Relatório Técnico: Plataforma de Gestão de Eventos e Bilheteira

Projeto de Final de Curso

Autor: Rhudan Moura

Data: 16 de Dezembro de 2025

---

1. Introdução e Objetivos

Desenvolvi este projeto com o objetivo de criar uma solução web *Full Stack* para a gestão integral do ciclo de vida de eventos, desde a criação no *backoffice* até à emissão de bilhetes digitais para o cliente final. O sistema foi projetado para responder a todos os requisitos funcionais propostos, garantindo uma separação estrita entre a interface pública (Front-office) e a área de gestão (Back-office).

A minha prioridade técnica foi construir uma arquitetura **segura, escalável e sem dependências excessivas**. Optei deliberadamente por não utilizar *frameworks* pesados (como Laravel ou React) para demonstrar um domínio profundo da linguagem PHP nativa, da manipulação avançada do DOM com JavaScript (Vanilla) e, crucialmente, da lógica de base de dados com SQL avançado (Transações, Triggers e Stored Procedures).

---

2. Arquitetura do Sistema e Decisões de Design

Embora tenha utilizado PHP nativo, estruturei o projeto seguindo princípios de **Separação de Responsabilidades (SoC)**, organizando o código em módulos lógicos:

- **Estrutura de Pastas:** Separei a lógica administrativa (admin/) da lógica pública e criei uma camada de serviços reutilizáveis na pasta php/ (ex: database.php para conexão singleton, cart.php para lógica de sessão).
- **Componentização:** Para evitar repetição de código (DRY - *Don't Repeat Yourself*), isolei elementos visuais comuns como navbar.php e footer.php, que são injetados dinamicamente e adaptam o seu conteúdo consoante o estado de autenticação do utilizador.
- **Gestão de Caminhos (Routing Dinâmico):** Um dos desafios técnicos que resolvi foi a gestão de caminhos relativos entre o painel admin e o site público. Implementei uma solução robusta no ficheiro config/api-config.js, que deteta automaticamente a profundidade da URL e ajusta os *endpoints* das APIs AJAX, prevenindo erros de "404 Not Found" independentemente da pasta onde o script é executado.

---

3. Stack Tecnológica e Justificação

- **Front-end:**
    - **HTML5 & CSS3:** Utilizei CSS moderno com variáveis (:root) no index.css para manter a consistência visual.
    - **Bootstrap 5:** Escolhido pela sua grelha flexível, garantindo responsividade total (Mobile-First) e componentes visuais rápidos (Modais, Alerts).
    - **JavaScript (Vanilla & Fetch API):** Optei por não usar jQuery para reduzir o peso da página. Toda a comunicação assíncrona (AJAX) — como adicionar ao carrinho ou validar formulários — é feita via fetch(), permitindo uma experiência fluida sem recarregamentos constantes de página.
- **Back-end:**
    - **PHP 8:** Utilizado com tipagem forte e tratamento de exceções (try-catch).
    - **PDO (PHP Data Objects):** A camada de acesso a dados utiliza PDO em vez de mysqli para garantir portabilidade e, mais importante, o uso obrigatório de *Prepared Statements*, eliminando riscos de SQL Injection.
    - **MySQL (MariaDB):** Não atua apenas como repositório de dados, mas contém lógica de negócio crítica (ver secção 5).

---

4. Implementação Detalhada dos Requisitos

A. Segurança e Autenticação (Defesa em Profundidade)

A segurança não foi uma funcionalidade "adicional", mas sim a base do sistema:

- **Criptografia:** As senhas nunca são salvas em texto limpo. Utilizei o algoritmo **Bcrypt** através de password_hash() no registo e password_verify() no login.
- **Proteção de Sessão:** Todas as páginas críticas (especialmente em admin/) iniciam com uma verificação rigorosa de $_SESSION['user_type']. Se um utilizador tentar aceder a uma URL de admin diretamente, é imediatamente redirecionado.

- **Padrão PRG (Post-Redirect-Get):** No painel de administração (create_event.php), implementei o padrão PRG para impedir que formulários sejam reenviados acidentalmente caso o administrador atualize a página (F5), evitando duplicidade de dados.

