

Documentação técnica

Innovators Ticket

Versão 1.0

2024

Autor: Innovators Ticket

ÍNDICE DETALHADO:

Início 1

INTRODUÇÃO 2

1.1. TEMA 2

1.2. OBJETIVO DO PROJETO 2

1.3. MÉTODO DE TRABALHO 3

1.4. ORGANIZAÇÃO DO TRABALHO 3

1.5. Membros do time e Papéis 4

DESCRIÇÃO GERAL DO SISTEMA 4

2.1. PROPÓSITO DO SISTEMA 4

2.2. OBJETIVOS 5

2.3. PÚBLICO-ALVO 5

2.4. BENEFÍCIOS

2.5. TECNOLOGIAS USADAS

2.7. AMBIENTE DE IMPLANTAÇÃO

2.8. INTEGRAÇÕES

2.9. POLÍTICAS DE PRIVACIDADE E TERMOS DE USO

REQUISITOS DO SISTEMA

3.1. REQUISITOS FUNCIONAIS

3.2. REQUISITOS NÃO-FUNCIONAIS

3.3. PROTÓTIPO

3.4. DIAGRAMA DE NAVEGAÇÃO

3.5. MÉTRICAS E CRONOGRAMA

ANÁLISE E Arquitetura do sistema

4.1. ARQUITETURA DO SISTEMA

4.2. AMBIENTE DE DESENVOLVIMENTO

4.4. TESTES

4.5 MANUAL DO USUÁRIO

5. CONCLUSÕES E CONSIDERAÇÕES FINAIS

6. BIBLIOGRAFIA

Introdução:

A Ticket Innovators é uma plataforma online desenvolvida para simplificar e otimizar o processo de compra e de ingressos para eventos diversos.

Este documento tem como objetivo fornecer uma visão geral sobre o desenvolvimento, funcionalidades e operações do sistema Ticket Innovators.

Através desta documentação, buscamos orientar e informar desenvolvedores, stakeholders e usuários sobre os aspectos essenciais da plataforma, como sua arquitetura, organização e desenvolvimento.

1.1. Tema:

- O tema deste documento é sobre o Projeto Innovators Ticket, uma plataforma online de venda de ingressos.

1.2. Objetivo do Projeto:

- O propósito principal deste projeto é criar uma plataforma de vendas de ingressos online que seja fácil de usar, segura e eficiente. Isso tem como objetivo simplificar tanto a compra de ingressos para uma variedade de eventos, como shows, eventos esportivos e exposições. Além disso, os objetivos incluem desenvolver uma interface fácil para os usuários, integrar diferentes métodos de pagamento e oferecer um meio para que organizadores de eventos possam colocar seus eventos para compra e divulgação.

1.3. Método de Trabalho:

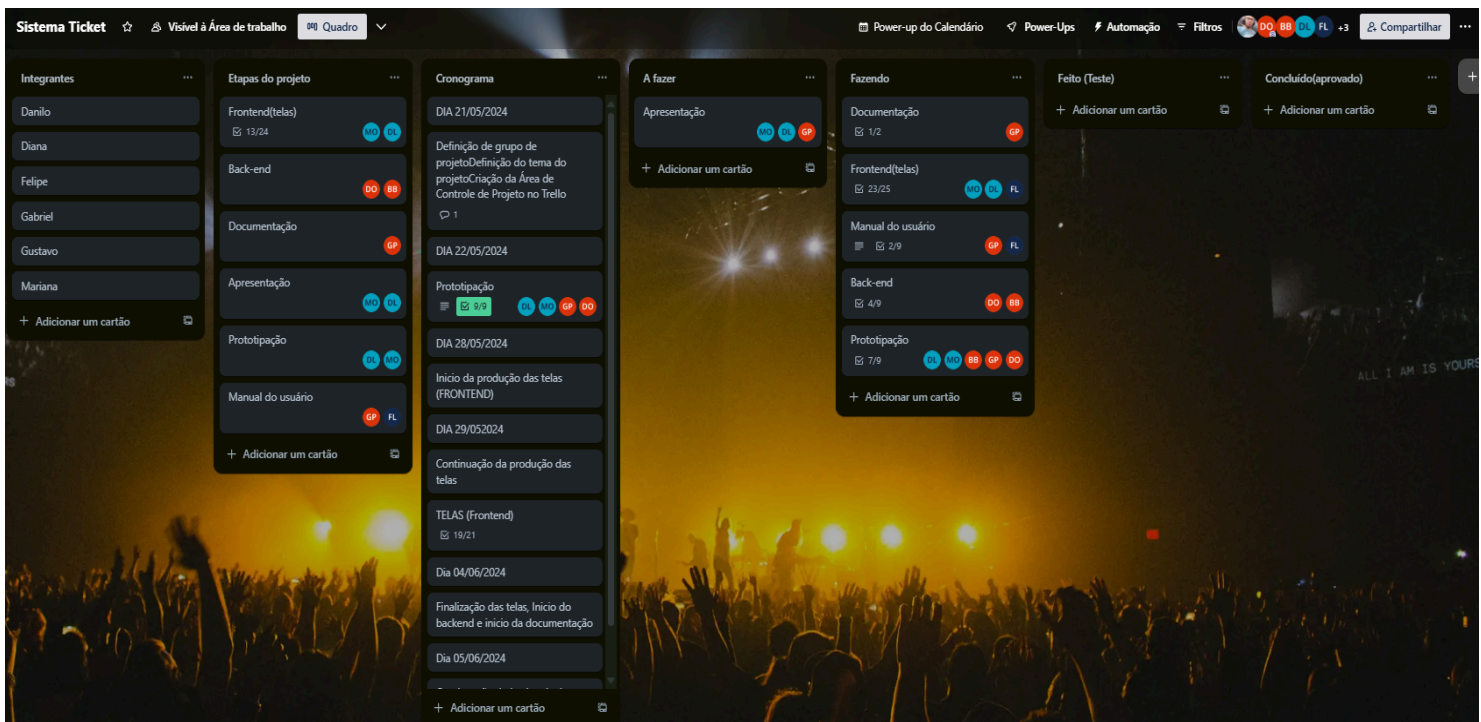
- O projeto será conduzido utilizando a metodologia ágil Scrum, que permite uma abordagem flexível e adaptável ao longo do desenvolvimento da API. A modelagem será orientada a objetos utilizando diagramas UML.
- Nós também utilizaremos o Trello para organizar o projeto e dirigir funções para os membros do grupo.

1.4. Organização do Trabalho:

O trabalho foi organizado pelo trello e realizado com a metodologia scrum.

Trello:

Sistema Ticket | Trello



Organização do Scrum:

- **Product Owner (PO):** Gustavo Pereira.
- **Scrum Master:** Mariana.
- **Development Team:** Danilo, Diana, Felipe.
- **Dev ops:** Danilo e Gabriel.

1.5 Membros e Papéis

Frontend: Diana, Felipe e Mariana.

Backend: Danilo e Gabriel.

Documentação: Gustavo.

Manual do usuário: Felipe e Gustavo.

DESCRIÇÃO GERAL DO SISTEMA:

2.1. Propósito do Sistema:

Desenvolver uma plataforma online de compra e venda de ingressos segura e eficiente para shows, eventos esportivos e exposições.

Principais propósitos do Innovators Ticket:

Facilidade de Uso: Criar uma interface que os usuários possam navegar e realizar suas compras de maneira rápida e sem complicações.

Integração de Pagamentos: Oferecer múltiplas opções de pagamento, como cartões de crédito, Pix e Dinheiro, para atender às preferências dos usuários.

Suporte ao Cliente: Oferecer um suporte ao cliente eficiente e acessível para resolver quaisquer problemas ou dúvidas que os usuários possam ter, garantindo uma experiência positiva.

2.2. Objetivos:

Objetivo Geral:

Desenvolver uma plataforma de venda de ingressos online que seja intuitiva, segura e eficiente, facilitando a compra e venda de ingressos para diversos tipos de eventos, como shows, exposições, e eventos esportivos.

Objetivos Específicos:

Interface Amigável ao usuário:

Criar uma interface de usuário intuitiva e fácil de usar para garantir uma experiência de navegação fluida e agradável para os usuários finais.

Integração de Pagamentos:

Integrar a plataforma com diversos métodos de pagamento, incluindo cartões de crédito, débito, transferências bancárias , para atender às preferências dos usuários.

Cadastrar Ingressos :

Desenvolver funcionalidades que permitam aos organizadores criar, editar e deletar ingressos de maneira eficiente.

Suporte ao Cliente:

Oferecer um suporte ao cliente eficiente para resolver problemas e esclarecer dúvidas.

2.3 Público alvo:

Nosso público alvo se resume em organizadores de eventos, empresas, e indivíduos que desejam comprar e vender ingressos online.

2.4 Benefícios de Usar o Innovators Ticket

- **Facilidade de Uso:** Interface intuitiva que torna a compra de ingressos rápida e simples.
- **Diversidade de Eventos:** Acesso a uma ampla gama de eventos, desde shows a eventos esportivos e exposições.
- **Flexibilidade de Pagamento:** Suporte a vários métodos de pagamento para maior conveniência.
- **Cadastrar ingressos:** funcionalidades que permitam os organizadores criar, editar e deletar ingressos para seus próprios eventos de maneira eficiente.

2.5 Tecnologias usadas:

As tecnologias usadas para o projeto foram:

Organização: Trello e Scrum.

Prototipação: Canvas.

Frontend: VsCode,html,css e javaScript.

backend: VsCode,JavaScript,node,nodemon,express, biblioteca http e ejs.

Documentação:Canvas,google docs e diagramas uml.

2.6 Ambiente de Implementação:

Para a implementação da plataforma "Ticket Innovators", utilizaremos um ambiente que maximize a eficiência e a flexibilidade, garantindo uma experiência fluida e segura para todos os usuários. O ambiente de implementação inclui os seguintes componentes:

Servidor

O servidor será desenvolvido utilizando JavaScript, com a biblioteca http. Isso oferece uma estrutura simples e eficiente para gerenciar solicitações e respostas HTTP, essencial para a comunicação entre o cliente (frontend) e o servidor (backend).

Base de Dados

A base de dados também será implementada em JavaScript, utilizando uma solução de banco de dados leve e eficiente, ideal para as necessidades da plataforma de venda de ingressos.

2.7 Integrações:

Para integrar a plataforma "Ticket Innovators" com os componentes essenciais utilizando JavaScript (com a biblioteca http) e uma base de dados também desenvolvida em JavaScript nós vamos usar:

Integração com o Servidor

Gerenciar requisições HTTP para diferentes endpoints.
Processar dados recebidos e enviar respostas apropriadas.

Integração com a Base de Dados

Armazenar e recuperar dados relacionados a eventos, ingressos e usuários.
Gerenciar operações CRUD (Create, Read, Update, Delete) na base de dados.

2.8 Políticas de Privacidade e Termos de Uso da Ticket Innovators:

Políticas de Privacidade

A Ticket Innovators se compromete a proteger a privacidade de todos os usuários que utilizam nossa plataforma. Esta política descreve como coletamos, usamos e protegemos suas informações pessoais.

Coleta de Informações

Informações Pessoais:

Coletamos informações como nome, endereço de e-mail, número de telefone e detalhes de pagamento quando você cria uma conta, compra ingressos ou entra em contato conosco.

Informações de Uso:

Coletamos informações sobre como você utiliza nossa plataforma, incluindo as páginas que visita, os eventos que visualiza e os ingressos que compra.

Uso das Informações

Fornecimento de Serviços:

Utilizamos suas informações pessoais para processar suas compras, fornecer suporte ao cliente e enviar atualizações relacionadas aos eventos que você comprou.

Compartilhamento de Informações

Conformidade Legal:

Podemos divulgar informações pessoais quando exigido por lei ou para proteger nossos direitos e segurança.

Segurança das Informações

Implementamos medidas de segurança técnicas e organizacionais para proteger suas informações pessoais contra acesso não autorizado, alteração, divulgação ou destruição.

Direitos dos Usuários

Você tem o direito de acessar, corrigir ou excluir suas informações pessoais a qualquer momento. Para exercer esses direitos, entre em contato conosco através dos canais fornecidos na nossa plataforma.

Termos de Uso

Os termos de uso da Ticket Innovators estabelecem as condições para a utilização de nossa plataforma. Ao utilizar nossos serviços, você concorda com os seguintes termos:

Uso da Plataforma

Conta de Usuário:

Você deve fornecer informações precisas e completas ao criar uma conta. Você é responsável por manter a confidencialidade de sua senha e por todas as atividades que ocorrem em sua conta.

Compra de Ingressos:

Todos os ingressos vendidos estão sujeitos à disponibilidade. Nos reservamos o direito de cancelar ou recusar qualquer pedido a nosso critério.

Proibições:

É proibido utilizar a plataforma para qualquer atividade ilegal, fraudulenta ou que viole os direitos de terceiros.

Propriedade Intelectual:

Todo o conteúdo da plataforma, incluindo textos, gráficos, logos, ícones, imagens e software, é propriedade da Ticket Innovators ou de seus licenciadores e está protegido por leis de direitos autorais e outras leis de propriedade intelectual.

Contato: ticketinnovatorsreclame@hotmail.com

Para quaisquer dúvidas ou preocupações sobre nossas políticas de privacidade e termos de uso, entre em contato conosco através dos canais fornecidos no rodapé de nossa Página na web.

Aceitação dos Termos

Ao utilizar a Ticket Innovators, você concorda com os termos de uso e políticas de privacidade aqui descritos. Caso não concorde com estes termos, recomendamos que não utilize nossa plataforma.

3. REQUISITOS DO SISTEMA

3.1. REQUISITOS FUNCIONAIS DO INNOVATORS TICKET

Interface Para Clientes:

Login e Cadastro de Usuário:

Se o cliente já possui conta, ele faz login com seu e-mail e senha.

Se não possui conta, ele se cadastra na plataforma, fornecendo nome, e-mail e senha.

Página Principal com Login:

Após o login, o cliente é redirecionado para a página principal da plataforma, onde pode visualizar os eventos disponíveis para compra.

Opções de Navegação:

O cliente tem acesso ao seu perfil, onde pode visualizar informações da conta e histórico de pedidos. Ele pode escolher um evento da página principal para comprar ingressos, clicando no evento desejado.

Compra de Ingressos:

Após selecionar um evento, o cliente é direcionado para a tela de quantidade de ingressos.

Ele seleciona e avança para a tela principal.

Após ele concluir seu pedido, o cliente prossegue para a tela de confirmação de compra(carrinho).

Ele confirma a compra e avança para a tela de pagamentos. Depois disso, o usuário escolhe um método de pagamento e avança para a tela de compra finalizada, onde uma mensagem é exibida confirmando a sua compra.

Interface Para Vendedores:

Login de Vendedor:

O vendedor é redirecionado para a página principal após o login.

Opções de Navegação:

O vendedor tem acesso ao seu perfil, onde pode visualizar informações da conta e cadastrar novos ingressos. Ele seleciona a opção "Cadastrar Ingressos" para adicionar novos ingressos ao sistema.

Cadastro de Ingressos:

Na tela de cadastro de ingressos, o vendedor preenche as informações necessárias para cadastrar um novo ingresso.

Após o preenchimento, os ingressos são cadastrados no sistema e ficam disponíveis para venda.

Observações Gerais:

Opção de Escolha de Tipo de Usuário:

Durante o processo de login, o usuário pode escolher se deseja acessar como cliente ou vendedor.

Cadastro Obrigatório para Login:

Para realizar o login, tanto clientes quanto vendedores devem estar cadastrados na plataforma.

3.2 Requisitos não funcionais

Usabilidade:

O sistema deve possuir uma interface intuitiva e fácil de usar para clientes e vendedores.

Desempenho:

O sistema deve responder rapidamente às solicitações do usuário, com um tempo de carregamento das páginas.

Confiabilidade:

O sistema deve ser estável e confiável, com poucas interrupções ou falhas.

Portabilidade:

O sistema deve ser acessível através de diferentes dispositivos e navegadores web com uma experiência consistente.

Compatibilidade:

O sistema deve ser compatível com as principais versões de navegadores web, como Chrome, Firefox e Edge.

Escalabilidade:

O sistema deve ser capaz de lidar com um aumento no número de usuários e transações sem comprometer o desempenho.

