# **Plano de Testes – EventHive**

## **Objetivo Geral**

Este documento fornece um guia completo e estruturado para testar todas as funcionalidades do sistema EventHive, abrangendo a base de dados, o backend e o frontend.

**Ferramentas Utilizadas:**

* **Postman:** Testes de API
* **Google Chrome DevTools:** Testes de Frontend
* **MySQL Workbench / phpMyAdmin:** Testes de Base de Dados
* **Visual Studio Code:** Desenvolvimento e análise

### **URLs de Teste**

* **Frontend:** http://localhost:3000
* **Backend API:** http://localhost/EventHive/backend/api/
* **Base de Dados:** http://localhost/phpmyadmin

Cada teste está organizado com objetivos claros, passos detalhados e resultados esperados para garantir a qualidade e o funcionamento correto do sistema.

## **1. Testes à Base de Dados (MySQL)**

**Objetivo:** Validar a integridade, constraints, triggers, views e regras de negócio da base de dados.

**Ferramentas:** phpMyAdmin (http://localhost/phpmyadmin) ou MySQL Workbench.

**Pré-requisito:** Base de dados eventhive\_db importada e funcional.

### **1.1. Teste BD-1: Constraints de Integridade**

**Objetivo:** Validar que a base de dados rejeita a inserção de dados inválidos.

**1.1.1. BD-1.1: Unicidade em Roles**

* **Ação:** INSERT INTO roles (role\_name) VALUES ('admin');
* **Resultado Esperado:** ERRO - Duplicate entry 'admin' for key 'roles.role\_name\_UNIQUE'

**1.1.2. BD-1.2: Unicidade de Email em Users**

* **Ação:** INSERT INTO users (id\_role, first\_name, last\_name, email, password\_hash, active) VALUES (2, 'Teste', 'Duplicado', 'admin@eventhive.com', '123456', 1);
* **Resultado Esperado:** ERRO - Duplicate entry 'admin@eventhive.com' for key 'e-mail'

**1.1.3. BD-1.3: Chave Estrangeira Inválida em Users**

* **Ação:** INSERT INTO users (id\_role, first\_name, last\_name, email, password\_hash, active) VALUES (99, 'Teste', 'Role Invalida', 'invalido@eventhive.com', '123456', 1);
* **Resultado Esperado:** ERRO - Cannot add or update a child row: a foreign key constraint fails (fk\_users\_roles)

**1.1.4. BD-1.4: Proteção ON DELETE RESTRICT em Roles**

* **Ação:** DELETE FROM roles WHERE id\_role = 2; (Assumindo que o role está em uso)
* **Resultado Esperado:** ERRO - Cannot delete or update a parent row: a foreign key constraint fails

### **1.2. Teste BD-2: Operações em Cascata e Proteções**

**Objetivo:** Testar o comportamento das regras ON DELETE CASCADE e ON DELETE RESTRICT.

**1.2.1. BD-2.1: CASCADE - Eliminar Evento Remove Bilhetes Associados**

**Passo 1:** Inserir um evento de teste.

INSERT INTO events (id\_event, id\_category, title, description, event\_date, location) VALUES (99, 1, 'Evento de Teste para Apagar', '...', '2026-01-01 20:00:00', 'Local de Teste');

**Passo 2:** Inserir bilhetes para esse evento.

INSERT INTO tickets (id\_event, ticket\_type, price, quantity\_available) VALUES (99, 'Bilhete Teste', 10.00, 100);

**Passo 3:** Confirmar que os bilhetes existem.

SELECT \* FROM tickets WHERE id\_event = 99;

**Passo 4:** Eliminar o evento.

DELETE FROM events WHERE id\_event = 99;

**Passo 5:** Verificar que os bilhetes foram eliminados.

SELECT \* FROM tickets WHERE id\_event = 99;

**Resultado Esperado:** A última consulta não retorna resultados, confirmando que os bilhetes foram eliminados em cascata.

