UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS TOLEDO
COORDENAÇÃO DO CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JACKSON VINICIUS FARIA DE OLIVEIRA

BIBLIOTECA COLABORATIVA BOOKER

TRABALHO DE CONCLUSÃO DE CURSO

TOLEDO
2021

JACKSON VINICIUS FARIA DE OLIVEIRA

BIBLIOTECA COLABORATIVA BOOKER

Trabalho de Conclusão de Curso de Graduação, apresentado ao Curso Superior de Tecnologia em Sistemas para Internet, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Roberto M. Scheffel

TOLEDO

2021

JACKSON VINICIUS FARIA DE OLIVEIRA

BIBLIOTECA COLABORATIVA BOOKER

Trabalho de Conclusão de Curso de Graduação, apresentado ao Curso Superior de Tecnologia em Sistemas para Internet, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Data de aprovação: 25 de agosto de 2021

Eduardo Pezutti Beletato Dos Santos
Mestre em Ciências da Computação e Matemática Computacional
Universidade Tecnológica Federal do Paraná

Marcelo Alexandre Da Cruz Ismael
Mestre em Informática
Universidade Tecnológica Federal do Paraná

Roberto Milton Scheffel
Doutor em Ciência da Computação
Universidade Tecnológica Federal do Paraná

TOLEDO

2021

AGRADECIMENTOS

Diversas pessoas estiveram envolvidas na realização desse trabalho, de forma direta ou indireta. Um grupo de pessoas essenciais, as quais tiveram paciência de me ensinar desde o nível em que eu não conhecia nada de programação, até o fim do meu processo de formação, são meus professores, que contribuíram cada um de sua forma para agregar o conhecimento necessário na realização desse trabalho. Dentre eles, destaco meu orientador, o Professor Roberto M. Scheffel e meu co-orientador na matéria de TCC 1, Professor Wesley K. G. Assunção. Ambos foram vitais para que eu pudesse notar diversos detalhes que fazem a diferença na construção de um sistema como o Booker.

Outro grupo que devo meus agradecimentos, são meus familiares, que permitiram que por uma grande fração do período em que estive matriculado nesse curso, pudesse me dedicar exclusivamente a ele e me deram todo o suporte necessário para alcançar o ponto em que estou agora. Também devo agradecimentos à minha companheira que demonstrou empatia nos períodos em que não me restou muito tempo disponível na minha organização de rotina diária, bem como em diversos momentos especiais.

RESUMO

Oliveira, Jackson V. F.. BIBLIOTECA COLABORATIVA BOOKER. 2021. 43f. Trabalho de Conclusão de Curso – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Toledo, 2021.

Pesquisas feitas pelo Instituto Pro Livro, em 2016, atestaram que os índices de pessoas leitoras no Brasil estão baixos, 56%. Isso se constata ao comparar com os dados da pesquisa feita pela FEDERACIÓN DE GREMIOS DE EDITORES DE ESPAÑA (2018), da Espanha, em que o número apresentado de leitores é de 67,2%. Esta proposta de trabalho de conclusão de curso, tem como objetivo o desenvolvimento de um sistema online para incentivar o empréstimo de livros entre seus usuários. Cada pessoa cadastrará os próprios livros, físicos, não e-books, assim, estes ficam disponíveis para outros usuários solicitarem o empréstimo. Esse software tem como objetivo aumentar, na população brasileira, o número de pessoas que efetivamente leem com frequência. Será um sistema que funcionará tanto em plataformas móveis, como o Android, quanto em navegadores web. O sistema terá arquitetura baseada em microserviços.

Palavras chave: Biblioteca colaborativa. Microserviços. Aplicação. Leitura.

ABSTRACT

Oliveira, Jackson V. F.. COLLABORATIVE LIBRARY BOOKER. 2021. 43f. Final Paper – Internet Systems Technology Course, Federal Technological University of Paraná. Toledo, 2021.

Research conducted by the Pro Livro Institute in 2016, testified that the indicesof readers in Brazil are low, 56%. This is verified when comparing with the data of the research carried out by the FEDERACIÓN DE GREMIOS DE EDITORES DE ESPAÑA (2018), Spain, in which the number of readers is 67.2%. This proposal of work has as goal the development of an online system to encourage the loan of books among its users. Each person will register the books themselves so they become available for other users to request for the loan. This software aims to increase, in the Brazilian population, the number of people who actually read often. It will be a system that will work on both mobile platforms such as Android and web browsers. The system will have microservice-based architecture.

Keywords: Colaborative library. Microservices. Application. Reading.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

API	APPLICATION PROGRAMMING INTERFACE
CEP	CÓDIGO DE ENDEREÇAMENTO POSTAL
CSS	CASCADING STYLE SHEETS
DOM	DOCUMENT OBJECT MODEL
HTML	HYPERTEXT MARKUP LANGUAGE
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
ISBN	INTERNATIONAL STANDARD BOOK NUMBER
JSON	JAVASCRIPT OBJECT NOTATION
JSX	JAVASCRIPT XML
JWT	*JSON WEB TOKEN*
ORM	OBJECT RELATIONAL MAPPER
REST	REPRESENTATIONAL STATE TRANSFER
SGBD	SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS
SQL	STRUCTURED QUERY LANGUAGE
URL	UNIFORM RESOURCE LOCATOR
UUID	UNIVERSALLY UNIQUE IDENTIFIER
W3C	WORLD WIDE WEB CONSORTIUM
WHATWG	WEB HYPERTEXT APPLICATION TECHNOLOGY WORKING GROUP
XML	EXTENSIBLE MARKUP LANGUAGE

SUMÁRIO

1 INTRODUÇÃO

1.1 TEMA

A leitura está presente em diversas atividades humanas desde o surgimento da escrita, com os sumérios utilizando a escrita cuneiforme, por volta de 3000 a.C. COOPER (2004). Com o passar do tempo, grande parte das teorias, soluções, leis e registros, de qualquer profissão ou até mesmo registros relacionados a lazer e entretenimento passaram a ser documentados. Considerando que, hipoteticamente, uma determinada pessoa tenha a necessidade de uma determinada informação, seja ela científica ou não, existe a probabilidade de em algum lugar essa informação ser encontrada escrita. Por isso, saber ler essa informação poderia seria uma vantagem perante quem não sabe, ou, promoveria satisfação pessoal por ter adquirido um conhecimento que antes não estava presente.

Uma correlação pode ser estabelecida, através de estudos já feitos, entre o quanto uma população lê e o impacto positivo disso na vida dela. Indicando os benefícios desta prática no que tange a inteligência emocional, capacidades de se ter empatia pelo próximo e até mesmo, segundo VEMURI e MORMINO (2013), em leituras extremamente estimulantes para nossa mente, é possível melhorar a memória e evitar doenças como a Demência e Alzheimer.

A partir do pressuposto de que esses benefícios podem ser obtidos através de exercícios cognitivos, como a leitura, surge a ideia que permeia esse trabalho de conclusão de curso. Como seria possível incentivar mais o hábito de consumo de livros, a fim de melhorar a qualidade de vida e trazer os benefícios mentais da leitura? Diante desta problemática, uma alternativa para amenizar tal problema seria facilitar o acesso aos materiais de leitura para todos. Muitas pessoas não têm condições de pagar R$ 300,00 ou mais em livros técnicos ou valores próximos a isso em outros gêneros. Diante deste ponto, o sistema desenvolvido durante este trabalho tem a intenção de apresentar-se como um facilitador, já que dado o devido tempo e tendo boa adesão, o acervo de livros acessíveis por meio de empréstimos seria, em média, desde centenas a dezenas de milhares de livros em uma determinada região.

O sistema desenvolvido nesse trabalho age sobre como as pessoas terão

acesso aos livros, permitindo empréstimos entre os próprios usuários dentro da plataforma de forma descentralizada. Usuários cadastrarão livros físicos que estão sob sua posse e, a partir desse momento, esses livros estarão disponíveis para outras pessoas solicitarem seu empréstimo.

1.2 PROBLEMA

Pela massiva transformação de obras literárias em conteúdo audiovisual, como séries e filmes, existe a perda do interesse da população pela leitura como entretenimento. As pessoas perdem esse contato pois é muito mais simples assistir ao filme Senhor dos Anéis doque ler a obra completa. Uma situação resultante disso é que será possível ter apenas uma experiência passiva ao visualizar um filme, pois já foi tudo previamente imaginado pelo diretor e equipe de produção. Enquanto que na leitura existe todo um trabalho mental de imaginação do que se está lendo.

Motivos para optar pela leitura existem em grande quantidade, mas como fazer alguém decidir ir por esse caminho um pouco mais trabalhoso, porém, muito recompensador? Uma das metodologias possíveis para atingir essa questão é facilitar o acesso.

Uma suposta situação de definição de qual opção será escolhida por uma pessoa que não tem preferência entre livros ou filmes é a seguinte: Há um filme sobre a história “x” e um livro sobre o mesmo “x”, sendo o livro a obra original. Em um fluxo de decisão quais são os primeiros direcionamentos que resultarão na opção escolhida? Várias pessoas podem pensar que o filme já está em um canal de *streaming* e vão optar por tal meio. Outras podem buscar o livro e se deparar com um prazo de 10 dias para a entrega deste, aliado à uma cobrança de R$ 100,00 com o preço do material somado ao frete e, com isso, desistir dessa escolha. Então, a partir dessas situações é factível que dentre vários pontos, um que pesa muito na decisão é a disponibilidade do material a horas ou até mesmo minutos de sair do estado de requisição até estar em mãos da parte interessada.

Tendo isso em vista, surgem desafios de como deter uma grande variedade de obras tanto quantitativamente quanto qualitativamente, de forma minimamente custosa possível. Existem as bibliotecas públicas, mas por uma questão de espaço, é praticamente impossível abordar uma fração significativa de materiais de leitura.

Portanto, os pontos mais problemáticos que serão abordados pelo Booker são: A limitação de espaço para os livros e a facilidade de acesso ao material, considerando o custo monetário ao adquirir o livro e o custo de tempo de logística para o produto estar nas mãos da parte interessada.

1.3 JUSTIFICATIVA

Quase tudo que aborda a leitura acaba sendo um assunto muito importante, ainda mais em um país emergente como o Brasil, que ainda não está entre os países do topo no ranking de educação. O que significa que há como melhorar nessa área. A leitura é um assunto importante porque, de fato, irá contribuir para essa melhora na educação. Tendo em vista que uma população acostumada a ler, consequentemente demonstra menos resistência ao precisar consumir qualquer material didático mais complexo para aprendizado. É possível inferir isso pois ela já está habituada a lidar com grandes quantidades de texto. Se considerar todos os pontos de melhora cognitiva que a leitura proporciona, anteriormente citados, além da eventual contribuição na educação, atribui muito significado em abordar essa área.

Outra justificativa que embasa a escolha do tema proposto, é a pesquisa estatística encomendada pelo INSTITUTO PRO LIVRO (2016), através do Instituto Brasileiro de Geografia e Estatística. Esta pesquisa classificou a população brasileira em leitor, ou seja, alguém que leu ao menos um livro ou partes de um livro no período de 3 meses, ou não leitor, que não cumpriu esse requisito. Essa pesquisa, resumida pela figura 1, mostrou como resultado um índice de 56% de leitores. Comparando esse índice com a Espanha, como mostra a figura 2, em pesquisa encomendada pela FEDERACIÓN DE GREMIOS DE EDITORES DE ESPAÑA (2018), que é de 67,2% para pessoas com 14 anos ou mais, demonstra ser um ponto a ser melhorado. Ainda na pesquisa da Espanha, está descrito que, ao considerar outros tipos de leituras como periódicos, revistas, quadrinhos e textos da internet como blogs e redes socias, o índice de leitores chega a 96,1% para as pessoas com 14 anos ou mais.

Figura 1 - Índice de leitores no Brasil em 2015

Fonte: Instituto Pro Livro

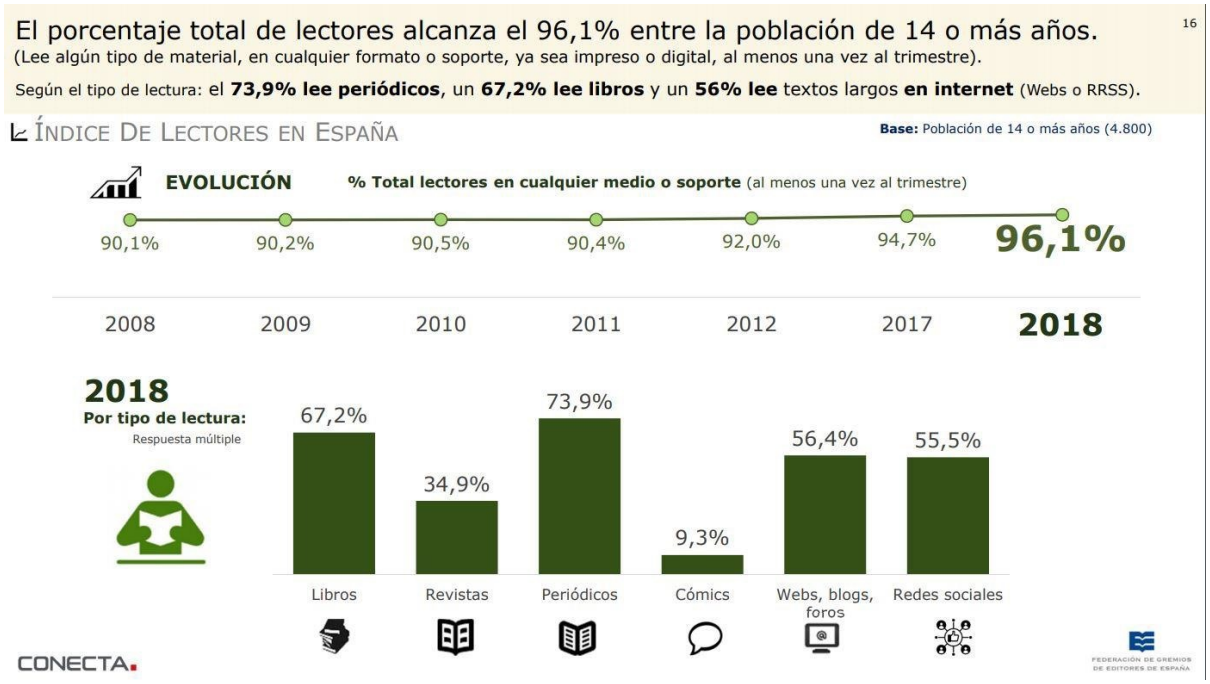

Figura 2 - Índices de leitores na Espanha em 2018

Fonte: Federacion de Gremios de Editores de España

A importância do sistema desenvolvido ao longo desse trabalho se dá na maior facilidade de acesso aos livros buscando atingir, além das pessoas que já leem, a geração atual e as futuras em um espaço de conforto delas, a internet. Dessa forma, o Booker pode ser um possível facilitador à leitura, contribuindo no nicho de empréstimos de livros físicos. Isso é relativamente novo, já que quase a totalidade de modelos de negócio envolvendo livros, que não seja de venda de material, trata de e-books, não materiais em papel. Um exemplo de software que atua hoje com e-books

é o *ReadFy.com*, entretanto, ele não utiliza o modelo de empréstimos, já que não há incentivo para isso ao tratar de e-books pela própria natureza digital que eles têm. Por permitir o empréstimo de livros físicos, leitores que não gostam da ideia de consumir esses materiais por tela poderão se sentir inclinados a aceitar a premissa do empréstimo.

