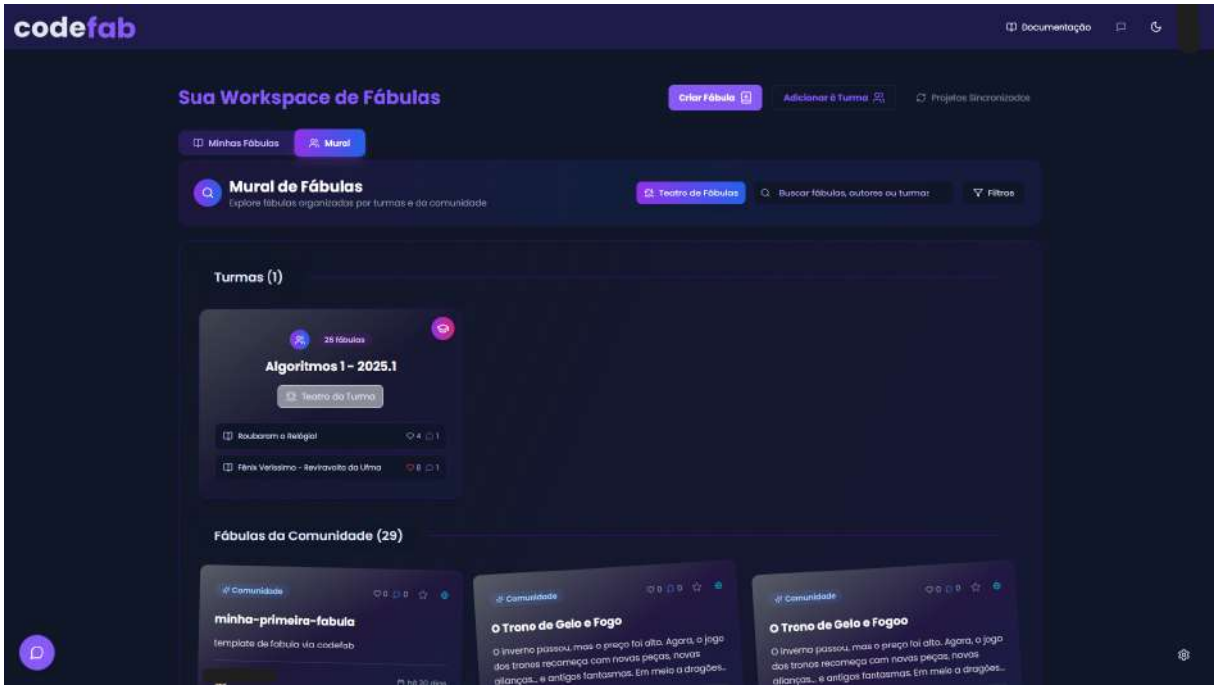
Figura 6 – Painel principal do CodeFab



Fonte: Elaborado pelo autor

Sempre que uma fábula é publicada, ela passa automaticamente para a seção Comunidade e recebe a respectiva marcação, diferindo das produções vinculadas a turmas. O mural interno, mostrado na Figura 7, evidencia essa distinção e facilita a navegação entre conteúdos pessoais e coletivos.

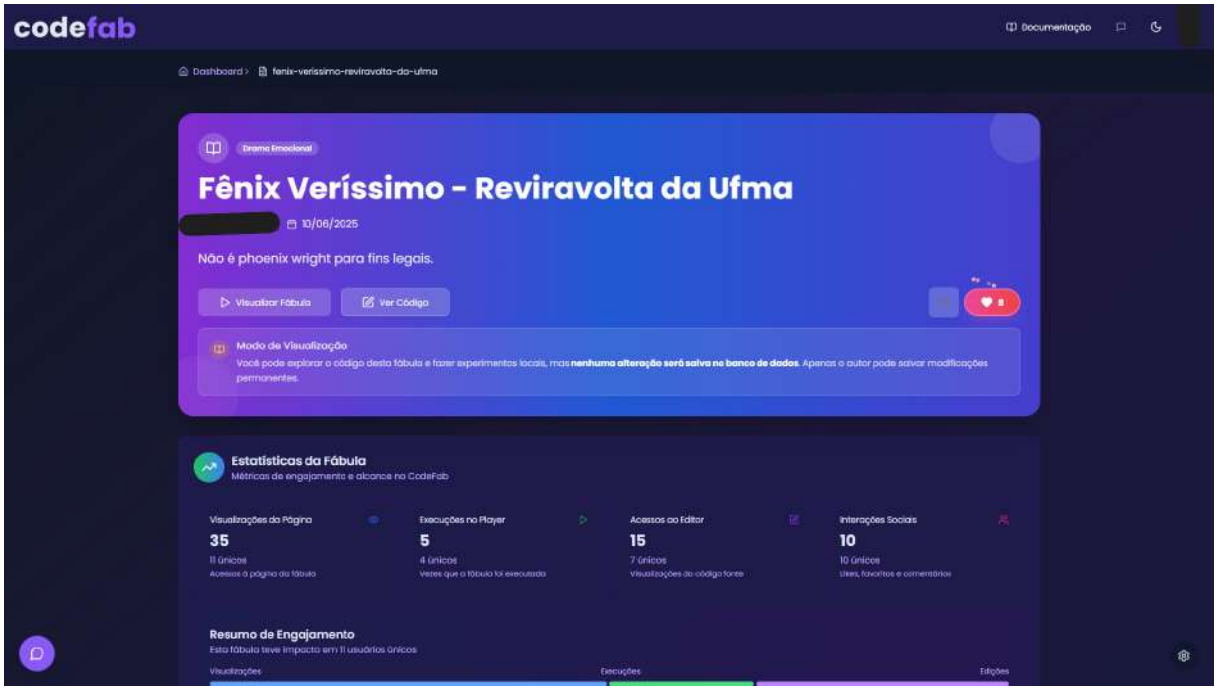
Figura 7 – Mural interno do CodeFab



Fonte: Elaborado pelo autor

Ao clicar em uma fábula, o usuário inicialmente a visualiza como uma publicação, com indicadores de interação que exibem quantidade de curtidas e comentários, conforme ilustrado na Figura 8.

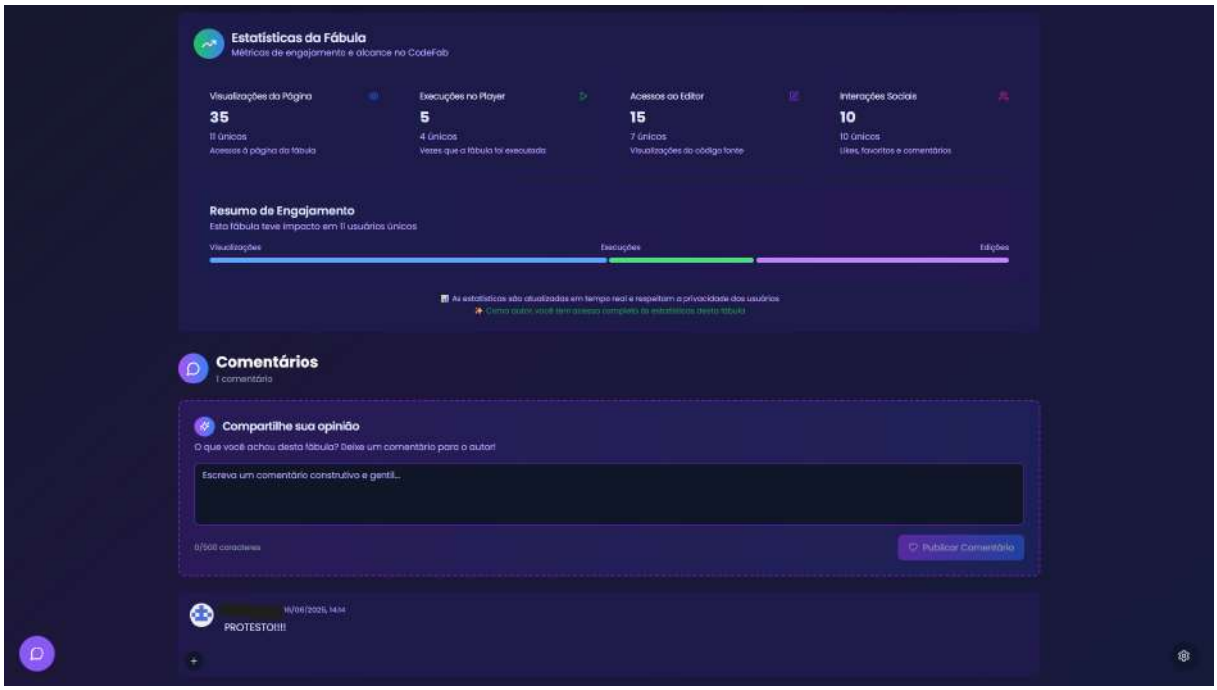
Figura 8 – Visão de uma publicação da fábula no CodeFab



Fonte: Elaborado pelo autor

Na mesma tela estão disponíveis os detalhes de engajamento, possibilitando a leitura dos comentários já registrados e a inserção de novas interações. Essa etapa encontra-se representada na Figura 9.

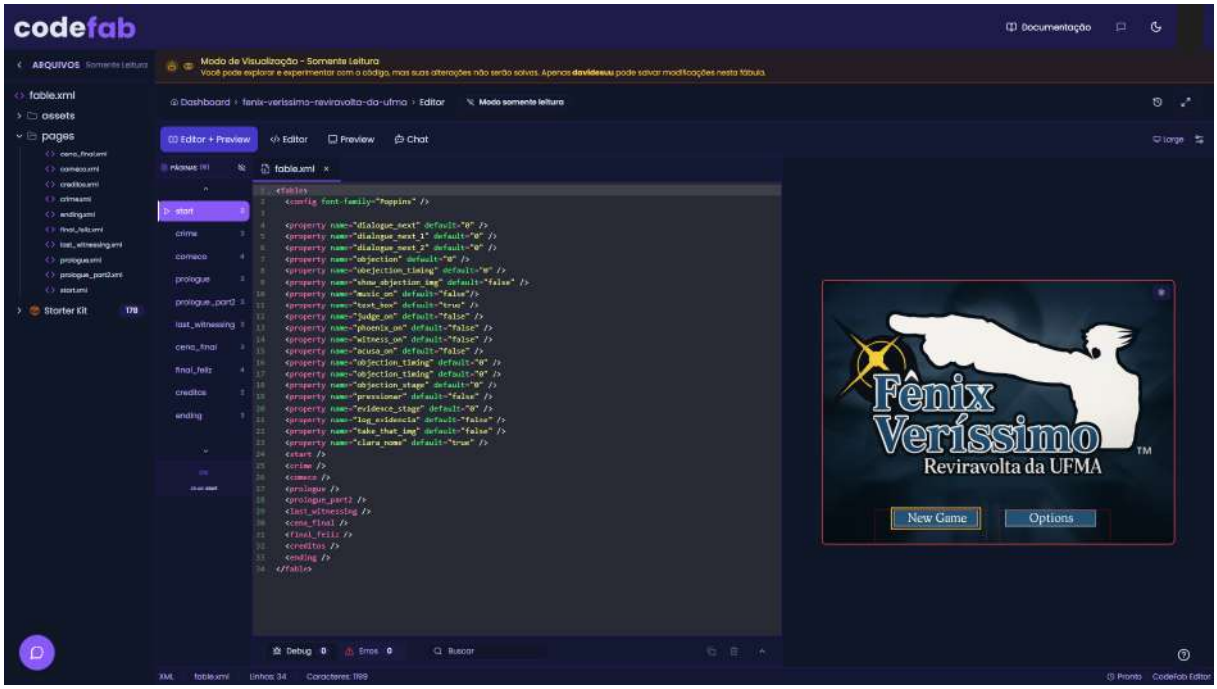
Figura 9 – Visão de comentários e engajamento da fábula



Fonte: Elaborado pelo autor

A partir dessa publicação, o usuário pode seguir dois fluxos distintos. O primeiro permite a inspeção do código-fonte da fábula, apresentado em ambiente de edição com recursos de sintaxe e versionamento, como visto na Figura 10.

Figura 10 – Visão do editor de código da fábula



Fonte: Elaborado pelo autor

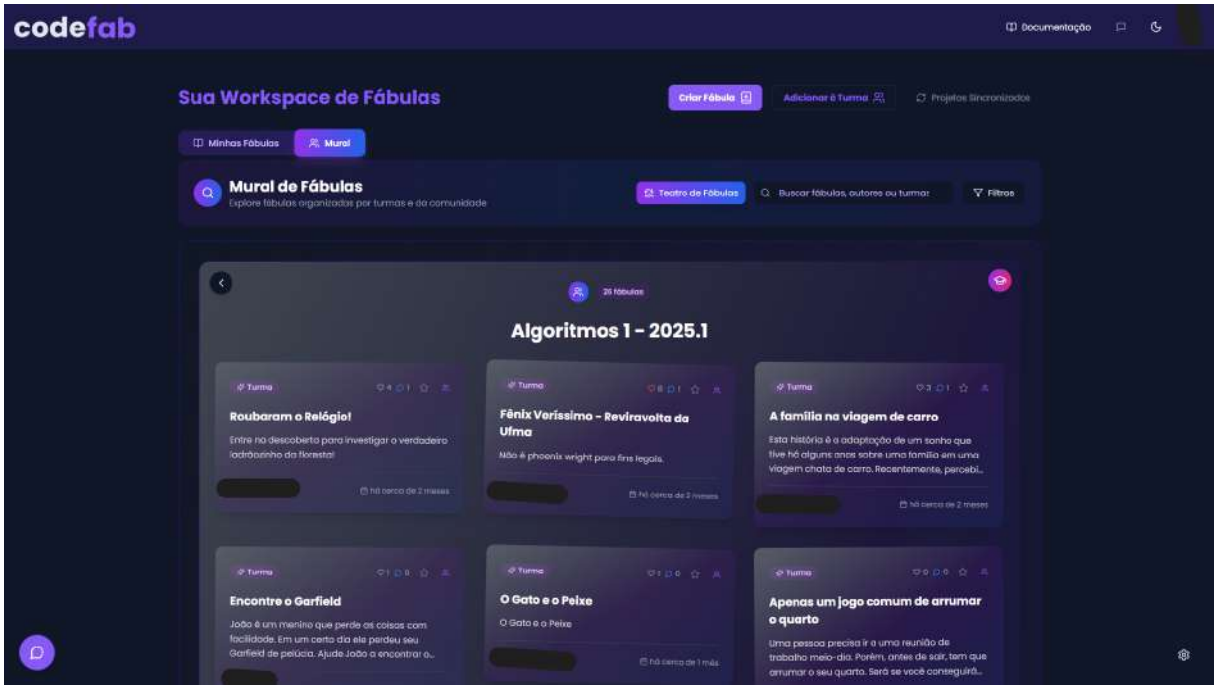
O segundo fluxo disponibiliza uma pré-visualização interativa, na qual o leitor experimenta a narrativa sem acessar o código, ilustrada na Figura 11.

Figura 11 – Visão de prévia interativa da fábula

Fonte: Elaborado pelo autor

Após a publicação de uma fábula em determinada turma, os trabalhos ficam agrupados na página da turma, conforme ilustrado na Figura 12. Esse ambiente apresenta todas as produções vinculadas ao grupo, permitindo que o usuário identifique rapidamente as fábulas próprias e as de colegas.