**1.2.2. BD-2.2: RESTRICT - Não é Possível Eliminar Bilhete com Vendas**

* **Ação:** DELETE FROM tickets WHERE id\_ticket = 1;

(Assumindo que id\_ticket=1 tem vendas associadas na tabela order\_items)

* **Resultado Esperado:** ERRO - Cannot delete or update a parent row: a foreign key constraint fails (fk\_order\_items\_tickets)

### **1.3. Teste BD-3: Triggers Automáticos**

**Objetivo:** Verificar que os triggers da base de dados funcionam corretamente.

**1.3.1. BD-3.1: Trigger de Subtração de Stock (trg\_after\_order\_item\_insert)**

**Passo 1:** Anotar o stock inicial de um bilhete (ex: id\_ticket = 3).

SELECT quantity\_available FROM tickets WHERE id\_ticket = 3; -- Ex: 500

**Passo 2:** Criar uma nova encomenda.

INSERT INTO orders (id\_user, total\_price) VALUES (1, 350.00);

SET @last\_order\_id = LAST\_INSERT\_ID();

**Passo 3:** Adicionar itens a essa encomenda, acionando o trigger.

INSERT INTO order\_items (id\_order, id\_ticket, quantity, price\_per\_ticket) VALUES (@last\_order\_id, 3, 10, 35.00);

**Passo 4:** Verificar o novo valor do stock.

SELECT quantity\_available FROM tickets WHERE id\_ticket = 3;

**Resultado Esperado:** O stock diminuiu em 10 unidades (de 500 para 490).

### **1.4. Teste BD-4: Views e Consultas Complexas**

**Objetivo:** Validar que as views retornam os dados corretos e estão sincronizadas.

**1.4.1. BD-4.1: View vw\_event\_details - Validação de JOIN**

* **Ação:** SELECT id\_event, title, category\_name FROM vw\_event\_details WHERE id\_event = 1;
* **Resultado Esperado:** O campo category\_name mostra o nome da categoria (ex: 'Música'), e não o seu ID.

**1.4.2. BD-4.2: View vw\_event\_details - Sincronização em Tempo Real**

**Passo 1:** Atualizar o título de um evento na tabela principal.

UPDATE events SET title = 'Concerto Acústico de Verão - Edição Especial' WHERE id\_event = 1;

**Passo 2:** Consultar a view para verificar a alteração.

SELECT title FROM vw\_event\_details WHERE id\_event = 1;

**Resultado Esperado:** A view reflete o novo título imediatamente.

## **2. Testes à API Backend (PHP)**

**Objetivo:** Validar todos os endpoints REST da API, incluindo lógica de negócio, autenticação e respostas.

**Ferramentas:** Postman

**Base URL:** http://localhost/EventHive/backend/api/

**Headers**: Content-Type: application/json

**Cookie**: PHPSESSID=... (usar o valor do login)

**NOTA**: Guardar o cookie **PHPSESSID** devolvido na resposta para usar nos próximos pedidos autenticados. "embaixo do botao send "

**Pré-requisito:** Servidor XAMPP ativo com a API configurada e funcional.

### **2.1. Módulo Auth - Autenticação e Gestão de Utilizadores**

**2.1.1. Registo de Utilizador**

**Método:** POST

**URL:** http://localhost/EventHive/backend/api/ /auth/register.php

**Body (JSON):**

{

"first\_name": "Joao",

"last\_name": "Silva",

"email": "joao@email.com",

"password": "123456"

}

**Respostas Esperadas:**

**Sucesso:** {"success": true, "message": "Registo efetuado com sucesso."}

**Email duplicado:** {"success": false, "message": "Erro ao registar utilizador."}

**Dados em falta:** {"success": false, "message": "Dados em falta."}

**2.1.2. Login**

**Método:** POST

**URL:** <http://localhost/EventHive/backend/api/auth/login.php>

**Headers**:

**content-Type:** application/json

**Body (JSON):**

{

"email": "joao@email.com",

"password": "123456"

}

**Respostas Esperadas:**

**Sucesso:** {"success": true, "message": "Login efetuado com sucesso."} (Verificar cookie PHPSESSID na resposta).

**Credenciais inválidas:** {"success": false, "message": "Credenciais inválidas."}

**Tentativas falhadas:** {"success": false, "message": "Demasiadas tentativas falhadas. Tenta novamente daqui a 10 minutos."}

