

**USABILIDADE, DESENVOLVIMENTO WEB, MOBILE E JOGOS**

**Projeto A3**

Salvador

2023

**EQUIPE:**

JAILSON RODRIGUES DE NEIVA

12722131344

CARLOS HENRIQUE BRAGA BOMFIM

12723211573

ARTHUR REINA LYRA

12722124645

ISAC DANIEL PEREIRA DE ALMEIDA

12723116417

MATHEUS DO É SANTOS

1272229444

Projeto A3

Trabalho apresentado para fins avaliativos da Unidade Curricular – USABILIDADE, DESENVOLVIMENTO WEB, MOBILE E JOGOS- correspondente ao ano letivo 2023.2

Orientador: Adailton, Lucas.

SALVADOR

2023

**SUMÁRIO**

[1 INTRODUÇÃO 7](#_Toc153227835)

[2 DESENVOLVIMENTO 8](#_Toc153227836)

[2.1 DESENVOLVENDO A EXPERIÊNCIA DO USUÁRIO 8](#_Toc153227837)

[2.1.1 Wireframes 8](#_Toc153227838)

[2.1.2 PROTOTIPAGEM 11](#_Toc153227839)

[2.2 DESENVOLVENDO A INTERFACE 15](#_Toc153227840)

[2.3 DESENVOLVENDO A API 18](#_Toc153227841)

[2.3.1 AUTENTICAÇÃO 24](#_Toc153227842)

[2.3.2 Bcrypt 25](#_Toc153227843)

[2.4 INTEGRANDO A INTERFACE COM A API 26](#_Toc153227844)

[REFERÊNCIAS 29](#_Toc153227845)

# 1 INTRODUÇÃO

O objetivo deste trabalho acadêmico é desenvolver uma aplicação web de ponta a ponta, utilizando tecnologias atuais de forte presença no mercado para desenvolver esse tipo de aplicação, como o React, para o desenvolvimento de interfaces web, e o Node.js, para o desenvolvimento de uma API REST. Além disso, a experiência do usuário foi desenvolvida considerando as heurísticas de Nielsen e os princípios que regem os critérios de usabilidade. Por fim, integramos a interface que foi prototipada e elaborada com a API REST desenvolvida em Node.js para tornar o projeto funcional.

# 2 DESENVOLVIMENTO

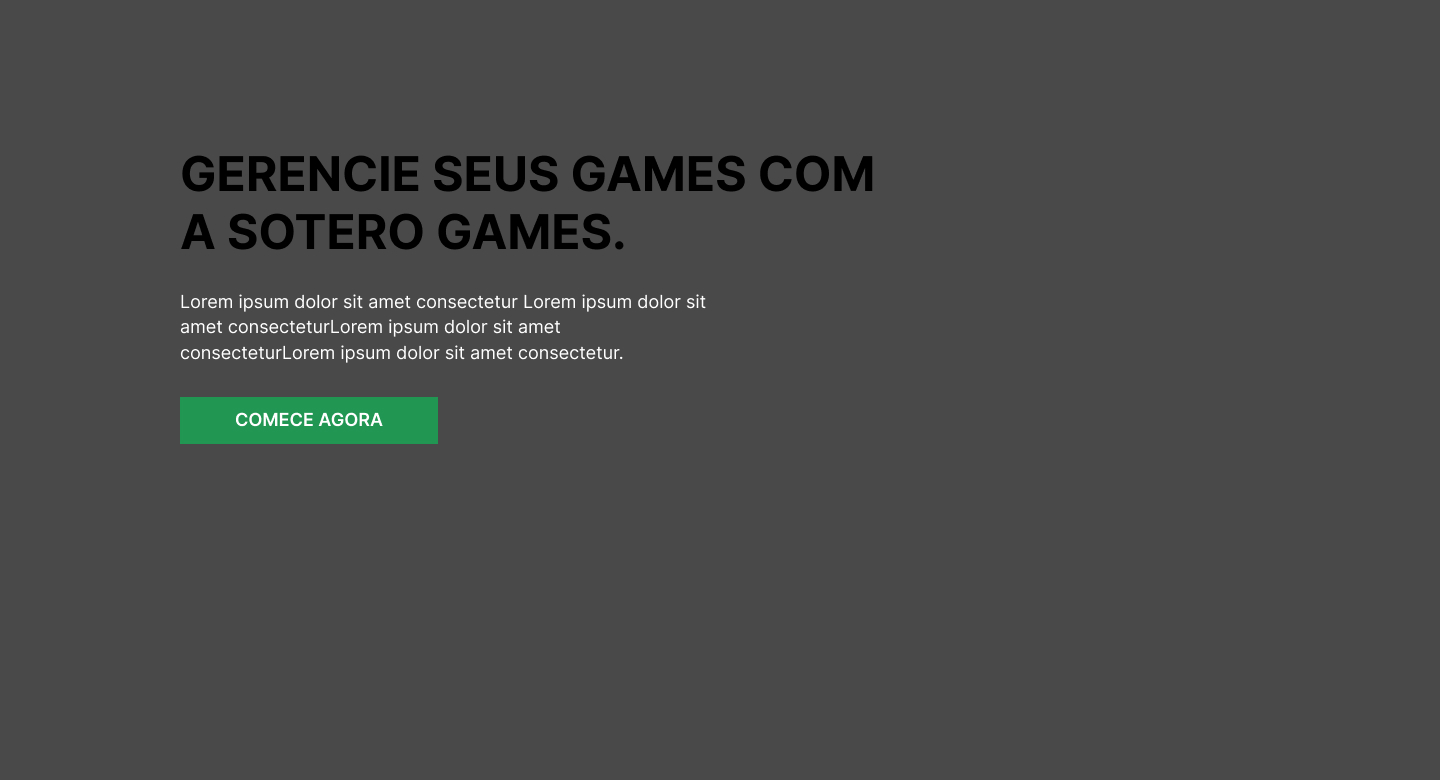
## 2.1 DESENVOLVENDO A EXPERIÊNCIA DO USUÁRIO

O projeto foi elaborado a partir do levantamento dos requisitos baseado no formato solicitado para estruturação do projeto. Nesse contexto, foi desenvolvido uma biblioteca pessoal de jogos, onde o usuário tem a possibilidade de gerir a biblioteca com os títulos organizados em categorias, tais como: jogar, jogado ou que tem interesse em jogar, e assim por diante. Desse modo, foi iniciado o desenvolvimento do fluxo do usuário a partir dos wireframes e prototipação da experiência.

## 2.1.1 Wireframes

Para elaboração dos Wireframes e um protótipo de alta fidelidade foi utilizado a plataforma Figma. Além disso, wireframes são um esboço da experiência do usuário, sendo indispensável para criar a estrutura de como será a experiência e arquitetura de informação da aplicação, sem considerar os aspectos que envolvem o design (cores, formas, etc) do projeto. Desse modo, a construção dos wireframes resultou em um fluxo que consiste de uma landing page, onde o usuário tem seu primeiro contato com a aplicação e através de um botão *call to action*, o usuário é redirecionado para a página de cadastro da aplicação.

