connection

Descripción del Archivo

Este archivo establece una conexión con una base de datos MySQL para una tienda de juegos. Utiliza la biblioteca `mysql` para crear y manejar la conexión.

Dependencias

- `mysql`: Una biblioteca de Node.js que proporciona la funcionalidad necesaria para interactuar con bases de datos MySQL.

Documentación de Módulo

Este módulo gestiona la conexión a la base de datos de una tienda de juegos. Es parte de una aplicación más grande que podría incluir operaciones como consultas, inserciones y actualizaciones de datos relacionados con la tienda.

Funcionalidades Clave

- Establece conexión con la base de datos MySQL.
- Exporta la conexión para su uso en otras partes de la aplicación.

Documentación de Funciones/Métodos

`mysql.createConnection(config)`

- **Resumen**: Crea una conexión a la base de datos MySQL.
- **Parámetros**:
  - `config`: Objeto que contiene la configuración de la conexión.
    - `host`: String, el hostname del servidor de la base de datos.
    - `user`: String, el usuario de la base de datos.
    - `password`: String, la contraseña del usuario.
    - `database`: String, el nombre de la base de datos a conectar.
    - `port`: String, el puerto del servidor de la base de datos.
- **Valor de Retorno**: Objeto de conexión a la base de datos.

`mysqlConnection.connect(callback)`

- **Resumen**: Inicia la conexión a la base de datos.

- **Parámetros**:
  - `callback`: Función que se ejecuta al intentar la conexión. Recibe un parámetro `error` que indica si hubo un error durante la conexión.
- **Valor de Retorno**: No retorna un valor. Imprime en consola el estado de la conexión.

Comentarios en Bloques de Código

El bloque de código principal maneja la conexión a la base de datos y reporta el estado de la conexión, ya sea exitosa o fallida, a través de la consola.

Casos Especiales y Suposiciones

- **Suposiciones**: Se asume que el servidor de MySQL está corriendo en `localhost` y escucha en el puerto `3306`.
- **Casos Especiales**: Si la conexión falla, se captura el error y se muestra en la consola, lo que ayuda en la depuración.

Ejemplos de Uso

```
const db = require('./path_to_this_file');

// Uso de la conexión exportada para realizar una consulta
db.query('SELECT * FROM products', (error, results, fields) => {
    if (error) throw error;
    console.log(results);
});
```

Este ejemplo muestra cómo importar y utilizar la conexión a la base de datos para realizar una consulta simple.

verifyToken

```
// Importing JSON Web Token library
const jwt = require('jsonwebtoken');

/**
 * Middleware to verify JWT tokens in HTTP requests.
 * This module provides functionality to decrypt and verify JWT tokens
included in the HTTP Authorization header.
 */
const verifyToken = {
  /**
   * Verifies the JWT token from the Authorization header of the request.
   *
   * @param {Object} req - The HTTP request object, containing headers and
other request information.
   * @param {Object} res - The HTTP response object, used to send responses
back to the client.
   * @param {Function} next - The callback function to pass control to the
next middleware function.
   */
  verify(req, res, next) {
    // Check if the 'Authorization' header is present
    if (!req.headers.authorization) {
      console.log(req.headers.authorization);
      res.status(401).json('Unauthorized');
    } else {
      // Extract the token from the Authorization header
      const token = req.headers.authorization.substr(7);

      // Check if the token is not empty
      if (token !== '') {
        try {
          // Decrypts the content of the token using the secret word
          const content = jwt.verify(token, 'Testing');
          req.data = content;
          next();
        } catch (error) {
          // Handle errors that occur during token verification
          res.status(401).json('Invalid Token');
        }
      } else {
        // Respond with 'Empty Token' if the token is empty
```

```
        res.json('Empty Token');
      }
    }
  }
};

// Export the verifyToken module
module.exports = verifyToken;
```

