Manual Técnico

[API]

Proyecto: [Portal web – cédula de mercados]

Elaborado por:

Alvarado Mombela Cristopher Yahir

Fernández Márquez Braulio Israel

Silva Palmas Angélica Araceli

Índice

Introducción

Bienvenidos al Manual Técnico de **Portal Web para Cedula de Mercados**. Este documento ha sido creado con el propósito de proporcionar a los usuarios, desarrolladores y profesionales de soporte una guía completa y detallada para comprender, implementar y mantener **Portal Web para Cedula de Mercados**.

En un mundo cada vez más impulsado por la tecnología, la comprensión y el dominio de sistemas y productos tecnológicos son esenciales para el éxito y la eficiencia en una amplia gama de industrias. Este manual está diseñado para ser su recurso de referencia confiable, ofreciendo información sobre todas las características, funciones y aspectos técnicos de **Portal Web para Cedula de Mercados**.

Archivos

Archivo Index

El archivo index.js cumple un papel central como punto de entrada y configuración principal de la aplicación. Aquí se encuentra la lógica inicial para establecer y configurar la aplicación Express, lo que lo convierte en una parte esencial de la configuración de la API.

Las líneas que comienzan con ‘server.use’ están configurando middlewares en una aplicación Express. El middleware en Express son funciones que se ejecutan en el proceso de solicitud y respuesta antes de que lleguen a su ruta final. Aquí está la explicación de cada una de estas líneas:

- **server.use(morgan("dev"))**: Este middleware utiliza el paquete ‘morgan’ para registrar información de solicitud (como el método HTTP, la URL y el código de estado) en la consola del servidor. “dev” es un formato de registro predefinido que proporciona información útil para el desarrollo.
- **server.use(cors())**: Este middleware utiliza el paquete ‘cors’ para habilitar el intercambio de recursos entre dominios cruzados (CORS). Permite que el servidor responda a solicitudes desde diferentes dominios o servidores.
- **server.use(taskRoutes)**: Aquí se utiliza un middleware personalizado llamado taskRoutes. Este middleware es un enrutador de Express que se define en otro archivo ("./routes/routes"). Este enrutador maneja las rutas específicas de la aplicación y las asociará con las funciones de controlador correspondientes.
- **server.use(body.json())**: Este middleware utiliza el paquete ‘body-parser’ para analizar el cuerpo de las solicitudes entrantes en formato JSON. Esto permite acceder a los datos enviados en el cuerpo de la solicitud a través de ‘req.body’ en las rutas y controladores posteriores.

Finalmente, después de configurar todos estos middlewares, la aplicación Express comienza a escuchar en un puerto específico (4000 en este caso) utilizando **server.listen(PortServer, ...)**. Cuando alguien hace una solicitud a este servidor,

las solicitudes pasarán a través de los middlewares configurados antes de llegar a las rutas y controladores específicos definidos en **taskRoutes**.

Archivo routes

El archivo de rutas (routes.js) define todas las rutas disponibles la API y asocia cada una de ellas con un controlador específico que se encargará de manejar las solicitudes entrantes y las respuestas correspondientes. Es una parte esencial de la estructura de una aplicación Express.js, ya que organiza la lógica de manejo de solicitudes en rutas y controladores separados para mantener el código más limpio y modular. Aquí está la explicación de lo que hace cada parte del archivo:

Importación de módulos y controladores:

Se importan varios módulos y controladores necesarios para definir las rutas y manejar las solicitudes entrantes.

Configuración del router:

Se crea un router Express utilizando ‘const router = Router();’. Este router se utiliza para definir las rutas y asociar controladores a esas rutas.

Uso de body-parser:

Se utiliza el middleware ‘body-parser’ para analizar el cuerpo de las solicitudes entrantes en formato JSON. Esto permite que las rutas accedan a los datos enviados en el cuerpo de la solicitud a través de ‘req.body’.

Definición de rutas y asociación de controladores:

A continuación, se definen varias rutas y se asocian con los controladores correspondientes utilizando métodos como ‘router.get()’, ‘router.post()’, ‘router.put()’, y ‘router.delete()’. Cada ruta tiene un controlador que manejará las solicitudes entrantes a esa ruta específica.

Exportación del router:

Finalmente, el router se exporta al final del archivo utilizando ‘module.exports = router;’. Esto permite que otras partes de la aplicación, como el archivo index.js, importen y utilicen este router para manejar las rutas definidas aquí.