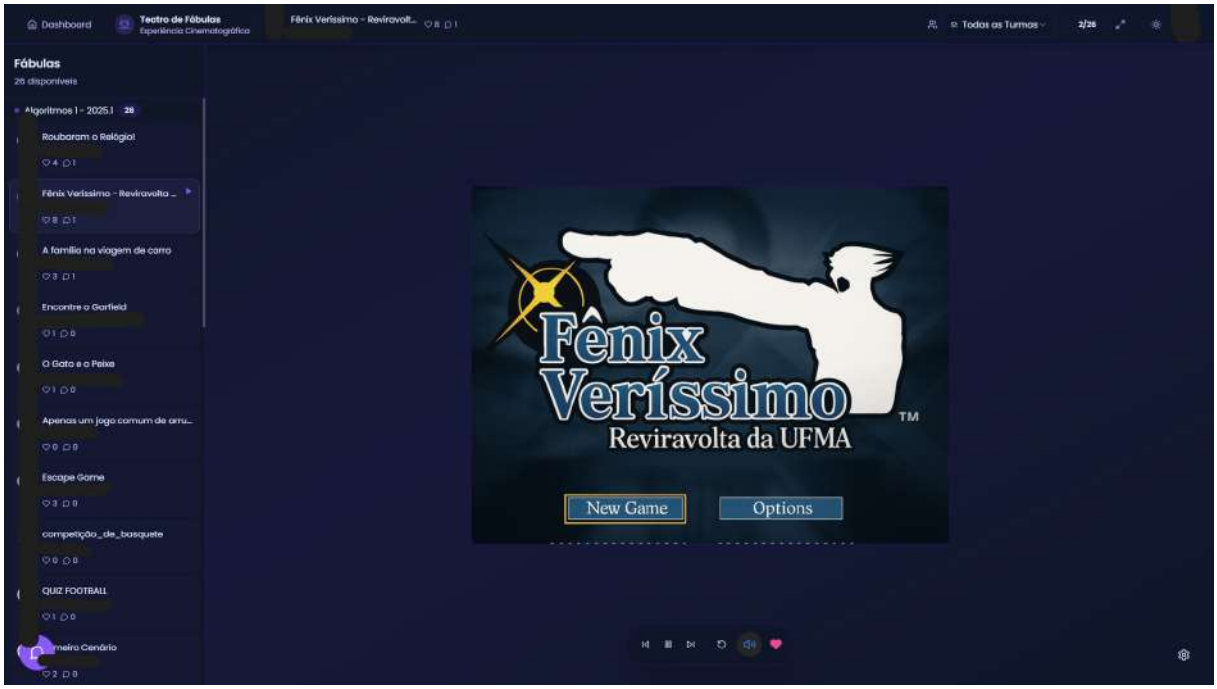
Figura 12 – Fábulas associadas a uma turma no CodeFab



Fonte: Elaborado pelo autor

Para fins de apresentação em sala e avaliação coletiva, a plataforma oferece o Teatro de Fábulas, tela mostrada na Figura 13. Nesse modo, o usuário percorre de forma contínua todas as fábulas postadas na turma, maximizando a experiência de navegação e facilitando a demonstração sequencial dos trabalhos.

Figura 13 – Teatro de fábulas disponível para uma turma



Fonte: Elaborado pelo autor

3.6 Considerações Finais sobre o Desenvolvimento

Este capítulo apresentou a concepção do CodeFab 2.0, descrevendo requisitos, tecnologias adotadas e principais interfaces da plataforma. Foram detalhados os fluxos de autenticação, os diagramas comportamentais, o modelo relacional e os resultados visuais obtidos com a implementação.

4 FableML

Neste capítulo é fundamentada e analisada a proposta experimental de uma linguagem de domínio específico destinada à criação de fábulas interativas, complementando a infraestrutura documentada no Capítulo 3.

4.1 Fundamentos e Conceitos Aplicados

Segundo (FOWLER; PARSONS, 2011), linguagens de domínio específico (DSLs) são linguagens criadas para resolver problemas bem delimitados dentro de um determinado escopo. Essas DSLs podem ser classificadas como *internas*, quando são construídas dentro de uma linguagem de programação de propósito geral hospedeira, como Haskell ou Ruby, aproveitando sua sintaxe e mecanismos de abstração, ou *externas*, quando possuem uma sintaxe própria e requerem ferramentas específicas para análise e execução, como parsers, interpretadores e validadores. Um exemplo clássico de DSL externa é a linguagem SQL.

O termo *Domain-Specific Embedded Language* (DSEL) foi cunhado por HUDAK (1996) para descrever linguagens de domínio específicas embutidas em linguagens de programação de propósito geral, como o Haskell. Em seu trabalho, o autor argumenta que esse tipo de abordagem permite ao desenvolvedor concentrar-se nos aspectos semânticos da linguagem de domínio, aproveitando a infraestrutura já existente da linguagem hospedeira para lidar com tarefas como parsing, execução e manipulação de estruturas sintáticas. Essa proposta antecede e fundamenta classificações posteriores, como as de FOWLER e PARSONS (2011), que consolidaram a distinção entre DSLs internas e externas no contexto da engenharia de software.

Análogo ao trabalho apresentado por (HUDAK, 1996), a FableML aproveita o ferramental já estabelecido na comunidade, como (LEONIDAS-FROM-XIV, 2025), interpretadores XML, *linters* e mecanismos de *autocomplete* viabilizados por bibliotecas como (CODEMIRROR, 2025). Essa abordagem está alinhada com a filosofia defendida por (HUDAK, 1996), que recomenda a reutilização de mecanismos existentes de uma linguagem hospedeira para que o foco do projeto recaia sobre a semântica da linguagem de domínio e não sobre a construção de infraestrutura do zero.

De maneira complementar, FOWLER e PARSONS (2011) também discute esse cenário ao diferenciar DSLs internas (embutidas em linguagens de programação de propósito geral, como Haskell ou Ruby) e DSLs externas, como a FableML, que utilizam uma "sintaxe portadora", neste caso, o XML, para definir seu vocabulário e estrutura. Embora o XML não seja uma linguagem de programação de propósito geral(como C, Python, entre outras),

sua adoção como base estrutural da FableML permite aproveitar uma ampla gama de ferramentas já consolidadas no ecossistema.

Portanto, diante do que foi exposto, este capítulo tem como objetivo analisar a FableML, uma DSL que teve como estrutura grande parte da DSL proposta por (GOMES, 2022), mas com certas partes reformuladas neste trabalho, projetada especificamente para a criação de narrativas visuais interativas, à luz dos conceitos apresentados por (FOWLER; PARSONS, 2011) e (HUDAK, 1996).

A escolha do XML como sintaxe portadora acompanha uma tendência destacada por (FOWLER; PARSONS, 2011), que reconhece seu valor estrutural e o suporte de ferramentas consolidadas no ecossistema. Ressalta-se, contudo, que o próprio autor observa previamente em (FOWLER, 2008) que tal escolha pode ser motivada por uma resistência ao desenvolvimento de parsers personalizados, prática que denomina *parser fear*[1]. No contexto da FableML, essa decisão se justifica não apenas pela viabilidade técnica e pela reutilização de ferramentas já disponíveis, mas também pela intenção de concentrar os esforços na definição semântica e na aplicabilidade pedagógica da linguagem.

Os trabalhos anteriores desempenharam papel fundamental na definição da linguagem de marcação utilizada para a criação de narrativas interativas. No trabalho de SILVA (2020), observou-se uma abordagem acoplada ao HTML e ao AngularJS, resultando em uma primeira versão verbosa e dependente das estruturas dessas ferramentas. Em contrapartida, o trabalho subsequente de GOMES (2022) promoveu uma simplificação significativa da DSL, priorizando a experiência de desenvolvimento por meio de melhorias como suporte a *linting* e *autocomplete*, auxiliando os usuários com uma experiência guiada enquanto desenvolviam.

A partir dessas iterações, consolidou-se a necessidade de evoluir a linguagem para atender aos pontos ausentes nos projetos anteriores, como a inexistência de variáveis globais, a repetição de código e o acoplamento rígido entre cenas. A FableML, portanto, busca aprimorar os aspectos das implementações passadas, propondo uma sintaxe mais focada na criação de narrativas visuais de forma acessível.

A FableML introduz um conjunto de abstrações algorítmicas mínimas, como variáveis, estruturas condicionais e laços de repetição, que facilitam a construção lógica de interações. Trata-se de uma linguagem de domínio específico (DSL) classificada como externa, conforme a tipologia apresentada por (FOWLER; PARSONS, 2011), por possuir uma sintaxe própria baseada em XML, independentemente de linguagens de programação de propósito geral. No entanto, sua implementação reutiliza a infraestrutura consolidada do

[1] Segundo FOWLER (2008), o termo *parser fear* descreve a tendência de desenvolvedores evitarem a criação de sintaxes próprias para DSLs externas devido à percepção de complexidade associada à implementação de parsers personalizados. Essa aversão leva, muitas vezes, à adoção de tecnologias como XML como alternativa estrutural, mesmo que essa escolha possa comprometer a expressividade da linguagem.

ecossistema XML, como parsers, validadores e editores, a fim de estruturar semanticamente narrativas interativas. Essa estratégia aproxima-se da filosofia proposta por (HUDAK, 1996), ao recomendar que linguagens de domínio se apoiem em mecanismos existentes de uma linguagem hospedeira para evitar a construção de infraestrutura do zero. No caso da FableML, essa base é representada pelo próprio conjunto de ferramentas associadas ao XML, adaptado para acomodar semânticas como lógica condicional, interpolação de variáveis e estruturas de controle declarativas.

Além disso, a FableML compartilha semelhanças estruturais com sistemas de template como Angular ou PHP, nos quais o conteúdo é descrito em marcação, enriquecido com lógica embutida, analisado e transformado em uma árvore sintática abstrata (AST), sendo então interpretado por um motor de execução que gera componentes visuais interativos. Essa arquitetura viabiliza a separação entre estrutura, lógica e apresentação, conferindo maior poder expressivo à linguagem e mantendo sua acessibilidade para usuários iniciantes.

As quatro metas centrais da nova versão da linguagem são:

1. Reduzir a verbosidade, evitando duplicações desnecessárias de elementos e promovendo maior coesão estrutural;
2. Permitir o *code splitting* por meio de páginas independentes, carregadas dinamicamente conforme o fluxo narrativo;
3. Introduzir abstrações algorítmicas básicas (*if*, *for*, variáveis, funções) com sintaxe declarativa e compatível com os princípios educacionais da plataforma;
4. Habilitar a interpolação de variáveis nos atributos dos agentes, permitindo que qualquer elemento em tela exiba dinamicamente valores derivados do estado global, como textos, imagens, expressões booleanas ou números.

Essa evolução deve ser compreendida como uma etapa natural no processo de amadurecimento da linguagem, resultado do acúmulo de experiências, limitações observadas e necessidades emergentes no contexto da autoria interativa. Ao introduzir novos recursos estruturais e semânticos, a FableML busca ampliar sua capacidade expressiva e pedagógica, contribuindo de forma mais efetiva para o desenvolvimento de narrativas interativas tecnicamente viáveis.

4.2 Sintaxe e Elementos da Linguagem

A FableML adota uma notação declarativa baseada em XML, projetada para ser acessível a iniciantes e, ao mesmo tempo, expressiva o suficiente para representar

estruturas interativas e lógicas narrativas. Sua concepção está alinhada com os princípios do *pensamento computacional*, que, segundo (WING, 2006), compreende habilidades como abstração, decomposição, reconhecimento de padrões e formulação de algoritmos. Cada elemento da linguagem visa exercitar esses pilares ao permitir que o autor estruture problemas narrativos e os resolva de maneira lógica e modular.

Na Tabela 2, são mapeadas as principais funcionalidades da FableML em relação às habilidades que compõem o pensamento computacional. Essa relação evidencia o potencial da linguagem não apenas como ferramenta de autoria, mas também como instrumento pedagógico para o desenvolvimento de competências computacionais de forma acessível e prática.

Tabela 2 – Habilidades do pensamento computacional apoiadas pela FableML

Habilidade	**Elemento da FableML**	**Descrição**
Abstração	`<property>`	Permite declarar variáveis simbólicas que representam o estado global da fábula, promovendo generalização de valores.
Decomposição	`<page>`	Segmenta a narrativa em cenas modulares, facilitando o desenvolvimento por partes e a reutilização de trechos.
Reconhecimento de Padrões	`<fab-for>` e interpolação	Automatiza estruturas repetitivas como listas ou menus, identificando padrões recorrentes na lógica narrativa.
Formulação de Algoritmos	`<fab-if>`, expressões e comandos declarativos	Utiliza estruturas condicionais e operações sobre variáveis para expressar regras e fluxos interativos.

Fonte: Elaborado pelo autor

Nesta seção, são apresentados os principais elementos da sintaxe da FableML, com exemplos ilustrativos e explicações sobre seu funcionamento semântico.

4.2.1 Elementos da Linguagem (Tags)

- ***<fable>***
  - Elemento raiz de toda fábula criada na plataforma.
  - Contém os elementos de configuração e a estrutura narrativa geral.

  - Filhos permitidos: *<config>* (opcional), múltiplos *<property>* e múltiplas *<page>*.

```
<fable>
  <page id="inicio">
    <!-- Agentes aqui -->
  </page>
</fable>
```

- ***<config>***
  - Define configurações globais da fábula, como tipografia.
  - Atualmente possui o atributo *font-family*.

```
<fable>
  <config font-family="Poppins"/>
  <primeira-pagina />
</fable>
```

- ***<property>***
  - Permite a declaração de variáveis globais que influenciam o comportamento da fábula.
  - Atributos: *name* (obrigatório) e *default* (obrigatório).
  - Suporta os tipos: número, string e booleano.

```
<fable>
  <property name="pontuacao" default="0"/>
  <primeira-pagina />
  <page id="segunda-pagina">
    <agent id="btn" text="Clique aqui"
           on_click="set(pontuacao, $pontuacao + 100)" />
  </page>
</fable>
```

- ***<page>***
  - Representa uma cena ou tela narrativa da fábula.

  - Atributos: *id* (obrigatório), *bg_color*, *bg_image* e *bg_music* (opcionais).
  - Permite carregamento modular e reutilização por meio de *code splitting*.

```
<fable>
  <page id="floresta" bg_color="#228B22">
    <!-- Agentes aqui -->
  </page>
</fable>
```

- ***<agent>***
  - Define um elemento visual ou interativo presente em uma página.
  - Atributos comuns: *id*, *x*, *y*, *width*, *height*, *text*, *image*, *video*.
  - Suporte a eventos como *on_click*, *on_drag*, *on_drop*, *on_hover*.
  - Todos os atributos permitem interpolação dinâmica de variáveis.

```
<agent id="heroi" x="100" y="200" text="Olá!"/>
```

- ***<fab-if>***
  - Elemento condicional que controla a exibição de filhos com base em uma expressão booleana.
  - Atributo principal: *when*, contendo a condição a ser avaliada.

```
<page id="primeira-pagina">
  <fab-if when="$pontuacao > 10">
    <agent id="bonus" x="150" y="150" text="Bônus!"/>
  </fab-if>
</page>
```

- ***<fab-for>***
  - Estrutura de repetição declarativa que permite renderizar múltiplos elementos a partir de uma lista.
  - Atributos: *each* (coleção) e *index* (variável de iteração).

```
<fab-for each="item in $inventario" index="i">
  <agent id="item-$i" x="$i*80" y="300" text="$item"/>
</fab-for>
```

4.2.2 Funções Principais

As funções utilizadas nos atributos de evento como *on_click*, *on_drop*, entre outros, são responsáveis por modificar o estado ou o fluxo da narrativa. Abaixo, um resumo das principais:

- ***set(var, valor)***
  - Atribui um novo valor a uma variável declarada com *<property>*.
  - Permite expressões envolvendo outras variáveis, números, strings, arrays ou booleanos.

```
<agent id="btn-increment" on_click="set(pontuacao, $pontuacao + 10)"/>
```

- ***goToPage(id)***
  - Redireciona a fábula para outra página com base no *id* especificado.
  - Caso a página passada não existir, no editor estará sinalizado que a página não existe.

```
<agent id="btn-next-page" on_click="goToPage('segunda-pagina')"/>
```

- ***reset()***
  - Redefine completamente o estado da fábula, útil quando os usuários precisam criar fluxos de *game over*.

```
<agent id="btn-game-over" on_click="reset()"/>
```

- ***playSound(src)***
  - Executa um efeito sonoro ou música a partir do caminho do arquivo informado.

```
<agent id="btn-error" on_click="playSound('erro.mp3')"/>
```

- ***setIf(condição, var, valor)***
  - Atribui valor a uma variável apenas se a condição for verdadeira.

```
<agent id="btn-increment" on_click="setIf($pontuacao > 50, nivel,
↪  'avançado')"/>
```

Os exemplos apresentados destacam as valências da FableML. Essa abordagem permite que usuários consigam estruturar lógicas narrativas de maneira intuitiva, promovendo ao mesmo tempo o contato com conceitos fundamentais do pensamento computacional.

