**Relatório A3 Sistemas Distribuídos** - **Aplicativo Cliente-Servidor**

**Professor: Eduardo Sidney da Silva Xavier**

**Turma: Sexta-Feira / Matutino**

**Integrantes:**

* **Adrian Dantas Silva - 12724141716**
* Alan Jorge Vitorio Santos - 1272325529
* Matheus Mascarenhas Alem - 12724131902
* João Caetano Barreto Lima - 1272220378
* Erik Leandro Santos da Costa - 12724235594
* Vinicius dos Santos Santana - 12724121934

# **Regras de Negócio e Funcionalidades do Aplicativo**

A aplicação de gerenciamento de reservas de restaurante é estruturada para atender a diferentes perfis de usuários, cada um com funcionalidades e permissões específicas. O restaurante opera das **12:00h (meio-dia) à 00:00h (meia-noite)**. Contudo, a última reserva possível deve ser feita para **22:00h**, garantindo tempo hábil para o atendimento e encerramento das atividades. As reservas são realizadas em intervalos de 2 em 2 horas, e as mesas disponíveis vão de 1 a 20, com capacidade para 1 a 10 pessoas.

**1. Perfil Atendente**

O atendente é o responsável por gerenciar as reservas de clientes.

* **Ações Permitidas:**
  + **Criar Nova Reserva:** O atendente pode registrar novas reservas informando o nome do responsável, data, hora, número da mesa e quantidade de pessoas.
  + **Cancelar Reserva:** O atendente pode cancelar reservas que ainda estejam com o status "reservada".
* **Regras e Validações na Criação de Reservas:**
  + **Horário de Funcionamento:** As reservas só podem ser feitas entre 12:00h e 22:00h, em intervalos de 2 em 2 horas.
  + **Mesas e Capacidade:** O número da mesa deve estar entre 1 e 20. A quantidade de pessoas por reserva deve ser entre 1 e 10.
  + **Data Passada:** Não é permitido criar uma reserva para uma data que já passou. Se o atendente tentar criar uma reserva para uma data anterior ao dia atual, um erro será exibido.
  + **Hora Passada (no dia atual):** Se a reserva for para o dia atual, não é permitido criar para um horário que já passou. Por exemplo, se agora são 15:00h, não será possível reservar para 12:00h ou 14:00h do mesmo dia.
  + **Mesa Já Reservada:** Não é possível criar uma reserva para uma mesa que já está "reservada" ou "confirmada" na mesma data e hora. Um alerta será exibido caso haja conflito.
* **Mensagens de Feedback:**
  + **Sucesso:** "Reserva criada com sucesso!"
  + **Erro:** "Erro: Mesa já reservada para essa data e hora." ou "Erro: Não é possível criar uma reserva em uma data passada." ou "Erro: Não é possível criar uma reserva para um horário que já passou hoje."
  + **Alerta:** "Reserva cancelada com sucesso!"

**2. Perfil Garçom**

O garçom é responsável por confirmar as reservas no dia.

* **Ações Permitidas:**
  + **Confirmar Reserva:** O garçom pode confirmar uma reserva que esteja com o status "reservada". Ao confirmar, ele deve informar seu nome, e o status da reserva será alterado para "confirmada".
* **Regras e Validações na Confirmação de Reservas:**
  + **Reservas do Dia:** O garçom só pode confirmar reservas para o dia atual. Se tentar confirmar uma reserva para uma data futura, um alerta será exibido.
  + **Status da Reserva:** Apenas reservas com o status "reservada" podem ser confirmadas. Se a reserva já estiver "confirmada" ou em outro status, uma mensagem de erro será mostrada.
* **Mensagens de Feedback:**
  + **Sucesso:** "Reserva confirmada com sucesso!"
  + **Alerta:** "Alerta: Só é possível confirmar reservas para o dia de hoje."
  + **Erro:** "Erro: A reserva não está mais disponível para confirmação." ou "Erro: Reserva não encontrada."

**3. Perfil Gerente**

O gerente tem acesso a relatórios detalhados sobre as reservas do restaurante.

* **Ações Permitidas:**
  + **Gerar Relatórios:** O gerente pode gerar três tipos de relatórios:
    - **Relação de Reservas (por período):** Permite visualizar todas as reservas dentro de um intervalo de datas específico.
    - **Reservas por Mesa:** Exibe todas as reservas realizadas para um número de mesa específico.
    - **Relação de Mesas Confirmadas por Garçom:** Mostra quais garçons confirmaram quais reservas, agrupando as reservas confirmadas por garçom.
* **Regras e Validações na Geração de Relatórios:**
  + **Período:** Para o relatório por período, é obrigatório informar a data inicial e a data final.
  + **Número da Mesa:** Para o relatório por mesa, é obrigatório informar o número da mesa.
  + **Tipo de Relatório:** É necessário selecionar um tipo de relatório válido para gerar a consulta.
* **Mensagens de Feedback:**
  + **Erro:** "Informe a data inicial e final.", "Informe o número da mesa." ou "Tipo de relatório inválido."
  + **Informação:** "Nenhuma reserva encontrada para os critérios selecionados." ou "Nenhuma mesa confirmada encontrada." (quando não há dados para o relatório).