1.4 OBJETIVOS

Nesta seção serão apresentados o objetivo geral e objetivos específicos do presente trabalho.

1.4.1 Geral

Desenvolver um sistema online para permitir o livre empréstimo de livros entre os usuários, com o fim de incentivar a leitura pela maior facilidade de acesso às obras.

1.4.2 Específicos

Os objetivos aqui apresentados irão abordar os pontos que dificultam o acesso aos materiais de leitura e como estes serão resolvidos a partir do sistema Booker, sendo essas as áreas em que o sistema pretende agir.

- Definir o modelo de arquitetura e tecnologias do sistema
- Projetar a arquitetura do sistema a partir dos diagramas de casos de uso, classe e sequência
- Implementar
- Testar e avaliar o sistema

2 EMBASAMENTO TEÓRICO

Esse trabalho de conclusão de curso é classificado como tecnológico, não cabendo em classificações de pesquisa, tais como a exploratória, descritiva ou explicativa.

2.1 ARQUITETURA

Todo sistema computacional, de forma resumida, tem duas opções de arquiteturas a serem implementadas. Arquitetura monolítica (RICHARDSON, 2019a) e não-monolítica (RICHARDSON, 2019b), sendo que no universo desta última, estão incluídos sistemas distribuídos como os microserviços. No caso desse sistema, será feita a implementação inicial na arquitetura monolítica. Após isso, será feita a migração para a arquitetura de microserviços.

2.1.1 Monolítica

É a arquitetura comumente usada nos mais diversos softwares. Segundo definição do CODING THE ARCHITECTURE (2014) “Se você tem um modulo monolítico, então todo o código do sistema está em uma única base de código que é compilada e produz um único artefato.”. Nela, o servidor inteiro existe em apenas um ambiente computacional, seja um cluster ou um único computador. A aplicação, conforme cresce, fica extremamente difícil de manter, já que existe muito código para ser verificado até encontrar o problema. Outro problema muito comum é com relação ao forte acoplamento das classes. Esse forte acoplamento faz com que uma mudança no funcionamento de uma classe gere, como consequência, a necessidade de corrigir outras classes dependentes dela. Outro ponto negativo do forte acoplamento ocorre quando um trecho do sistema tem mal funcionamento devido a uma alteração. Com isso, toda a aplicação deixa de funcionar até a retificação deste problema.

Existem outros pontos negativos, porém, ainda existem vantagens em relação a arquiteturas não-monolíticas. Evitar duplicidade de código é muito mais simples em aplicações monolíticas e o deploy precisa ser feito em apenas um ponto.

2.1.2 Microserviço

Será a arquitetura final empregada nesse trabalho. Na definição de FOWLER e LEWIS (2014), “O estilo arquitetural de microserviços, é uma abordagem para o desenvolvimento de uma única aplicação como uma suíte de pequenos serviços, cada qual rodando em seu próprio processo e se comunicando usando mecanismos leves, como frequentemente usado, um recurso API sobre HTTP.” Ela trabalha com a virtualização de ambientes, fazendo com que o sistema seja composto de diversas pequenas aplicações trabalhando em conjunto, com alta coesão e baixo acoplamento. Não é necessariamente nova, mas ganhou há pouco tempo o termo de definição “microserviço”, em 2011.

Essa arquitetura é muito utilizada em sistemas distribuídos para promover o uso de serviços com baixo acoplamento e alta coesão. Os benefícios apresentados por NEWMAN (2015a) são muitos, desde heterogeneidade de tecnologias, permitindo em uma mesma aplicação dividir-se em várias linguagens no lado do servidor e/ou o lado do cliente, dessa forma uma equipe pode trabalhar com a linguagem “x” em uma parte da aplicação e a outra equipe trabalhar com a linguagem “y” nos serviços de sua responsabilidade, bastando apenas ter uma conversação padronizada entre as partes.

Além disso, existe uma base de código menor para cada microserviço, o que facilita a compreensão do software e sua manutenção. Como desvantagem, tem alta duplicação de código e por ter que fazer a comunicação entre os serviços usando HTTP ou “*Hypertext Transfer Protocol*”, gera uma latência mais alta no sistema que exige um cuidado maior na codificação visando o desempenho da comunicação.

O uso desse estilo de arquitetura se dá, também, porque o Booker já está preparado para o *cross-platform*, já que o uso de microserviços possibilita o uso da implementação da estrutura de *Backend For Frontend* ou *BFF* proposto por NEWMAN (2015b). Essa estrutura consiste em um microserviço que fará as primeiras tratativas nas comunicações com a interface. Cada tipo de interface, ou *frontend*, terá o seu respectivo *BFF.* Então, o *BFF* do *frontend web* captura suas requisições, trata seus dados e envia para o microserviço *gateway*. Esse *gateway* tem a função de endereçar as requisições para os microserviços envolvidos em processar os dados recebidos e gerar um retorno, de forma que terá um papel voltado a orquestrar a comunicação entre cada *frontend* e os microserviços. O *frontend mobile* também precisa de um *BFF*

para tratar seus dados. Dessa forma, com a comunicação em HTTP enviando dados em JSON, se houver um *BFF* para cada *frontend*, o tipo do *frontend* se torna transparente para as camadas de regra de negócio do *backend*. Um modelo para essa arquitetura está representado na figura 3.

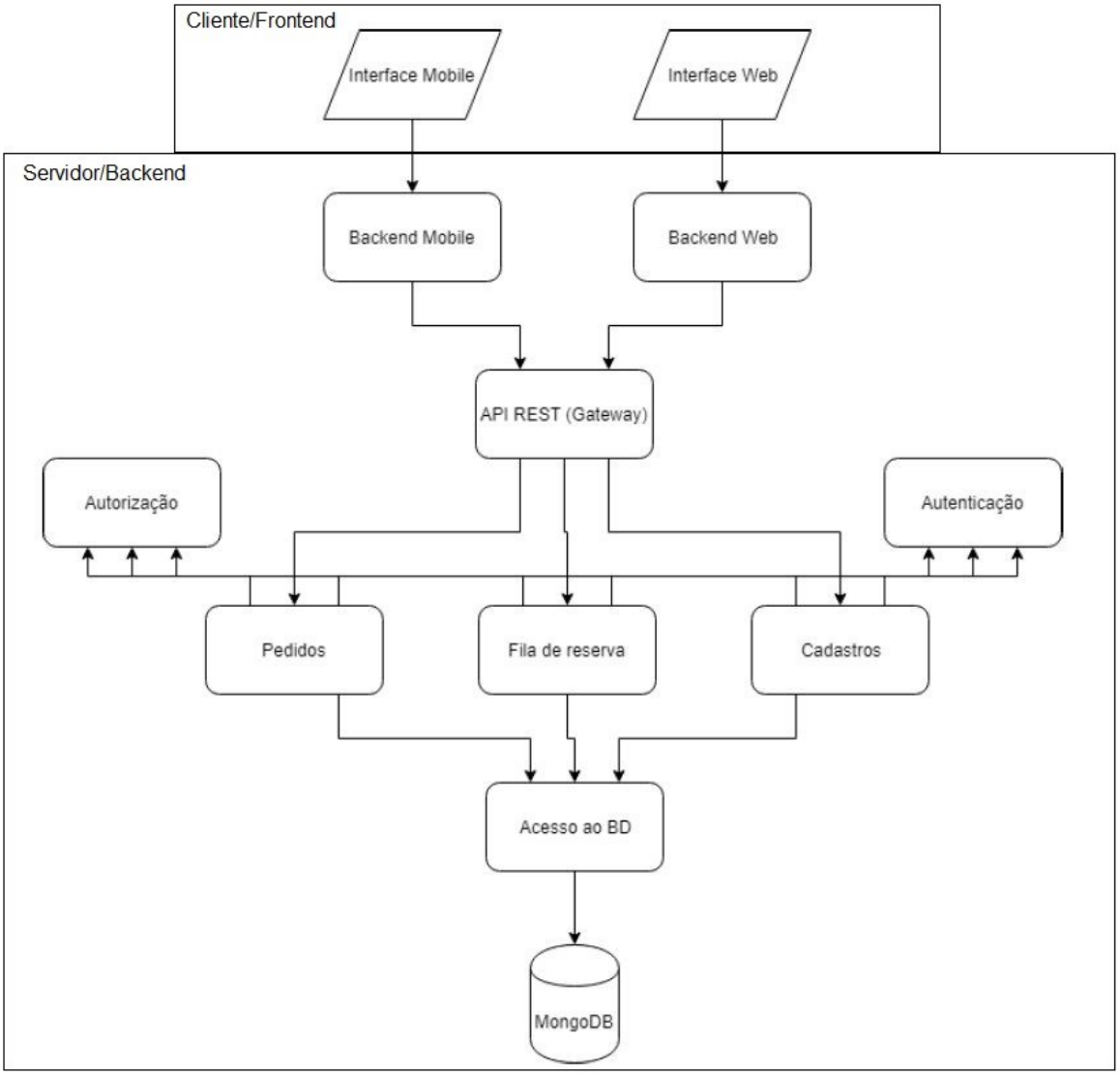

Figura 3 – Exemplo de arquitetura de microserviços com BFF
Fonte: Próprio autor

2.2 TECNOLOGIAS

Os conceitos teóricos utilizados nesse trabalho situam-se no conjunto de desenvolvimento web e microserviços. As tecnologias discutidas são: Node.js, React, Typescript, HTML5, CSS3, PostgreSQL, NoSQL, Docker e Kubernetes.

2.2.1 Node.js

Node.js é, segundo os próprios criadores, um ambiente de execução desenvolvido para funcionar com o motor Chrome V8 de Javascript. Sua funcionalidade faz com que seja possível usar Javascript no servidor da aplicação, da mesma forma é usado, por exemplo, C, Java e Python, mas, com foco no assincronismo orientado à eventos.

Node.js é o framework Javascript que é usado na maior parte do *server side* da aplicação. O Node tem muitas bibliotecas mantidas por sua comunidade para a integração com as tecnologias utilizadas nesse sistema, como o Docker, NoSQL e o PostgreSQL. Tendo isso em vista, seu uso é justificado nesse sistema dada a maior facilidade de encontrar conteúdo para lidar com a integração das tecnologias. Os conhecimentos teóricos do Node.js se deram através de leitura da documentação Javascript, do EXPRESS (2021), que é o framework usado no ambiente Node.js, e ainda, de eventos de programação gratuitos e cursos da ROCKETSEAT.

Tem como alternativa o Java. Essa linguagem fornece extenso suporte para o desenvolvimento de um sistema com arquitetura de microserviços utilizando JAX-RS. Spring também permite várias abordagens aos microserviços, porém, no futuro há planos para a integração com uma interface *mobile*, sendo assim, desejável a utilização de facilitadores para o desenvolvimento, como a biblioteca React Native ou o *framework* Angular, sendo os dois dependentes do Node.js

2.2.2 React.js

Criada pelo Facebook Inc., essa coleção de bibliotecas escrita em Javascript facilita muito o desenvolvimento web, tendo em vista o modelo de programação funcional e reativo. Com esse modelo foi fácil construir estruturas HTML e até mesmo CSS com condições lógicas.

No caso do CSS no React, é possível evitar muito código javascript se for feito o uso de **Styled Components.** Eles permitem o condicionamento interno de um componente em tela baseado no estado em que aquele componente se encontra. Dessa forma, quando um atributo interno no componente React recebe uma mudança de estado, é possível disparar uma mudança visual na aplicação de forma reativa, facilitando muito a parte de User Experience pois que será exigido muito menos código para ter um resultado aceitável.

```
28  export const Content = styled.div<CreateAccountProps>`
29    width: 30vw;
30    height: 80vh;
31    min-height: ${({ create }) => (create ? "55rem" : "30rem")};
32    backdrop-filter: blur(10px);
33
34    display: flex;
35    flex-direction: column;
36    align-items: center;
37    justify-content: center;
38    transition: 0.1s all linear;
39
40    ${({ create }) =>
41      create &&
42      css`
43        width: 50vw;
44        height: 90vh;
45        transition: 0.1s all linear;
46      `}
```

Figura 4 – CSS condicional usando styled components
Fonte: Próprio autor

Na figura 4, através da interface *CreateAccountProps,* é especificado que este componente irá receber um parâmetro booleano com o nome "*create*". Neste caso, estamos nos referindo à tela de **Login** do sistema. Então, o *create* se estiver com valor verdadeiro, indica que o formulário de login terá configuração para a criação de usuário, definindo seu comportamento para tal. Com os **Styled Components**, é possível condicionar o CSS de duas formas: Na linha 31, caso o "*create*" seja

verdadeiro, a altura mínima do componente “*Content*” será de 55rem, se não, será de 30 rem. Já na linha 40, é feito um condicionamento que gera um bloco de código de estilo. Caso o “*create*” seja verdadeiro, então será retornado para o escopo principal do “*Content*” as propriedades contidas em “*css* ` ` ”. Pela natureza do CSS de ser uma folha de estilos em cascata, o que vier depois, no mesmo escopo, reescreverá as propriedades que vem antes. Portanto, a nova largura e altura sobrescreverão a largura e altura definidas nas linhas 29 e 30. Essa abordagem foi usada em diversas partes da estilização do sistema para reduzir a base de código Typescript.

Ainda nas facilidades da programação funcional, é possível gerar alterações estruturais no HTML com condições lógicas como ternários e funções *lambda*, essa última, no Javascript, é chamada de *arrow function*. Assim, graças a essa facilidade de retornar um trecho de código se uma variável de estado contiver um dado que satisfaça uma condição ou não, é possível, novamente, evitar muito código Javascript ou até a criação de novos documentos HTML. Para permitir essas condições lógicas dentro do HTML, o React faz o uso de JSX. O JSX é uma extensão de sintaxe para o Javascript que parece uma linguagem de *template* como o HTML ou o XML, entretanto, as marcações do JSX criam nós React que são transmitidos ao DOM do navegador de forma mapeada. Dessa forma, é permitido por essa linguagem a inserção de código Javascript de forma limitada entre as marcações e a inserção de atributos dinâmicos bem como chamadas de funções que geram uma nova renderização do componente JSX caso seu estado tenha mudado.

