INSTITUTO FEDERAL DE SÃO PAULO
DESENVOLVIMENTO DE SISTEMAS

FERNANDO PEREIRA FILIPE DUARTE
JOÃO PEDRO DE ALMEIDA MARTINS

PROJETO HOTELARIA

DESENVOLVIMENTO WEB

SÃO PAULO
2024

Arquivos HTML

Os arquivos HTML formam o front-end da aplicação, ou seja, a interface visual que o usuário vê e utiliza para interagir com o sistema.

inicio.html:

O arquivo **inicio.html** é o ponto de entrada da aplicação, funcionando como uma página inicial que oferece acesso às principais funcionalidades. Nele, o usuário encontra links para áreas essenciais, como Cadastro, Login, Reservas, Frigobar, Serviços, Pagamentos, Registro e Checkout, facilitando a navegação e o uso do sistema.

Código:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Início</title>
</head>
<body>
    <h1>Bem-vindo ao Sistema de Gestão do Hotel</h1>
    <a href="/cadastro">Cadastro</a>
    <a href="/login">Login</a>
    <!-- Mais links para as outras funcionalidades -->
</body>
</html>
```

Esse arquivo HTML apresenta uma lista de links que facilita a navegação entre as páginas do sistema. Por exemplo, ao utilizar `href="/cadastro"`, o link redireciona o usuário para a rota `/cadastro` no servidor Flask, permitindo que ele acesse facilmente a página de cadastro ao clicar. Essa estrutura ajuda a criar uma navegação fluida e intuitiva.

---

cadastro.html:

Esta página contém um formulário para que o usuário preencha seus dados de cadastro. Ao submeter o formulário, as informações são enviadas ao backend via método `POST`, onde serão processadas e armazenadas.

Código:

```
<form action="/cadastro" method="post">
    <label>Nome:</label>
```

```
    <input type="text" name="nome" required>
    <label>CPF:</label>
    <input type="text" name="cpf" required>
    <label>Email:</label>
    <input type="email" name="email" required>
    <label>Senha:</label>
    <input type="password" name="senha" required>
    <button type="submit">Cadastrar</button>
</form>
```

O formulário utiliza o método `POST` para enviar os dados com segurança ao backend. Os campos de entrada (`input`) capturam informações essenciais do usuário, como nome, CPF, e-mail e senha, garantindo que os dados necessários para o cadastro sejam coletados e transmitidos de maneira protegida.

---

login.html:

Nesta página, o usuário pode inserir seus dados para acessar o sistema.

Código:

```
<form action="/login" method="post">
    <label>Nome:</label>
    <input type="text" name="nome" required>
    <label>Senha:</label>
    <input type="password" name="senha" required>
    <button type="submit">Entrar</button>
</form>
```

Assim como na página de cadastro, este formulário também utiliza o método `POST` para enviar os dados ao backend. O sistema verifica as credenciais fornecidas e, com base nessa validação, permite ou nega o acesso ao sistema.

---

reservas.html:

Nesta página, o usuário tem a oportunidade de fazer a reserva de um quarto. Ele pode selecionar o tipo de quarto, definir as datas de check-in e check-out, além de escolher a forma de pagamento. Essa interface foi projetada para permitir que o usuário personalize sua estadia conforme suas preferências e necessidades.

Código:

```
<form action="/reservar" method="post">
    <label>Nome:</label>
    <input type="text" name="nome" required>
    <label>Quarto:</label>
    <input type="number" name="quarto" required>
    <label>Check-in:</label>
    <input type="date" name="checkin" required>
    <label>Check-out:</label>
    <input type="date" name="checkout" required>
    <button type="submit">Reservar</button>
</form>
```

A estrutura deste formulário é semelhante à do formulário de cadastro, mas inclui campos específicos para coletar informações de reserva. O método `POST` é utilizado para enviar esses dados ao backend, onde eles serão processados adequadamente.

---

frigobar.html:

Esta página permite que o usuário registre o consumo de itens do frigobar, como bebidas e snacks. Cada item é registrado com uma quantidade específica.

Código:

```
<form action="/frigobar" method="post">
    <label>Cerveja:</label>
    <input type="number" name="cerveja" min="0" value="0">
    <label>Suco:</label>
    <input type="number" name="suco" min="0" value="0">
    <!-- Mais campos para outros itens -->
    <button type="submit">Enviar</button>
</form>
```

O formulário envia a quantidade de cada item consumido ao backend, que, por sua vez, calcula o custo total com base nas informações fornecidas. Isso permite que o usuário tenha um resumo claro dos gastos antes de finalizar o pagamento.

---

pagamentos.html:

A página de pagamentos apresenta o valor total a ser pago e oferece ao usuário a opção de selecionar a forma de pagamento que prefere. Essa interface torna o processo de finalização mais claro e fácil para o usuário.

Código:

```
<h2>Total a pagar: R$<span id="total"></span></h2>
<form action="/pagar" method="post">
    <label>Forma de Pagamento:</label>
    <select name="forma_pagamento">
        <option value="cartao">Cartão</option>
        <option value="dinheiro">Dinheiro</option>
    </select>
    <button type="submit">Pagar</button>
</form>
```

Arquivos JavaScript

Os arquivos JavaScript são utilizados principalmente para realizar validações no front-end e executar cálculos, permitindo que essas operações sejam feitas sem a necessidade de enviar dados para o servidor. Isso melhora a eficiência e a experiência do usuário ao interagir com a aplicação.

cadastro.js:

Esse script é responsável por enviar os dados de cadastro de um usuário. Ao enviar o formulário, a função `enviar()` coleta as informações do usuário, verifica se as senhas coincidem e, em seguida, envia os dados para o servidor. Se o cadastro for bem-sucedido, o usuário é redirecionado para a próxima página; caso contrário, uma mensagem de erro é exibida, informando sobre o problema.

Código:

```
const nome = document.getElementById("nome").value.trim();
if (senha !== confirmarSenha) {
    mensagem.textContent = "As senhas não coincidem.";
    return;
}
const response = await fetch('/cadastro', {
    method: 'POST',
    headers: {
```

```
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({ nome, endereco, cpf, telefone, email, senha }),
});
```

---

checkout.js:

Esse script é responsável por criar botões dinâmicos na interface web, permitindo que o usuário execute ações de reserva. A função `criarBotoes()` limpa um contêiner específico e gera um botão para cada item na lista. Quando um botão é clicado, uma função é acionada, que registra no console qual botão foi ativado, proporcionando um feedback instantâneo sobre a interação do usuário.

Código:

```
const botoesContainer = document.getElementById('botoes');
botoesContainer.innerHTML = "";

for (const item of lista) {
    const botao = document.createElement('button');
    botao.textContent = `Reserva ${item}`;
    botao.onclick = () => executarFuncao(item);
}
```

---

frigobar.js:

Esse script é responsável por calcular o total de consumo dos itens de um frigobar. Quando a função `enviarConsumo()` é chamada, ela coleta os valores dos itens, como cerveja, suco, água, lanche natural e barrinha de cereal. O total é calculado multiplicando a quantidade de cada item pelo seu respectivo preço, e o resultado é exibido em um elemento HTML específico, proporcionando ao usuário uma visão clara dos gastos.

Código:

```
const cerveja = parseInt(document.getElementById('cerveja').value);
const total = (cerveja*9) + (suco*8) + (agua*5) + (lancheNatural*10) + (barrinhaCereal*3.50);
document.getElementById("total").innerHTML = total;
```

---

reservas.js:

Esse script é responsável por adicionar uma reserva ao sistema. Quando o formulário é enviado, a função `adicionarReserva()` evita o recarregamento da página e coleta os dados do usuário, incluindo nome, número do quarto e datas de entrada e saída. Os dados são então enviados para o servidor usando o método `POST`. Após o envio, uma mensagem de sucesso ou erro é exibida em um alerta. Se a reserva for criada com sucesso, o usuário é redirecionado para a página de registros.

Código:

```
event.preventDefault();
const nomeUsuario = document.getElementById("nome").value;
const numeroQuarto = document.getElementById("quarto").value;
const response = await fetch('/reservar', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
     body: JSON.stringify({ nome: nomeUsuario, quarto: numeroQuarto, pagamento, data1, data2 }),
});
const result = await response.json();
```

---

servicos.js:

Esse script é encarregado de calcular o custo dos serviços selecionados pelo usuário. A função `enviarServicos()` verifica quais serviços foram escolhidos, como café da manhã, serviço de quarto, lavanderia, almoço e jantar, e soma os respectivos custos. O total é então exibido em um elemento HTML específico, permitindo que o usuário visualize facilmente o custo total dos serviços selecionados.

Código:

```
const cafedamanhaSim = document.getElementById('cafe-da-manha-sim');
let custoServico = 0;
if (servicoDeQuartoSim.checked) {
    custoServico = 150;
}
if (cafedamanhaSim.checked) {
    custoServico += 100;
```

```
}
document.getElementById("total").innerHTML = custoServico;
```

Arquivo CSS

Esse código define o estilo visual da página, empregando uma paleta de cores sóbria e uma tipografia moderna.

A utilização de `box-sizing: border-box` garante que o padding e a borda sejam incluídos nas dimensões totais dos elementos, o que torna o layout mais consistente e previsível. Essa abordagem facilita o design responsivo e melhora a experiência do usuário.

Código:

```
{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: Arial, sans-serif;
}
```

Estilo do Corpo: O fundo da página é definido como um tom claro (#f0f4f8), enquanto a cor do texto é escura (#333). Essa combinação foi escolhida para garantir uma boa legibilidade, proporcionando um contraste agradável e facilitando a leitura do conteúdo.

Código:

```
body {
  background-color: #f0f4f8;
  color: #333;
  line-height: 1.6;
}
```

Header e Navegação: O cabeçalho apresenta um fundo escuro, estilizado com texto claro que se destaca, e possui uma sombra que adiciona profundidade ao visual. Os botões de navegação são organizados em linha, centralizados e oferecem transições suaves de cor quando o mouse passa sobre eles,

proporcionando uma experiência de navegação mais interativa e agradável.

Código:

```
header {
  background-color: #2c2f48;
  color: #ffffff;
  padding: 1.5rem 0;
  text-align: center;
}
nav ul {
  display: flex;
  justify-content: center;
  gap: 2rem;
}
```

Estilo do Main e Seções: O elemento **main** possui padding para garantir um espaçamento interno adequado e é limitado em largura para facilitar a legibilidade do conteúdo. Cada **section** tem um fundo branco, bordas arredondadas e uma sombra suave, criando um efeito visual agradável e destacando as diferentes áreas de conteúdo na página.

Código:

```
main {
  padding: 3rem 1rem;
  max-width: 900px;
  margin: auto;
}
section {
  background-color: #fff;
  padding: 2rem;
  margin-bottom: 2rem;
  border-radius: 10px;
  box-shadow: 0 8px 12px rgba(0, 0, 0, 0.1);
}
```

Formulários e Botões: Os formulários são dispostos verticalmente, apresentando inputs que possuem bordas e transições suaves quando recebem foco. Os botões são estilizados para serem atraentes e responsivos, mudando de cor e elevando-se levemente ao passar o mouse, o que torna a interação mais

dinâmica e agradável para o usuário.

Código:

```
.form-group {
  display: flex;
  flex-direction: column;
  gap: 0.5rem;
}
button {
  background-color: #1389e9;
  color: #fff;
  padding: 0.8rem 1.5rem;
  border-radius: 6px;
}
```

Listas de Reservas: Os itens de reserva são estilizados com um fundo claro e uma borda à esquerda, o que proporciona uma visualização clara e organizada. Essa abordagem ajuda a destacar cada item, facilitando a leitura e a identificação das reservas pelo usuário.

Código:

```
.reservas-list {
  list-style: none;
  padding: 0;
  text-align: center;
}
.reserva-item {
  padding: 1.5rem;
  margin: 1rem 0;
  background-color: #f1f5f9;
  border-left: 6px solid #2c2f48;
}
```

Mensagens: As mensagens de erro ou confirmação são estilizadas para se destacarem, garantindo que sejam facilmente percebidas pelo usuário. Essa ênfase visual ajuda a comunicar rapidamente informações importantes, como a necessidade de correção ou a confirmação de uma ação bem-sucedida.