# **Requisitos de Software**

**1. Visão Geral**

Este documento detalha as tecnologias, linguagens e bibliotecas necessárias para a compilação, execução e manutenção da aplicação de gerenciamento de reservas de restaurante. A aplicação é composta por um back-end que lida com a lógica de negócio e um front-end que apresenta a interface ao usuário.

**2. Requisitos do Ambiente de Back-End (Servidor)**

O back-end é responsável por processar as requisições, gerenciar a lógica de negócio e persistir os dados no banco de dados.

* **Linguagem de Programação:**
  + **Python (versão 3.8 ou superior):** Toda a lógica do servidor é escrita em Python. É fundamental ter um interpretador Python instalado no ambiente de execução.
* **Framework Web:**
  + **Flask:** É o micro-framework utilizado como base para a aplicação. Ele gerencia o roteamento de URLs, o processamento de requisições HTTP e a renderização das páginas.
* **Bibliotecas Python (Dependências):**
  + **Flask-SQLAlchemy:** Extensão que integra o SQLAlchemy ao Flask, simplificando as operações de banco de dados.
  + **SQLAlchemy:** É um ORM (Mapeador Objeto-Relacional) que traduz o modelo de dados da classe Reserva em uma tabela no banco de dados, permitindo a manipulação dos dados através de objetos Python.
* **Banco de Dados:**
  + **SQLite:** A aplicação está configurada para usar SQLite, um sistema de gerenciamento de banco de dados que armazena os dados em um único arquivo (database.db). Não requer um processo de servidor separado, sendo ideal para desenvolvimento e aplicações de pequeno a médio porte.

**3. Requerimentos do Ambiente de Front-end (Cliente)**

O front-end é a camada de apresentação da aplicação, com a qual o usuário interage diretamente através do navegador.

* **Linguagens Web Essenciais:**
  + **HTML5:** Utilizada para a estruturação semântica do conteúdo de todas as páginas (index.html, atendente.html, etc.).
  + **CSS3:** Utilizada para a estilização e o layout visual da aplicação, conforme definido no arquivo static/css/style.css.
* **Linguagem de Script:**
  + **JavaScript (Vanilla JS):** O arquivo static/js/script.js contém o código JavaScript puro, sem dependência de frameworks externos. Ele é responsável por funcionalidades dinâmicas, como a exibição de notificações (pop-ups) e a manipulação da visibilidade de campos no formulário de relatórios do gerente.
* **Motor de Templates:**
  + **Jinja2:** O Flask utiliza o Jinja2 para renderizar as páginas HTML. Esta tecnologia permite a inserção de dados do backend diretamente no HTML e o uso de estruturas de controle (como {% for %} e {% if %}), tornando as páginas dinâmicas.

# **Instruções de Instalação e Execução**

### ****1. Instale o Python****

Inicialmente, você precisa ter o Python instalado no seu computador. Ele é a linguagem de programação que vamos usar.

* **Como instalar:** Baixe o instalador direto do site oficial do Python: [python.org](https://www.python.org/downloads/).

### ****2. Prepare o Visual Studio****

O Visual Studio é o editor que usaremos para trabalhar com o código. Ele precisa de uma extensão para entender e executar o Python.

* **Instale a extensão Python:** Dentro do visual Studio, vá até o menu de extensões, procure por "Python" e instale a extensão oficial da Microsoft. A extensão "Code Runner" também pode auxiliar na execução do código.

### ****3. Configure o Projeto****

Agora que as ferramentas estão prontas, vamos arrumar o projeto.

* **Abra a pasta:** No visual studio, clique em "Arquivo" > "Abrir Pasta" e selecione a pasta do projeto.
* **Instale as dependências:** As dependências são pacotes de código que o nosso programa precisa para funcionar.
  + Abra o terminal integrado do visual studio. Para isso, clique em "Terminal" > "Novo Terminal" na barra de menu superior.
  + No terminal que apareceu na parte de baixo da tela, digite o comando abaixo e pressione “Enter”:

- pip install -r requirements.txt

### ****4. Rode o Programa****

Com tudo instalado e configurado, é hora de executar!

**4.1 Via Visual Studio**

* **Encontre o arquivo principal:** Na lista de arquivos à esquerda, procure e abra o arquivo app.py.
* **Execute:** Com o arquivo app.py aberto, clique no botão de "play" (normalmente um triângulo ▶️) que fica no canto superior direito do visual studio e selecione a opção "Run Python File in Terminal".
* **Abra a URL:** Uma URL aparecerá no terminal. Para abri-la no navegador, basta clicar nela enquanto segura a tecla Ctrl.