Ejemplos de Uso

```
const express = require('express');
const app = express();
const verifyToken = require('./path_to_verifyToken');

// Example of a protected route
app.get('/protected', verifyToken.verify, (req, res) => {
  res.send('Access granted to protected content');
});

// Start the server
app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Este ejemplo muestra cómo utilizar el middleware `verifyToken` para proteger rutas en una aplicación Express, asegurando que solo los usuarios con tokens válidos puedan acceder a ciertos endpoints.

register

Documentación del Módulo: Gestión de Usuarios

Este módulo se encarga de la gestión de usuarios, permitiendo la creación de nuevos usuarios mediante el método POST. Utiliza `express` para el manejo de rutas y `mysql` para la conexión a la base de datos.

Dependencias:

- **express**: Framework de Node.js para aplicaciones web.
- **mysql**: Cliente de MySQL para Node.js.
- **jsonwebtoken**: Utilizado para la creación de tokens de seguridad.

Clases y Módulos:

- **router**: Gestiona las rutas específicas para las operaciones de usuarios.
- **mysqlConnection**: Maneja la conexión a la base de datos MySQL.

Funciones y Métodos:

- **router.post('/')**: Método para crear un nuevo usuario.
  - **Parámetros**:
    - `req` (Object): Objeto de solicitud HTTP, contiene información como el cuerpo de la solicitud.
    - `res` (Object): Objeto de respuesta HTTP, utilizado para devolver respuestas al cliente.
  - **Cuerpo de la solicitud**:
    - `mail` (String): Correo electrónico del usuario.
    - `completeName` (String): Nombre completo del usuario.
    - `password` (String): Contraseña del usuario.
    - `creditCard` (String): Número de tarjeta de crédito del usuario.
    - `expirationDate` (String): Fecha de expiración de la tarjeta de crédito.
    - `ccv` (String): Código de seguridad de la tarjeta de crédito.
  - **Respuesta**:
    - Si la inserción es exitosa, devuelve un JSON con la operación realizada.
    - En caso de error, devuelve un mensaje indicando datos inválidos.
  - **Excepciones**:

- Errores de conexión a la base de datos.
- Errores en la ejecución de la consulta SQL.

Ejemplos de Uso:

```
const axios = require('axios');

// Datos del usuario a crear
const userData = {
  mail: "usuario@example.com",
  completeName: "Nombre Completo",
  password: "contraseña123",
  creditCard: "1234567890123456",
  expirationDate: "01/23",
  ccv: "123"
};

// Llamada al endpoint para crear un usuario
axios.post('http://localhost:3000/', userData)
  .then(response => {
    console.log('Respuesta:', response.data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Casos Especiales y Suposiciones:

- Se asume que los datos de entrada siempre están completos y en el formato correcto.
- El código maneja entradas inesperadas mediante la validación de errores en la consulta SQL.

Este módulo es fundamental para la gestión de usuarios dentro de la aplicación, permitiendo la expansión y mantenimiento de la base de usuarios de manera segura y eficiente.

login

Documentación del Módulo de Autenticación

Descripción General

Este módulo implementa la funcionalidad de autenticación para una aplicación utilizando Express y MySQL. Permite a los usuarios iniciar sesión mediante la verificación de credenciales y devuelve un token JWT si las credenciales son válidas.

Dependencias

- **express**: Framework utilizado para manejar las solicitudes HTTP.
- **mysql**: Módulo para la conexión y manejo de la base de datos MySQL.
- **jsonwebtoken**: Utilizado para generar tokens JWT que se utilizan para la autenticación.

Documentación de Clase/Módulo

Este módulo no define clases explícitas pero configura un router de Express para manejar las solicitudes de autenticación.

Funcionalidades Principales:

- Autenticación de usuarios.
- Generación de token JWT.

Documentación de Funciones/Métodos

router.post('/', (req, res) => {...})

Resumen: Maneja las solicitudes POST para iniciar sesión de usuarios.

Parámetros:

- `req`: Objeto de solicitud HTTP, esperando que contenga un cuerpo con `mail` y `password`.
- `res`: Objeto de respuesta HTTP utilizado para enviar respuestas al cliente.

Valor de Retorno: No aplica directamente, pero envía una respuesta JSON al cliente.

