ESCOLA SENAI “HENRIQUE LUPO”
DESENVOLVIMENTO DE SISTEMAS

ALLICIA TONDATO MORETE
GIOVANA MAZZONE VARUSSA
LETICIA GABRIELY MORAES
YASMIN DE MOURA MORGADO SILVA

BIBLIOTECA VIRTUAL: Jardim dos livros

ARARAQUARA
2024

ALLICIA TONDATO MORETE

GIOVANA MAZZONE VARUSSA

LETICIA GABRIELY MORAES

YASMIN DE MOURA MORGADO SILVA

BIBLIOTECA VIRTUAL: Jardim dos livros

Trabalho de Conclusão de Curso/Projeto apresentado como requisito parcial para a obtenção do certificado de Desenvolvimento de sistemas, da Escola SENAI “Henrique Lupo”.

Orientadores: Prof. Alex Fernando Stocco
Prof. Ivo Conceição Neto

ARARAQUARA

2024

AGRADECIMENTOS

Às nossas famílias, expressamos nossa mais profunda gratidão por todo o apoio incondicional ao longo dessa jornada. Vocês foram a base sólida que nos sustentou, celebrando conosco cada vitória e oferecendo conforto e força nos momentos mais desafiadores.

Aos amigos e colegas, que compartilharam risadas, aprendizados e desafios ao longo do caminho, nosso muito obrigado. A amizade e a parceria de vocês foram fundamentais para tornar essa trajetória mais leve e significativa.

Aos professores, que não apenas transmitiram conhecimento, mas também nos inspiraram com suas palavras, exemplos e ensinamentos, agradecemos por cada troca de ideias e cada incentivo. Vocês foram parte essencial da construção de nossa trajetória acadêmica e pessoal.

Vivemos esperando
O dia em que seremos melhores
Melhores no amor,
Melhores na dor,
Melhores em tudo...
(Jota Quest – “Dias Melhores”).

RESUMO

Este trabalho apresenta o desenvolvimento de uma plataforma intuitiva para o gerenciamento de livros em bibliotecas, integrando funcionalidades como cadastro, busca, reserva e monitoramento de exemplares. O sistema foi projetado para atender tanto administradores, responsáveis pelo gerenciamento e manutenção do acervo, quanto usuários, que podem realizar consultas e reservas de forma eficiente. A implementação foi realizada utilizando tecnologias como Flask, banco de dados relacional e integração com a API do Google Books para obtenção de informações adicionais sobre os livros, como capas e descrições. Além disso, o sistema prioriza a usabilidade, incorporando princípios de design responsivo para garantir acessibilidade em diferentes dispositivos. Funcionalidades adicionais, como upload de imagens e registro de múltiplas categorias por livro, foram implementadas para atender às demandas específicas do público-alvo. O projeto também aborda questões de segurança, como autenticação de usuários, controle de permissões e criptografia de senhas. A validação da plataforma foi realizada por meio de testes de usabilidade, cujos resultados indicaram sua eficácia em melhorar os processos de gerenciamento e consulta de livros. O sistema contribui para a modernização e otimização das bibliotecas, promovendo uma gestão mais eficiente e uma experiência aprimorada para os usuários.

Palavras-chave: gerenciamento de livros; automatização; reservas; banco de dados; plataforma web.

ABSTRACT

This paper presents the development of an intuitive platform for managing books in libraries, integrating features such as registration, searching, reserving and monitoring copies. The system was designed to serve both administrators, who are responsible for managing and maintaining the collection, and users, who can make queries and reservations efficiently. It was implemented using technologies such as Flask, a relational database and integration with the Google Books API to obtain additional information about the books, such as covers and descriptions. In addition, the system prioritizes usability, incorporating responsive design principles to ensure accessibility on different devices. Additional functionalities, such as uploading images and registering multiple categories per book, have been implemented to meet the specific demands of the target audience. The project also addresses security issues, such as user authentication, permission control and password encryption. The platform was validated through usability tests, the results of which indicated its effectiveness in improving book management and consultation processes. The system contributes to the modernization and optimization of libraries, promoting more efficient management and an improved user experience.

Keywords: book management; automation; reservations; database; web platform.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript and XML (JavaScript e XML assíncronos.)
API	Application Programming Interface (Interface de Programação de Aplicativos)
CSS	Cascading Style Sheets (Folhas de Estilo em Cascata)
HTML	Hypertext Markup Language (Linguagem de Marcação de Hipertexto)
HTTP	Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)
JSON	JavaScript Object Notation (Notação de objeto JavaScript)
ISBN	International Standard Book Number (Padrão Internacional de Numeração de Livro)
IP	Internet Protocol (Protocolo de Rede)
SQL	Structured Query Language (Linguagem de Consulta Estruturada)
URL	Uniform Resource Locator (Localizador Uniforme de Recursos)
VCS	Version Control System (Sistema de Controle Versão)

SUMÁRIO

1 INTRODUÇÃO

Uma biblioteca desempenha um papel crucial na propagação do saber e no estímulo à educação. Historicamente, ela tem servido como um local para a conservação e o acesso a informações em várias formas, incluindo livros, documentos e outros materiais. Mais do que o armazenamento de livros, a biblioteca é um espaço de aprendizado e partilha de conhecimentos, onde alunos, pesquisadores e a comunidade em geral encontram apoio para suas atividades acadêmicas e intelectuais. Sua relevância transcende a disponibilização de recursos informativos, destacando-se na formação de pessoas críticas, no estímulo à leitura e no avanço da investigação científica.

Com o avanço tecnológico, as bibliotecas enfrentam o desafio de se ajustar às novas necessidades da sociedade. A digitalização tem transformado o ambiente acadêmico, oferecendo novas maneiras de interação com o acervo e os usuários. Essa transformação traz diversas vantagens, como o acesso mais ágil às informações, a organização aprimorada dos acervos e a simplificação das tarefas cotidianas dos bibliotecários. Sistemas digitais possibilitam que estudantes realizem pesquisas, reservas e consultas de forma remota, eliminando a necessidade de comparecimento presencial à biblioteca e otimizando o processo de aprendizado.

Por outro lado, muitas bibliotecas ainda enfrentam dificuldades decorrentes da dependência de processos manuais. O controle de empréstimos, devoluções e reservas, quando realizado manualmente, demanda um grande esforço operacional dos bibliotecários, causando filas, atrasos e insatisfação dos usuários. Essa situação reflete um sistema ineficiente que, além de frustrar alunos e funcionários, compromete a experiência de uso. À medida que as expectativas por serviços mais rápidos e dinâmicos aumentam, cresce também a necessidade de modernização e automatização.

Nesse contexto, surge o problema central deste trabalho, como melhorar a eficiência e a experiência do usuário no uso das bibliotecas por meio da implementação de um sistema digital integrado?

A hipótese deste estudo é que a adoção de uma plataforma digital que permita a automatização de tarefas e a gestão integrada do acervo, combinada com

funcionalidades acessíveis aos usuários, resultará em uma biblioteca mais eficiente e funcional, atendendo às demandas de estudantes e bibliotecários.

A implementação de um sistema digital busca oferecer autonomia aos alunos para acessar informações e recursos bibliográficos, enquanto otimiza as atividades rotineiras dos bibliotecários. Com a digitalização, processos como consulta, empréstimo e devolução podem se tornar mais rápidos e eficientes, beneficiando toda a comunidade acadêmica e promovendo um ambiente mais alinhado às exigências contemporâneas.

1.1 Objetivo geral

O objetivo deste trabalho é desenvolver um sistema de biblioteca virtual, com foco em otimizar o atendimento aos usuários e facilitar a gestão interna do acervo.

1.2 Objetivos específicos

- Entrevistar as funcionárias do setor da biblioteca para identificar as demandas e necessidades;
- Levantar/Pesquisar bibliografia sobre conceitos a serem abordados no trabalho, tais como biblioteca, linguagens de programação etc.;
- Identificar quais linguagens serão necessárias para o desenvolvimento do sistema;
- Desenvolver o sistema com base nas demandas e necessidades levantadas;
- Realizar teste de usabilidade com o objetivo de investigar questões que envolvem navegação e entendimento da interface.

1.3 Justificativa

A modernização do sistema de bibliotecas tradicionais é essencial diante dos desafios enfrentados na gestão e na experiência dos usuários. Processos manuais ou com pouca digitalização, como empréstimos, devoluções e reservas, ainda predominam em muitas instituições, comprometendo a eficiência. Essa lentidão resulta em filas, erros nos registros e frustrações para os usuários. Além disso, a ausência de automação restringe a autonomia dos frequentadores, que precisam se

deslocar fisicamente até a biblioteca para verificar a disponibilidade de livros ou renovar prazos, limitando a conveniência, assim como a catalogação manual, que dificulta a organização do acervo e o planejamento de reposições.

A implementação de um sistema digital é relevante tanto para a otimização dos serviços quanto para a experiência dos usuários. Funcionalidades como categorização eficiente e acesso remoto a informações podem transformar o modelo atual, proporcionando mais agilidade e organização. No contexto acadêmico, essa modernização atende à crescente demanda por informações rápidas e acessíveis, além de contribuir para o aprimoramento teórico da automação bibliotecária. Portanto, espera-se que essa inovação torne as bibliotecas mais atrativas e acessíveis, beneficiando tanto estudantes quanto profissionais da área. Este trabalho visa não apenas modernizar os processos, mas também fortalecer a relevância das bibliotecas em um mundo cada vez mais digital.

2 DESENVOLVIMENTO

2.1 Escolha do tema

A escolha deste tema surgiu a partir da observação cuidadosa de diferentes setores do Senai e da identificação dos desafios enfrentados em seus funcionamentos diários. Percebemos que questões como a lentidão em processos manuais, a falta de automação e a dificuldade de gestão de acervo são problemas recorrentes, impactando tanto os funcionários da biblioteca quanto os usuários. Essas limitações não apenas comprometem a eficiência operacional, mas também reduzem a qualidade do serviço oferecido. Diante dessa realidade, optamos por desenvolver um sistema que pudesse oferecer soluções práticas e acessíveis para esses desafios.

2.2 Escolha do tema

Após definirmos o tema do nosso projeto sobre a biblioteca virtual, decidimos aprofundar nossa pesquisa por meio de entrevistas com profissionais da área de biblioteconomia. O objetivo dessas entrevistas foi entender, de maneira mais detalhada, as necessidades e dificuldades reais enfrentadas no dia a dia pelas funcionárias do setor, a fim de direcionar o desenvolvimento de uma solução que realmente atendesse suas demandas.

Formulamos perguntas estratégicas, como os principais desafios no gerenciamento de livros, especialmente em relação à devolução, que frequentemente gera problemas, com muitos usuários não devolvendo os livros no prazo. Também perguntamos sobre a reserva de livros, investigando se os usuários conseguem verificar facilmente a disponibilidade de um livro e como isso poderia ser aprimorado. Outro ponto importante abordado nas entrevistas foi o gerenciamento do estado físico dos livros, como a verificação de danos e a necessidade de restaurações. Essas entrevistas também focaram nas funcionalidades desejadas para o sistema de reservas. Exploramos como poderia ser o processo de adicionar e editar informações no acervo, além do acompanhamento das reservas efetuadas pelos usuários.

2.3 Requisitos

Os requisitos são elementos essenciais para que uma solução se torne funcional, segura e eficiente, atendendo ao problema que se propõe resolver.

Sem requisitos como conectividade, sistema operacional, plataforma e interface, por exemplo, é inviável desenvolver um software que realmente funcione e atenda às necessidades dos usuários.

Em resumo, os requisitos funcionais e não funcionais desempenham um papel fundamental na criação e na experiência de uso de qualquer sistema. Eles garantem que o software seja útil, acessível e eficiente para quem irá utilizá-lo.

2.3.1 Requisitos funcionais

Os requisitos funcionais são responsáveis por definir o que um sistema precisa fazer para atender às necessidades dos usuários, especificando funcionalidades e comportamentos essenciais. Eles garantem que o sistema cumpra suas funções principais de maneira eficiente e eficaz, resolvendo diretamente os problemas enfrentados pelos usuários.

2.3.2 Requisitos Não Funcionais

Os requisitos não funcionais definem como o sistema deve operar, com foco em aspectos como qualidade, segurança, desempenho, usabilidade e escalabilidade. Diferentemente dos requisitos funcionais, que especificam o "o que" o sistema faz, os requisitos não funcionais se concentram no "como". Eles são fundamentais para garantir que o sistema funcione de forma eficiente e atenda aos padrões esperados de qualidade.

2.3.3 Requisitos funcionais criados

2.3.3.1 RF001 – Cadastrar usuário

O administrador deve cadastrar um novo usuário na plataforma, fornecendo informações como nome completo, CPF, e-mail, telefone e senha. O cadastro é inserido no banco de dados, e o sistema exibe uma mensagem de confirmação. Após o cadastro, o administrador pode cadastrar outro usuário ou navegar para outras funcionalidades do sistema. É necessário que o e-mail seja único no sistema,

e apenas administradores podem realizar o cadastro de novos usuários, garantindo que clientes não possam acessar essa funcionalidade. Este requisito é essencial para garantir a gestão de acesso e o controle de permissões dentro do sistema.

Figura 1 – Página de cadastro de usuário

Fonte: Jardim dos livros (2024)

2.3.3.2 RF002 – Cadastrar funcionário

O administrador deve cadastrar um novo funcionário no sistema, inserindo informações como nome completo, CPF, e-mail, telefone e senha. Após o preenchimento do formulário, o sistema salva os dados no banco de dados e confirma o cadastro com uma mensagem. Esse requisito é essencial para garantir que a equipe que utiliza o sistema tenha seus acessos controlados, promovendo a segurança e a organização no ambiente de trabalho.

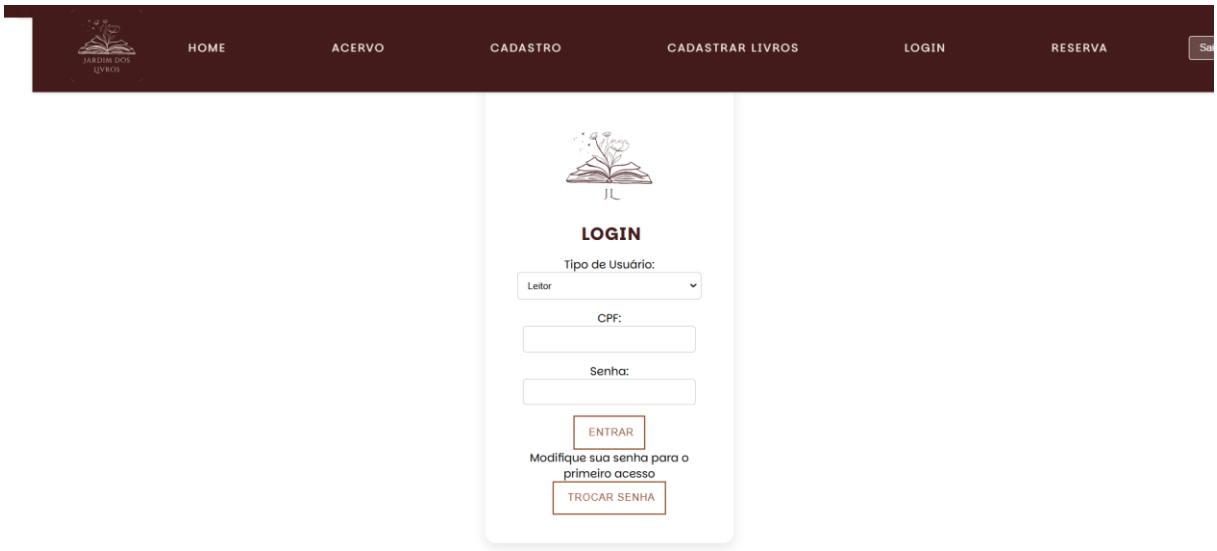
Figura 2 – Página de cadastro de funcionário

Fonte: Jardim dos livros (2024)

2.3.3.3 RF003 – Login de usuário

O sistema permite que o usuário faça login de forma simples e segura. Para isso, ele precisa fornecer seu CPF e senha cadastrados. Uma vez autenticado, o usuário será direcionado para a área principal da plataforma, onde poderá acessar as funcionalidades de acordo com suas permissões. Caso o e-mail ou a senha estejam incorretos, o sistema informará o erro e permitirá que o usuário tente novamente. Esse requisito é fundamental para garantir que apenas usuários autorizados tenham acesso ao sistema, proporcionando segurança e personalização da experiência de acordo com o perfil de cada usuário.

Figura 3 – Página de login de usuário

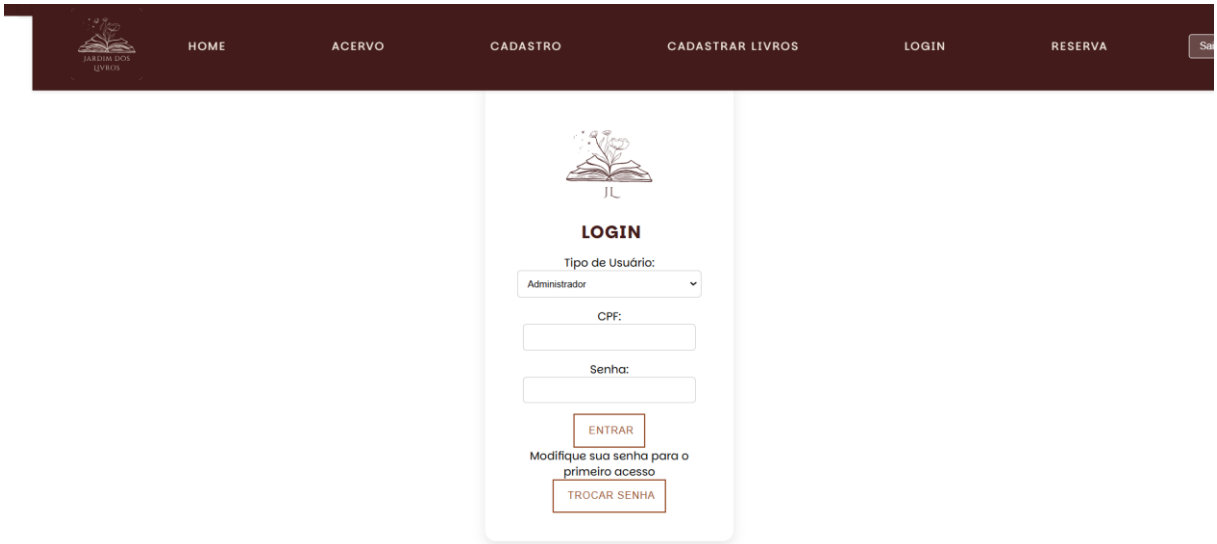


Fonte: Jardim dos livros (2024)

2.3.3.4 RF004 – Login de funcionário

O sistema permite que o funcionário faça login de maneira rápida e segura, utilizando seu CPF e senha cadastrados. Após a autenticação, o funcionário será direcionado para a página inicial do sistema, onde terá acesso às funcionalidades adequadas ao seu perfil. Se as credenciais estiverem incorretas, o sistema exibirá uma mensagem de erro, dando ao funcionário a chance de tentar novamente. Este requisito garante que o acesso ao sistema seja controlado, permitindo que apenas funcionários autorizados executem suas atividades de forma segura.

Figura 4 – Página de login de funcionário



Fonte: Jardim dos livros (2024)

2.3.3.5 RF005 – Cadastro de livro

O administrador tem a possibilidade de cadastrar novos livros no sistema de forma prática e eficiente. Para isso, ele pode inserir apenas o ISBN do livro, e uma API será acionada para trazer automaticamente informações relevantes, como título, autor, categoria, data de publicação e uma imagem de capa. Após o cadastro, as informações do livro são armazenadas no banco de dados e ficam disponíveis para consulta e reserva pelos usuários. O sistema exibirá uma mensagem de confirmação ao administrador quando o livro for cadastrado com sucesso, e, caso o livro já esteja no sistema, uma mensagem informará que o cadastro já existe e exibe uma mensagem para caso o administrador queira cadastrar o mesmo livro, mas como um exemplar diferente. Esse requisito é essencial para garantir que o acervo de livros seja constantemente atualizado e que os usuários tenham acesso rápido e fácil às novas adições na plataforma.