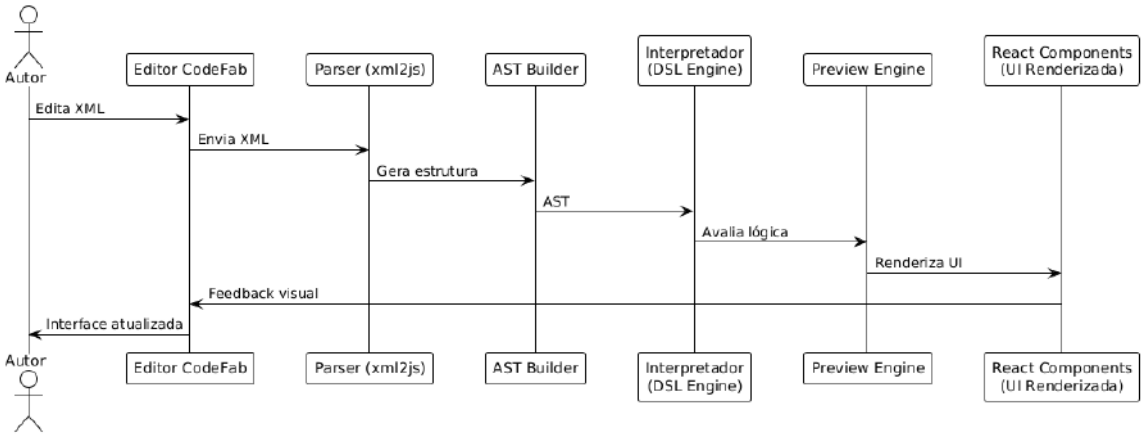
4.3 Arquitetura e Recursos Internos da FableML

4.3.1 Ciclo de Interpretação

O processo se inicia com o código XML criado pelo autor, carregado no editor presente no CodeFab. Esse conteúdo é analisado por um parser que valida a estrutura sintática e transforma os elementos da linguagem em um objeto via xml2js. A partir disso, a engine interpreta os dados e gera uma estrutura visual que será renderizada dinamicamente com base no estado atual da narrativa.

A Figura 14 apresenta o ciclo completo de processamento da linguagem FableML, desde a edição do XML até a visualização da interface final, destacando as etapas de análise, interpretação e renderização.

Figura 14 – Etapas de processamento da linguagem FableML



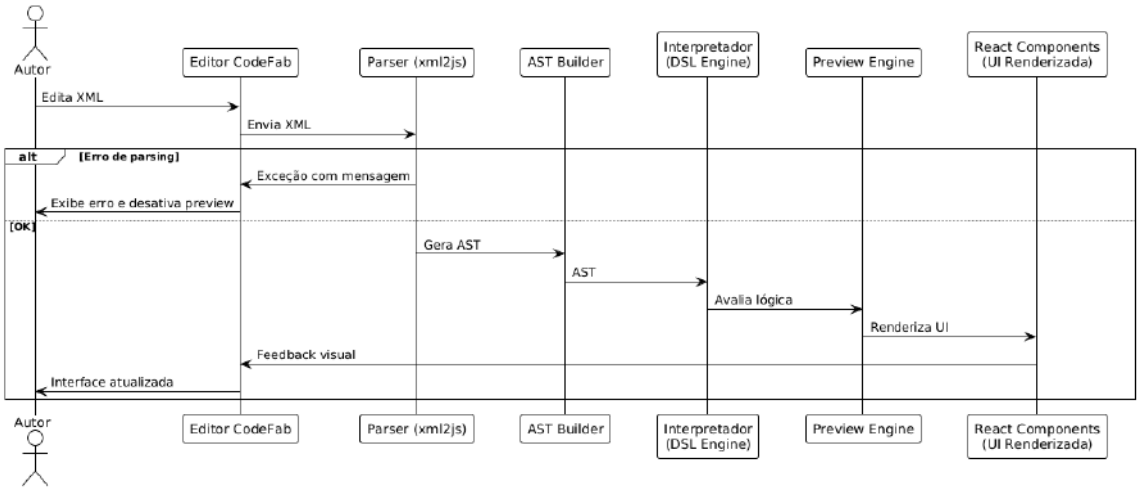
Fonte: Elaborado pelo autor.

Durante a etapa de interpretação, a engine executa a lógica declarada na AST (do inglês Árvore Sintática Abstrata). Isso inclui a avaliação de expressões condicionais e matemáticas, a interpolação de variáveis em atributos e a expansão de estruturas como fab-for, que funcionam como laços de repetição. Cada agente é processado com base em um contexto de execução, composto por variáveis globais, valores locais e informações da página atual.

Em seguida, a engine de *preview* converte os nós da AST em elementos visuais React, utilizando propriedades como posição, imagem, texto, animação e eventos. Esse processo gera uma representação visual que reflete dinamicamente o estado do sistema, com suporte a interações como clique, arrasto e *hover*. Como todo o fluxo é reativo, alterações no estado global desencadeiam reavaliações e renderizações automáticas dos agentes afetados.

A Figura 15 complementa o fluxo anterior, detalhando o comportamento do sistema em situações de erro de parsing. Caso o XML contenha problemas de formatação ou estrutura, o interpretador interrompe o ciclo e desativa a pré-visualização, exibindo uma mensagem ao usuário com orientações para correção.

Figura 15 – Ciclo de interpretação com tratamento de Erros



Fonte: Elaborado pelo autor.

Esse ciclo de transformação XML → AST → UI permite que a FableML funcione como uma linguagem de marcação com comportamentos lógicos embutidos, operando de forma semelhante a sistemas de *template engine*, como os utilizados em Angular ou PHP. Ao mesmo tempo, preserva a clareza da estrutura XML e mantém compatibilidade com ferramentas padronizadas, demonstrando seu caráter de DSL simbiótica com a linguagem de marcação.

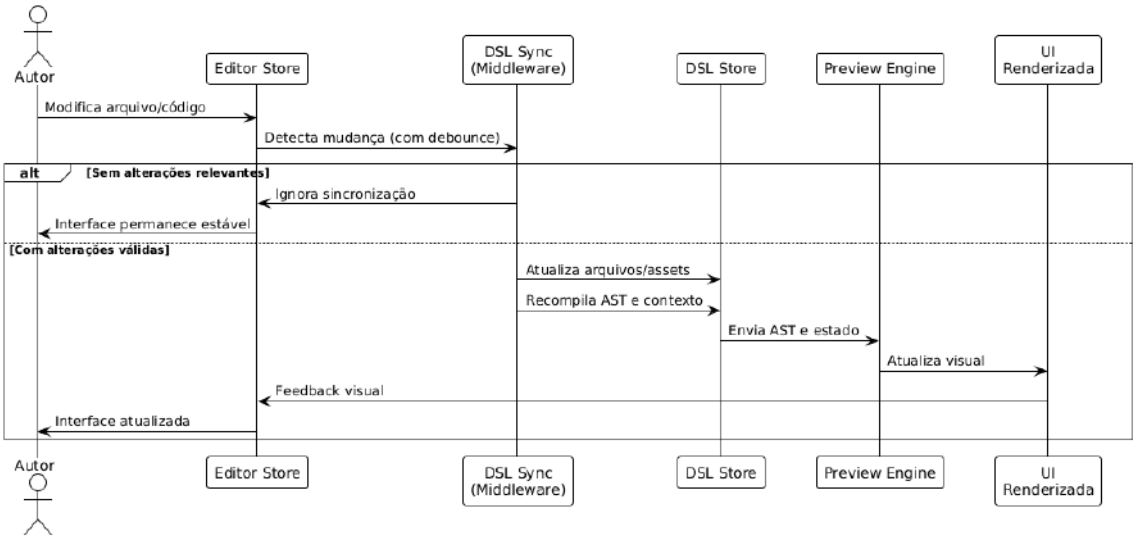
4.3.2 Gerenciamento de Estado e Otimização

Durante a edição das fábulas, a plataforma CodeFab precisa garantir que alterações no código não comprometam o desempenho da visualização em tempo real. Para isso, foi adotada uma arquitetura baseada em três camadas de controle de estado que operam de maneira encadeada e unidirecional. Essa abordagem facilita a identificação de mudanças relevantes e evita atualizações desnecessárias da interface.

A primeira camada é representada pela *Editor Store*, responsável por armazenar o conteúdo editável, arquivos XML, árvore de páginas e configurações do usuário. A segunda camada, implementada via um mecanismo intermediário denominado *DSL Sync Store*, atua como observador, detectando mudanças reais por meio de verificação incremental e sincronização com atraso programado (*debounce*). A terceira camada, denominada *DSL Store*, mantém o estado final interpretado, contendo a AST atual, propriedades globais e contexto da página ativa.

A Figura 16 representa a orquestração entre essas camadas, destacando o fluxo principal de dados, o ciclo de detecção e atualização, e a retroalimentação visual para o autor.

Figura 16 – Orquestração das camadas de estado da FableML



Fonte:Elaborado pelo autor.

Além desse fluxo principal, a plataforma incorpora mecanismos de prevenção a renderizações excessivas: cache de agentes renderizados(via memoização do React), filtragem granular por atributos, contadores de mudança e reatividade seletiva por componente. Esses recursos permitem que apenas os elementos realmente impactados por alterações sejam rerenderizados, promovendo maior estabilidade e fluidez de uso.

4.3.3 Sistema de *Lint* e *Autocomplete*

A plataforma CodeFab incorpora um sistema integrado de *lint* e *autocomplete* para a linguagem FableML, com o objetivo de auxiliar o autor na criação de fábulas bem estruturadas e semanticamente corretas. Essa funcionalidade oferece feedback em tempo real durante a digitação, alertando sobre erros de sintaxe, inconsistências lógicas e sugestões contextuais, tornando o processo de escrita mais acessível, especialmente para iniciantes.

A implementação foi desenvolvida sobre o editor CodeMirror 6, cujas extensões nativas permitem a construção de mecanismos personalizados de *lint* e *autocomplete*, conforme descrito em sua documentação oficial (CODEMIRROR, 2025). Para realizar a integração com o ecossistema React, foi necessário utilizar a biblioteca *@uiw/react-codemirror*, a fim de facilitar sua adaptação ao paradigma de aplicações de página única (SPA). A documentação apresentada em (REACT CODEMIRROR, 2025) foi de suma importância para a conexão entre a camada de validação e a camada de interface do usuário.

Na prática, o editor funciona como uma camada intermediária que repassa as ações do usuário, como digitação e movimentação do cursor, para módulos especializados

implementados no navegador. Esses módulos atuam como servidores de linguagem embutidos para a FableML, interpretando sua gramática, validando seu uso e oferecendo sugestões contextuais.

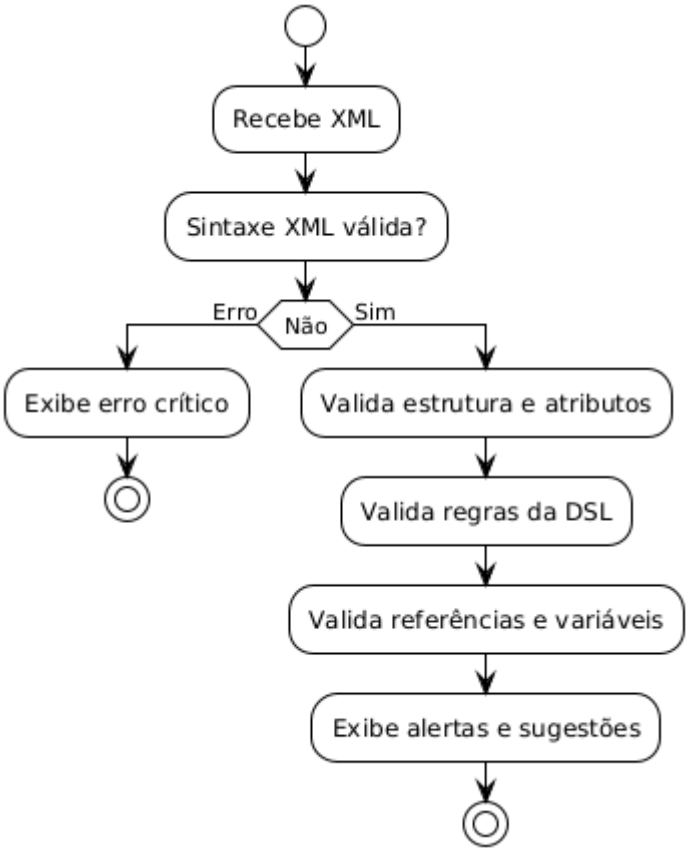
Essa arquitetura adota os princípios do Language Server Protocol (LSP), conforme proposto por (BORK, 2023), que defende a separação entre as *language smarts*, como análise sintática, *lint*, *autocomplete* e interpretação da AST, cuja responsabilidade cabe ao servidor da linguagem, e as *editor smarts*, como renderização e interação, que são tratadas pelo cliente, neste caso, o CodeMirror. Ainda segundo (BORK, 2023), essa divisão permite que o editor permaneça agnóstico à linguagem, enquanto o analisador se encarrega da lógica específica, promovendo facilidade de integração e simplificando o suporte a linguagens em ambientes web.

4.3.3.1 *Lint*

O mecanismo de validação estática (*lint*) da FableML é executado em camadas, conduzindo diagnósticos com base em sua gravidade. Primeiramente, é realizada a análise sintática do XML, seguida por checagens de estrutura, atributos obrigatórios, regras específicas da DSL e consistência de referências. Esse fluxo em cascata prioriza erros críticos, interrompendo etapas posteriores até que sejam resolvidos, o que facilita a identificação de falhas por parte do usuário.

A Figura 17 apresenta, de forma resumida, este fluxo de validação, mostrando as principais etapas e o comportamento do sistema em caso de erro ou aviso.

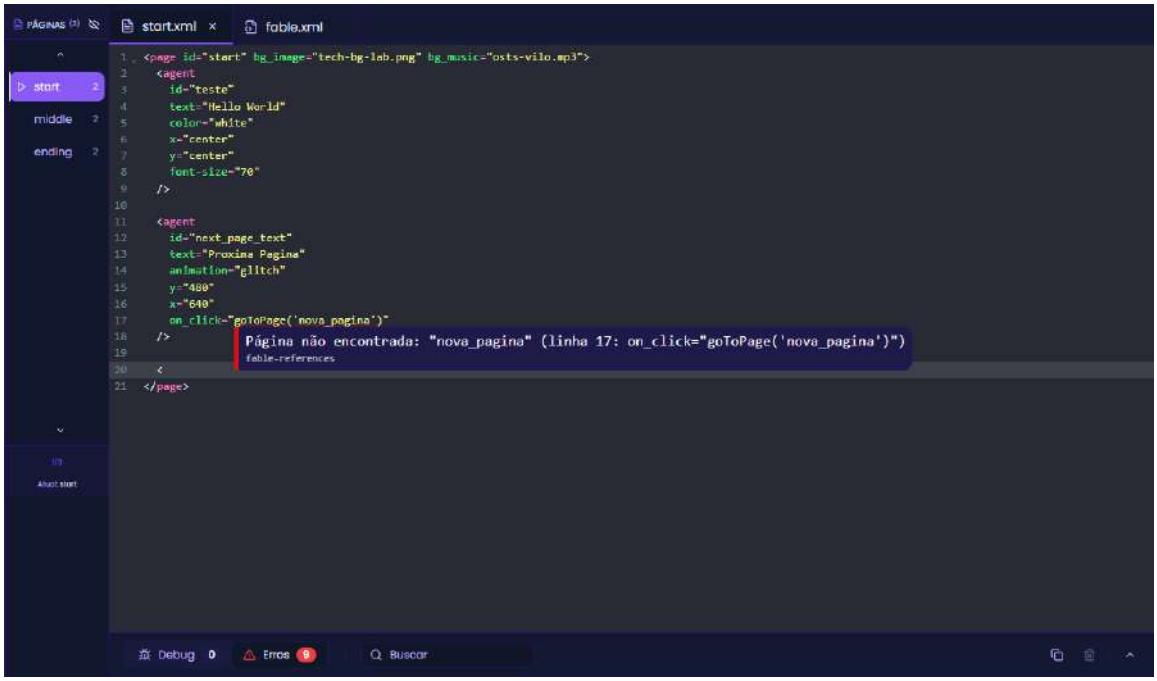
Figura 17 – Fluxo de *linting* da DSL



Fonte: Elaborado pelo autor.