3.3 Protótipo

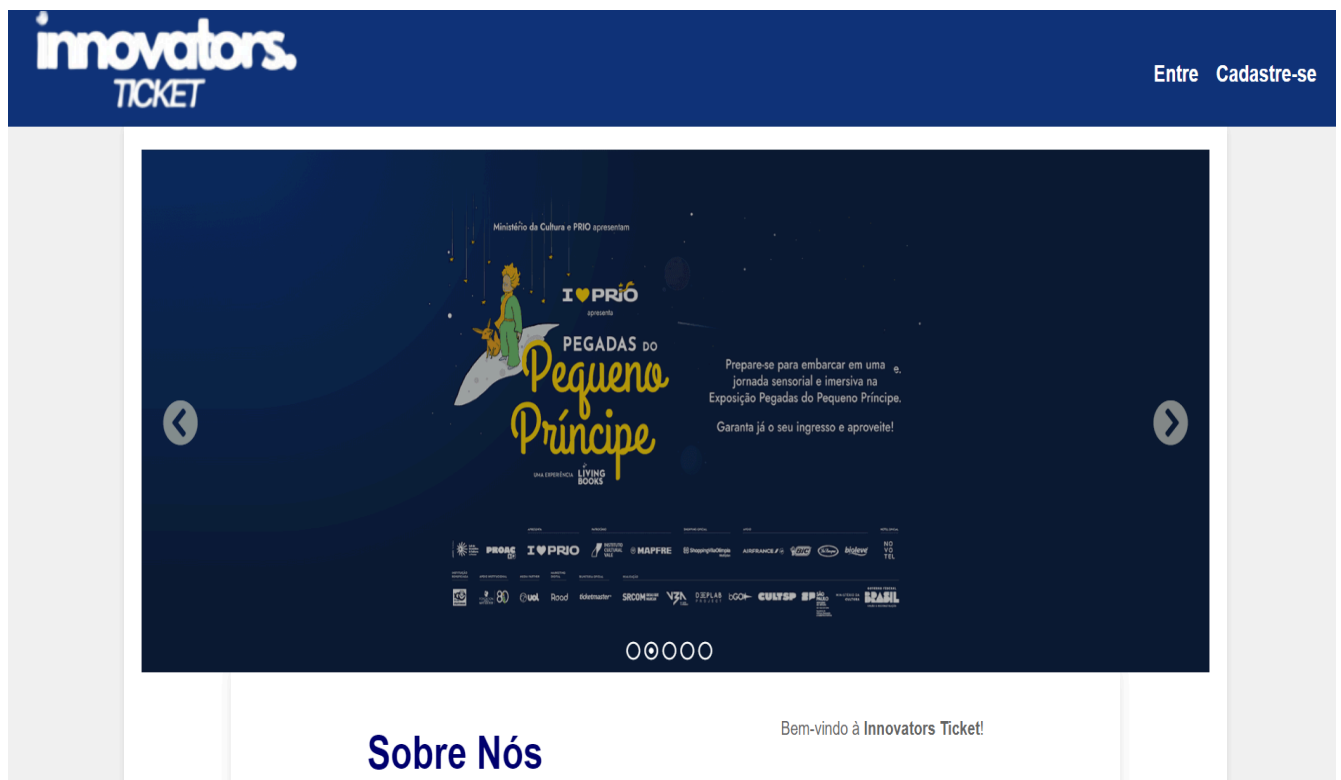
Neste capítulo apresentaremos toda a prototipagem de telas e funcionalidades-chave do sistema.

Nós dividimos o sistema em dois tipos de interface ambas são similares porém a primeira interface é destinada para o usuário comum (Comprador) a segunda é destinada para os usuários que querem vender ingressos (Vendedores).

Interface para usuários comuns (Compradores)

Interface do cliente(Comprador)

Página principal sem login:



Nossa História:

A **Innovators Ticket** nasceu da paixão por eventos ao vivo e do desejo de facilitar a vida dos amantes de entretenimento. Com uma equipe experiente no mercado de eventos e tecnologia, identificamos a necessidade de uma plataforma que fosse intuitiva e eficiente, proporcionando uma experiência de compra sem complicações.

O Que Oferecemos:

Variedade de Eventos: Trabalhamos com uma ampla gama de eventos para atender todos os gostos e interesses. Seja um grande fã de música, teatro, esportes ou festivais, você encontrará o ingresso perfeito aqui.

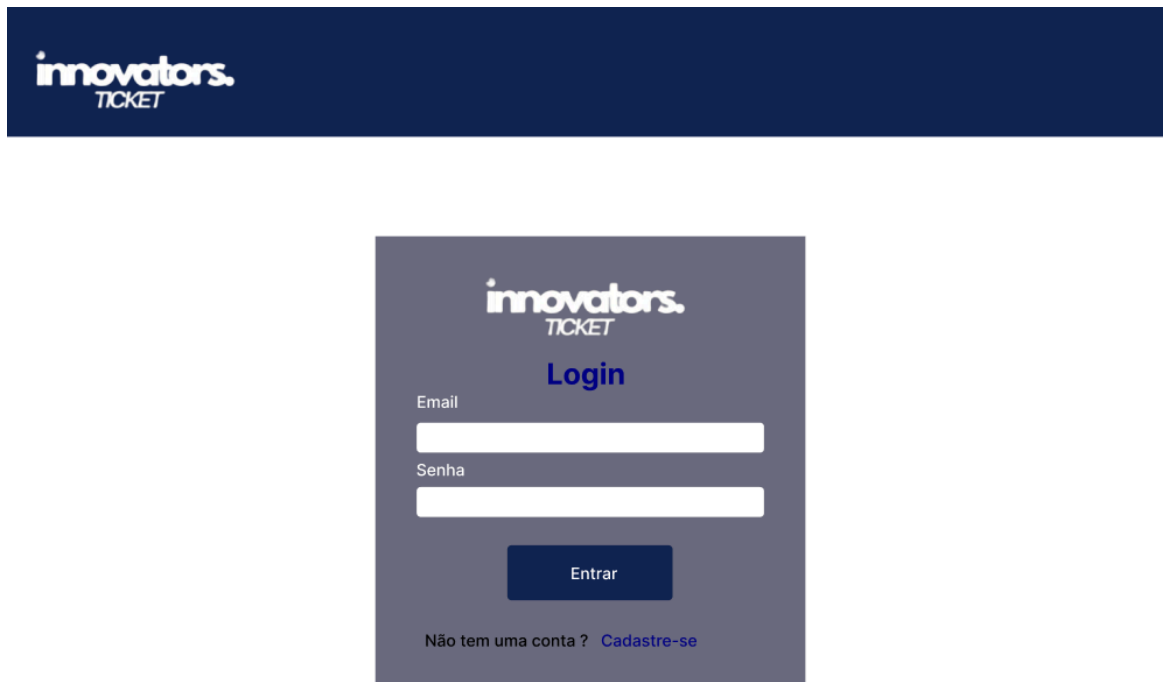
Facilidade de Uso: Nosso site é projetado para ser intuitivo e fácil de navegar, garantindo que você encontre e compre seus ingressos em poucos cliques.

Segurança: Valorizamos sua segurança. Utilizamos tecnologia de ponta para proteger suas informações pessoais e garantir transações seguras.

Atendimento ao Cliente: Nossa equipe de suporte está sempre pronta para ajudar. Temos canais de atendimento disponíveis para tirar suas dúvidas e resolver quaisquer problemas que possam surgir.

- Esta é a tela inicial do nosso site. Aqui nós oferecemos diversas informações de nossa plataforma.
- Para o usuário usufruir de nosso sistema e utilizar todas as funcionalidades do ticket innovators o usuário deve fazer login em nosso sistema.

Página de login:



innovators.
TICKET

Login

Email

Senha

Entrar

Não tem uma conta ? [Cadastre-se](#)

- Para fazer login em nosso site, é necessário ter um cadastro. Caso o usuário não possua, ele pode criar um clicando em 'Cadastre-se'.
- Se o usuário já tiver um cadastro, ele pode fazer login normalmente e será redirecionado para a página principal, já logado.

Página de Cadastro (Compradores)

innovators.
TICKET

Entre Cadastre-se

SE CADASTRE

Nome:

Sobrenome:

Email:

Tipo de Documento:

Selecione...

Número do Documento:

Sexo:

Selecione...

Senha:

Número de Telefone:

Data de Nascimento:

dd/mm/aaaa

Enviar

- Para usar o Innovators Ticket, você precisa se cadastrar e fazer login. Isso garante acesso completo ao nosso site.
- Na tela de cadastro, preencha os campos obrigatórios com suas informações para criar sua conta e fazer login no site.

Página principal Cadastrado:


innovators.
TICKET


Home Meu Perfil


Exposições
Aqui você encontra informações sobre as últimas exposições.


Shows
Aqui você encontra informações sobre os shows mais esperados.

Esportes
Aqui você encontra informações sobre eventos esportivos.

**PodParty**
Descrição: Evento que contém a presença de artistas como Kyan Douglas e etc.
Local: São paulo-SP
Valor: R\$75
Data: 2024-06-29
Tipo: show

**Pegadas do Pequeno Príncipe**
Descrição: Exposição baseada no livro o pequeno príncipe
Local: Vila Olímpia-SP
Valor: R\$30
Data: 2024-07-14
Tipo: exposição

**NBA HOUSE**
Descrição: Evento esportivo
Local: São paulo-SP
Valor: R\$140
Data: 2024-08-17
Tipo: esporte

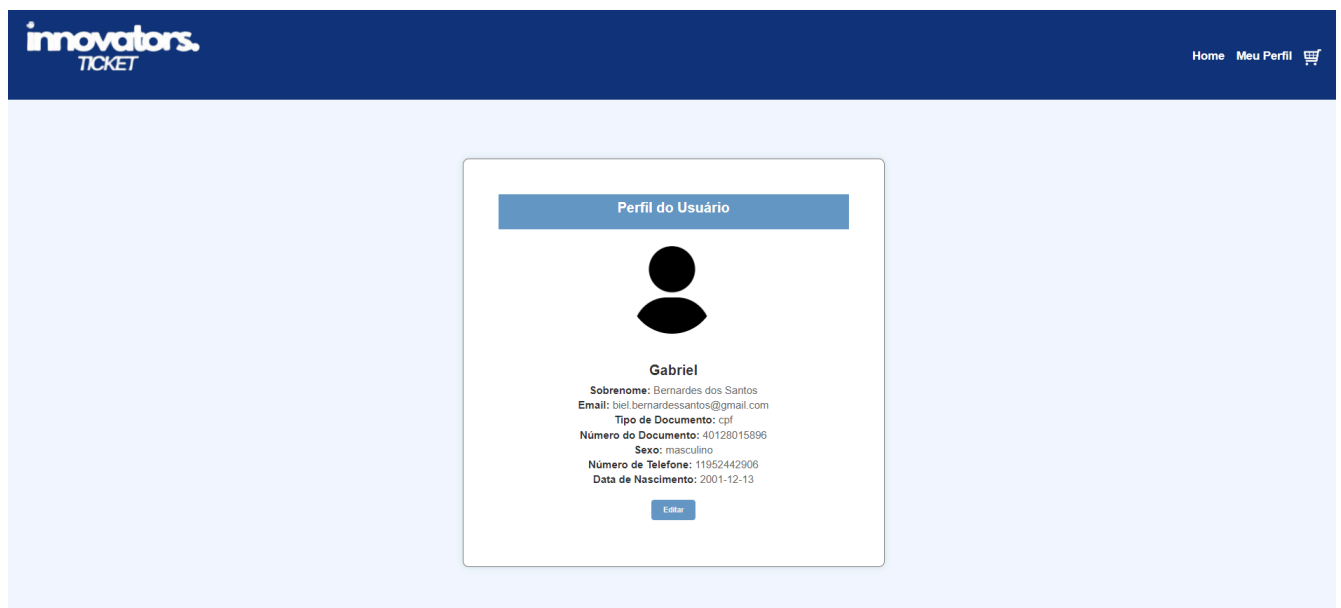
**Bruno Mars Concert**
Descrição: Bruno Mars se apresenta no Brasil com seu show cheio de emoção!
Local: Nilton Santos, RJ
Valor: R\$250
Data: 2024-10-10
Tipo: show

- Após fazer o login, o usuário será encaminhado para a página principal com todas as funcionalidades.

A página principal tem como funcionalidades:

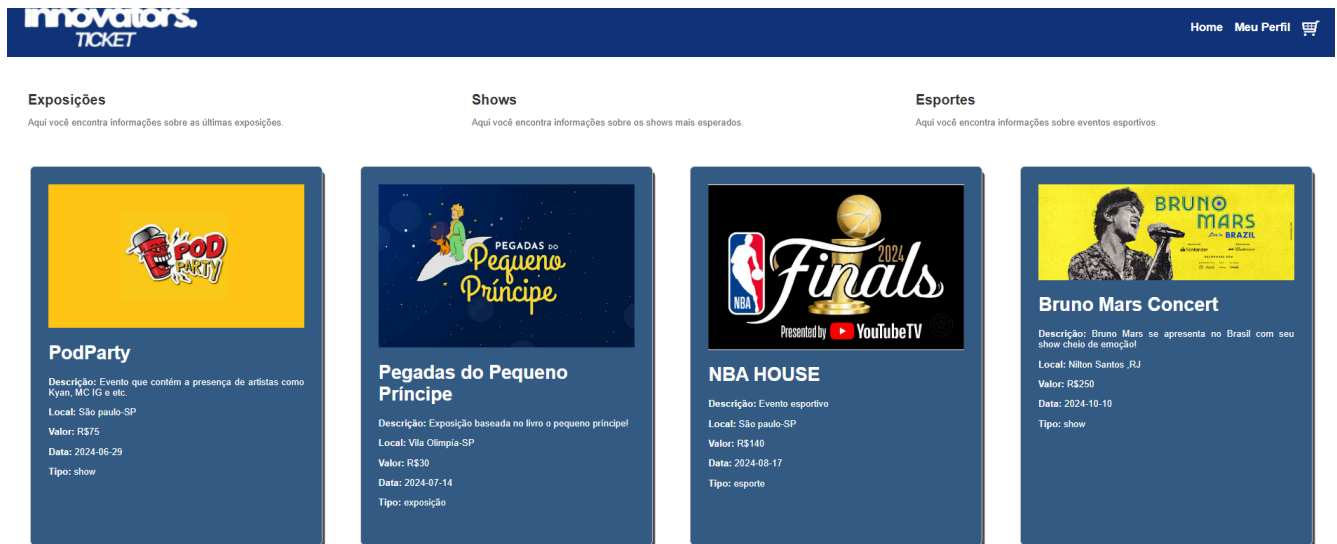
- A tela de Perfil.
- A tela de meus pedidos(Carrinho).
- Clicando em um dos eventos disponíveis do site você é encaminhado para a tela de Quantidade de ingressos.

Tela de perfil



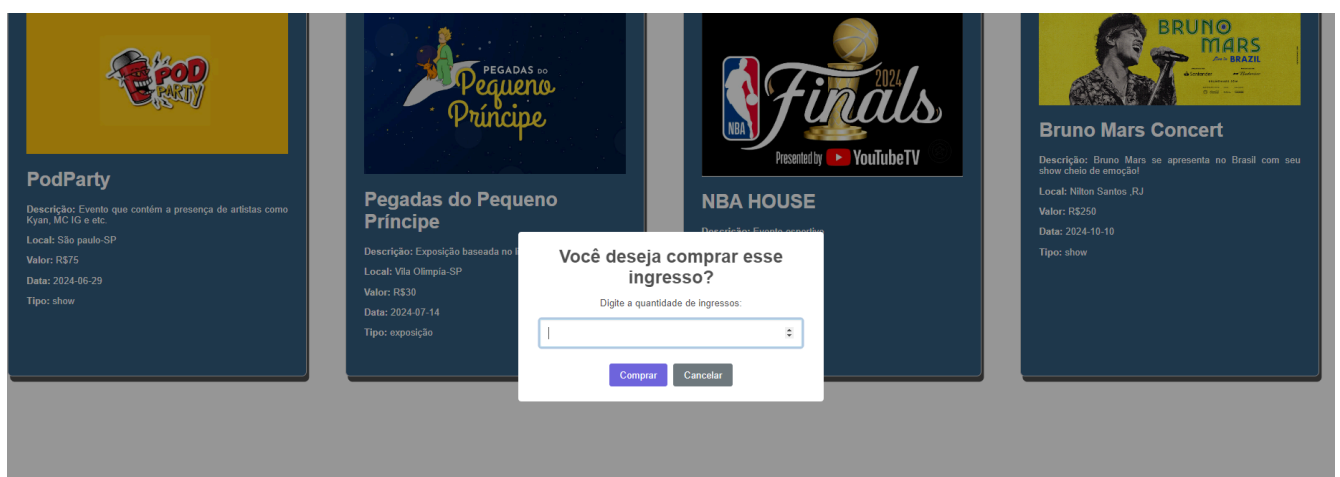
- Nesta tela o usuário poderá visualizar todas as informações cadastradas no site após o processo de cadastro.

