**Caso de Uso 1: Autenticar Usuário**

* **Perspectiva do Usuário:**
  + **Objetivo:** Quero acessar o sistema de forma segura para usar minhas funcionalidades.
  + **Expectativas:** O fluxo é claro e direto: acesso a página, preencho e-mail e senha, e entro. Espero que o sistema me avise de forma explícita se eu errar as credenciais ou deixar um campo em branco, permitindo que eu tente novamente.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Formulário de Login), Backend (Serviço de Autenticação, Controlador de Sessão), Banco de Dados (Tabela de Usuários).
  + **Regras de Negócio:** O sistema deve validar se as credenciais correspondem a um registro no banco de dados. A senha deve ser comparada usando um algoritmo de hash seguro. Em caso de sucesso, uma sessão de usuário deve ser criada, associada ao seu papel (Usuário ou Administrador). É necessário implementar a lógica para lidar com as duas condições de erro: credenciais inválidas e campos vazios.
  + **Desafios:** Implementar a gestão de sessão (ex: JWT) e o controle de acesso baseado em papéis (RBAC) que será usado em todo o sistema. Garantir a segurança contra ataques de força bruta (ex: limitando tentativas de login).
* **Perspectiva do Testador:**
  + **Cenário de Sucesso (Fluxo Principal):**
    1. **Dado** um usuário com credenciais válidas, **quando** ele preenche o formulário e clica em "Entrar", **então** o sistema deve autenticá-lo e redirecioná-lo para a tela inicial.
  + **Cenários de Falha (Sequências Alternativas):**
    1. **Dado** um usuário com e-mail ou senha incorretos, **quando** ele tenta logar, **então** o sistema deve exibir a mensagem "credenciais são inválidas" e permanecer na tela de login.
    2. **Dado** um usuário que deixa um dos campos em branco, **quando** ele tenta logar, **então** o sistema deve exibir a mensagem "solicitando o preenchimento dos campos em branco".
  + **Validações:** Verificar se as mensagens de erro correspondem a cada cenário. Confirmar que a sessão é criada apenas no cenário de sucesso. Validar se o acesso a páginas restritas é bloqueado sem autenticação.

**Caso de Uso 2: Gerenciar Usuários**

* **Perspectiva do Usuário (Administrador):**
  + **Objetivo:** Quero ter controle total sobre as contas de usuário do sistema.
  + **Expectativas:** O processo de gerenciar usuários (criar, editar, excluir) deve ser centralizado e claro. Espero que o sistema me impeça de cometer erros, como criar um usuário com um e-mail que já existe ou me excluir acidentalmente do sistema.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Painel de Administração de Usuários), Backend (API com endpoints CRUD para /users), Banco de Dados.
  + **Regras de Negócio:** Acesso restrito ao papel "Administrador". A validação de dados no backend é crucial: verificar campos obrigatórios, formato de e-mail e, principalmente, a unicidade do e-mail no banco antes de criar um novo usuário. Implementar uma lógica de proteção que impeça a exclusão da própria conta do administrador logado.
  + **Desafios:** Construir uma UI que lide com múltiplos fluxos (criação, edição, visualização) de forma clara. Garantir que a lógica de validação no backend seja robusta para cobrir todos os cenários alternativos.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Validar todas as quatro operações CRUD (criar, visualizar, editar, excluir) com dados válidos.
  + **Cenários de Falha:**
    1. Tentar criar um usuário com um e-mail que já está em uso.
    2. Tentar criar/editar um usuário com dados inválidos (ex: nome em branco).
    3. Tentar excluir a própria conta de administrador logado.
    4. Tentar acessar a funcionalidade com um usuário comum.
  + **Validações:** Verificar no banco de dados se as operações foram refletidas corretamente. Confirmar que as mensagens de erro são específicas para cada tipo de falha. Validar que o bloqueio de autoexclusão funciona.

**Caso de Uso 3: Criar Churrasco**

* **Perspectiva do Usuário (Criador):**
  + **Objetivo:** Registrar um novo churrasco de forma simples e rápida.
  + **Expectativas:** Espero um formulário claro com todos os campos necessários para o evento. Se eu esquecer algum campo obrigatório, o sistema deve me avisar para que eu possa corrigir.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Formulário de Criação), Backend (API com endpoint POST /churrascos), Banco de Dados (Tabela churrascos).
  + **Regras de Negócio:** O sistema deve associar automaticamente o usuário logado como o criador do evento. Uma validação no backend deve garantir que todos os campos obrigatórios (título, data, hora, local, etc.) foram preenchidos antes de salvar.
  + **Desafios:** Criar componentes de UI amigáveis para a seleção de data e hora. Garantir a consistência dos tipos de dados (ex: formato de data, valor monetário).
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Preencher todos os campos do formulário com dados válidos e salvar, verificando a mensagem de sucesso.
  + **Cenário de Falha:** Tentar salvar o formulário com um ou mais campos obrigatórios em branco.
  + **Validações:** Após o sucesso, verificar na base de dados se o novo registro foi criado com os dados corretos e associado ao ID do criador. No cenário de falha, validar se a mensagem de erro é exibida e se nenhum registro é criado no banco.