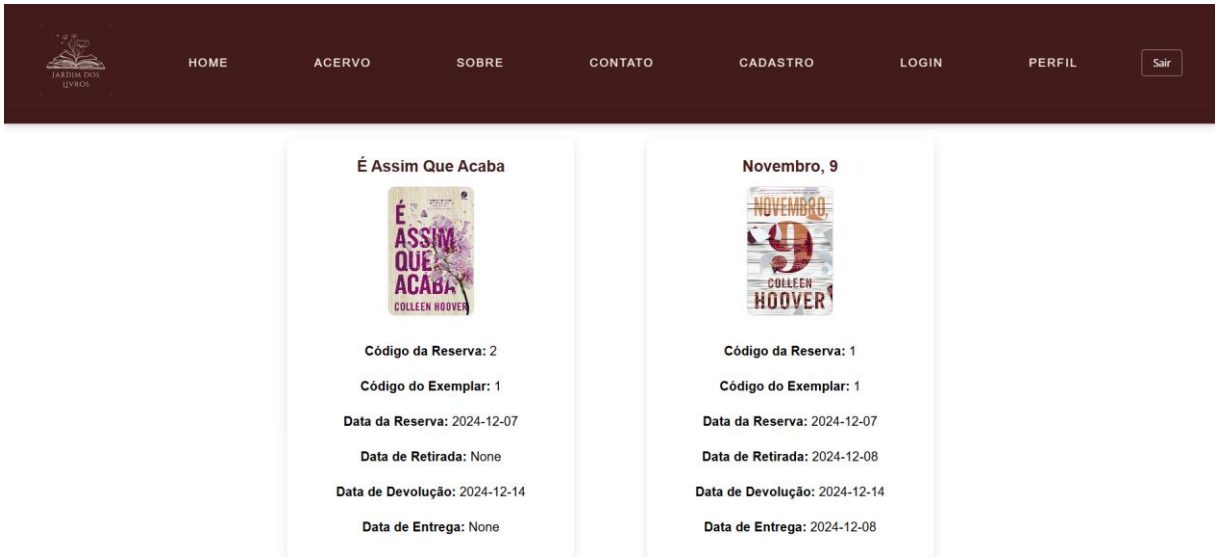
Figura 5 – Página de cadastro de livro

Fonte: Jardim dos livros (2024)

2.3.3.6 RF006 – Reserva de livro

O sistema permite que um usuário reserve um livro que não está disponível no momento, colocando-se em uma lista de espera para obtê-lo assim que ele retornar ao acervo. Quando um usuário encontrar um livro desejado que está emprestado, ele poderá clicar na opção de "Reservar", o que o adiciona automaticamente à tabela disponível para o funcionário que contém as informações das reservas. Esse requisito é importante para garantir que os usuários tenham uma maneira organizada de solicitar livros que estão temporariamente indisponíveis, promovendo um melhor gerenciamento do acervo.

Figura 6 – Página de reservas realizadas pelo usuário



Fonte: Jardim dos livros (2024)

2.3.3.7 RF007 – Busca de livro

O sistema permite que os usuários busquem livros de forma fácil e intuitiva, utilizando filtros como título ou autor. Após realizar a busca, o sistema exibirá uma lista de livros que correspondem aos critérios informados, incluindo informações como título, autor e a disponibilidade do livro. O usuário poderá visualizar detalhes adicionais de cada livro, como descrição e imagem de capa. Essa funcionalidade é essencial para que o usuário encontre rapidamente os livros de seu interesse e explore o acervo de forma eficiente.

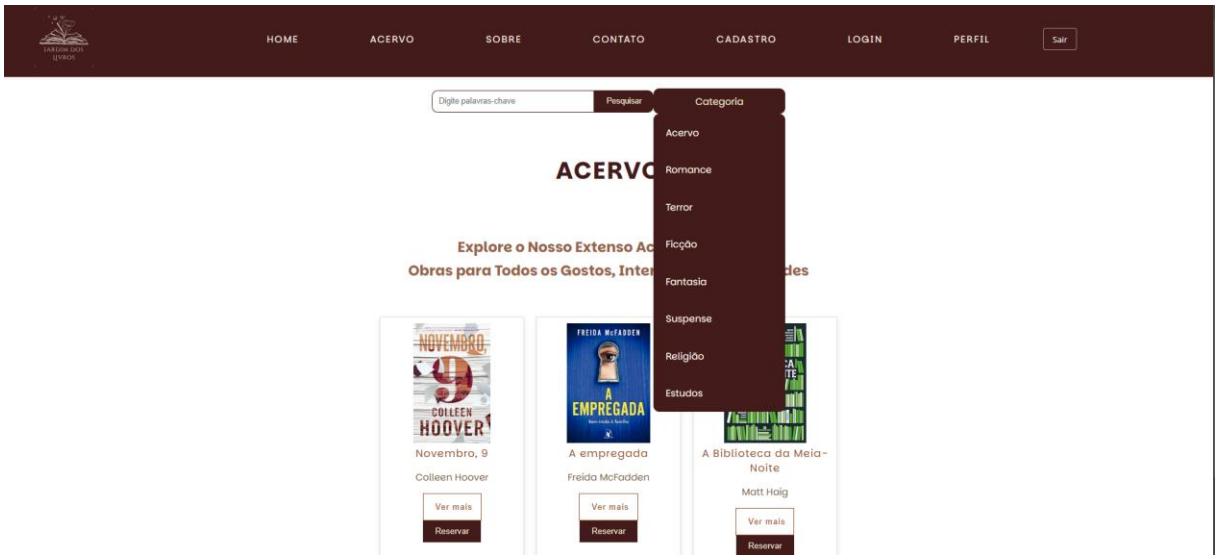
Figura 7 – Página de acervo correspondente a pesquisa pelo título

Fonte: Jardim dos livros (2024)

2.3.3.8 RF008 – Categoria do livro

O sistema oferece uma maneira simples e intuitiva para os usuários explorarem os livros por categorias. Ao acessar a funcionalidade de pesquisa por categoria, o usuário pode selecionar uma categoria que corresponda ao tipo de livro desejado. O sistema, então, exibirá uma lista de livros dentro da categoria escolhida, fornecendo informações como título, autor e a disponibilidade do item. Além disso, o usuário terá acesso a detalhes adicionais de cada livro, como descrição e imagem de capa. Essa funcionalidade é fundamental para que o usuário navegue facilmente pelo acervo, encontre livros de seu interesse de maneira rápida e eficiente, e explore as diferentes opções disponíveis em cada categoria.

Figura 8 – Página de acervo correspondente a categoria



Fonte: Jardim dos livros (2024)

2.3.3.9 RF009 – Comentários sobre o livro

A funcionalidade de comentários permite que os usuários compartilhem suas opiniões sobre os livros que leram, proporcionando uma interação mais rica com o acervo. Ao visualizar os detalhes de um livro, o cliente pode acessar a seção de comentários, onde poderá deixar sua avaliação e feedback sobre o conteúdo, a escrita e outros aspectos do livro. Além disso, é possível ler as avaliações de outros usuários, o que auxilia na escolha do próximo livro a ser lido. Essa funcionalidade oferece uma forma de engajamento entre os leitores, ajudando-os a tomar decisões informadas e a enriquecer a experiência de leitura de toda a comunidade da biblioteca.

Figura 9 – Página de comentário

Fonte: Jardim dos livros (2024)

2.3.3.10 RF010 – Mapa

Quando o usuário deseja se cadastrar sozinho, o sistema exibe uma mensagem informando que é necessário se dirigir até o local correto. Ele precisará ajustar a localização para o endereço desejado, garantindo que o registro seja vinculado à unidade correta da biblioteca. Esse processo é essencial para assegurar que o cadastro seja feito de forma precisa e que o usuário tenha acesso à biblioteca mais próxima ou àquela em que deseja realizar as atividades. O mapa proporciona uma experiência mais intuitiva e ajuda a evitar erros no cadastro, tornando o processo mais eficiente e fácil de completar.

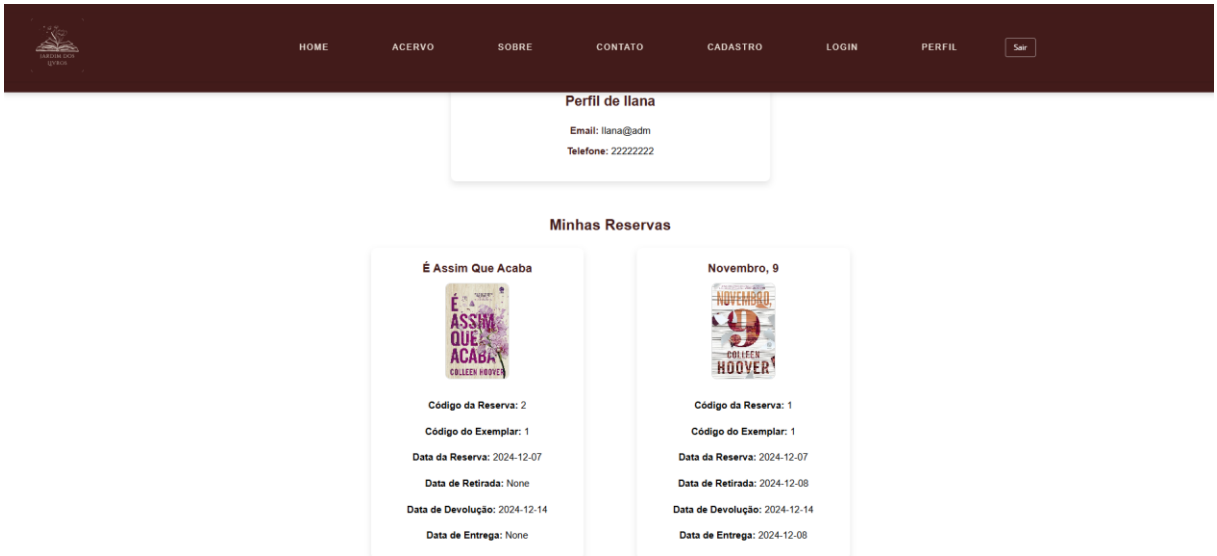
Figura 10 – Página de mensagem de local do cadastro

Fonte: Jardim dos livros (2024)

2.3.3.11 RF011 – Perfil

A página de perfil do usuário oferece uma visão completa e detalhada das suas reservas. Nela, o usuário pode visualizar seu nome, e-mail e telefone, além de informações sobre os livros que reservou. Para cada reserva, são exibidos o título da imagem do livro reservado, o código da reserva e o código do exemplar correspondente. Também são apresentadas as datas relevantes: a data da reserva, a data prevista para a retirada do livro, a data de devolução e a data de entrega, permitindo que o usuário acompanhe o status de sua reserva de forma clara e organizada. Essa funcionalidade proporciona uma maneira prática de gerenciar as reservas, ajudando o usuário a se manter atualizado e a não perder prazos importantes.

Figura 11- Página de perfil



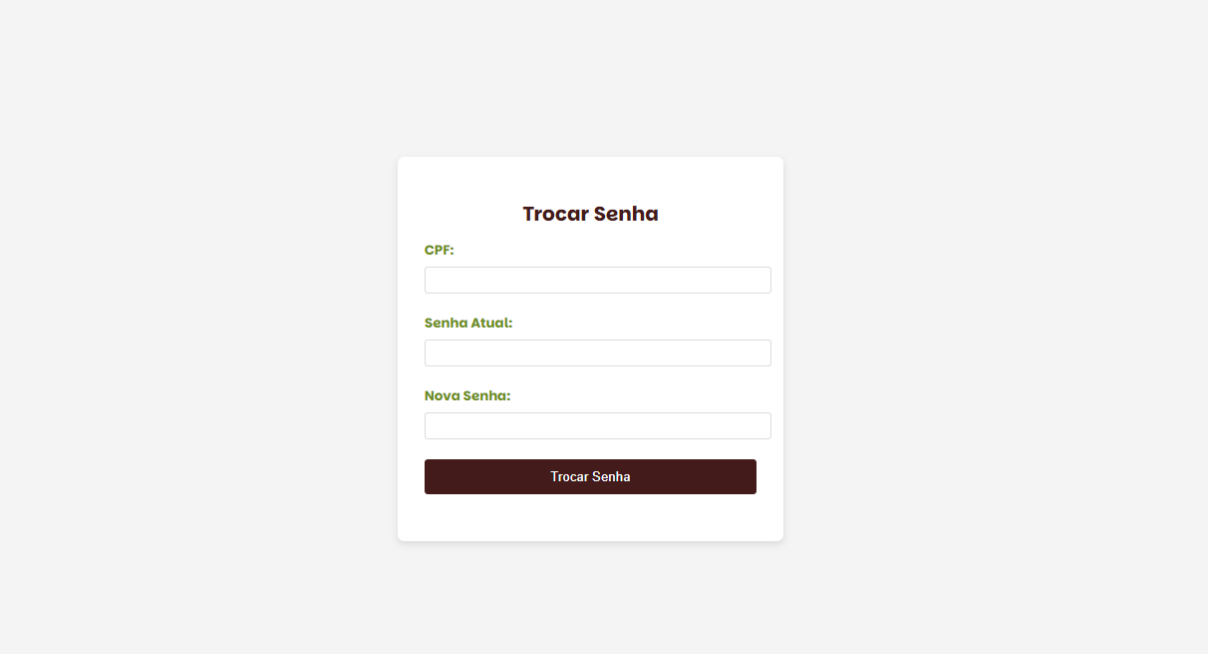
Fonte: Jardim dos livros (2024)

2.3.3.12 RF012 – Trocar senha

A página de troca de senha oferece uma interface simples e segura para os usuários atualizarem suas credenciais. Para realizar a alteração, o usuário deve informar seu CPF, a senha atual e a nova senha desejada. Esse procedimento foi projetado para permitir que os usuários substituam a senha padrão fornecida pela biblioteca por

uma senha personalizada, aumentando a segurança de suas contas. A validação dos dados inseridos garante que somente usuários autorizados consigam alterar suas senhas. Essa funcionalidade foi desenvolvida com foco em praticidade e proteção, ajudando os usuários a manterem suas contas seguras de maneira eficiente e confiável.

Figura 12- Página de troca de senha



2.3.3.13 RF013 – Sair da conta

O botão de "Sair" oferece ao usuário uma maneira segura e simples de encerrar sua sessão. Ao clicar nesse botão, o sistema solicita uma confirmação para garantir que a ação seja intencional. Após a confirmação, o usuário é desconectado e redirecionado automaticamente para a página inicial. Esse processo assegura que o usuário possa sair da plataforma de maneira tranquila e sem riscos de acessar informações pessoais após o término da sessão. A funcionalidade oferece mais segurança e praticidade, proporcionando uma experiência de uso mais fluida e confiável.

Figura 13 – Página inicial após sair da conta

Fonte: Jardim dos livros (2024)

2.3.4 Requisitos não funcionais criados

2.3.4.1 RNF001 – Criptografia de senha

O sistema assegura que todas as senhas dos usuários sejam armazenadas de maneira criptografada no banco de dados, garantindo a proteção das informações pessoais. Sempre que um novo usuário é cadastrado, a senha fornecida é automaticamente criptografada antes de ser salva. Essa prática é fundamental para garantir que, mesmo em caso de uma eventual violação de dados, as senhas permaneçam protegidas e inacessíveis a terceiros.

2.3.4.2 RNF002 – Facilidade de uso

O sistema oferece uma experiência intuitiva e agradável, adaptada a usuários de todos os níveis de familiaridade com tecnologia. Com uma interface clara, menus organizados e funcionalidades de fácil acesso, os usuários conseguem navegar e realizar suas tarefas de maneira rápida e descomplicada. Essa abordagem é fundamental para assegurar que todos possam utilizar a plataforma de forma confortável e eficiente.

2.4. Regras de negócios

As regras de negócio traduzem necessidades específicas do negócio, como validações, restrições e outros critérios, em regras lógicas. Esse alinhamento

permite que as áreas de desenvolvimento, produto e negócio trabalhem em sintonia, guiando-se por essas necessidades e aplicando-as de forma clara. Assim, o desenvolvimento e o crescimento do produto ocorrem de maneira mais eficiente (Alura, 2024).

2.4.1 Regras de negócios criadas

2.4.1.1. REG001 – Limite de reservas

Cada usuário pode ter no máximo 2 livros reservados simultaneamente. Essa regra tem o objetivo de garantir que todos os usuários tenham acesso justo aos recursos da biblioteca, evitando o acúmulo excessivo de livros por uma única pessoa e promovendo a circulação do acervo. Se o usuário já tiver 2 livros em suas reservas, o sistema bloqueará a realização de novas reservas e exibirá uma mensagem informando que o limite foi alcançado. Caso o usuário tenha menos de 2 livros reservados, ele poderá continuar a realizar novos empréstimos até atingir o limite máximo. Essa regra de negócio é essencial para a organização do acervo e para manter a plataforma funcionando de maneira equilibrada, proporcionando uma experiência justa para todos os usuários.

2.4.1.2. REG002 – Local de cadastro

O cadastro de novos usuários só pode ser realizado presencialmente no local da biblioteca. Essa regra tem como objetivo garantir a segurança e a veracidade das informações fornecidas, evitando fraudes ou cadastros indevidos. Caso o usuário tente realizar o cadastro remotamente, o sistema exibirá uma mensagem informando que o procedimento deve ser realizado presencialmente. Essa regra de negócio é essencial para preservar a integridade do sistema e manter a confiança nos serviços oferecidos, proporcionando maior segurança para todos os usuários.

2.5. Wireframe

Um wireframe de site, ou website wireframe, é um esboço que define as bases da arquitetura de um site. Amplamente utilizado por agências e desenvolvedores, esse

"rascunho" oferece uma visão preliminar do projeto, requer poucos recursos e pode ser facilmente ajustado conforme necessário (Rock Content, 2024).

Figura 14 – Wireframes criados para nosso sistema

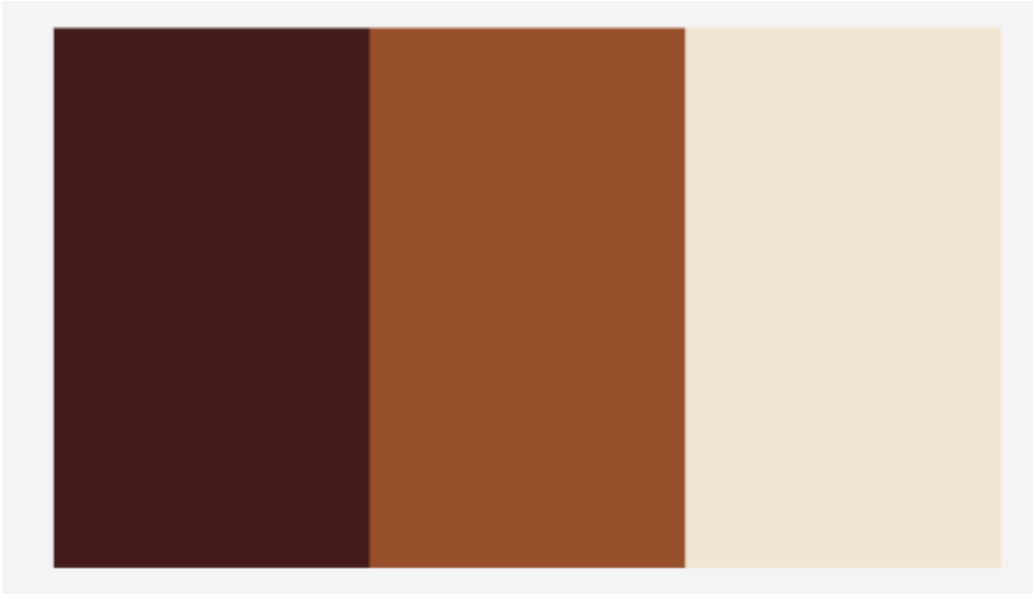
Fonte: Jardim dos livros (2024)

2.6. Estudo de cores

O estudo das cores busca compreender como elas influenciam o comportamento, a percepção e as experiências das pessoas.