**Validação:** {"success": false, "message": "Email inválido."} ou {"success": false, "message": "Password demasiado curta."}

**2.1.3. Logout**

**Método:** POST

**URL:** /auth/logout.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login )

**Body:** Vazio

**Respostas Esperadas:**

**Sucesso:** {"success": true, "message": "Sessão terminada com sucesso."}

**Após logout (teste de acesso):** Ao aceder a um endpoint protegido (ex: /auth/profile.php), a resposta deve ser {"success": false, "message": "Não autenticado."}.

**2.1.4. Profile (Requer Autenticação)**

**Método:** GET

**URL:** [http://localhost/EventHive/backend/api](http://localhost/EventHive/backend/api/auth/login.php)/auth/profile.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login )

**Respostas Esperadas:**

**Sucesso:** {"success": true, "user": {"id\_user": 1, "first\_name": "Admin", ...}}

**Sem sessão:** {"success": false, "message": "Não autenticado."}

**2.1.5. Alterar Password (Requer Autenticação)**

**Método:** POST

**URL:** h[ttp://localhost/EventHive/backend/api](http://localhost/EventHive/backend/api/auth/login.php)/auth/change\_password.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login )

**Body (JSON):**

{

"current\_password": "123456",

"new\_password": "novaSenha123"

}

**Respostas Esperadas:**

**Sucesso:** {"success": true, "message": "Password alterada com sucesso."}

**Password atual incorreta:** {"success": false, "message": "Password atual incorreta."}

**Sem sessão:** {"success": false, "message": "Não autenticado."}

### **2.2. Módulo Events - Gestão de Eventos**

**2.2.1. Listar todos os eventos**

**Método:** GET

**URL:** <http://localhost/EventHive/backend/api/events/get_all_events.php>

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login login)

**Respostas Esperadas:**

**Sucesso:** {"success": true, "events": [{"id\_event": 1, ...}]}

**Sem eventos:** {"success": false, "message": "Nenhum evento encontrado."}

**2.2.2. Detalhes de um evento**

**Método:** GET

**URL:** <http://localhost/EventHive/backend/api/events/get_event_details.php?id=1>

**Headers**:

**Content-Type:** application/json

**Cookie**: não e necessária a autenticação

**Respostas Esperadas:**

**Sucesso:** {"success": true, "event": {"id\_event": 1, "tickets": [...]}}

**ID inválido:** {"success": false, "message": "ID do evento inválido."}

**Não encontrado:** {"success": false, "message": "Evento não encontrado."}

### **2.3. Módulo Cart - Carrinho de Compras**

**2.3.1. Adicionar ao carrinho (Requer Autenticação)**

**Método:** POST

**URL:** http://localhost/EventHive/backend/api/cart/add\_to\_cart.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login)

**Body (JSON):**

{

"ticket\_id": 1,

"quantity": 2

}

**Respostas Esperadas:**

**Sucesso:** {"success": true, "message": "Item adicionado ao carrinho."}

**Dados inválidos:** {"success": false, "message": "Dados inválidos."}

**2.3.2. Ver carrinho (Requer Autenticação)**

**Método:** GET

**URL:** http://localhost/EventHive/backend/api/cart/get\_cart.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login)

**Resposta Esperada:** {"success": true, "cart": {"items": [...], "total": 50.00}}

**2.3.3. Remover do carrinho (Requer Autenticação)**

**Método:** POST

**URL:** http://localhost/EventHive/backend/api/cart/remove\_from\_cart.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login)

**Body (JSON):**

{

"ticket\_id": 1

}

**Respostas Esperadas:**

**sucesso:** {"success": true, "message": "Item removido do carrinho."}

**Item não encontrado:** {"success": false, "message": "Item não encontrado no carrinho."}

**2.3.4. Checkout (Requer Autenticação)**

**Método:** POST

**URL:** http://localhost/EventHive/backend/api/cart/checkout.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login)