**Caso de Uso 4: Montar Carrinho com Alertas de Consumo**

* **Perspectiva do Usuário (Criador):**
  + **Objetivo:** Montar a lista de compras para o churrasco recebendo ajuda do sistema para calcular as quantidades certas.
  + **Expectativas:** Quero uma interface onde eu adicione itens e o sistema me dê feedback em tempo real, com alertas, se a quantidade for pouca ou exagerada, baseado nas regras. Se eu ignorar os alertas, quero poder salvar mesmo assim.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Tela do Carrinho, com lógica de cálculo), Backend (API para salvar o carrinho e buscar regras), Banco de Dados (Tabelas de Itens, Carrinho, Regras).
  + **Regras de Negócio:** A lógica principal reside no frontend. Ao adicionar/alterar um item, o sistema deve: (1) buscar o número de participantes confirmados; (2) buscar as regras globais; (3) calcular a proporção por pessoa; (4) comparar com a regra e exibir o alerta. O backend deve validar os dados antes de salvar (ex: quantidades não podem ser negativas).
  + **Desafios:** Garantir a performance da interface para que os cálculos e a exibição de alertas sejam instantâneos, proporcionando uma boa experiência ao usuário. Lidar com o caso de não haver regras globais definidas.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Adicionar itens e quantidades, e salvar o carrinho com sucesso.
  + **Cenários de Falha e Alternativos:**
    1. Tentar salvar um carrinho com quantidades inválidas (ex: -1kg de picanha).
    2. Montar um carrinho quando não há regras de consumo definidas pelo administrador.
    3. Salvar o carrinho ignorando os alertas de consumo.
  + **Validações:** Validar a precisão dos cálculos de proporção por pessoa. Verificar se os alertas aparecem e desaparecem corretamente conforme as quantidades são ajustadas. Confirmar que o carrinho é salvo corretamente no banco de dados.

**Caso de Uso 5: Consolidar Prestação de Contas**

* **Perspectiva do Usuário (Criador):**
  + **Objetivo:** Quero fechar as contas do evento para saber o resultado final (lucro ou prejuízo).
  + **Expectativas:** Um botão para "Consolidar" que me mostre um resumo claro: total arrecadado, total gasto e o saldo. Espero que o sistema me avise se houver pendências (ex: pagamentos não confirmados) que impeçam o fechamento final.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Página de Prestação de Contas), Backend (API com endpoint para POST /churrascos/{id}/consolidar), Banco de Dados.
  + **Regras de Negócio:** O sistema deve agregar (SUM) o valor de todos os participantes com status "confirmado" para obter o total arrecadado. Deve agregar o valor de todas as compras registradas para obter o total gasto. O saldo é a diferença entre os dois. A opção "Consolidação Final" deve ser bloqueada se existirem pendências e, quando executada, deve travar o evento para novas edições financeiras.
  + **Desafios:** Garantir a precisão e a consistência dos cálculos, que dependem da integridade de múltiplos registros (participantes e compras). Implementar a lógica de "travamento" do evento após o fechamento final.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Realizar uma consolidação parcial e, posteriormente, uma consolidação final com todos os dados corretos.
  + **Cenários de Falha e Alternativos:**
    1. Tentar consolidar um evento sem nenhum pagamento confirmado ou compra registrada.
    2. Tentar realizar o fechamento final quando ainda existem participantes com pagamento pendente.
    3. Tentar registrar uma nova compra após o fechamento final ter sido realizado.
  + **Validações:** Montar um cenário completo com múltiplos participantes e compras, e validar se os valores calculados pelo sistema batem com um cálculo manual. Verificar se o evento é efetivamente bloqueado para novas edições financeiras após o fechamento.