**Figura 1:** Landing page.



**Fonte:** Autoria própria

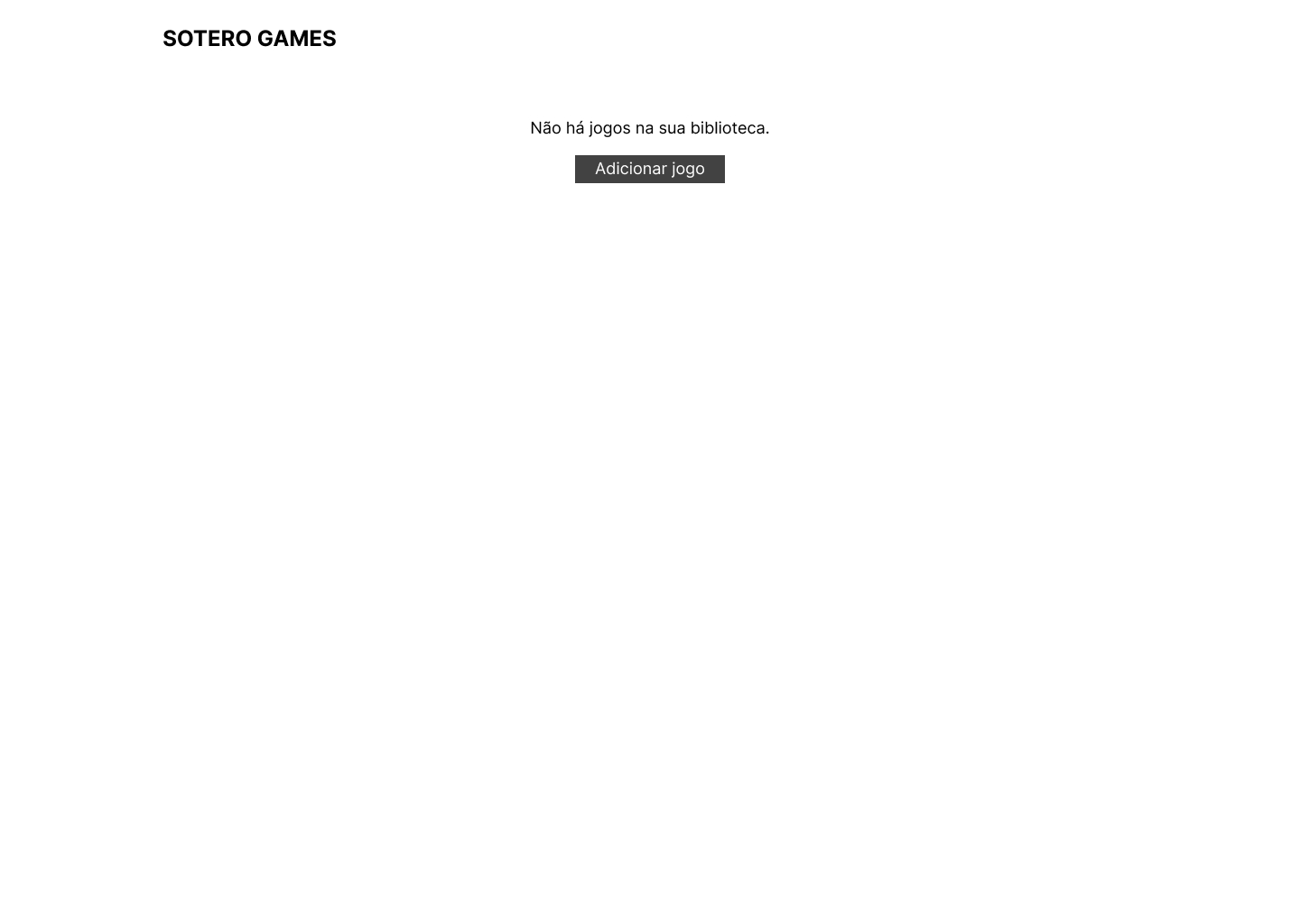
**Figura 2:** Página de Cadastro

****

**Fonte:** Autoria Própria

Após o cadastro, o usuário será redirecionado para a página inicial da biblioteca de jogos, onde é possível adicionar jogos à biblioteca. Nessa página, há um botão (Adicionar jogo) que ao ser clicado direciona o usuário para a tela de adição de um jogo na biblioteca.

**Figura 3:** Biblioteca



**Fonte:** Autoria Própria

**Figura 4:** Página de adição de jogo a biblioteca



**Fonte:** Autoria Própria

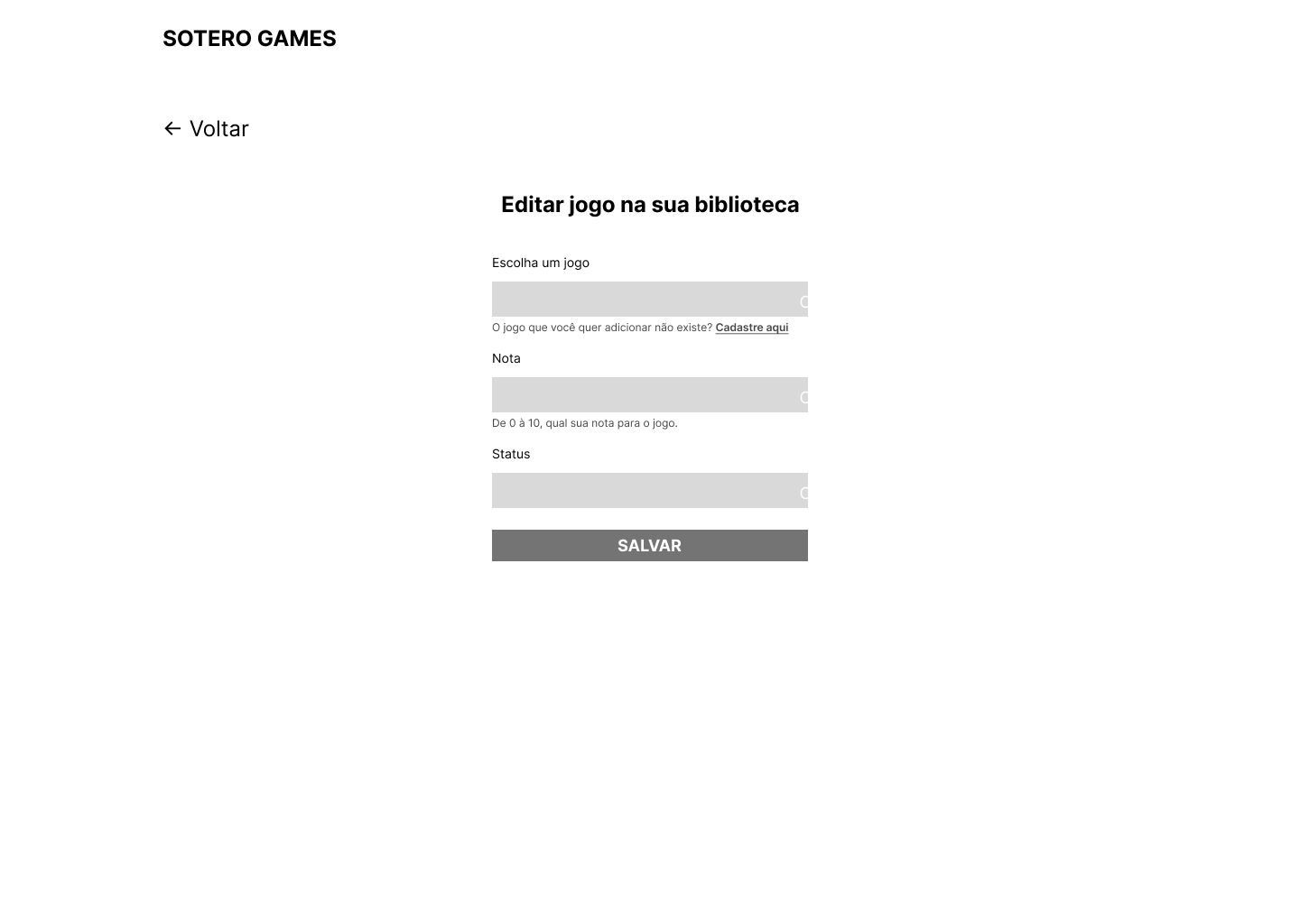
Dessa forma, ao clicar no botão “Adicionar e Voltar”, o usuário será redirecionado para a página da biblioteca com o jogo já cadastrado a ela, e com a possibilidade de editar o jogo que foi adicionado ao clicar na listagem de jogos. Além disso, na tela de "Adicionar jogo”, o usuário pode continuar adicionando novos jogos ao clicar no botão “Adicionar outro jogo”.