**Body:** Vazio

**Respostas Esperadas:**

**Sucesso:** {"success": true, "message": "Compra finalizada com sucesso."}

**Sem sessão:** {"success": false, "message": "Não autenticado."}

**Stock insuficiente:** {"success": false, "message": "Erro no checkout: Stock insuficiente."}

**Preço manipulado:** {"success": false, "message": "Erro no checkout: Preço inválido."}

### **2.4. Módulo User - Dados do Utilizador**

**2.4.1. Histórico de compras (Requer Autenticação)**

**Método:** GET

**URL:** http://localhost/EventHive/backend/api/user/purchase\_history.php

**Headers**:

**Content-Type:** application/json

**Cookie**: PHPSESSID=... (usar o valor do cookie PHPSESSID criado para o login)

**Resposta Esperada:**

{

"success": true,

"history": [

{

"id\_order": 1,

"order\_date": "2025-07-31 14:26:00",

"total\_price": "100.00",

"event\_title": "Concerto Acústico de Verão",

...

}

]

}

**2.4.2 MODULO get\_all\_categories - Obter todas as categorias**

**Method**: GET

**URL**: http://localhost/EventHive/backend/api/categories/get\_all\_categories.php

**Headers**: Nenhum necessário

**Body**: Nenhum  
  
**Resposta esperada:**  
{

"success": true,

"data": [

{

"id\_category": "1",

"name": "Música"

},

{

"id\_category": "2",

"name": "Desporto"

},

{

"id\_category": "3",

"name": "Teatro"

},

{

"id\_category": "4",

"name": "Cinema"

},

{

"id\_category": "5",

"name": "Arte"

},

{

"id\_category": "6",

"name": "Gastronomia"

}

]

}  
  
**Resposta de ERRO:**  
{  
 "success": false,  
 "message": "Nenhuma categoria encontrada."  
}

### **2.5. Módulo Admin - Gestão Administrativa (Requer Role Admin)**

**NOTA:** Todos os endpoints seguintes requerem um Cookie: PHPSESSID=... de um utilizador com role= 1 (admi) .

**2.5.1. CRUD de Utilizadores (/admin/users\_crud.php)**

**Listar (GET):** Retorna {"success": true, "users": [...]}.

**Criar (POST) → Body** : com id\_role, first\_name, last\_name, email, password\_hash. Retorna {"success": true}.

**Atualizar (PUT) ->** **Body** com id\_user e os campos a alterar. Retorna {"success": true}.

**Eliminar (DELETE) ->** **Body** com {"id\_user": 10}. Retorna {"success": true}.

**2.5.2. CRUD de Eventos (/admin/events\_crud.php)**

**Listar (GET):** Retorna {"success": true, "events": [...]}.

**Criar (POST) → Body** : com id\_category, title, description, etc. Retorna {"success": true}.

**Atualizar (PUT) ->** **Body** com id\_event e os campos a alterar. Retorna {"success": true}.

**Eliminar (DELETE) ->** **Body** com {"id\_event": 3}. Retorna {"success": true}.

**2.5.3. CRUD de Categorias (/admin/categories\_crud.php)**

**Listar (GET):** Retorna {"success": true, "categories": [...]}.

**Criar (POST) → Body** com {"name": "Nova Categoria"}. Retorna {"success": true}.

**Atualizar (PUT) -> Body** com {"id\_category": 2, "name": "Categoria Editada"}. Retorna {"success": true}.

**Eliminar (DELETE) ->** **Body** com {"id\_category": 2}. Retorna {"success": true}.

### **2.6. Nota Importante: Proteções da Base de Dados**

Devido às proteções de integridade, a API deve impedir a eliminação de:

* **Categorias** que tenham eventos associados.
* **Eventos** que tenham já bilhetes vendidos.
* **Roles** que estejam a ser utilizadas por utilizadores.
* O sistema implementa **soft delete** para utilizadores e eventos, marcando-os como inativos em vez de os apagar fisicamente, preservando assim o histórico.