Comprando Ingressos:



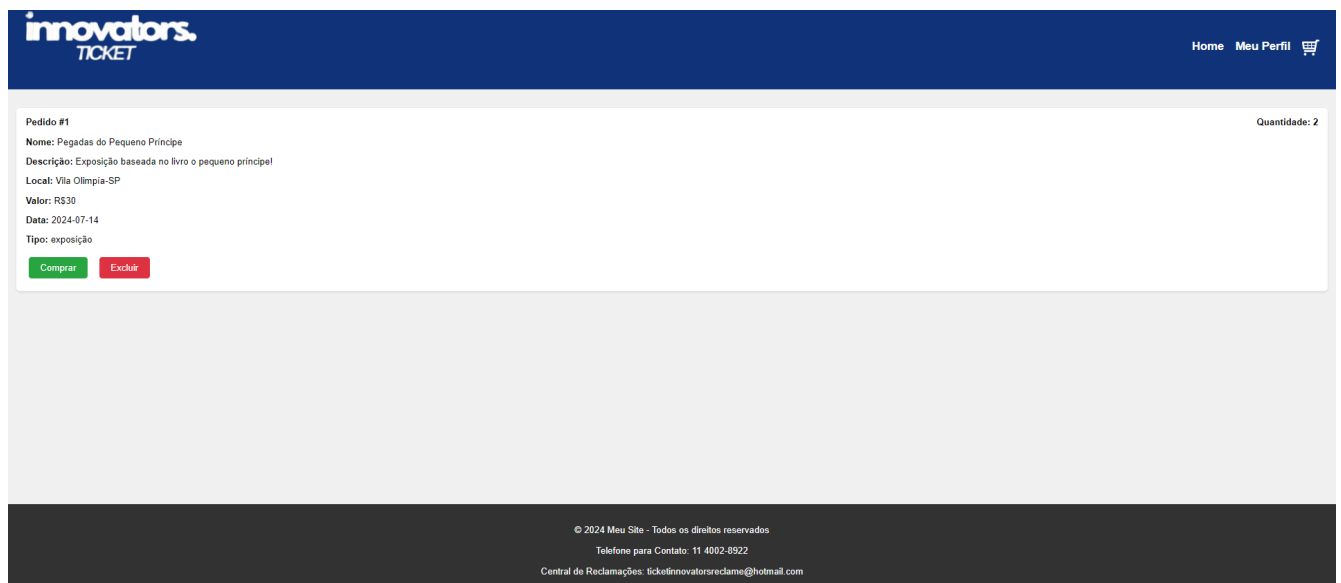
- Para realizar a compra dos ingressos, você deve clicar em um dos eventos na página principal.

Tela de Quantidade de Ingressos:



- Aqui nesta tela o usuário diz ao sistema quantos ingressos ele deseja para fazer um pedido.

Tela de Meus pedidos (Carrinho):



- Após o usuário escolher a quantidade de ingressos ele deve clicar no carrinho para confirmar sua compra.
- Esta tela mostra os pedidos de ingressos solicitados pelo usuário.
- Por meio desta tela podemos ver as informações do pedido de ingresso como descrição, local, valor, data e tipo do evento.

Tela de Pagamento:



- Após o usuário clicar em “Comprar” ele chegará na tela de pagamento. Aqui o usuário encontrará diversas formas integradas de pagamento, ele pode escolher qual ele achar melhor para realizar a compra.

tela de pagamento com pix:

innovators.
TICKET

Home Meu Perfil

Pagamento via Pix

Chave: 546.668.9898

Chave Pix

5466699998

Valor (R\$) 60

Gerar QR Code



Avançar

© 2024 Meu Site - Todos os direitos reservados
Telefone para Contato: 11 4002-8922

tela de pagamento com cartão:

Pagamento com Cartão

Nome Completo

Gabriel Bernardes dos Santos

CPF

40128015896

Número do Cartão

Data de Expiração

CVV

Valor (R\$)

60.00

Finalizar Pagamento

Valor a ser pago: R\$ 60.00

© 2024 Meu Site - Todos os direitos reservados
Telefone para Contato: 11 4002-8922
Central de Reclamações: ticketinnovatorsreclama@hotmail.com

tela de pagamento com dinheiro:

innovators
TICKET

Home Meu Perfil

Pagamento com Boleto Bancário

Nome Completo

Gabriel Bernardes dos Santos

CPF

40128015896

Valor (R\$)

60

Gerar Boleto

Boleto Bancário

Nome: Gabriel Bernardes dos Santos

CPF: 40128015896

Valor: R\$ 60

Linha Digitável: 12345.67890 12345.678901
12345.678901 1 23456789012345

Avançar

© 2024 Meu Site - Todos os direitos reservados

Telefone para Contato: 11 4002-8922

Central de Reclamações: ticketinnovatorsreclame@hotmail.com

Tela de Compra realizada:

innovators
TICKET

HOME MEU PERFIL

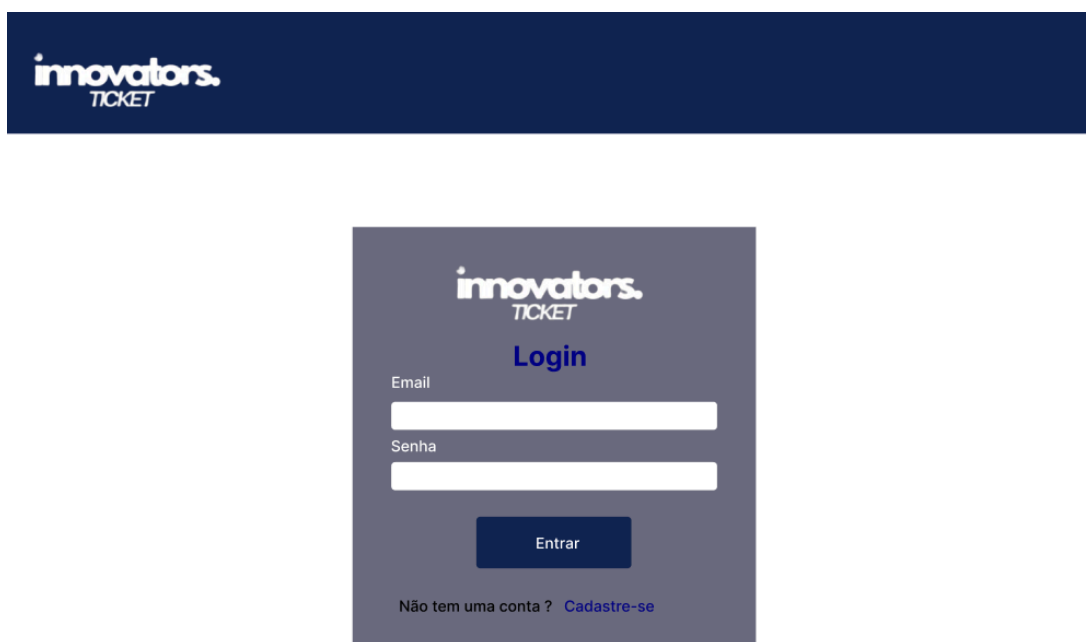
Seu pedido foi finalizado!

Obrigado por comprar com a Innovators Ticket! Para visualizar seu pedido, clique na aba MEU PERFIL para checar os detalhes da sua compra.

- Com a compra feita o usuário verá a tela de compra realizada.
- Aqui o usuário verá que sua compra foi confirmada.

Interface do vendedor:

Página de login:



The image shows a login page for 'innovators.TICKET'. The page has a dark blue header with the logo. The main content area is light gray and contains the login form. The form has two input fields: 'Email' and 'Senha'. Below the fields is a dark blue button labeled 'Entrar'. At the bottom of the form, there is a link that says 'Não tem uma conta ? Cadastre-se'.

- Para fazer login em nosso site, é necessário ter um cadastro. Caso o usuário não possua, ele pode criar um clicando em 'Cadastre-se'.

- Se o usuário já tiver um cadastro, ele pode fazer login normalmente e será redirecionado para a página principal, já logado.

Página de Cadastro (Vendedores):

JUNTE-SE A NÓS

Nome da empresa:

Email:

CNPJ da empresa:

Categoria de Ingressos:

Código do País (para o número de telefone):

Número de Telefone:

Enviar

- Para usar o Innovators Ticket, você precisa se cadastrar e fazer login. Isso garante acesso completo ao nosso site.

- Na tela de cadastro, preencha os campos obrigatórios com suas informações para criar sua conta e fazer login no site.
- Esta tela de cadastro é diferente da dos clientes normais, nesta tela temos campos dedicados a empresas ou vendedores de ingressos como nome da empresa e CNPJ.

Página principal (Vendedor):

innovators.
TICKET

[Cadastrar Ingresso](#) [Home](#)

Exposições
Aqui você encontra informações sobre as últimas exposições.

Shows
Aqui você encontra informações sobre os shows mais esperados.

Esportes
Aqui você encontra informações sobre eventos esportivos.

PodParty
Descrição: Evento que contém a presença de artistas como Kyan, MC IG e etc.
Local: São paulo-SP
Valor: R\$75
Data: 2024-06-29
Tipo: show
[Excluir](#) [Atualizar](#)

Pegadas do Pequeno Príncipe
Descrição: Exposição baseada no livro o pequeno príncipe
Local: Vila Olímpia-SP
Valor: R\$30
Data: 2024-07-14
Tipo: exposição
[Excluir](#) [Atualizar](#)

NBA HOUSE
Descrição: Evento esportivo
Local: São paulo-SP
Valor: R\$140
Data: 2024-08-17
Tipo: esporte
[Excluir](#) [Atualizar](#)

Bruno Mars Concert
Descrição: Bruno Mars se apresenta no Brasil com seu show cheio de emoção!
Local: Nilton Santos, RJ
Valor: R\$250
Data: 2024-10-10
Tipo: show
[Excluir](#) [Atualizar](#)

- Após fazer o login, o usuário será encaminhado para a página principal com todas as funcionalidades.

A página principal tem como funcionalidades:

- Cadastrar ingressos

Tela de venda de Cadastro dos ingressos:

innovators.
TICKET

Cadastrar Ingresso Home

Cadastro

Nome do evento:

Descrição do evento:

Local do evento:

Valor:

Data:

05/01/2020

Tipo:

Selecione

Imagem do evento:

Escolher arquivo Nenhum arquivo escolhido

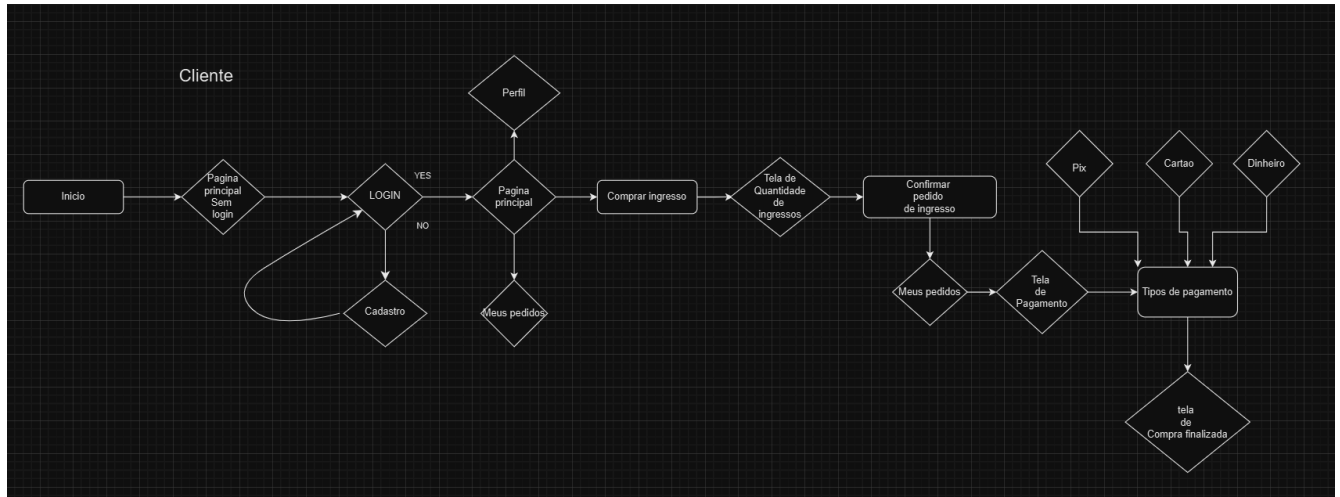
Cadastrar

- A tela de cadastro de ingressos permite aos usuários cadastrar seus próprios ingressos para que seus eventos apareçam no site e no catálogo.
- Essa funcionalidade pode ser acessada clicando no link "Venda seu ingresso aqui".
- Quando os ingressos forem cadastrados, o usuário será levado para tela principal novamente.

3.4 DIAGRAMA DE NAVEGAÇÃO :

Interface do Cliente:

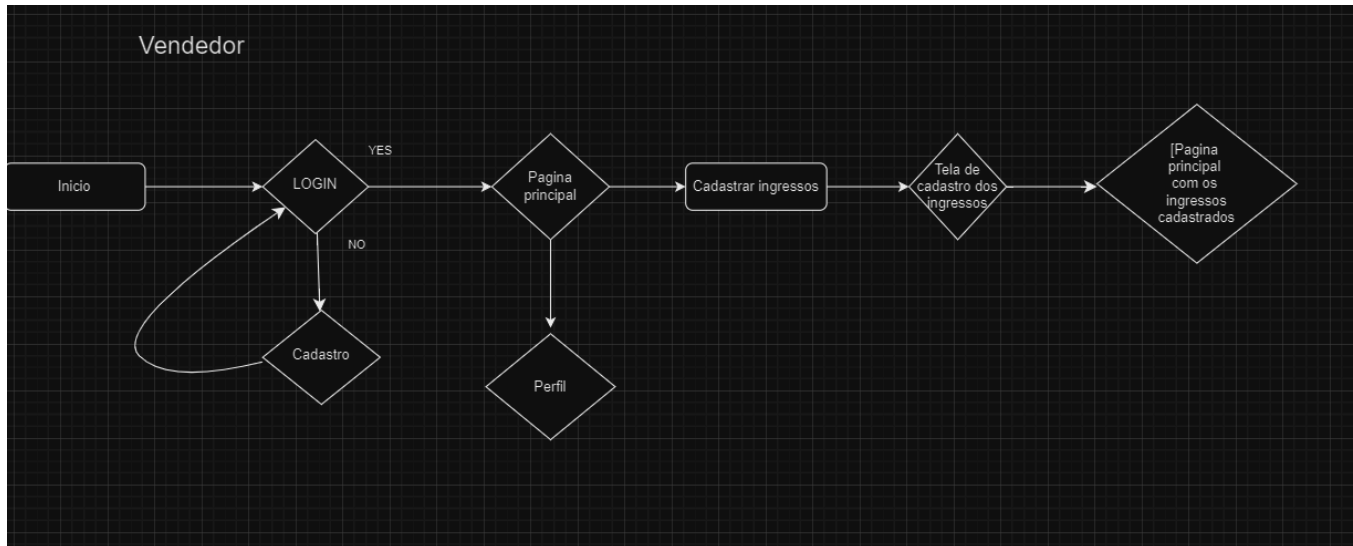
<https://app.diagrams.net/#G1CBQ2mrWEBYbNkDZtigQsYlQIwUPCCoz#%7B%22pageId%22%3A%22C5RBs43oDa-KdzZeNtuy%22%7D>



A interface do cliente é composta por :

login => Cadastro => Perfil=> Quantidade de ingresso=> tela de meus pedidos(Carrinho) => tela De Pagamentos => dependendo do tipo de pagamento escolhido ele pode ir para tela de pagamento com pix
tela de pagamento com cartão,tela de pagamento com dinheiro,tela de pagamento com cupom ou vale presente =>tela de Tela de compra finalizada.

Interface de vendedor:



A interface do Vendedor é composta por:

login => Cadastro => Página principal => Perfil = Tela dos ingressos cadastrados
=> tela de ingresso cadastrados (pagina principal).

3.5. MÉTRICAS E CRONOGRAMA:

Métricas de Desempenho do Sistema:

Tempo de Carregamento das Páginas:

- Manter o tempo de carregamento abaixo de 3 segundos.
- Verificação manual com cronômetro ou ferramentas básicas de navegador.

Métricas de Uso do Sistema:

- Número de Cadastros de Usuários:
- Acompanhar o número de novos usuários registrados.
- Contagem semanal dos novos cadastros.

Número de Ingressos Vendidos:

- Monitorar quantos ingressos foram vendidos.
- Contagem diária e semanal das vendas de ingressos.

Métricas de Satisfação do Usuário:

- Número de comentários ou avaliações positivas recebidas.

Métricas de Suporte:

Número de Chamados de Suporte:

- Acompanhar a quantidade de chamados de suporte.
- Contagem semanal dos chamados recebidos.

Cronograma:

Dia 21/05/2024

Definição de grupo, Definição do tema, criação do trello.

Dia 22/05/2024

Elaboração da prototipação.

Dia 28/05/2024

Início do frontend

Dia 29/05/2024

Continuação da produção do frontend

Dia 04/06/2024

Finalização das telas ,início do backend e documentação.

Dia 05/06/2024

Continuação do backend , da documentação e revisão do frontend

Dia 11/06/2024

Continuação do backend,finalização do frontend , continuação da documentação e início da apresentação.

Dia 12/06/2024

Continuação do backend,finalização do frontend , continuação da documentação e a continuação da apresentação.

Dia 14/06/2024

Continuação do backend,finalização do frontend , continuação da documentação e continuação da apresentação.