**Figura 5**: Página inicial da biblioteca

****

**Fonte:** Autoria Própria

**Figura 6:** Página de edição de jogo

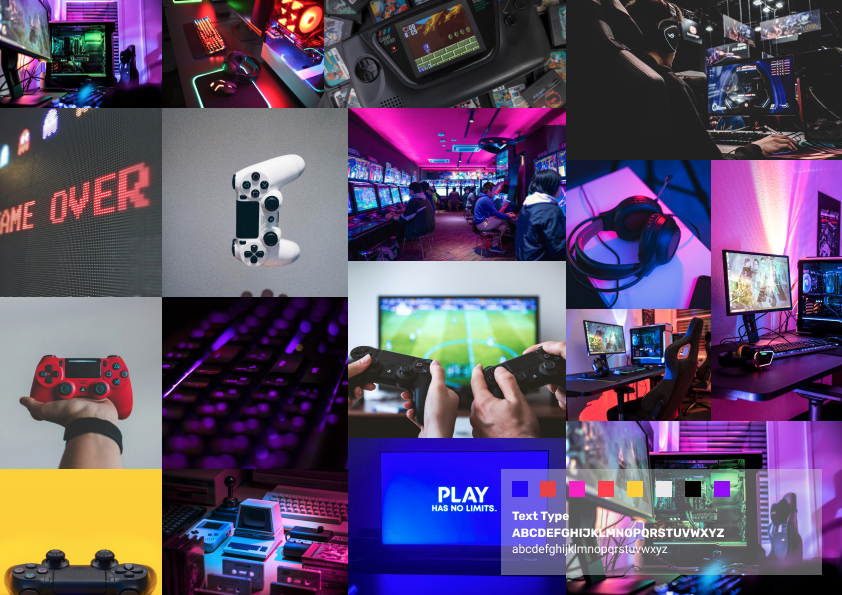
****

**Fonte:** Autoria Própria

## 2.1.2 PROTOTIPAGEM

Assim como os wireframes, o protótipo foi desenvolvido utilizando o Figma. A prototipagem foi iniciada a partir da criação de uma *Moodboard*, que seria uma técnica para extrair o design da aplicação através de um quadro de imagens, definindo o *feeling* do visual da experiência do usuário. Ela é utilizada para extrair os elementos visuais que irão compor a interface do usuário, tais como: as cores, fontes e as formas que serão úteis na criação da interface do usuário.

**Figura 6:** Moodboard

****

**Fonte:** Autoria Própria

Dessa forma, após a criação da *moodboard* e a extração das cores, fontes, e formas a serem utilizadas na experiência, foi iniciada a prototipação das telas tendo como ponto de partida a estrutura construída na elaboração dos wireframes.

