Descrição dos Arquivos

Arquivo .env

Descrição

Este arquivo é o .env, responsável por armazenar as configurações sensíveis de ambiente, como as credenciais de acesso ao banco de dados. Ele facilita a configuração da aplicação em diferentes ambientes (desenvolvimento, produção, etc.), permitindo que as informações não fiquem hardcoded no código-fonte.

Descrição das variáveis:

* DB_HOST=localhost: Especifica o endereço do servidor de banco de dados. Neste caso, o banco está rodando localmente.
* DB_USER=root: Define o nome de usuário que será utilizado para acessar o banco de dados.
* DB_PASSWORD=desafio: Senha associada ao usuário do banco de dados. Importante ser mantida em segredo.
* DB_NAME=StarWars_DB: Nome do banco de dados onde as informações relacionadas ao projeto serão armazenadas.

Esse arquivo mantém os dados de conexão de forma segura e centralizada, simplificando a manutenção e a configuração do ambiente.

Arquivo App.js

Este arquivo é o ponto de entrada principal da aplicação Node.js utilizando o framework Express. Ele configura o servidor, define as rotas e também gerencia a criação da tabela de personagens no banco de dados, além de habilitar o CORS para permitir requisições de outras origens.

Descrição das funções e variáveis:

* express: Framework web para Node.js que facilita a criação de APIs e servidores HTTP.
* cors: Middleware para habilitar CORS (Cross-Origin Resource Sharing), permitindo que outras aplicações façam requisições ao servidor.
* routes: Importa as rotas definidas no arquivo routes.js, que será usado para a lógica de rotas da API.
* createCharactersTable: Função responsável por criar a tabela de personagens no banco de dados, se ela não existir.
* app: Instância da aplicação Express, que será usada para configurar as rotas e middleware.
* port: Porta na qual o servidor irá rodar (3000).

Middleware:

* app.use(cors()): Ativa o CORS, permitindo que o servidor aceite requisições de domínios diferentes, facilitando a comunicação entre o front-end e back-end.

* app.use(express.json()): Middleware para interpretar o corpo das requisições HTTP em formato JSON.

Rotas:

* app.use('/api', routes): Todas as rotas definidas no arquivo routes.js serão acessadas sob o caminho /api. Por exemplo, uma rota em routes.js como /characters seria acessível via /api/characters.
* app.get('/', ...): Define uma rota básica na raiz (/), que retorna uma página HTML simples de boas-vindas com uma mensagem personalizada.

Inicialização do Servidor:

* app.listen(port, async () => {...}): Inicia o servidor na porta definida e executa um callback assim que o servidor começa a rodar.
* await createCharactersTable(): Ao iniciar o servidor, chama a função para criar a tabela de personagens, garantindo que a estrutura do banco de dados esteja pronta.
* console.log(...): Exibe uma mensagem no console indicando que o servidor está ativo e escutando na porta definida.

Este arquivo configura o servidor para aceitar requisições, aplica middleware para tratar dados e define a rota de boas-vindas e as rotas da API, além de garantir que a tabela no banco de dados seja criada quando o servidor é iniciado.

Arquivo db.js

Este arquivo configura a conexão com o banco de dados MySQL, além de fornecer funções para manipular a tabela de personagens, incluindo inserção, atualização, marcação como favorito, exclusão e consulta. Ele utiliza o módulo mysql2/promise, que permite o uso de conexões com o banco de dados de forma assíncrona com Promises.

Descrição das funções e variáveis:

* mysql.createPool({...}): Cria uma pool de conexões com o banco de dados, permitindo que múltiplas conexões sejam reutilizadas de forma eficiente.
* pool: A pool de conexões que será usada para todas as interações com o banco de dados, evitando a criação de novas conexões a cada operação.

Funções CRUD e de manipulação de personagens:

createCharactersTable():

* Cria a tabela de personagens no banco de dados, se ainda não existir.
* Query: CREATE TABLE IF NOT EXISTS characters
* Campos da tabela incluem nome, altura, cor dos olhos, entre outros

```
async function createCharactersTable() {
  const query = `
    CREATE TABLE IF NOT EXISTS characters (
      id INT AUTO_INCREMENT PRIMARY KEY,
      name VARCHAR(255) NOT NULL UNIQUE,
```

```
      height VARCHAR(50),
      mass VARCHAR(50),
      hair_color VARCHAR(50),
      skin_color VARCHAR(50),
      eye_color VARCHAR(50),
      birth_year VARCHAR(50),
      gender VARCHAR(50),
      favorite BOOLEAN DEFAULT 0
    );
  `;
  await pool.query(query);
  console.log('Tabela "characters" criada com sucesso!');
}
```

importCharacter(character):

*Insere ou atualiza um personagem no banco de dados. Se o nome já existir, as demais informações serão atualizadas.

Query: INSERT INTO ... ON DUPLICATE KEY UPDATE

favoriteCharacter(name):

* Marca um personagem como favorito, alterando o campo favorite para 1.

Query: UPDATE characters SET favorite = 1 WHERE name = ?

unfavoriteCharacter(name):

* Desmarca um personagem como favorito, alterando o campo favorite para 0.