Dia 16/06/2024

Entrega do projeto.

4.ANÁLISE E Arquitetura do Sistema

4.1. ARQUITETURA DO SISTEMA

- A arquitetura do nosso sistema é composta principalmente por JavaScript. Utilizamos essa linguagem para implementar as funções de cadastro, login, página principal, meus pedidos, perfil, tipo de ingresso, data, horário, seguro, tela de termos e condições, pagamento, login de vendedor, cadastro de vendedor, perfil e cadastro de ingressos.

Backend

o projeto foi organizado no padrão Mvc e as seguintes pastas contêm as partes do sistema:

pasta Src:

Pasta Js

arquivo scriptcad.js

```
// Aguarda o carregamento completo do conteúdo da página
document.addEventListener('DOMContentLoaded', function() {
  // Seleciona o formulário com o ID 'eventoForm'
  const form = document.getElementById('eventoForm');

  // Adiciona um evento de escuta para o evento de envio do formulário
  form.addEventListener('submit', function(event) {
    // Previne o comportamento padrão do navegador de enviar o formulário
    event.preventDefault();
  });
});
```


// Captura os valores inseridos pelo usuário nos campos do formulário

```
const nome = document.getElementById('nome').value;  
const sobrenome =  
document.getElementById('sobrenome').value;  
const email = document.getElementById('email').value;  
const documento =  
document.getElementById('tipo_documento').value;  
const nDocumento =  
document.getElementById('numero_documento').value;  
const genero = document.getElementById('sexo').value;  
const senha = document.getElementById('senha').value;  
const telefone =  
document.getElementById('numero_telefone').value;  
const nascimento =  
document.getElementById('data_nascimento').value;
```

// Cria um objeto 'user' com os dados capturados

```
const user = {  
  nome,  
  sobrenome,  
  documento,  
  email,  
  nDocumento,  
  genero,  
  senha,  
  telefone,  
  nascimento  
};
```

// Envia os dados do usuário para o servidor usando uma solicitação HTTP POST

```
fetch('http://localhost:3001/register', {  
  method: 'POST', // Define o método da solicitação como POST
```

```

        headers: {
            'Content-Type': 'application/json' // Define o cabeçalho para
indicar JSON
        },
        body: JSON.stringify(user) // Converte o objeto 'user' para uma
string JSON
    })
    .then(response => response.text()) // Processa a resposta da
solicitação, convertendo-a em texto
    .then(data => {
        console.log(data); // Exibe a resposta no console
        alert('Dados enviados com sucesso!'); // Exibe uma mensagem
de sucesso
    })
    .catch(error => {
        console.error('Erro ao enviar dados:', error); // Exibe o erro no
console
        alert('Erro ao enviar dados. Verifique o console para mais
informações. '); // Exibe uma mensagem de erro
    });
});
});

```

Este código JavaScript captura dados de um formulário de cadastro, previne o envio padrão, e envia os dados ao servidor via uma solicitação HTTP POST. Ele também exibe mensagens de sucesso ou erro conforme a resposta do servidor.

Arquivo scriptcad.js

```

// Aguarda o carregamento completo do DOM
document.addEventListener('DOMContentLoaded', () => {
    // Seleciona o contêiner onde os cards dos ingressos serão exibidos
    const cardContainer = document.getElementById('card-container');

```

```

// Faz uma solicitação GET para obter os ingressos do servidor
fetch('http://localhost:3001/tickets')
  .then(response => {
    // Verifica se a resposta do servidor está OK
    if (!response.ok) {
      throw new Error('Erro na resposta do servidor');
    }
    // Converte a resposta para JSON
    return response.json();
  })
  .then(data => {
    console.log('Dados recebidos do servidor:', data);
    // Verifica se os dados recebidos são um array
    if (!Array.isArray(data)) {
      throw new Error('Formato de dados inválido');
    }

    // Itera sobre cada ingresso no array de dados
    data.forEach(ticket => {
      // Cria um elemento div para o card do ingresso
      const card = document.createElement('div');
      card.classList.add('card');
      card.style.cursor = 'pointer';
      // Preenche o conteúdo do card com os dados do ingresso
      const cardContent = `
        <h1 class="titulo">${ticket.nome}</h1>
        <p class="descricao"><strong>Descrição:</strong>
${ticket.descricao}</p>
        <p class="local"><strong>Local:</strong>
${ticket.local}</p>
        <p class="valor"><strong>Valor:</strong>
R${ticket.valor}</p>

```

```

        <p class="data"><strong>Data:</strong>
        ${ticket.data}</p>
        <p class="tipo"><strong>Tipo:</strong> ${ticket.tipo}</p>
        `;

        card.innerHTML = cardContent;

        // Adiciona um evento de clique ao card para iniciar o processo
        de compra
        card.addEventListener('click', () => {
            if (confirm('Você deseja comprar esse ingresso?')) {
                // Solicita a quantidade de ingressos desejada pelo usuário
                const quantidade = prompt('Digite a quantidade de
                ingressos:');
                if (quantidade && !isNaN(quantidade) && quantidade > 0) {
                    // Solicita o email do usuário
                    const userEmail = prompt('Digite seu email:');
                    // Faz uma solicitação POST para adicionar o ingresso aos
                    pedidos
                    fetch('http://localhost:3001/addTicketToPedidos', {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json'
                        },
                        body: JSON.stringify({ userEmail, ticketId: ticket.id,
                        quantidade: parseInt(quantidade, 10) })
                    })
                    .then(response => {
                        // Verifica se a resposta do servidor está OK
                        if (!response.ok) {
                            throw new Error('Erro ao adicionar o ingresso aos
                            pedidos');
                        }
                        return response.json();
                    })
                }
            }
        });

```

```

    })
    .then(data => {
        // Exibe uma mensagem de sucesso ao usuário
        alert('Ingresso adicionado aos pedidos com sucesso');
        console.log(data);
    })
    .catch(error => console.error('Erro ao adicionar o
ingresso aos pedidos:', error));
    } else {
        // Exibe uma mensagem de erro se a quantidade for
inválida
        alert('Quantidade inválida');
    }
}
});

// Adiciona o card ao contêiner de cards
cardContainer.appendChild(card);
});
})
.catch(error => console.error('Erro ao buscar os dados:', error));
});

```

Este código espera o carregamento completo do DOM e, em seguida, faz uma solicitação GET para obter dados de ingressos do servidor. Para cada ingresso recebido, cria um card com detalhes como nome, descrição, local, valor, data e tipo. Ao clicar em um card, o usuário pode confirmar a compra, especificar a quantidade de ingressos e fornecer seu email. Se os dados forem válidos, faz uma solicitação POST para adicionar o ingresso aos pedidos do usuário, exibindo mensagens de sucesso ou erro conforme a resposta do servidor.

arquivo scriptDel.js

```
document.addEventListener('DOMContentLoaded', () => {  
  // Seleciona o contêiner onde os cards dos ingressos serão exibidos  
  const cardContainer = document.getElementById('card-container');  
  
  // Faz uma requisição para buscar os ingressos do servidor  
  fetch('http://localhost:3001/tickets')  
    .then(response => {  
      // Verifica se a resposta do servidor está ok  
      if (!response.ok) {  
        throw new Error('Erro na resposta do servidor');  
      }  
      return response.json();  
    })  
    .then(data => {  
      console.log('Dados recebidos do servidor:', data);  
      // Verifica se os dados recebidos são um array  
      if (!Array.isArray(data)) {  
        throw new Error('Formato de dados inválido');  
      }  
  
      // Cria e adiciona os elementos dos cards para cada ingresso  
      data.forEach(ticket => {  
        const card = document.createElement('div');  
        card.classList.add('card');  
        card.style.cursor = 'pointer';  
        const cardContent = `  
          <h1 class="titulo">${ticket.nome}</h1>  
          <p class="descricao"><strong>Descrição:</strong>  
${ticket.descricao}</p>  
          <p class="local"><strong>Local:</strong>  
${ticket.local}</p>
```

```

        <p class="valor"><strong>Valor:</strong>
R$$${ticket.valor}</p>
        <p class="data"><strong>Data:</strong>
${ticket.data}</p>
        <p class="tipo"><strong>Tipo:</strong> ${ticket.tipo}</p>
        <div class="card-options">
            <button class="delete-btn">Excluir</button>
            <button class="update-btn">Atualizar</button>
        </div>
    `;

    card.innerHTML = cardContent;

    // Adiciona um listener para o botão de excluir
    card.querySelector('.delete-btn').addEventListener('click', ()
=> {
        if (confirm('Você deseja excluir esse ticket?')) {
            deleteTicket(ticket.id);
        }
    });

    // Adiciona um listener para o botão de atualizar
    card.querySelector('.update-btn').addEventListener('click', ()
=> {
        updateTicket(ticket.id);
    });

    cardContainer.appendChild(card);
    });
})
.catch(error => console.error('Erro ao buscar os dados:', error));

// Função para excluir um ingresso
function deleteTicket(ticketId) {

```

```

fetch(`http://localhost:3001/tickets/${ticketId}`, {
  method: 'DELETE'
})
.then(response => {
  // Verifica se a resposta do servidor está ok
  if (!response.ok) {
    throw new Error('Erro ao excluir o ticket');
  }
  return response.json();
})
.then(data => {
  console.log(data);
  alert('Ticket excluído com sucesso');
  location.reload(); // Recarrega a página após a exclusão
})
.catch(error => console.error('Erro ao excluir o ticket:', error));
}

```

```

// Função para atualizar um ingresso
function updateTicket(ticketId) {
  fetch(`http://localhost:3001/tickets/${ticketId}`)
    .then(response => {
      // Verifica se a resposta do servidor está ok
      if (!response.ok) {
        throw new Error('Erro ao buscar os dados do ticket');
      }
      return response.json();
    })
    .then(ticketData => {
      const updateForm = document.createElement('form');
      updateForm.innerHTML = `
        <h2>Atualizar Ingresso</h2>
        <div class="form-group">
          <label for="update-nome">Nome:</label>

```



```

        <input type="text" id="update-nome"
name="update-nome" value="{ticketData.nome}" required>
    </div>
    <div class="form-group">
        <label for="update-descricao">Descrição:</label>
        <input type="text" id="update-descricao"
name="update-descricao" value="{ticketData.descricao}" required>
    </div>
    <div class="form-group">
        <label for="update-local">Local:</label>
        <input type="text" id="update-local"
name="update-local" value="{ticketData.local}" required>
    </div>
    <div class="form-group">
        <label for="update-valor">Valor:</label>
        <input type="number" id="update-valor"
name="update-valor" value="{ticketData.valor}" step="0.01"
required>
    </div>
    <div class="form-group">
        <label for="update-data">Data:</label>
        <input type="date" id="update-data"
name="update-data" value="{ticketData.data}" required>
    </div>
    <div class="form-group">
        <label for="update-tipo">Tipo:</label>
        <select id="update-tipo" name="update-tipo" required>
            <option value="">Selecione</option>
            <option value="Show" {ticketData.tipo === 'Show' ?
'selected' : ''}>Show</option>
            <option value="Exposições" {ticketData.tipo ===
'Exposições' ? 'selected' : ''}>Exposições</option>
            <option value="Esportes" {ticketData.tipo ===
'Esportes' ? 'selected' : ''}>Esportes</option>
        </select>
    </div>

```

```

        </select>
    </div>
    <button type="submit">Atualizar</button>
`;

// Listener para submissão do formulário de atualização
updateForm.addEventListener('submit', event => {
    event.preventDefault();
    const updatedData = {
        id: ticketId, // Inclui o ID do ingresso
        nome: document.getElementById('update-nome').value,
        descricao:
document.getElementById('update-descricao').value,
        local: document.getElementById('update-local').value,
        valor: document.getElementById('update-valor').value,
        data: document.getElementById('update-data').value,
        tipo: document.getElementById('update-tipo').value
    };

    fetch(`http://localhost:3001/tickets/${ticketId}`, {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(updatedData)
    })
    .then(response => {
        // Verifica se a resposta do servidor está ok
        if (!response.ok) {
            throw new Error('Erro ao atualizar o ingresso');
        }
        return response.text(); // Altera para texto para capturar
respostas não JSON
    })

```

```

        .then(data => {
            try {
                const jsonData = JSON.parse(data); // Tenta parsear JSON
                alert('Ingresso atualizado com sucesso');
                location.reload(); // Recarrega a página após a
atualização
            } catch (e) {
                console.error('Resposta não é um JSON válido:', data); //
Log da resposta em texto
                alert('Ingresso atualizado com sucesso');
                location.reload(); // Recarrega a página de qualquer
forma
            }
        })
        .catch(error => console.error('Erro ao atualizar o ingresso:',
error));
    });

    cardContainer.innerHTML = ''; // Limpa o conteúdo existente
    cardContainer.appendChild(updateForm); // Adiciona o
formulário de atualização
    })
    .catch(error => console.error('Erro ao buscar os dados do
ticket:', error));
    }
    });

```

Este script JavaScript é executado quando o DOM está totalmente carregado. Ele busca ingressos de um servidor e os exibe como cards na página. Cada card inclui botões para excluir ou atualizar o ingresso. Ao clicar no botão de excluir, o ingresso é removido do servidor e a página é recarregada. Ao clicar no botão de atualizar, um formulário é exibido para modificar os detalhes do ingresso. As mudanças são enviadas ao servidor e a página é recarregada após a atualização.

Pasta controller

combinedController.js

```
// Importa os controladores de usuários e ingressos
const usersController = require('./userController');
const ticketsController = require('./ticketController');

// Define e exporta uma função para obter dados combinados de
usuários e ingressos
exports.getCombinedData = (req, res) => {
  // Obtém os dados de usuários do controlador de usuários
  const users = usersController.getDbData().users;
  // Obtém os dados de ingressos do controlador de ingressos
  const tickets = ticketsController.getTicketsData().tickets;

  // Envia uma resposta em formato JSON com os dados de usuários e
  ingressos
  res.json({ users, tickets });
};
```

Este código define uma função `getCombinedData` que combina dados de usuários e ingressos. Ele importa dois controladores (`userController` e `ticketController`), obtém os dados de usuários e ingressos desses controladores, e envia uma resposta JSON contendo esses dados.

TicketController.js

```

// Importa os módulos 'fs' (file system) e 'path'
const fs = require('fs');
const path = require('path');

// Define o caminho do arquivo JSON que armazena os dados dos ingressos
const TICKETS_DB = path.join(__dirname,
'../database/tickets.json');

// Função para obter dados dos ingressos
function getTicketsData() {
  // Verifica se o arquivo JSON existe; se não, retorna um objeto com
  uma lista vazia de ingressos
  if (!fs.existsSync(TICKETS_DB)) {
    return { tickets: [] };
  }
  // Lê o arquivo JSON e retorna os dados parseados
  return JSON.parse(fs.readFileSync(TICKETS_DB, 'utf-8'));
}

// Função para salvar dados dos ingressos
function saveTicketsData(data) {
  // Escreve os dados fornecidos no arquivo JSON, formatando-os com
  espaçamento de 2 caracteres
  fs.writeFileSync(TICKETS_DB, JSON.stringify(data, null, 2));
}

// Função para registrar um novo ingresso
exports.registerTicket = (req, res) => {
  // Extrai as propriedades do corpo da requisição
  const { nome, descricao, local, valor, data, tipo } = req.body;

  // Obtém os dados dos ingressos existentes
  let ticketsData = getTicketsData();

```

```

// Define um novo ID para o ingresso, incrementando o último ID
existente ou começando de 1 se não houver ingressos
const id = ticketsData.tickets.length ?
ticketsData.tickets[ticketsData.tickets.length - 1].id + 1 : 1;

// Adiciona o novo ingresso aos dados existentes
ticketsData.tickets.push({ id, nome, descricao, local, valor, data, tipo
});
// Salva os dados atualizados
saveTicketsData(ticketsData);

// Envia uma resposta de sucesso
res.send('Ingresso cadastrado com sucesso!');
};

// Função para obter todos os ingressos
exports.getTickets = (req, res) => {
// Obtém os dados dos ingressos
const ticketsData = getTicketsData();
// Envia os ingressos como resposta JSON
res.json(ticketsData.tickets);
};

// Função para deletar um ingresso por ID
exports.deleteTicket = (req, res) => {
// Extrai o ID do ingresso dos parâmetros da requisição
const { ticketId } = req.params;
// Obtém os dados dos ingressos
let ticketsData = getTicketsData();
// Filtra os ingressos para remover o ingresso com o ID fornecido
ticketsData.tickets = ticketsData.tickets.filter(ticket => ticket.id !==
parseInt(ticketId));
// Salva os dados atualizados
saveTicketsData(ticketsData);
};