**Figura 7:** Landing page

****

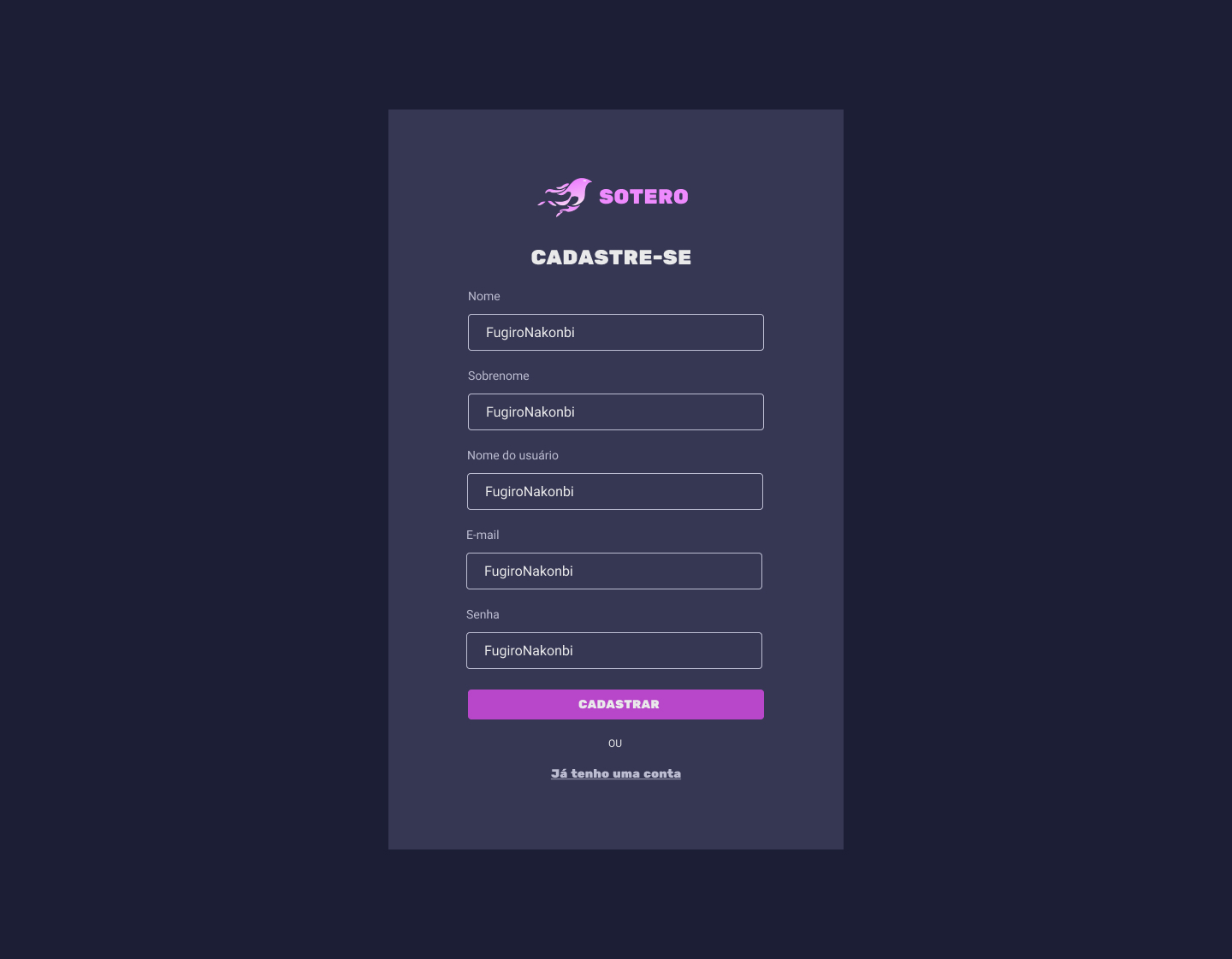
**Fonte:** Autoria Própria

**Figura 8:** Login

****

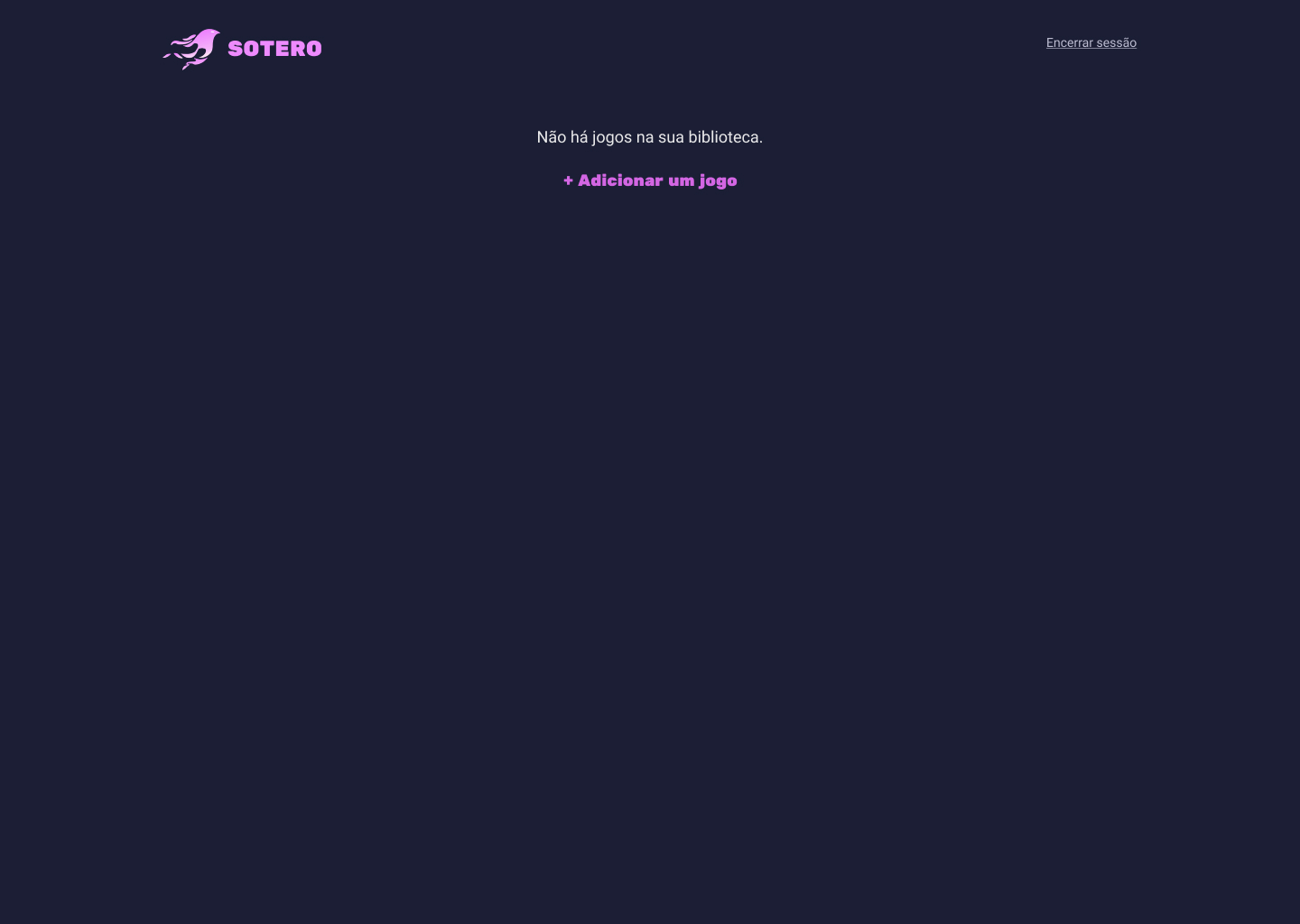
**Fonte:** Autoria Própria

**Figura 9** - Cadastro

****

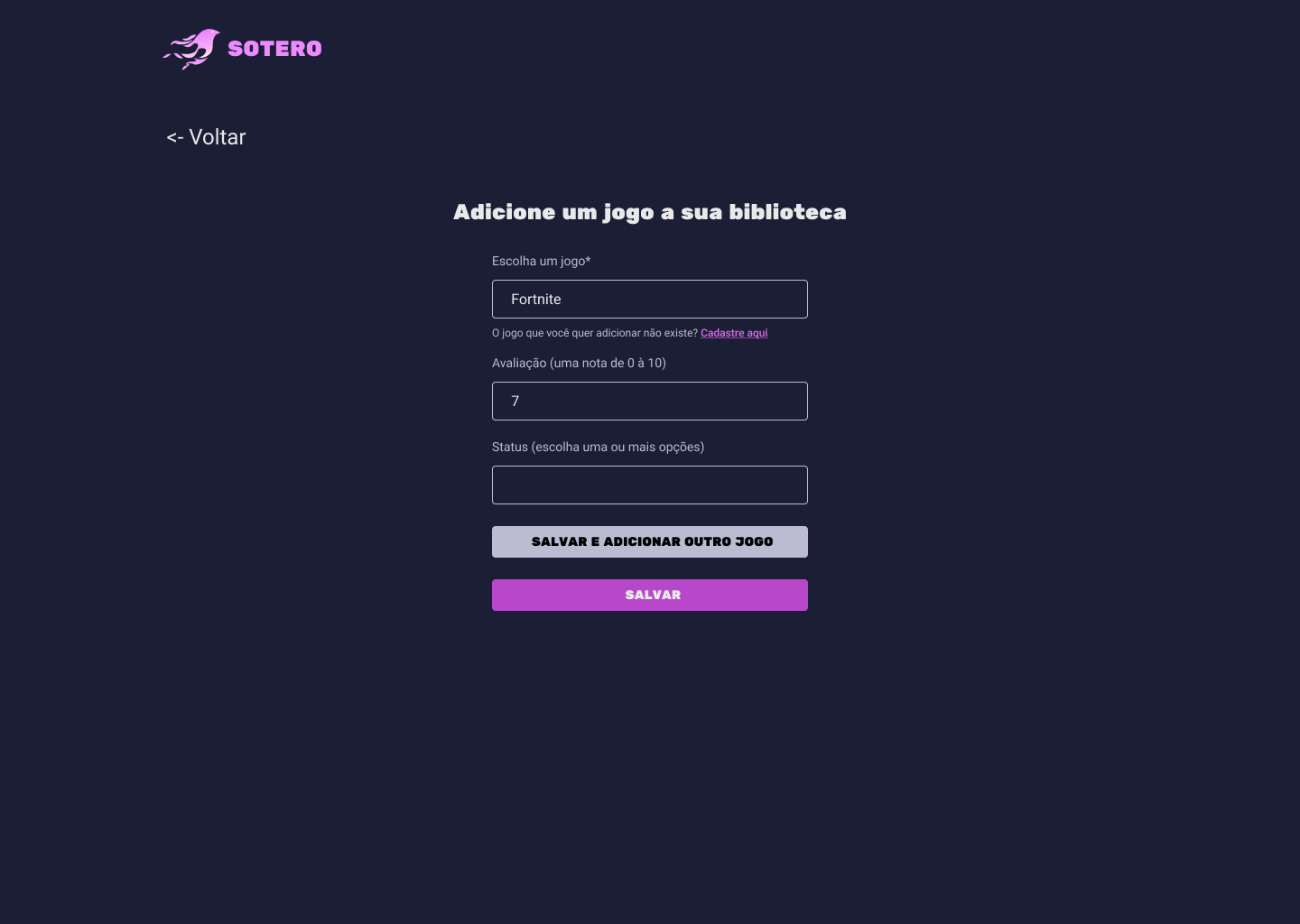
**Fonte:** Autoria Própria

**Figura 10** — Biblioteca

****

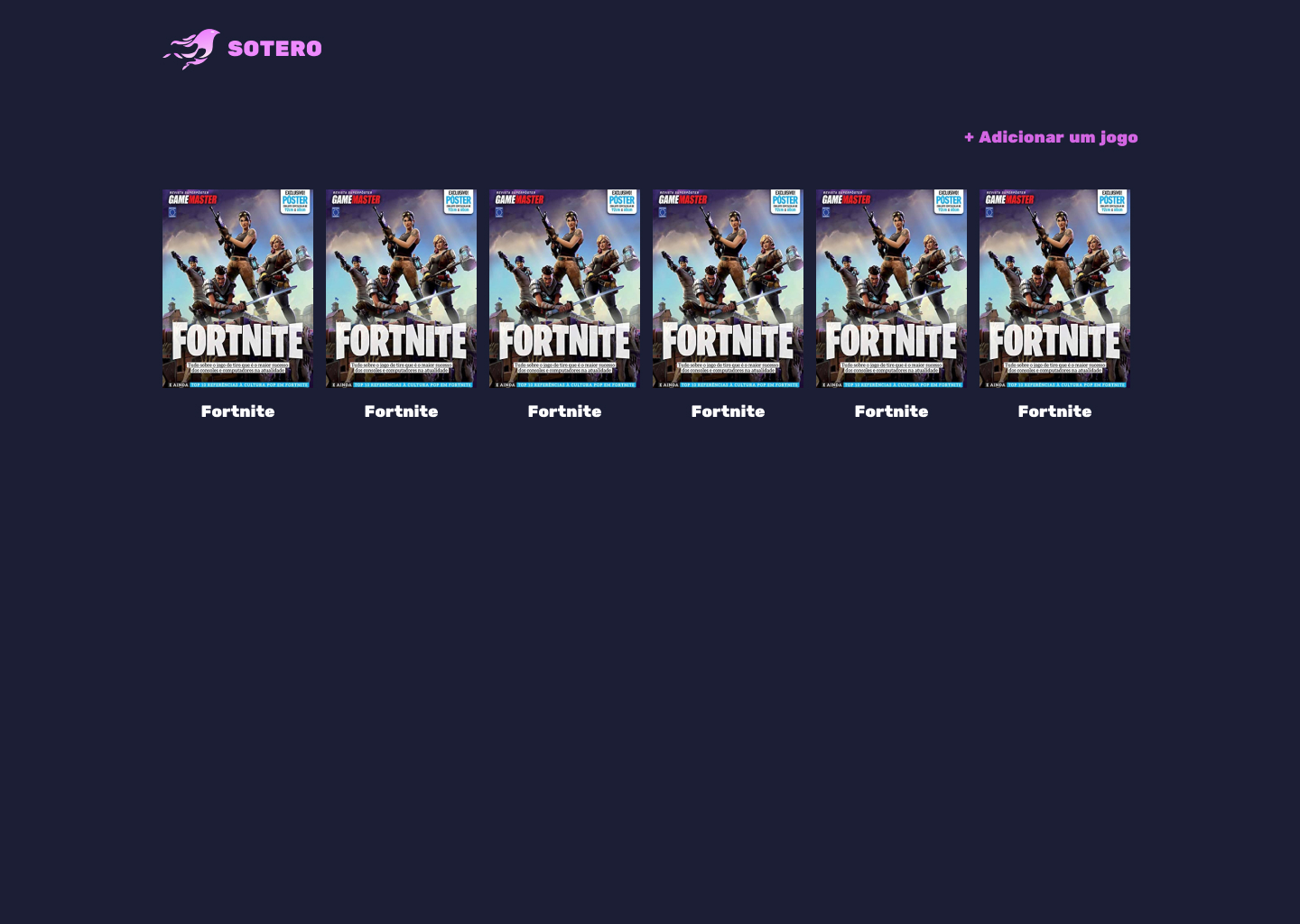
**Fonte:** Autoria Própria

**Figura 11** — Adicionar jogo a biblioteca

****

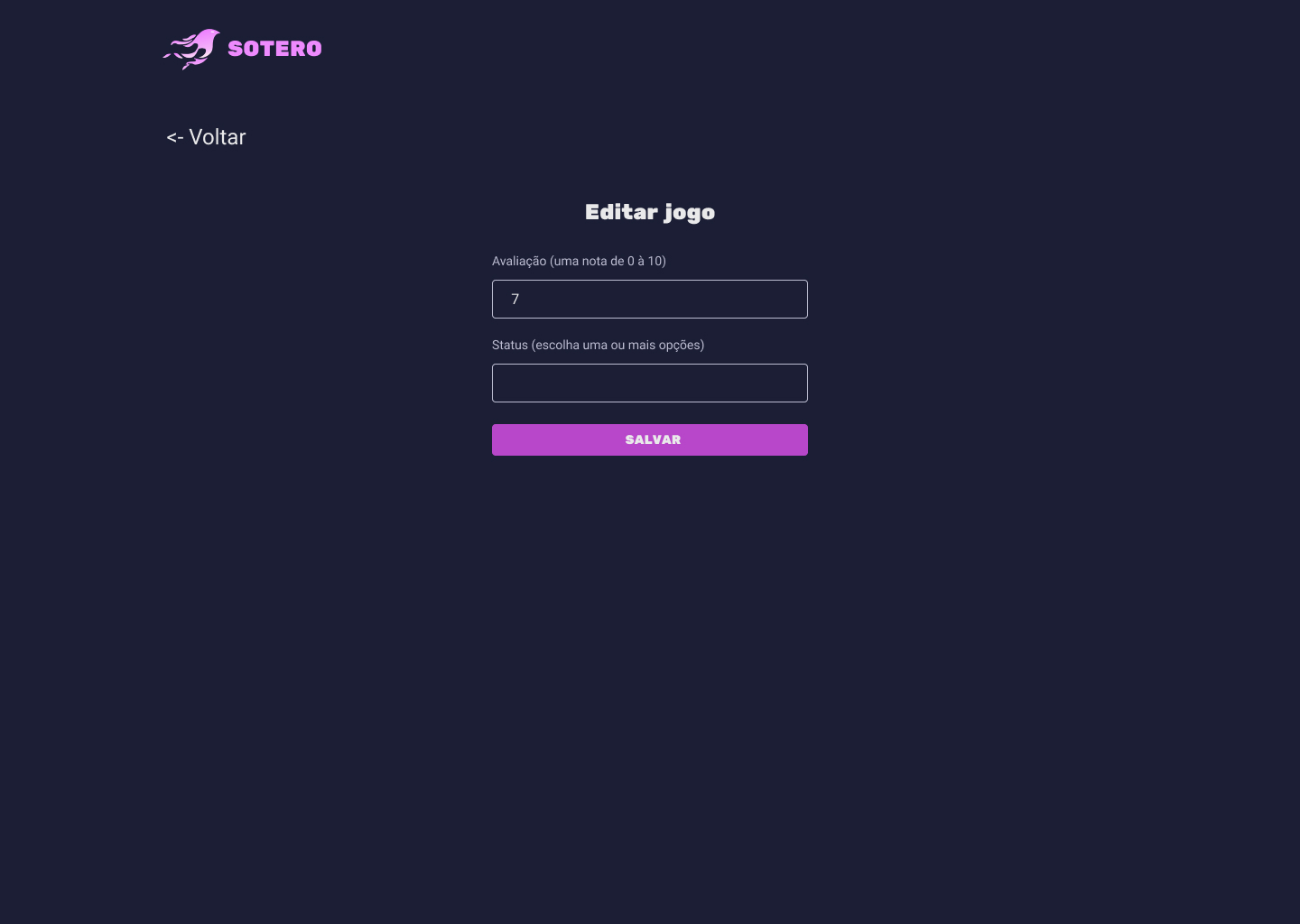
**Fonte:** Autoria Própria

**Figura 12** — Biblioteca com jogos cadastrados

****

**Fonte:** Autoria Própria

**Figura 13** — Editar jogo da biblioteca

****

**Fonte:** Autoria Própria

## 2.2 DESENVOLVENDO A INTERFACE

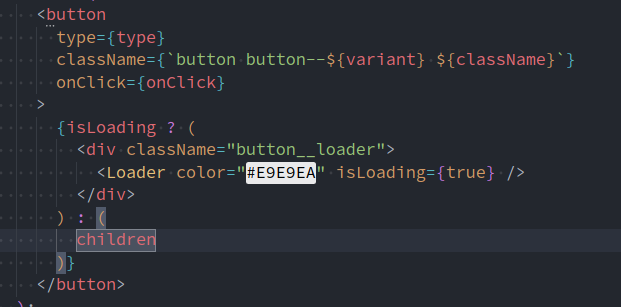
Após a criação do protótipo de alta-fidelidade no Figma, foi iniciado o desenvolvimento da interface utilizando o React. O React é uma biblioteca JavaScript cujo objetivo principal é facilitar o desenvolvimento de interfaces web (e mobile com o React Native).

Nesse contexto, utiliza-se o princípio de componentização, em que a interface é dividida em pequenas partes chamadas de componentes. Cada componente tem uma estrutura, lógica, e estilização própria que pode ser reutilizada em toda a aplicação. Um dos objetivos de se utilizar componentes é reduzir a repetição de código.

