Projeto Biblioteca Softwaresul

Documento de Arquitetura

Versão 0.1

Histórico da Revisão

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 23/06/2023 | 0.1 | Documento Inicial | Jean Miotto |

Sumário

[1. Introdução 4](#_Toc138350726)

[1. Objetivo 4](#_Toc138350727)

[2. Escopo do projeto 4](#_Toc138350728)

[2. Arquitetura do Banco de Dados 4](#_Toc138350729)

[1. Diagrama de Entidade Relacionamento – Sistema 4](#_Toc138350730)

[3. Arquitetura do Back-End 5](#_Toc138350731)

[4. API 6](#_Toc138350732)

[1. Login 6](#_Toc138350733)

[2. Usuário 6](#_Toc138350734)

[Listar 6](#_Toc138350735)

[Inserir 6](#_Toc138350736)

[Atualizar 6](#_Toc138350737)

[Excluir 6](#_Toc138350738)

[Abrir por ID 6](#_Toc138350739)

[3. Livro 6](#_Toc138350740)

[Listar 6](#_Toc138350741)

[Inserir 6](#_Toc138350742)

[Atualizar 6](#_Toc138350743)

[Excluir 6](#_Toc138350744)

[Abrir por ID 6](#_Toc138350745)

[4. Empréstimo 6](#_Toc138350746)

[Listar 6](#_Toc138350747)

[Inserir 6](#_Toc138350748)

[Atualizar 6](#_Toc138350749)

[Excluir 6](#_Toc138350750)

[Abrir por ID 6](#_Toc138350751)

[5. Arquitetura do Front-End 7](#_Toc138350752)

[1. Layouts 7](#_Toc138350753)

[Cadastros 7](#_Toc138350754)

[Novo Item / Alterar Item 8](#_Toc138350755)

[Dashboard 8](#_Toc138350756)

[2. Telas 9](#_Toc138350757)

[6. Integração entre Back-End e Front-End 9](#_Toc138350758)

# Introdução

## Objetivo

Este projeto possui como objetivo o desenvolvimento de uma aplicação web que será responsável pelo gerenciamento da biblioteca da Softwaresul Dev (filial de Toledo-PR).

A aplicação deve contar com um mecanismo de gerenciamento de empréstimos dos ativos da biblioteca, garantindo a acuracidade dos itens disponíveis e a rastreabilidade dos itens retirados para empréstimo.

## Escopo do projeto

* Cadastros: o sistema deve permitir o cadastro e manutenção de usuários, livros e agendamentos de empréstimos.
* Dashboard: o sistema deve apresentar um dashboard com as informações mais relevantes, como uma listagem dos empréstimos ativos, total de empréstimos efetuados no ano e empréstimos cujos prazos de devoluções estejam expirados.  
  Também deve conter uma listagem dos últimos 05 empréstimos finalizados e com os empréstimos ativos/pendentes.

# Arquitetura do Banco de Dados

## Diagrama de Entidade Relacionamento – Sistema

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

# Arquitetura do Back-End

Este Back-end utiliza uma variedade de tecnologias para construir aplicativos web escaláveis e robustos. Node é a plataforma de desenvolvimento de software escolhida, que é construída com base no interpretador V8 do Google Chrome e permite o desenvolvimento de aplicativos com entrada/saída de alta intensidade. O framework Express é utilizado para construir aplicativos web e APIs, permitindo que as rotas sejam configuradas e o middleware seja gerenciado.

O construtor de consultas SQL Knex é utilizado para suportar múltiplos bancos de dados, incluindo MySQL, fornecendo uma maneira fácil de escrever consultas SQL e se integra facilmente com o Express. A biblioteca Joi é utilizada para validar dados recebidos por meio de solicitações HTTP, como campos de formulário e parâmetros de consulta.

O middleware Helmet ajuda a proteger aplicativos Express adicionando cabeçalhos HTTP de segurança, evitando ataques comuns.

O middleware Body-Parser analisa o corpo das solicitações HTTP e o transforma em um objeto JavaScript, suportando vários tipos de codificação, como JSON e URL-encod

A arquitetura deste Back-end segue o padrão Model-View-Controller (MVC) amplamente utilizado em aplicativos web. O modelo representa os dados e a lógica de negócios do aplicativo, a visualização apresenta os dados ao usuário e o controlador lida com as solicitações HTTP e gerencia a comunicação entre o modelo e a visualização.