```

```

// Envia uma resposta de sucesso
res.send(`Ingresso com ID ${ticketId} foi deletado.`);
};

// Função para obter um ingresso por ID
exports.getTicketById = (req, res) => {
  // Extrai o ID do ingresso dos parâmetros da requisição
  const { ticketId } = req.params;
  // Obtém os dados dos ingressos
  const ticketsData = getTicketsData();
  // Encontra o ingresso com o ID fornecido
  const ticket = ticketsData.tickets.find(ticket => ticket.id ===
parseInt(ticketId));

  // Se o ingresso não for encontrado, envia uma resposta de erro 404
  if (!ticket) {
    return res.status(404).send('Ingresso não encontrado.');
```

```

  }

  // Envia o ingresso encontrado como resposta JSON
  res.json(ticket);
};
```

```

// Função para atualizar um ingresso
exports.updateTicket = (req, res) => {
  // Extrai as propriedades do corpo da requisição
  const { id, nome, descricao, local, valor, data, tipo } = req.body;
  // Obtém os dados dos ingressos
  let ticketsData = getTicketsData();
  // Encontra o índice do ingresso com o ID fornecido
  const ticketIndex = ticketsData.tickets.findIndex(ticket => ticket.id
=== id);
```

```

  // Se o ingresso não for encontrado, envia uma resposta de erro
```

```

if (ticketIndex === -1) {
  return res.status(400).send('Ingresso não encontrado.');
```

}

```

// Atualiza as propriedades do ingresso encontrado com os novos
valores ou mantém os valores antigos se não forem fornecidos
ticketsData.tickets[ticketIndex] = {
  id,
  nome,
  descricao: descricao || ticketsData.tickets[ticketIndex].descricao,
  local: local || ticketsData.tickets[ticketIndex].local,
  valor: valor || ticketsData.tickets[ticketIndex].valor,
  data: data || ticketsData.tickets[ticketIndex].data,
  tipo: tipo || ticketsData.tickets[ticketIndex].tipo
};

// Salva os dados atualizados
saveTicketsData(ticketsData);
// Envia uma resposta de sucesso
res.send(`Ingresso ${id} foi atualizado.`);
};
```

Este código é um módulo que manipula operações CRUD (Create, Read, Update, Delete) para ingressos usando um arquivo JSON como banco de dados. Ele permite registrar, obter, deletar e atualizar ingressos, lendo e escrevendo dados no arquivo JSON.

user Controller

```

// Importa os módulos 'fs' (file system) e 'path'
const fs = require('fs');
const path = require('path');
```



```

// Define o caminho dos arquivos JSON que armazenam os dados dos
usuários e ingressos
const DATABASE = path.join(__dirname, '../database/users.json');
const TICKETS_DB = path.join(__dirname,
'../database/tickets.json');

// Função para obter dados dos ingressos
function getTicketsData() {
  if (!fs.existsSync(TICKETS_DB)) {
    return { tickets: [] };
  }
  return JSON.parse(fs.readFileSync(TICKETS_DB, 'utf-8'));
}

// Função para obter dados dos usuários
function getDbData() {
  return JSON.parse(fs.readFileSync(DATABASE, 'utf-8'));
}

// Função para salvar dados dos usuários
function saveDbData(data) {
  fs.writeFileSync(DATABASE, JSON.stringify(data, null, 2));
}

// Função para registrar um novo usuário
exports.register = (req, res) => {
  const { nome, sobrenome, documento, email, nDocumento, genero,
senha, telefone, nascimento } = req.body;

  let dbData = getDbData();
  const userExists = dbData.users.some(user => user.email === email);
  if (userExists) {
    return res.status(400).send('Email já cadastrado.');
```

```
const id = dbData.users.length ? dbData.users[dbData.users.length - 1].id + 1 : 1;
```

```
dbData.users.push({  
  id,  
  nome,  
  sobrenome,  
  documento,  
  email,  
  nDocumento,  
  genero,  
  senha,  
  telefone,  
  nascimento,  
});  
saveDbData(dbData);
```

```
res.send('Usuário registrado com sucesso.');
```

```
// Função para realizar login de um usuário
```

```
exports.login = (req, res) => {  
  const { email, senha } = req.body;  
  const dbData = getDbData();  
  const user = dbData.users.find(user => user.email === email);
```

```
  if (user && user.senha === senha) {  
    req.session.userEmail = user.email;  
    return res.status(200).json({ message: 'Login bem-sucedido',  
email: user.email, redirectUrl: '/logado' });  
  }
```

```
  res.status(400).json({ message: 'Email ou senha inválidos.' });
```

```
};
```

```
// Função para exibir o dashboard de um usuário autenticado
```

```
exports.dashboard = (req, res) => {
```

```
  if (!req.session.userId) {
```

```
    return res.redirect('/inicio');
```

```
  }
```

```
  res.send(`Logado, ${req.session.userName}! Bem-vindo ao sistema.`);
```

```
};
```

```
// Função para realizar logout de um usuário
```

```
exports.logout = (req, res) => {
```

```
  req.session.destroy();
```

```
  res.send('Sua sessão foi expirada.');
```

```
};
```

```
// Função para obter um usuário específico pelo email
```

```
exports.getUsers = (req, res) => {
```

```
  const { email } = req.query;
```

```
  const dbData = getDbData();
```

```
  const user = dbData.users.find(user => user.email === email);
```

```
  if (!user) {
```

```
    return res.status(404).json({ message: 'Usuário não encontrado.' });
```

```
  }
```

```
  res.json(user);
```

```
};
```

```
// Função para obter todos os usuários
```

```
exports.getUsersGeral = (req, res) => {
```

```
  const dbData = getDbData();
```

```

    res.json(dbData.users);
};

// Função para deletar um usuário pelo email
exports.deleteUser = (req, res) => {
    const { email } = req.body;
    let dbData = getDbData();
    dbData.users = dbData.users.filter(user => user.email !== email);
    saveDbData(dbData);
    res.send(`Usuário com email ${email} foi deletado.`);
};

// Função para atualizar um usuário pelo email
exports.updateUser = (req, res) => {
    const { email, nome, sobrenome, documento, nDocumento, genero,
    senha, telefone, nascimento } = req.body;
    let dbData = getDbData();
    const userIndex = dbData.users.findIndex(user => user.email ===
    email);

    if (userIndex === -1) {
        return res.status(400).json({ message: 'Usuário não encontrado.' });
    }

    const user = dbData.users[userIndex];
    user.nome = nome || user.nome;
    user.sobrenome = sobrenome || user.sobrenome;
    user.documento = documento || user.documento;
    user.nDocumento = nDocumento || user.nDocumento;
    user.genero = genero || user.genero;
    user.senha = senha || user.senha;
    user.telefone = telefone || user.telefone;
    user.nascimento = nascimento || user.nascimento;

```

```

saveDbData(dbData);

res.json({ message: `Usuário com email ${email} foi atualizado.` });
};

// Função para adicionar um ingresso aos pedidos de um usuário
exports.addTicketToPedidos = (req, res) => {
  const { userEmail, ticketId, quantidade } = req.body;
  let dbData = getDbData();
  let ticketsData = getTicketsData();
  const userIndex = dbData.users.findIndex(user => user.email ===
userEmail);
  if (userIndex === -1) {
    return res.status(400).send('Usuário não encontrado.');
```

```

  }

  const ticket = ticketsData.tickets.find(ticket => ticket.id === ticketId);
  if (!ticket) {
    return res.status(400).send('Ingresso não encontrado.');
```

```

  }

  if (!dbData.users[userIndex].pedidos) {
    dbData.users[userIndex].pedidos = [];
  }

  const novoIdTicket =
getNextTicketId(dbData.users[userIndex].pedidos);
  const pedido = { ...ticket, id: novoIdTicket, quantidade:
parseInt(quantidade, 10) };

  dbData.users[userIndex].pedidos.push(pedido);
  saveDbData(dbData);

  res.send('Ingresso adicionado aos pedidos do usuário com sucesso.');
```

```
};
```

```
// Função auxiliar para obter o próximo ID de pedido
```

```
function getNextTicketId(pedidos) {
```

```
  if (pedidos.length === 0) {
```

```
    return 1;
```

```
  }
```

```
  const maxId = Math.max(...pedidos.map(pedido => pedido.id));
```

```
  return maxId + 1;
```

```
}
```

```
// Função para remover um pedido de um usuário pelo ID do pedido
```

```
exports.removePedido = (req, res) => {
```

```
  const pedidoId = req.params.id;
```

```
  let dbData = getDbData();
```

```
  const pedidoIndex = dbData.users.findIndex(user => user.pedidos &&  
user.pedidos.some(pedido => pedido.id === parseInt(pedidoId)));
```

```
  if (pedidoIndex === -1) {
```

```
    return res.status(404).json({ message: 'Pedido não encontrado.' });
```

```
  }
```

```
  dbData.users[pedidoIndex].pedidos =
```

```
  dbData.users[pedidoIndex].pedidos.filter(pedido => pedido.id !==  
parseInt(pedidoId));
```

```
  saveDbData(dbData);
```

```
  res.json({ message: `Pedido com ID ${pedidoId} foi removido com  
sucesso.` });
```

```
};
```

```
// Função para obter um pedido específico pelo ID
```

```
exports.getPedidoById = (req, res) => {
```

```

const pedidoId = parseInt(req.params.pedidoId, 10);
const dbData = getDbData();

const userEmail = req.session.userEmail;
if (!userEmail) {
  return res.status(401).send('Usuário não autenticado.');
```

```

}

const user = dbData.users.find(user => user.email === userEmail);
if (!user) {
  return res.status(404).send('Usuário não encontrado.');
```

```

}

if (!user.pedidos || user.pedidos.length === 0) {
  return res.status(404).send('Usuário não possui pedidos.');
```

```

}

const pedido = user.pedidos.find(pedido => pedido.id === pedidoId);
if (!pedido) {
  return res.status(404).send('Pedido não encontrado.');
```

```

}

res.status(200).json({ pedido });
};

```

Este código é um módulo de back-end que gerencia operações CRUD para usuários e ingressos. Ele permite registrar e autenticar usuários, atualizar e deletar usuários, além de adicionar, remover e listar ingressos e pedidos de usuários, utilizando arquivos JSON como banco de dados.

Pasta router

combined.js

```
// Importa o módulo 'express'
const express = require('express');

// Cria um novo router utilizando o método Router() do Express
const routerCombined = express.Router();

// Importa o controlador responsável por lidar com as requisições para
dados combinados
const combinedController =
require('../controller/combinedController');

// Define uma rota GET na URL '/combined' que chama o método
'getCombinedData' do controlador
routerCombined.get('/combined',
combinedController.getCombinedData);

// Exporta o routerCombined para ser utilizado em outros arquivos
module.exports = routerCombined;
```

Este código define um router (routerCombined) utilizando o framework Express.js. Ele gerencia requisições relacionadas aos dados combinados de usuários e ingressos através da rota /combined. Essas requisições são tratadas pelo controlador (combinedController), onde são processadas operações como busca e combinação de dados de usuários e ingressos.

Router.js

```
const express = require('express');
const ticketController = require('../controller/ticketController');
```



```
const router = express.Router();
const cors = require('cors');

router.use(cors()); // Habilita o middleware CORS para todas as rotas
deste router

const path = require('path');

// Rota para a página inicial
router.get('/inicio', (req, res) => {
  res.sendFile(path.join(__dirname,
    '../..public/html/paginaInicial.html'));
});

// Rota para a página de login
router.get('/log', (req, res) => {
  res.render('login'); // Renderiza a view 'login'
});

// Rota para a página de cadastro
router.get('/cadastro', (req, res) => {
  res.render('cadastro'); // Renderiza a view 'cadastro'
});

// Rota para a página logada
router.get('/logado', (req, res) => {
  res.render('paginaLogada'); // Renderiza a view 'paginaLogada'
});

// Rota para a página de pedidos
router.get('/pedidos', (req, res) => {
  res.render('pedidos'); // Renderiza a view 'pedidos'
});
```

```
// Rota para a página de cadastro de empresa
router.get('/cadEmpresa', (req, res) => {
  res.render('cadAdm'); // Renderiza a view 'cadAdm'
});

// Rota para a página de opções de pagamento
router.get('/opPagamento', (req, res) => {
  res.render('opcaoPagamento'); // Renderiza a view
  'opcaoPagamento'
});

// Rota para a página de administração
router.get('/admin', (req, res) => {
  res.render('admin'); // Renderiza a view 'admin'
});

// Rota para a página de venda
router.get('/vender', (req, res) => {
  res.render('vender'); // Renderiza a view 'vender'
});

// Rota para a página de finalização de pagamento
router.get('/finalPagamento', (req, res) => {
  res.render('finalPagamento'); // Renderiza a view 'finalPagamento'
});

// Rota para a página de pagamento com cartão
router.get('/cartao', (req, res) => {
  res.render('cartao'); // Renderiza a view 'cartao'
});

// Rota para a página de pagamento com PIX
router.get('/pix', (req, res) => {
  res.render('pix'); // Renderiza a view 'pix'
});
```

```

});

// Rota para a página de perfil do usuário
router.get('/perfil', (req, res) => {
  res.render('perfil'); // Renderiza a view 'perfil'
});

// Rota para registrar um novo ingresso
router.post('/tickets', ticketController.registerTicket);

// Rota para obter todos os ingressos
router.get('/tickets', ticketController.getTickets);

// Rota para excluir um ingresso específico
router.delete('/tickets/:ticketId', ticketController.deleteTicket);

// Rota para atualizar um ingresso específico
router.put('/tickets/:ticketId', ticketController.updateTicket);

// Rota para obter informações de um ingresso específico
router.get('/tickets/:ticketId', ticketController.getTicketById);

module.exports = router;

```

Este código organiza as rotas da aplicação, separando claramente a lógica de renderização de views estáticas das operações relacionadas aos ingressos, seguindo as práticas comuns de desenvolvimento web com Node.js e Express.js.

Users.js

```

const express = require('express');
const routerU = express.Router();
const usersController = require('../controller/userController');

```

```
// Definindo as rotas usando o routerU

// Rota para registrar um novo usuário
routerU.post('/register', usersController.register);

// Rota para autenticar um usuário (login)
routerU.post('/login', usersController.login);

// Rota para acessar o dashboard do usuário
routerU.get('/dashboard', usersController.dashboard);

// Rota para fazer logout do usuário
routerU.post('/logout', usersController.logout);

// Rota para obter informações de um usuário específico por email
routerU.get('/users', usersController.getUsers);

// Rota para obter a lista geral de usuários
routerU.get('/listUsers', usersController.getUsersGeral);

// Rota para deletar um usuário por email
routerU.delete('/users', usersController.deleteUser);

// Rota para atualizar informações de um usuário por email
routerU.put('/users', usersController.updateUser);

// Rota para obter informações de um pedido específico por ID de
pedido
routerU.get('/pedido/:pedidoId', usersController.getPedidoById);

// Rota para adicionar um ingresso aos pedidos de um usuário
routerU.post('/addTicketToPedidos',
usersController.addTicketToPedidos);
```

```
// Rota para remover um pedido específico por ID de pedido
routerU.delete('/removerPedido/:id', usersController.removerPedido);

// Rota para renderizar a página de opção de pagamento para um
pedido específico
routerU.get('/opPagamento/:pedidoId', (req, res) => {
  const pedidoId = req.params.pedidoId; // Obtém o ID do pedido da
  URL

  // Aqui, 'getPedidoById' deve ser 'usersController.getPedidoById'
  para usar o método correto do controlador
  const pedido = usersController.getPedidoById(pedidoId); // Obtém
  informações detalhadas do pedido

  res.render('opcaoPagamento', { pedido }); // Renderiza a view
  'opcaoPagamento' com os dados do pedido
});

module.exports = routerU;
```

Este código organiza as rotas relacionadas a usuários e pedidos, delegando as operações específicas para funções definidas no controlador `UserController`. Ele também inclui a rota de renderização para a página de opção de pagamento, que utiliza dados obtidos do controlador antes de renderizar a view.

pasta Database

`tickets.json`

```
{
  "tickets": [
```

```

{
  "id": 2,
  "nome": "podyPary",
  "descricao": "evento que contem a presença de artistas como kyan
e etc.",
  "local": "são paulo-SP",
  "valor": "300",
  "data": "2024-06-29",
  "tipo": "Show"
},
{
  "id": 3,
  "nome": "vintage - Derek",
  "descricao": "evento que contem a presença de um grande nome da
cena do trap",
  "local": "taboão-SP",
  "valor": "60",
  "data": "2024-07-14",
  "tipo": "Show"
},
{
  "id": 3,
  "nome": "Bruno mars",
  "descricao": "grande nome da cultura pop, presente em nosso
país",
  "local": "são paulo-SP",
  "valor": "100",
  "data": "2024-08-17",
  "tipo": "Show"
}
]
}

```

Esse código são dados estruturados em formato JSON, especificamente uma lista de ingressos (tickets) com suas respectivas informações de Eventos da pagina principal da Api.

users.json

```
{
  "users": [
    {
      "id": 1,
      "nome": "danilo",
      "sobrenome": "Oliveira",
      "documento": "rg",
      "email": "danilo2006@gmail.com",
      "nDocumento": "67897678978",
      "genero": "masculino",
      "senha": "123456789",
      "telefone": "578987897",
      "nascimento": "2002-07-10",
      "pedidos": []
    },
    {
      "id": 2,
      "nome": "Otavio",
      "sobrenome": "Ramos",
      "documento": "cpf",
      "email": "otavioRamos@gmail.com",
      "nDocumento": "123456788",
      "genero": "masculino",
      "senha": "987654321",
      "telefone": "1234645576",
      "nascimento": "2004-06-06",
      "pedidos": [
```

```

{
  "id": 1,
  "nome": "Plantão festival",
  "descricao": "O Plantão Festival é o primeiro festival de rap
realizado em Fortaleza, que acontece no Marina Park Hotel. O evento
conta com shows de grandes nomes do rap nacional como Matuê,
Teto, WIU, Marcelo D2, L7nnon, entre outros. Ele contribui para o
movimento cultural e turístico da cidade, atraindo fãs de várias regiões
do país",
  "local": "cotia-SP",
  "valor": "300",
  "data": "2024-10-11",
  "tipo": "Exposições",
  "quantidade": 3
},
{
  "id": 2,
  "nome": "Plantão festival",
  "descricao": "O Plantão Festival é o primeiro festival de rap
realizado em Fortaleza, que acontece no Marina Park Hotel. O evento
conta com shows de grandes nomes do rap nacional como Matuê,
Teto, WIU, Marcelo D2, L7nnon, entre outros. Ele contribui para o
movimento cultural e turístico da cidade, atraindo fãs de várias regiões
do país",
  "local": "cotia-SP",
  "valor": "300",
  "data": "2024-10-11",
  "tipo": "Exposições",
  "quantidade": 3
}
]
}
]
}

```


Esse código são dados estruturados em formato JSON, especificamente uma lista de usuários e ingressos (tickets) com suas respectivas informações de Eventos da página principal da Api.