No React, diferente do Angular e do Vue.js, os quais são dois frameworks JavaScript que competem no mercado com React, todo o componente é feito utilizando o JavaScript (exceto a estilização, que também pode ser feito em JavaScript utilizando *CSS in JS* com o *styled-components*, por exemplo). Já no Angular, além do JavaScript, tanto o HTML quanto o CSS são utilizados na construção do componente. Outro ponto divergente é que tanto o Angular quanto o Vue.js utilizam uma sintaxe que adiciona recursos ao HTML, como: interpolação de dados, renderização condicional ou renderização iterada por uma lista de dados, chamadas de diretivas. No React não há necessidade de aprender a trabalhar com diretivas, já que o próprio JavaScript é utilizado para adicionar lógica de programação na renderização da estrutura dos componentes.

Ademais, na estruturação dos componentes, o React utiliza uma “linguagem de marcação própria” denominada: JSX. O JSX é uma notação que se assemelha com o XML e o HTML, onde a estrutura dos componentes pode ser feita como se estivesse sendo escrita em HTML, só que com algumas diferenças importantes. A primeira é que como a marcação do componente é elaborada no JavaScript, há alguns nomes de propriedades que se conflitam com palavras reservadas do próprio JavaScript. Por exemplo, no JSX se utiliza a propriedade className no lugar da propriedade class, pois o *class* é uma palavra reservada do JavaScript. Outra diferença importante é que o JSX permite interpolar