Problemas típicos incluem tags sem fechamento, variáveis não declaradas, conflitos de ID ou regras hierárquicas incorretas. As mensagens de erro são exibidas no editor, geralmente acompanhadas de sugestões de correção automática.

Figura 18 – Exemplo de validação de erro no editor da plataforma CodeFab



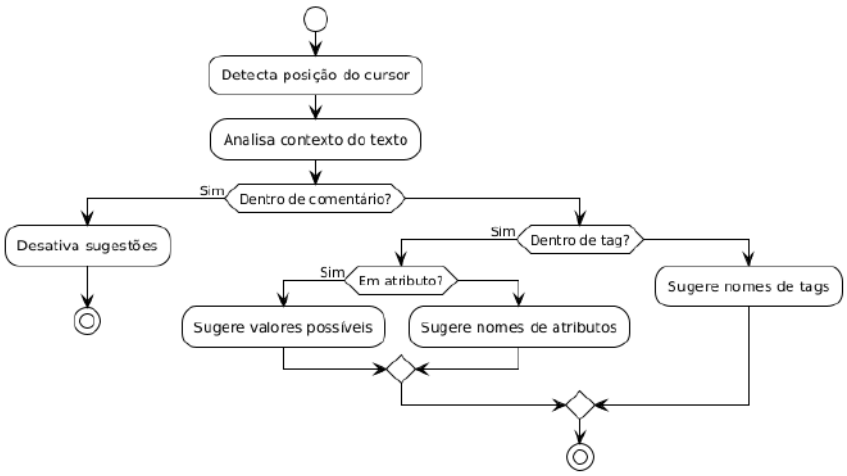
Fonte: Elaborado pelo autor.

4.3.3.2 *Autocomplete*

O sistema de *autocomplete* fornece sugestões contextuais com base na posição do cursor, nome do elemento, atributo ou valor. Sugestões são filtradas conforme o contexto atual, por exemplo, sugerindo apenas elementos válidos dentro de uma página ou valores esperados de atributos como cor ou tipo de um agente(texto, imagem, vídeo)

A Figura 19 ilustra esse fluxo de sugestões, que começa com a análise do contexto sintático e termina com a exibição de opções relevantes diretamente no editor.

Figura 19 – Fluxo de sugestão do *autocomplete* dentro do Codefab

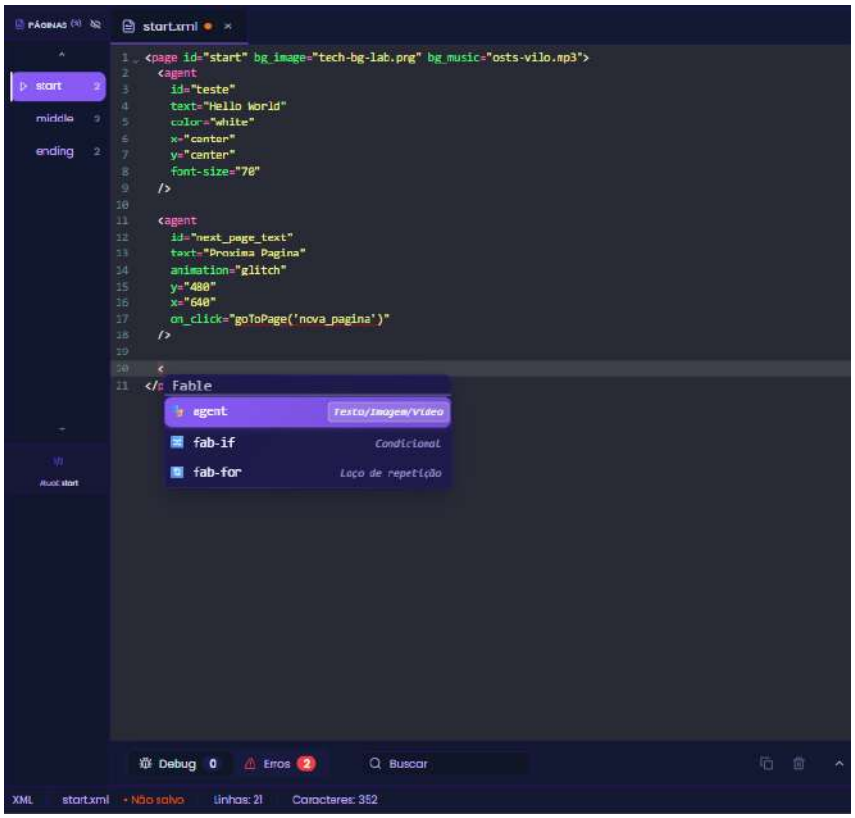


Fonte: Elaborado pelo autor.

Esse recurso funciona como uma forma de documentação interativa, ajudando os autores a aprenderem a gramática da FableML enquanto escrevem, reduzindo erros e reforçando boas práticas.

A Figura 20 mostra o sistema de *autocomplete* em funcionamento dentro de uma fábula real. O editor sugere, de forma contextual, apenas as tags válidas esperadas dentro da estrutura atual do código, neste caso, uma tag dentro do corpo de uma página. Esse comportamento evidencia a capacidade da plataforma de adaptar as sugestões com base na posição do cursor, facilitando a escrita da DSL mesmo sem conhecimento prévio.

Figura 20 – Sugestões de *autocomplete* no editor da plataforma CodeFab



Fonte: Elaborado pelo autor.

4.3.3.3 Desempenho e Integração

Ambos os sistemas de *lint* e *autocomplete* são executados de forma incremental e assíncrona, oferecendo respostas rápidas mesmo em arquivos extensos. Tudo isso ocorre com base em verificações com *delay* acionadas por eventos de digitação ou navegação pelo código, garantindo estabilidade e alta responsividade.

Com esse conjunto de recursos, a plataforma promove uma experiência de autoria eficaz, amigável e alinhada às melhores práticas de ambientes de desenvolvimento assistido, como demonstrado por JIANG Shaokang; COBLENZ (2024), em que o uso de *autocomplete*

facilitou que programadores aprendessem APIs com as quais tinham pouca familiaridade e reduziu o tempo gasto com a leitura da documentação.

4.4 Considerações Finais

A FableML consolida uma proposta de linguagem de marcação voltada à criação de narrativas interativas, integrando a estrutura do XML a essas abstrações algorítmicas. Entretanto, a dependência ainda existente de linguagens de marcação pode ser uma dificuldade para usuários sem familiaridade prévia.

Ao adotar mecanismos de análise e sugestões contextuais, a linguagem oferece suporte à autoria assistida e contribui para reduzir essa barreira inicial desempenhando papel fundamental na experiência guiada de desenvolvimento. No capítulo seguinte, abordamos como o CodeFab foi aplicado em sala de aula.

5 Ensaio de Interação

Este capítulo descreve o processo de aplicação da plataforma CodeFab em ambiente de sala de aula, detalhando cronograma, participantes, atividades práticas e instrumentos de coleta de dados. São apresentados ainda os fundamentos dos questionários SUS e TAM utilizados para avaliar a usabilidade e a aceitação da ferramenta respectivamente.

5.1 Contexto do Estudo

O ensaio foi conduzido em uma turma da disciplina Algoritmos I do curso de Ciência da Computação da Universidade Federal do Maranhão (UFMA), durante o primeiro semestre de 2025. Dos 39 alunos matriculados, 31 (*79 %*) assinaram o Termo de Consentimento Livre e Esclarecido (TCLE) e participaram da dinâmica.

5.2 Planejamento das Atividades

5.2.1 Cronograma

A intervenção teve duração de duas semanas, de 2 a 16 de junho de 2025. A Tabela 3 resume o cronograma.

Tabela 3 – Cronograma das atividades

Data	**Atividade**	**Duração**
02/jun	Workshop inicial (markup, Git/GitHub, FableML)	2 h
03–16/jun	Suporte assíncrono via WhatsApp, Documentação, Assistente IA	—
16/jun	Apresentação das fábulas + aplicação de SUS/TAM	2 h

Fonte: Elaborado pelo autor

5.2.2 Objetivos de Aprendizagem

Ao final do ensaio, esperava-se que os alunos fossem capazes de:

- **Criação de fábula**: elaborar uma narrativa interativa utilizando a linguagem FableML;
- **Manipulação de variáveis**: declarar variáveis com `property` e utilizar estruturas condicionais como `fab-if`;
- **Estrutura em múltiplas páginas**: organizar a fábula em pelo menos duas páginas distintas, utilizando transições entre elas;

- **Uso de agentes interativos**: empregar ao menos um `agent` com evento associado, como o `on_click`;
- **Autenticação na plataforma**: realizar login utilizando conta do GitHub;
- **Apoio ao desenvolvimento**: reconhecer e utilizar os recursos de documentação interna e o chat assistente.

5.3 Encontros com a Turma

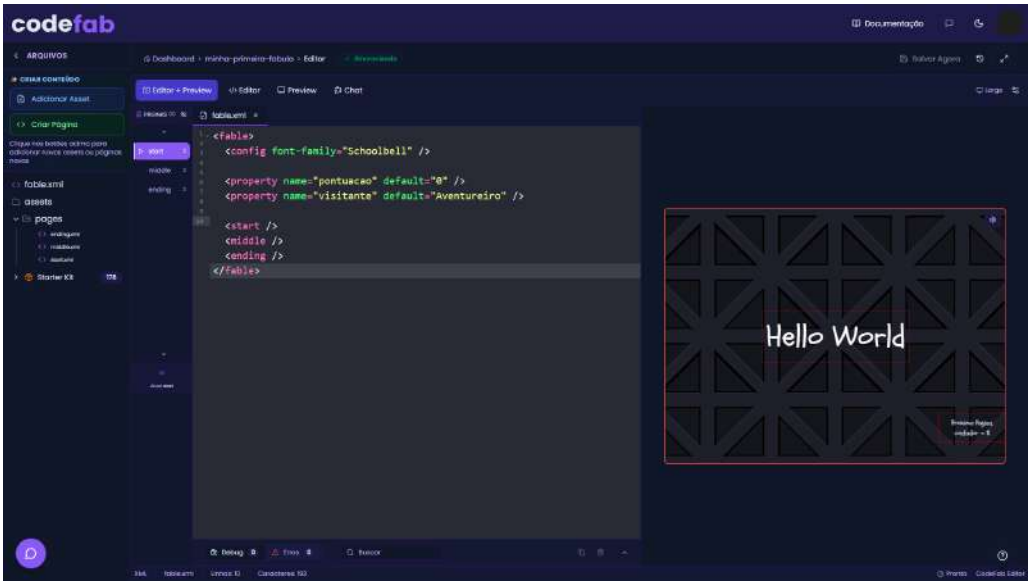
5.3.1 Workshop Inicial

O primeiro encontro introduziu:

- **Linguagens de marcação**: apresentação dos conceitos fundamentais, com exemplos práticos em HTML e XML;
- **Controle de versão**: introdução ao uso de Git e GitHub, considerados pré-requisitos para autenticação na plataforma;
- **Sintaxe da FableML**: explicação da estrutura básica da linguagem utilizada na construção das fábulas.

Durante a exposição, foi criada coletivamente a fábula ilustrada na Figura 21, a qual representa o esqueleto inicial comum a todas as fábulas desenvolvidas na plataforma. A partir dessa estrutura base, foram introduzidos e praticados os elementos descritos em 5.2.2.

Figura 21 – Fábula introdutória desenvolvida coletivamente durante o *work-shop*



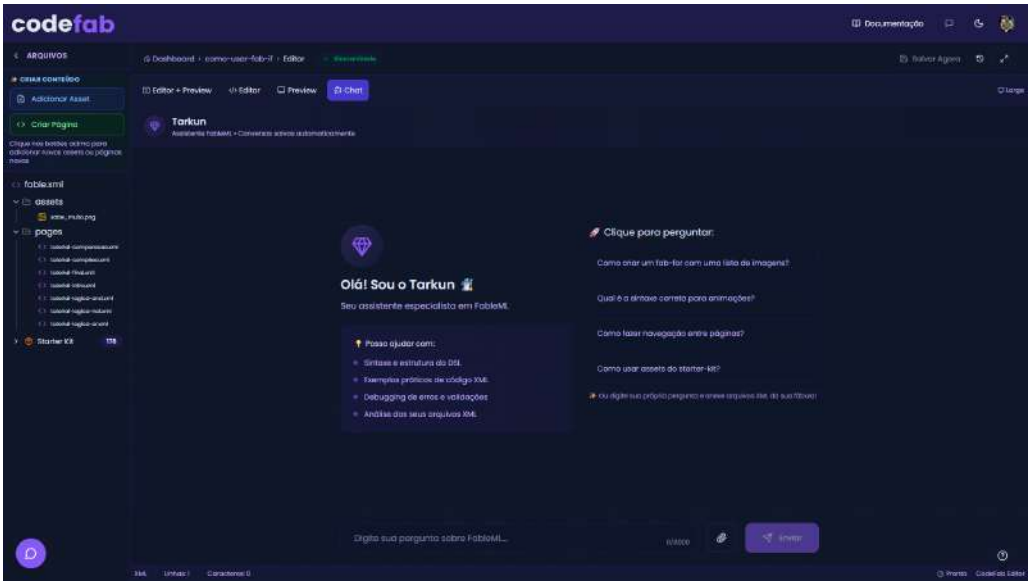
Fonte: Elaborado pelo Autor

5.3.2 Suporte Assíncrono

A aplicação contava com duas ferramentas de documentação: uma ferramenta de assistente de IA embutida no editor, e uma documentação completa explicando todo o manual da linguagem FableML.

Na Figura 22, é possível visualizar o início da navegação no chat e como ele se assemelha a outros assistentes existentes no mercado. O usuário poderia fazer perguntas soltas sobre temas específicos da plataforma ou da linguagem de marcação. Para dúvidas relacionadas a arquivos do projeto, era possível anexá-los diretamente ao chat.

Figura 22 – Visão do chat assistente I.A



Fonte: Elaborado pelo Autor

Em conjunto ao chat assistente, foi criada uma documentação que explicava todas as funções, tags e estruturas da FableML dentro da plataforma, com o intuito de facilitar o acesso dos usuários, como é visto na Figura 23.

Figura 23 – Documentação interna do Codefab

Fonte: Elaborado pelo Autor

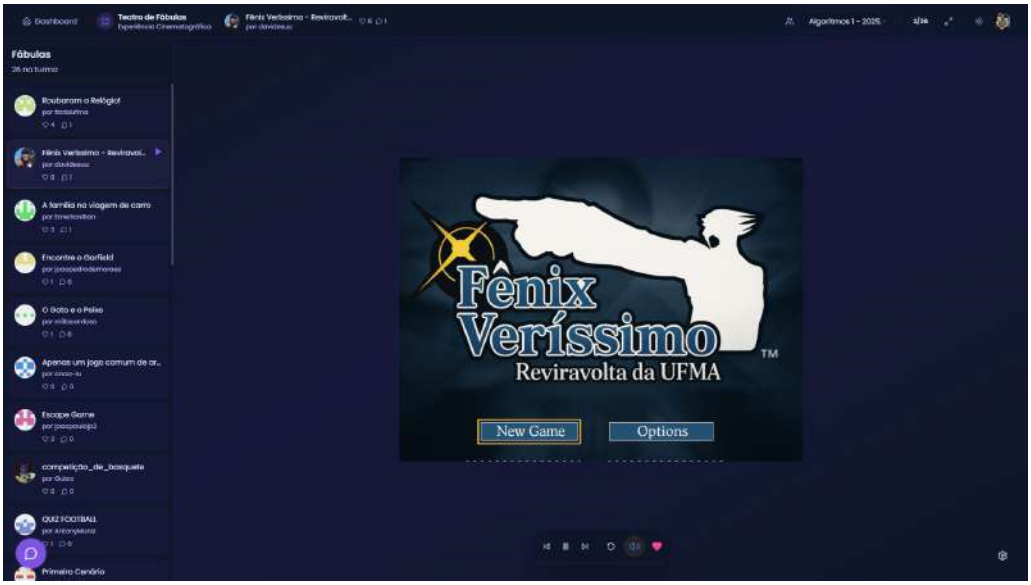
A documentação contava com uma linha do tempo explicando termos e conceitos. Ela foi utilizada em aula para exemplificar seu uso, além de apresentar suas funcionalidades para os usuários.