```
#mensagem-hospede,
#mensagem-reserva {
  color: #2c2f48;
  font-weight: bold;
}
```

Python / Flask:

Flask é um microframework para Python que se destaca por sua leveza e facilidade de uso, sendo ideal para aplicações pequenas e médias. Ele permite a criação rápida de uma API para gerenciar solicitações HTTP, como aquelas feitas pelos formulários nas páginas HTML. Além disso, Flask é altamente flexível e modular, o que possibilita a adição de componentes e funcionalidades conforme a necessidade do projeto.

app.py:

Este script cria um aplicativo web utilizando o Flask. Ele permite que os usuários se cadastrem, façam login, realizem reservas e visualizem os serviços disponíveis. Essa estrutura facilita a interação dos usuários com o sistema, proporcionando uma experiência completa e integrada.

Importações e Configurações: O Flask e outras funcionalidades necessárias são importados para o projeto. O aplicativo é configurado para utilizar uma pasta específica destinada a templates, permitindo a organização das páginas HTML. Além disso, uma chave secreta é definida para gerenciar as sessões de usuário, garantindo a segurança das informações durante a interação com o aplicativo.

Código:

```
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
app = Flask(__name__, template_folder='Arquivos')
app.secret_key = 'chave ultra secreta'
```

Classes de Modelo: Duas classes são definidas no aplicativo: **Usuario**, que contém informações sobre o usuário, como seus dados pessoais e reservas associadas, e **Reserva**, que armazena detalhes sobre as reservas realizadas, incluindo informações como tipo de quarto, datas de entrada e saída, e status da

reserva. Essa estrutura orientada a objetos facilita a gestão e o acesso aos dados relacionados aos usuários e suas reservas.

Código:

```
class Usuario:
    def __init__(self, nome, endereco, cpf, telefone, email, senha):
        # Inicializa os atributos do usuário
        self.nome = nome
```

Funções para Gerenciamento de Usuários: A função `adicionar_usuario` verifica se um usuário já existe no sistema; se não existir, ela cria um novo usuário. Por outro lado, a função `adicionar_reserva_usuario` é responsável por associar uma nova reserva a um usuário específico, garantindo que cada reserva fique vinculada ao seu respectivo usuário no sistema. Essa lógica facilita a organização e o gerenciamento de usuários e suas reservas.

Código:

```
def adicionar_usuario(nome, endereco, cpf, telefone, email, senha):
    if any(usuario.nome == nome for usuario in usuarios):
        return False  # Nome de usuário já existe
```

Rotas do Flask: O aplicativo possui várias rotas que correspondem a diferentes páginas da web, cada uma destinada a uma funcionalidade específica. Essas rotas permitem que os usuários naveguem entre as páginas, como:

- `/` - página inicial.
- `/cadastrar` - formulário de cadastro.
- `/cadastro` - trata o envio de dados do formulário de cadastro, armazenando um novo usuário e redirecionando para a página de login.

Código:

```
@app.route('/cadastro', methods=['POST'])
def cadastro():
        data = request.get_json()
        …
        return jsonify({"redirect": "/login"}), 200
```

Login e Reservas: A rota `/login` é responsável por verificar se as credenciais do usuário estão corretas e, se forem válidas, inicia uma sessão para o usuário. A rota `/reservar` renderiza a página onde os usuários podem fazer suas reservas, enquanto a rota `/reservas` processa a criação de uma nova reserva com base nas informações fornecidas pelo usuário. Essa estrutura de rotas permite um fluxo de navegação claro e eficiente dentro do aplicativo.

Código:

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

Outras Rotas: As rotas `/servicos`, `/pagamentos`, `/frigobar`, `/registros` e **/checkout** são responsáveis por renderizar diferentes páginas que permitem gerenciar suas respectivas funcionalidades.

Código:

```
@app.route('/servicos')
def servicos():
    return render_template('servicos.html')
```

Logout: A rota `/sair` deve ser implementada para permitir que o usuário encerre sua sessão. Embora a funcionalidade ainda esteja incompleta no código atual, a implementação dessa rota é essencial para garantir que os usuários possam sair do sistema de forma segura.

Código:

```
@app.route('/sair')
def sair():
    return
```

Execução do Servidor: Por fim, o aplicativo é executado em modo de depuração.

Código:

```
if __name__ == '__main__':
    app.run(debug=True)
```