dados e código JavaScript de forma dinâmica na estrutura de marcação do componente. É possível também utilizar loops para aproveitar estruturas que irão ser reutilizadas na renderização de uma lista de dados, como no exemplo abaixo reduzindo a repetição de código:

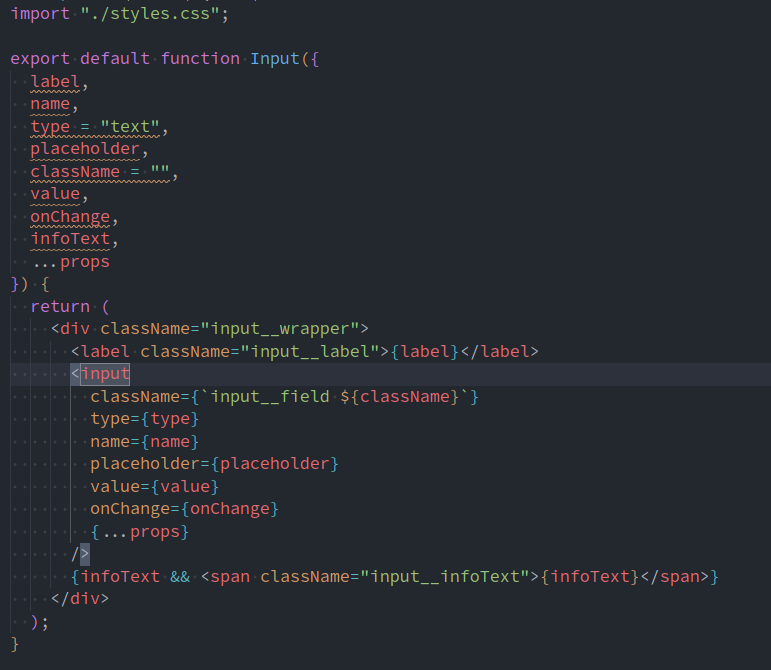
**Figura 14** — Utilização do className.

****

**Fonte:** Autoria Própria

Por fim, a estrutura de um componente React é uma função, onde o JSX é retornado dessa função, e no corpo dela fica toda a lógica que permite com que esse componente funcione da forma que é esperada.

**Figura 15** — Estrutura de um componente em React

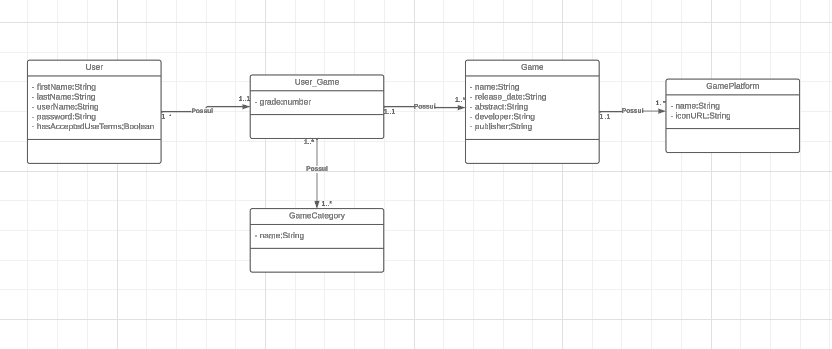
****

**Fonte:** Autoria Própria

## 2.3 DESENVOLVENDO A API

O desenvolvimento da API foi iniciada através da modelagem das entidades que compõem banco de dados, em seguida foi realizada a configuração do banco de dados construído em Node.js com o Express, e por fim, o desenvolvimento dos CRUDs referente a cada uma das entidades. A elaboração das entidades foi realizada por meio do diagrama de classes, um diagrama estrutural utilizado na modelagem de software. Sua finalidade é representar a estrutura das entidades e suas inter-relações na aplicação. Isso facilita a criação das tabelas no banco, já que o diagrama de classes mostra como as tabelas irão se relacionar, bastando apenas “traduzir” o que foi feito na criação das tabelas.