Por meio desse estudo, constatou-se que a escolha estratégica de cores para representar uma marca, produto ou serviço pode transmitir sensações que impactam significativamente a tomada de decisão dos usuários e proporcionam uma experiência mais envolvente e positiva. Assim, as cores tornam-se um elemento essencial para o desenvolvimento e o sucesso de estratégias de negócios.

Figura 15 – Paleta de cores utilizadas em nosso sistema



Fonte: Jardim dos livros (2024)

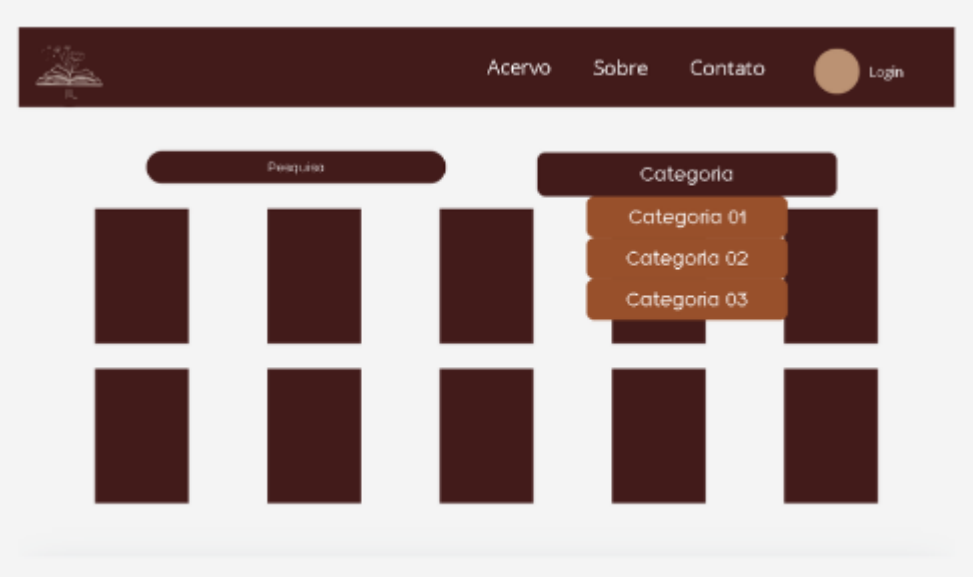
2.7. Protótipo

O protótipo do sistema é uma versão inicial, simplificada e funcional, desenvolvida com o objetivo de visualizar e testar as principais funcionalidades e o design do projeto antes de sua implementação completa. Ele permite que desenvolvedores e usuários avaliem e validem conceitos, identifiquem falhas ou inconsistências, e proponham melhorias de maneira ágil e eficaz.

A criação do protótipo é fundamental porque reduz os riscos associados ao desenvolvimento, economizando tempo e recursos. Ele facilita a comunicação entre as partes envolvidas, tornando mais claro o que será entregue no produto final. Além disso, o protótipo ajuda a garantir que o sistema atenda às necessidades dos usuários e esteja alinhado com os objetivos do projeto.

No caso deste sistema, o protótipo foi essencial para ajustar o fluxo de navegação, testar a experiência do usuário e verificar se os requisitos de negócio estavam sendo atendidos, promovendo um desenvolvimento mais seguro e eficiente.

Figura 16 – Protótipo inicial do nosso sistema

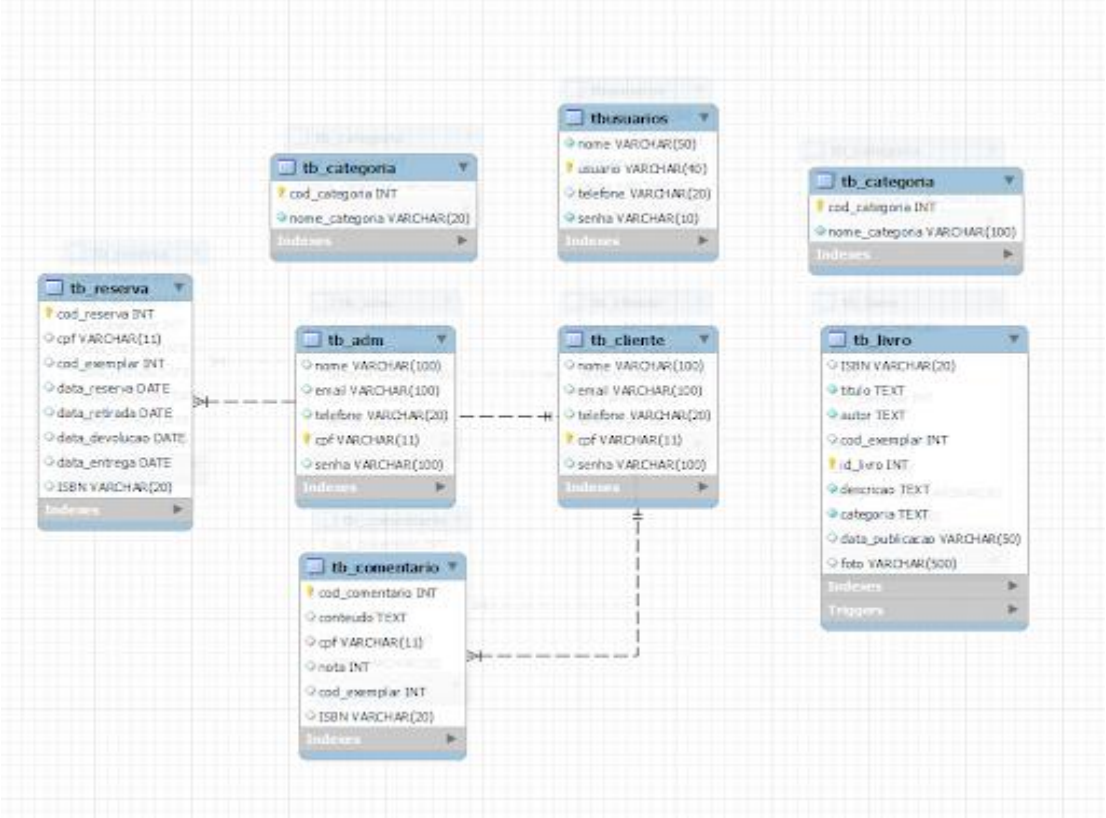


Fonte: Jardim dos livros (2024)

2.8. Banco de dados

Um banco de dados é uma forma organizada de guardar informações, ou seja, dados, de maneira estruturada e eletrônica, geralmente em um computador. Ele é gerenciado por um software chamado sistema de gerenciamento de banco de dados (DBMS). Esse sistema, junto com os dados e os aplicativos que trabalham com ele, forma o que chamamos de sistema de banco de dados, ou simplesmente banco de dados.

Figura 17 – Tabelas do banco de dados do nosso sistema



Fonte: Jardim dos livros (2024)

2.8.1. MySQL

MySQL é um dos sistemas de gerenciamento de bancos de dados relacionais mais populares e amplamente utilizados no mundo. Ele permite realizar tarefas como buscar, adicionar, atualizar ou excluir informações de maneira eficiente, além de oferecer ferramentas para gerenciar o acesso e a segurança dos dados.

Optamos por utilizar o MySQL devido à sua compatibilidade com diversos sistemas e linguagens de programação. Ele é projetado para lidar com grandes volumes de dados, permitindo consultas rápidas e desempenho confiável, mesmo em aplicações de grande escala. Além disso, o MySQL é fácil de configurar e oferece suporte a SQL, uma linguagem declarativa que torna o processo de interação com os dados mais direto e intuitivo, pois você foca no que deseja fazer, enquanto o sistema cuida do como.

2.8.2. Estrutura do banco de dados

2.8.2.1. CREATE DATABASE Biblio

- CREATE DATABASE: Comando SQL para criar o banco de dados.
- Biblio: Nome do banco de dados.

2.8.2.2. USE

Após a criação do banco, é necessário usar o comando USE Biblio; para especificar que as próximas operações (como criação de tabelas ou inserção de dados) serão executadas nesse banco.

2.8.2.3. Criação da tabela “tb_adm”

- nome (VARCHAR(100): Armazena o nome do administrador, com até 100 caracteres.
- email (VARCHAR(100): Armazena o email do administrador, com até 100 caracteres.
- telefone (VARCHAR(20): Armazena o telefone, com até 20 caracteres, permitindo incluir símbolos como traços ou parênteses.
- cpf (VARCHAR(11): Armazena o CPF do administrador, com exatamente 11 caracteres. Esse campo é definido como PRIMARY KEY, ou seja, será único para cada administrador e não pode ser duplicado.
- senha (VARCHAR(100): Armazena a senha do administrador, com até 100 caracteres, permitindo a utilização de senhas criptografadas ou codificadas.

2.8.2.5. Criação da tabela “tb_livro”

- ISBN (VARCHAR(20): Identificador único dos livros, com até 20 caracteres.
- titulo (TEXT NOT NULL): Título do livro, obrigatório.
- autor (TEXT NOT NULL): Nome do autor, obrigatório.

- cod_exemplar (INT): Código de cada exemplar, para identificar cópias do mesmo livro.
- id_livro (INT AUTO_INCREMENT PRIMARY KEY): Identificador único de cada livro, gerado automaticamente.
- descricao (TEXT NOT NULL): Descrição detalhada do livro, obrigatória.
- categoria (TEXT NOT NULL): Categoria do livro, obrigatória.
- data_publicacao (VARCHAR(50): Data de publicação do livro.
- foto (VARCHAR(500): Caminho para a foto do livro.

2.8.2.6. Criação da tabela “tb_cliente”

- nome(VARCHAR(100): Armazena o nome do cliente, com até 100 caracteres.
- email(VARCHAR(100): Armazena o e-mail do cliente, com até 100 caracteres.
- telefone(VARCHAR(20): Armazena o telefone do cliente, com até 20 caracteres.
- cpf(VARCHAR(11): Armazena o CPF do cliente, com exatamente 11 caracteres. Esse campo é definido como PRIMARY KEY, sendo único e obrigatório para cada cliente.
- senha(VARCHAR(100): Armazena a senha do cliente, com até 100 caracteres. Permite a utilização de senhas criptografadas para maior segurança no sistema.

2.8.2.7. Criação da tabela “tb_categoria”

- cod_categoria(INT AUTO_INCREMENT PRIMARY KEY): Armazena o código da categoria. Esse campo é um número inteiro e é definido como AUTO_INCREMENT, ou seja, o valor é gerado automaticamente para cada nova categoria inserida, garantindo que cada categoria tenha um código único.
- nome_categoria(VARCHAR(100) NOT NULL): Armazena o nome da categoria, com até 100 caracteres. O campo é definido como NOT NULL, o que significa que não pode ficar em branco, garantindo que todas as categorias tenham um nome válido.

2.8.2.8. Criação da tabela “tb_reserva”

- cpf(VARCHAR(11): Armazena o CPF do cliente que fez a reserva. Este campo é definido como chave estrangeira (FOREIGN KEY) e faz referência ao campo cpf da tabela tb_cliente, garantindo que apenas usuários cadastrados possam fazer reservas.
- cod_exemplar(INT): Armazena o código do exemplar reservado. Esse código identifica um exemplar específico dentro do acervo.
- data_reserva(DATE): Armazena a data em que a reserva foi realizada.
- data_retirada(DATE): Armazena a data em que o cliente retirou o livro na biblioteca.
- data_devolucao(DATE): Armazena a data prevista para a devolução do livro.
- data_entrega(DATE): Armazena a data em que o cliente efetivamente devolveu o livro.

2.8.2.9. Criação da tabela “tb_comentario”

- cod_comentario(INT AUTO_INCREMENT PRIMARY KEY): É o código do comentário. Este campo é gerado automaticamente e serve como chave primária (PRIMARY KEY), garantindo que cada comentário tenha um identificador único.
- conteudo(TEXT): Armazena o texto do comentário que o cliente escreveu sobre o exemplar. O campo permite conteúdo extenso, sem limite rígido de caracteres.
- cpf(VARCHAR(11): Armazena o CPF do cliente que fez o comentário. Esse campo é uma chave estrangeira (FOREIGN KEY), referenciando o campo cpf da tabela tb_cliente, garantindo que apenas usuários cadastrados possam comentar.
- nota(INT): Armazena a nota dada pelo cliente ao livro, variando de 0 a 5.

- cod_exemplar(INT): Identifica o código do exemplar sobre o qual o comentário foi feito.

2.8.2.10. Explicação do Trigger

“before_insert_tb_livro”

O trigger before_insert_tb_livro é um mecanismo que é acionado automaticamente antes da inserção de um novo registro na tabela tb_livro. A finalidade desse trigger é garantir que o código do exemplar (cod_exemplar) seja gerado de forma única para cada livro que possui o mesmo ISBN.

Detalhes do Trigger:

- DELIMITER $$:
    - Modifica o delimitador padrão do MySQL (que é ;) para $$. Isso permite que o corpo do trigger, que pode conter múltiplas instruções SQL, seja interpretado corretamente sem confundir o delimitador de finalização.
- CREATE TRIGGER before_insert_tb_livro:
    - Inicia a definição do trigger chamado before_insert_tb_livro.
- BEFORE INSERT ON tb_livro:
    - Especifica que o trigger deve ser executado antes da inserção de um novo registro na tabela tb_livro.
- FOR EACH ROW:
    - Indica que o trigger será executado para cada linha que está sendo inserida.
- BEGIN ... END:
    - Define o bloco de instruções que serão executadas quando o trigger for acionado.
- DECLARE max_cod_exemplar INT DEFAULT 0:
    - Declara uma variável chamada max_cod_exemplar que armazena o maior código de exemplar existente para o ISBN do livro sendo inserido, inicializando com 0.
- SELECT COALESCE(MAX(cod_exemplar), 0) INTO max_cod_exemplar:

    - Realiza uma consulta para encontrar o maior cod_exemplar já utilizado para o ISBN do novo livro (NEW.ISBN). A função COALESCE assegura que, se não houver registros, o valor padrão será 0.
- SET NEW.cod_exemplar = max_cod_exemplar + 1:
    - Define o valor do cod_exemplar do novo livro como o maior valor encontrado (max_cod_exemplar) acrescido de 1, garantindo que cada exemplar tenha um código único.
- DELIMITER ;:
    - Retorna o delimitador ao padrão (;) após a definição do trigger.

2.8.2.11. Criação do usuário

As instruções fornecidas criam um usuário no sistema de gerenciamento de banco de dados e concedem a ele permissões específicas.

2.8.2.11.1. CREATE USER 'Biblio'@'%' IDENTIFIED BY '12345'

- CREATE USER: Comando usado para criar um novo usuário no banco de dados.
- 'Biblio'@'%':
    - 'Biblio' é o nome do novo usuário.
    - '%' indica que o usuário pode se conectar a partir de qualquer host. Isso significa que o usuário pode acessar o banco de dados de qualquer endereço IP.
- IDENTIFIED BY '12345': Define a senha do usuário como 12345. Esta senha será necessária para o usuário se autenticar ao acessar o banco de dados.

2.8.2.11.2 GRANT ALL PRIVILEGES ON *.*TO 'Biblio'@'%' WITH GRANT OPTION

GRANT ALL PRIVILEGES: Concede todas as permissões disponíveis no sistema de gerenciamento de banco de dados ao usuário. Isso inclui permissões como SELECT, INSERT, UPDATE, DELETE, entre outras.

- O asterisco (*) antes do ponto (.) indica que as permissões se aplicam a todos os bancos de dados disponíveis no sistema.
- O asterisco (*) após o ponto (.) indica que as permissões se aplicam a todas as tabelas dentro desses bancos de dados.
- TO 'Biblio'@'%': Especifica que as permissões estão sendo concedidas ao usuário Biblio que pode se conectar a partir de qualquer host.
- WITH GRANT OPTION: Permite que o usuário Biblio conceda as permissões que recebeu a outros usuários. Isso é útil em cenários onde você quer permitir que um usuário gerencie permissões de outros.

2.8.2.11.3. FLUSH PRIVILEGES

O comando FLUSH PRIVILEGES é usado para recarregar as tabelas de privilégios no banco de dados. Isso é necessário para garantir que as alterações feitas na criação do usuário e nas permissões sejam reconhecidas imediatamente pelo servidor do banco de dados.

2.9 Git Hub

O GitHub é um serviço baseado em nuvem que hospeda um sistema de controle de versão (VCS) chamado Git. Ele permite que os desenvolvedores colaborem e façam mudanças em projetos compartilhados enquanto mantêm um registro detalhado do seu progresso. Nós o escolhemos para facilitar na dinâmica de programação entre o grupo.

2.9.1 Repositórios

Repositório, ou repo, é um diretório onde os arquivos do seu projeto ficam armazenados. Ele pode ficar em um depósito do GitHub ou em seu computador. Você pode armazenar códigos, imagens, áudios ou qualquer outra coisa relacionada ao projeto no diretório (HOSTINGER, 2024).

2.9.2. Branches (Ramificações)

Ao criar *branches*, ou ramificações, você gera versões diferentes de um repositório. Quando você modifica o projeto nas *branches* de recursos, um desenvolvedor pode

ver como isso vai afetar o projeto principal na hora que tudo for integrado. (HOSTINGER, 2024).

2.9.3. Commits e Pull Requests

Um commit é uma ação que registra as alterações feitas nos arquivos de um repositório. Cada commit cria um ponto de verificação no histórico do projeto, permitindo que você acompanhe as modificações realizadas ao longo do tempo. Com um commit, você pode incluir uma descrição explicativa sobre as mudanças feitas, facilitando a compreensão de quem está acompanhando o projeto. Enquanto um pull request (PR) é uma solicitação feita para integrar as alterações de uma branch (ramo) para a branch principal do repositório. Quando um colaborador faz modificações em uma branch separada, ele pode abrir um pull request para que outro desenvolvedor revise as alterações antes de serem incorporadas ao código principal. Isso é fundamental em projetos colaborativos, pois permite revisar, discutir e testar as mudanças antes de elas serem efetivamente aplicadas, garantindo que o código esteja de acordo com os padrões do projeto.

2.9.4. Issues e Projetos

Uma issue (ou "problema") no GitHub é uma maneira de registrar tarefas, erros ou discussões relacionadas ao projeto. As issues podem ser usadas para rastrear problemas no funconamento, sugerir novas funcionalidades, pedir ajuda ou qualquer outro item que precise de atenção. Os projetos no GitHub são uma maneira de organizar e gerenciar o fluxo de trabalho em torno das issues e outras tarefas do projeto. Um projeto é basicamente uma tabela ou quadro visual onde você pode agrupar, categorizar e mover as issues de acordo com o progresso.

3 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo o desenvolvimento de uma plataforma de gerenciamento de livros, proporcionando uma solução eficiente para a organização e controle do acervo de uma biblioteca. Durante o processo, realizamos testes de usabilidade para garantir que a experiência do usuário fosse simples e intuitiva, identificando pontos de melhoria e ajustando a interface para torná-la mais acessível. Os testes foram fundamentais para otimizar a navegação, garantir a clareza nas interações e promover uma experiência satisfatória para todos os tipos de usuários.

Além das habilidades técnicas desenvolvidas, o projeto também permitiu o aprimoramento de habilidades socioemocionais, como o trabalho em equipe, a comunicação eficaz e a gestão do tempo. Estes aspectos foram essenciais para superar desafios e garantir que o projeto fosse concluído com êxito.