## **3. Testes ao Frontend (React)**

**Objetivo:** Validar a interface, navegação, reatividade, integração com a API e a experiência geral do utilizador.

**Ferramentas:** Google Chrome, DevTools.

**Base URL:** http://localhost:3000

**Pré-requisito:** npm start executado, com backend e base de dados funcionais.

### **3.1. Cenário 1: Fluxo Completo do Utilizador Final**

**Objetivo:** Simular a jornada completa de um utilizador, do regi

**Fluxo a Testar:**

**Registo (/register):**

* + Criar uma nova conta (ex: "Maria Silva", "maria@teste.com", "123456").
  + Verificar se é redirecionado para a página de login após o sucesso.

**Login (/login):**

* + Entrar com as credenciais recém-criadas.
  + Verificar se o AuthContext é atualizado e o nome do utilizador aparece no header.

**Navegação (/events):**

* + Explorar a lista de eventos.
  + Testar a funcionalidade de filtro por categoria.

**Seleção de Evento (/event/1):**

* + Aceder à página de detalhes de um evento.
  + Adicionar 2 bilhetes ao carrinho.

**Carrinho (/cart):**

* + Verificar se os itens e as quantidades estão corretos.
  + Confirmar que o total é calculado corretamente.

**Checkout:**

Clicar em "Finalizar Compra".

* + Verificar se recebe uma mensagem de sucesso e o carrinho fica vazio.
  + Confirmar na base de dados que o stock do bilhete foi atualizado.

**Histórico de Compras (/profile):**

* + Aceder à página de perfil e verificar se a nova compra aparece no histórico com os dados corretos.

**Resultado Esperado:** O fluxo completo deve ocorrer sem erros visuais ou de consola, com todos os dados a serem exibidos e atualizados de forma consistente entre o frontend e o backend.

### **3.2. Cenário 2: Fluxo Completo do Administrador**

**Objetivo:** Testar todas as funcionalidades do painel de administração.

**Fluxo a Testar:**

**Login Admin:**

* + Fazer login com credenciais de administrador (ex: "admin@eventhive.com").
  + Verificar se o botão de acesso ao "Dashboard" aparece na página de perfil.

**Dashboard (/admin):**

* + Confirmar que as estatísticas (utilizadores, vendas, etc.) são carregadas corretamente.

**Gestão de Categorias:**

* + Criar, editar e eliminar uma categoria.
  + Tentar eliminar uma categoria que está associada a um evento e verificar se a ação é bloqueada com uma mensagem de erro.

**Gestão de Eventos:**

* + Criar um evento com múltiplos tipos de bilhetes e fazer upload de uma imagem.
  + Editar um evento existente.
  + "Eliminar" um evento (soft delete) e confirmar que o seu estado muda para 'Terminado'.
  + Verificar que não é possível editar ou eliminar um evento com bilhetes já vendidos.

**Gestão de Utilizadores:**

* + Visualizar a lista completa de utilizadores.
  + Editar os dados de um utilizador.
  + "Eliminar" um utilizador (soft delete).
  + Tentar eliminar ou editar a conta de super-admin e confirmar que a ação é bloqueada.
  + Tentar criar um utilizador com um email já existente e verificar a mensagem de erro.

**Lista de Encomendas:**

* + Verificar a lista de todas as encomendas do sistema.
  + Inspecionar os detalhes de uma encomenda para confirmar que todos os dados são exibidos.

## **4. Notas para Execução dos Testes**

### **4.1. Pré-requisitos Técnicos**

**XAMPP:** Instalado e com os módulos Apache e MySQL ativos.

**Node.js:** Versão 18 ou superior.

**Composer:** Instalado e funcional.

**Base de Dados:** eventhive\_db importada no MySQL.

**Dependências:** Executar composer install no backend e npm install no frontend.

### **4.2. Dicas de Ferramentas**

**Postman:** Para os testes de API, lembre-se de guardar o cookie PHPSESSID após o login para usar em pedidos autenticados.

**Headers:** Para pedidos POST, PUT, e DELETE, use sempre o header Content-Type: application/json.

**Browser DevTools:** Mantenha o painel "Network" e "Console" abertos para monitorizar pedidos de API e detetar erros no frontend.