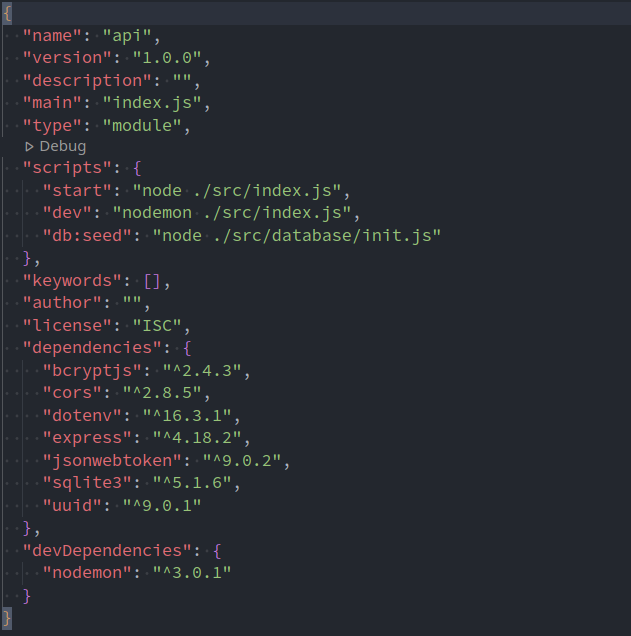
**Figura 16** — Diagrama de classes



**Fonte:** Autoria Própria

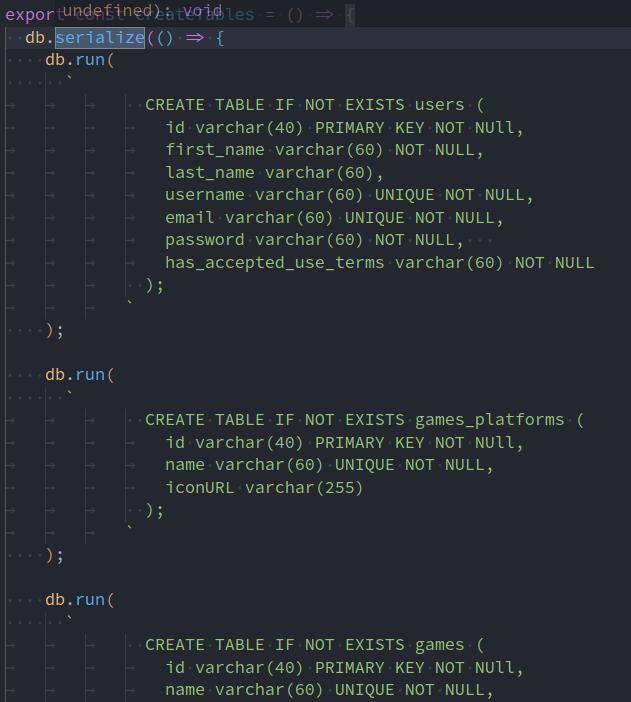
Em seguida, deu-se início ao desenvolvimento da API com a criação do projeto em Node.js. Foi realizado o desenvolvimento dos arquivos para a estruturação das tabelas no banco de dados, junto com o código JavaScript contendo consultas em SQL. Estas consultas têm como objetivo preencher previamente as tabelas com dados que são essenciais para o funcionamento das regras de negócio da API. Além disso, um script no package.json foi desenvolvido para preencher as tabelas isoladamente. Isso é crucial, pois impede a necessidade de executar consultas toda vez que a aplicação é reiniciada durante o desenvolvimento, proporcionando uma execução mais eficiente do projeto.

**Figura 17:** Arquivo de configuração do projeto



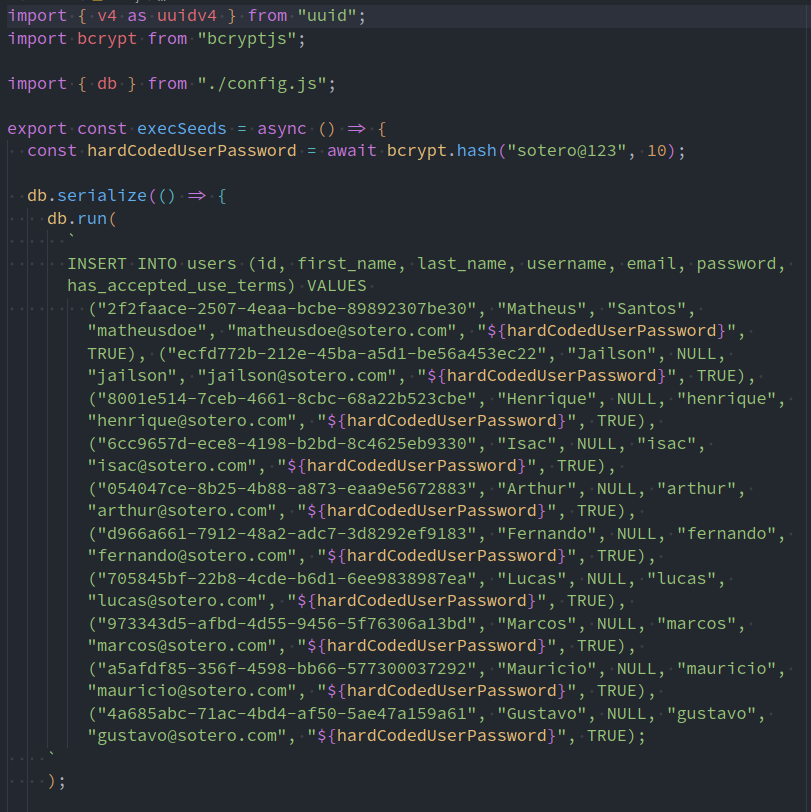
**Fonte:** Autoria Própria

**Figura 18** — Consultas para criação das tabelas



**Fonte:** Autoria Própria

**Figura 19:** Consultas para pre-popular tabelas

****

Fonte: Autoria Própria

A partir desse momento e após a configuração do banco de dados, todos os CRUDs necessários para o funcionamento das regras de negócio do projeto foram realizados. CRUD (Create, Read, Use, e Delete) são as operações básicas que regem a manipulação de um dado em uma API. Além disso, foram desenvolvidos os DAOs (Data Access Object), que representam um padrão de design utilizado para acessar o banco de dados, abstraindo a execução de consultas por meio de métodos incorporados em um objeto. Dessa forma, conclui-se a parte essencial do desenvolvimento da API relacionada à camada de dados, restando apenas a implementação das regras de negócios e da autenticação na API.

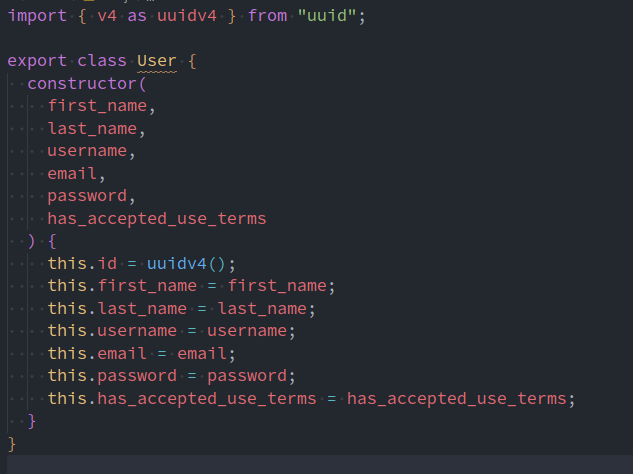
**Figura 19** — DAO da entidade Usuário.



**Fonte:** Autoria Própria

Por fim, o padrão de arquitetura utilizado no desenvolvimento da API foi o MVC (*Model View Controller*). Esse modelo de arquitetura de projeto é muito utilizado no mercado para desenvolver APIs. O padrão MVC envolve a concepção de distribuir as responsabilidades em uma API por meio de diversas camadas, cada uma encarregada de gerenciar um tipo específico de atividade na aplicação, sendo subdividida da seguinte forma: A camada *Model* é responsável por gerenciar o acesso aos dados, processamento e manipulação da lógica de negócios. O *Controller* é responsável por gerenciar a interação entre a camada model e a view da API, processando as solicitações do usuário e retornando as respostas para cada requisição. A view é responsável por apresentar a interface do usuário, ela está encarregada de gerar a saída da aplicação, bem como por receber a entrada do usuário.