Query: UPDATE characters SET favorite = 0 WHERE name = ?

deleteCharacterByName(name):

* Exclui um personagem específico com base no nome.

Query: DELETE FROM characters WHERE name = ?

deleteAllCharacters():

* Exclui todos os personagens da tabela.

Query: DELETE FROM characters

getFavoriteCharacters():

* Retorna uma lista de todos os personagens que estão marcados como favoritos.

Query: SELECT * FROM characters WHERE favorite = 1

getCharacterByName(name):

*Busca um personagem específico pelo nome.

Query: SELECT * FROM characters WHERE name = ?

getAllCharacters():

* Retorna todos os personagens da tabela.

Query: SELECT * FROM characters

Exportações:

```
module.exports = {
  pool,
  createCharactersTable,
  importCharacter,
  favoriteCharacter,
  unfavoriteCharacter,
  deleteCharacterByName,
  deleteAllCharacters,
  getFavoriteCharacters,
  getCharacterByName,
  getAllCharacters
};
```

* As funções e a pool de conexões são exportadas para que possam ser utilizadas em outros módulos, como as rotas da API.

Este arquivo configura a conexão com o banco de dados e fornece funções que gerenciam os dados dos personagens no banco,
permitindo que sejam importados, atualizados, marcados/desmarcados como favoritos, listados e excluídos.

Arquivo Routes.js

Descrição

Este código define várias rotas utilizando o framework Express e faz uso da API SWAPI (Star Wars API) para buscar informações
de personagens. Também faz a integração com um banco de dados para manipulação de personagens, como marcar/desmarcar favoritos,
listar, importar ou apagar personagens. O código ainda utiliza a biblioteca Axios para realizar requisições HTTP à API SWAPI.

Descrição das Funções e Rotas:

Função fetchCharacterFromAPI(name):

* Faz uma requisição à API SWAPI e busca um personagem específico pelo nome.
* A função percorre a resposta da API e tenta encontrar o personagem pelo nome.
* Caso o personagem seja encontrado, retorna-o; caso contrário, retorna null.
* Comentário: Essa função é útil para casos em que o personagem solicitado não esteja no banco de dados local, sendo então buscado diretamente na API.

Rota GET /characters/external:

* Busca personagens da API StarWars, com a opção de filtrar pelo nome usando um parâmetro de consulta (query).
* A rota percorre todas as páginas da API (em SWAPI, os dados são paginados) e retorna todos os personagens encontrados.
* Caso seja passado um termo de pesquisa, a lista de personagens é filtrada com base no nome.
* Comentário: Esta rota pode ser usada para importar novos personagens diretamente da API StarWars, sem depender do banco de dados local.

Rota PATCH /characters/favorite:

* Marca um personagem como favorito pelo nome, alterando o estado do banco de dados.
* Se o personagem não estiver no banco de dados, tenta buscá-lo na API, adicioná-lo ao banco e marcá-lo como favorito.
* Comentário: Esta rota garante que, mesmo que um personagem ainda não tenha sido importado, ele possa ser marcado como favorito automaticamente após a importação.

Rota PATCH /characters/unfavorite:

* Desmarca um personagem como favorito.
* Se o personagem não for encontrado, retorna uma mensagem de erro.
* Comentário: Permite que os usuários alterem o status de um personagem favorito sem removê-lo completamente do banco.

Rota GET /characters/favorites:

* Lista todos os personagens marcados como favoritos no banco de dados.
* Se não houver personagens favoritos, retorna uma mensagem apropriada.
* Comentário: Esta rota é útil para visualizar rapidamente todos os personagens que foram salvos como favoritos pelos usuários.

Rota POST /characters/import:

* Importa todos os personagens disponíveis na API SWAPI e armazena-os no banco de dados.
* Percorre todas as páginas da API e adiciona os personagens um por um ao banco.
* Comentário: Usada para garantir que o banco de dados local tenha uma cópia de todos os personagens disponíveis na API SWAPI.

Rota GET /characters:

* Lista todos os personagens armazenados no banco de dados.
* Se não houver personagens, retorna uma mensagem de erro.
* Comentário: Esta rota exibe todos os personagens que já foram importados ou adicionados ao banco de dados, possibilitando sua visualização local.

Rota DELETE /characters:

* Apaga um personagem do banco de dados pelo nome, baseado em um parâmetro de consulta (query).
* Se o personagem não for encontrado, retorna uma mensagem de erro.
* Comentário: Esta rota permite que personagens sejam removidos individualmente do banco de dados.

Rota DELETE /characters/all:

* Remove todos os personagens do banco de dados.
* Comentário: Útil para operações de limpeza, removendo rapidamente todos os personagens armazenados.

Comentários Importantes:

* A integração com o banco de dados permite manter uma lista local de personagens, facilitando operações como favoritos e filtragens.
* A utilização de Axios para chamadas à API externa garante que dados de personagens sempre possam ser recuperados da SWAPI, mesmo se não estiverem localmente no banco.
* A paginização na API SWAPI é tratada adequadamente, garantindo que todas as páginas de personagens sejam percorridas.

Esta e uma descrição dos principais arquivos do StarWars-Api.