Excepciones:

- Genera excepciones relacionadas con errores de conexión a la base de datos o errores en la ejecución de la consulta.

Comentarios en Bloques de Código

El bloque principal maneja la lógica de autenticación:

1. Extrae `mail` y `password` del cuerpo de la solicitud.
2. Realiza una consulta SQL para verificar si existe un usuario con las credenciales proporcionadas.
3. Si el usuario existe, genera un token JWT y lo envía junto con información del usuario como respuesta.
4. Si no existen datos válidos o hay un error, envía una respuesta de datos inválidos.

Casos Especiales y Suposiciones

- **Suposiciones**: Se asume que los datos de entrada siempre estarán en formato JSON con campos `mail` y `password`.
- **Casos Especiales**: Si no se encuentran usuarios o las credenciales son incorrectas, se devuelve un mensaje de "Invalid data".

Documentación para el Usuario Final

Para utilizar este endpoint, el desarrollador debe enviar una solicitud POST con un JSON que incluya `mail` y `password`. Si las credenciales son correctas, el servidor responderá con un token JWT y detalles del usuario, que pueden utilizarse para autenticaciones subsiguientes.

Ejemplo de Uso:

```
axios.post('/api/login', {
    mail: 'usuario@example.com',
    password: 'password123'
}).then(response => {
    console.log(response.data);
}).catch(error => {
    console.error(error);
});
```

Este ejemplo muestra cómo un cliente podría utilizar Axios para enviar una solicitud de inicio de sesión y manejar la respuesta del servidor.

categories

Documentación del Código para la Gestión de Videojuegos, Consolas y Accesorios

Descripción General

Este archivo de código forma parte de una aplicación de backend desarrollada con Node.js y Express. Su propósito principal es manejar las solicitudes HTTP relacionadas con videojuegos, consolas y accesorios, así como la gestión de libros mediante una API RESTful. Utiliza MySQL como sistema de gestión de base de datos.

Dependencias

- **express**: Framework de Node.js para construir aplicaciones web y APIs.
- **mysql**: Módulo de Node.js para interactuar con bases de datos MySQL.

Clases y Módulos

Este archivo no define clases propias, pero utiliza el módulo `router` de Express para definir rutas de la API.

Funciones y Métodos

GET /get-games

- **Descripción**: Retorna todos los videojuegos disponibles en la base de datos.
- **Parámetros**: No aplica.
- **Retorno**: JSON con la lista de videojuegos.
- **Excepciones**: Loguea errores en la consola si la consulta falla.

GET /get-consoles

- **Descripción**: Retorna todas las consolas disponibles en la base de datos.
- **Parámetros**: No aplica.
- **Retorno**: JSON con la lista de consolas.
- **Excepciones**: Loguea errores en la consola si la consulta falla.

GET /get-accessories

- **Descripción**: Retorna todos los accesorios disponibles en la base de datos.
- **Parámetros**: No aplica.
- **Retorno**: JSON con la lista de accesorios.
- **Excepciones**: Loguea errores en la consola si la consulta falla.

POST /insert-book

- **Descripción**: Permite insertar un nuevo libro en la base de datos si el usuario tiene permisos adecuados.
- **Parámetros**:
  - `title`: Título del libro.
  - `author`: Autor del libro.
  - `quantity`: Cantidad de libros a añadir.
- **Retorno**: Mensaje de éxito o error.
- **Excepciones**: Retorna un error si los datos JSON están mal formados o si el usuario no tiene suficientes permisos.

PUT /update-book

- **Descripción**: Actualiza los detalles de un libro existente en la base de datos si el usuario tiene permisos adecuados.
- **Parámetros**:
  - `book_id`: ID del libro a actualizar.
  - `title`: Nuevo título del libro.
  - `author`: Nuevo autor del libro.
  - `quantity`: Nueva cantidad de libros.
- **Retorno**: Mensaje de éxito o error.
- **Excepciones**: Retorna un error si los datos JSON están mal formados, el `book_id` no existe o si el usuario no tiene suficientes permisos.