Em resumo, este Back-end é altamente eficiente e escalável, utilizando várias tecnologias para torná-lo robusto e capaz de gerenciar diferentes tipos de solicitações e tarefas. O uso do padrão MVC garante que o aplicativo seja bem-organizado e facilmente manejável.

1. Node: Node.js é uma plataforma de desenvolvimento de software de código aberto construída com base no interpretador V8 do Google Chrome. É usado para construir aplicativos de rede escaláveis, pois permite o desenvolvimento de aplicativos com I/O de alta intensidade (entrada/saída).
2. Express: É um framework Node.js leve, flexível e minimalista que fornece recursos para construir aplicativos da web e APIs. Ele permite que você configure rotas para lidar com solicitações HTTP e use middleware para gerenciar solicitações e respostas.
3. Knex: É um construtor de consultas SQL para Node.js que suporta vários bancos de dados, incluindo MySQL. Ele fornece uma maneira fácil de escrever consultas SQL e se integra facilmente com o Express.
4. Joi: É uma biblioteca de validação de dados para o Node.js que é usada para validar dados recebidos por meio de solicitações HTTP. Ele pode ser usado para validar campos de formulário, parâmetros de consulta e outros dados de entrada.
5. Helmet: É um middleware que ajuda a proteger aplicativos Express adicionando vários cabeçalhos HTTP de segurança. Ele ajuda a evitar ataques XSS, injeção de conteúdo e outros ataques comuns.
6. Body-Parser: É um middleware que analisa o corpo de solicitações HTTP e o transforma em um objeto JavaScript para que possa ser manipulado facilmente. Ele suporta vários tipos de codificação, como JSON e URL-encoded.
7. MySQL: É um sistema de gerenciamento de banco de dados relacional de código aberto. Ele é amplamente utilizado para armazenar e recuperar dados em aplicativos da web e é suportado pelo Knex.

# API

## Login

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /login | POST |  |
| **Objeto de Entrada** | | |
| {  "email": "exemplo@email.com",  "senha": "senha123" } | | |
| **Objeto de Saída** | | |
| {  id: 1 } | | |
|  |  | **Status de Saída** |
|  |  | 200 |

## Usuário

### Listar

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /user | GET |  |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída - Array** | | |
| [  {  id,  name,  password,  email,  status,  obs  }  ] | | |
|  |  | **Status de Saída** |
|  |  | 200 |

### Inserir

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /user | POST |  |
| **Objeto de Entrada** | | |
| {  name,  password,  email,  status,  obs  } | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Atualizar

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /user/:id | PATCH | Id (INT) |
| **Objeto de Entrada** | | |
| {  id,  name,  password,  email,  status,  obs  } | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Excluir

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /user/:id | DELETE | Id (INT) |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Abrir por ID

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /user/:id | GET | Id (INT) |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída** | | |
| {  id,  name,  password,  email,  status,  obs  } | | |
|  |  | **Status de Saída** |
|  |  | 200 |

## Livro

### Listar

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /book | GET |  |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída - Array** | | |
| [  {  id,  name,  gender,  author,  quantity\_pages,  date\_acquisition,  status,  obs  }  ] | | |
|  |  | **Status de Saída** |
|  |  | 200 |

### Inserir

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /book | POST |  |
| **Objeto de Entrada** | | |
| {  name,  gender,  author,  quantity\_pages,  date\_acquisition,  status,  obs  } | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Atualizar

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /book/:id | PATCH | Id (INT) |
| **Objeto de Entrada** | | |
| {  id,  name,  gender,  author,  quantity\_pages,  date\_acquisition,  status,  obs  } | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Excluir

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /book/:id | DELETE | Id (INT) |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída - Array** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Abrir por ID

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /book/:id | GET | Id (INT) |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída - Array** | | |
| {  id,  name,  gender,  author,  quantity\_pages,  date\_acquisition,  status,  obs  } | | |
|  |  | **Status de Saída** |
|  |  | 200 |

