RELATÓRIO DO HERMES API

Relato das divergências entre a documentação e os testes

Auth

Rota de cadastro e login do usuário, além da renovação da sua chave principal.

POST (/auth/register)

Cadastro de um novo usuário.

Testes realizados:

- Seguindo o modelo de entrada padrão;
- Tentar cadastrar um e-mail já existente;
- Tentar cadastrar sem escrever a senha;
- Tentar cadastrar com senha menor que 8 caracteres;
- Tentar cadastrar um e-mail diferente de "@usp.br";
- Tentar cadastrar um e-mail sem o "@";
- Tentar cadastrar sem escrever um nome;
- Tentar cadastrar com nome menor que 3 letras.

Caso o usuário desrespeite alguma das regras impostas na documentação, o sistema retorna o erro correspondente corretamente.

POST (/auth/login)

Inicia a sessão do usuário.

Testes realizados:

- Seguindo o modelo de entrada padrão;
- Tentar entrar com e-mail não cadastrado;
- Tentar entrar com senha incorreta;
- Tentar com o campo de e-mail vazio;
- Tentar com o campo de senha vazio.

Os testes demonstraram que a aplicação retorna as respostas corretas de erro e sucesso em cada caso.

GET (/auth/refresh)

Renova o token de acesso.

Ele atualiza novamente o token, mas acaba por deixar o usuário não verificado.

POST (/auth/logout)

Finaliza a sessão do usuário.

Funcionando corretamente.

GET (/auth/verify-email/)

Manda o email de verificação para o usuário.

A verificação funciona, mas o email cai na caixa de SPAM.

PATCH (/auth/change-password)

Atualiza a senha do usuário.

Sem erros aparentes.

Testes realizados:

- Modelo padrão;
- Senha antiga incorreta;
- Nova senha vazia;
- Nova senha fora dos padrões.

Todos os testes obtiveram sucesso.

Ponto a destacar: O backend poderia reforçar a verificação de requisitos mínimos de segurança da senha (ex: tamanho mínimo, caracteres especiais, etc.).

Events

POST (/events)

Criação de novos eventos (restrito a administradores).

Testes realizados:

- Tentar criar sem ser administrador;
- Criar usando modelo padrão;
- Tentar criar sem título;
- Tentar criar sem body;
- Tentar criar sem local;
- Tentar criar sem data inicial;
- Tentar criar sem data final;
- Tentar criar sem imagem;
- Tentar criar sem lista de tags;
- Tentar criar o mesmo evento duas ou mais vezes.

Todos os testes funcionaram conforme esperado, exceto o último, pois é possível criar o mesmo evento múltiplas vezes (problema de duplicidade).

GET (/events)

Lista de eventos.

Testes realizados:

- Sem eventos;
- Com eventos, sem filtro;
- Com filtro por título;
- Com filtro por tags;
- Com filtros combinados (título + tags).

Não apresentou anormalidades em relação à documentação. Tanto o *limit* e o *offset* funcionam como esperado.

GET (/events/{id})

Ele não está retornando os dados das tags.

Diferente esperado, não retornando os dados das tags cadastradas no evento, não retornando as tags e o nome do autor.

PATCH (/events/{id})

Atualiza os dados dos eventos.

As validações dos campos são feitas corretamente. Rota em funcionamento.

Pontos a destacar: Campos como local e status não são atualizados. Seriam campo importantes para atualizar?

DELETE (/events/{id})

Deleta um evento pelo seu ID.

Funcionalidade sem erros.

Info

POST (/infos)

Cadastro de blocos de informação (restrito a administradores).

Testes realizados:

- Tentar criar sem ser administrador;
- Criar usando modelo padrão;
- Tentar criar sem título;
- Tentar criar sem body;
- Tentar criar sem lista de tags;
- Tentar criar o mesmo evento duas ou mais vezes.

Assim como em Events, é possível criar informações duplicadas.

GET (/infos)

Lista de blocos de informação. Tanto o *limit* e o *offset* funcionam como esperado.

Testes realizados:

- Sem registros;
- Com registros, sem filtro;
- Com filtro por título;
- Com filtro por tags;
- Com filtros combinados.

O filtro por tags não está funcionando corretamente.

GET (/infos/{id})

Lista um dos eventos.

Dados não condizentes com o esperado. A rota não retorna as tags e o autor do post.

PATCH (/infos/{id})

Atualiza os dados das informações.

Os campos estão estão sendo verificados.

Nota: não atualiza o local.

DELETE (/infos/{id})

Delete uma informação específica pelo ID.

Sem erros demonstrados.

Tags

GET (/tags)

Lista todas as tags ativas.

Funcionando corretamente.

GET (/tags/{name})

Busca uma tag específica pelo nome.

Funcionando corretamente.

POST (/tags)

Cria uma nova tag.

É obrigatório escrever algum texto, porém não há validação de tamanho mínimo acima de 1 caractere.

User

GET (/users/me)

Retorna os dados do usuário com base no token.

A documentação indica que apenas administradores teriam acesso, porém a rota retorna os dados do próprio usuário autenticado. Provavelmente trata-se de erro na documentação.

PATCH (/users/me)

Atualiza os dados do usuário (atualmente apenas o nome).

Testes realizados:

- Nome padrão;
- Campo vazio;
- Nome com menos de 3 caracteres.

A aplicação retorna erro, mas a mensagem não explica claramente o motivo. Poderia haver mensagens mais explícitas para evitar confusão.

Também é possível utilizar um nome já existente no banco.

POST (/users/me/tags)

Relaciona uma tag com o usuário, entrando para a lista de favoritos do usuário (tb_user_tag).

Quando é enviada uma tag inexistente, é retornado um erro genérico, sem indicar claramente se é erro de servidor ou tag não encontrada.

A resposta está diferente do Swagger, não retornando a tag seguida pelo usuário.

DELETE (/users/me/tags/{tagId})

Apaga um tag dos favoritos do usuário pelo ID dela.

Funcionando corretamente.

Observações Gerais

- A verificação de administrador e autenticação está funcionando adequadamente.
- O controle para que o usuário apenas atualize e delete seus próprios dados está correto.

- O usuário precisa ser verificado para poder realizar ações no aplicativo.

Sugestões

- Aplicar middleware de autenticação em todas as rotas para evitar abusos e possíveis ataques (ex: DDOS).
- Implementar tratamento para usuários que não confirmam o e-mail até a data limite (7 dias).
- No POST /auth/register, caso o envio de e-mail falhe, o usuário ainda é criado no banco. Essa exceção deve ser tratada para evitar inconsistência de dados.