Embora o sistema esteja totalmente funcional e tenha atendido às necessidades inicialmente propostas, existem oportunidades de aprimoramento para trabalhos futuros. Dentre as funcionalidades que gostaríamos de implementar, destacam-se: a adição de um alerta para notificar os usuários sobre livros pendentes após o prazo de devolução, uma seção de recomendações personalizadas com base nas preferências dos usuários e uma interface dinâmica que se atualiza automaticamente, destacando os livros mais reservados. Em resumo, apesar dos desafios enfrentados, o projeto foi concluído com sucesso, atingindo os objetivos propostos e proporcionando uma solução viável e eficiente para o gerenciamento de livros na biblioteca.

REFERÊNCIAS

ROCK CONTENT. O que é URL e como ela funciona na internet. Disponível em: https://rockcontent.com/br/blog/url/. Acesso em: 1 dez. 2024.

CASA DO DESENVOLVEDOR. Requisitos funcionais e não funcionais: qual a diferença e como aplicá-los em seu projeto. Disponível em: https://blog.casadodesenvolvedor.com.br/requisitos-funcionais-e-nao-funcionais/. Acesso em: 28 nov. 2024.

RED HAT. O que são APIs (interfaces de programação de aplicativos). Disponível em: https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces. Acesso em: 8 dez. 2024.

HARVE. O que é Backend? Guia completo para entender o desenvolvimento de sistemas. 2020. Disponível em: https://harve.com.br/blog/desenvolvimento-web/o-que-e-backend-guia-completo/. Acesso em: 3 dez. 2024.

CURSOS PM3. Psicologia das cores: Como as cores influenciam as emoções e o comportamento. Disponível em: https://www.cursospm3.com.br/blog/psicologia-das-cores/. Acesso em: 8 dez. 2024.

HOSTGATOR. O que é protótipo e qual a sua importância para o seu projeto. Disponível em: https://www.hostgator.com.br/blog/prototipo-o-que-e/. Acesso em: 11 nov. 2024.

ORACLE. O que é banco de dados? Disponível em: https://www.oracle.com/br/database/what-is-database/. Acesso em: 8 dez. 2024.

KINSTA. SQLite vs MySQL: Qual é a diferença e qual você deve usar? 2020. Disponível em: https://kinsta.com/pt/blog/sqlite-vs-mysql/. Acesso em: 8 dez. 2024.

HOSTINGER. O que é GitHub e como funciona: entenda a plataforma de hospedagem de código. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-github. Acesso em: 27 nov. 2024.

GITHUB DOCS. Committing changes to a pull request branch created from a fork. Disponível em: https://docs.github.com/pt/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/committing-changes-to-a-pull-request-branch-created-from-a-fork. Acesso em: 2 dez. 2024.

CBL Serviços. **O que é ISBN?** Disponível em: https://www.cblservicos.org.br/isbn/o-que-e-isbn/. Acesso em: 4 out. 2024.

Patel, Neil. **Design: o que é?** Disponível em: https://neilpatel.com/br/blog/design-o-que-e/. Acesso em: 20 set. 2024.

Universidade de Brasília (UnB). **MySQL Delete: como funciona?** Disponível em: https://sae.unb.br/cae/conteudo/unbfga/sbd/new_mysql_delete.html. Acesso em: 1 set. 2024.

TechTudo. **O que é Host e como funciona?** Disponível em: https://www.techtudo.com.br/guia/2023/12/o-que-e-host-e-como-funciona-edsoftwares.ghtml. Acesso em: 8 dez. 2024.

Significados. **Software: o que é?** Disponível em: https://www.significados.com.br/software/. Acesso em: 29 out. 2024.

RL System. **Select, Insert, Delete e Update no MySQL.** Disponível em: https://www.rlsystem.com.br/select-insert-delete-update-mysql. Acesso em: 1 nov. 2024.

GLOSSÁRIO

Banco de dados: Sistema organizado para armazenar, gerenciar e recuperar informações, permitindo a persistência de dados de forma eficiente e segura.

Branch: Uma cópia separada de um projeto dentro de um repositório no Git. Ela permite que desenvolvedores trabalhem em alterações sem interferir no código principal até que as modificações sejam aprovadas.

Commit: Ação de salvar alterações no repositório de um projeto, criando um registro que representa uma versão específica do código.

Conteiner/Container: Usado para descrever a área ou estrutura que agrupa outros elementos na interface de usuário.

Criptografia: Técnica de proteger informações por meio de algoritmos, garantindo que apenas pessoas autorizadas possam acessá-las.

Delete: Função usada para remover registros de uma tabela ou banco de dados, eliminando permanentemente as informações.

Design: Processo de criar soluções visuais e funcionais para resolver problemas, combinando técnica e estética para agregar valor ao produto ou serviço.

Feedback: Resposta ou reação dada a uma ação, geralmente usada para avaliar o desempenho ou fazer ajustes em um processo ou comportamento.

Flask: Ferramenta que permite aos desenvolvedores criarem aplicações web de forma rápida e eficiente.

Flexbox: Conceito que visa organizar os elementos de uma página HTML dentro de seus containers de forma dinâmica.

GET: Método HTTP usado para solicitar dados de um servidor, geralmente utilizado para recuperar informações sem modificar nada no servidor.
Host: Computador ou servidor conectado à rede que fornece recursos e serviços para sites, blogs e aplicativos, tornando-os acessíveis na internet.

Hover: Efeito visual ativado quando o usuário passa o mouse sobre um elemento da interface, geralmente utilizado para destacar elementos interativos.

Insert: Função responsável por adicionar novas linhas de dados em uma tabela de um banco de dados.

Issue: Item criado em plataformas como GitHub para rastrear problemas, melhorias ou tarefas pendentes dentro de um projeto, facilitando o gerenciamento de desenvolvimento.

Interfaces: Conjunto de elementos visuais e interativos que permitem a comunicação entre o usuário e o sistema, facilitando a navegação e o uso da aplicação.

JavaScript/Javascript: Linguagem de programação usada para criar interatividade e dinamicidade em páginas web, tornando-as mais interativas para os usuários.

jQuery: Biblioteca JavaScript que simplifica a manipulação de elementos HTML, eventos e animações em páginas web.

Layout: Disposição e organização dos elementos em uma página ou tela, determinando como os conteúdos e as informações serão visualmente apresentados.

Link: Elemento em uma página web que permite a navegação para outra página ou recurso, geralmente representado por texto ou imagem clicável.

Login: Processo de autenticação de um usuário em um sistema, geralmente envolvendo um nome de usuário e senha para acessar funcionalidades restritas.

Padding: Espaçamento interno de um elemento, criando uma área entre o conteúdo e a borda do elemento.

POST: Método HTTP usado para enviar dados ao servidor, geralmente utilizado para submeter formulários e criar ou atualizar recursos no servidor.

Pull Request: Solicitação feita para mesclar alterações feitas em uma branch com o código principal de um projeto, frequentemente usada em ambientes colaborativos para revisão de código.

Python: Linguagem de programação usada em áreas como desenvolvimento web, automação, ciência de dados, entre outras.

Requisição: Ato de solicitar algo, especialmente no contexto de comunicação entre cliente e servidor, como uma requisição HTTP para acessar ou enviar dados.

Renderizar: Processo de gerar a versão visual de um conteúdo a partir de código ou dados, como transformar um template em uma página HTML final.

Rota: Caminho ou URL que determina qual função ou recurso deve ser acessado em um servidor web.

Select: Função utilizada para consultar e recuperar dados específicos de um banco de dados, permitindo ao usuário escolher exatamente o que deseja ver.

Software: Conjunto de programas e instruções que permitem que um computador execute tarefas e interaja com o usuário.

Status: Refere-se ao estado ou condição atual de algo, como o progresso de uma tarefa, o estado de um item ou a situação de um processo.
Template: Modelo ou estrutura pré-definida que serve como base para criar páginas ou documentos, facilitando a criação de conteúdo consistente.

Trigger: Mecanismo de programação usado em bancos de dados que executa automaticamente uma ação quando um evento específico ocorre, como inserção ou atualização de dados.

Update: Função que altera ou atualiza dados existentes em uma tabela de banco de dados, substituindo valores antigos por novos.

Upload: Processo de enviar arquivos de um dispositivo local para um servidor ou plataforma online.

Web: Rede global de sites e aplicativos acessíveis pela internet composta por páginas de conteúdo interligadas que podem ser acessadas através de um navegador.

APÊNDICE A – RF001- Cadastrar usuário

HTML:

Essa página é responsável pelo cadastro de novos clientes na plataforma. Nela, os usuários podem preencher um formulário com informações como nome completo, CPF, e-mail, telefone e senha. O objetivo é registrar esses dados de forma segura e permitir que o cliente tenha acesso às funcionalidades do sistema, como pesquisa e reserva de livros. Além disso, a página inclui um link para redirecionar usuários já cadastrados para a página de login, promovendo uma navegação intuitiva e eficiente.

```
<main class="main-content">
    <div class="box">
        <div class="img-box">
            <img src="/static/img-logo/JL-semfundo.svg" alt="Logo da Biblioteca">
        </div>
        <div class="form-box">
            <h2>Cadastre-se</h2>
            <p>Já é um membro? <a href="/pagina_login">Faça login</a></p>
            <form action="javascript:void(0);">
                <div class="input-group">
                    <label for="nome">Nome completo</label>
                    <input type="text" id="nome" name="nome" placeholder="Digite seu nome completo" required>
                </div>
                <div class="input-group">
                    <label for="cpf">CPF</label>
                    <input type="text" id="cpf" name="cpf" placeholder="Digite seu CPF" required>
                </div>
                <div class="input-group">
                    <label for="email">E-mail</label>
                    <input type="email" id="email" name="email" placeholder="Digite seu e-mail" required>
                </div>
                <div class="input-group">
                    <label for="telefone">Telefone</label>
                    <input type="tel" id="telefone" name="telefone" placeholder="Digite seu telefone" required>
                </div>
                <div class="input-group">
                    <label for="senha">Senha</label>
                    <input type="password" id="senha" name="senha" placeholder="Digite sua senha" required>
                </div>
                <div class="input-group">
                    <button id="btn" onclick="envia_cadastro_cliente()">Cadastrar</button>
                </div>
            </form>
        </div>
    </div>
</main>
```

CSS:

Este arquivo de estilo CSS define o design visual da página de cadastro de clientes do sistema, com foco na organização e com uma paleta de cores suave. O cabeçalho é fixado no topo da página, permitindo acesso constante ao menu administrativo. O corpo da página utiliza tons terrosos (#e6dad1 no fundo e #815b45 no texto), oferecendo um visual harmonioso. O layout principal divide o espaço entre o lado esquerdo, que exibe o logotipo da biblioteca, e o lado direito, onde está o formulário de cadastro. Os campos do formulário são projetados para facilitar a

interação, com bordas arredondadas e destaque visual ao receber foco, promovendo acessibilidade e usabilidade.

```css
/* Estilos gerais */
body {

    background-color: #e6dad1; /* Cor de fundo do site */
    color: #815b45;
    margin: 0;
    padding: 0;
}

/* Fixando o cabeçalho no topo */
header {
    width: 100%;
    background-color: #ffffff;
    padding: 10px 0;
    position: fixed;
    top: 0;
    left: 0;
    z-index: 1000;
}

header nav .menu_adm {
    display: flex;
    justify-content: center;
    color: #ffffff;
}

/* Espaço para o conteúdo não sobrepor o cabeçalho */
.main-content {
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
    min-height: calc(100vh - 70px);
    padding-top: 70px;
    padding-left: 5px;
    padding-left: 5px;
    padding-right: 20px;
    max-width: 900px;
    max-width: 900px;
    margin: 20px auto;
    background-color: #ffffff;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
```

```css
/* Logo do lado esquerdo */
.left-side {
    flex: 1;
    display: flex;
    align-items: center;
    justify-content: center;
    margin: 0;
    padding: 0;
    margin: 0;
    padding: 0;
}

.large-logo img {
    max-width: 400px;
    max-width: 400px;
    width: 100%;
    height: auto;
    display: block;
}

/* Formulário no lado direito */
.right-side {
    flex: 1;
    padding: 20px;
    padding: 20px;
}

.right-side h2 {
    text-align: center;
    color: #572725;
    font-size: 2.0rem;


    letter-spacing: 2px;
    text-transform: uppercase;
}

/* Estilos dos campos do formulário */
.right-side div {
    margin-bottom: 15px;
}

label {
    display: block;
    font-weight: bold;
    color: #815b45;
    margin-bottom: 5px;
}
```

```
input[type="text"],
input[type="email"],
input[type="tel"],
input[type="password"] {
    width: 100%;
    padding: 10px;
    border: 1px solid #bb9273;
    border-radius: 5px;
    background-color: #f8f1ed;
    color: #98502b;
    font-size: 16px;
}

input[type="text"]:focus,
input[type="email"]:focus,
input[type="tel"]:focus,
input[type="password"]:focus {
    outline: none;
    border-color: #815b45;
}

/* Estilos dos botões */
button {
    background-color: #421b1a;
    color: #f0e6d2;
    padding: 10px 20px;
    border: none;
    cursor: pointer;


    transition: background-color 0.3s;
    font-size: 14px;
    border-radius: 0;
    width: 110%;
    max-width: 300px;
    margin: 10px auto;
    width: 110%;
    max-width: 300px;
    margin: 10px auto;
}

button:hover {
    background-color: #f0e6d2;
    color: #98502b;
}

button[type="button"] {
    background-color: #98502b;
}

button[type="button"]:hover {
    background-color: #f0e6d2;
    color: #98502b;
}
```

Python:

Este código implementa a funcionalidade de cadastro de clientes no sistema através de uma API. Ele recebe as informações do cliente de forma assíncrona (via AJAX) e realiza as operações necessárias para salvar os dados e enviar um e-mail de boas-vindas. A rota é configurada para processar requisições do tipo POST. Quando o servidor recebe os dados enviados, ele valida se todos os campos obrigatórios foram preenchidos, como nome, CPF, telefone, e-mail e senha. Se algum campo estiver ausente, o sistema retorna uma mensagem informando qual campo está faltando. Se todos os dados forem válidos, a função tenta cadastrar o cliente no sistema utilizando o método cadastrar da classe correspondente. Caso o cadastro seja bem-sucedido, o sistema prepara um e-mail de boas-vindas personalizado com base em um template HTML. Em seguida, ele tenta enviar esse e-mail para o endereço fornecido.

```python
#função para cadastro de cliente via AJAX
@app.route("/api/set/cadastrar_cliente", methods=["POST"])
def post_cadastro_ajax():
    dados = request.get_json()

    # Campos obrigatórios para o cadastro
    required_fields = ["nome", "cpf", "telefone", "email", "senha"]
    for field in required_fields:
        if field not in dados:
            return jsonify({'mensagem': f"Campo ausente: {field}"}), 400

    nome = dados["nome"]
    cpf = dados["cpf"]
    telefone = dados["telefone"]
    email = dados["email"]  # Endereço do usuário que será usado para enviar o e-mail
    senha = dados["senha"]

    usuario = Usuario()

    # Cadastro do usuário
    if usuario.cadastrar(nome, cpf, telefone, email, senha):
        # Definir o conteúdo do e-mail de boas-vindas
        assunto = "Bem-vindo(a) ao Jardim dos Livros"
        corpo = render_template("email.html", nome=nome)

        # Enviar o e-mail para o endereço cadastrado
        resultado = enviar_email(mail, email, assunto, corpo_html=corpo)

        if resultado:
            return jsonify({'mensagem': 'Cadastro OK e e-mail enviado com sucesso!'}), 200
        else:
            return jsonify({'mensagem': 'Cadastro OK, mas houve um erro ao enviar o e-mail.'}), 500
    else:
        return jsonify({'mensagem': 'Erro ao cadastrar.'}), 500
```

Javascript:

Essa função tem como objetivo enviar os dados preenchidos no formulário de cadastro de cliente para o servidor de forma assíncrona. Ela utiliza a biblioteca jQuery para simplificar a coleta de dados e o envio via AJAX. Primeiro, a função

coleta as informações fornecidas pelo usuário no formulário (nome, CPF, e-mail, telefone e senha) e as organiza em um objeto chamado dados. Esse objeto armazena os valores inseridos nos campos, que são identificados pelos IDs correspondentes no HTML. Depois, a função realiza uma chamada AJAX para a API no servidor, utilizando o método POST. O endereço da API é especificado pela URL /api/set/cadastrar_cliente, e os dados coletados são enviados no formato JSON.

```
function envia_cadastro_cliente() {
    var dados = {
        nome: $("#nome").val(),
        cpf: $("#cpf").val(),
        email: $("#email").val(),
        telefone: $("#telefone").val(),
        senha: $("#senha").val()
    };

    $.ajax({
        url: "/api/set/cadastrar_cliente",
        type: "POST",
        data: JSON.stringify(dados),
        contentType: "application/json",
        success: function(response) {
            console.log("Success:", response);

            Swal.fire("Cadastro realizado com sucesso!");

        },
        error: function(response) {
            console.log("Erro:", response);
            Swal.fire("Erro ao cadastrar", "Verifique o console para mais detalhes.", "error");
        }
    });
}
```

Banco de dados:

Essa tabela é usada para armazenar as informações básicas de cada cliente. O CPF, como chave primária, é o identificador único de cada cliente na tabela.

```
CREATE TABLE tb_cliente (
    nome VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    telefone VARCHAR(20),
    cpf VARCHAR(11) PRIMARY KEY,
    senha VARCHAR(100)
);
```

APÊNDICE B – RF002- Cadastrar funcionário

HTML:

Essa página é responsável pelo cadastro de novos administradores na plataforma. Nela, os usuários podem preencher um formulário com informações como nome completo, CPF, e-mail, telefone e senha. O objetivo é registrar esses dados de

forma segura e permitir que o funcionário tenha acesso às funcionalidades do sistema, como cadastro de livros e acesso a tabela de reservas feitas pelos leitores.

```
<main>
    <div class="box">
        <div class="img-box">
            <img src="/static/img-logo/4.svg" alt="Logo">
        </div>
        <div class="form-box">
            <h2>Criar conta de administrador</h2>
            <p>Já possui uma conta? <a href="/pagina_login">Faça login</a></p>
            <form id="form-cadastro-adm">
                <div class="input-group">
                    <label for="nome">Nome completo:</label>
                    <input type="text" id="nome" name="nome" placeholder="Digite o nome completo" required>
                </div>
                <div class="input-group">
                    <label for="cpf">CPF:</label>
                    <input type="text" id="cpf" name="cpf" placeholder="Digite o CPF" required>
                </div>
                <div class="input-group">
                    <label for="email">E-mail:</label>
                    <input type="email" id="email" name="email" placeholder="Digite o e-mail" required>
                </div>
                <div class="input-group">
                    <label for="telefone">Telefone:</label>
                    <input type="tel" id="telefone" name="telefone" placeholder="Digite o telefone" required>
                </div>
                <div class="input-group">
                    <label for="senha">Senha:</label>
                    <input type="password" id="senha" name="senha" placeholder="Digite sua senha" required>
                </div>
                <div class="input-group">
                    <label for="perfil">Perfil:</label>
                    <select id="perfil" name="perfil" required>
                        <option value="administrador">Administrador</option>
                        <option value="funcionario">Funcionário</option>
                    </select>
                </div>
                <div class="input-group">
                    <button type="button" onclick="envia_cadastro_adm()">Cadastrar</button>
                </div>
            </form>
        </div>
    </div>
</main>
```

CSS:

Este arquivo de estilo CSS define o design visual da página de cadastro, priorizando uma estrutura limpa e moderna. O cabeçalho é fixado no topo da página, garantindo que o menu de navegação esteja sempre acessível, com fundo marrom (#98502b) e texto branco. A paleta de cores da página usa tons suaves e terrosos, como o fundo bege (#e6dad1), criando um ambiente acolhedor e agradável aos olhos. O layout da página é organizado em duas partes: à esquerda, uma área para exibição de uma imagem ou logotipo, e à direita, o formulário de cadastro, que é apresentado em um fundo semitransparente com efeito de desfoque, criando um visual moderno e sofisticado. Os campos de entrada do formulário possuem bordas arredondadas e efeitos de foco suaves, destacando-se quando selecionados, o que facilita a navegação e melhora a usabilidade, tornando a página acessível e fácil de usar.