5.3.3 Workshop de Apresentação

No dia da apresentação, cada aluno exibiu sua fábula utilizando o modo Teatro, uma funcionalidade da plataforma que facilita a visualização sequencial de múltiplas fábulas. Esse modo estimula a navegação e interação com projetos de outros usuários, como é possível visualizar na Figura 24.

Ao final da dinâmica, foram produzidas 50 fábulas, das quais 26 foram postadas no mural da turma por meio da ferramenta interna de publicação do Codefab.

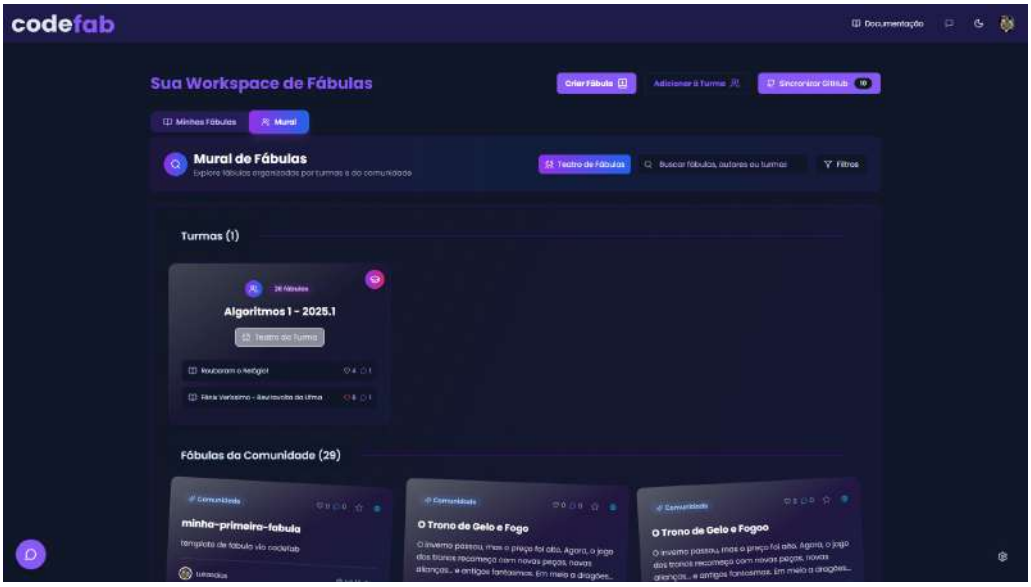
Figura 24 – Visão do modo teatro dentro do Codefab



Fonte: Elaborado pelo Autor

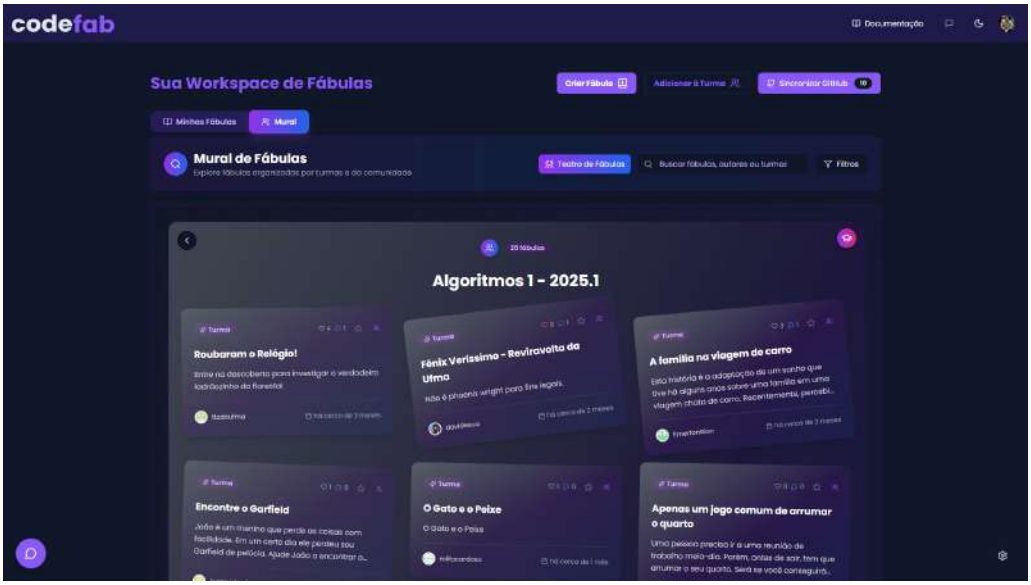
Na Figura 25 e na Figura 26, é possível visualizar a interface do mural. Em (GOMES, 2022), as funcionalidades de postagem de conteúdo foram terceirizadas para a plataforma Padlet. Nesta nova abordagem, o Codefab tornou-se autossustentável tanto na criação quanto na divulgação do conteúdo.

Figura 25 – Visão do mural público do Codefab



Fonte: Elaborado pelo Autor

Figura 26 – Visão do mural da turma do Codefab



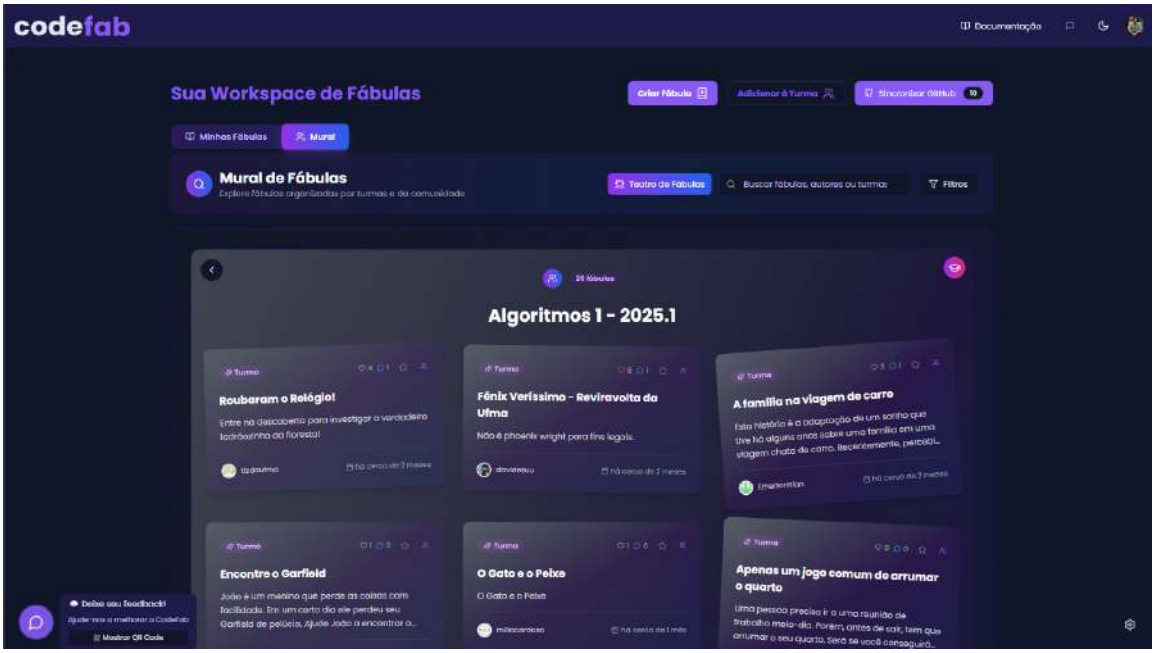
Fonte: Elaborado pelo Autor

5.4 Instrumentos de Coleta

Para avaliar essa nova abordagem do Codefab de modo qualitativo, ao final da apresentação das fábulas de tema livre criadas pelos usuários, foi aplicado um formulário com TCLE que continha questões relacionadas ao SUS (System Usability Scale) e ao TAM (Technology Acceptance Model), totalizando 38 questões.

Tanto o SUS quanto o TAM utilizaram a escala de Likert de 5 pontos (VAGIAS, 2006). O formulário foi aplicado via Google Forms, com o link presente dentro da plataforma para facilitar a usabilidade, como é possível visualizar na Figura 27.

Figura 27 – Visão do acesso ao formulário de pesquisa no Codefab



Fonte: Elaborado pelo Autor

5.4.1 Sobre o SUS

O *System Usability Scale* (SUS), proposta por (BROOKE et al., 1996), é uma ferramenta amplamente utilizada para avaliar a usabilidade percebida de sistemas. É composta por 10 questões que, somadas e normalizadas, geram uma pontuação de 0 a 100. Valores acima de 68 indicam que a usabilidade é considerada aceitável.

5.4.2 Sobre o TAM

O *Technology Acceptance Model* (TAM), proposto por (DAVIS, 1989) e posteriormente estendido por (VENKATESH, 2000), tem como objetivo explicar e prever a aceitação de tecnologias por parte dos usuários. O modelo postula que a intenção comportamental de utilizar um sistema é influenciada por dois fatores centrais:

- **Utilidade Percebida (PU)**: o grau em que o usuário acredita que a ferramenta melhora seu desempenho;
- **Facilidade de Uso Percebida (PEU)**: o quanto o sistema é percebido como fácil de aprender e utilizar;

5.5 Considerações Finais

O plano de duas semanas permitiu testar a nova iteração do CodeFab em cenário real de sala de aula, acumulando mais de cinquenta fábulas produzidas e evidências iniciais de usabilidade e aceitação. O capítulo seguinte apresenta os resultados quantitativos e qualitativos obtidos a partir dos questionários SUS e TAM, bem como métricas internas da plataforma.

6 Resultados

Este capítulo apresenta os resultados obtidos a partir da interação dos usuários com a plataforma CodeFab, incluindo métricas de comportamento, uso da linguagem FableML e engajamento com o assistente virtual. Os dados foram coletados por meio de três frentes: análise automatizada via Microsoft Clarity (CLARITY, 2025), inspeção de repositórios GitHub, e extração das conversas feitas com o chat embutido na plataforma. Ao final, são reservadas subseções para os resultados obtidos via aplicação dos questionários SUS e TAM.

Durante o levantamento do perfil dos participantes, observou-se que a maior parte dos participantes possuía 18 anos, representando 70,9 % da amostra. Outras faixas etárias estiveram presentes em menor proporção, com destaque para os estudantes de 19 anos (12,9 %) e os demais distribuídos igualmente entre 17, 21, 22, 24 e 29 anos, cada uma com 3,2 % de representação.

Quanto à distribuição de gênero, observou-se que 74,2 % dos participantes se identificaram com o gênero masculino, enquanto 25,8 % se identificaram com o gênero feminino.

Em relação ao tempo de contato com programação, 48,4% dos participantes relataram possuir menos de seis meses de experiência, 25,8% indicaram entre seis meses e um ano; 6,5% relataram entre um e dois anos, e 19,4% entre dois e quatro anos. Os dados estão sumarizados na Tabela 4.

Tabela 4 – Perfil dos participantes

Característica	**Distribuição (%)**
Idade	
18 anos	70,9
Outras idades	29,1
Gênero	
Masculino	74,2
Feminino	25,8
Tempo de experiência com programação	
Menos de 6 meses	48,4
De 6 meses a 1 ano	25,8
De 1 a 2 anos	6,5
De 2 a 4 anos	19,4

Fonte: Elaborado pelo autor.

6.1 Métricas Comportamentais

Além da coleta por meio dos formulários TAM e SUS, foi também utilizada a ferramenta Microsoft Clarity (CLARITY, 2025) para o monitoramento das ações do usuário na aplicação.

Durante as duas semanas de aplicação, foram registradas 338 sessões, com tempo ativo médio de 12,9 minutos[1]. Além disso, cada usuário acessou, em média, 11 páginas por sessão.

A plataforma foi acessada por diferentes navegadores, atendendo aos requisitos de compatibilidade descritos na Seção 3.1. O navegador mais utilizado foi o Google Chrome, com 233 sessões (68,93%), seguido pelo Microsoft Edge, com 69 sessões (20,41%), e pelo Opera, com 15 sessões (4,4%). As sessões restantes foram distribuídas entre outros navegadores com menor representatividade.

Esses dados indicam uma taxa de engajamento significativa, com sessões longas e grande parte do conteúdo da página sendo visualizado pelos usuários.

6.2 Análise de Uso dos Recursos da DSL

Foi realizado um processo de busca nos repositórios criados pelos alunos utilizando uma biblioteca Python para integração com o GitHub (PYGITHUB, 2025), juntamente com o uso de *Fine-grained personal access tokens* para evitar restrições por parte da API do Github (GITHUB, 2025a). O objetivo foi extrair e analisar os arquivos das fábulas para compreender o uso dos usuários em relação à sintaxe da linguagem.

Apesar de algumas fábulas mal formatadas terem dificultado o parsing detalhado de todos os arquivos, foi possível analisar 46 repositórios válidos. Para a leitura e interpretação das fábulas, utilizou-se a biblioteca nativa de manipulação XML da linguagem Python (PYTHON, 2024). Essa estratégia reforça a importância de ter adotado uma abordagem semelhante à proposta por (HUDAK, 1996), que defende o uso de DSELs como forma de aproveitar o ferramental já existente da linguagem hospedeira, focando mais na semântica da DSL e reduzindo o custo de implementação de infraestrutura. A extração automatizada dos arquivos permitiu a coleta das métricas que serão detalhadas a seguir.