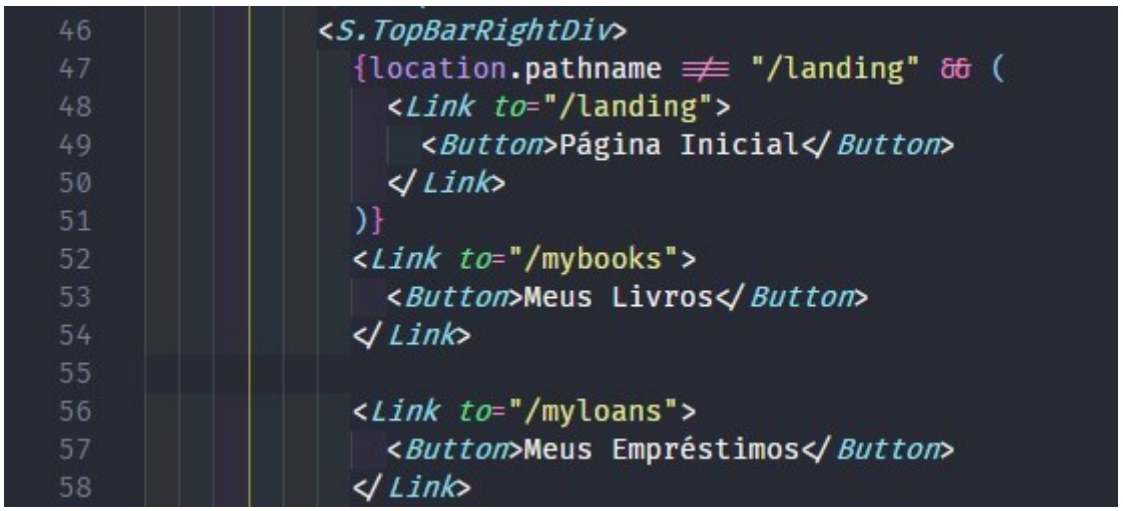

Figura 5 – Renderização JSX condicional

Fonte: Próprio autor

Na Figura 5 é exemplificada a renderização condicional que o JSX permite. O trecho abordado é o dos *links* de navegação entre páginas do sistema. A condição se dá definindo se um botão para retornar à página inicial será renderizado ou não. Caso o endereço *URL* contenha imediatamente após o domínio, "*/landing*", significa que o usuário está na página inicial, não havendo a necessidade de exibir o botão para navegar para esta página inicial.

O estudo do React se baseou na documentação da biblioteca e, da mesma forma que o node.js, em eventos de construção de aplicações e cursos da ROCKETSEAT.

Tem como alternativa o Angular, o Vue e o Svelte, porém, frameworks acabam gerando um acoplamento um pouco maior na base do código do sistema, o que não é desejável. Portanto a escolha ficou com o uso de uma biblioteca ao invés de um framework para o desenvolvimento da interface.

2.2.3 HTML5

Praticamente obrigatório para qualquer aplicação que funcione em navegadores, o HTML ou *Hypertext Markup Language* (Linguagem de marcação de hipertexto), é amplamente utilizado no front-end para definir os elementos de uma página. Mantido pela World Wide Web Consortium (W3C) e Web Hypertext Application Technology Working Group (WHATWG), essa linguagem de hipertexto foi extensivamente estudada de forma aplicada durante esse curso de graduação em Sistemas para Internet. Como material de consulta foi usada principalmente a W3SCHOOLS.

2.2.4 CSS3

CSS ou *Cascading Style Sheets* (Folha de Estilo em Cascatas), criado e mantido pela W3C, é utilizada para a estilização de páginas na internet. Seu uso se dá em todos os locais onde o HTML é usado. Com isso, é possível melhorar a usabilidade e a apresentabilidade de seus elementos.

Como o CSS tem seu uso interligado ao HTML. O material de consulta usado é o W3schools. Além disso, são empregados padrões de usabilidade de interface na implementação dessa tecnologia, como o Design Centrado no Usuário de ABRAS,

MALONEY-KRICHMAR e PREECE (2004).

2.2.5 NoSQL

NoSQL é usado para tratar de consultas em bancos não-relacionais. Será utilizada em conjunto com o Sistema de Gerenciamento de Banco de Dados (ou SGBD) MongoDB. O NoSQL trabalha muito bem com grandes volumes de dados, de forma que é possível obter baixa latência e alto desempenho em comparação com bancos de dados relacionais. A baixa latência, em especial, é uma meta para qualquer software que faz uso de microserviços, pois, por existir uma comunicação interna na parte de servidor entre os serviços, se houver grande latência, toda a usabilidade do sistema ficará lenta devido as trocas de informações constantes e diversos eventos disparados por múltiplos usuários.

Portanto, como a alta responsividade da interface deve ser tratada com muito mais cuidado com esse tipo de arquitetura não-monolítica, o MongoDB se encaixa perfeitamente no projeto e será implementado no futuro para permitir o cadastro de livros baseados no retorno de APIs externas. Como referência, será utilizada a própria documentação do SGBD. Entretanto, atualmente, está sendo usado apenas o PostgreSQL.

Existem diversas alternativas ao MongoDB no universo de bancos de dados orientados à documentos, entre eles, os mais conhecidos são:

- Apache CouchDB: Criado pela Apache Software Foundation, tem licença proprietária e um modo de uso para avaliação. Algumas funcionalidades são: Leitura e escrita ACID, suporte nativo à BLOB (Binary Large Objects), interface gráfica no navegador para gerenciamento e operações de Map / Reduce em Views e Índices.
- Azure CosmosDB: Criado pela Microsoft, visa à o modelo de escalonamento horizontal dos dados através de uma base de dados distribuída. É proprietário sob a marca Azure. Algumas funcionalidades que outros bancos não costumam oferecer são: Integração nativa com chats, grafos sociais, separação dos dados por país de origem para garantir o cumprimento da legislação local e sistema de recuperação de

dados nativo.

- RavenDB: Criado pela Hibernating Rhinos, é semelhante ao MongoDB em vários aspectos. Tem licença proprietária e a licença comunitária com limitações. Tendo vantagens no tamanho de sua imagem Docker ser bem reduzido, bem como ter uma linguagem que busca que lembra muito o SQL, sendo o Raven Query Language (RQL). Oferece interface gráfica.

Entre os bancos de dados orientados à documentos, a escolha do MongoDB fica por conta da grande disponibilidade de materiais de treinamento, tanto em inglês quanto em português, já que sua comunidade é uma das maiores entre esse paradigma de banco de dados. Sendo assim, qualquer outro dos bancos listados acima, na sua versão de avaliação ou comunitária, também atenderia as necessidades do Booker.

2.2.6 PostgreSQL

A utilização de um banco relacional nos estágios iniciais do sistema se dará pela facilidade de encontrar material que relacione o uso de Typescript com o React e principalmente o node.js. Isso porque a tipagem forte favorece o relacionamento de entidades através de relacionamentos estabelecidos no código-fonte e deixa uma estrutura pronta para enviar ao PostgreSQL, exigindo mais tratamento apenas ao lidar com datas. Após essa fase inicial do sistema, o PostgreSQL será utilizado apenas nos cenários que demandem o uso de relações entre entidades do sistema.

2.2.7 Docker

O Docker será essencial para conseguir um maior desempenho de toda a estrutura de microserviços alocados cada um em seu ambiente independente. Na implementação sem o conceito de Conteinerização do Docker, é necessário gerenciar várias Máquinas Virtuais, cada uma simulando seu próprio Sistema Operacional. Isso consome muito poder computacional e, nesse projeto, dado os recursos disponíveis, cada porcentagem desse poder computacional desperdiçado custará muito no desempenho da aplicação.

Então, com o uso dessa ferramenta, será possível gerar vários ambientes completamente independentes, onde um serviço, se for parado por qualquer motivo que seja, não afetará os outros diretamente. Tudo estará funcionando em cima de um único *Kernel* evitando simular outros Sistemas Linux para cada serviço. Esses ambientes consumirão apenas o essencial para o microserviço conseguir desempenhar sua funcionalidade, o que terá um impacto muito positivo na responsividade do software e capacidade de gerenciar requisições dos usuários.

A base teórica para implementação do Docker veio de cursos do canal LINUXTIPS, tutoriais da REDHAT e a própria documentação criada pela Docker Inc.

2.2.8 Kubernetes

Com uso do Docker, surge a necessidade de gerenciar os seus Containers e é isso que a tecnologia Kubernetes faz. Permite configurar, automatizar e gerenciar as imagens Docker. Seu uso se justifica na facilidade que é oferecida se comparada a curva de aprendizado do uso dessa ferramenta, com a curva de aprendizado de gerenciamento manual dos Containers.

Tanto o Docker quanto o Kubernetes são relativamente recentes, mas já existe bastante conteúdo sobre a implementação de ambos em conjunto. O LINUXTIPS fornece um treinamento inicial gratuito no uso de tais ferramentas. De qualquer forma, essas duas tecnologias estão com muita documentação feita pela comunidade.

Em um primeiro momento, o gerenciamento de Containers Docker se dará através do Docker Desktop e após, será feita a migração para o Kubernetes.

2.2.9 Typescript

O uso do typescript no projeto se dá para uma maior qualidade de código e facilidade de construção de uma estrutura coesa. Ele define uma tipagem forte no Javascript, dessa forma, o uso de Interfaces e heranças é muito facilitado e traz diversas vantagens ao usar o React e node.js

Algumas delas são uma maior inteligência da IDE na verificação de problemas de código e permite o uso do Javascript modules sem precisar de algumas

dependências como o Babel.

Uma alternativa para o Typescript seria o flow. Mas, por este ser mais recente, ele ainda não está tão maduro quanto o próprio Typescript, que já possui muitas correções e uma comunidade consideravelmente grande.

3 METODOLOGIA

Aqui serão abordados os métodos e princípios usados no desenvolvimento do sistema Booker.

3.1 PROJETO

O conjunto de funcionalidades do sistema proposto é apresentado pela tabela 1.

ID	Descrição
RF1	O software deve permitir que usuário se cadastre e gerencie sua conta no sistema
RF2	O software deve permitir e garantir que apenas usuários cadastrados consigam emprestar, cadastrar livros e fazer avaliações
RF3	O software deve permitir a filtragem de busca por gênero, autor, título, raio de distância e data de publicação
RF4	O software deve permitir que usuários não cadastrados possam fazer buscas e visualizar os itens do catálogo
RF5	O software deve permitir a sugestão de usuários para novas classificações de gêneros literários
RF6	O software deve suportar um usuário moderador com poderes de moderar usuários/livros/avaliações e fazer o cadastro de gêneros
RF7	O software deve suportar um sistema de balanceamento de empréstimos baseado em créditos de empréstimo
RF8	O software deve permitir a extensão de prazo de empréstimo mais de uma vez
RF9	O software deve permitir o sistema de doações
RF10	O software deve permitir o cadastro de fotos pré-emprestimo para averiguar o estado de conservação do livro caso ocorra algum dano. Permitir contestação da parte recebedora.
RN1	Os créditos devem ser distribuidos da seguinte forma: Ao fornecer o livro do empréstimo e este empréstimo ser bem sucedido, o usuário que forneceu o livro irá ganhar 1 crédito de empréstimo
RN2	Os créditos devem ser consumidos da seguinte forma: Ao solicitar um empréstimo e este ser bem sucedido, o usuário beneficiário irá consumir 1 crédito de empréstimo
RN3	A solicitação só será permitida caso o usuário requisitante tenha ao menos 1 crédito
RN4	Ao cadastrar o primeiro livro e apenas nessa situação, o usuário receberá 1 crédito para que possa iniciar o seu uso na plataforma
RN5	O empréstimo irá ter duração fixa inicial de 1 semana
RN6	A extensão de prazo de empréstimo deverá ser feita através de pagamento em BRL, rateado em 60% entre o dono do livro e 40% para a plataforma
RN7	O local e data de empréstimo serão acordados entre os usuários
RN8	As avaliações serão baseadas em conservação do material durante o empréstimo, tempo de entrega, tempo de devolução
RN9	A conta do usuário deverá conter informações como Endereço, CPF, Nome completo e Nome de usuário
RN10	A partir do endereço será possível determinar a distância entre as partes
RN11	O cadastro do livro deverá conter: Título, Nº de páginas, gênero, autor, editora e imagem do livro
RN12	Ocorrendo dano e averiguado, a seguintes opções serão fornecidas: Pagamento em dinheiro no valor atual do livro (sem considerar promoções), reposição do material por um novo por conta de quem inflingiu o dano ao livro, esse livro novo deve ser do mesmo valor do antigo, sem considerar frete, devendo ser acordado se será o mesmo titulo, ou outro diferente.
RN13	Em último caso, haverá negativação do usuário devedor.
RN14	Informa-la enquanto o usuário visualiza o catálogo.

Tabela 1 - Levantamento de requisitos

Fonte: Próprio autor

A partir dos requisitos, foram desenvolvidos os diagramas de casos de uso, sequência e classe. Esses diagramas documentam os fluxos e entidades envolvidas nos processos desempenhados pelos usuários.

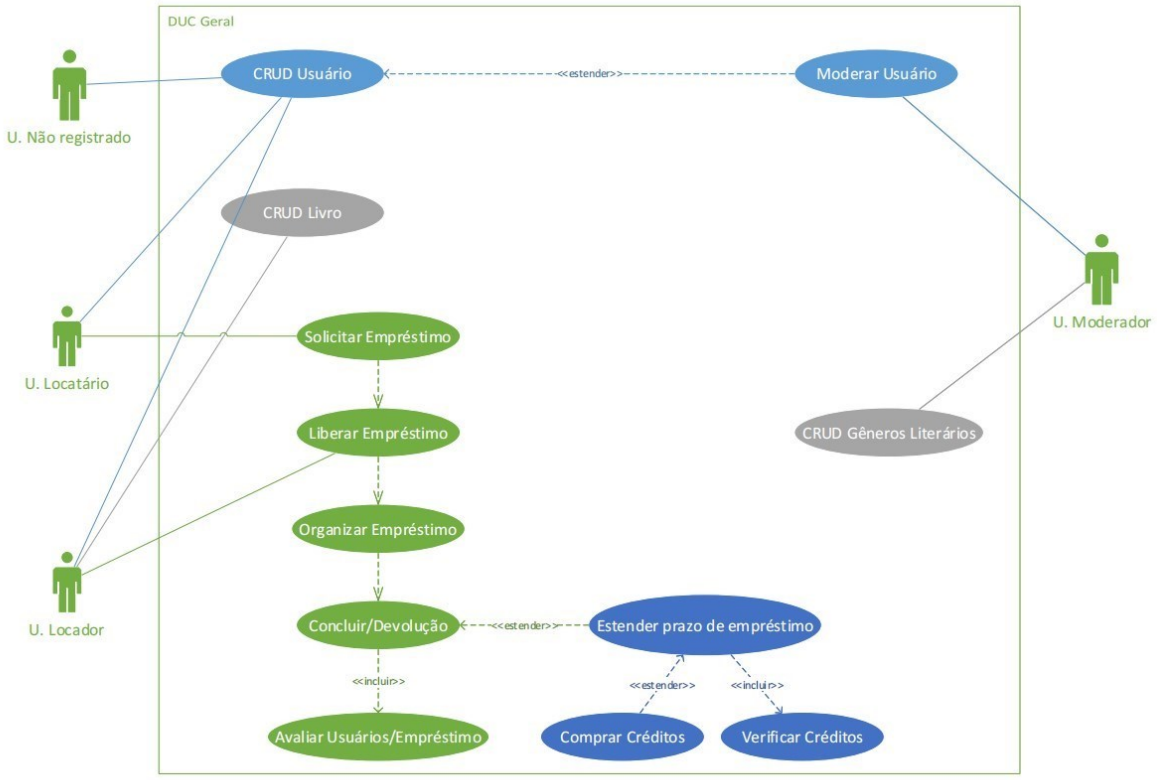

Figura 6 - Diagrama de casos de uso geral

Fonte: Próprio autor

O “usuário não cadastrado” terá que se cadastrar para se tornar um “usuário”. A partir disso, ele poderá transitar sendo um usuário locatário ou um usuário locador, dependendo do seu papel, no empréstimo que estiver lidando. Esse usuário cadastrado, poderá cadastrar seus livros e assim, ao emprestar seus livros para outros usuários, receberá créditos para solicitar empréstimos. Esses empréstimos, que tem tempo inicial fixo em 1 semana, poderão ser estendidos através de um outro tipo de crédito, este, pago com dinheiro real. O motivo para desenvolver pagamento para a extensão de empréstimo é que assim o dono do livro tende a aceitar essa requisição, já que receberá 60% do valor. Ademais, o restante irá para pagamento de domínio do site e eventual custeio fixo do servidor, como a eletricidade e/ou manutenção.