**Caso de Uso 5: Editar Churrasco**

* **Perspectiva do Usuário (Criador):**
  + **Objetivo:** Quero corrigir ou atualizar as informações de um churrasco que já criei.
  + **Expectativas:** Espero encontrar facilmente o churrasco na minha lista, e ao clicar em "Editar", ver um formulário já preenchido com os dados atuais. A alteração deve ser simples e, ao salvar, quero ver as informações atualizadas na página de detalhes. O sistema deve me impedir de salvar se eu apagar uma informação obrigatória.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Formulário de Edição pré-preenchido), Backend (API com endpoints GET /churrascos/{id} para buscar os dados e PUT /churrascos/{id} para atualizar), Banco de Dados.
  + **Regras de Negócio:** A principal regra é a autorização: o sistema deve verificar se o usuário logado é o mesmo que criou o evento antes de permitir a edição. A validação dos dados (campos obrigatórios, formatos) deve ser reaplicada ao salvar as alterações. O sistema também deve lidar com a tentativa de editar um churrasco que não existe mais.
  + **Desafios:** Garantir que a camada de autorização seja robusta para impedir que um usuário edite o evento de outro.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso (Fluxo Principal):**
    1. **Dado** um criador de um evento, **quando** ele seleciona o evento, altera um ou mais campos com dados válidos e salva, **então** o sistema deve confirmar a alteração e os novos dados devem ser persistidos.
  + **Cenários de Falha (Sequências Alternativas):**
    1. Tentar editar um churrasco que foi excluído ou cujo ID é inválido.
    2. Apagar o conteúdo de um campo obrigatório (como o título) e tentar salvar.
    3. Um usuário que não é o criador do evento tentar acessar a funcionalidade de edição.
  + **Validações:** Verificar no banco de dados se os campos foram efetivamente atualizados. Validar que as mensagens de erro são apropriadas para cada cenário de falha. Confirmar que a regra de autorização está sendo aplicada corretamente.

**Caso de Uso 6: Visualizar Meus Churrascos**

* **Perspectiva do Usuário:**
  + **Objetivo:** Quero um local centralizado para ver todos os churrascos com os quais estou envolvido, seja como criador ou como convidado.
  + **Expectativas:** Uma lista simples e organizada que me mostre rapidamente os eventos. Se eu não tiver nenhum churrasco, espero uma mensagem amigável que me incentive a criar um.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Tela "Meus Churrascos"), Backend (API com endpoint GET /churrascos/meus), Banco de Dados.
  + **Regras de Negócio:** A consulta ao banco de dados é o ponto central. Ela deve retornar todos os eventos onde o id do usuário logado corresponda ao criador\_id OU exista uma entrada para ele na tabela de participantes daquele evento.
  + **Desafios:** Otimizar a consulta para garantir a performance, especialmente para usuários com muitos eventos, possivelmente implementando paginação.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:**
    1. Logar com um usuário que criou churrascos e verificar se todos eles aparecem.
    2. Logar com um usuário que foi apenas convidado e verificar se os convites aparecem.
    3. Logar com um usuário que criou alguns e foi convidado para outros, e verificar se a lista unificada está correta.
  + **Cenário de Falha (Sequência Alternativa):**
    1. Logar com um usuário novo que não tem nenhum evento associado.
  + **Validações:** A lista de eventos na tela deve corresponder exatamente ao resultado da consulta no banco de dados para aquele usuário. No cenário de falha, validar que a mensagem "não há eventos disponíveis" é exibida.

**Caso de Uso 7: Convidar Participantes**

* **Perspectiva do Usuário (Criador):**
  + **Objetivo:** Quero adicionar pessoas ao meu churrasco de forma prática.
  + **Expectativas:** Espero poder escolher pessoas de uma lista de usuários do sistema e também poder adicionar o e-mail de quem não tem conta. Após o envio, quero uma confirmação de que os convites foram disparados.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (UI de Convite), Backend (API, Serviço de Notificação por E-mail), Banco de Dados (Tabela de associação churrasco\_participantes).
  + **Regras de Negócio:** O sistema deve validar o formato dos e-mails inseridos. Deve haver uma lógica para não permitir convites duplicados para o mesmo evento.
  + **Desafios:** A integração com um serviço de e-mail de terceiros é o principal desafio, incluindo o tratamento de erros como falhas no envio ou e-mails inválidos.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Convidar um usuário já cadastrado e um e-mail externo, e confirmar o envio.
  + **Cenários de Falha (Sequências Alternativas):**
    1. Tentar enviar convites sem selecionar nenhum participante ou inserir e-mails.
    2. Tentar enviar um convite para um e-mail com formato inválido (ex: "usuario@dominio").
    3. Tentar convidar a mesma pessoa duas vezes para o mesmo evento.
  + **Validações:** Verificar se o e-mail de convite é de fato recebido pelo destinatário. Validar no banco de dados se a associação entre o evento e os participantes foi criada corretamente. Confirmar que as mensagens de erro são exibidas em cada cenário de falha.