Pasta view => pasta partials

footer.ejs

```
<footer>
  <p>&copy; 2024 Meu Site - Todos os direitos reservados <br> <br>
    Telefone para Contato: 11 4002-8922 <br> <br>
    Central de Reclamações:
    ticketinnovatorsreclame@hotmail.com
  </p>
</footer>
```

Esta parte do código pertence ao footer de todas as páginas do site que possuem um.

header.ejs

```
<header>
  <div class="logo"></div>
  <div class="menu">
    <a href="/log">Entre </a>
    <a href="/cadastro">Cadastre-se </a>

  </div>
</header>
```

Esta parte do código pertence ao Header de todas as páginas do site que possuem um.

headerAdm.ejs

```
<header>
  <div class="logo"><a href="/admin"></a></div>
  <div class="menu">
    <a href="/vender">Cadastrar Ingresso</a>
    <a href="/admin">Home</a>
  </div>
</header>
```

Esta parte do código pertence ao Header da parte do administrador, de todas as páginas do site.

```
<header>
  <div class="logo"><a href="/logado"></a></div>
  <div class="menu">
    <a href="/perfil">Meu perfil</a>
    <a href="/pedidos"></a>
    <a href="/logado">Home</a>
  </div>
</header>
```

headerLog.ejs

```

<header>
  <div class="logo"><a href="/logado"></a></div>
  <div class="menu">
    <a href="/perfil">Meu perfil</a>
    <a href="/pedidos"></a>
    <a href="/logado">Home</a>
  </div>
</header>

```

headerLog.ejs

Esta parte do código pertence ao Header da parte do administrador após logado.

admin.ejs

```

<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Página de Início</title>
  <link rel="stylesheet" href="../css/sytlePi.css">
  <link rel="stylesheet" href="../css/styleCard.css">

</head>
<body>
  <style>
    form {
      max-width: 600px;
      margin: 20px auto;
    }
  </style>

```

```
padding: 20px;
border: 1px solid #ccc;
border-radius: 10px;
background-color: #f9f9f9;
}
```

```
form h2 {
  text-align: center;
  margin-bottom: 20px;
}
```

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

```
.form-group label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}
```

```
.form-group input,
.form-group select {
  width: calc(100% - 22px);
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
  font-size: 16px;
}
```

```
button[type="submit"] {
  width: 100%;
  padding: 10px;
  border: none;
```

```

border-radius: 5px;
background-color: #28a745;
color: white;
font-size: 16px;
cursor: pointer;
}

button[type="submit"]:hover {
  background-color: #218838;
}
</style>
<%- include('partials/headerAdm') %>
<div class="container">
  <div class="section">
    <h2>Exposições</h2>
    <p>Aqui você encontra informações sobre as últimas
exposições.</p>
  </div>
  <div class="section">
    <h2>Shows</h2>
    <p>Aqui você encontra informações sobre os shows mais
esperados.</p>
  </div>
  <div class="section">
    <h2>Esportes</h2>
    <p>Aqui você encontra informações sobre eventos
esportivos.</p>
  </div>
</div>
<div class="card-container" id="card-container">

</div>
</body>
<footer>

```

```
<p>&copy; 2024 Meu Site - Todos os direitos reservados <br> <br>
    Telefone para Contato: 11 4002-8922 <br> <br>
    Central de Reclamações:
ticketinnovatorsreclame@hotmail.com
</p>
</footer>
<script src='../js/scriptDel.js'></script>

</html>
```

Aqui na pagina do administrador temos a tela principal onde os eventos adicionados por ele acabam indo.

cadAdm.ejs

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Formulário de Cadastro</title>

</head>
<body>
  <%- include('partials/header') %>
  <div class="container">
    <h1>Junte-se a nós</h1>
    <form action="#">
      <label for="nome">Nome da empresa:</label>
      <input type="text" id="nome" name="nome" required>

      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required>
```

```
<label for="numero_documento">CNPJ da empresa:</label>
<input type="text" id="numero_documento"
name="numero_documento" required>
```

```
<label for="sexo">Categoria de Ingressos:</label>
```

```
<label for="codigo_pais">Código do País (para o número de
telefone):</label>
<input type="text" id="codigo_pais" name="codigo_pais"
required>
```

```
<label for="numero_telefone">Número de Telefone:</label>
<input type="text" id="numero_telefone"
name="numero_telefone" required>
```

```
<button class="submit-btn" type="submit">Enviar</button>
</form>
</div>
<%- include('partials/footer') %>
</body>
</html>
```

```
<style>
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: white;
  color: black;
}
```

```
h1{
  text-transform: uppercase;
  color: #001275;
  text-align: center;
}
```

```
header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #003366;
  padding: 10px 20px;
  height: 80px;
}
```

```
.logo {
  float: left;
  line-height: 20px;
  height: 100%;
  display: flex;
  align-items: center;
}
```

```
nav a {
  color: white;
  text-decoration: none;
  margin: 0 10px;
  font-size: 18px;
}
```

```
nav a.cart-icon {
  font-size: 24px;
}
```



```
main {  
  padding: 20px;  
}  
.container {  
  max-width: 700px;  
  margin: 0 auto;  
  margin-top: 80px;  
  padding: 60px;  
  background-color: #666;  
  color: white;  
}  
  
h1 {  
  margin: 0;  
  line-height: 60px;  
}  
  
label {  
  display: block;  
  margin-bottom: 5px;  
}  
  
input,  
select {  
  width: calc(100% - 10px);  
  padding: 8px;  
  margin-bottom: 10px;  
  border: 1px solid #ccc;  
  border-radius: 4px;  
  box-sizing: border-box;  
}  
  
.submit-btn {  
  background-color: #001275;
```

```

    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

.submit-btn:hover {
    background-color: #000000;
}

footer {
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 20px 0;
    margin-top: 300px;
    bottom: 0;
    width: 100%;
    position: fixed;
}

header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color: #133579;
    color: white;
    padding: 30px;
    height: 100px;
}

.carrinho{
    width: 30px;
}

```

```
body {
  font-family: Arial, sans-serif;
  background-color: #f5f5f5;
}

.container {
  max-width: 400px;
  margin: 40px auto;
  padding: 20px;
  background-color: rgba(127,127,127,255);
  border: 1px solid rgba(127,127,127,255);
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  margin-top: 70px;
}

.logo {
  text-align: center;
  margin-bottom: 20px;
}

.logo img {
  width: 300px;
  height: auto;

;
}

form {
  display: flex;
  flex-direction: column;
  align-items: center;
}
```

```
h2 {  
  margin-top: 0;  
  color: #001275;  
}
```

```
input[type="email"], input[type="password"] {  
  width: 100%;  
  height: 40px;  
  margin-bottom: 20px;  
  padding: 10px;  
  border: 1px solid #ccc;  
  border-radius: 5px;  
}
```

```
input[type="email"]:focus, input[type="password"]:focus {  
  border-color: #aaa;  
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
}
```

```
.btn-login {  
  width: 100%;  
  height: 40px;  
  background-color: #001275;  
  color: #fff;  
  border: none;  
  border-radius: 5px;  
  cursor: pointer;  
}
```

```
.conta {  
  margin-top: 20px;
```

```

    text-align: center;
    font-weight: bold;
}

p a {
    text-decoration: none;
    color: rgba(0,18,117,255);
}

p a:hover {
    color: white;
}
img{
    width:80px;
    height: auto;

}
.carrinho{
width: 30px;
}
header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color: #133579;
    color: white;
    padding: 30px;
    height: 100px;
}

.logo {
    font-size: 24px;
    font-weight: bold;

```

```
}
```

```
.menu {  
  display: flex;  
  gap: 20px;  
}
```

```
.menu a {  
  color: white;  
  text-decoration: none;  
  font-weight: bold;  
  font-size: 20px;  
  margin-top: 20px;  
}
```

```
.menu a:hover {  
  text-decoration: underline;  
}
```

```
.container {  
  padding: 60px;  
  border-radius: 10px;  
}
```

```
.section {  
  margin-bottom: 40px;  
}
```

```
.section h2 {  
  margin-bottom: 10px;  
  color: #333;  
}
```

```

.section p {
  color: #666;
}

footer {
background-color: #333;
color: #fff;
text-align: center;
padding: 20px 0;
margin-top: 300px;
bottom: 0;
width: 100%;
position: fixed;
}
.fundo{
background-image: url(jornal,.png);
}
.imagem{
width: 80px;
}

</style>

```

Esta parte do código faz parte do cadastro do administrador.

cadastro.ejs

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">

```

```

<title>Formulário de Cadastro</title>
<link rel="stylesheet" href="css/cadastro.css">
</head>
<body>
  <%- include('partials/header') %>
  <br>
  <br>
  <div class="container">
    <h1>Se Cadastre</h1>
    <form id="eventoForm" action="#">
      <label for="nome">Nome:</label>
      <input type="text" id="nome" name="nome" required>

      <label for="sobrenome">Sobrenome:</label>
      <input type="text" id="sobrenome" name="sobrenome"
required>

      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required>

      <label for="tipo_documento">Tipo de Documento:</label>
      <select id="tipo_documento" name="tipo_documento"
required>
        <option value="">Selecione...</option>
        <option value="rg">RG</option>
        <option value="cpf">CPF</option>
        <option value="passport">Passaporte</option>
      </select>

      <label for="numero_documento">Número do
Documento:</label>
      <input type="text" id="numero_documento"
name="numero_documento" required>

```



```

<label for="sexo">Sexo:</label>
<select id="sexo" name="sexo" required>
  <option value="">Selecione...</option>
  <option value="masculino">Masculino</option>
  <option value="feminino">Feminino</option>
  <option value="outro">Outro</option>
</select>

<label for="codigo_pais">Senha:</label>
<input type="text" id="senha" name="codigo_pais" required>

<label for="numero_telefone">Número de Telefone:</label>
<input type="text" id="numero_telefone"
name="numero_telefone" required>

<label for="data_nascimento">Data de Nascimento:</label>
<input type="date" id="data_nascimento"
name="data_nascimento" required>

<button class="submit-btn" type="submit">Enviar</button>
</form>
</div>
</body>
<%- include('partials/footer') %>
<script src="js/scriptCad.js"></script>

</html>

```

Este é o cadastro do usuario.

cartao.ejs

```

<!DOCTYPE html>
<html lang="pt-BR">

```

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Pagamento com Cartão</title>
  <link rel="stylesheet" href="/css/login.css">
  <style>
    /* Estilos gerais para o corpo da página */
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f4f4f4;
    }

    /* Estilos para o elemento main */
    main {
      padding: 1em;
    }

    /* Estilos para o container principal */
    .container {
      padding: 20px;
      margin-top: 50px;
      margin-bottom: 200px;
      background-color: #133579;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      text-align: center;
    }

    /* Estilos para os grupos de formulário */
    .form-group {
      margin-bottom: 20px;

```

```

}

/* Estilos para os labels */
label {
    display: block;
    margin-bottom: 5px;
    color: white;
}

/* Estilos para os inputs e spans */
input, span {
    display: block;
    width: 100%;
    padding: 8px;
    border: 1px solid #ccc;
    border-radius: 4px;
    background-color: #fff;
    margin-bottom: 10px;
}

/* Estilos para o botão */
button {
    padding: 10px 15px;
    background-color: #001275;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

/* Estilos para o botão ao passar o mouse */
button:hover {
    background-color: #003493;
}

```

```

/* Estilos para o rodapé */
footer {
  background-color: #333;
  color: #fff;
  text-align: center;
  padding: 20px 0;
  bottom: 0;
  width: 100%;
  position: fixed;
}
</style>
</head>
<body>
  <!-- Inclui o cabeçalho logado -->
  <%- include('partials/headerLog') %>
  <main>
    <div class="container">
      <h1>Pagamento com Cartão</h1>
      <form id="cartaoForm">
        <!-- Grupo de formulário para nome completo -->
        <div class="form-group">
          <label for="name">Nome Completo</label>
          <span id="name"><%= user.nome %> <%= user.sobrenome
%></span>
        </div>
        <!-- Grupo de formulário para CPF -->
        <div class="form-group">
          <label for="cpf">CPF</label>
          <span id="cpf"><%= user.nDocumento %></span>
        </div>
        <!-- Grupo de formulário para número do cartão -->
        <div class="form-group">
          <label for="cardNumber">Número do Cartão</label>

```

```

        <input type="text" id="cardNumber"
name="cardNumber">
    </div>
    <!-- Grupo de formulário para data de expiração -->
    <div class="form-group">
        <label for="expirationDate">Data de Expiração</label>
        <input type="text" id="expirationDate"
name="expirationDate">
    </div>
    <!-- Grupo de formulário para CVV -->
    <div class="form-group">
        <label for="cvv">CVV</label>
        <input type="text" id="cvv" name="cvv">
    </div>
    <!-- Grupo de formulário para valor -->
    <div class="form-group">
        <label for="amount">Valor (R$)</label>
        <span id="amount"><%= pedido.valor * pedido.quantidade
%></span>
    </div>
    <!-- Botão para confirmar pagamento -->
    <button type="button"
onclick="displayAmount()">Confirmar Pagamento</button>
</form>
    <!-- Local onde o valor será exibido -->
    <div id="amountDisplay"></div>
    <!-- Espaço entre o formulário e o footer -->
    <div style="height: 50px;"></div>
</div>
</main>
<!-- Inclui o rodapé -->
<%- include('partials/footer') %>
<script>
    // Função para exibir o valor a ser pago

```

```

    function displayAmount() {
        const amount =
document.getElementById('amount').textContent;
        const amountDisplay =
document.getElementById('amountDisplay');
        amountDisplay.innerHTML = `

<strong>Valor a ser
pago:</strong> R$ ${amount}</p>`;
    }
</script>
</body>
</html>


```

Este código HTML renderiza uma página de pagamento com cartão, incluindo campos de entrada para informações do cartão e dados do usuário, e um botão para confirmar o pagamento. Ele também inclui estilos personalizados e uma função JavaScript para exibir o valor total a ser pago.

dinheiro.ejs

```

<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Pagamento com Boleto Bancário</title>
    <link rel="stylesheet" href="/css/login.css">
    <style>
        /* Estilos gerais para o corpo e o main */
        body {
            font-family: Arial, sans-serif;
            margin: 0;

```

```

padding: 0;
background-color: #f4f4f4;
}

main {
padding: 1em;
}

/* Estilo para a lista de pedidos */
#order-list {
list-style-type: none;
padding: 0;
}

#order-list li {
background-color: white;
margin-bottom: 1em;
padding: 1em;
border-radius: 5px;
box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
cursor: pointer;
}

.order-header {
display: flex;
justify-content: space-between;
align-items: center;
font-weight: bold;
}

.order-details {
margin-top: 0.5em;
}

```

```

/* Estilo para o footer */
footer {
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 20px 0;
    margin-top: 300px;
    bottom: 0;
    width: 100%;
    position: fixed;
}

/* Estilos para o container do formulário */
.container {
    padding: 20px;
    margin-top: 50px;
    margin-bottom: 200px;
    background-color: #133579;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
}