Envolvendo o empréstimo, a primeira etapa é a solicitação do material. Nessa etapa o objeto “empréstimo” é criado no sistema. Caso o livro esteja atualmente em um processo de empréstimo confirmado, o usuário locatário será questionado se deseja entrar para a fila de requisitantes daquele livro. Caso a resposta seja sim, os

empréstimos serão ordenados conforme a data de criação da solicitação, sendo que o usuário que fez a solicitação primeiro terá prioridade, como em uma estrutura de fila. Esse fluxo está representado na figura 7.

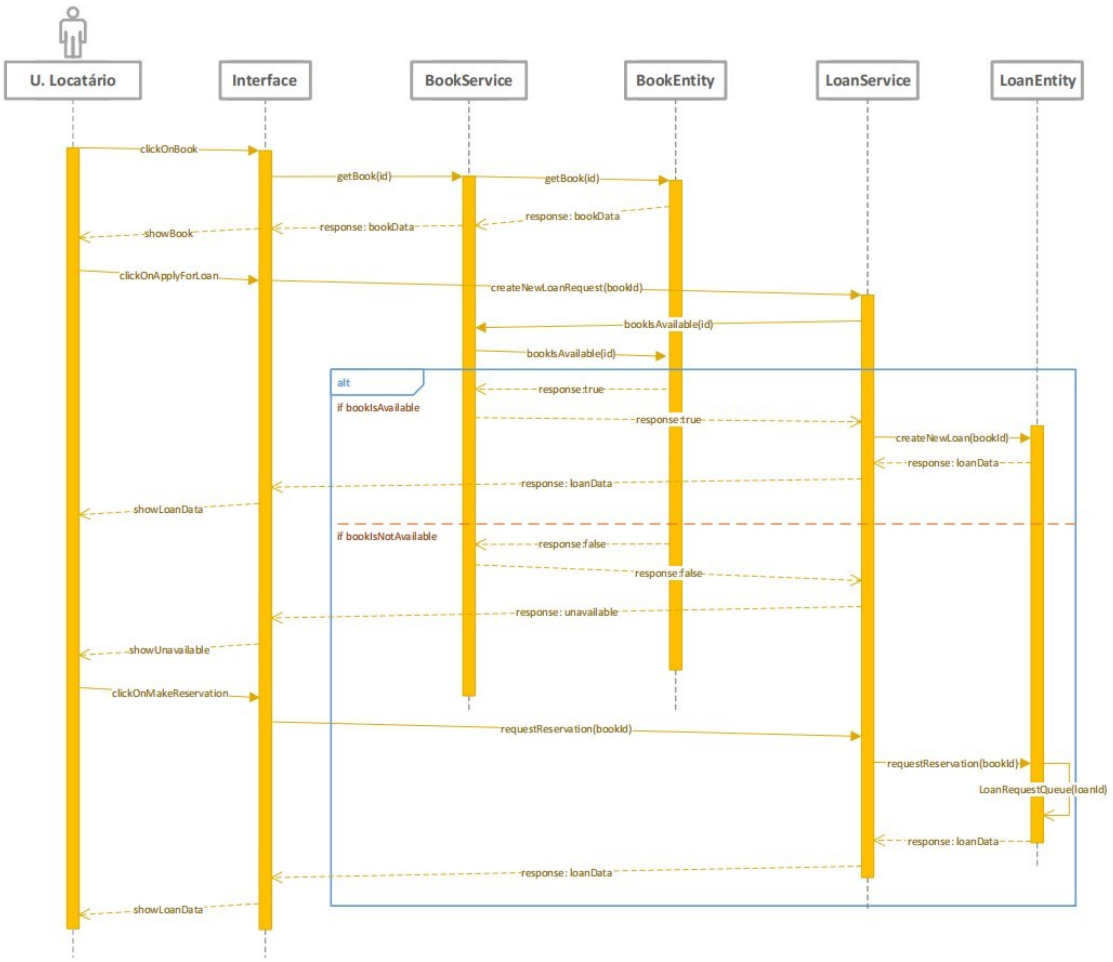

Figura 7 - Diagrama de sequência para a solicitação de empréstimo

Fonte: Próprio autor

Após a ação de solicitar empréstimo pelo usuário locatário, é necessário que o usuário locador, ou seja, o proprietário do livro, autorize ou rejeite esta solicitação. Caso o locador rejeite o empréstimo, então o processo será encerrado e o livro estará novamente disponível. De outra forma, se o empréstimo for autorizado, os usuários seguirão para o processo de organização do empréstimo, em que as duas partes ficarão responsáveis por definir onde e como será feito o contato e o fornecimento do material. A liberação é representada pelo diagrama da figura 8.

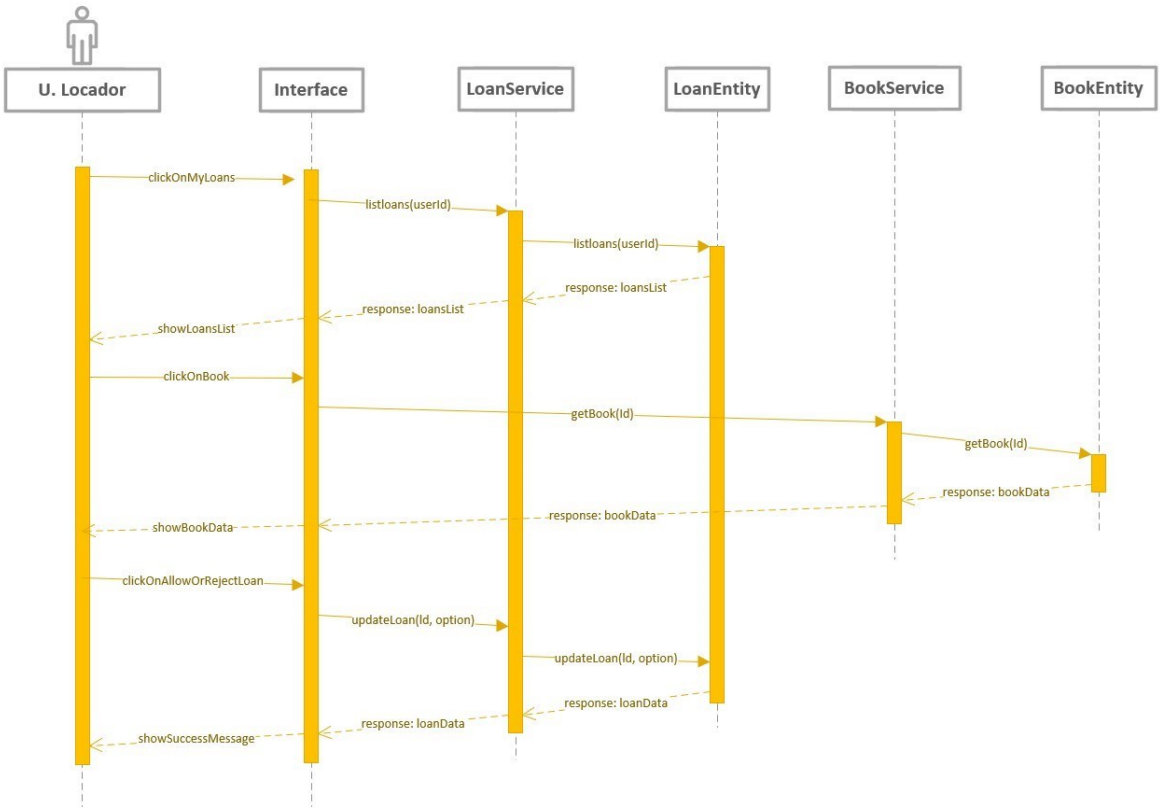

Figura 8 - Diagrama de sequência para a liberação de empréstimo

Fonte: Próprio autor

Ainda tratando da organização do empréstimo, o intuito de deixar essa responsabilidade aos próprios envolvidos na locação, é para que aumente a taxa de sucesso de empréstimo firmado e concluído. Assim, as partes terão a liberdade de definir local, data e hora para o empréstimo, ao mesmo tempo em que essa camada do negócio fica transparente para o sistema. Da mesma forma que na organização de entrega, será possível utilizar os mesmos meios no momento da devolução. Os usuários ficam livres para tratar disso, sendo que o Booker oferecerá os dados de e-mail e telefone para os envolvidos. Cada papel do empréstimo tem seu diagrama de organização, seguindo a figura 9 e 10.

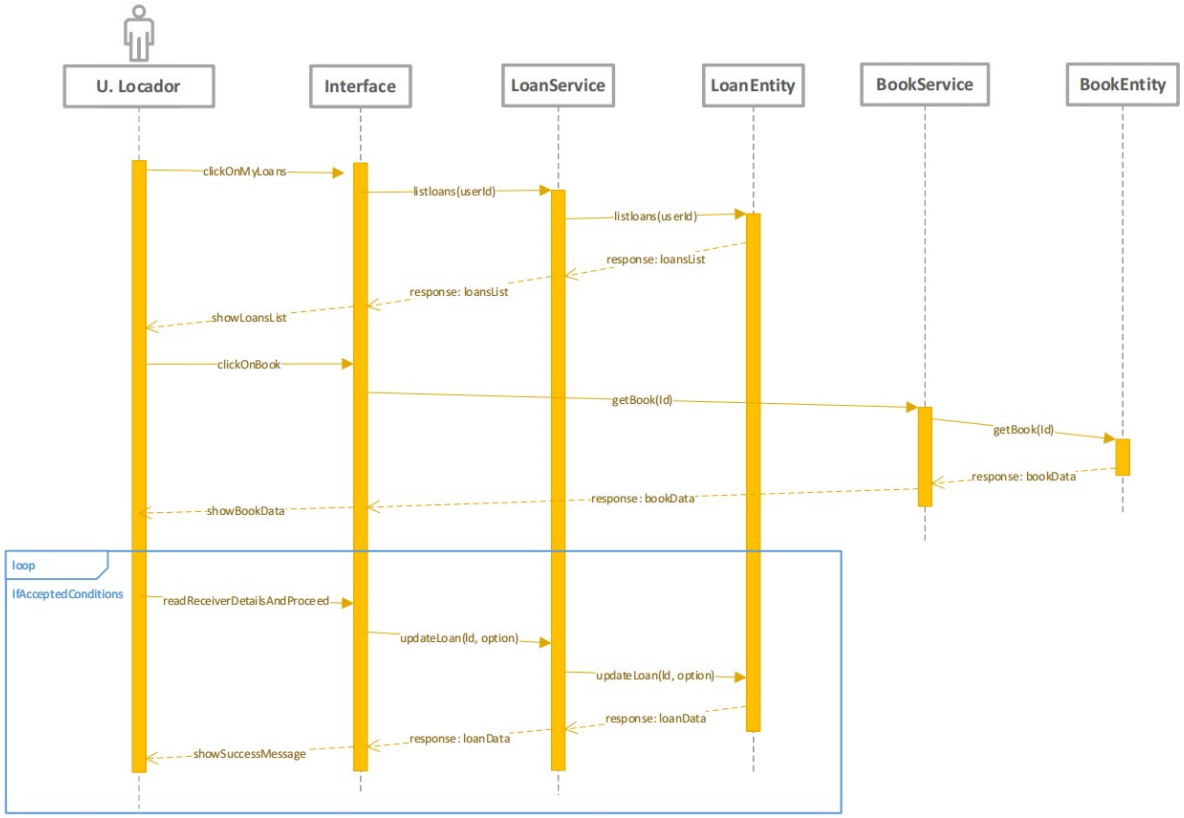

Figura 9 - Diagrama de sequência para a organização de empréstimo (Locador)

Fonte: Próprio autor

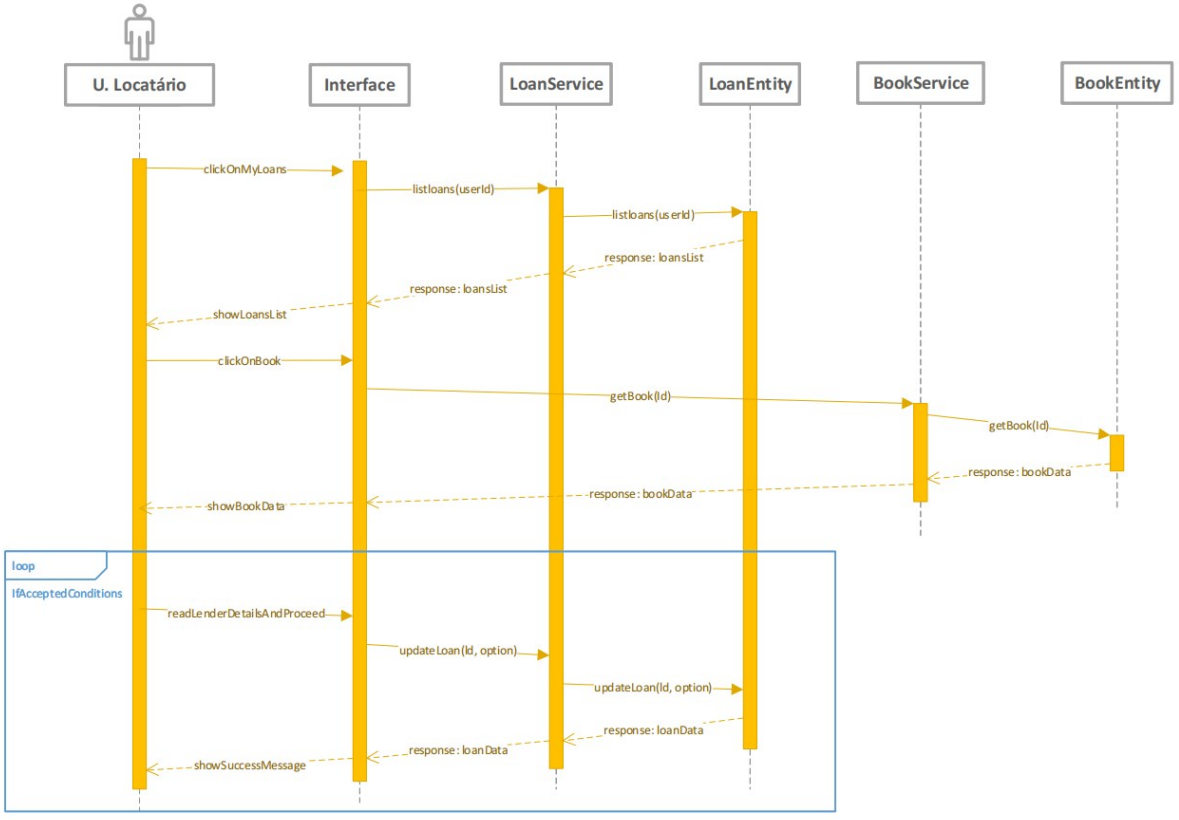

Figura 10 - Diagrama de sequência para a organização de empréstimo (Locatário)
Fonte: Próprio autor

Após o prazo de 1 semana, o processo de devolução é iniciado. O sistema irá notificar o locatário com antecedência de 1 dia. O processo de devolução pode ser estendido ou não, mediante requisição do usuário locatário e aprovação do locador. A extensão de empréstimo tem o acréscimo de mais uma semana, sendo necessário requisitar outra extensão para aumentar novamente o prazo de devolução, sendo que, contanto que o dono do livro aprove, não há um limite de extensão de empréstimo. A figura 11 exemplifica esse processo. Após os processos de notificação e, caso necessário, extensão do empréstimo feitos pelo sistema, fica a cargo dos usuários novamente se organizarem para efetuar a devolução e a posterior avaliação.