Ao final da análise do conteúdo dos repositórios, foi possível identificar o uso de mais de 2.610 tags, 10.417 atributos e 1.040 chamadas de funções. Também foram encontradas 196 páginas personalizadas, o que reforça a adoção, por parte dos usuários, de práticas de modularização e evidencia a habilidade de decomposição discutida por WING (2006), especialmente considerando que uma das melhorias incorporadas à FableML foi

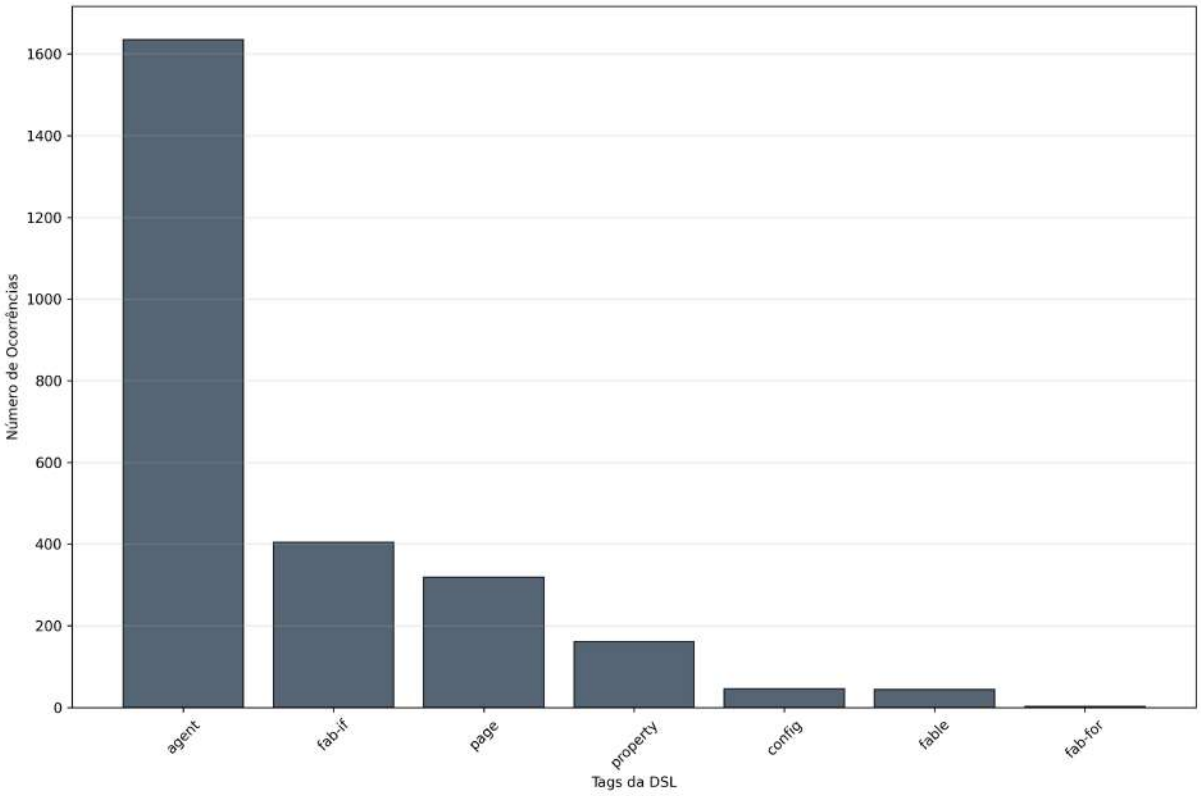
[1] Segundo CLARITY (2025), considera-se tempo ativo o período em que o usuário está de fato interagindo com a página, quando a aba ou janela da aplicação fica em segundo plano, esse tempo é desconsiderado.

justamente a possibilidade de dividir a fábula em páginas independentes, recurso ausente nas versões anteriores, que concentravam todo o conteúdo em um único arquivo.

6.2.1 Uso das Tags

Na Figura 28, observa-se que as tags mais utilizadas foram *agent*, *fab-if* e *page*, caracterizando o padrão de uso recorrente dos participantes nesta iteração. Nota-se, também, a baixa adesão à tag *fab-for*. Como mencionado anteriormente, a dinâmica ocorreu pouco tempo após a turma ter sido introduzida aos conceitos de laços de repetição, o que pode justificar a hesitação ou incerteza no uso dessa funcionalidade. Em quarto lugar, destaca-se a utilização da tag *property*, evidenciando o emprego do princípio da abstração por meio do uso de variáveis globais(WING, 2006).

Figura 28 – Principais tags utilizadas da FableML



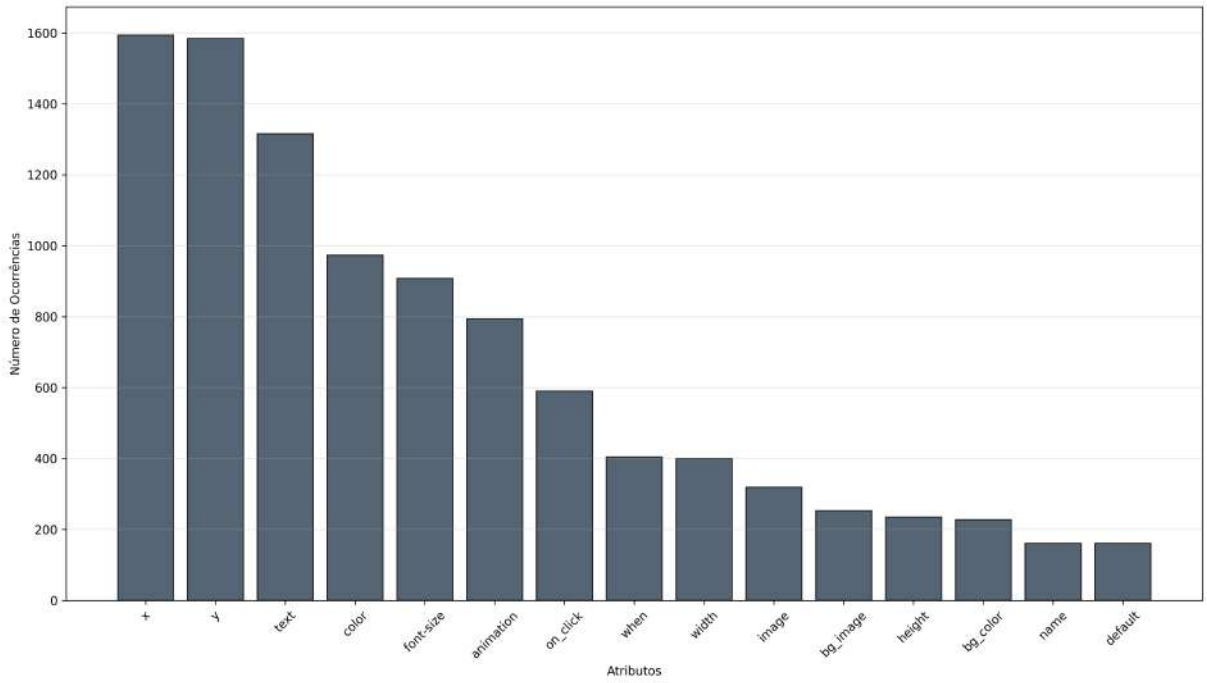
Fonte: Elaborado pelo autor.

6.2.2 Uso dos Atributos

Já na Figura 29, é possível observar uma altíssima adesão de atributos de posicionamento dos agentes (x e y), que se mostram praticamente indispensáveis para a estrutura visual da fábula. Em segundo lugar, tem-se o atributo *text*, indicando que os agentes vêm sendo utilizados, majoritariamente, como elementos textuais principais, e não como imagens ou vídeos. O atributo *image*, por sua vez, aparece apenas na décima posição

no ranking de uso, sugerindo uma frequência significativamente inferior em comparação ao *text*. Já agentes de vídeo praticamente não foram utilizados, o que pode refletir uma possível falha pedagógica no workshop introdutório descrito na Seção 5.

Figura 29 – Principais atributos utilizadas da FableML



Fonte: Elaborado pelo autor.

Observa-se ainda um uso moderado do atributo *animation*, funcionalidade presente no trabalho de SILVA (2020), mas ausente na versão proposta por GOMES (2022). Seu retorno na nova versão da FableML parece ter despertado interesse por parte dos usuários, conforme indicam os resultados.

O evento mais utilizado foi o *on_click*, enquanto os demais eventos tiveram menor adesão que os 15 principais atributos. Destaca-se também o uso do atributo *when*, obrigatório na tag *fab-if*, evidenciando uma adoção mediana dessa funcionalidade. Esse dado reforça a presença de elementos algorítmicos no raciocínio dos usuários, em linha com as competências mapeadas na Tabela 2.

6.2.3 Uso das Funções

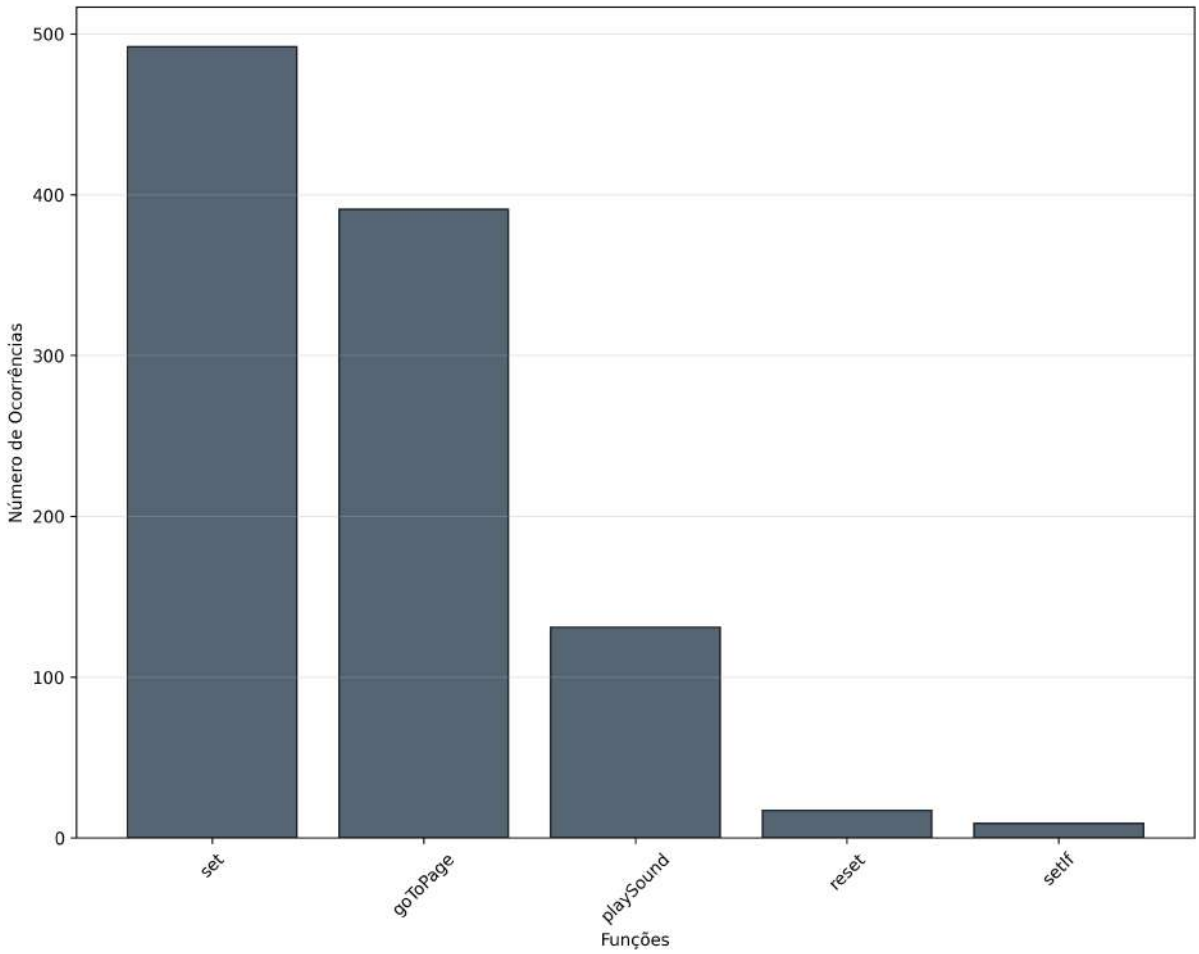
Quanto às principais funções utilizadas, é possível observar, na Figura 30, que a função *set* se destaca como a mais empregada. Seu uso recorrente revela um alto grau de manipulação de variáveis por parte dos usuários, o que indica que o sistema de controle de estado das fábulas foi amplamente explorado. Em segundo lugar, aparece a função *goToPage*, responsável pela transição entre páginas, evidenciando a aplicação de conceitos

de decomposição estrutural por meio da construção de narrativas divididas em múltiplas páginas.

A função *playSound* também foi utilizada com frequência moderada. Importante destacar que essa funcionalidade não estava presente no início da dinâmica, tendo sido implementada após uma solicitação feita via comunicação assíncrona com os participantes pelo WhatsApp. O fato de ter sido rapidamente adotada, mesmo com sua adição tardia, sugere que se tratava de um recurso essencial para enriquecer a experiência narrativa.

Por outro lado, a função *setIf*, também incorporada após o início da atividade, apresentou baixa adesão. Ainda assim, o uso geral das funções demonstra um envolvimento significativo dos participantes na construção lógica das fábulas. O surgimento de demandas por novas funcionalidades, como no caso de *playSound*, reflete um nível saudável de engajamento e uma dinâmica de evolução incremental típica de sistemas em ambientes reais de uso.

Figura 30 – Principais funções utilizadas da FableML



Fonte: Elaborado pelo autor.

6.3 Análise de Uso do Chat Assistente

Durante a atividade, todas as mensagens enviadas ao chat foram registradas e armazenadas para posterior análise. Ao todo, foram contabilizadas 472 perguntas feitas pelos participantes ao longo de duas semanas, resultando em uma média de 33,7 mensagens por dia. As dúvidas abrangeram tópicos como uso da linguagem FableML, erros de sintaxe, manipulação de variáveis e estruturação de páginas. O pico de mensagens ocorreu em 15 de junho, um dia antes do prazo final para entrega dos trabalhos.

Ao longo da dinâmica, 28 usuários distintos utilizaram o chat, com uma média de 16,9 perguntas por participante. O usuário mais ativo enviou um total de 64 mensagens durante as duas semanas. No total, 34 fábulas diferentes foram associadas a perguntas, com uma média de 13,9 dúvidas por fábula. A fábula que recebeu mais interações concentrou 58 perguntas.

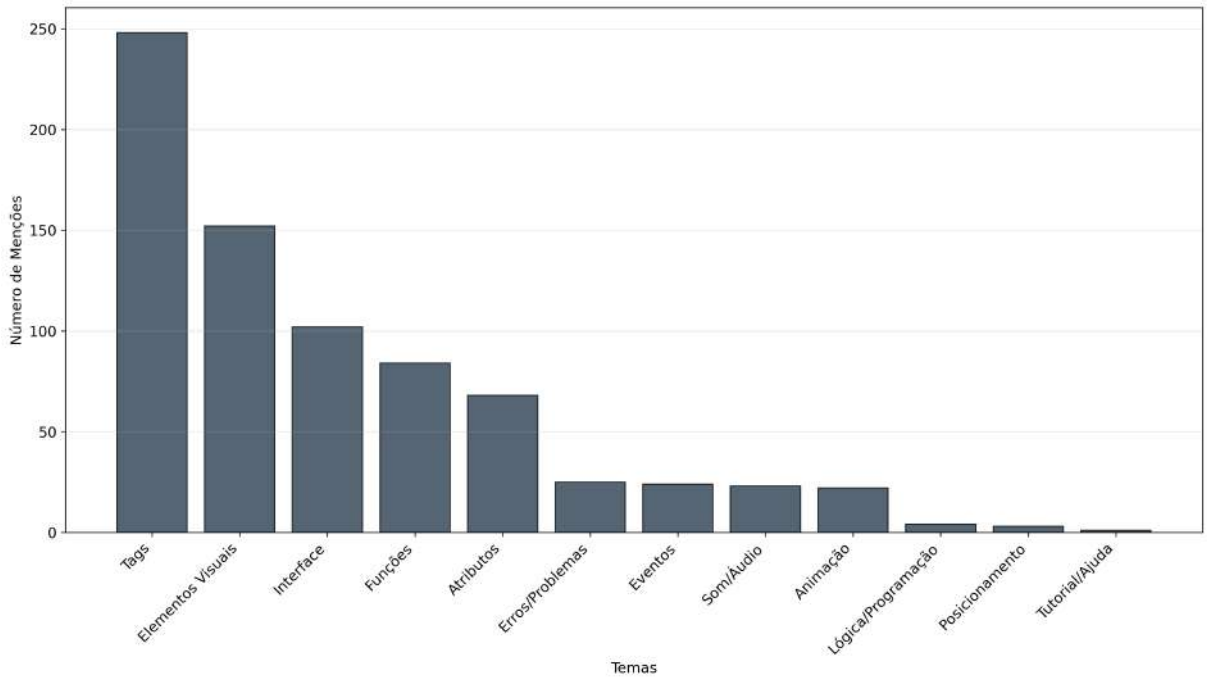
6.3.1 Metodologia de Classificação Temática

Para classificar e segmentar os tipos de perguntas enviadas, elaborou-se uma categorização temática fundamentada em critérios técnicos, semânticos e estruturais. A seguir, apresenta-se a divisão adotada:

Para fins analíticos, as mensagens foram agrupadas em categorias temáticas com base em palavras-chave presentes no corpo das perguntas. As categorias foram organizadas a partir de quatro eixos principais: (i) estrutura da DSL, com menções a tags (*agent*, *fab-if*, etc.), atributos (*name*, *when*, *default*) e funções (*set*, *goToPage*, *playSound*), (ii) aspectos técnicos de implementação, que englobam eventos (*on_click*, *hover*), lógica e controle de fluxo (*if*, *for*, variáveis) e também manipulação de som e áudio, (iii) aspectos visuais e de apresentação, incluindo elementos gráficos (imagem, texto, cor), posicionamento espacial (x, y) e animações, e por fim, (iv) aspectos de usabilidade e suporte, que envolvem questões sobre a interface, mensagens de erro e pedidos de ajuda ou exemplos.