## Empréstimo

### Listar

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /loan | GET |  |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída - Array** | | |
| [  {  id,  id\_user,  id\_book,  loan\_start,  loan\_end,  status,  obs  }  ] | | |
|  |  | **Status de Saída** |
|  |  | 200 |

### Inserir

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /loan | POST |  |
| **Objeto de Entrada** | | |
| {  id\_user,  id\_book,  loan\_start,  loan\_end,  status,  obs  } | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Atualizar

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /loan/:id | PATCH | Id (INT) |
| **Objeto de Entrada** | | |
| {  id,  id\_user,  id\_book,  loan\_start,  loan\_end,  status,  obs  } | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Excluir

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /loan/:id | DELETE | Id (INT) |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída** | | |
| NA | | |
|  |  | **Status de Saída** |
|  |  | 204 |

### Abrir por ID

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Método HTTP** | **Parâmetros de Entrada** |
| /loan/:id | GET | Id (INT) |
| **Objeto de Entrada** | | |
| NA | | |
| **Objeto de Saída - Array** | | |
| {  id,  id\_user,  id\_book,  loan\_start,  loan\_end,  status,  obs  } | | |
|  |  | **Status de Saída** |
|  |  | 200 |

# Arquitetura do Front-End

## Layouts

### Cadastros

Um layout de tela eficaz para exibir dados em uma tabela deve ser intuitivo, fácil de usar e deve oferecer opções de filtragem e pesquisa para ajudar o usuário a encontrar os dados desejados. Além disso, deve haver botões claros para adicionar, alterar e excluir os dados.

A tabela deve ser o elemento central da tela, com os dados organizados em colunas claramente rotuladas. Cada linha deve representar um único item de dados e as informações mais importantes devem ser destacadas na coluna principal.

Para ajudar o usuário a encontrar os dados desejados, o layout deve incluir opções de filtros e pesquisa. As opções de filtro devem estar disponíveis no topo da tabela, com opções para selecionar uma ou várias opções de filtragem, como datas, categorias ou outros critérios relevantes.

Os botões de adicionar, alterar e excluir devem ser claramente rotulados e localizados acima ou abaixo da tabela, em um local onde o usuário possa facilmente encontrá-los. O botão de adicionar deve abrir um formulário para inserir novos dados, enquanto o botão de alterar deve abrir um formulário preenchido com os dados existentes, permitindo que o usuário edite as informações. O botão de excluir deve abrir uma caixa de diálogo de confirmação para evitar a exclusão acidental de dados.

Com um design cuidadoso e bem planejado, é possível criar uma interface intuitiva que permite aos usuários visualizar e gerenciar facilmente grandes conjuntos de dados.

Objetivos:

* Exibir dados em uma tabela
* Oferecer opções de filtros e pesquisa
* Disponibilizar botões para adicionar, alterar e excluir informações

### Novo Item / Alterar Item

Uma tela de cadastro ou edição de itens deve ser simples, intuitiva e fácil de usar. Ela deve ser clara e permitir que o usuário insira ou edite informações com facilidade.

No topo da tela, deve haver um título claro e descritivo, indicando se o usuário está cadastrando um novo item ou editando um existente. Abaixo do título, deve haver um formulário que permita ao usuário inserir ou editar as informações necessárias.

O formulário deve ser organizado de forma lógica e ter rótulos claros para cada campo de entrada. Campos obrigatórios devem ser marcados com asteriscos e o usuário deve receber uma mensagem de erro clara caso não preencha esses campos.

Se a tela estiver sendo usada para edição de um item existente, o formulário deve ser preenchido com as informações existentes do item. Se for uma tela de cadastro, os campos do formulário devem estar vazios, prontos para que o usuário insira novas informações.

Para ajudar o usuário a visualizar os dados inseridos ou a serem editados, a tela deve incluir uma prévia do item, mostrando as informações inseridas até aquele momento. Essa prévia deve ser atualizada à medida que o usuário preenche o formulário.

A tela também deve incluir botões claros para salvar ou cancelar a ação atual. O botão de salvar deve estar em destaque e ser facilmente identificável, enquanto o botão de cancelar deve estar localizado próximo ao botão de salvar e permitir que o usuário desista da ação atual.