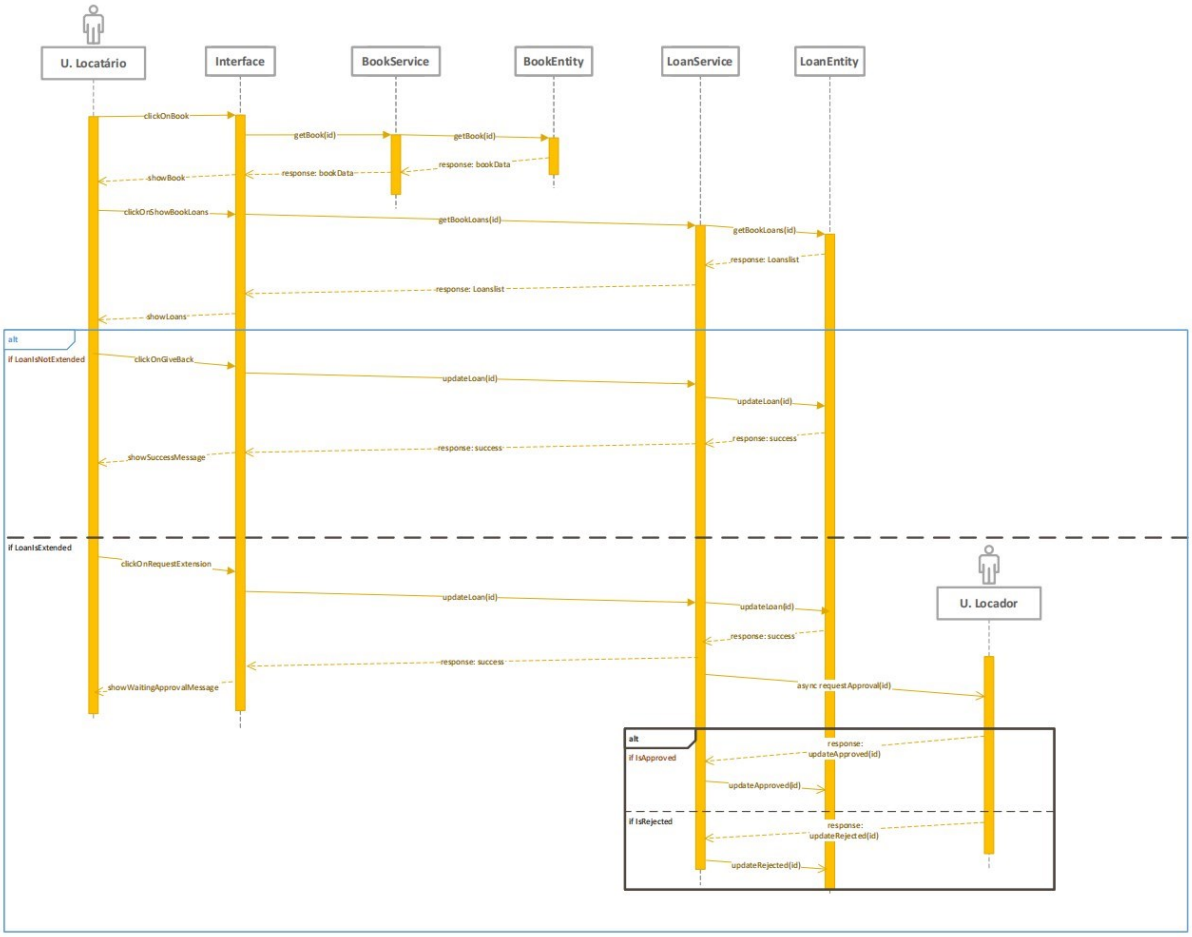

Figura 11 - Diagrama de sequência para a devolução de empréstimo

Fonte: Próprio autor

Já a avaliação do usuário envolve uma classificação de 1 a 5 estrelas em diversos aspectos como: Conservaçãodo material durante o empréstimo, agilidade na entrega, agilidade de devolução e outros aspectos envolvendo uma identificação de bons usuários.

A figura do administrador/moderador fica limitada a moderar usuários e a cadastrar gêneros literários.

A estrutura do projeto segue um modelo orientado a eventos. Para tratar dos eventos, há os serviços. Cada serviço fica responsável por lidar com uma rota do sistema. Essas rotas recebem requisições HTTPS. A figura 12 é um caso do sistema em que é persistido um novo registro "*Book*".

```
276  booksRouter.post('/', ensureAuthenticated, async (request, response) => {
277    const {
278      isbn10,
279      isbn13,
280      bookTitle,
281      bookAuthor,
282      bookPublisher,
283      categories,
284      publishedDate,
285      maturity,
286      description,
287      pageCount,
288      language,
289      ratingCount,
290      averageRating,
291      imageLinks,
292    } = request.body;
293    const createBook = new CreateBookService();
294
295    const book = await createBook.execute({
296      isbn10: isbn10 || '',
297      isbn13: isbn13 || '',
298      bookTitle,
299      bookAuthor,
300      bookPublisher,
301      categories: categories || '',
302      publishedDate,
303      maturity: maturity || '',
304      description,
305      pageCount: pageCount || 0,
306      language: language || '',
307      ratingCount: ratingCount || 0,
308      averageRating: averageRating || 0,
309      imageLinks,
310      user: request.user.id,
311    });
312
313    return response.json(book);
314  });
```

Figura 12 – Código da rota post referente à entidade Book.
Fonte: Próprio autor

Na figura 12, o usuário fez uma requisição HTTPS com corpo contendo dados em JSON e o header contendo um *token* de autenticação, para uma rota como esta:

```
Post: serverURL/book
```

Então, tudo referente a requisição estará contido no parâmetro *request*. A seguir, os dados são desestruturados do corpo da requisição e o serviço de criação de livro é instanciado. É enviado para a função de execução, todos os dados desestruturados. Nessa função de execução, são feitas validações, bem como tratativas de erro. Acabando seu processamento, se bem sucedido, é retornado um objeto livro preparado para ser persistido no banco de dados. Se persistido com sucesso, um exemplar do objeto criado, contendo o *id* gerado pelo banco de dados, é retornado para o usuário e é emitida uma mensagem de sucesso. As outras rotas seguem o mesmo formato.

Desta forma, essas rotas já estão um passo à frente para a conversão do monolito em microserviços, já suportando requisições à *API* no formato *REST.*

3.2 MODELAGEM DE ENTIDADES E A PERSISTÊNCIA DE DADOS

As classes do sistema estão vinculadas a pelo menos uma das três principais entidades, sendo elas: Livro, usuário e empréstimo. Sua estruturação segue o esquema da figura 13.

Todas essas classes, ao serem persistidas, introduzem pelo menos três colunas no banco de dados que são geradas com um valor padrão. Todos os campos que são chave primária da tabela, têm o seu id sendo um *UUID*. Além deste, os outros dois campos são o *createdAt* e *updatedAt*. Essas colunas são no formato data e hora, e são geradas em duas situações: O *createdAt* e *updatedAt* são inicialmente gerados quando um registro é criado, por um procedimento do próprio banco de dados. A outra situação, é no momento em que um registro já criado, sofre alguma alteração nos seus valores. Quando isso ocorre, o *updatedAt* é gerado novamente com a data e hora em que isso ocorreu.

Todas tabelas que tem uma relação de muitos para muitos, estão usando uma classe associativa. Isso ocorre com maior relevância em domicílios e empréstimos, já que são classes de grande importância para os processos básicos do empréstimo. Os domicílios mantêm um histórico de todos os endereços que um usuário já registrou, bem como o e seu endereço ativo. O endereço ativo é o endereço vinculado a um

usuário através da tabela domicílios que não contém um valor para o *eliminatedAt*, indicando que, por não ter sido eliminado, este ainda é o endereço sendo usado pelo usuário. A outra tabela associativa é a de empréstimos, que regula desde o estado de autorização para empréstimo, até a devolução. Vinculado a ele, estão as ocorrências e as avaliações dos usuários.

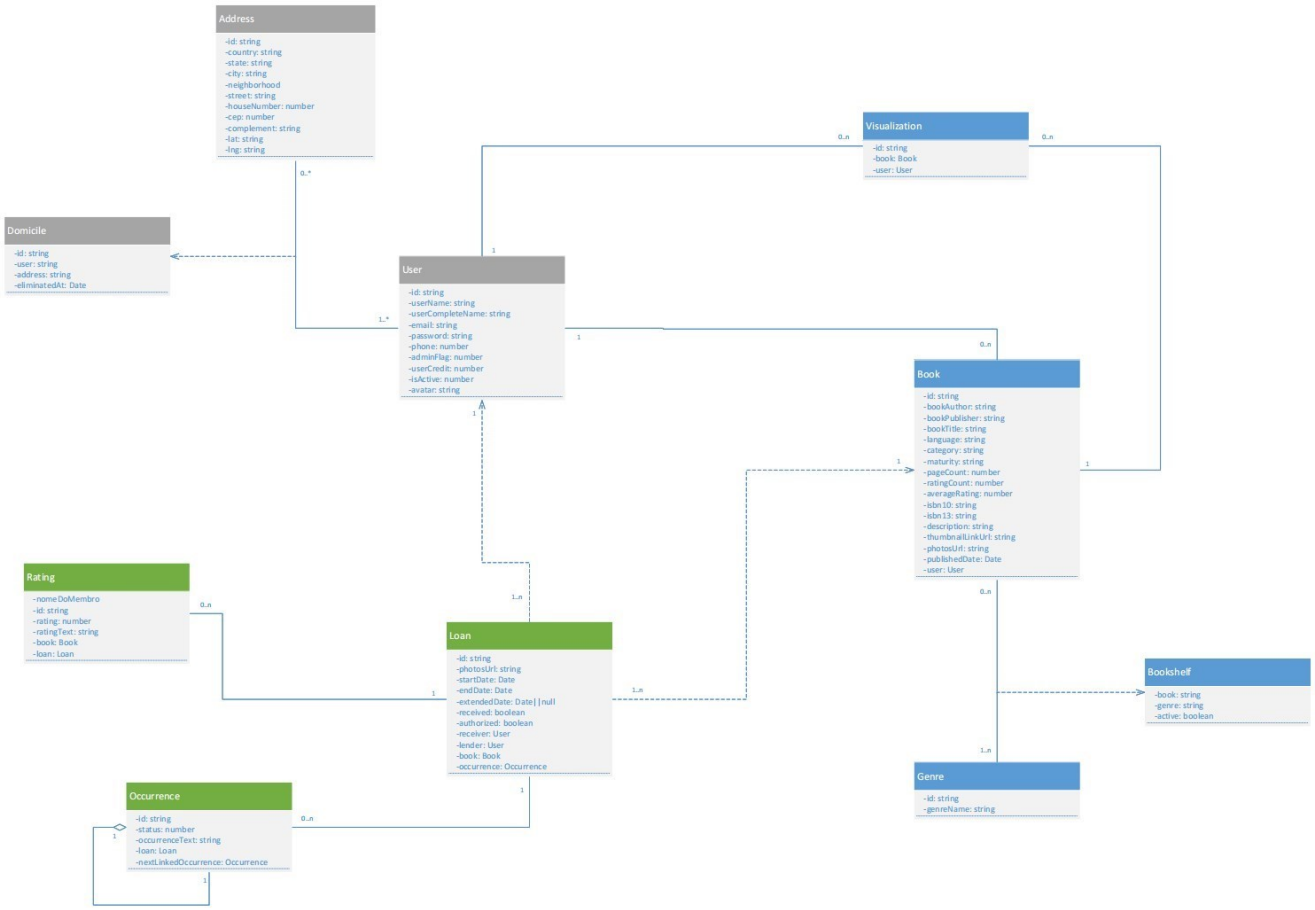

Figura 13 – Diagrama de Classes.

Fonte: Próprio autor

3.3 A CONVERSÃO EM MICROSERVIÇOS

Com os domínios do sistema já definidos, a conversão em microserviços se dará seguindo etapas de componentização.

- Os serviços vinculados a uma rota serão isolados junto a essa rota, de forma que os dois se tornem uma API única.
- Será feita a imagem de um container Docker referente a esse bloco único de código. Assim, o serviço executará contido no container e será acessível pelo servidor que hospedar o container.

- Dentro dessa imagem Docker, estará um sistema operacional Linux Alpine configurado para, ao ser iniciado, instanciar um servidor node.js. Esse servidor estará usando express.js com dependências mínimas para executar sua função.

Com essa arquitetura estabelecida, a conversão do banco de dados para o MongoDB será benéfica. Isso porque a sobrecarga na rede, causada pelo cruzamento da modelagem de diversas entidades usadas para estabelecer o relacionamento entre os dados no banco de dados será bem reduzida, devido à estrutura de documentos e coleções do MongoDB. Essa sobrecarga em processos relacionais só deve ocorrer nos casos de microserviços que usam o modelo de troca de informações baseados em API. Então, caso exista forte relacionamento entre entidades no sistema, a necessidade de fazer chamadas individuais em cada microserviço aumenta a cada relacionamento. Usando a classe de empréstimos para exemplificar: Todo empréstimo tem ao menos dois usuários envolvidos, sendo um deles o dono do livro e o outro o usuário locatário, que solicita um material. Também há 1 livro envolvido e possivelmente uma ocorrência.

4 DESENVOLVIMENTO

As etapas desempenhadas durante o desenvolvimento do projeto serão elucidadas nesta seção.

4.1 FRONT-END

A parte do sistema considerada *client-side,* foi desenvolvida utilizando a biblioteca React.js. A estrutura do projeto front-end é definida por:

- **Assets**: Diretório que contém imagens, logos e o CSS responsável pela importação de fontes utilizando a *API* do *Google Fonts*.
- **Components**: Aqui foram desenvolvidos elementos da interface que são diversas vezes reutilizados. Exemplificando, botões, campos de formulário, estrutura de exibição de dados de um livro e o próprio template da aplicação estão contidos aqui.
- **Pages**: Contém o núcleo das páginas, bem como seus estados e regras condicionais.
- **Routes**: Define as rotas das *URLs* do sistema para o mapeamento das páginas e navegação. Também contém uma estrutura para fornecer estados e contextos pela *DOM*, através de provedores.
- **Services**: O serviço do *Axios*, que é a biblioteca usada para fazer requisições assíncronas para APIs, é configurado aqui. Também é definida a especificação da API de Geocodificação.
- **Storage**: É onde os contextos do projeto são definidos, a partir dele é possível providenciar dados através de todo o front-end.
- **Styles**: Os estilos globais do React estão definidos no arquivo global.ts desse diretório.