Na Figura 31, observa-se que a maioria das perguntas concentrou-se na estrutura da linguagem, sobretudo no uso correto das tags, evidenciando o esforço dos alunos em compreender os fundamentos da DSL. Em seguida, aparecem questões voltadas à manipulação visual e, em terceiro lugar, dúvidas relacionadas a erros e problemas.

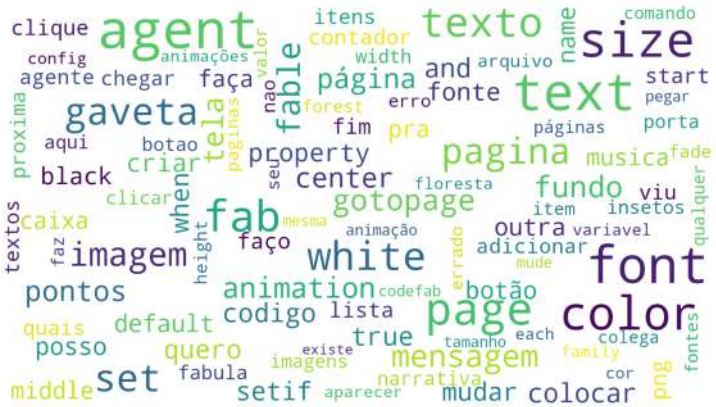
Figura 31 – Principais temas perguntados ao chat assistente



Fonte: Elaborado pelo autor.

Nuvens de palavras foram utilizadas como recurso visual para analisar o conteúdo textual criado pelos usuários durante a dinâmica. Essa técnica, como descrito por KALMUKOV (2021), permite uma visualização rápida da frequência relativa de termos em grandes volumes de texto, facilitando a identificação de padrões temáticos e áreas de maior recorrência lexical. Na Figura 32, é apresentada uma nuvem de palavras gerada a partir das mensagens enviadas ao assistente IA, evidenciando os termos mais recorrentes nas dúvidas submetidas ao longo do período de aplicação da plataforma.

Figura 32 – Nuvem de palavras pesquisadas no chat assistente



Fonte: Elaborado pelo autor.

6.4 Resultados SUS e TAM

Foram aplicadas 10 questões com base na metodologia proposta por (BROOKE et al., 1996), conforme descrito no Capítulo 5. A partir das respostas, foi gerada o *SUS Score* (SUS), cuja finalidade é mensurar a percepção de usabilidade de sistemas interativos.

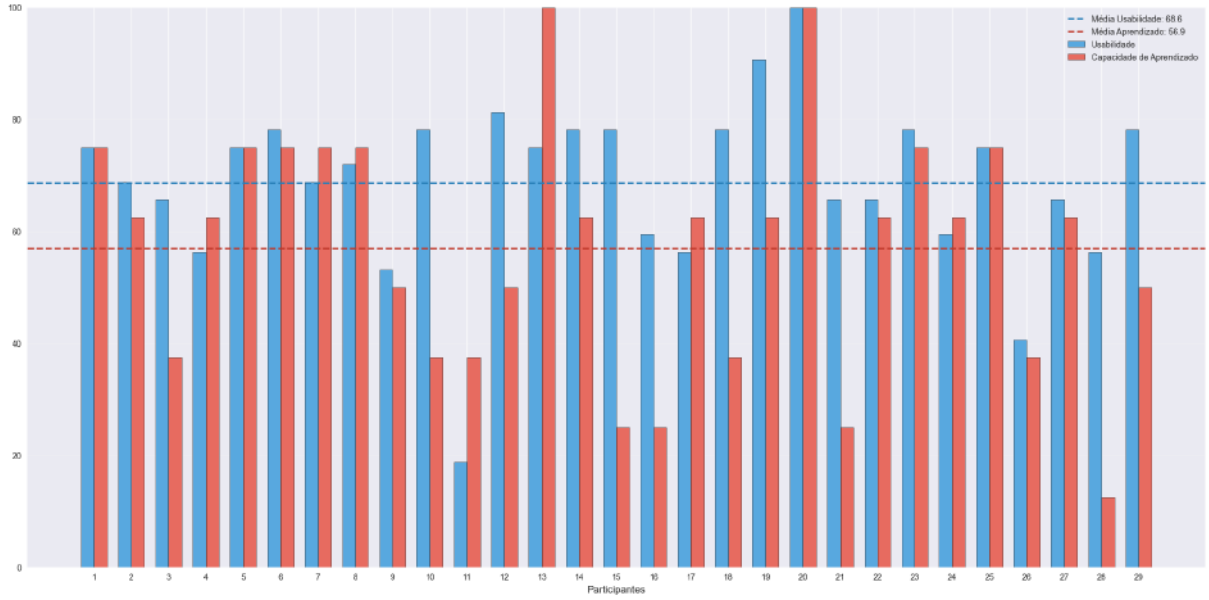
Segundo SAURO (2021), a pontuação SUS pode ser interpretada por meio de uma escala de letras (de A a F), permitindo uma classificação qualitativa dos resultados. Ainda de acordo com o autor, a média histórica do SUS, considerando mais de 500 estudos analisados, é de 68 pontos. Notas acima desse valor são consideradas acima da média, enquanto pontuações inferiores são interpretadas como abaixo da média.

A média geral obtida nesta aplicação foi de 66,29, com desvio padrão de 14,74. Essa pontuação posiciona a plataforma ligeiramente abaixo da média histórica do SUS, que é de 68 pontos, mas ainda acima do nível considerado insatisfatório. Isso sugere que, embora haja espaço para melhorias, a percepção geral de usabilidade não foi totalmente negativa.

Em comparação, o trabalho de GOMES (2022) apresentou uma pontuação SUS consideravelmente mais alta, com média de 79,70, classificando o Codefab como nível B de usabilidade. Essa pontuação reflete uma boa aceitação por parte dos usuários, especialmente no aspecto de utilidade percebida. O sistema também obteve 81,06 em usabilidade e 74,24 em capacidade de aprendizado. No entanto, é importante considerar que, apesar dos resultados expressivos, o próprio autor reconhece que a ferramenta "não é tão simples de ser utilizada em um primeiro momento", sendo necessárias mais repetições para facilitar seu uso ao longo do tempo.

Ainda em contraste com os dados obtidos no trabalho de (GOMES, 2022), observa-se, na Figura 33, uma queda significativa nos números de usabilidade (68,6) e aprendizado (56,9) segundo o modelo proposto por (SAURO, 2021). Essa redução pode ser explicada pela hipótese de que o aumento da complexidade da plataforma e da linguagem de marcação utilizada tenha elevado a barreira de entrada para usuários iniciantes. A nova versão da DSL, embora mais expressiva e funcional, incorporou estruturas condicionais, operadores dinâmicos e mecanismos de iteração que exigem do usuário maior familiaridade com lógica e abstrações computacionais. Soma-se a isso a ampliação da interface da plataforma, que passou a integrar áreas simultâneas de edição, visualização e controle de contexto, o que pode ter contribuído para uma maior sobrecarga cognitiva durante os primeiros contatos. Esses fatores, considerados em conjunto, fornecem indícios para justificar a queda nos indicadores de experiência quando comparados ao trabalho anterior.

Figura 33 – Gráfico com os resultados da Usabilidade e Capacidade de Aprendizado da avaliação de cada participante

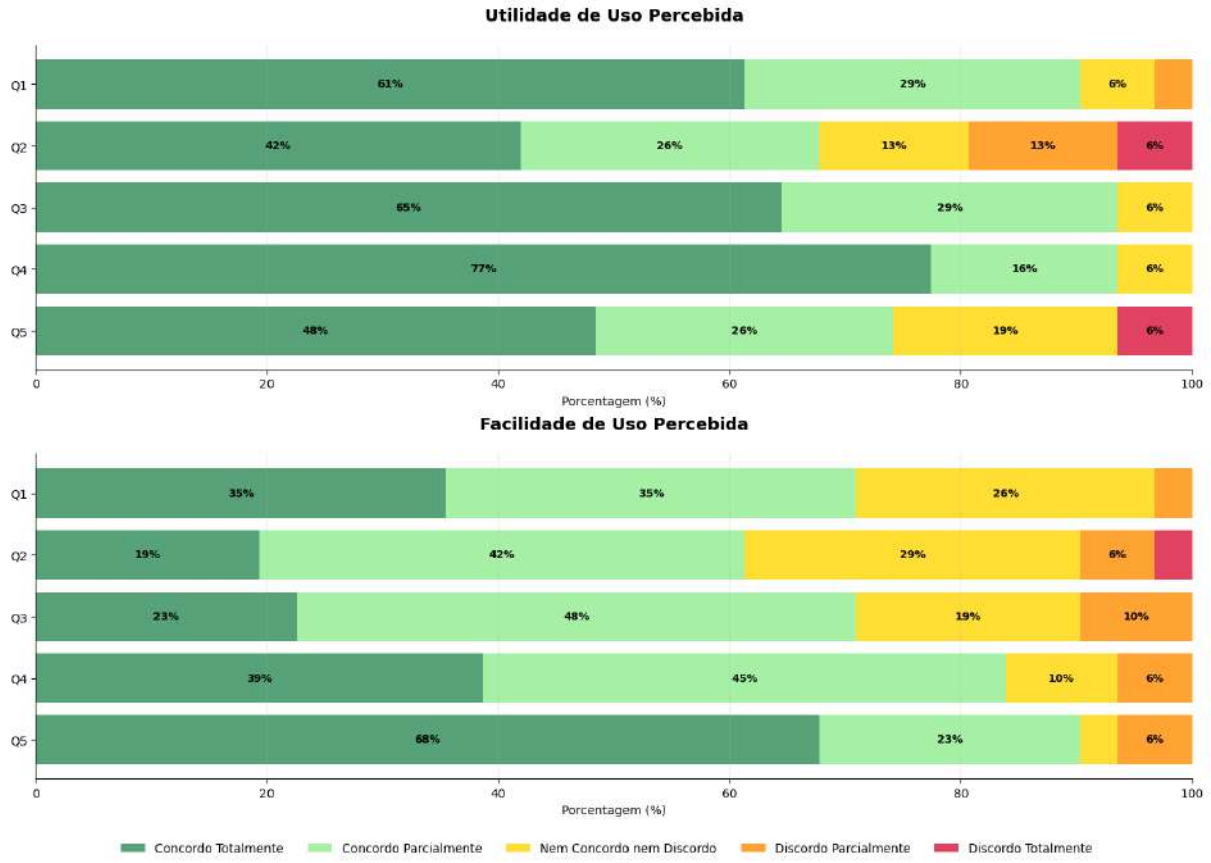


Fonte: Elaborado pelo autor.

Na Figura 34 apresentam-se os resultados do questionário TAM, organizados nos eixos Utilidade de Uso Percebida e Facilidade de Uso Percebida. No primeiro eixo, a sentença Q4 (“O sistema de versionamento integrado com GitHub é útil para o desenvolvimento”) obteve o maior índice de concordância total, com 77% das respostas. A seguir, destacaram-se Q3, que avalia a utilidade da divisão da fábula em múltiplos arquivos (65% de concordância total), e Q1, relativa à agilidade no processo de construção de fábulas (61%).

No eixo de Facilidade de Uso Percebida, a maior aprovação foi registrada em Q5 (“Importar um asset é simples e compreensível”), com 68% de concordância total. Em contrapartida, Q2 (“Construir uma fábula dentro da ferramenta é um processo simples”) apresentou o menor percentual de respostas “concordo totalmente”, limitando-se a 19%. Esse resultado sugere que, embora a plataforma seja reconhecida como útil, ainda existem obstáculos de usabilidade no fluxo de criação para usuários iniciantes.

Figura 34 – Resultados de Utilidade de Uso e Facilidade de Uso Percebida



Fonte: Elaborado pelo autor.

A confiabilidade das respostas foi verificada pelo coeficiente alfa de Cronbach (CRONBACH, 1951). Para Facilidade de Uso Percebida o valor obtido foi $\alpha = 0{,}748$, classificado como aceitável, enquanto Atitude em Relação ao Uso e Intenção de Uso apresentaram $\alpha = 0{,}738$ e $\alpha = 0{,}870$, respectivamente, indicando consistência interna aceitável e boa. Já a Utilidade de Uso Percebida obteve $\alpha = 0{,}470$, valor considerado pobre e, portanto, insuficiente para assegurar que seus cinco itens medem corretamente a confiabilidade do mesmo conceito.

Foram aplicadas três questões abertas sobre pontos positivos, negativos e sugestões de melhoria. Os participantes elogiaram a interface do CodeFab por ser "lúdica e bonita", destacando a pré-visualização em tempo real, o autocompletar, a integração direta com GitHub e a galeria de *assets*. Por outro lado, apontaram instabilidade frequente, falhas no salvamento automático, necessidade de recarregar a página, limite de 10 MB para *uploads* e dependência exclusiva do mouse em eventos. As melhorias mais demandadas incluem um salvamento automático mais confiável, aumento do limite de arquivos por upload(atualmente é permitido enviar somente 10 MB por *upload*), suporte a teclado, assistente de IA mais consistente, mais fábulas tutoriais e adaptação para dispositivos móveis.

Em síntese, os instrumentos aplicados apontam para resultados de usabilidade medianos, inferiores aos observados no trabalho anterior. Isso sugere que, apesar de se tratar de uma expansão considerável da proposta original, a nova versão da plataforma ainda requer ajustes e maior refinamento para alcançar estabilidade e posicionamento em um ranking B ou superior, conforme os critérios estabelecidos por (SAURO, 2021).

7 Conclusão

Esta nova versão do CodeFab teve como objetivo a extensão de diversas vertentes do trabalho anterior proposto por (GOMES, 2022) e, anteriormente, por (SILVA, 2020), com foco na experiência de desenvolvimento e na colaboração entre os autores das narrativas interativas.

Para avaliar a nova proposta foi planejada uma dinâmica de duas semanas na qual os usuários participaram de uma aula introdutória e, posteriormente, tiveram liberdade para explorar e utilizar a plataforma. Durante esse período puderam também tirar dúvidas com o autor por meio do WhatsApp.

Ao final da dinâmica foi possível coletar dados estatísticos de diferentes naturezas que permitiram observar tanto o comportamento dos usuários quanto o desempenho da plataforma em si. Destaca-se a redução nos marcadores de usabilidade e aprendizado mesmo com o acréscimo expressivo de funcionalidades. Esse cenário indica a necessidade de refinamento das funcionalidades já existentes antes de futuras extensões do projeto.

Com estruturas mais diretas e com melhor capacidade de abstração tornou-se mais simples mapear tópicos relacionados ao pensamento computacional conforme proposto por (WING, 2006). As funcionalidades da linguagem, como variáveis, laços, condicionais e funções, reforçam a ideia de que algum dos pilares do pensamento computacional foi exercitado mesmo nas fábulas mais simples criadas durante o processo.

7.1 Contribuições

A principal contribuição deste trabalho está na consolidação do playground web proposto por (GOMES, 2022), agora complementado por uma área social que permite a divulgação de fábulas, eliminando a dependência de plataformas terceiras como o (PADLET, 2025). Destaca-se também a formalização de uma linguagem de domínio específico com abstrações algorítmicas mínimas que, embora simplifique a criação de interações, também acentuou a curva de aprendizado da ferramenta, como evidenciado nos resultados apresentados no Capítulo 6.