DELETE /delete-book

- **Descripción**: Elimina un libro de la base de datos si el usuario tiene permisos adecuados.
- **Parámetros**:
  - `book_id`: ID del libro a eliminar.
- **Retorno**: Mensaje de éxito o error.
- **Excepciones**: Retorna un error si los datos JSON están mal formados, el `book_id` no existe o si el usuario no tiene suficientes permisos.

Ejemplos de Uso

Obtener videojuegos:

```
fetch('/get-games')
  .then(response => response.json())
  .then(data => console.log(data));
```

Insertar un libro:

```
fetch('/insert-book', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    title: 'Nuevo Libro',
    author: 'Autor Desconocido',
    quantity: 10
  })
})
.then(response => response.json())
.then(data => console.log(data));
```

Casos Especiales y Suposiciones

- Se asume que los datos de entrada para las operaciones de inserción y actualización están correctamente formateados y completos.
- Los endpoints de gestión de libros requieren que el usuario esté autenticado y tenga roles de 'admin' o 'user'. Si no se cumplen estas condiciones, se retorna un mensaje de error.

offers

Documentación del Código para el Módulo de Ofertas en Videojuegos

Descripción General

Este módulo de Node.js utiliza Express y MySQL para proporcionar endpoints que permiten a los clientes obtener información sobre ofertas en videojuegos, consolas y accesorios. Está diseñado para ser parte de una aplicación más grande que maneja datos de productos de videojuegos.

Dependencias

- **express**: Framework de Node.js para construir aplicaciones web y APIs.
- **mysql**: Módulo de Node.js para interactuar con bases de datos MySQL.

Documentación de Clases/Módulos

Este archivo no define clases explícitas, sino que configura rutas en un router de Express que interactúan con una base de datos MySQL.

Funciones/Métodos

`router.get('/get-game-offers', callback)`

Descripción

Obtiene una lista de juegos que están actualmente en oferta.

Parámetros

- `req` (Object): El objeto de solicitud HTTP.
- `res` (Object): El objeto de respuesta HTTP.

Retorno

No retorna valores directamente, pero envía un JSON con los datos de los juegos en oferta.

Excepciones

Loguea errores en la consola si la consulta a la base de datos falla.

`router.get('/get-console-offers', callback)`

Descripción

Obtiene una lista de consolas que están actualmente en oferta.

Parámetros

- `req` (Object): El objeto de solicitud HTTP.
- `res` (Object): El objeto de respuesta HTTP.

Retorno

No retorna valores directamente, pero envía un JSON con los datos de las consolas en oferta.

Excepciones

Loguea errores en la consola si la consulta a la base de datos falla.

`router.get('/get-accessory-offers', callback)`

Descripción

Obtiene una lista de accesorios que están actualmente en oferta.

Parámetros

- `req` (Object): El objeto de solicitud HTTP.
- `res` (Object): El objeto de respuesta HTTP.

Retorno

No retorna valores directamente, pero envía un JSON con los datos de los accesorios en oferta.

Excepciones

Loguea errores en la consola si la consulta a la base de datos falla.

Comentarios en Bloques de Código

Los bloques de código dentro de cada método de ruta están documentados internamente para explicar el proceso de consulta a la base de datos y manejo de errores.

Casos Especiales y Suposiciones

- **Suposiciones**: Se asume que la base de datos está correctamente configurada y que las tablas `games`, `consoles`, y `accessories` existen y tienen una columna `offer`.
- **Casos Especiales**: Si no hay ofertas disponibles, se retorna un arreglo vacío.

Ejemplos de Uso

Obtener ofertas de juegos

```
axios.get('/get-game-offers')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error al obtener ofertas de juegos:', error);
  });
```

Obtener ofertas de consolas

```
axios.get('/get-console-offers')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error al obtener ofertas de consolas:', error);
  });
```

Obtener ofertas de accesorios

```
axios.get('/get-accessory-offers')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error al obtener ofertas de accesorios:', error);
  });
```

Este módulo facilita la integración de ofertas en una aplicación de comercio electrónico o en un portal de información sobre videojuegos, proporcionando datos actualizados sobre productos en oferta.