Existem duas chamadas à API de terceiros em formulários durante a interação com o f*ront-end*. Sendo uma, a busca de endereço via CEP e outra a busca de

informação do livro baseada em ISBN. A primeira chamada ocorre quando o usuário estiver fazendo o cadastro de sua conta. Ao informar seu CEP e sair desse campo, será feita a busca pelos dados de sua residência para o preenchimento dos campos de endereço. Essa chamada não exige autenticação. Da resposta da API, é desestruturado o logradouro, bairro, localidade e uf para inserir no formulário. Na figura 14, é exibida a chamada à API inserindo o CEP preenchido pelo usuário como parâmetro de requisição.

```
const { logradouro, bairro, localidade, uf } = await fetch(
  `https://viacep.com.br/ws/${cep}/json/`
).then((res) => res.json());
```

Figura 14 – Chamada à API para busca de endereço baseado em CEP.
Fonte: Próprio autor

O ISBN também é usado para fazer uma chamada à API. No cadastro de livros, ao informar um ISBN, seja ele ISBN10 ou ISBN13, é feita uma requisição para o Google Books. Essa requisição não exige autenticação e retorna dados visando preencher todos os campos do formulário de cadastro de livro. Também é possível retornar uma *URL* que contém uma imagem para exibir na listagem de livros.

Durante o período em que *o front-end* não estava desenvolvido, foi necessária a utilização do Insomnia para testar as rotas do *back-end* durante o seu desenvolvimento. O Insomia é uma ferramenta que mimetiza um cliente, agindo como um *front-end*. A partir disso, é possível fazer requisições HTTP para as rotas e verificar as respostas emitidas, desde os dados até os metadados da requisição.

```
const {
  items: {
    0: { volumeInfo },
  },
} = await fetch(
  `https://www.googleapis.com/books/v1/volumes?q=:isbn${value}`
).then((res) => res.json());

setBookProps({
  ...bookProps,
  bookTitle: volumeInfo.title,
  bookAuthor: volumeInfo.authors?.toString() || [""],
  bookPublisher: volumeInfo.publisher || "",
  publishedDate:
    volumeInfo.publishedDate || new Date("2021/01/01").toUTCString(),
  categories: volumeInfo.categories || [""],
  maturity: volumeInfo.maturityRating === "NOT_MATURE" ? "Livre" : "+18",
  description: volumeInfo.description || "",
  pageCount: volumeInfo.pageCount,
  language: volumeInfo.language || "",
  isbn10: volumeInfo.industryIdentifiers[1]?.identifier || 0,
  isbn13: volumeInfo.industryIdentifiers[0]?.identifier || 0,
  ratingCount: volumeInfo.ratingCount || 0,
  averageRating: volumeInfo.averageRating || 0,
  imageLinks: volumeInfo.imageLinks || "",
});
```

Figura 15 – Chamada à API para busca de livro.

Fonte: Próprio autor

Na figura 15 está a chamada que é feita para o Google Books. O parâmetro passado em *value* é o que está contido no campo do formulário de ISBN. Essa chamada só é feita caso o tamanho de *value* seja idêntico ao tamanho de um ISBN10 ou ISBN13.

A orientação de design usada no front-end envolve *material design* e *Glassmorphism*. O *material design* é o que costuma fazer o uso de camadas usando sombras, para a impressão de sobreposição de elementos com sensação de altura. Já o *Glassmorphism*, utiliza a transparência como núcleo de sua orientação. Adicionado a isso, em alguns locais é utilizado o *Blur* Gaussiano para gerar a sensação de embaçamento da transparência. O modo de iluminação padrão e inicialmente o único é o tema escuro.

4.2 BACK-END

O projeto em back-end, que é totalmente desacoplado do front-end, é desenvolvido em *node.js* com o framework *express.js*. Modelado com o formato de API, quando uma rota é chamada, são feitas as instanciações de serviços necessárias

para a consulta ao banco de dados. Sua estrutura de diretórios contém:

- **Config**: Diretório que contém a especificação da encriptação do *token JWT* e a configuração do serviço de upload de arquivos *Multer.*
- **Database**: Define a instanciação da conexão com o banco de dados, bem como especifica suas *migrations*.
- **Errors**: Especifica o modo de retorno de erros customizados, contendo a mensagem e o código desejado.
- **Middlewares**: Contém o serviço de codificação, decodificação e validação de *tokens JWT*.
- **Models**: As entidades relacionais usadas pelo ORM para estruturar os dados vindos da base de dados são definidas dentro desta pasta.
- **Repositories**: Os repositórios que precisam ser estendidos a partir do comportamento padrão, são implementados aqui.
- **Routes**: Todas as rotas que o express.js ouve, estão definidas aqui, junto com as tratativas para cada método *HTTP* esperado para elas.
- **Services**: Os serviços de criação, busca, atualização e consultas externas à APIs, estão definidos nesse diretório.
- **Tmp**: Arquivos, como imagens de perfil, são salvos nessa pasta, sendo que no banco é salva apenas uma URL que faz referência a um arquivo neste diretório.

Para lidar com o banco de dados, a biblioteca **TypeORM** é usada para contornar ameaças básicas, como o *SQLInjection*, que consiste na inserção de comandos SQL em variáveis que deveriam conter dados úteis para o sistema, com o fim de extrair informação sensível ou causar danos na estrutura do banco de dados. Essa biblioteca também pode ser usada para tratar outras falhas comuns de segurança da comunicação com o banco de dados, de forma transparente para o desenvolvedor. As chamadas ao banco são feitas através do modelo de *Repository* ou, em casos mais específicos, é usada a funcionalidade de *QueryBuilder* oferecida pelo ORM.

No momento em que um usuário é cadastrado no sistema, no front-end, existe a busca de dados baseada em CEP, feita pela API da viacep.com.br. Já no back-end, os dados buscados são enviados para um serviço de Geocodificação. Esse serviço, que é autenticado através de uma *API key,* é fornecido pela here.com. Os dados de nome de rua, número da residência, cidade, estado e país, são enviados à API de Geocodificação para entregar uma série de dados sobre o endereço fornecido. Desses dados retornados, o sistema aproveita a latitude e longitude e adiciona à chamada de criação de endereço.

Portanto, no momento de criação de um usuário acontecem três processos que chamam cada um, a sua respectiva rota *REST,* sendo por hora, o processo mais complexo desempenhado pelo servidor. Após encerrar esse processo, é feita a criação de um registro na tabela *log* da base de dados, seguindo o modelo da figura 16.

```
const registerAction = await api().post(`http://localhost:3333/logs`, {
  logText: `User with id: ${userWithId.id} created \n
  Address with id: ${createAddress.data.id} created \n
  Domicile with id: ${createDomicile.data.id} created.`,
});
```

Figura 16 – Chamada à API para registro de log.
Fonte: Próprio autor

4.3 FUNCIONALIDADES IMPLEMENTADAS

Todos os processos que o sistema desempenha, que vão além dos processos de *CRUD,* são:

- Busca de Endereço baseada no CEP fornecido pelo usuário
- Busca de coordenadas geográficas usando como parâmetro o endereço obtido pela busca baseada em CEP, ou, caso a busca tenha falhado, é baseada no endereço que o usuário editou nos campos de texto de endereço.
- Essas coordenadas geográficas são usadas para estimar a distância entre

a residência do dono do livro que está sendo listado e a residência do usuário logado no sistema.

- Ao logar no sistema, o usuário é redirecionado para a página de listagem de livros. Nesta página é possível filtrar os livros que estão disponíveis, ou, exibir uma listagem com todos os livros nas proximidades.
- Se livros que já estão em um empréstimo forem requisitados, o usuário requisitante será informado disso e indagado se deseja prosseguir com o empréstimo. Caso a resposta seja que deseja prosseguir, o empréstimo requisitado entrará para a fila de requisições pendentes para aquele material.

O cálculo de distância geográfica é determinado pela seguinte equação 1, definida por SILVEIRA (2021):

$$D = \sqrt{((LatA - LatB) * 111{,}1)^2 + ((LngA - LngB) * 111{,}1)^2 * 1{,}15} \quad (1)$$

Lat = Latitude; Lng = Longitude;

O fator de multiplicação 111,1 é definido pela curvatura da terra a cada grau de deslocamento na superfície do planeta, seja entre paralelos ou meridianos. Já o fator de multiplicação 1,15 é usado para reduzir a margem de erro com relação à média dos trajetos, já que sem esse adicional de 15%, é considerado que o deslocamento é sempre em linha reta entre as duas coordenadas.

4.4 REPOSITÓRIOS

O projeto ficará como código-aberto. O código fonte ficará em dois repositórios, um para o *front-end* e um para o *back-end*. Atualmente, ambos estão com acesso privado e serão publicados assim que a versão 1.0 for lançada. Os links para esses repositórios são:

- Front-end: https://github.com/jacksonvfo/booker-front
- Back-end: https://github.com/jacksonvfo/booker-back

5 TRABALHOS FUTUROS

Implementações necessárias para curto e médio prazo, considerando os requisitos levantados e diagramas UML, são as seguintes:

- Refinamento da interface.
- Após migrar as entidades para satisfazer a estrutura de documentos do MongoDB, é possível desacoplar ainda mais os *models* do sistema. Com isso a refatoração para microserviços será facilitada.
- Implementar delimitações na filtragem de livros além da atual, que é por cidade, tais como: bairro, estado e por um raio de n quilômetros a partir da geolocalização do usuário logado.
- Além dos filtros baseados em localização, outros métodos de filtragem devem ser: Gênero, autor e data de publicação.
- Hospedar em um serviço na nuvem, tal como, mas não necessariamente, o Microsoft Azure.
- Implementar as funcionalidades de ocorrência durante empréstimo e visualização de livro. Essa visualização é um registro de que um determinado usuário abriu um card de um livro para ver suas informações.
- Definir um *Dockerfile* que forneça parâmetros de configuração de um *container* de forma padronizada. O intuito é deixar uma estrutura reutilizável para vários microserviços, já que a arquitetura deles será muito semelhante.

Com essas atividades acima concluídas, o sistema poderá ser lançado como uma versão 1.0.

6 CONSIDERAÇÕES FINAIS

Como um processo de desenvolvimento utilizando metodologias tradicionais, uma característica inerente a esses processos, é que desde o levantamento de requisitos, até a fase de testes, é estabelecido um ciclo com diversas iterações. Esta é apenas uma de várias versões que virão, cada uma refinando um aspecto ou introduzindo uma funcionalidade. Tendo isso como verdade, é notável que ainda há muito a desenvolver e compreender a respeito das especificidades de cada tecnologia, das técnicas de levantamento e análise de requisitos e dos novos padrões da indústria que surgem para auxiliar na manutenibilidade dos sistemas dadas as novas necessidades de infraestrutura e hardware.

A arquitetura do projeto, faz o uso da comunicação JSON entre o *back-end* e o *front-end*, o que se provou bastante eficiente quando uma funcionalidade específica da interface não estava operando. Nesse contexto, diversas vezes as rotas do servidor foram testadas através de uma aplicação de requisições HTTP, como o **Insomnia**. Isso permitiu que o *back-end* fosse implementado primeiro, sem necessidade de ao alterar uma especificação de rota para fazer testes de retorno de requisição, ter que ajustar no projeto React.js o formato em que a resposta virá, para que a interface não apresente problemas e consiga executar seus processos normalmente.

Implementar cada passo do processo de desenvolvimento de software, foi essencial para um amadurecimento acelerado das minhas técnicas e métodos como um desenvolvedor. Como o projeto será open-source, é possível também a participação de outros desenvolvedores através dos repositórios do Github.com.

É esperado que após finalizada a primeira versão do Booker, seja possível participar como um facilitador à melhoria dos hábitos de leitura dos usuários que fizerem o uso do sistema. O potencial para crescimento, e desenvolvimento da ideia existe, bastando existir a oportunidade.

REFERÊNCIAS

ABRAS, C.; MALONEY-KRICHMAR, D.; PREECE, J. **User-Centered Design**. Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications, 2004.

CODING THE ARCHITECTURE. **What is a Monolith?** 2014 <http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html>. Acesso em 18 jun. 2019.

COOPER, Jerold S. **Babylonian Beginnings: The Origin of the Cuneiform Writing System in Comparative Perspective**. 2004. Disponível em: <https://krieger2.jhu.edu/neareast/pdf/jcooper/jc%20Babylonian%20Beginnings.pdf>. Acesso em 27 ago. 2021

ELINET. **European literacy policy network.** 2015 **<**http://www.eli-net.eu/fileadmin/ELINET/Redaktion/Factsheet-Literacy_in_Europe-A4.pdf>. Acesso em 17 jun. 2019.

EXPRESS. **Framework para Node.js.** 2021 < https://expressjs.com/pt-br/guide/routing.html>. Acesso em 01 ago. 2021.

FEDERACIÓN DE GREMIOS DE EDITORES DE ESPAÑA. **Hábitos de Lectura y Compra de Libros en España.** 2018. <https://www.federacioneditores.org/lectura-y-compra-de-libros-2018.pdf>. Acesso em 18 jun. 2019.

FOWLEY, Martin; LEWIS, James. **Microservices:** a definition of this new architectural term. 2014. <https://martinfowler.com/articles/microservices.html>. Acesso em 18 jun. 2019.

INSTITUTO PRO LIVRO. **Retratos da leitura no Brasil.** 4. ed. 2016. Disponível em: <https://www.prolivro.org.br/wp-content/uploads/2020/07/Pesquisa_Retratos_da_Leitura_no_Brasil_-_2015.pdf >. Acesso em 05 Jun. 2019.

NEWMAN, Sam. **Building microservices:** Designing fine-grained systems. 1 ed. Sebastopol: O'Reilly Media. 2015.

NEWMAN, Sam. **Pattern: Backends For Frontends:** Single-purpose Edge Services for UIs and external parties. Disponível em: < https://samnewman.io/patterns/architectural/bff/>. Acesso em 28 ago. 2021

RICHARDSON, Chris. **Pattern: Microservice Architecture**. 2019. Disponível em: <https://microservices.io/patterns/microservices.html>. Acesso em 29 ago. 2021

RICHARDSON, Chris. **Pattern: Monolithic Architecture**. 2019. Disponível em: <https://microservices.io/patterns/monolithic.html>. Acesso em 29 ago. 2021

ROCKETSEAT. **Cursos sobre Node.js e React.** 2019. <https://rocketseat.com.br>. Acesso em 05 jun. 2019.

SILVEIRA, Fernando Lang da. **Cálculo aproximado de distâncias com base em coordenadas de latitude e longitude.** Disponível em: <https://cref.if.ufrgs.br/?contact-pergunta=calculo-aproximado-de-distancias-com-base-em-coordenadas-de-latitude-e-longitude>. Acesso em 29 ago. 2021

VEMURI, Prashanthi; MORMINO, Elizabeth C. **Cognitively stimulating activities to keep dementia at bay.** Disponível em: <https://n.neurology.org/content/81/4/308>. Acesso em 05 jun. 2019.

W3SCHOOLS. **The world's largest web developer site.** 2019 <https://www.w3schools.com>. Acesso em 14 jun. 2019.

APÊNDICE A – TELAS DO SISTEMA.

Neste apêndice são ilustradas a principais telas do sistema desenvolvido neste TCC, mostrando as interfaces dos processos de cadastro e listagem de livros, bem como do cadastro e atualização de informações do usuário e do gerenciamento de empréstimos. Todas as imagens são capturas de tela feitas pelo autor do trabalho a partir da execução do sistema.