**Figura 20** — Exemplo de Model.



**Fonte:** Autoria Própria

**Figura 21** — Exemplo de Controller.



**Fonte:** Autoria Própria

## 2.3.1 AUTENTICAÇÃO

A implementação da autenticação no projeto foi feita utilizando o *JSON Web Token*, que é uma das bibliotecas muito utilizadas na implementação de autenticação de projetos. O *JSON Web Token* (JWT) é uma biblioteca utilizada para gerar um token. Este token é utilizado pelo cliente nas requisições para validar se o usuário está de fato autenticado e autorizado a consumir determinados *endpoints* da API. Essa verificação é feita com o JWT através de um *middleware*, que intercepta as requisições que vem do cliente e verifica se o token enviado no cabeçalho da requisição (token gerado pelo JWT) é válido. Além disso, o JWT possibilita a definição de um período de validade para o token, tornando-o inválido após o limite de tempo estabelecido.

**Figura 22** — Middleware que verifica o token nas requisições



**Fonte:** Autoria Própria

## 2.3.2 Bcrypt

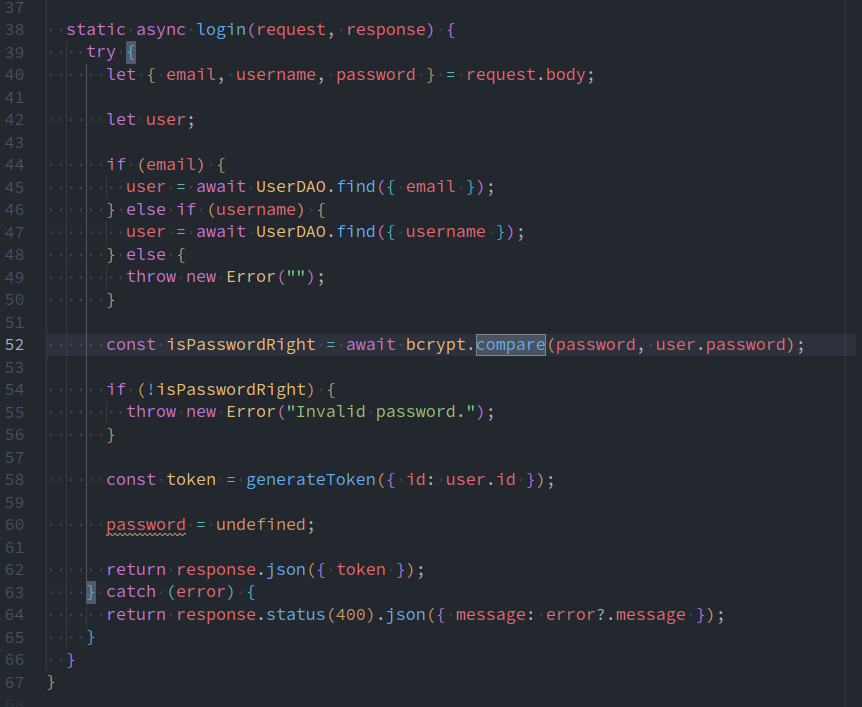
O Bcrypt é uma biblioteca utilizada no projeto para poder criptografar as senhas dos usuários antes de salvá-las no banco de dados. Uma prática recomendada em qualquer software que inclui funcionalidades de autenticação é garantir que as senhas dos usuários sejam devidamente protegidas antes de serem armazenadas no banco de dados, preferencialmente por meio de técnicas de mascaramento. Nesse contexto, o Bcrypt criptografa a senha do usuário transformando-o em um *hash*, que é uma sequência aleatória de caracteres. O Bcrypt também descriptografa a senha quando o usuário tenta entrar na aplicação, para verificar se a senha digitada pelo usuário é válida.

**Figura 23:** Utilização do Bcrypt no cadastro do usuário para criptografar a senha.



**Fonte:** Autoria Própria

**Figura 23:** Utilização do Bcrypt no login do usuário para validar a senha,

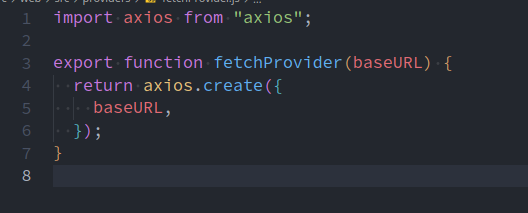


**Fonte:** Autoria Própria

## 2.4 INTEGRANDO A INTERFACE COM A API

Após finalizar o desenvolvimento da API, foi realizada a integração com a interface do projeto. Nessa etapa, a biblioteca Axios foi utilizada para realizar as requisições HTTP do cliente para a API. O Axios simplifica a elaboração de solicitações, eliminando a necessidade de configurar detalhes como cabeçalhos para especificar o tipo de conteúdo. Ele abstrai certos detalhes, que precisam ser especificados ao usar a Fetch API, uma ferramenta nativa do JavaScript. Além disso, a escrita das requisições é mais limpa já que não há a necessidade de escrever a quantidade de código que seria necessário com a utilização da Fetch API (ou do próprio Ajax).

**Figura 23** — Criando instância do axios para fazer as requisições para a API.



**Fonte:** Autoria Própria

**Figura 23** — Utilizando a instância do axios para criar as requisições.



**Fonte:** Autoria Própria

**3 CONSIDERAÇÕES FINAIS**

O desenvolvimento e conclusão do projeto de uma API REST para alimentar um catálogo de jogos, com frontend construído em React, representa um marco significativo. Ao longo desse processo, foram enfrentados desafios que contribuíram para o amadurecimento técnico e a entrega de um sistema robusto e funcional.

A implementação da API REST serviu como principal parte do catálogo de jogos, demonstrou eficiência na manipulação e entrega de dados. A biblioteca do React proporcionou uma experiência de usuário dinâmica e responsiva atendendo os critérios de usabilidade e heurística de Nielsen.

A integração da API e o frontend, utilizando o React, permitiu a criação de uma aplicação coesa e de fácil manutenção. A estrutura de componentes do React permitiu o desenvolvimento de códigos reutilizáveis, promovendo a escalabilidade e eficiência no desenvolvimento.

Ao longo do projeto, a atenção foi dada não apenas à funcionalidade, mas também à segurança e autenticação de usuário, garantindo que a API esteja protegida contra as básicas e possíveis vulnerabilidades e que as informações do catálogo de jogos sejam manipuladas de maneira segura.

Portanto, ao concluir o projeto é gratificante observar o sucesso técnico e a aplicação prática dos conceitos aprendidos ao longo do desenvolvimento do trabalho.

# REFERÊNCIAS

AKITA F. Entendendo Como Containers Funcionam. **Akitaonrails**. 02 Mar. 2023. Disponível em:. <https://www.akitaonrails.com/2023/03/02/akitando-139-entendendo-como-containers-funcionam>. Acesso em: 19 out. 2023.

BERNSTEIN, David. Containers and Cloud: from lxc to docker to kubernetes. **Ieee Cloud Computing**, [S.L.], v. 1, n. 3, p. 81-84, set. 2014. Institute of Electrical and Electronics Engineers (IEEE). <http://dx.doi.org/10.1109/mcc.2014.51>.

Node.js. (2022). Node.js Documentation. Node.js. Disponível em: <https://nodejs.org/en/docs/>. Acesso em: 11 de dezembro de 2023.

React. (2022). React - A JavaScript library for building user interfaces. React Documentation. Disponível em: <https://reactjs.org/docs/getting-started.html>. Acesso em: 11 de dezembro de 2023.