7.2 Trabalhos Futuros

A plataforma descrita e estendida neste trabalho deve manter um processo contínuo de aprimoramento assim como ocorreu com sua versão anterior. Diversas melhorias se aplicam para além da camada de interface. Um exemplo é a dependência do GitHub que,

além de servir como porta de entrada para conceitos de versionamento, exige que o usuário crie uma conta na plataforma. Essa integração gera dependência de duas bases de dados (GitHub e CodeFab) que devem estar sempre sincronizadas. O ideal seria unificar essas bases para melhorar a integralidade dos dados e facilitar integrações futuras, pois assim haveria apenas uma fonte de verdade.

Uma melhoria relevante seria a introdução de um script de geração de arquivos que rodasse em modo *watch*[1], o que proporcionaria uma experiência mais fluida para desenvolvedores. Atualmente, a extensão da DSL em nível de código exige modificações em múltiplos arquivos, tornando o processo pouco flexível. Um gerador automático baseado em um esquema central, similar ao processo de geração de rotas em TypeScript fornecido pelo (TANSTACK, 2025), reduziria esse tipo de atrito ao criar automaticamente estruturas de código a partir da estrutura do esquema principal.

Devido à dependência da arquitetura React a camada de interface da plataforma e, por consequência, a FableML sofre impactos diretos do ciclo de vida da Virtual DOM[2], uma possível evolução seria a adoção de uma renderização direta em *canvas*, eliminando a necessidade de construção manual de estruturas reativas, conforme detalhado no Capítulo 4. Essa mudança poderia mitigar problemas de performance e conferir maior controle gráfico à engine da plataforma.

Além das questões técnicas, futuras versões podem incluir gamificação aplicada ao perfil do usuário e fluxos de colaboração, tais como a criação de pull requests diretamente na plataforma e mecanismos de "*check-out*" entre versões do projeto, que permitirão ao usuário reverter alterações indesejadas com maior facilidade. Essas funcionalidades serão viabilizadas pela integração com o GitHub e pelo uso de suas APIs. Por outro lado, a unificação das bases de dados (CodeFab e GitHub) impõe *trade-offs* que devem ser criteriosamente avaliados, não apenas sob o ponto de vista técnico, como latência, consistência e segurança, mas também econômico. O GitHub é utilizado como CDN para imagens, vídeos e áudios das fábulas, o que reduz a zero o custo de banda para hospedagem desses ativos. Em contrapartida, tal dependência pode restringir a flexibilidade em personalizações futuras e gerar múltiplos pontos de falha em caso de indisponibilidades ou mudanças nas políticas do provedor.

[1] Um processo Node.js que monitora alterações no esquema central da DSL e gera automaticamente os arquivos necessários, mantendo a estrutura de diretórios e atualizando as tipagens em TypeScript sem intervenção manual do desenvolvedor.

[2] A Virtual DOM é uma representação em memória da árvore de elementos HTML que o React mantém para otimizar atualizações de tela.

Referências

BORK, D. e. Language server protocol: An introduction to the protocol, its use, and adoption for web modeling tools. *EMISA Journal*, v. 18, n. 1, p. 24–46, 2023. Acesso em: 14 jul. 2025. Disponível em: <https://emisa-journal.org/emisa/article/view/320/210>. Citado na página 48.

BROOKE, J. et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, London, England, v. 189, n. 194, p. 4–7, 1996. Citado 2 vezes nas páginas 59 e 68.

BUSSON, A.; DAMASCENO, A.; LIMA, T.; NETO, C. S. Scenesync: A hypermedia authoring language for temporal synchronism of learning objects. In: . [S.l.: s.n.], 2016. p. 175–182. Citado na página 17.

CLARITY. *Clarity – Free Behavior Analytics Tool for Websites*. 2025. <https://clarity.microsoft.com/>. Acesso em: 25 jul. 2025. Citado 2 vezes nas páginas 61 e 62.

CODEMIRROR. *CodeMirror 6 — Reference Manual*. 2025. <https://codemirror.net/docs/ref/>. Acesso em: 14 jul. 2025. Citado 2 vezes nas páginas 37 e 47.

CODESANBOX. *CodeSandbox: Instant Cloud Development Environments — codesandbox.io*. 2025. <https://codesandbox.io/>. Acesso em: 25 jul. 2025. Citado na página 15.

CODESANDBOX. *codesandbox.io*. 2025. <https://codesandbox.io/templates>. Acesso em: 25 jul. 2025. Citado na página 15.

CRONBACH, L. J. Coefficient alpha and the internal structure of tests. *Psychometrika*, v. 16, n. 3, p. 297–334, set. 1951. Citado na página 70.

CRUZ, P. T. M. B. d. *Padrões de interação para ferramentas de autoria de storytellings infantis*. Dissertação (Dissertação (Mestrado em Design)) — Universidade Federal do Maranhão, São Luís, MA, 2016. Citado na página 17.

DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, Management Information Systems Research Center, University of Minnesota, v. 13, n. 3, p. 319–340, 1989. Citado na página 59.

DIAS. *Codefab*. 2025. <https://codefab-v2.vercel.app/dashboard>. Acesso em: 25 jul. 2025. Citado na página 19.

FOWLER, M. *Parser Fear*. 2008. <https://martinfowler.com/bliki/ParserFear.html>. Acesso em: 25 jul. 2025. Citado na página 38.

FOWLER, M.; PARSONS, R. *Domain-Specific Languages*. Upper Saddle River, NJ: Addison-Wesley, 2011. ISBN 9780321712943. Citado 2 vezes nas páginas 37 e 38.

GESTEVES, R. *Zustand, When, how and why — dev.to*. 2024. <https://dev.to/ricardogesteves/zustand-when-how-and-why-1kpi>. Acesso em: 25 jul. 2025. Citado na página 24.

GITHUB. *Managing your personal access tokens - GitHub Docs — docs.github.com*. 2025. <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens>. Acesso em: 25 jul. 2025. Citado na página 62.

GITHUB. *Scopes for OAuth apps - GitHub Docs — docs.github.com*. 2025. <https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/scopes-for-oauth-apps>. Acesso em: 25 jul. 2025. Citado na página 20.

GOMES, M. M. Playground web para aplicações hipermídia não-lineares. *Universidade Federal do Maranhão*, 2022. Citado 12 vezes nas páginas 15, 18, 19, 25, 27, 29, 30, 38, 57, 64, 68 e 72.

HARDT, D. *The OAuth 2.0 Authorization Framework*. RFC Editor, 2012. RFC 6749. (Request for Comments, 6749). Disponível em: <https://www.rfc-editor.org/info/rfc6749>. Citado na página 28.

HUDAK, P. Building domain-specific embedded languages. *ACM Computing Surveys (CSUR)*, ACM, v. 28, n. 4es, p. 196, 1996. Citado 4 vezes nas páginas 37, 38, 39 e 62.

JIANG SHAOKANG; COBLENZ, M. An analysis of the costs and benefits of autocomplete in ides. *Proceedings of the ACM on Software Engineering*, Association for Computing Machinery, New York, NY, USA, v. 1, n. FSE, p. 1–23, 2024. Acesso em: 25 jul. 2025. Disponível em: <https://doi.org/10.1145/3660765>. Citado na página 51.

KALMUKOV, Y. *Using Word Clouds for Fast Analysis of Textual Survey Data*. 2021. <https://arxiv.org/abs/2112.14861>. Preprint. Disponível em: https://arxiv.org/abs/2112.14861. Acesso em: 25 jul. 2025. Citado na página 67.

LEONIDAS-FROM-XIV. *xml2js — XML to JavaScript object converter*. 2025. <https://www.npmjs.com/package/xml2js>. Acesso em: 25 jul. 2025. Citado na página 37.

PADLET. *Padlet - Visual Collaboration for Creative Work and Education — padlet.com*. 2025. <https://padlet.com/>. Acesso em: 25 jul. 2025. Citado 3 vezes nas páginas 16, 19 e 72.

PINTO, H. F. *Autoria de e-books Interativos: modelo conceitual fábulas e requisitos*. Dissertação (Mestrado) — Universidade Federal do Maranhão, 2017. Citado 2 vezes nas páginas 15 e 17.

PYGITHUB. *PyGithub*. 2025. <https://pygithub.readthedocs.io/en/stable/>. Acesso em: 25 jul. 2025. Citado na página 62.

PYTHON. *xml.etree.ElementTree — The ElementTree XML API*. [S.l.], 2024. Disponível em: <https://docs.python.org/3/library/xml.etree.elementtree.html>. Acesso em: 25 jul. 2025. Citado na página 62.

REACT. *React — react.dev*. 2025. <https://react.dev/>. Acesso em: 25 jul. 2025. Citado na página 24.

REACT CODEMIRROR. *@uiw/react-codemirror - React Component for CodeMirror6*. 2025. <https://uiwjs.github.io/react-codemirror/>. Acesso em: 14 jul. 2025. Citado na página 47.

REDHAT. *O que é serverless? — redhat.com*. 2025. <https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-serverless>. Acesso em: 25 jul. 2025. Citado na página 25.

ROOK, R. *Tailwind for productivity - iO tech_hub — techhub.iodigital.com*. 2023. <https://techhub.iodigital.com/articles/tailwind-for-productivity>. Acesso em: 25 jul. 2025. Citado na página 24.

SAURO, J. *Measuring Usability with the System Usability Scale (SUS)*. 2021. <https://measuringu.com/sus>. Acesso em: 25 jul. 2025. Citado 2 vezes nas páginas 68 e 71.

SCRATCH. *Scratch - Imagine, Program, Share — scratch.mit.edu*. 2025. <https://scratch.mit.edu/>. Acesso em: 25 jul. 2025. Citado na página 15.

SILVA, A. T. Fablejs: biblioteca para criação de histórias interativas. *Universidade Federal do Maranhão*, 2020. Citado 7 vezes nas páginas 15, 17, 18, 19, 38, 64 e 72.

STACKBLITZ. *StackBlitz | Instant Dev Environments | Click. Code. Done. — stackblitz.com*. 2025. <https://stackblitz.com/>. Acesso em: 25 jul. 2025. Citado na página 15.

SUPABASE. *Database | Supabase Docs — supabase.com*. 2025. <https://supabase.com/docs/guides/database/overview>. Acesso em: 25 jul. 2025. Citado na página 27.

SUPABASE. *Login with GitHub | Supabase Docs — supabase.com*. 2025. <https://supabase.com/docs/guides/auth/social-login/auth-github?queryGroups=language&language=js&queryGroups=environment&environment=client&queryGroups=framework&framework=nextjs>. Acesso em: 25 jul. 2025. Citado na página 29.

TAILWIND. *Installing with Vite - Installation — tailwindcss.com*. 2025. <https://tailwindcss.com/docs/installation/using-vite>. Acesso em: 25 jul. 2025. Citado na página 24.

TANSTACK. *Overview | TanStack Router React Docs — tanstack.com*. 2025. <https://tanstack.com/router/latest/docs/framework/react/overview>. Acesso em: 25 jul. 2025. Citado na página 73.

VAGIAS, W. M. *Likert-Type Scale Response Anchors*. Clemson, SC, 2006. Acesso em: 24 jul. 2025. Disponível em: <https://media.clemson.edu/cbshs/prtm/research/resources-for-research-page-2/Vagias-Likert-Type-Scale-Response-Anchors.pdf>. Citado na página 58.

VENKATESH, V. e. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science*, v. 46, p. 186–204, 2000. Citado na página 59.

VERCEL. *Next.js Docs | Next.js — nextjs.org*. 2025. <https://nextjs.org/docs>. Acesso em: 25 jul. 2025. Citado na página 25.

VERCEL: Build and deploy the best web experiences with the AI Cloud – Vercel — vercel.com. <https://vercel.com/>. Acesso em: 25 jul. 2025. Citado na página 23.

WING, J. M. Computational thinking. *Communications of the ACM*, ACM, v. 49, n. 3, p. 33–35, 2006. Citado 5 vezes nas páginas 15, 40, 62, 63 e 72.

ZUSTAND. *Introduction - Zustand — zustand.docs.pmnd.rs*. 2025. <https://zustand.docs.pmnd.rs/getting-started/introduction>. Acesso em: 25 jul. 2025. Citado na página 24.

Apêndices

APÊNDICE A – Questionário de pesquisa

Questionário de avaliação do Codefab

Termo de Consentimento Livre e Esclarecido
Concordo em participar do estudo que tem como pesquisador responsável o aluno de graduação Lukas Henrique Braga Dias, do Departamento de Informática, sob a orientação do professor Carlos de Salles Soares Neto da Universidade Federal do Maranhão - UFMA, que pode ser contatado pelo e-mail dias.lukas@discente.ufma.br
Tenho ciência de que o estudo tem em vista realizar entrevistas com alunos de graduação visando, por parte do(a) referido(a) aluno(a) a realização da sua monografia de conclusão de curso, cujo título é "Estendendo a Plataforma Codefab para Colaboração e Divulgação de Fábulas Interativas.".
Minha participação consistirá em conceder uma entrevista que será gravada e transcrita. Entendo que esse estudo possui finalidade de pesquisa acadêmica, que os dados obtidos serão divulgados para fins de pesquisa científica e publicados no seu trabalho.
O aluno providenciará uma cópia da transcrição da entrevista para meu conhecimento. Além disso, sei que posso abandonar minha participação na pesquisa quando quiser e que não receberei nenhum pagamento por esta participação.
São Luís, 16 de junho de 2025.

* Indica uma pergunta obrigatória

1. E-mail *

Informações Pessoais

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

3. Qual o seu nome completo? *

4. Qual sua idade? *

_____

5. Com qual gênero você mais se identifica? *

Marcar apenas uma oval.

- ◯ Feminino
- ◯ Masculino
- ◯ Não-binário
- ◯ Prefiro não informar

Perfil Acadêmico

6. Há quanto tempo você tem contato com programação? *

Marcar apenas uma oval.

- ◯ Menos de 6 meses
- ◯ 6 meses a 1 ano
- ◯ 1 – 2 anos
- ◯ 2 – 4 anos
- ◯ Mais de 4 anos

Avaliação da Plataforma

7. Eu gostaria de usar o CodeFab frequentemente *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

8. Eu achei o CodeFab desnecessariamente complexo *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

9. Eu achei o CodeFab fácil de usar *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

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

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

11. Eu achei que as várias funções do CodeFab estavam bem integradas *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

12. Eu achei que havia muita inconsistência no CodeFab *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

13. Eu imagino que a maioria das pessoas aprenderia a usar o CodeFab rapidamente *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

14. Eu achei o CodeFab muito complicado de usar *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

15. Eu me senti confiante ao usar o CodeFab *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

16. Eu precisei aprender várias coisas novas antes de conseguir usar o CodeFab *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

Utilidade Percebida

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

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

18. Os assets disponibilizados na galeria são úteis para construir fábulas *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

19. A capacidade de dividir a fábula em múltiplos arquivos facilita a organização *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

20. O sistema de versionamento integrado com GitHub é útil para o desenvolvimento *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

21. O chat integrado com IA é útil para tirar dúvidas durante o desenvolvimento *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

Facilidade de Uso Percebida

22. Aprender a usar o CodeFab foi fácil *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

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

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

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

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

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

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

26. Importar um asset é simples e compreensível *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

Atitude em Relação ao Uso

27. Usar o CodeFab é uma boa ideia *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

28. Usar o CodeFab é uma escolha inteligente *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

29. Eu gosto da ideia de usar o CodeFab para criar fábulas *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

30. Usar o CodeFab é prazeroso *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

31. A ferramenta incentiva a criatividade na construção de narrativas *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

Intenção de Uso

32. Eu pretendo usar o CodeFab no futuro *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

33. Eu prevejo que usarei o CodeFab regularmente *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

34. Eu recomendaria o CodeFab para outros estudantes *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

35. Eu planejo continuar usando o CodeFab para criar fábulas *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

36. Eu usaria o CodeFab em projetos futuros *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

Opiniões e Ideias

Sinta-se à vontade para listar vários pontos positivos, negativos ou novas ideias

37. Liste todos os pontos positivos que você percebeu no CodeFab, se houver.

38. Quais dificuldades ou pontos negativos encontrou ao usar o CodeFab?

39. Que melhorias ou novos recursos você gostaria de ver no CodeFab?