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    background-color: #e6dad1;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start; /* Ajusta o conteúdo para começar do topo */
    height: 100vh;
    padding: 0;
    margin: 0;
    padding-top: 120px; /* Adicionando o espaço para afastar do cabeçalho */
}


/* Cabeçalho */
header {
    width: 100%;
    position: fixed; /* Fixa o cabeçalho no topo */
    top: 0;
    z-index: 1000;
    background-color: #98502b;
}

nav {
    display: flex;
    justify-content: space-around;
    align-items: center;
    height: 60px;
}

nav a {
    color: #fff;
    text-decoration: none;
    font-size: 16px;
}

.main-content {
    margin-top: 220px; /* Ajuste o valor conforme necessário */
    display: flex;
    justify-content: center;
    align-items: center;
    flex: 1;
    width: 100%;
}
```

```css
.box {
    display: flex;
    background: #ffffff;
    border-radius: 30px;
    overflow: hidden;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
    max-width: 930px;
    width: 100%;
    padding-top: 100px; /* Aumente o espaço no topo do formulário */
}


/* Lado esquerdo - Logo */
.img-box {
    width: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
    background-color: rgba(255, 255, 255, 0.5);
}

.img-box img {
    max-width: 100%;
    height: auto;
}

/* Lado direito - Formulário */
.form-box {
    width: 50%;
    padding: 30px 40px;
    background-color: rgba(255, 255, 255, 0.4);
    backdrop-filter: blur(40px);
    margin-top: 50px; /* Afastar um pouco mais o formulário */
}

.form-box h2 {
    font-size: 28px;
    color: #421b1a;
    text-align: center;
}

.form-box p {
    text-align: center;
    margin-bottom: 20px;
    color: #98502b;
}

.form-box p a {
    color: #ff0000;
    text-decoration: none;
}
```

```css
.input-group {
    margin-bottom: 15px;
}

.input-group label {
    display: block;
    margin-bottom: 5px;
    color: #98502b;
}

.input-group input,
.input-group select {
    width: 100%;
    height: 47px;
    padding: 10px;
    border: 2px solid transparent;
    border-radius: 20px;
    background-color: rgba(255, 255, 255, 0.32);
    outline: none;
    color: #421b1a;
    font-size: 15px;
    transition: all 0.3s ease;
}

.input-group input:focus,
.input-group select:focus {
    border-color: #98502b;
}

.input-group button {
    width: 100%;
    height: 47px;
    background: #421b1a;
    border: none;
    border-radius: 20px;
    color: #f0e6d2;
    font-size: 16px;
    cursor: pointer;
    transition: all 0.3s ease;
}

.input-group button:hover {
    background: #98502b;
    color: #ffffff;
}
```

```css
/* Responsividade */
@media (max-width: 930px) {
    .img-box {
        display: none;
    }
    .box {
        flex-direction: column;
    }
    .form-box {
        width: 100%;
    }
}
```

Python:

Este código implementa a funcionalidade de cadastro de administradores no sistema através de uma API. Ele recebe as informações do administrador de forma assíncrona (via AJAX) e realiza as operações necessárias para salvar os dados e enviar um e-mail de boas-vindas. A rota é configurada para processar requisições do tipo POST. Quando o servidor recebe os dados enviados, ele verifica se todos os campos obrigatórios foram fornecidos, como nome, CPF, telefone, e-mail e senha. Caso algum dado esteja ausente, o sistema retorna uma mensagem informando a falha. Se todos os campos estiverem corretos, o sistema tenta cadastrar o administrador no banco de dados utilizando o método cadastrar_adm da classe UsuarioAdm. Se o cadastro for bem-sucedido, o sistema gera um e-mail de boas-vindas personalizado, utilizando um template HTML específico, e tenta enviar esse e-mail para o endereço fornecido. Se o envio do e-mail for bem-sucedido, o sistema responde com uma mensagem de sucesso. Caso contrário, ele retorna uma mensagem informando que houve um erro no envio do e-mail, mas o cadastro foi realizado.

```python
@app.route("/api/set/cadastrar_adm", methods=["POST"])
def post_cadastro_ajax_adm():
    dados = request.get_json()

    # Extrair informações do JSON
    nome = dados["nome"]
    cpf = dados["cpf"]
    telefone = dados["telefone"]
    email = dados["email"]  # E-mail do administrador que será usado para enviar o e-mail de boas-vindas
    senha = dados["senha"]

    # Instanciando o objeto administrador
    usuarioadm = UsuarioAdm()

    # Realizar o cadastro do administrador
    if usuarioadm.cadastrar_adm(nome, cpf, telefone, email, senha) == True:
        # Configura o conteúdo do e-mail de boas-vindas
        assunto = "Bem-vindo(a) ao Jardim dos Livros - Administração"
        corpo = render_template("email_adm.html", nome=nome)  # Usa um template específico para administradores

        # Envia o e-mail de boas-vindas para o administrador cadastrado
        resultado = enviar_email(mail, email, assunto, corpo_html=corpo)

        if resultado:
            return jsonify({'mensagem': 'Cadastro OK e e-mail enviado com sucesso!'}), 200
        else:
            return jsonify({'mensagem': 'Cadastro OK, mas houve um erro ao enviar o e-mail.'}), 500
    else:
        return jsonify({'mensagem': 'Erro ao cadastrar.'}), 500
```

Javascript:

Este código implementa a função que é responsável por enviar os dados de cadastro de um administrador para a API do sistema. A função coleta os valores dos campos do formulário de cadastro (nome, CPF, e-mail, telefone e senha) e os organiza em um objeto dados. Esse objeto é, então, convertido para o formato JSON e enviado via requisição AJAX para a URL /api/set/cadastrar_adm usando o método POST.

```
function envia_cadastro_adm() {
    var dados = {
        nome: $("#nome").val(),
        cpf: $("#cpf").val(),
        email: $("#email").val(),
        telefone: $("#telefone").val(),
        senha: $("#senha").val()
    }

    $.ajax({
        url: "/api/set/cadastrar_adm",
        type: "POST",
        data: JSON.stringify(dados),
        contentType: "application/json",
        success: function(response) {
            console.log("Success:", response);
            Swal.fire("Cadastro realizado com sucesso!");
        },
        error: function(response) {
            console.log("Erro:", response);
            alert("Erro ao cadastrar. Verifique o console para mais detalhes.");
        }
    });
}
```

Banco de dados:

Esta parte define os campos necessários para o cadastro de administradores, garantindo que cada administrador tenha um CPF único e informações associadas ao seu nome, e-mail, telefone e senha.

```
CREATE TABLE tb_adm (
    nome VARCHAR(100),
    email VARCHAR(100),
    telefone VARCHAR(20),
    cpf VARCHAR(11) PRIMARY KEY,
    senha VARCHAR(100)
);
```

APÊNDICE C – RF003- Login de usuário

APÊNDICE D – RF004- Login de usuário

HTML:

Esta página é responsável pelo login dos usuários na plataforma, permitindo o acesso ao sistema como leitores (clientes) ou administradores. Os usuários devem informar seu CPF e senha, e selecionar seu tipo de usuário antes de enviar o formulário. A página inclui validação dos campos e um botão de login que direciona o usuário para a página principal do sistema, caso as credenciais estejam corretas.

Para usuários que precisam modificar a senha no primeiro acesso, há um link que os direciona para a página de troca de senha.

```
<div class="main-content">
    <div class="login-box">
        <img src="/static/img-logo/3.svg" alt="Logo">
        <h2>Login</h2>
        <form action="/login" method="POST">
            <div class="form-group">
                <label for="tipo_usuario">Tipo de Usuário:</label>
                <select id="tipo_usuario" name="tipo_usuario" required>
                    <option value="" disabled selected>Selecione</option>
                    <option value="cliente">Leitor</option>
                    <option value="admin">Administrador</option>
                </select>
            </div>
            <div class="form-group">
                <label for="cpf">CPF:</label>
                <input type="text" id="cpf" name="cpf" required>
            </div>
            <div class="form-group">
                <label for="senha">Senha:</label>
                <input type="password" id="senha" name="senha" required>
            </div>
            <div class="form-group">
                <button type="submit">Entrar</button>
            </div>
        </form>
        <div class="form-group register-link">
            <p>Modifique sua senha para o primeiro acesso</p>
            <a href="/atualizar_senha"><button type="submit">Trocar senha</button></a>
        </div>
```

CSS:

Este arquivo de estilo CSS define o layout e a aparência da página de login, focando em uma estética simples e elegante. A página tem um fundo suave (#f5e8df) e um cabeçalho fixo no topo, com uma cor de fundo escura (#421b1a) e texto destacado. O formulário de login é centralizado e estilizado com bordas arredondadas e uma caixa de sombra discreta, proporcionando um contraste agradável. O botão de login possui um estilo limpo, com bordas coloridas (#98502b) e um efeito de hover que destaca a interação, oferecendo uma experiência de usuário agradável. Além disso, o link para modificar a senha é estilizado de forma acessível, com um efeito de hover que muda a cor de fundo para um tom mais suave (#bb9273), incentivando a navegação intuitiva. O layout responsivo e a escolha de cores criam uma interface agradável e fácil de usar.

```css
body {
    background-color: #f5e8df;
    margin: 0;
    padding: 0;
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
}

header {
    position: fixed;
    top: 0;
    width: 100%;
    background-color: #421b1a;
    padding: 10px 20px;
    z-index: 10;
}

h2 {
    color: #421b1a;
    margin-bottom: 5px;
    padding: 14px;
    cursor: pointer;
    font-size: 25px;
    text-transform: uppercase;
    text-decoration: none;
    margin: 0 10px;
    margin-top: -35px;

    padding-left: 20px;
    letter-spacing: 1px;
}
```

```css
.main-content {
    margin-top: 100px; /* Ajustar para não interferir no cabeçalho */
    display: flex;
    justify-content: center;
    align-items: center;
    height: calc(100vh - 100px); /* Altura da tela menos o cabeçalho */
}

.login-box {
    background-color: #ffffff;
    padding: 40px;
    border-radius: 15px;
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
    width: 350px;
    text-align: center;
}

.login-box img {
    width: 150px; /* Aumenta o tamanho da imagem */
    margin-bottom: 20px;
}

.login-box select {
    width: 96%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

.login-box input {
    width: 90%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
```

```css
.login-box button {
    background-color: transparent;
    color: #98502b;
    border: 2px solid #98502b;
    border-radius: 0px;
    padding: 14px;
    cursor: pointer;
    font-size: 14px;
    transition: background-color 0.3s, color 0.3s;
    text-decoration: none;
    margin: 0 10px;

    padding-left: 20px;
    letter-spacing: 1px;
    text-transform: uppercase;
}

.login-box button:hover {
    background-color: #f0e6d2;
    color: #98502b;
}

.link-modificar-senha {
    text-decoration: none;
    color: #98502b;
    font-size: 15px;
    padding: 20px 15px;
    margin: 3px 0; /* Adiciona 15px de espaço em cima e embaixo */
    display: inline-block; /* Garante que o padding e o margin funcionem corretamente */
}

.link-modificar-senha:hover {
    background-color: #bb9273;
    color: white;
}
```

Python:

Este trecho de código implementa a funcionalidade de login no sistema, permitindo que o usuário se autentique tanto como administrador quanto como cliente. Quando a requisição é do tipo POST, o código captura os dados do formulário (tipo de usuário, CPF e senha). Dependendo do tipo de usuário selecionado (administrador ou cliente), o sistema verifica as credenciais. Se o tipo de usuário for "admin", o sistema tenta autenticar o administrador, verificando o CPF e a senha através do método logar_adm da classe UsuarioAdm. Se as credenciais forem válidas, o administrador é redirecionado para a página de administração (/adm), e a sessão é iniciada com as informações do administrador. Se o tipo de usuário for "cliente", o sistema tenta autenticar o cliente usando o método autenticar da classe Usuario. Se o login for bem-sucedido, o cliente é redirecionado para a página inicial, e suas informações de sessão são armazenadas. Em ambos os casos, se as credenciais forem incorretas, uma mensagem de erro é exibida e o usuário é redirecionado de

volta para a tela de login. Se a requisição for do tipo GET, o sistema apenas renderiza a página de login.

```python
@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        tipo_usuario = request.form.get("tipo_usuario")  # Captura o tipo de usuário
        cpf = request.form.get("cpf")
        senha = request.form.get("senha")

        if tipo_usuario == "admin":
            usuarioadm = UsuarioAdm()
            if usuarioadm.logar_adm(cpf, senha):
                session["usuarioadm_logado"] = {"cpf": usuarioadm.cpf, "nome": usuarioadm.nome}
                flash("Login realizado com sucesso!", "success")
                return redirect("/adm")  # Redirecionar para a página do admin
            else:
                flash("CPF ou senha incorretos.", "error")
                return render_template("login.html")  # Retorna o login com mensagem de erro

        elif tipo_usuario == "cliente":
            usuario = Usuario()
            if usuario.autenticar(cpf, senha):
                session["usuario_logado"] = {"cpf": usuario.cpf,
                                             "nome": usuario.nome,
                                             "email": usuario.email,
                                             "telefone": usuario.telefone}



                flash("Login realizado com sucesso!", "success")
                return redirect("/")  # Redirecionar para a página do cliente
            else:
                flash("CPF ou senha incorretos.", "error")
                return render_template("login.html")  # Retorna o login com mensagem de erro

    # Se o método for GET, apenas renderiza o formulário de login
    return render_template("login.html")
```

APÊNDICE E – RF005- Cadastro de livro

HTML:

Esta página tem como objetivo principal permitir que o administrador adicione novos livros ao banco de dados de forma simples e eficiente. Ao acessar a página, o administrador encontra um formulário dividido em duas seções principais. Na primeira seção, o formulário solicita dados como ISBN, título e autor do livro. O ISBN pode ser usado para buscar automaticamente os detalhes do livro, como título e autor, facilitando o processo de cadastro. Além disso, a página oferece a opção de adicionar uma imagem do livro. Caso o administrador já tenha uma imagem disponível, ele pode fazer o upload diretamente da máquina, ou, se preferir, a imagem pode ser buscada automaticamente. A imagem, se disponível, será exibida em uma área específica.

```
<main>

        <!-- Container flex para as colunas -->
        <div class="form-container">
            <div class="left-column">
                <div class="form-group">
                    <label for="isbn">ISBN:</label>
                    <input type="text" id="isbn" name="isbn" placeholder="ex: 9781484213056" required>
                    <button type="button" onclick="buscarDetalhesLivro()">Buscar</button>
                </div>
                <div class="form-group">
                    <label for="titulo">Título:</label>
                    <input type="text" id="titulo" name="titulo" required>
                </div>
                <div class="form-group">
                    <label for="autor">Autor:</label>
                    <input type="text" id="autor" name="autor" required>
                </div>
            </div>

            <!-- Linha de separação -->
            <div class="divider"></div>

            <div class="right-column">
                <div class="form-group">
                    <label for="categoria">Categoria:</label>
                    <input type="text" id="categoria" name="categoria" required>
                </div>
                <div class="form-group">
                    <label for="descricao">Descrição:</label>
                    <textarea id="descricao" name="descricao" required></textarea>
                </div>
                <div class="form-group">
                    <label for="data_publicacao">Data de publicação:</label>
                    <input type="text" id="data_publicacao" name="data_publicacao">
                </div>
            </div>
        </div>
```

```
        <div class="container-image">
            <div>
                <label for="imagem">Imagem do livro:</label>
                <img id="imagem-livro" src="" alt="Imagem do Livro" style="display:none; max-width: 200px; max-height: 300px;">
            </div>
            <div>
                <label for="imagem_upload">Carregar imagem do seu computador:</label>
                <input type="file" id="imagem_upload" name="imagem_upload" accept="image/*">
            </div>

        </div>

        <button type="button" onclick="cadastrarLivro()">Cadastrar livro</button>
    </form>
</main>
```

CSS:

Este código é responsável pelo design e estilo de uma página de cadastro de livros. A página utiliza um layout limpo e centrado, com o corpo da página tendo um fundo suave e cores neutras que garantem boa leitura e contraste. A primeira coluna contém campos para informações essenciais do livro, como ISBN, título e autor, enquanto a segunda coluna foca em dados adicionais, como categoria, descrição e data de publicação. Os campos de entrada, como texto e upload de arquivos, são estilizados para proporcionar uma boa usabilidade, com bordas arredondadas e efeitos de foco que indicam claramente onde o usuário está interagindo. Os botões, como o de "Cadastrar livro", são destacados com cores que seguem o tema da página, com um fundo marrom escuro que muda de cor quando o mouse passa sobre ele.

```css
/* Estilo geral da página */
body {
    background-color: #e6dad1;
    color: #333;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    font-family: 'Arial', sans-serif; /* Ajuste para garantir uma boa base */
}

/* Estilo geral */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Estilo principal */
main {
    margin-top: 30px;
    width: 90%;
    max-width: 800px;
    background: #ffffff;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    padding: 20px;
}

main h2 {
    text-align: center;
    color: #572725;
    font-size: 2rem;
    letter-spacing: 2px;
    margin-top: 10px;
    margin-bottom: 20px;
}
```

```css
/* Container das colunas do formulário */
.form-container {
    display: flex;
    justify-content: space-between;
    gap: 20px;
    margin-top: 20px;
}

.left-column, .right-column {
    flex: 1;
}

.divider {
    width: 1px;
    background-color: #ccc;
    margin: 0 10px;
}

/* Estilos uniformes para os campos */
.form-group {
    margin-bottom: 20px;
}

label {
    display: block;
    font-weight: bold;
    margin-bottom: 8px;
    color: #421b1a;
    font-size: 14px;
}
```

```css
input[type="text"],
textarea,
input[type="file"] {
    width: 100%;
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 5px;
    outline: none;
    font-size: 14px;
    transition: border-color 0.3s ease-in-out;
    background-color: #fafafa;
}

input[type="text"]:focus,
textarea:focus {
    border-color: #421b1a;
    background-color: #fff;
}

textarea {
    resize: vertical;
}

/* Estilo para o botão "Buscar" */
#isbn {
    width: calc(100% - 90px);
    display: inline-block;
}

button[type="button"] {
    background-color: #572725;
    color: white;
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease-in-out;
    font-size: 14px;
    margin-top: 10px;
}
```

```css
button[type="button"]:hover {
    background-color: #421b1a;
}

/* Estilos do campo de upload de imagem */
#imagem_upload {
    padding: 10px;
    border: 1px solid #572725;
    border-radius: 5px;
    cursor: pointer;
    color: #572725;
    font-weight: bold;
    background-color: #f9f9f9;
    transition: background-color 0.3s ease-in-out, color 0.3s ease-in-out;
}

#imagem_upload:hover {
    background-color: #572725;
    color: #fff;
}

/* Imagem do livro */
.container-image {
    margin-top: 20px;
    text-align: center;
}

#imagem-livro {
    display: inline-block;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 5px;
    max-width: 200px;
    max-height: 300px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
```

```css
/* Botão Cadastrar livro */
button[type="submit"] {
    width: 100%;
    padding: 12px;
    font-size: 16px;
    background-color: #572725;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease-in-out;
    margin-top: 20px;
}

button[type="submit"]:hover {
    background-color: #421b1a;
}

/* Responsividade */
@media (max-width: 768px) {
    .form-container {
        flex-direction: column;
    }

    .divider {
        display: none;
    }

    main h2 {
        font-size: 1.5rem;
    }

    button[type="button"],
    button[type="submit"] {
        font-size: 14px;
        padding: 10px;
    }
```

Python:

Este trecho de código implementa a funcionalidade de cadastro de livros no sistema. Quando a requisição é do tipo POST, o código captura os dados enviados pelo formulário, como ISBN, título, autor, categoria, descrição, data de publicação e imagem do livro. A imagem pode ser fornecida de duas maneiras: via API ou por upload manual. Se a API retornar uma imagem, o link dessa imagem é utilizado. Caso contrário, o sistema verifica se o administrador fez o upload de um arquivo de imagem.