**4.2 Via CMD (Prompt de Comando)**

* **Abra o prompt de comando** e navegue até o diretório onde o arquivo app.py está localizado.
* **Execute o comando:** python app.py
* **Abra a URL:** Uma URL será exibida no terminal. Clique nela enquanto segura a tecla Ctrl para abrir a aplicação no seu navegador.

# **Justificativa para a Abordagem de Comunicação**

A arquitetura de comunicação da aplicação foi projetada com base em um modelo clássico e robusto de cliente-servidor, priorizando simplicidade, confiabilidade e uma boa experiência para o usuário. A seguir, detalhamos as escolhas e os motivos para a sua adoção.

#### ****1. Modelo de Comunicação: Requisição-Resposta com Renderização no Servidor (SSR)****

A interação fundamental entre o navegador do usuário (cliente) e a aplicação (servidor) segue o ciclo de **Requisição-Resposta** do protocolo HTTP, com a lógica de apresentação sendo majoritariamente controlada pelo servidor.

* **Como Funciona:**
  1. O usuário realiza uma ação na interface (ex: clica em um link ou envia um formulário).
  2. O navegador envia uma requisição HTTP para o servidor Flask.
  3. O servidor processa a requisição: executa a lógica de negócio, interage com o banco de dados (SQLite via SQLAlchemy) e prepara os dados a serem exibidos.
  4. O servidor utiliza o motor de templates Jinja2 para gerar uma página HTML completa, já com os dados inseridos.
  5. Esta página HTML é enviada como resposta ao navegador, que a renderiza para o usuário.
* **Justificativa:** Esta abordagem de **Server-Side Rendering (SSR)** foi escolhida por sua **simplicidade e robustez**. A lógica de negócio e a de apresentação estão centralizadas no back-end, o que facilita o desenvolvimento, a depuração e a manutenção. Para uma aplicação de gerenciamento como esta, o SSR evita a complexidade desnecessária de construir e manter uma API separada e um framework de front-end pesado, tornando o projeto mais enxuto e direto.

#### ****2. Comunicação de Feedback ao Usuário: O Padrão Post-Redirect-Get (PRG) Aprimorado****

Um dos desafios em aplicações web é fornecer um feedback claro ao usuário após uma ação (como criar ou cancelar uma reserva) sem causar problemas, como o reenvio de um formulário ao atualizar a página. Para isso, adotamos o padrão **Post-Redirect-Get (PRG)** com uma melhoria para a exibição de mensagens.

* **Como Funciona:**
  1. **POST:** O usuário envia um formulário (ex: criar reserva). O servidor recebe a requisição POST, processa os dados e salva no banco.
  2. **REDIRECT:** Em vez de retornar uma página HTML diretamente, o servidor responde com um **redirecionamento** para uma nova URL (geralmente a página anterior). Crucialmente, ele adiciona a mensagem de feedback (ex: mensagem='Reserva criada com sucesso!') como um **parâmetro na URL**.
  3. **GET:** O navegador recebe a instrução de redirecionamento e faz uma nova requisição GET para a URL indicada.
  4. **Feedback Visual:**
     + O servidor, ao receber esta nova requisição GET, detecta o parâmetro mensagem na URL.
     + Ele renderiza a página e insere a mensagem em um elemento HTML oculto com atributos data-\*.
     + O script.js no lado do cliente lê esses atributos quando a página carrega e exibe uma notificação **pop-up** não obstrutiva e temporária, informando o usuário sobre o resultado da sua ação.
     + Finalmente, o script limpa a URL para remover o parâmetro da mensagem, garantindo uma experiência limpa.
* **Justificativa:** Esta abordagem oferece múltiplos benefícios:
  1. **Previne Duplicidade:** O padrão PRG é uma prática recomendada que impede o reenvio acidental de formulários.
  2. **Comunicação Clara:** O usuário recebe um feedback visual imediato e claro sobre o sucesso ou falha de suas ações.
  3. **Experiência Não Intrusiva:** As mensagens são exibidas em pop-ups temporários que não exigem uma ação do usuário para desaparecer e não poluem a interface permanentemente.
  4. **Statelessness:** A comunicação do feedback é feita via URL, evitando a necessidade de gerenciar o estado da mensagem em sessões no servidor, o que mantém a arquitetura mais simples.

# **Conclusão**

A abordagem de comunicação escolhida combina a solidez e a simplicidade da renderização no lado do servidor com técicas modernas de feedback ao usuário. Ela é perfeitamente adequada ao escopo da aplicação, garantindo um sistema funcional, de fácil manutenção e que oferece uma experiência de uso clara e eficiente, sem a sobrecarga de arquiteturas mais complexas.