B. Gestão de Eventos e Uploads

- **Tratamento de Ficheiros:** No módulo create_event.php, implementei uma lógica de upload segura. O sistema renomeia o ficheiro utilizando uniqid() para evitar conflitos de nomes no servidor e valida o tipo de arquivo para garantir que apenas imagens são carregadas.
- **Exclusão Segura (Double Authentication):** Desenvolvi um mecanismo de segurança elevado no ficheiro secure_delete.php. Para apagar registos sensíveis, o sistema exige que o administrador **reintroduza a sua senha**. Esta validação é feita via AJAX, retornando um JSON que autoriza ou nega a exclusão no banco de dados e remove fisicamente a imagem do servidor.

C. Lógica de Carrinho e Compras (Transações ACID)

Esta é a componente tecnicamente mais complexa do projeto:

1. **Persistência:** O carrinho é salvo na base de dados (tabela cart) e não apenas na sessão. Isso permite que o utilizador troque de dispositivo sem perder os itens selecionados.
2. **Transações Atómicas:** No ficheiro checkout.php, utilizei CreatePurchase encapsulado numa transação ($pdo->beginTransaction()). Isto garante a integridade **ACID**:
    - Ou a compra é registada, o stock é baixado e os bilhetes gerados com sucesso;
    - Ou, se houver qualquer erro (ex: falha na rede ou stock esgotado no milissegundo da compra), é feito um rollBack(), cancelando tudo. Isso impede a venda de bilhetes sem stock ("overbooking").

D. Interface Dinâmica

- **Modais Assíncronos:** A visualização de detalhes do evento não requer mudança de página. O script events.js popula um Modal Bootstrap injetando dados via JSON, melhorando a UX (Experiência do Utilizador).
- **Feedback em Tempo Real:** O ícone do carrinho na navbar e os alertas de sucesso utilizam AJAX para atualizar instantaneamente sem recarregar o site.

---

5. Engenharia de Dados (SQL)

Fui além do CRUD básico, transferindo regras de negócio críticas para a camada de dados para garantir performance e integridade:

Estrutura Normalizada

- **users:** Tabela única para Clientes e Admins, diferenciados por uma flag de permissão (user_type).
- **events & categories:** Relacionamento para categorização eficiente.
- **tickets:** Tabela de ligação que gera códigos únicos (ticket_code) para cada bilhete individual dentro de uma compra.

Automação via Banco de Dados

1. **Stored Procedures:**
    - CreatePurchase: Procedimento que recebe o ID do utilizador, itera sobre o carrinho, gera os registos na tabela purchases e tickets e limpa o carrinho automaticamente.
    - CleanOldCartItems: Rotina de manutenção que remove itens abandonados no carrinho há mais de 1 hora, libertando stock "preso".
2. **Triggers:**
    - after_ticket_delete: Um gatilho crucial. Se um bilhete for cancelado ou apagado pelo administrador, este trigger **devolve automaticamente** a unidade ao campo available_tickets da tabela de eventos. Isso elimina o erro humano na reposição de stock.
3. **Views:**
    - event_sales: Uma visão que pré-calcula o total de vendas e receita por evento, utilizada pelo Dashboard para gerar gráficos sem sobrecarregar o PHP com cálculos matemáticos.

---

6. Instruções de Instalação e Configuração

Requisitos do Servidor

- PHP 8.0+ com extensão PDO ativada.
- MySQL/MariaDB com suporte a Triggers e Procedures.
- Servidor Apache com mod_rewrite (opcional, mas recomendado).

Procedimento de Instalação

1. **Base de Dados:** Importar o *dump* sistema_eventos.sql. Este ficheiro contém a estrutura (DDL), os dados iniciais (DML) e a definição das Stored Procedures e Triggers.
2. **Configuração:** Editar php/database.php com as credenciais locais (localhost, root, etc.).
3. **Permissões:** Conceder permissão de escrita/leitura (chmod 755 ou 777 em ambiente dev) à pasta uploads/events/ para permitir o armazenamento de imagens.

Acesso Administrativo (Default)

Para efeitos de avaliação, o sistema possui um super-administrador pré-configurado:

- **Email:** admin@eventos.pt
- **Senha:** admin123

---

7. Conclusão e Lições Aprendidas

O desenvolvimento deste projeto permitiu consolidar competências transversais de desenvolvimento web. O maior desafio técnico superado foi a **gestão de concorrência no stock de bilhetes**, problema resolvido através da implementação de Transações SQL e Procedimentos Armazenados.

O resultado final é uma plataforma robusta, onde a segurança (validação de inputs, proteção de rotas e integridade de dados) foi priorizada tanto quanto a funcionalidade. O sistema está preparado para escalar, com uma base de dados normalizada e um código PHP modular e de fácil manutenção.