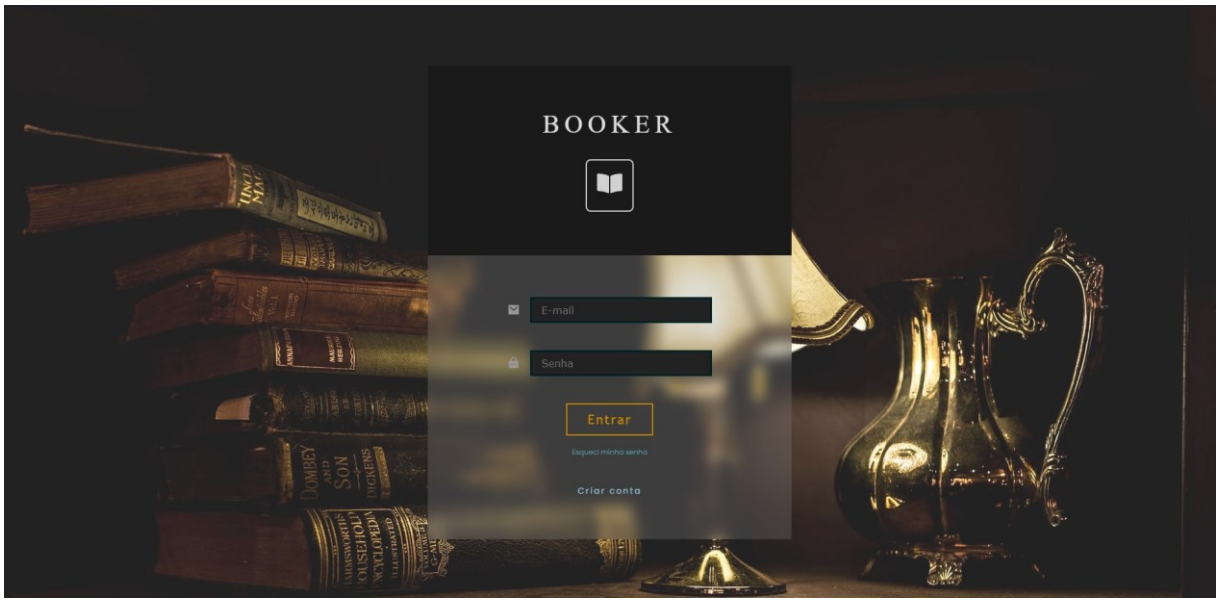

Figura A.1 – Tela de login.

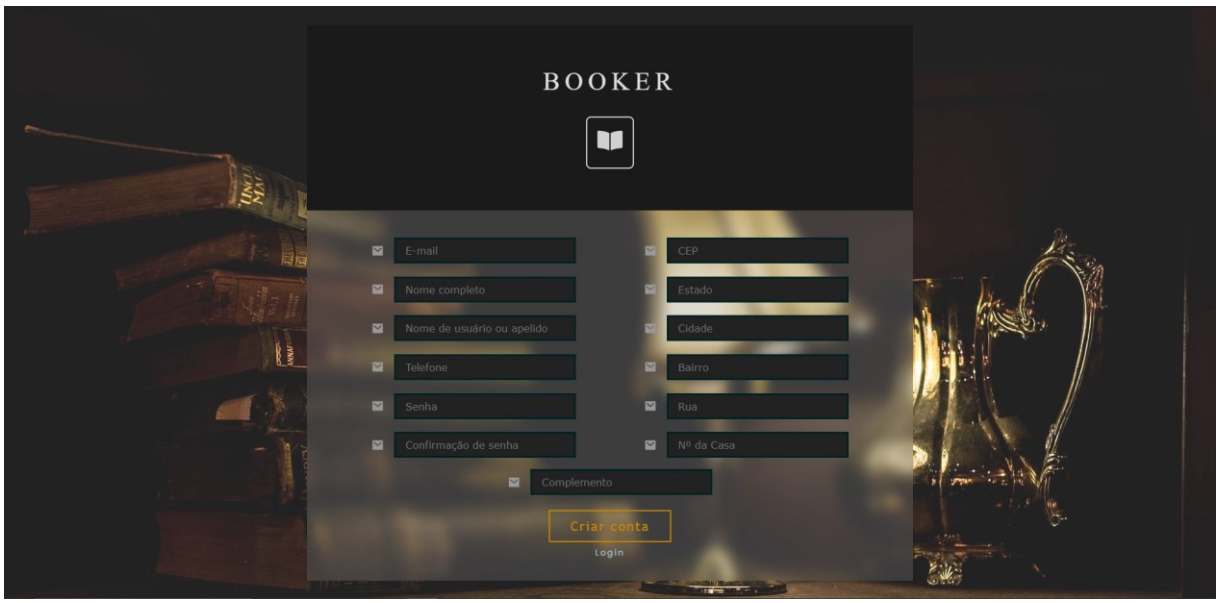

Figura A.2 – Página de criação de usuário.

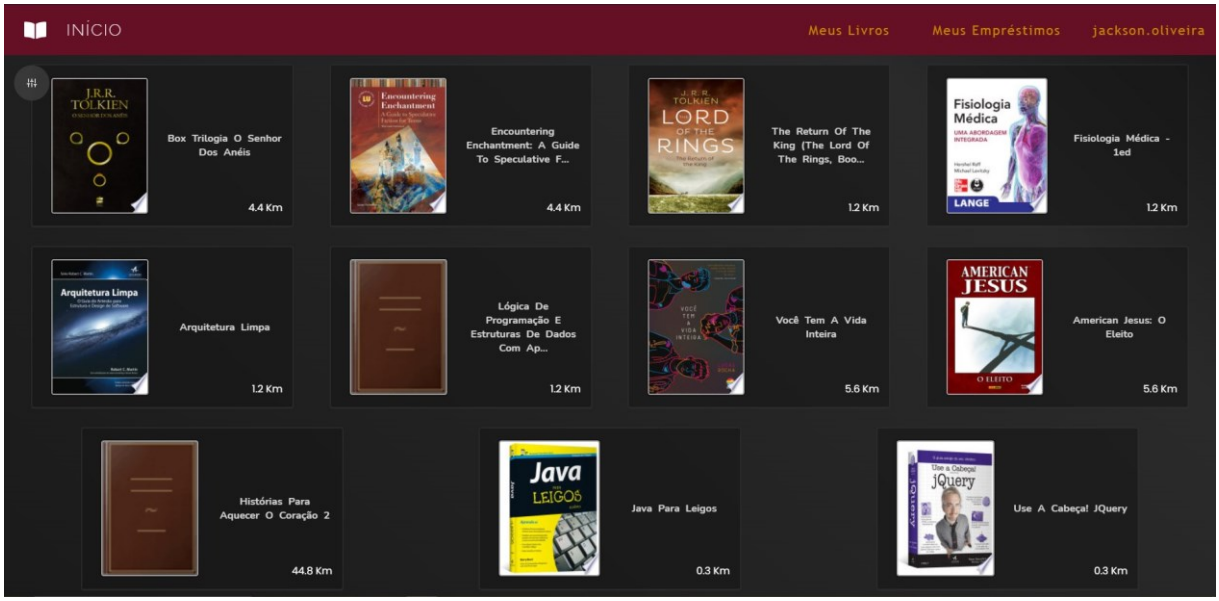

Figura A.3 – Página de listagem de livros inicial.

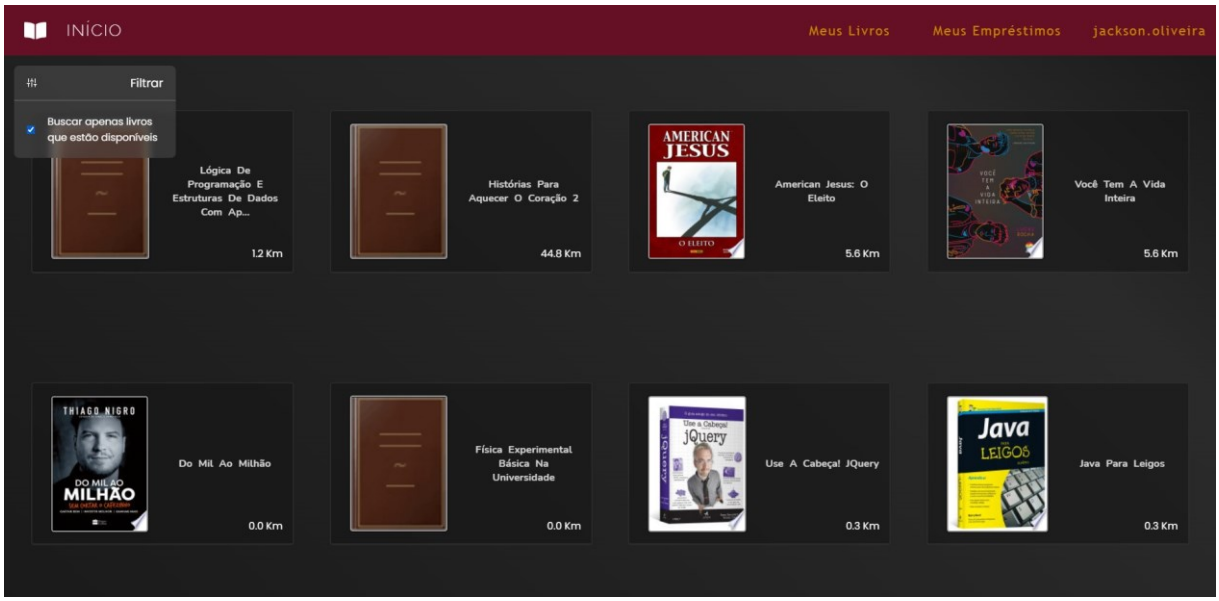

Figura A.4 – Página de listagem de livros inicial com filtro de disponibilidade de livro.

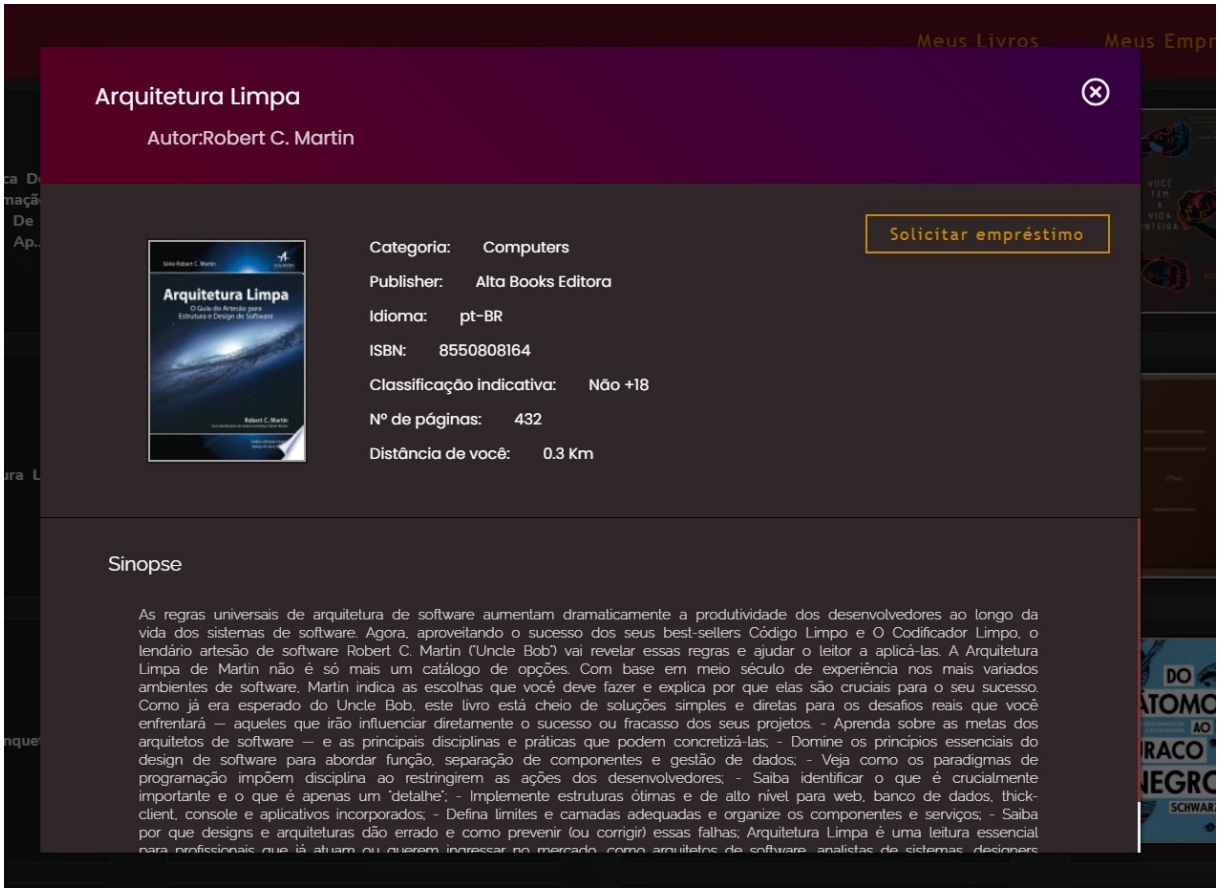

Figura A.5 – Exibição de dados do livro na página de listagem inicial.

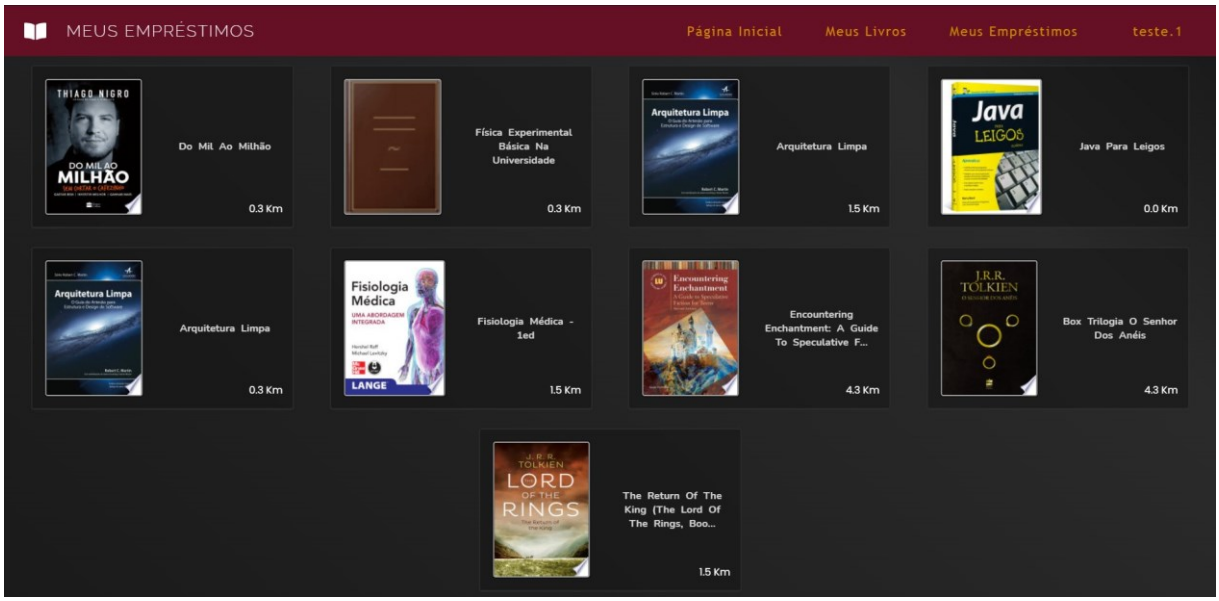

Figura A.6 – Listagem de livros com processo de empréstimo passado ou atual envolvendo o usuário logado.