**Caso de Uso 8: Registrar Compras do Evento**

* **Perspectiva do Usuário (Criador):**
  + **Objetivo:** Quero lançar todas as despesas do churrasco para manter o controle financeiro.
  + **Expectativas:** Um formulário simples e direto para registrar cada compra com item, quantidade, valor e fornecedor. Após salvar, quero ver o item na lista de compras do evento.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Formulário de Despesas), Backend (API com endpoint POST /churrascos/{id}/compras), Banco de Dados (Tabela compras).
  + **Regras de Negócio:** O backend deve validar os dados de entrada, garantindo que campos obrigatórios não estão vazios e que os valores numéricos (quantidade, valor) são válidos (ex: maiores que zero).
  + **Desafios:** Garantir uma boa usabilidade para que o registro de múltiplas compras seja uma tarefa rápida e não tediosa.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Preencher todos os campos com dados válidos e salvar a compra com sucesso.
  + **Cenários de Falha (Sequências Alternativas):**
    1. Tentar salvar o formulário com campos obrigatórios em branco.
    2. Tentar salvar com dados inválidos (ex: valor com letras, quantidade negativa).
  + **Validações:** Verificar no banco de dados se o registro da compra foi criado com as informações corretas. Validar que as mensagens de erro apropriadas são exibidas nos cenários de falha.

**Caso de Uso 9: Anexar Comprovantes**

* **Perspectiva do Usuário (Criador):**
  + **Objetivo:** Quero associar as notas fiscais a cada compra que registrei, para ter tudo organizado para a prestação de contas.
  + **Expectativas:** Em cada compra listada, espero ver um botão "Anexar Comprovante". Ao clicar, o sistema deve me permitir escolher um arquivo (foto ou PDF) do meu computador e enviá-lo.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Componente de Upload), Backend (API com endpoint POST /compras/{id}/comprovante, Serviço de Arquivos), Armazenamento de Arquivos (ex: Cloud Storage).
  + **Regras de Negócio:** O sistema deve validar o tipo de arquivo (extensão) e o tamanho máximo permitido. O arquivo não é salvo diretamente no banco relacional; em vez disso, ele é enviado para um serviço de armazenamento e o link/caminho é salvo no registro da compra.
  + **Desafios:** A implementação do upload de arquivos é o principal desafio, incluindo a segurança (para evitar o envio de arquivos maliciosos) e a gestão do armazenamento.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Selecionar uma compra, fazer o upload de um arquivo válido (ex: .jpg, .pdf) e confirmar que ele ficou associado à compra.
  + **Cenários de Falha (Sequências Alternativas):**
    1. Tentar fazer upload de um arquivo com formato não suportado (ex: .exe).
    2. Tentar fazer upload de um arquivo maior que o limite definido.
    3. Tentar anexar sem selecionar um arquivo.
  + **Validações:** Após o upload, verificar se o arquivo pode ser visualizado ou baixado. Validar que o sistema exibe as mensagens de erro corretas para cada cenário de falha.

**Caso de Uso 10: Visualizar Prestação de Contas**

* **Perspectiva do Usuário:**
  + **Objetivo:** Quero ver o resultado financeiro do churrasco de forma clara e transparente.
  + **Expectativas:** Uma tela de resumo que mostre os totais (arrecadado, gasto) e o saldo final. Espero poder ver a lista de despesas e os comprovantes que o criador anexou.
* **Perspectiva do Projetista:**
  + **Componentes:** Frontend (Página de Prestação de Contas), Backend (API com endpoint GET /churrascos/{id}/prestacaodecontas).
  + **Regras de Negócio:** O acesso a esta tela só deve ser permitido se a prestação de contas já tiver sido consolidada pelo criador. A API deve buscar os totais calculados e a lista de compras com seus respectivos links para os comprovantes.
  + **Desafios:** Apresentar as informações de forma clara e de fácil entendimento tanto para o criador quanto para os participantes.
* **Perspectiva do Testador:**
  + **Cenário de Sucesso:** Acessar um evento encerrado e visualizar o resumo financeiro e os comprovantes corretamente.
  + **Cenário de Falha (Sequência Alternativa):** Tentar visualizar a prestação de contas de um evento que ainda não foi consolidado pelo criador.
  + **Validações:** Confirmar que os dados exibidos no resumo correspondem aos totais consolidados no banco de dados. Validar que a mensagem "resumo ainda não está disponível" é mostrada no cenário de falha.