```python
@app.route('/cadastrar_livro', methods=['POST'])
def cadastrar_livro():

    isbn = request.form.get('isbn')
    titulo = request.form.get('titulo')
    autor = request.form.get('autor')
    categoria = request.form.get('categoria')
    descricao = request.form.get('descricao')
    data_publicacao = request.form.get('data_publicacao')
    imagem = request.form.get('imagem')
    # Verifique se existe uma imagem carregada via API
    imagem_api = request.form.get('imagem')  # Imagem obtida pela API (se disponível)
    # Se a imagem da API não estiver disponível, verifique se há um arquivo enviado
    if imagem_api != '' and imagem_api != None:
        link_imagem = imagem_api  # Se a API retornar imagem, usar o link da API
    else:
        # Verificar se foi feito um upload manual
        imagem = request.files.get('imagem_upload')  # Imagem enviada manualmente pelo form
        if imagem and imagem.filename != '':
            link_imagem = upload_file(imagem) # Faz o upload da imagem para o Azure
        else:
            link_imagem = None  # Caso nenhuma imagem seja fornecida
    # Verifica se o ISBN foi fornecido
    if not isbn:
        return jsonify({'mensagem': 'Por favor, insira o ISBN.'}), 400
    # Conectar ao banco e verificar se o ISBN já existe
    mydb = Conexao.conectar()
    mycursor = mydb.cursor()
    mycursor.execute("SELECT * FROM tb_livro WHERE ISBN = %s", (isbn,))
    livro_existente = mycursor.fetchone()
    if livro_existente:
        # Se o livro já existir, retornar mensagem de erro
        return jsonify({'mensagem': 'Este ISBN já está cadastrado.'}), 400
    else:
        # Inserir o novo livro no banco de dados
        sql = """
            INSERT INTO tb_livro (ISBN, titulo, autor, categoria, descricao, data_publicacao, foto)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """
        mycursor.execute(sql, (isbn, titulo, autor, categoria, descricao, data_publicacao, link_imagem))
        mydb.commit()
```

```python
mydb.close()

return jsonify({'status': 'sucesso', 'mensagem': 'Livro cadastrado com sucesso!'}), 200
```

Javascript:

Este trecho de código contém várias funcionalidades para gerenciar o cadastro de livros e buscar seus detalhes com base no ISBN. A função buscarDetalhesLivro é responsável por buscar as informações do livro a partir do ISBN informado pelo usuário. Quando o ISBN é fornecido e a requisição é bem-sucedida, a função preenche os campos do formulário com os dados do livro. A segunda função, cadastrarLivro, envia os dados preenchidos no formulário para o servidor via AJAX, utilizando FormData para envia os dados do livro. Se o livro já estiver cadastrado (verificado pelo ISBN), o sistema pergunta ao usuário se deseja cadastrar um novo exemplar. A função cadastrarExemplar é chamada quando o usuário opta por cadastrar um novo exemplar para o ISBN já existente. Ela envia a requisição AJAX com o ISBN para o servidor e, se o cadastro do exemplar for bem-sucedido, o formulário é limpo e uma mensagem é exibida. Por fim, a função limparFormulario é

responsável por limpar todos os campos do formulário após a operação, incluindo o campo de imagem, que é ocultado.

```
//função para buscar os detalhes desejados do livro usando o ISBN
function buscarDetalhesLivro() {
    var isbn = document.getElementById("isbn").value;

    if (isbn) {
        $.ajax({
            url: `/buscar_livro/${isbn}`,
            type: "GET",
            success: function(response) {
                if (response.error) {
                    alert("Livro não encontrado.");
                } else {
                    document.getElementById("titulo").value = response.titulo || "N/A";
                    document.getElementById("autor").value = response.autor || "N/A";
                    document.getElementById("categoria").value = response.categoria || "N/A";
                    document.getElementById("descricao").value = response.descricao || "N/A";
                    document.getElementById("data_publicacao").value = response.data_publicacao || "N/A";


                    if (response.imagem) {
                        var img = document.getElementById("imagem-livro");
                        img.src = response.imagem;
                        img.style.display = "block";
                    } else {
                        document.getElementById("imagem-livro").style.display = "none";
                    }
                }
            },
            error: function(xhr, status, error) {
                console.error(`Erro ao buscar detalhes do livro: ${xhr.statusText}`);
                alert("Erro ao buscar detalhes do livro.");
            }
        });
    } else {
        alert("Por favor, insira o ISBN.");
    }
}
```

```
//Função para cadastrar um livro criando um objeto 'dados' que contém as informações do livro obtidas dos campos do formulário HTML
function cadastrarLivro() {
    var dados = new FormData();
    dados.append("isbn", $("#isbn").val());
    dados.append("titulo", $("#titulo").val());
    dados.append("autor", $("#autor").val());
    dados.append("categoria", $("#categoria").val());
    dados.append("descricao", $("#descricao").val());
    dados.append("data_publicacao", $("#data_publicacao").val());
    dados.append("imagem-livro", $("imagem-livro").val());
    dados.append("imagem_upload", $('#imagem_upload')[0].files[0]);


    // Faz uma requisição AJAX para enviar os dados ao servidor
    $.ajax({
        url: "/cadastrar_livro",
        type: "POST",
        data: dados,
        contentType: false,
        processData: false,
        success: function(response) {

            if (response.status === 'sucesso') {
                Swal.fire("Cadastro realizado com sucesso!");
                limparFormulario();
            }
        },
        error: function(xhr) {

            if (xhr.status === 400 && xhr.responseJSON.mensagem.includes("ISBN já está cadastrado")) {

                var confirmar = confirm(xhr.responseJSON.mensagem + " Deseja cadastrar um novo exemplar?");
                if (confirmar) {
                    cadastrarExemplar($("#isbn").val());
                }
            } else {
                alert("Erro ao cadastrar o livro.");
            }
        }
    });
}
```

```javascript
//Função que cadastra o mesmo ISBN com outro código exemplar
function cadastrarExemplar(isbn) {
    $.ajax({
        url: "/cadastrar_exemplar",
        type: "POST",
        data: JSON.stringify({ isbn: isbn }),
        contentType: "application/json",
        success: function(response) {
            if (response.status === 'sucesso') {
                alert(response.mensagem);
                limparFormulario();
            }
        },
        error: function(xhr) {
            alert("Erro ao cadastrar o exemplar.");
        }
    });
}

//Função que limpa o formulário depois que os dados são enviados
function limparFormulario() {
    $("#isbn").val('');
    $("#titulo").val('');
    $("#autor").val('');
    $("#categoria").val('');
    $("#descricao").val('');
    $("#data_publicacao").val('');
    $("#imagem-livro").attr('src', '').hide();


}
```

Banco de dados:

Essa tabela permite que os dados dos livros, como título, autor, descrição, categoria e imagem, sejam armazenados de forma estruturada no banco de dados, com um identificador único para cada livro através do campo id_livro.

```sql
CREATE TABLE tb_livro (
    ISBN VARCHAR(20),
    titulo TEXT NOT NULL,
    autor TEXT NOT NULL,
    cod_exemplar INT,
    id_livro INT AUTO_INCREMENT PRIMARY KEY,
    descricao TEXT NOT NULL,
    categoria TEXT NOT NULL,
    data_publicacao VARCHAR(50),
    foto VARCHAR(500)
);
```

APÊNDICE F – RF006- Reserva de livro

HTML:

Esta página tem como objetivo principal permitir que o administrador visualize as reservas de livros feitas pelos usuários. Ao acessar a página, o administrador encontrará uma tabela que exibe informações relevantes sobre cada reserva. A tabela é composta por colunas com dados como o nome do livro, ISBN, código do exemplar, CPF do usuário, data da reserva, data de devolução e data de retirada. Além disso, há uma coluna dedicada a ações, onde o administrador pode gerenciar as reservas.

Os dados da tabela serão carregados dinamicamente via JavaScript, garantindo que as informações sejam atualizadas automaticamente conforme novas reservas são feitas. A página foi desenvolvida para facilitar o controle de reservas, proporcionando uma visão clara e organizada das informações dos livros e seus respectivos status de empréstimo.

```html
<h1>LIVROS RESERVADOS</h1>

<table>
    <thead>
        <tr>
            <th>Nome</th>
            <th>Livro</th>
            <th>ISBN</th>
            <th>cod_exemplar</th>
            <th>CPF</th>
            <th>Data de Reserva</th>
            <th>Data de Devolução</th>
            <th>Data de Retirada</th>
            <th>Ações</th>
        </tr>
    </thead>
    <tbody id="tabela-reservas">
        <!-- Reservas serão carregadas aqui via JavaScript -->
    </tbody>
</table>
```

CSS:

Este código é responsável pelo design e estilo de uma página de reservas de livros. A página apresenta um fundo suave e cores terrosas que proporcionam boa legibilidade e contraste. O cabeçalho contém o título centralizado e estilizado com uma cor mais escura para destacar o texto. A tabela de reservas ocupa grande parte

da página, com colunas para informações essenciais, como nome do livro, ISBN, código do exemplar e dados de reserva, devolução e retirada. As células da tabela são estilizadas com cores alternadas e efeitos de hover, proporcionando uma navegação visualmente agradável e interativa. Os botões de ação, como os de "Confirmar", possuem um fundo vermelho escuro que muda de cor ao passar o mouse, oferecendo feedback visual e destacando a interatividade da página. Além disso, a tabela e os botões são projetados para garantir uma boa usabilidade e facilitar a interação do administrador com o sistema de reservas.

```css
/* Estilo Geral */
body {
    background-color: #FDF6EC; /* Fundo claro */
    color: #4B3F2F; /* Tom terroso escuro */

    margin: 0;
    padding: 0;
}

h1 {
    text-align: center;
    color: #572725;
    font-size: 2rem;

    letter-spacing: 2px;
}



.menu_adm {
    display: flex;
    gap: 15px;
}

.menu_adm a {
    color: #FDF6EC;
    text-decoration: none;
    font-size: 16px;
    padding: 8px 12px;
    border-radius: 5px;
    transition: background-color 0.3s, color 0.3s;
}

.menu_adm a:hover {
    background-color: #f0e6d2; /* Fundo suave */
    color: #421b1a; /* Vermelho terroso escuro */
}

/* Estilo da Tabela */
table {
    width: 90%;
    margin: 20px auto;
    border-collapse: collapse;
    font-size: 16px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    border-radius: 8px;
    overflow: hidden;
}

table thead {
    background-color: #421b1a; /* Vermelho terroso escuro */
    color: #FDF6EC; /* Fundo claro */
}
```

```css
table thead th {
    padding: 15px;
    text-align: left;
}

table tbody td {
    padding: 10px;
    border-bottom: 1px solid #f0e6d2;
    color: #4B3F2F; /* Tom terroso escuro */
}

table tbody tr:nth-child(even) {
    background-color: #f7f2ec; /* Fundo mais claro */
}

table tbody tr:hover {
    background-color: #f0e6d2;
}

/* Estilo dos Botões de Ação */
button {
    background-color: #421b1a; /* Vermelho terroso escuro */
    color: #FDF6EC; /* Fundo claro */
    border: none;
    padding: 8px 12px;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #341514; /* Vermelho terroso mais escuro */
}

/* Estilo dos Botões de Confirmação */
button#confirmar {
    background-color: #98502b; /* Marrom claro */
}

button#confirmar:hover {
    background-color: #7b3d22; /* Marrom mais escuro */
}
```

Python:

Este código implementa um sistema de reservas de livros em uma plataforma de biblioteca. Ele permite que clientes reservem livros, verificando se o usuário está logado, se já atingiu o limite de reservas e se o livro escolhido está disponível. Após a reserva, é gerada uma data de devolução, que é registrada no banco de dados. O código também oferece funcionalidades para listar todas as reservas ativas e permitir que o administrador atualize o status das reservas, marcando quando um livro é retirado ou entregue. Essas operações garantem o controle eficiente das reservas de livros e a gestão de exemplares disponíveis na biblioteca.

```python
@app.route('/reservar_livro', methods=['POST'])
def reservar_livro():
    dados = request.get_json()
    isbn = dados.get('isbn')
    cpf = session.get('usuario_logado', {}).get('cpf')  # Assume que o cliente está logado e o CPF está na sessão

    if not isbn:
        return jsonify({'mensagem': 'Por favor, insira o ISBN.'}), 400
    if not cpf:
        return jsonify({'mensagem': 'Usuário não está logado.'}), 401

    try:
        # Conectar ao banco de dados
        mydb = Conexao.conectar()
        if mydb is None:
            return jsonify({'mensagem': 'Erro na conexão com o banco de dados.'}), 500

        mycursor = mydb.cursor()

        # Verificar quantos livros já estão reservados pelo CPF e se o ISBN é diferente
        query_reservas_existentes = """
        SELECT COUNT(*), GROUP_CONCAT(ISBN) as isbn
        FROM tb_reserva
        WHERE cpf = %s AND data_entrega IS NULL
        """
        mycursor.execute(query_reservas_existentes, (cpf,))
        reserva_info = mycursor.fetchone()

        reservas_count = reserva_info[0]
        isbns_reservados = reserva_info[1].split(',') if reserva_info[1] else []

        if reservas_count >= 2:
            return jsonify({'mensagem': 'Você já reservou o máximo de 2 livros.'}), 403

        if isbn in isbns_reservados:
            return jsonify({'mensagem': 'Você já reservou um livro com esse ISBN.'}), 403

        # Buscar um exemplar disponível do mesmo ISBN que ainda não foi reservado
        query_livro_disponivel = """
        SELECT l.cod_exemplar
        FROM tb_livro l
        WHERE l.ISBN = %s
        AND l.cod_exemplar NOT IN (
            SELECT r.cod_exemplar
            FROM tb_reserva r
            WHERE r.ISBN = %s
            AND r.data_entrega IS NULL
        )
        LIMIT 1
        """
```

```
    mycursor.execute(query_livro_disponivel, (isbn, isbn))
    exemplar_disponivel = mycursor.fetchone()

    if exemplar_disponivel is None:
        return jsonify({'mensagem': 'Nenhum exemplar disponível para reserva.'}), 404

    cod_exemplar = exemplar_disponivel[0]

    # Calcula a data de devolução (7 dias a partir da data de reserva)
    data_reserva = datetime.now()  # Data atual
    data_devolucao = (data_reserva + timedelta(days=7)).replace(hour=23, minute=59, second=59) # Soma 7 dias para a devolução

    # Inserir na tabela de reservas com a data de devolução
    query_reserva = """
    INSERT INTO tb_reserva (cod_exemplar, ISBN, cpf, data_reserva, data_devolucao)
    VALUES (%s, %s, %s, %s, %s)
    """
    mycursor.execute(query_reserva, (cod_exemplar, isbn, cpf, data_reserva, data_devolucao))
    mydb.commit()

    return jsonify({'                                    do com sucesso! Data de devolução: ' + data_devolucao.strftime('%Y-%m-%d')}), 200
                    (variable) e: Exception
except Exception as e:
    mydb.rollback()
    return jsonify({'mensagem': f'Erro ao reservar livro: {str(e)}'}), 500

finally:
    mydb.close()
```

```
@app.route('/livros_reservados', methods=['GET'])
def listar_reservas():
    try:
        # Conectar ao banco de dados
        mydb = Conexao.conectar()
        if mydb is None:
            return jsonify({'mensagem': 'Erro na conexão com o banco de dados.'}), 500

        mycursor = mydb.cursor(dictionary=True)

        # Consulta para obter todos os livros reservados
        query = """
        SELECT r.cod_exemplar, r.ISBN, l.titulo, r.cpf, r.data_reserva, r.data_entrega, r.data_retirada, r.data_devolucao, c.nome
        FROM tb_reserva r
        JOIN tb_livro l ON r.ISBN = l.ISBN and r.cod_exemplar=l.cod_exemplar
        JOIN tb_cliente c on r.cpf = c.cpf
        WHERE  r.data_entrega IS NULL
        ORDER BY cod_reserva
        """
        mycursor.execute(query)
        reservas = mycursor.fetchall()

        return jsonify(reservas), 200

    except Exception as e:
        return jsonify({'mensagem': f'Erro ao buscar reservas: {str(e)}'}), 500

    finally:
        mydb.close()
```

```python
@app.route('/atualizar_reserva', methods=['POST'])
def atualizar_reserva():
    dados = request.get_json()
    cod_exemplar = dados.get('cod_exemplar')
    isbn = dados.get('isbn')
    acao = dados.get('acao')  # "retirado" ou "entregue"

    if not cod_exemplar or not acao:
        return jsonify({'mensagem': 'Dados incompletos.'}), 400

    try:
        # Conectar ao banco de dados
        mydb = Conexao.conectar()
        if mydb is None:
            return jsonify({'mensagem': 'Erro na conexão com o banco de dados.'}), 500

        mycursor = mydb.cursor()

        if acao == 'retirado':
            # Atualizar a data de retirada
            query = """
            UPDATE tb_reserva
            SET data_retirada = NOW()
            WHERE cod_exemplar  = %s AND ISBN = %s AND data_retirada IS NULL
            """
            mycursor.execute(query, (cod_exemplar,isbn,))

        elif acao == 'entregue':
            # Atualizar a data de entrega
            query = """
            UPDATE tb_reserva
            SET data_entrega = NOW()
            WHERE cod_exemplar = %s AND ISBN = %s  AND data_retirada IS NOT NULL AND data_entrega IS NULL
            """
            mycursor.execute(query, (cod_exemplar,isbn,))

        mydb.commit()

        return jsonify({'mensagem': f'Livro {acao} com sucesso.'}), 200

    except Exception as e:
        mydb.rollback()
        return jsonify({'mensagem': f'Erro ao atualizar reserva: {str(e)}'}), 500

    finally:
        mydb.close()
```

Javascript:

Este código em JavaScript gerencia e exibe reservas de livros em uma tabela dinâmica na página. Ele carrega os dados do servidor, ajusta as datas para o fuso horário local e exibe informações como cliente, título, ISBN, e status da reserva. Além disso, permite ao administrador atualizar o status das reservas (como "retirado" ou "entregue") por meio de botões interativos, confirmando ações com alertas visuais. Após qualquer atualização, a tabela é recarregada automaticamente para manter os dados atualizados.