.form-group {
    margin-bottom: 20px;
}

label {
    display: block;
    margin-bottom: 5px;
    color: white;
}

span {

```



```

display: block;
padding: 8px;
border: 1px solid #ccc;
border-radius: 4px;
background-color: #fff;
}

button {
padding: 10px 15px;
background-color: #001275;
color: white;
border: none;
border-radius: 4px;
cursor: pointer;
}

button:hover {
background-color: #003493;
}
</style>
</head>
<body>
<%- include('partials/headerLog') %> <!-- Inclui o header -->
<main>
<div class="container">
<h1>Pagamento com Boleto Bancário</h1>
<form id="boletoForm">
<!-- Campo para exibir o nome completo do usuário -->
<div class="form-group">
<label for="name">Nome Completo</label>
<span id="name"><%= user.nome %> <%= user.sobrenome
%></span>
</div>
<!-- Campo para exibir o CPF do usuário -->

```

```

<div class="form-group">
  <label for="cpf">CPF</label>
  <span id="cpf"><%= user.nDocumento %></span>
</div>
<!-- Campo para exibir o valor total -->
<div class="form-group">
  <label for="amount">Valor (R$)</label>
  <span id="amount"><%= pedido.valor * pedido.quantidade
%></span>
</div>
<!-- Botão para gerar o boleto -->
<button type="button" onclick="generateBoleto()">Gerar
Boleto</button>
</form>
<!-- Div para exibir os detalhes do boleto gerado -->
<div id="boletoContainer" class="boleto-container"></div>
<!-- Espaço entre o formulário e o footer -->
<div style="height: 50px;"></div>
<!-- Botão oculto para avançar -->
<button id="nextButton" class="hidden">Avançar</button>
</div>
</main>
<%- include('partials/footer') %> <!-- Inclui o footer -->
<script>
function generateBoleto() {
  // Obtém os dados do usuário
  const name = document.getElementById('name').textContent;
  const cpf = document.getElementById('cpf').textContent;
  const amount =
document.getElementById('amount').textContent;

  // Verifica se todos os dados estão presentes
  if (!name || !cpf || !amount) {
    alert('Erro ao obter os dados do usuário.');
```

```

    return;
}

// Limpa o conteúdo do container do boleto
const boletoContainer =
document.getElementById('boletoContainer');
boletoContainer.innerHTML = '';

// Cria os detalhes do boleto
const boletoDetails = `
    <h2>Boleto Bancário</h2>
    <p><strong>Nome:</strong> ${name}</p>
    <p><strong>CPF:</strong> ${cpf}</p>
    <p><strong>Valor:</strong> R$ ${amount}</p>
    <p><strong>Linha Digitável:</strong> 12345.67890
12345.678901 12345.678901 1 23456789012345</p>
`;

boletoContainer.innerHTML = boletoDetails;

// Adiciona a imagem do código de barras
const barcodeImg = document.createElement('img');
barcodeImg.src =
'https://via.placeholder.com/350x70.png?text=Código+de+Barras';
boletoContainer.appendChild(barcodeImg);
}

function redirectToFinal() {
    // Redireciona para a página de pagamento final
    window.location.href = '/finalPagamento';
}
</script>
</body>
</html>

```

Esta página realiza o pagamento com boleto bancário, exibindo os dados do usuário e o valor do pedido, e gerando os detalhes do boleto e código de barras. O código também inclui estilos personalizados e funções JavaScript para manipulação do boleto.

Final pagamento.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <link rel="stylesheet" href="/css/login.css">
  <title>Document</title>
</head>
<body>
  <%- include('partials/headerLog') %>

<main>
  <h1> Seu pedido foi finalizado!</h1>
  <p> Obrigado por comprar com a Innovators Ticket! Para visualizar
seu pedido, clique no ícone de "Carrinho" para checar os detalhes da
sua compra. </p>
</main>
<%- include('partials/footer') %>
</body>
<style>
  body{
    background-color: white;
    text-align: center;
  }
```

```
main {  
  padding: 20px;  
}
```

```
main{  
  border: 1px 1px 1px solid black;  
  display: inline-block;  
  margin: 300px ;  
  color:white;  
  width: 50%;  
  padding: 20px;  
  
  font-size: 20px;  
  background-color: rgb(3, 3, 132);  
}
```

```
h1{  
  font-size: 30px;  
}
```

```
@media (max-width: 768px) {  
  main {  
    width: 70%;  
    margin: 150px auto;  
    font-size: 18px;  
    padding: 15px;  
  }
```

```
  h1 {  
    font-size: 25px;  
  }  
}
```

```

@media (max-width: 480px) {
  main {
    width: 90%;
    margin: 100px auto;
    font-size: 16px;
    padding: 10px;
  }

  h1 {
    font-size: 20px;
  }
}
</style>
</html>

```

Mensagem final exibida ao usuario apos a compra.

Login.ejs

```

<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Ticket - Login</title>
  <link rel="stylesheet" href="/css/login.css">
</head>
<body>
  <%- include('partials/header') %> <!-- Inclui o cabeçalho -->
  <div class="container">
    <div class="logo">

```

```

         <!-- Exibe a
logo -->
    </div>
    <form id="loginForm">
        <h2>Login</h2>
        <!-- Campos de entrada para email e senha -->
        <input type="email" id="email" placeholder="Insira o Email"
required>
        <input type="password" id="senha" placeholder="Senha"
required>
        <!-- Botão de login -->
        <button type="submit" class="btn-login">Entrar</button>
        <!-- Link para a página de cadastro -->
        <p class="conta">Não tem conta? <a
href="/cadastro">Cadastre-se</a></p>
    </form>
</div>
<%- include('partials/footer') %> <!-- Inclui o rodapé -->
<script>
    // Adiciona um listener para o evento de submit do formulário

document.getElementById('loginForm').addEventListener('submit',
async function(event) {
    event.preventDefault(); // Evita o comportamento padrão do
formulário

    const email = document.getElementById('email').value;
    const senha = document.getElementById('senha').value;

    // Verifica se o email e senha são de um administrador
    if (email === 'admin@123' && senha === 'admin') {
        window.location.href = '/admin/dashboard'; // Redireciona
para o dashboard do admin
        return;

```

```

    }

    // Faz uma requisição POST para a rota de login
    const response = await fetch('/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ email, senha })
    });

    const result = await response.json();

    if (response.ok) {
      localStorage.clear(); // Limpa o localStorage
      localStorage.setItem('userEmail', result.email); // Armazena o
email do usuário
      window.location.href = result.redirectUrl; // Redireciona o
usuário
    } else {
      alert(result.message); // Exibe uma mensagem de erro
    }
  });
</script>
</body>
</html>

```

Essa página de login permite que os usuários entrem com email e senha, redirecionando administradores para um painel especial. Também valida os dados e comunica com o servidor para autenticação.

opicaoPagamento.ejs


```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <link rel="stylesheet" href="/css/login.css">
  <title>Pagamento</title>
</head>
<body>

  <%- include('partials/headerLog') %> <!-- Inclui o cabeçalho -->
  <div class="container">
    <div class="payment-option" data-payment-type="dinheiro">
      
      <p class="highlight-on-hover"><a
href="/dinheiro">Dinheiro</a></p>
    </div>
    <div class="payment-option" data-payment-type="cartao">
      
      <p class="highlight-on-hover"><a href="/cartao">Cartão de
Crédito e Débito</a></p>
    </div>
    <div class="payment-option" data-payment-type="pix">
      
      <p class="highlight-on-hover"><a href="/pix">Pix</a></p>
    </div>

  </div>
  <%- include('partials/footer') %> <!-- Inclui o rodapé -->

  <style>

```

```
/* Estilo da página */
body {
    margin: 0;
    padding: 0;
    background-color: white;
}

main {
    padding: 20px;
}

h1 {
    margin: 0;
    font-size: 60px;
    font-family: Arial, Helvetica, sans-serif;
    font-weight: bold;
    color: white;
}

h2 {
    margin: 0;
    font-size: 45px;
    color: white;
    font-family: Arial, Helvetica, sans-serif;
}

.container {
    max-width: 600px;
    height: auto;
    margin: 0 auto;
    padding: 90px;
    margin-top: 50px;
    margin-bottom: 200px;
    background-color: white;
}
```

```

    border: 0;
}

.payment-option {
    display: flex;
    align-items: center;
    box-shadow: 1px 1px 2px 2px black;
    padding: 20px;
}

.payment-option img {
    width: 100px;
    margin-right: 20px;
    padding-bottom: 30px;
    padding-top: 40px;
}

.payment-option p {
    margin: 0;
    color: black;
    font-weight: bold;
    margin-left: 30px;
    font-size: 30px;
}

.or {
    text-align: center;
    margin-top: 40px;
    font-size: 30px;
    font-weight: bold;
}

.gift-card-container {
    text-align: center;

```

```
    color: black;
}

.gift-card {
    width: 200px;
}

.gift-card-text {
    display: inline-block;
    vertical-align: middle;
    color: black;
    font-weight: bold;
    padding-bottom: 150px;
    font-size: 25px;
}

a {
    text-decoration: none;
    color: black;
}

.gift-card-text:hover {
    transform: scale(1.05);
    transition: transform 0.2s ease-in-out;
    cursor: pointer;
    color: black;
    background-color: #133579;
    padding: 10px;
    border-radius: 10px;
}

.highlight-on-hover:hover {
    transform: scale(1.05);
    transition: transform 0.2s ease-in-out;
```

```

    cursor: pointer;
    color: black;
    background-color: #133579;
    padding: 10px;
    border-radius: 10px;
}

@media (max-width: 768px) {
    .container {
        padding: 40px;
    }

    .payment-option {
        flex-direction: column;
        text-align: center;
    }

    .payment-option img {
        margin-right: 0;
        margin-bottom: 10px;
    }

    .payment-option p {
        font-size: 24px;
    }

    .gift-card-text {
        font-size: 20px;
    }

    .or {
        font-size: 24px;
    }
}

```

```

@media (max-width: 480px) {
  .container {
    padding: 20px;
  }

  .payment-option p {
    font-size: 18px;
  }

  .gift-card-text {
    font-size: 18px;
  }

  .or {
    font-size: 20px;
  }
}
</style>

<script>
  document.addEventListener('DOMContentLoaded', function() {
    const paymentOptions =
document.querySelectorAll('.payment-option');

    paymentOptions.forEach(function(option) {
      option.addEventListener('click', function() {
        const paymentType =
option.getAttribute('data-payment-type');
        if (paymentType) {
          const pedidoId = 2; // Substitua pelo ID do pedido
apropriado
          window.location.href = `/${paymentType}/${pedidoId}`;
        }
      }
    }
  )
}

```

```

    });
  });

  const giftCardText = document.querySelector('.gift-card-text');
  giftCardText.addEventListener('click', function() {
    alert('Você escolheu usar um cupom de desconto/vale
presente.');
```

```

  });
});
</script>
</body>
</html>

```

Esta página HTML exibe opções de pagamento (Dinheiro, Cartão, Pix) e redireciona para as páginas correspondentes ao clicar. Inclui cabeçalho, rodapé e estilos responsivos, além de um script para tratar cliques nas opções de pagamento.

paginaLogada.ejs

```

<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Página de Início</title>
  <link rel="stylesheet" href="../css/sytlePi.css">
  <link rel="stylesheet" href="../css/styleCard.css">

</head>
<body>

  <%- include('partials/headerLog') %>

```

```

<div class="container">
  <div class="section">
    <h2>Exposições</h2>
    <p>Aqui você encontra informações sobre as últimas
exposições.</p>
  </div>
  <div class="section">
    <h2>Shows</h2>
    <p>Aqui você encontra informações sobre os shows mais
esperados.</p>
  </div>
  <div class="section">
    <h2>Esportes</h2>
    <p>Aqui você encontra informações sobre eventos
esportivos.</p>
  </div>
  <div class="card-container" id="card-container">

  </div>
</body>
<footer>
  <p>&copy; 2024 Meu Site - Todos os direitos reservados <br> <br>
    Telefone para Contato: 11 4002-8922 <br> <br>
    Central de Reclamações:
    ticketinnovatorsreclame@hotmail.com
  </p>
</footer>
<script src="../js/scriptCard.js"></script>
</html>

```

Esta pagina apresenta a pagina principal para o clienete apos logar.

pedidos.ejs


```

<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <!-- Configurações básicas do documento HTML -->
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Meus Pedidos</title>
  <link rel="stylesheet" href="../css/stylePi.css">
  <style>
    /* Estilos para o corpo da página */
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f4f4f4;
    }

    /* Estilos para o cabeçalho */
    header {
      background-color: #001275;
      color: white;
      padding: 1em;
      text-align: center;
    }

    /* Estilos para a seção principal */
    main {
      padding: 1em;
    }

    /* Estilos para a lista de pedidos */
    #order-list {

```

```

    list-style-type: none;
    padding: 0;
}

/* Estilos para os itens da lista de pedidos */
#order-list li {
    background-color: white;
    margin-bottom: 1em;
    padding: 1em;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    cursor: pointer;
}

/* Estilos para o cabeçalho de cada pedido */
.order-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    font-weight: bold;
}

/* Estilos para os detalhes de cada pedido */
.order-details {
    margin-top: 0.5em;
}

/* Estilos para o rodapé */
footer {
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 20px 0;
    margin-top: 300px;
}

```

```

    bottom: 0;
    width: 100%;
    position: fixed;
}

/* Estilos para o container principal */
.container {
    max-width: 400px;
    margin: 40px auto;
    padding: 20px;
    background-color: rgba(127, 127, 127, 255);
    border: 1px solid rgba(127, 127, 127, 255);
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    margin-top: 70px;
}

/* Estilos para a logo */
.logo {
    text-align: center;
    margin-bottom: 20px;
}

/* Estilos para a imagem da logo */
.logo img {
    width: 300px;
    height: auto;
}

/* Estilos para o formulário */
form {
    display: flex;
    flex-direction: column;
    align-items: center;
}

```

```
/* Estilos para o título */
```

```
h2 {  
    margin-top: 0;  
    color: #001275;  
}
```

```
/* Estilos para os campos de entrada de email e senha */
```

```
input[type="email"], input[type="password"] {  
    width: 100%;  
    height: 40px;  
    margin-bottom: 20px;  
    padding: 10px;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
}
```

```
/* Estilos para o foco nos campos de entrada */
```

```
input[type="email"]:focus, input[type="password"]:focus {  
    border-color: #aaa;  
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
}
```

```
/* Estilos para o botão de login */
```

```
.btn-login {  
    width: 100%;  
    height: 40px;  
    background-color: #001275;  
    color: #fff;  
    border: none;  
    border-radius: 5px;  
    cursor: pointer;  
}
```

```
/* Estilos para a seção de conta */
```

```
.conta {  
    margin-top: 20px;  
    text-align: center;  
    font-weight: bold;  
}
```

```
/* Estilos para os links */
```

```
p a {  
    text-decoration: none;  
    color: rgba(0, 18, 117, 255);  
}
```

```
/* Estilos para os links ao passar o mouse */
```

```
p a:hover {  
    color: white;  
}
```

```
/* Estilos para as imagens */
```

```
img {  
    width: 80px;  
    height: auto;  
}
```

```
/* Estilos para o ícone do carrinho */
```

```
.carrinho {  
    width: 30px;  
}
```

```
/* Estilos adicionais para o cabeçalho */
```

```
header {  
    display: flex;  
    justify-content: space-between;  
    align-items: center;
```

```
background-color: #133579;
color: white;
padding: 30px;
height: 100px;
}

/* Estilos para a logo no cabeçalho */
.logo {
    font-size: 24px;
    font-weight: bold;
}

/* Estilos para o menu */
.menu {
    display: flex;
    gap: 20px;
}

/* Estilos para os links do menu */
.menu a {
    color: white;
    text-decoration: none;
    font-weight: bold;
    font-size: 20px;
    margin-top: 20px;
}

/* Estilos para os links do menu ao passar o mouse */
.menu a:hover {
    text-decoration: underline;
}

/* Estilos adicionais para o container principal */
.container {
```

```

padding: 60px;
border-radius: 10px;
}

/* Estilos para as seções */
.section {
margin-bottom: 40px;
}

/* Estilos para os títulos das seções */
.section h2 {
margin-bottom: 10px;
color: #333;
}

/* Estilos para os parágrafos das seções */
.section p {
color: #666;
}

/* Estilos adicionais para o rodapé */
footer {
background-color: #333;
color: #fff;
text-align: center;
padding: 20px 0;
margin-top: 300px;
bottom: 0;
width: 100%;
position: fixed;
}

/* Estilos para o fundo */
.fundo {

```

```

    background-image: url('jornal.png');
}

/* Estilos para as imagens */
.imagem {
    width: 80px;
}

/* Estilos para a seção principal */
.main {
    margin-bottom: 20%;
}

/* Estilos adicionais para o cabeçalho */
header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color: #133579;
    color: white;
    padding: 30px;
    height: 100px;
}

/* Estilos para o ícone do carrinho no cabeçalho */
.carrinho {
    width: 30px;
}
</style>
</head>
<body>
    <!-- Inclui o cabeçalho -->
    <%- include('partials/headerLog') %>
    <main class="main">