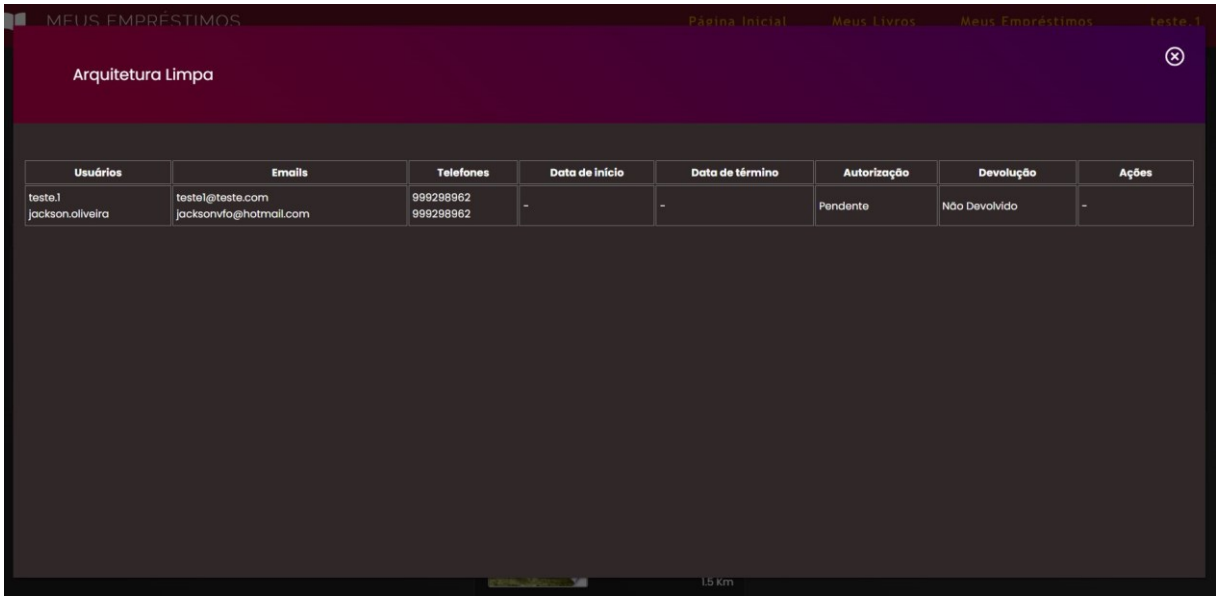

Figura A.7 – Exibição de dados de empréstimo pendente na visão do usuário locatário.

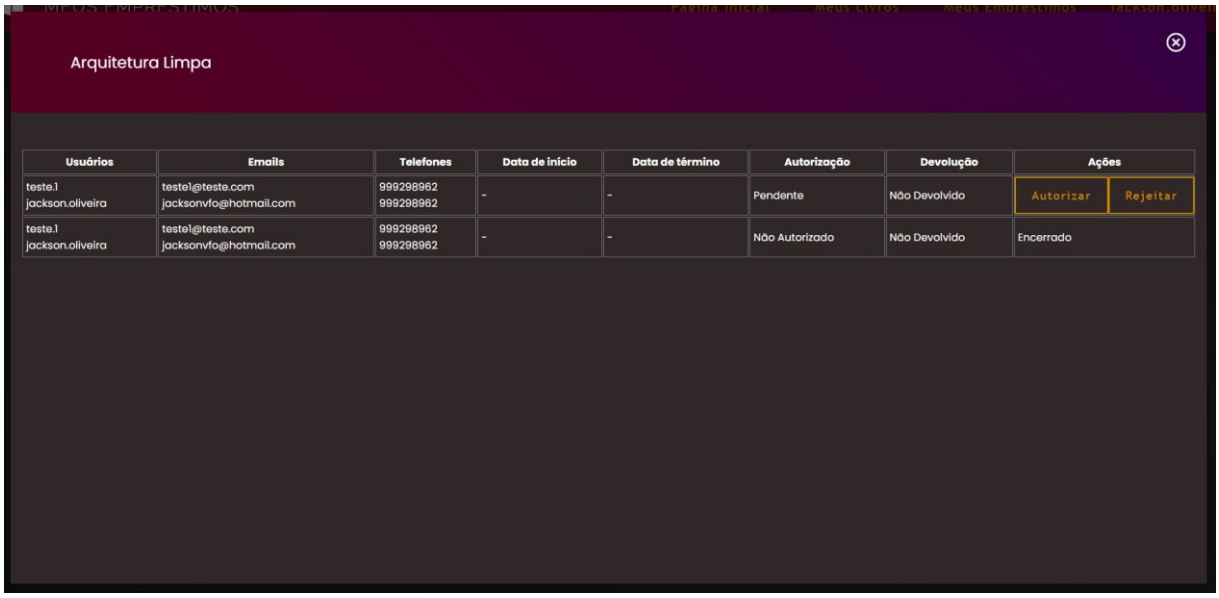

Figura A.8 – Exibição de dados de empréstimo pendente na visão do usuário locador.

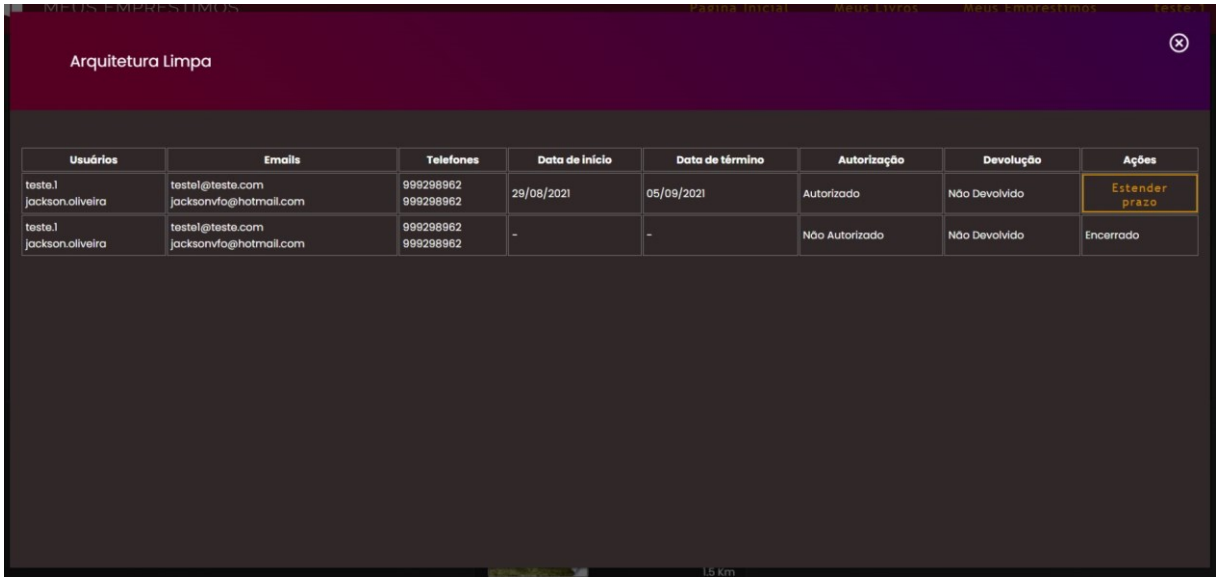

Figura A.9 – Exibição de dados de empréstimo autorizado na visão do usuário locatário.

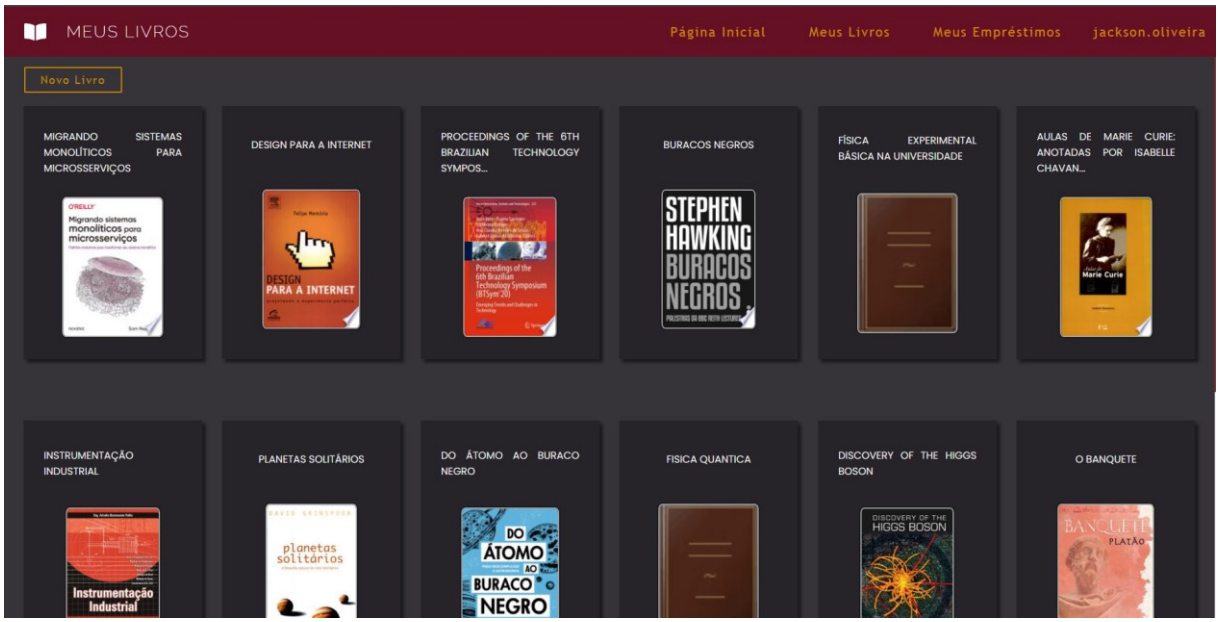

Figura A.10 – Listagem de livros cadastrados pelo usuário logado.

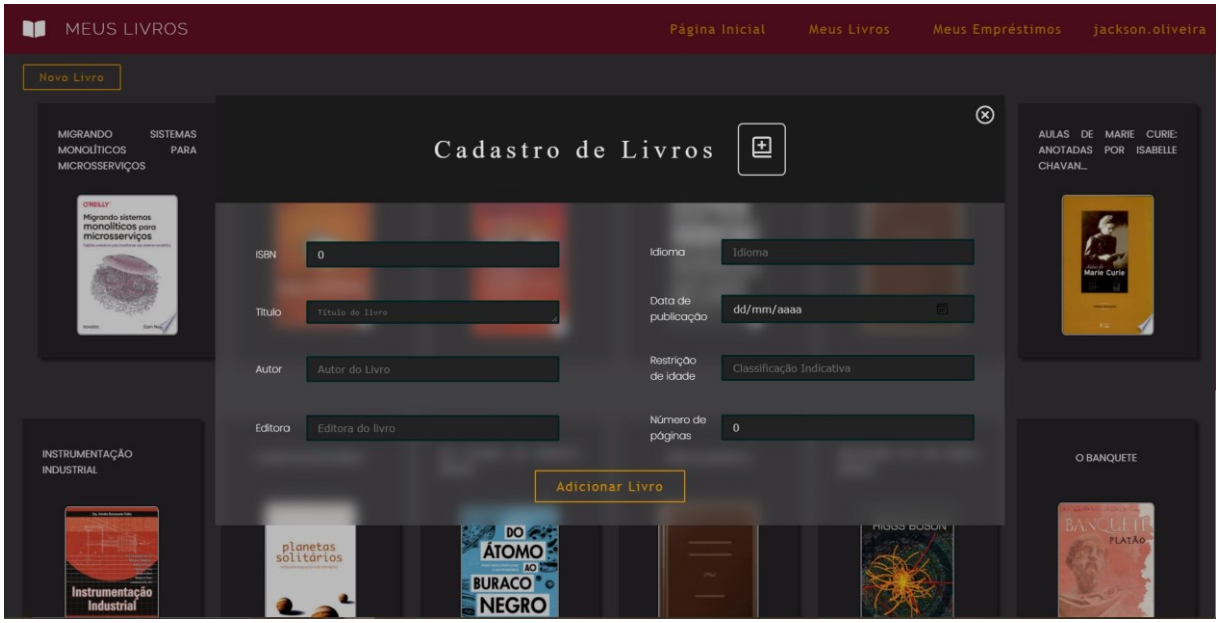

Figura A.11 – Tela de cadastro de livros.

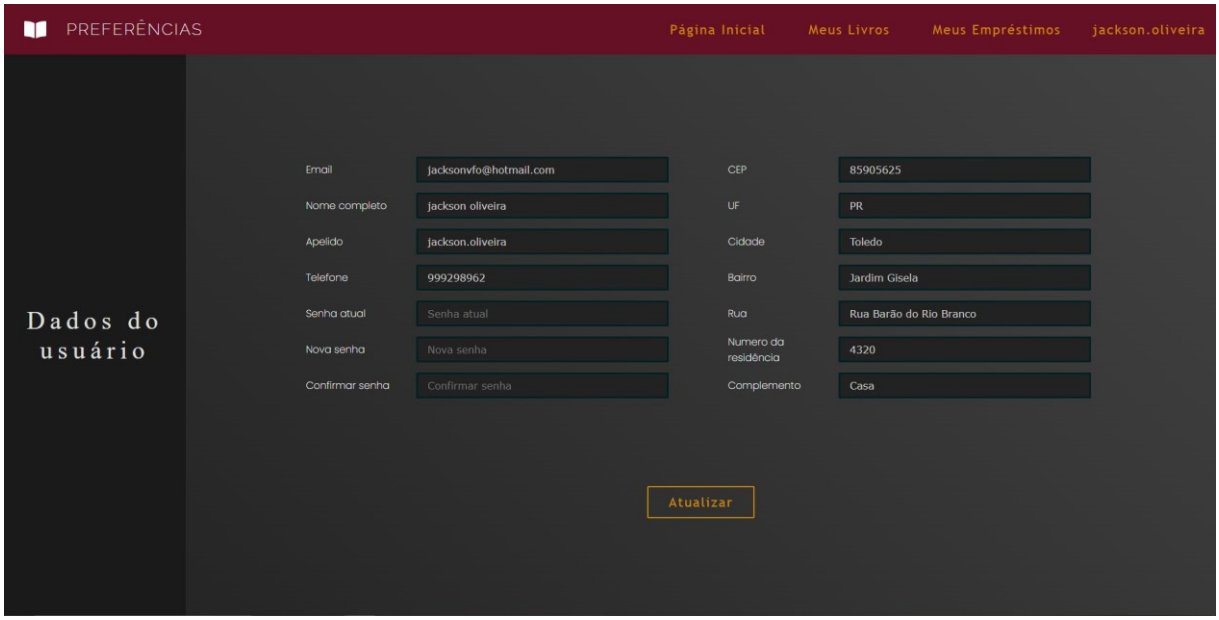

Figura A.12 – Tela de preferências e dados do usuário.