```
<script>
    // Função para ajustar a data com o fuso horário local
    function ajustarDataParaLocal(dataString) {
        const data = new Date(dataString);
        return new Date(data.getTime() + data.getTimezoneOffset() * 60000).toLocaleDateString('pt-BR');
    }

    // Função para carregar todas as reservas
    function carregarReservas() {
        fetch('/livros_reservados')
            .then(response => response.json())
            .then(data => {
                const tabelaReservas = document.getElementById('tabela-reservas');
                tabelaReservas.innerHTML = ''; // Limpar a tabela


                data.forEach(reserva => {
                    const linha = document.createElement('tr');

                    linha.innerHTML = `
                        <td>${reserva.nome}</td>
                        <td>${reserva.titulo}</td>
                        <td>${reserva.ISBN}</td>
                        <td>${reserva.cod_exemplar}</td>
                        <td>${reserva.cpf}</td>
                        <td>${ajustarDataParaLocal(reserva.data_reserva)}</td>
                        <td>${reserva.data_devolucao ? ajustarDataParaLocal(reserva.data_devolucao) : 'Não devolvido'}</td>
                        <td>${reserva.data_retirada ? ajustarDataParaLocal(reserva.data_retirada) : 'Não retirado'}</td>
                        <td>
                            ${reserva.data_retirada === null ?
                                `<button onclick="confirmarAtualizacaoReserva(${reserva.cod_exemplar}, '${reserva.ISBN}', 'retirado')">Retirado</button>`
                                : ''}
                            ${reserva.data_entrega === null && reserva.data_retirada != null ?
                                `<button onclick="confirmarAtualizacaoReserva(${reserva.cod_exemplar}, '${reserva.ISBN}', 'entregue')">Entregue</button>`
                                : ''}
                        </td>
                    `;

                    tabelaReservas.appendChild(linha);
                });
```

```
}

// Função para mostrar confirmação e atualizar a reserva (retirada ou entrega)
function confirmarAtualizacaoReserva(cod_exemplar, isbn, acao) {
    Swal.fire({
        title: "Tem certeza?",
        text: "Você não poderá reverter esta ação!",
        icon: "warning",
        showCancelButton: true,
        confirmButtonColor: "#98502b",
        cancelButtonColor: "#421b1a",
        confirmButtonText: "Sim, confirmar!",
        cancelButtonText: "Cancelar"
    }).then((result) => {
        if (result.isConfirmed) {
            atualizarReserva(cod_exemplar, isbn, acao);
        }
    });
}

// Função para atualizar a reserva (retirada ou entrega)
function atualizarReserva(cod_exemplar, isbn, acao) {
    fetch('/atualizar_reserva', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ cod_exemplar: cod_exemplar, acao: acao, isbn: isbn })
    })
    .then(response => response.json())
    .then(data => {
        Swal.fire({
            title: "Atualizado!",
            text: data.mensagem,
            icon: "success"
        });
        carregarReservas(); // Recarregar a tabela após a atualização
    })
    .catch(error => {
        console.error('Erro ao atualizar reserva:', error);
    });
}

// Carregar reservas ao carregar a página
window.onload = carregarReservas;
```

Banco de dados:

Este código cria a tabela **tb_reserva**, que registra as reservas de livros realizadas por clientes. Ela inclui um identificador único da reserva, CPF do cliente, código do exemplar, ISBN do livro, e datas relacionadas ao processo de reserva, como data da reserva, retirada, devolução e entrega. Além disso, estabelece uma relação com a

tabela tb_cliente por meio de uma chave estrangeira no campo cpf.

```sql
CREATE TABLE tb_reserva (
    cod_reserva INT AUTO_INCREMENT PRIMARY KEY,
    cpf VARCHAR(11),
    cod_exemplar INT,
    data_reserva date,
    data_retirada date,
    data_devolucao date,
    data_entrega date,
    ISBN VARCHAR(20),
    FOREIGN KEY (cpf) REFERENCES tb_cliente(cpf)
);
```

APÊNDICE G – RF007- Busca de livro

HTML:

Este código cria um formulário simples com um campo de texto para o usuário digitar palavras-chave de pesquisa e um botão para enviar a busca. Ele é usado para coletar dados de pesquisa de maneira interativa.

```html
<form id="search-form">
    <input type="text" id="pesquisa" placeholder="Digite palavras-chave"> <!-- Campo para termos de pesquisa -->
    <button type="submit">Pesquisar</button> <!-- Botão de submissão do formulário -->
</form>
```

CSS:

Este código define estilos para um contêiner centralizado e uma barra de pesquisa com um campo de texto e um botão estilizados. O contêiner usa flexbox para centralizar seu conteúdo, enquanto o campo de texto e o botão têm bordas arredondadas e se ajustam lado a lado. O botão possui uma transição suave para alterar sua cor ao passar o mouse, criando um efeito visual dinâmico. O estilo

combina cores e bordas para criar um design atraente e intuitivo.

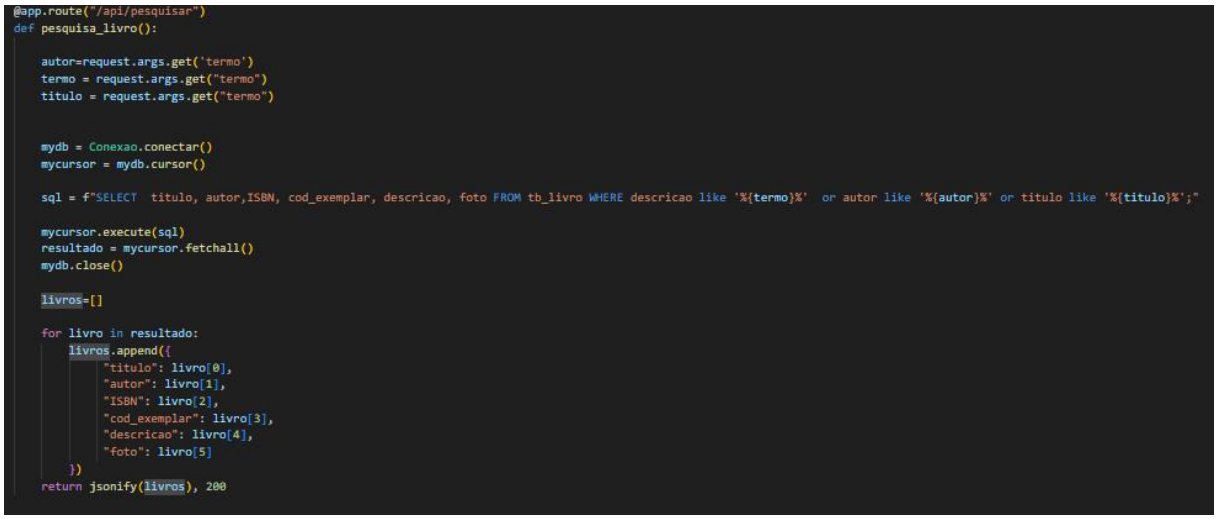
```css
/* Container para a barra de pesquisa e categoria */
#container {
    display: flex;
    justify-content: center;
    align-items: center;
    margin: 20px 0;
}

/* Barra de pesquisa */
#search-form {
    display: flex;
}

#pesquisa {
    padding: 10px;
    border: 1px solid #421b1a;
    border-radius: 10px 0 0 10px;
    outline: none;
    width: 250px;
}

button {
    padding: 10px 20px;
    background-color: #421b1a;
    color: #f0e6d2;
    border: none;
    border-radius: 0 10px 10px 0; /* antos arredondados do lado direito */
    cursor: pointer;
    transition: background-color 0.3s;

}
```

Python:

Este código define uma rota no Flask (/api/pesquisar) que permite realizar buscas de livros no banco de dados com base em termos fornecidos pelo usuário. Ele aceita um parâmetro termo na URL, que pode corresponder ao título, autor ou descrição de um livro. A consulta SQL filtra os livros que contêm o termo buscado em qualquer um desses campos.

Após executar a consulta, os resultados são formatados em uma lista de dicionários contendo informações sobre cada livro (título, autor, ISBN, código do exemplar, descrição e link da foto). Por fim, a função retorna os dados em formato JSON com o

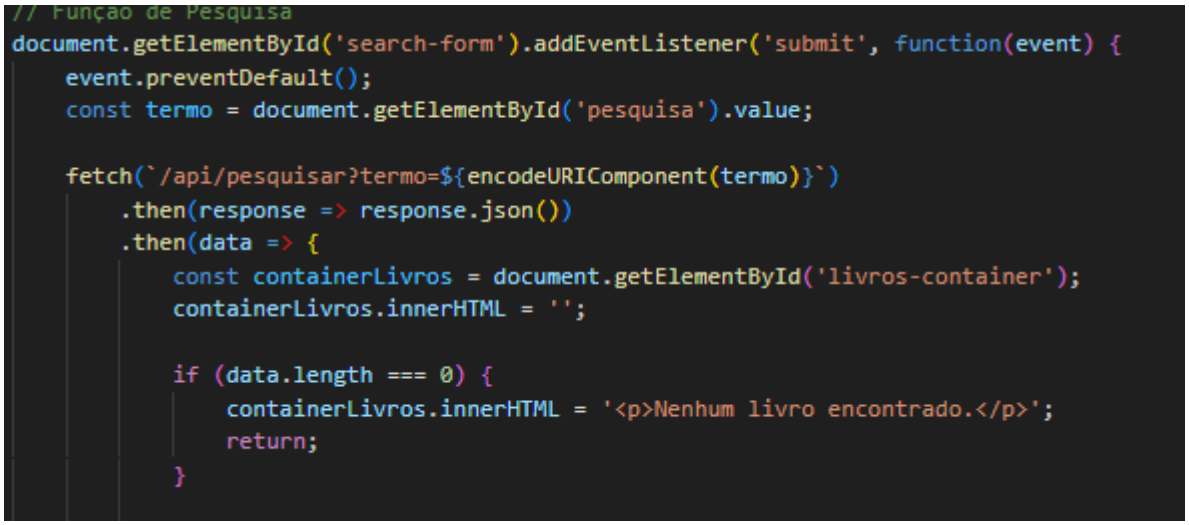
código de status HTTP 200.



```python
@app.route("/api/pesquisar")
def pesquisa_livro():

    autor=request.args.get('termo')
    termo = request.args.get("termo")
    titulo = request.args.get("termo")


    mydb = Conexao.conectar()
    mycursor = mydb.cursor()

    sql = f"SELECT  titulo, autor,ISBN, cod_exemplar, descricao, foto FROM tb_livro WHERE descricao like '%{termo}%'  or autor like '%{autor}%' or titulo like '%{titulo}%';"

    mycursor.execute(sql)
    resultado = mycursor.fetchall()
    mydb.close()

    livros=[]

    for livro in resultado:
        livros.append({
            "titulo": livro[0],
            "autor": livro[1],
            "ISBN": livro[2],
            "cod_exemplar": livro[3],
            "descricao": livro[4],
            "foto": livro[5]
        })
    return jsonify(livros), 200
```

Javascript:

Este código implementa a funcionalidade de busca de livros em uma página web. Ele intercepta a submissão do formulário de pesquisa, impede seu comportamento padrão e captura o termo digitado no campo de entrada. Em seguida, faz uma requisição à API (/api/pesquisar) enviando o termo como parâmetro na URL. Quando a resposta é recebida, o código processa os dados retornados e exibe os resultados no contêiner designado na página. Se nenhum livro for encontrado, uma mensagem indicando isso é exibida.



```javascript
// Função de Pesquisa
document.getElementById('search-form').addEventListener('submit', function(event) {
    event.preventDefault();
    const termo = document.getElementById('pesquisa').value;

    fetch(`/api/pesquisar?termo=${encodeURIComponent(termo)}`)
        .then(response => response.json())
        .then(data => {
            const containerLivros = document.getElementById('livros-container');
            containerLivros.innerHTML = '';

            if (data.length === 0) {
                containerLivros.innerHTML = '<p>Nenhum livro encontrado.</p>';
                return;
            }
```

APÊNDICE H – RF008- Categoria do livro

HTML:

Este código cria uma página web que exibe uma lista de livros pertencentes a uma categoria específica. Ao carregar a página, o título da página será dinâmico,

exibindo o nome da categoria de livros. Dentro do corpo da página, há uma lista de livros disponíveis, onde para cada livro são mostrados a capa, o título e o autor.

```
<section>
    <div class="categoria">
        Categoria
        <ul>
            <li><a href="/acervo">Acervo</a></li>
            <li><a href="/categoria?categoria=Romance">Romance</a></li>
            <li><a href="/categoria?categoria=Terror">Terror</a></li>
            <li><a href="/categoria?categoria=Ficcao">Ficção</a></li>
            <li><a href="/categoria?categoria=Fantasia">Fantasia</a></li>
            <li><a href="/categoria?categoria=Suspense">Suspense</a></li>
            <li><a href="/categoria?categoria=Religion">Religião</a></li>
            <li><a href="/categoria?categoria=Estudos">Estudos</a></li>
            <!-- Outras categorias -->
        </ul>
    </div>
</section>
```

CSS:

Este código CSS define o estilo visual de um menu suspenso para categorias em uma página web. A classe .categoria é usada para estilizar o elemento principal que contém o nome de uma categoria. Ele tem um fundo marrom escuro (#421b1a), com texto em um tom claro (#f0e6d2), bordas arredondadas e um padding para espaçamento interno. O elemento possui um cursor de ponteiro para indicar que é

clicável.

```css
/* Categoria */
/* Estilo do botão de Categoria */
.categoria {
    background-color: #421b1a;
    color: #f0e6d2;
    width: 200px;
    padding: 10px;
    border-radius: 10px;
    text-align: center;

    cursor: pointer;
    position: relative;
}

.categoria ul {
    list-style-type: none;
    padding: 0;
    margin: 0;
    display: none; /* Oculta as categorias inicialmente */
    position: absolute; /* Faz a lista aparecer sobre outros elementos */
    top: 100%;
    left: 0;
    background-color: #421b1a;
    border-radius: 10px;
    z-index: 1;
    width: 220px; /* Mesmo tamanho do botão */
}

.categoria:hover ul {
    display: block; /* Exibe as categorias ao passar o mouse */
}

.categoria ul li {
    padding: 10px;
    text-align: left;
```

Python:

Este código define uma rota no Flask, /categoria, que exibe livros de uma categoria específica solicitada pelo usuário.

```python
@app.route("/categoria")
def livros_por_categoria():
    nome_categoria = request.args.get("categoria")
    print(f"Acessando a categoria: {nome_categoria}")  # Verificar se a rota é acessada
    mydb = Conexao.conectar()
    if mydb is None:
        flash("Erro na conexão com o banco de dados.", "error")
        return render_template("acervo.html")

    mycursor = mydb.cursor()
    query = "SELECT ISBN,titulo, autor, foto, descricao FROM tb_livro WHERE categoria = %s"
    mycursor.execute(query, (nome_categoria,))
    resultado = mycursor.fetchall()
    mydb.close()

    lista_livros = []
    for livro in resultado:
        lista_livros.append({
            "ISBN":livro[0],
            "titulo": livro[1],
            "autor": livro[2],
            "foto": livro[3],
            "descricao": livro[4]
        })

    return render_template('acervo.html', lista_livros=lista_livros)
```

Banco de dados:

A tabela tb_categoria é usada para armazenar categorias de livros, identificadas por um código único (cod_categoria), e o nome da categoria (nome_categoria).

```sql
CREATE TABLE tb_categoria (
    cod_categoria INT AUTO_INCREMENT PRIMARY KEY,
    nome_categoria VARCHAR(100) NOT NULL
);
```

APÊNDICE I – RF009- Comentários sobre o livro

HTML:

Este código cria uma seção de comentários interativos para um site, permitindo que usuários logados possam deixar comentários e avaliar um conteúdo, como um livro, utilizando estrelas.

```
<div id="comentarios">
    <h3>Comentários</h3>
    <div id="lista-comentarios">
        <!-- Comentários serão carregados aqui via JavaScript -->
    </div>
</div>

{% if session['usuario_logado'] %}
<div id="form-comentario">
    <h4>Deixe seu comentário:</h4>
    <textarea id="comentario-texto" placeholder="Escreva seu comentário"></textarea>
    <div id="avaliacao">
        <span class="estrela" data-valor="1">&#9733;</span>
        <span class="estrela" data-valor="2">&#9733;</span>
        <span class="estrela" data-valor="3">&#9733;</span>
        <span class="estrela" data-valor="4">&#9733;</span>
        <span class="estrela" data-valor="5">&#9733;</span>
    </div>
    <button onclick="enviarComentario()">Enviar</button>
</div>
{% else %}
<p>Você precisa estar logado para comentar.</p>
{% endif %}
```

CSS:

Este código CSS define o estilo de uma seção de comentários em uma página web, com um design limpo e interativo. A área de comentários possui um fundo branco, bordas arredondadas e sombra suave para se destacar na página. O título e a lista de comentários são organizados, com os comentários exibidos em caixas com fundo cinza e bordas arredondadas. O formulário de envio de comentários tem um campo de texto, uma avaliação por estrelas e um botão de envio, que muda de cor ao passar o mouse. As estrelas de avaliação são inicialmente cinzas e se tornam douradas quando interagidas. O design foca em uma interface amigável e visualmente agradável para os usuários.

```css
#comentarios {
    background-color: #ffffff;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    margin: 20px auto;
    max-width: 1200px;
    border-radius: 8px;
    padding: 20px;
}

#comentarios h3 {
    font-size: 1.6em;
    color: #421b1a;

    margin-bottom: 15px;
}

#lista-comentarios {
    display: flex;
    flex-direction: column;
    gap: 15px;
}

#lista-comentarios div {
    background-color: #f4f4f4;
    border-radius: 8px;
    padding: 15px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

#lista-comentarios strong {
    color: #98502b;
    font-size: 1.1em;

}

#lista-comentarios p {
    font-size: 1rem;
    color: #555;
    margin-top: 5px;
}
```

```css
#form-comentario {
    margin-top: 20px;
    background-color: #f4f4f4;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

#form-comentario h4 {
    color: #421b1a;
    font-size: 1.4em;

    margin-bottom: 10px;
}

#form-comentario textarea {
    width: 100%;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 5px;
    font-size: 1rem;
    color: #333;
    margin-bottom: 10px;
}

#form-comentario select {
    padding: 5px;
    font-size: 1rem;
    border-radius: 5px;
    border: 1px solid #ddd;
    margin-bottom: 10px;
    color: #555;
}

#form-comentario button {
    display: inline-block;
    padding: 10px 20px;
    text-align: center;
    background-color: #98502b;
    color: #f0e6d2;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
    text-transform: uppercase;

    letter-spacing: 1px;
}

#form-comentario button:hover {
    background-color: #421b1a;
}
```

```css
#avaliacao {
    display: flex;
    gap: 5px;
    margin-bottom: 10px;
}

.estrela {
    font-size: 2rem;
    color: #ddd;
    cursor: pointer;
    transition: color 0.3s;
}

.estrela.ativa,
.estrela:hover,
.estrela:hover ~ .estrela {
    color: #f5c518;
}
```

Python:

As duas rotas fornecem funcionalidades para gerenciar comentários sobre livros. A rota /comentarios/<isbn> (GET) retorna uma lista de comentários e notas de usuários sobre um livro específico, identificado pelo ISBN. Já a rota /comentar/<isbn> (POST) permite que um usuário logado envie um comentário e nota para um livro, verificando se todos os campos estão preenchidos e se o usuário está autenticado antes de salvar o comentário no banco de dados. Ambas as rotas interagem com o banco de dados para inserir e exibir informações de forma dinâmica.

```python
# Rota para listar comentários de um livro específico usando o ISBN
@app.route('/comentarios/<isbn>', methods=['GET'])
def listar_comentarios(isbn):
    mydb = Conexao.conectar()
    mycursor = mydb.cursor()

    # Buscar comentários relacionados ao ISBN
    mycursor.execute("SELECT tb_cliente.nome, tb_comentario.conteudo, tb_comentario.nota FROM tb_comentario "
                     "JOIN tb_cliente ON tb_comentario.cpf = tb_cliente.cpf "
                     "WHERE tb_comentario.ISBN = %s", (isbn,))
    comentarios = mycursor.fetchall()
    mydb.close()

    lista_comentarios = [
        {'nome': c[0], 'conteudo': c[1], 'nota': c[2]} for c in comentarios
    ]

    return jsonify(lista_comentarios)
```

```python
# Rota para enviar comentário usando o ISBN
@app.route('/comentar/<isbn>', methods=['POST'])
def comentar(isbn):
    if 'usuario_logado' not in session:
        return jsonify({'mensagem': 'Você precisa estar logado para comentar.'}), 401

    dados = request.get_json()
    conteudo = dados.get('conteudo')
    nota = dados.get('nota')
    cpf_cliente = session['usuario_logado']['cpf']

    if not conteudo or not nota:
        return jsonify({'mensagem': 'Todos os campos são obrigatórios.'}), 400

    mydb = Conexao.conectar()
    mycursor = mydb.cursor()
    try:
        # Inserir o comentário associado ao ISBN
        sql = "INSERT INTO tb_comentario (conteudo, cpf, nota, ISBN) VALUES (%s, %s, %s, %s)"
        mycursor.execute(sql, (conteudo, cpf_cliente, nota, isbn))
        mydb.commit()
        return jsonify({'mensagem': 'Comentário adicionado com sucesso!'}), 201
    except Exception as e:
        mydb.rollback()
        return jsonify({'mensagem': f'Erro ao adicionar comentário: {e}'}), 500
    finally:
        mydb.close()
```

Javascript:

Este código lida com a exibição e envio de comentários para um livro específico. Quando a página é carregada, ele faz uma requisição para obter os comentários já existentes sobre o livro, exibindo-os na página. Também permite que os usuários enviem novos comentários, incluindo uma avaliação por estrelas. Quando o usuário clica nas estrelas, a nota é registrada e, ao enviar o comentário, ele é enviado via requisição POST para o servidor. O código também exibe uma mensagem de

sucesso ou erro após o envio e recarrega os comentários na página.