Por fim, a tela deve incluir mensagens claras para o usuário, indicando se a ação foi bem-sucedida ou se ocorreu algum erro. Essas mensagens devem ser exibidas de forma clara e destacada, ajudando o usuário a entender o que aconteceu e a tomar as medidas necessárias.

Objetivos

* Permitir o cadastro de um novo item ou a edição de um antigo
* Exibir formulário com campos para preenchimento de informações
* Oferecer opções para seleção de dados em dropdowns ou outros componentes

### Dashboard

Uma tela de dashboard deve permitir que o usuário visualize e acompanhe de forma rápida e fácil as principais informações e métricas do sistema ou aplicativo. Essa tela deve ser clara, concisa e apresentar as informações mais importantes de forma resumida.

Geralmente, uma tela de dashboard é dividida em seções ou widgets, cada um representando uma informação ou métrica importante.

Os widgets devem ser organizados de forma lógica e intuitiva, com títulos claros e gráficos ou ícones que facilitem a compreensão das informações apresentadas

Outra funcionalidade importante de uma tela de dashboard é permitir que o usuário filtre as informações apresentadas de acordo com suas necessidades. Isso pode ser feito por meio de opções de filtro ou seletores de período, permitindo que o usuário visualize as informações em um intervalo de tempo específico ou com determinados critérios.

Por fim, a tela de dashboard deve incluir uma navegação clara e fácil, permitindo que o usuário acesse outras partes do sistema ou aplicativo com facilidade. Isso pode ser feito por meio de menus de navegação ou botões de atalho, por exemplo.

Objetivos

* Apresentar informações e métricas importantes do sistema ou aplicativo
* Ser clara, concisa e apresentar informações de forma resumida
* Dividir a tela em seções ou widgets, cada um representando uma informação ou métrica importante
* Permitir que o usuário filtre as informações apresentadas de acordo com suas necessidades
* Incluir uma navegação clara e fácil para acesso a outras partes do sistema ou aplicativo.

## Telas

|  |  |  |
| --- | --- | --- |
| **Nome da Tela** | **Layout** | **Rotas Backend** |
| 1. Login | Próprio | /login |
| 1. Home | Dashboard |  |
| 1. Usuários | Listagem | GET: /user  DELETE: /user/:id |
| 1. Novo Usuário | Novo Item / Editar | POST: /user  PATCH: /user/:id |
| 1. Livros | Listagem | GET: /book  DELETE: /book/:id |
| 1. Novo Livro | Novo Item / Editar | POST: /book  PATCH: /book/:id |
| 1. Empréstimos | Listagem | GET: /loan  DELETE: /loan/:id |
| 1. Novo Empréstimo | Novo Item / Editar | POST: /loan  PATCH: /loan/:id |

# Integração entre Back-End e Front-End

Quando se trata de desenvolver aplicativos web, é essencial ter uma maneira de fazer com que o front-end se comunique com o back-end para enviar e receber dados. O Axios é uma biblioteca de cliente HTTP que facilita o envio de solicitações HTTP do front-end para o back-end.

A autenticação é uma parte crucial de muitos aplicativos web e o JSON Web Token (JWT) é uma maneira comum de gerenciar a autenticação e autorização de usuários. O JWT é um token criptografado que é enviado de volta ao cliente após a autenticação e é usado para verificar a identidade do usuário em solicitações subsequentes.

Para integrar o front-end com o back-end usando o Axios e o JWT, o primeiro passo é enviar uma solicitação de autenticação do front-end para o back-end. Este verifica as credenciais do usuário e se as informações de login estiverem corretas, um JWT é gerado e enviado de volta ao front-end. Esse JWT é então armazenado no local de armazenamento do cliente, como o armazenamento local do navegador.

Quando o usuário tenta acessar uma rota protegida do back-end, o JWT é enviado junto com a solicitação HTTP do front-end usando o Axios. O primeiro pode verificar o JWT para confirmar que o usuário está autenticado e tem permissão para acessar a rota protegida.

Para enviar solicitações HTTP usando o Axios, é necessário especificar a URL da API do back-end e o método HTTP a ser usado, como GET, POST, PATCH ou DELETE. Também é possível enviar parâmetros, cabeçalhos e corpo de solicitação juntamente com a solicitação HTTP.

Usando o Axios e o JWT, a integração do front-end com o back-end pode ser simplificada e segura.