```



```

    <ul id="order-list"></ul>
</main>
<!-- Inclui o rodapé -->
<%- include('partials/footer') %>

<script>
    document.addEventListener('DOMContentLoaded', () => {
        // Obtém o email do usuário armazenado no localStorage
        const userEmail = localStorage.getItem('userEmail');
        if (!userEmail) {
            // Se o usuário não estiver logado, redireciona para a página de
início
            alert('Usuário não logado');
            window.location.href = '/inicio';
            return;
        }

        // Faz uma requisição para obter os dados do usuário
        fetch(`http://localhost:3001/users?email=${userEmail}`)
            .then(response => {
                if (!response.ok) {
                    throw new Error('Erro na resposta do servidor');
                }
                return response.json();
            })
            .then(user => {
                const orderList = document.getElementById('order-list');
                orderList.innerHTML = ''; // Limpa a lista de pedidos

                if (user.pedidos) {
                    // Itera sobre os pedidos do usuário e cria elementos para
cada pedido
                    user.pedidos.forEach(pedido => {
                        const listItem = document.createElement('li');

```

```

listItem.setAttribute('data-id', pedido.id);

listItem.addEventListener('click', () => {
    const confirmMessage = confirm('Deseja realizar o
pagamento ou excluir o pedido?');
    if (confirmMessage) {
        // Redireciona para a página de pagamento
        window.location.href = `/opPagamento/`;
    } else {
        // Faz uma requisição para remover o pedido

fetch(`http://localhost:3001/removePedido/${pedido.id}`, {
    method: 'DELETE'
})
    .then(response => {
        if (!response.ok) {
            throw new Error('Erro ao excluir pedido.');
```

```

orderHeader.innerHTML = `
  <span>Pedido #${pedido.id}</span>
  <span>Quantidade: ${pedido.quantidade}</span>
`;
orderDetails.innerHTML = `
  <p><strong>Nome:</strong> ${pedido.nome}</p>
  <p><strong>Descrição:</strong>
${pedido.descricao}</p>
  <p><strong>Local:</strong> ${pedido.local}</p>
  <p><strong>Valor:</strong> R${pedido.valor}</p>
  <p><strong>Data:</strong> ${pedido.data}</p>
  <p><strong>Tipo:</strong> ${pedido.tipo}</p>
`;

// Adiciona os elementos criados à lista
listItem.appendChild(orderHeader);
listItem.appendChild(orderDetails);
orderList.appendChild(listItem);
});
}
})
.catch(error => console.error('Erro ao buscar os pedidos:',
error));
});
</script>
</body>
</html>

```

Esse código fornece uma interface simples e intuitiva para que os usuários visualizem e gerenciem seus pedidos, com um design responsivo e funcionalidade de back-end para suportar as operações necessárias.

pix ejs

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"> <!-- Definição do conjunto de caracteres
-->
  <meta name="viewport" content="width=device-width,
initial-scale=1.0"> <!-- Meta tag para responsividade -->
  <link rel="stylesheet" href="/css/login.css"> <!-- Importação de
folha de estilo externa -->
  <title>Document</title> <!-- Título da página -->
</head>
<body>
  <%- include('partials/headerLog') %> <!-- Inclusão dinâmica do
cabeçalho -->

  <div class="container2"> <!-- Divisão para centralização de
conteúdo -->

  <div class="container"> <!-- Contêiner principal -->
    <h1>Pagamento via Pix</h1> <!-- Título principal -->
    <h3> Chave: 546.668.9898</h3> <!-- Subtítulo com chave Pix -->
    <form id="pixForm"> <!-- Formulário para entrada de dados -->
      <div class="form-group">
        <label for="pixKey">Chave Pix</label> <!-- Campo para
inserção da chave Pix -->
        <input type="text" id="pixKey" name="pixKey" required>
<!-- Campo de entrada -->
      </div>
      <div class="form-group">
        <label for="amount">Valor (R$)</label> <!-- Campo para
inserção do valor -->

```

```

        <input type="number" id="amount" name="amount"
required> <!-- Campo de entrada numérica -->
    </div>
    <button class="btnCode" type="button"
onclick="generateQRCode()">Gerar QR Code</button> <!-- Botão
para gerar QR Code -->
    </form>
    <div id="qrCodeContainer" class="qr-code-container"></div>
<!-- Contêiner para exibição do QR Code gerado -->
    <button id="nextButton" class="hidden" >Avançar</button>
<!-- Botão oculto para avançar -->

</div>
</div>
    <%- include('partials/footer') %> <!-- Inclusão dinâmica do rodapé
-->

<style>
/* Estilos CSS internos para a página */
</style>

<script>
// Script JavaScript para gerar QR Code e controlar a ação do botão de
avançar
</script>
</body>
</html>

```

Esta página HTML apresenta um formulário para pagamento via Pix, permitindo aos usuários inserir uma chave Pix e o valor a ser pago, gerando um QR Code dinamicamente.

vender.ejs

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8"> <!-- Definição do conjunto de caracteres
-->
  <meta name="viewport" content="width=device-width,
initial-scale=1.0"> <!-- Meta tag para responsividade -->
  <link rel="stylesheet" href="../css/stylePi.css"> <!-- Importação de
folha de estilo externa -->
  <title>Formulário</title> <!-- Título da página -->
</head>
<style>
/* Estilos CSS internos para a página */
</style>
<body>
  <%- include('partials/headerAdm') %> <!-- Inclusão dinâmica do
cabeçalho para administradores -->
  <div class="container"> <!-- Contêiner principal do formulário -->
    <h2 class="titulo">Cadastro</h2> <!-- Título do formulário -->
    <form id="eventoForm" action="#" method="POST"> <!--
Formulário de cadastro de evento -->
      <div class="form-group">
        <label for="nome">Nome do evento:</label> <!-- Campo para
inserção do nome do evento -->
        <input type="text" id="nome" name="nome" required> <!--
Campo de entrada de texto -->
      </div>
      <div class="form-group">
        <label for="descricao">Descrição do evento:</label> <!-- Campo
para inserção da descrição do evento -->
        <input type="text" id="descricao" name="descricao" required>
<!-- Campo de entrada de texto -->
      </div>

```

```

<div class="form-group">
  <label for="local">Local do evento:</label> <!-- Campo para
inserção do local do evento -->
  <input type="text" id="local" name="local" required> <!--
Campo de entrada de texto -->
</div>
<div class="form-group">
  <label for="valor">Valor:</label> <!-- Campo para inserção do
valor do evento -->
  <input type="number" id="valor" name="valor" step="0.01"
required> <!-- Campo de entrada numérica -->
</div>
<div class="form-group">
  <label for="data">Data:</label> <!-- Campo para inserção da data
do evento -->
  <input type="date" id="data" name="data" required> <!--
Campo de entrada de data -->
</div>
<div class="form-group">
  <label for="tipo">Tipo:</label> <!-- Campo para seleção do tipo
de evento -->
  <select id="tipo" name="tipo" required> <!-- Lista suspensa para
seleção -->
    <option value="">Selecione</option>
    <option value="Show">Show</option>
    <option value="Exposições">Exposições</option>
    <option value="Esportes">Esportes</option>
  </select>
</div>
<button type="submit">Cadastrar</button> <!-- Botão de envio
do formulário -->
</form>
</div>

```

```
<%- include('partials/footer') %> <!-- Inclusão dinâmica do rodapé
-->
</body>

<script>
// Script JavaScript para envio de dados do formulário via POST
</script>
</html>
```

Esta página apresenta um formulário para cadastro de eventos, permitindo aos administradores inserir nome, descrição, local, valor, data e tipo de evento, e enviar os dados para um servidor através de uma requisição POST.

`index.js`

```
// Importa o módulo Express para criação do servidor web
const express = require('express');

// Importa o módulo path para lidar com caminhos de arquivo e
diretório
const path = require('path');

// Importa o módulo fs para manipulação do sistema de arquivos
const fs = require('fs');

// Importa o módulo express-session para gerenciar sessões de
usuário
const session = require('express-session');

// Importa os roteadores personalizados para diferentes partes da
aplicação
const router = require('./src/router/router');
const routerU = require('./src/router/users');
```



```
const routerCombined = require('./src/router/combined');

// Cria uma instância do aplicativo Express
const app = express();

// Define a porta em que o servidor vai escutar
const porta = 3001;

// Define o caminho para o banco de dados de usuários
const DATABASE = path.join(__dirname, '/src/database/users.json');

// Define o caminho para o banco de dados de ingressos
const DATABASE2 = path.join(__dirname,
'/src/database/tickets.json');

// Middleware para tratar requisições JSON
app.use(express.json());

// Middleware para tratar dados de formulário URL-encoded
app.use(express.urlencoded({ extended: true }));

// Configuração do uso de sessão
app.use(session({
  secret: 'seuSegredoAqui', // Chave secreta para assinar o cookie da
  sessão
  resave: false, // Evita que a sessão seja salva novamente se não houver
  alterações
  saveUninitialized: true, // Força uma sessão a ser salva mesmo que
  não esteja inicializada
  cookie: { secure: false } // Configurações do cookie da sessão
}));

// Middleware para servir arquivos estáticos do diretório 'public'
app.use(express.static("public"));
```

```

// Define as rotas principais usando os roteadores importados
app.use("/", router);
app.use("/", routerU);
app.use("/", routerCombined);

// Configura o mecanismo de visualização como EJS e o diretório onde
os arquivos de visualização estão localizados
app.set('view engine', 'ejs');
app.set('views', 'src/view/');

// Função para inicializar o banco de dados de usuários se não existir
function initDb() {
  if (!fs.existsSync(DATABASE)) {
    fs.writeFileSync(DATABASE, JSON.stringify({ users: [] }, null, 8));
  }
}

// Função para inicializar o banco de dados de ingressos se não existir
function initDb2() {
  if (!fs.existsSync(DATABASE2)) {
    fs.writeFileSync(DATABASE2, JSON.stringify({ tickets: [] }, null, 5));
  }
}

// Inicia o servidor na porta especificada
app.listen(porta, () => {
  // Chama as funções de inicialização dos bancos de dados
  initDb();
  initDb2();
  // Exibe mensagem no console indicando que o serviço está sendo
  executado na porta especificada
  console.log(`Serviço executado na porta ${porta}`);
  http://localhost:${porta}/log`);

```

```
});
```

Este código configura um servidor web usando Express em Node.js. Ele inclui gerenciamento de sessões, roteamento usando módulos externos (router, routerU, routerCombined), inicialização de bancos de dados JSON para usuários e ingressos, e configuração de um mecanismo de visualização usando EJS. O servidor escuta na porta 3001 e utiliza arquivos estáticos do diretório 'public' para servir recursos estáticos como CSS e imagens

4.2. AMBIENTE DE DESENVOLVIMENTO

Frontend:

Editor de Código: Visual Studio Code (VsCode) - Utilizado para escrever e editar o código fonte do frontend.

Linguagens e Tecnologias:

HTML: Para a estruturação das páginas web.

CSS: Para a estilização e layout das páginas.

JavaScript: Para a interação e dinamismo das páginas.

Backend:

Editor de Código: Visual Studio Code (VsCode) - Utilizado para escrever e editar o código fonte do backend.

Linguagens e Tecnologias:

JavaScript: Linguagem principal utilizada para o desenvolvimento do backend.

Node.js: Ambiente de execução para JavaScript no lado do servidor.

Nodemon: Ferramenta que ajuda no desenvolvimento ao reiniciar

automaticamente o servidor sempre que alterações são feitas.

Express: Framework para Node.js utilizado para construir a API do servidor.

Biblioteca HTTP: Utilizada para a criação de servidores HTTP.

EJS: Engine de templates para geração de HTML a partir de dados dinâmicos.

Prototipação:

Design:Canvas

Organização:

Trello e Scrum.

4.4. TESTES:

Testes Funcionais

Cadastro e Login de Usuários:

Verificar se um usuário pode se cadastrar com e-mail e senha.

Confirmar que o usuário pode fazer login com as credenciais corretas.

Garantir que o sistema bloqueia tentativas de login com credenciais incorretas.

Navegação e Visualização de Eventos:

Confirmar que todos os eventos são exibidos na página principal.

Testar se a navegação para a página de detalhes de um evento funciona corretamente.

Compra de Ingressos:

Verificar a seleção de tipos de ingresso e a adição ao carrinho.

Confirmar o fluxo de compra até a finalização, incluindo a aceitação dos termos e a escolha do método de pagamento.

Histórico de Compras:

Testar se o usuário pode visualizar o histórico de compras em “meus pedidos”.

Cadastro e Gestão de Ingressos por Vendedores:

Verificar se vendedores podem cadastrar novos ingressos com todas as informações necessárias.

Confirmar a edição e exclusão de ingressos cadastrados.

Feedback:

Navegação e Visualização de Eventos:

Positivo: Todos os eventos são exibidos corretamente na página principal, e As informações são fáceis de encontrar.

Negativo: A navegação para a página de detalhes do evento às vezes é lenta. Pode ser necessário otimizar o carregamento da página.

Positivo: A interface de navegação é intuitiva, facilitando a busca por eventos específicos.

Compra de Ingressos:

Positivo: A seleção de tipos de ingresso é clara e permite fácil adição ao carrinho.

Positivo: Alguns usuários não relataram dificuldade em encontrar a tela de termos e condições.

Positivo: O fluxo de compra é eficiente e inclui todas as etapas necessárias, desde a seleção até o pagamento.

Histórico de Compras:

Positivo: A visualização do histórico de compras é clara e bem organizada, permitindo que os usuários encontrem facilmente seus ingressos comprados.

Cadastro e Gestão de Ingressos por Vendedores

Positivo: A interface para cadastro de novos ingressos é abrangente, permitindo a inserção de todas as informações necessárias, como descrição, imagens e preços.

Positivo: vendedores relataram Facilidade na edição de ingressos já cadastrados.

Positivo: A exclusão de ingressos é rápida e eficiente, garantindo que eventos cancelados sejam removidos prontamente do sistema.

4.5 MANUAL DO USUÁRIO:

O manual do usuário está disponível em:

https://www.canva.com/design/DAGHMnSYHXs/MVR4c7p5LLaqJFOcZL3Wxw/edit?utm_content=DAGHMnSYHXs&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

5. CONCLUSÕES E CONSIDERAÇÕES FINAIS:

CONSIDERAÇÕES FINAIS:

O projeto Innovators Ticket foi desenvolvido com o objetivo de criar uma plataforma de venda de ingressos online que seja intuitiva, segura e eficiente.

Ao longo do desenvolvimento, utilizamos tecnologias modernas e metodologias ágeis para garantir a qualidade e a funcionalidade do sistema.

Benefícios e Impacto:

Para Usuários: Facilitamos o processo de compra de ingressos, proporcionando uma Experiência de usuário simplificada e segura. Com funcionalidades como histórico de compras e gerenciamento de perfil, os usuários podem ter maior controle sobre suas transações e eventos.

Para Organizadores de Eventos:

Oferecemos ferramentas eficazes para o cadastro e gestão de ingressos, permitindo que os organizadores monitorem vendas e gerencie seus eventos de maneira prática e eficiente.

Tecnologias Utilizadas:

No frontend, utilizamos HTML, CSS e JavaScript para criar uma interface amigável e responsiva.

No backend, empregamos Node.js, Express e EJS, proporcionando uma infraestrutura robusta e escalável.

Metodologia de Trabalho:

Adotamos a metodologia ágil Scrum, que nos permitiu trabalhar de maneira iterativa e

incremental, respondendo rapidamente a mudanças e assegurando a entrega contínua de valor ao longo do projeto.

Desafios e Soluções:

Durante o desenvolvimento, enfrentamos desafios relacionados à integração de sistemas e a otimização da segurança. Através de pesquisa e aplicação de melhores práticas, conseguimos superar esses obstáculos, resultando em uma plataforma confiável e eficiente.

Futuro do Innovators Ticket:

Continuaremos a aprimorar a plataforma com base no feedback dos usuários e nas tendências de mercado. Planejamos integrar novas funcionalidades, como notificações em tempo real e parcerias com mais organizadores de eventos, para expandir o alcance e melhorar a experiência dos nossos usuários.

Conclusão:

O Innovators Ticket se posiciona como uma solução eficaz para a venda e gestão de ingressos online, atendendo tanto às necessidades dos usuários finais quanto dos organizadores de eventos. Agradecemos a todos os envolvidos no desenvolvimento deste projeto e estamos ansiosos para ver o impacto positivo que ele terá no mercado de eventos.

6.BIBLIOGRAFIA:

- <https://guia.dev/pt/pillars/software-architecture/technical-documentation.html>
- http://www.waltenomartins.com.br/esof2_projeto_documento_de_sw.pdf

-<https://logap.com.br/blog/o-que-e-prototipacao-software/#:~:text=A%20prototipa%C3%A7%C3%A3o%20de%20software%20%C3%A9,desenvolvedor%2C%20infelizmente%2C%20vai%20passar.>

- <https://ebaonline.com.br/blog/o-que-e-figma-e-como-usar>