```
<script>
    document.addEventListener("DOMContentLoaded", function() {
   const isbn = "{{ dicionario_livro['ISBN'] }}";

   // Função para carregar comentários
   function carregarComentarios() {
       fetch(`/comentarios/${isbn}`)
           .then(response => response.json())
           .then(data => {
               const listaComentarios = document.getElementById("lista-comentarios");
               listaComentarios.innerHTML = "";
               data.forEach(comentario => {
                   const divComentario = document.createElement("div");
                   divComentario.innerHTML = `<strong>Usuário:</strong> ${comentario.nome}<br>
                                              <strong>Nota:</strong> ${comentario.nota} estrelas<br>
                                              <p>${comentario.conteudo}</p><hr>`;
                   listaComentarios.appendChild(divComentario);
               });
           });
   }
```

```
// Função para enviar comentário
window.enviarComentario = function() {
    const conteudo = document.getElementById("comentario-texto").value;
    const estrelas = document.querySelectorAll(".estrela.ativa");
    const nota = estrelas.length; // Conta quantas estrelas foram selecionadas

    if (!nota) {
        Swal.fire("Por favor, selecione uma nota.");
        return;
    }

    fetch(`/comentar/${isbn}`, {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            conteudo: conteudo,
            nota: nota
        })
    })
    .then(response => response.json())
    .then(data => {
        Swal.fire(data.mensagem);
        carregarComentarios();
    })
    .catch(error => console.error("Erro ao enviar comentário:", error));
};

// Sistema de avaliação por estrelas
let estrelas = document.querySelectorAll(".estrela");
estrelas.forEach(estrela => {
    estrela.addEventListener("click", function() {
        const valor = parseInt(this.getAttribute("data-valor"));
        estrelas.forEach(e => e.classList.remove("ativa"));
        for (let i = 0; i < valor; i++) {
            estrelas[i].classList.add("ativa");
        }
    });
});
```

```
    carregarComentarios();
});




</script>
```

Banco de dados:

Este código SQL cria uma tabela chamada tb_comentario no banco de dados, destinada a armazenar comentários feitos por clientes sobre livros. A tabela contém

as colunas: cod_comentario (identificador único do comentário), conteudo (texto do comentário), cpf (CPF do cliente que fez o comentário, que é uma chave estrangeira referenciando a tabela tb_cliente), nota (avaliação do livro em formato numérico), cod_exemplar (identificador do exemplar do livro comentado) e ISBN (código do livro, para associar o comentário a um livro específico). A tabela também define a chave primária como cod_comentario e estabelece uma relação com a tabela tb_cliente por meio da chave estrangeira cpf.

```
CREATE TABLE tb_comentario (
    cod_comentario INT AUTO_INCREMENT PRIMARY KEY,
    conteudo TEXT,
    cpf VARCHAR(11),
    nota INT,
    cod_exemplar INT,
    ISBN VARCHAR(20),
    FOREIGN KEY (cpf) REFERENCES tb_cliente(cpf)
);
```

APÊNDICE J – RF010- Mapa

HTML:

Este código HTML exibe uma mensagem de notificação na página, informando ao usuário que ele precisa se cadastrar em um local físico. A mensagem inclui o endereço do local e um botão que, ao ser clicado, redireciona o usuário para o Google Maps, mostrando a localização do endereço fornecido.

```
<main>
    <div class="notification">
        <h1>Atenção!</h1>
        <p>Você precisa ir ao local para se cadastrar.</p>
        <p><strong>Endereço:</strong> R. Hugo Negrini, 60 - Vila Bela Vista, Araraquara - SP</p>
        <button onclick="window.location.href='https://www.google.com/maps/place/R.+Hugo+Negrini,+60+-+Vila+Bela+Vista,+Araraquara+-+SP,+14800-028/
    </div>
</main>
```

CSS:

Este código CSS estiliza o corpo de uma página e a notificação exibida nela. O corpo da página tem um fundo cinza claro, com conteúdo centralizado tanto vertical quanto horizontalmente. A notificação é estilizada com um fundo branco, borda vermelha e arredondada, com um leve sombreamento para dar um efeito de destaque. O texto dentro da notificação é alinhado ao centro. O título tem a cor vermelha e o texto do parágrafo é escuro, enquanto o botão é estilizado com um

fundo vermelho, texto branco, e bordas arredondadas. O botão também possui um efeito de transição de cor ao ser hoverizado.

```css
body {

    background-color: #f4f4f4;
    margin: 0;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
}


.notification {
    background-color: #fff;
    border: 1px solid #421b1a;
    border-radius: 8px;
    padding: 20px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    text-align: center;
    margin: 20px auto;
    max-width: 400px;
}

h1 {
    color: #421b1a;
}

p {
    color: #333;
}

button {
    background-color: #421b1a;
    border: none;
    border-radius: 5px;
    color: #fff;
    padding: 10px 20px;
    cursor: pointer;
    transition: background-color 0.3s;
}
```

```css
button:hover {
    background-color: #421b1a;
}
```

Python:

Este código define uma rota no Flask, responsável por exibir uma página de notificação de cadastro. Quando o usuário acessa a URL "/notificacao_cadastro"

usando os métodos GET ou POST, o servidor renderiza e retorna o template "notificacao_cadastro_cliente.html", que provavelmente contém a notificação informando o usuário sobre a necessidade de realizar um cadastro.

```python
@app.route("/notificacao_cadastro", methods=["GET", "POST"])
def pagina_notificacao_cadastro():
    return render_template("notificacao_cadastro_cliente.html")
```

APÊNDICE K – RF011- Perfil

HTML:

Este código define que se o usuário estiver logado (verificado pela existência de 'usuario_logado' na sessão), ele exibirá as informações do perfil do usuário, como nome, email e telefone. Além disso, se o usuário tiver reservas, uma seção listando essas reservas será mostrada, incluindo detalhes como título do livro, foto do livro, código da reserva, data da reserva, retirada, devolução e entrega. Caso o usuário não tenha reservas, será exibida uma mensagem informando isso. Se o usuário não estiver logado ou houver algum erro ao carregar as informações, uma mensagem de erro será exibida.

```
<main class="profile-container">
    {% if 'usuario_logado' in session %}
        <!-- Se o usuário estiver logado, exibe as informações do perfil -->
        <section class="profile-info">
            <h2>Perfil de {{ session['usuario_logado']['nome'] }}</h2> <!-- Nome do usuário -->
            <p><strong>Email:</strong> {{ session['usuario_logado']['email'] }}</p> <!-- Email -->
            <p><strong>Telefone:</strong> {{ session['usuario_logado']['telefone'] }}</p> <!-- Telefone -->
        </section>

        <!-- Se o usuário tiver reservas, exibe a seção de reservas -->
        <section class="reservas-info">
            <h3>Minhas Reservas</h3>
            {% if user_data %}
                <div class="reservas-list">
                    {% for reserva in user_data %}
                        <div class="reserva-card">
                            <div class="reserva-info">
                                <h4>{{ reserva[9] }}</h4> <!-- Título do Livro -->
                                <img src="{{ reserva[10] }}" alt="Foto do Livro" class="book-photo"> <!-- Foto do Livro -->
                            </div>
                            <div class="reserva-details">
                                <p><strong>Código da Reserva:</strong> {{ reserva[3] }}</p> <!-- Código da Reserva -->
                                <p><strong>Código do Exemplar:</strong> {{ reserva[4] }}</p> <!-- Código do Exemplar -->
                                <p><strong>Data da Reserva:</strong> {{ reserva[5] }}</p> <!-- Data da Reserva -->
                                <p><strong>Data de Retirada:</strong> {{ reserva[6] }}</p> <!-- Data de Retirada -->
                                <p><strong>Data de Devolução:</strong> {{ reserva[7] }}</p> <!-- Data de Devolução -->
                                <p><strong>Data de Entrega:</strong> {{ reserva[8] }}</p> <!-- Data de Entrega -->
                            </div>
                        </div>
                    {% endfor %}
                </div>
            {% else %}
                <p>Não há reservas para este usuário.</p> <!-- Caso o usuário não tenha reservas -->
            {% endif %}
        </section>
    {% else %}
        <p>Não foi possível carregar as informações do perfil. Por favor, tente novamente mais tarde.</p>
    {% endif %}
</main>
```

CSS:

Este código CSS estiliza uma página de perfil de usuário, com seções para informações pessoais e reservas. O cabeçalho tem uma cor escura e efeitos de destaque nos links. O perfil do usuário é exibido de forma centralizada, com bordas arredondadas e sombras.

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Corpo da página */
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    color: #333;
    padding: 20px;
    line-height: 1.6;
    overflow-x: hidden;
}

/* Cabeçalho */
header {
    background-color: #421b1a;
    color: #fff;
    padding: 20px 0;
    text-align: center;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.15);
    position: sticky;
    top: 0;
    z-index: 10;
}

header nav a {
    color: #fff;
    text-decoration: none;
    margin: 0 15px;
    font-size: 18px;
    padding: 5px 10px;
    border-radius: 5px;
    transition: background-color 0.3s, transform 0.3s;
}
```

```css
header nav a:hover {
    background-color: #5a2523;
    transform: scale(1.1);
}

/* Container do perfil */
.profile-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    min-height: 60vh;
    gap: 20px;
    animation: fadeIn 1s ease-in-out;
}

@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

/* Estilo da seção de informações do perfil */
.profile-info {
    background-color: #fff;
    border-radius: 10px;
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
    padding: 30px;
    width: 100%;
    max-width: 600px;
    text-align: center;
    border: 2px solid transparent;
    transition: border-color 0.3s, transform 0.3s, box-shadow 0.3s;
}
```

```css
.profile-info:hover {
    border-color: #421b1a;
    transform: translateY(-5px);
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
}

.profile-info h2 {
    font-size: 26px;
    color: #421b1a;
    margin-bottom: 20px;
}

.profile-info p {
    font-size: 18px;
    margin-bottom: 10px;
}

.profile-info p strong {
    color: #421b1a;
}

/* Seção de Reservas */
.reservas-info {
    width: 100%;
    max-width: 900px;
    margin-top: 40px;
    text-align: center;
    animation: fadeIn 1.2s ease-in-out;
}

.reservas-info h3 {
    font-size: 28px;
    color: #421b1a;
    margin-bottom: 20px;
}
```

```css
/* Lista de reservas */
.reservas-list {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    gap: 20px;
}

/* Card de reserva */
.reserva-card {
    background-color: #fff;
    border-radius: 10px;
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 400px;
    padding: 20px;
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: left;
    transition: transform 0.3s, box-shadow 0.3s;
    border: 2px solid transparent;
}

.reserva-card:hover {
    transform: scale(1.02);
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);
    border-color: #ffffff;
}

.reserva-info {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-bottom: 15px;
}
```

```css
.reserva-info h4 {
    font-size: 22px;
    color: #421b1a;
    margin-bottom: 10px;
}

.book-photo {
    width: 120px;
    height: 180px;
    object-fit: cover;
    border-radius: 10px;
    border: 1px solid #ddd;
    transition: transform 0.3s;
}

.book-photo:hover {
    transform: rotate(-2deg) scale(1.05); /* Efeito de destaque */
}

.reserva-details {
    font-size: 16px;
    color: #555;
    text-align: center;
}

.reserva-details p {
    margin-bottom: 8px;
}

/* Mensagens de erro ou informações */
.reservas-info p {
    font-size: 18px;
    color: black;
    margin-top: 20px;
}
```

Python:

A função get_cliente_data(cpf) busca dados de um cliente e suas reservas no banco de dados. Ela conecta ao banco, executa uma consulta SQL entre as tabelas de cliente, reserva e livro, para obter informações como nome, email, telefone do cliente, e detalhes das reservas (códigos, datas e título/foto dos livros). A consulta é filtrada pelo CPF do cliente e retorna todos os resultados de reservas encontradas para esse cliente.

```
def get_cliente_data(cpf):
    # Estabelece a conexão com o banco de dados
    db_connection = Conexao.conectar()
    cursor = db_connection.cursor()

    # Executa a consulta SQL com o INNER JOIN para obter os dados do cliente e suas reservas
    cursor.execute("""
        SELECT
    c.nome,
    c.email,
    c.telefone,
    r.cod_reserva,
    r.cod_exemplar,
    r.data_reserva,
    r.data_retirada,
    r.data_devolucao,
    r.data_entrega,
    l.titulo,            -- Título do livro
    l.foto               -- Foto do livro
FROM
    tb_reserva r
INNER JOIN
    tb_cliente c ON r.cpf = c.cpf
INNER JOIN
    tb_livro l ON r.ISBN = l.ISBN
WHERE
    r.cpf = %s
    """, (cpf,))

    # Obtém todos os resultados, no caso de haver múltiplas reservas
    data = cursor.fetchall()

    return data
```

APÊNDICE L – RF012- Trocar senha

HTML:

Este código HTML cria uma página com um formulário para trocar a senha de um usuário. O formulário solicita o CPF, a senha atual e a nova senha do usuário, e os dados são enviados via método POST para a rota "/trocar_senha" no servidor. O formulário inclui validação para garantir que todos os campos sejam preenchidos antes do envio.

```
<body>
    <div class="container">
    <h2>Trocar Senha</h2>
    <form action="/trocar_senha" method="POST">
        <div class="form-group">
            <label for="cpf">CPF:</label>
            <input type="text" id="cpf" name="cpf" required>
        </div>
        <div class="form-group">
            <label for="senha_atual">Senha Atual:</label>
            <input type="password" id="senha_atual" name="senha_atual" required>
        </div>
        <div class="form-group">
            <label for="nova_senha">Nova Senha:</label>
            <input type="password" id="nova_senha" name="nova_senha" required>
        </div>
        <div class="form-group">
           <button type="submit">Trocar Senha</button>
        </div>
    </form>
</div>

</body>
```

CSS:

Este código CSS estiliza o formulário de troca de senha, fornecendo uma aparência limpa e moderna. Ele define uma estrutura centralizada, com o formulário dentro de um container de fundo branco, bordas arredondadas e sombra suave. Os campos de entrada e o botão são estilizados com bordas e transições para efeitos de foco e hover.

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;

}

body {
    background-color: #f4f4f4;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background-color: #fff;
    padding: 2rem;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    max-width: 400px;
    width: 100%;
}

h2 {
    color: #421b1a;
    margin-bottom: 1rem;
    text-align: center;
}

.form-group {
    margin-bottom: 1.5rem;
}

label {
    color: #6B8E23;
    font-weight: bold;
    display: block;
    margin-bottom: 0.5rem;
}
```

```css
input[type="text"],
input[type="password"] {
    width: 100%;
    padding: 0.5rem;
    border: 1px solid #ddd;
    border-radius: 4px;
    outline: none;
    transition: border-color 0.3s;
}

input[type="text"]:focus,
input[type="password"]:focus {
    border-color: #6B8E23;
}

button {
    width: 100%;
    padding: 0.75rem;
    background-color: #421b1a;
    color: #fff;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 1rem;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #6B8E23;
}

body {

    background-color: #f4f4f4;
}
```

```css
.main-content {
    width: 300px;
    margin: auto;
    padding: 20px;
    background: white;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h2 {
    text-align: center;
}

form {
    display: flex;
    flex-direction: column;
}

label {
    margin-bottom: 5px;
}

input {
    margin-bottom: 15px;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

button {
    padding: 10px;
    border: none;
    border-radius: 4px;
    background: #6B8E23;
    color: white;
    cursor: pointer;
}
```

```css
button:hover {
    background: #4f6d1b;

}
```

Python:

Este código Python, utilizando o Flask, define a rota /trocar_senha que gerencia tanto as requisições GET quanto POST. Quando a requisição POST é recebida, ele verifica se os campos "cpf", "senha_atual" e "nova_senha" foram preenchidos. Caso contrário, um erro é disparado e o usuário é redirecionado de volta para a página de troca de senha.

Se os campos forem preenchidos corretamente, a função trocar_senha da classe Usuario é chamada para validar e atualizar a senha. Se a senha for alterada com

sucesso, uma mensagem de sucesso é exibida e o usuário é redirecionado para a página inicial. Caso contrário, se o CPF ou a senha atual estiverem incorretos, uma mensagem de erro é exibida e o usuário é redirecionado para a página de atualização de senha.

```python
@app.route("/trocar_senha", methods=["POST", "GET"])
def trocar_senha_endpoint():
    cpf = request.form.get("cpf")
    senha_atual = request.form.get("senha_atual")
    nova_senha = request.form.get("nova_senha")

    if not cpf or not senha_atual or not nova_senha:
        flash("Todos os campos são obrigatórios.", "error")
        return redirect("/trocar_senha")

    usuario=Usuario()

    if usuario.trocar_senha(cpf, senha_atual, nova_senha):
        flash("Senha trocada com sucesso!", "success")
        return redirect("/")
    else:
        flash("CPF ou senha atual incorretos.", "error")
        return redirect("/atualizar_senha")
```

APÊNDICE M – RF013- Sair da conta

HTML:

Este código cria um formulário para o botão de "Sair" em um site. Quando o usuário clica no botão "Sair", o navegador pede uma confirmação antes de prosseguir. A função confirmarLogout() é chamada quando o formulário é enviado, e ela é responsável por mostrar uma mensagem de perguntando se o usuário tem certeza de que quer sair. Se o usuário confirmar, o formulário será enviado para a URL "/logout", desconectando da sessão. Caso contrário, o processo de encerramento é cancelado e o usuário permanece conectado.

```html
<form id="logout-form" action="/logout" method="post" onsubmit="return confirmarLogout()">
    <button type="submit"class="sair" onclick="confirmarLogout()">Sair</button>
</form>
```

CSS:

Este código CSS define o estilo de um botão com a classe .sair, criando um visual limpo e interativo. O botão tem fundo transparente, texto branco, borda fina e arredondada, e transições suaves. Ao passar o mouse, o fundo se torna levemente transparente e a borda fica branca, oferecendo um efeito visual atraente e dinâmico. Além disso, o cursor muda indicando que o botão é clicável.

```css
.sair {
    background: none;
    color: #fff;
    border: 1px solid rgba(255, 255, 255, 0.5);
    padding: 8px 15px;
    font-size: 14px;
    font-family: "Open Sans", sans-serif;
    border-radius: 4px;
    cursor: pointer;
    transition: all 0.2s ease-in-out;
}

.sair {
    background-color: rgba(255, 255, 255, 0.2);
    color: #fff;
    border-color: #fff;
}
```

Python:

Este código define a rota "/logout" em uma aplicação Flask. Quando o usuário faz uma requisição POST para essa rota, a função de se desconectar é executada. Ela verifica se há uma sessão ativa de um administrador (usuarioadm_logado) ou de um usuário comum (usuario_logado). Se existirem, essas sessões são removidas. Após encerrada, uma mensagem de sucesso é exibida e o usuário é redirecionado para a página inicial ("/").

```python
@app.route("/logout", methods=["POST"])
def logout():
    # Logout conforme descrito anteriormente
    if "usuarioadm_logado" in session:
        session.pop("usuarioadm_logado", None)
    if "usuario_logado" in session:
        session.pop("usuario_logado", None)

    flash("Logout realizado com sucesso!", "success")
    return redirect("/")
```