ENDPOINTS

Endpoint Loginlogic

Este endpoint maneja la lógica de inicio de sesión y autenticación de usuarios en una API. Realiza una consulta a la base de datos para verificar si el usuario proporcionado existe y, en función de eso, responde con el tipo de usuario o un mensaje de error. Aquí están las funciones y sus respectivas descripciones:

Loginlogic:

- Esta es una función asíncrona que toma dos parámetros, ‘req’ y ‘res’, que representan la solicitud HTTP entrante y la respuesta HTTP saliente, respectivamente.
- Extrae el ‘username’ y la ‘password’ del cuerpo de la solicitud ‘(req.body)’, asumiendo que se espera un objeto JSON con estos campos.
- Realiza una consulta a una base de datos utilizando la biblioteca ‘pool’ (es una instancia de conexión a la base de datos) para buscar un usuario que coincida con el nombre de usuario y la contraseña proporcionados en la solicitud.
- Llama a la función ‘returnPathIfUserExist’ para determinar si el usuario existe en la base de datos y devuelve su tipo de usuario.
- Responde a la solicitud con un código de ‘status 200’ (éxito) y devuelve el usuario encontrado en formato JSON como respuesta. Si el usuario no existe, responde con un código de ‘status 204’ (sin contenido) y devuelve un objeto JSON que indica un error.

returnPathIfUserExist:

- Esta función toma un parámetro ‘user’, que parece ser el resultado de la consulta a la base de datos.
- Llama a la función ‘evalIfUserExists’ para verificar si el usuario existe en función del número de filas devueltas por la consulta.

- Si el usuario existe, devuelve el tipo de usuario del primer registro encontrado en el resultado de la consulta.
- Si el usuario no existe, lanza un error con el mensaje "User does not exist".

evalIfUserExists:

- Esta función toma un parámetro ‘user’, que aparentemente es el resultado de la consulta a la base de datos.
- Comprueba si el número de filas en el resultado ‘(user.rowCount)’ es mayor que cero ‘(notExist)’. Si es así, devuelve ‘true’, lo que indica que el usuario existe. De lo contrario, devuelve ‘false’.

Exportación:

Al final del archivo, se exporta la función ‘Loginlogic’ para que pueda ser importada y utilizada en otros lugares de la aplicación.

Endpoint findByNameOrId

El endpoint findByNameOrId permite realizar búsquedas en la base de datos en función de diversos filtros y parámetros proporcionados en la solicitud. A continuación, se proporciona una explicación más detallada:

findByNameOrId:

- Esta función toma dos parámetros: ‘req’ (la solicitud HTTP entrante) y ‘res’ (la respuesta HTTP que se enviará).
- Inicializa algunas variables locales, como ‘limitedConsult’ y ‘resultOfConsult’.
- Verifica si se proporcionan filtros en la solicitud a través de ‘req.query’. Los filtros se esperan en forma de parámetros de consulta (por ejemplo: ‘?filtrarPor=terceraEdad&mostrar=permanentes’ en la URL).
- Si se proporcionan filtros, se construye una consulta SQL con la función ‘getConsult’ utilizando los filtros y se limita a un número específico de

resultados. Luego, se ejecuta la consulta a la base de datos con 'queryToDatabaseWithFilters'.

- Si no se proporcionan filtros, se realiza una consulta SQL sin filtros y se limita a un número específico de resultados. Luego, se ejecuta la consulta a la base de datos con 'queryToDatabaseWithOutFilters'.
- Finalmente, se responde con un código de 'status' 200 (éxito) y los resultados de la consulta en formato JSON.

getConsult(filters, parametersArray):

- Esta función es la función principal para construir la consulta SQL. Toma dos parámetros: 'filters' y 'parametersArray'.
- 'Filters' es un objeto que contiene los filtros proporcionados en la solicitud. Puede incluir opciones como 'filtrarPor', 'mostrar', 'isNameOrId', etc.
- 'parametersArray' es un objeto que se utiliza para almacenar los valores de los parámetros de consulta que se utilizarán en la consulta SQL.
- En función de los filtros proporcionados, esta función construye la parte principal de la consulta SQL y luego agrega filtros adicionales utilizando las funciones 'consultWithFilters' y 'getOptions'. Finalmente, devuelve la consulta SQL completa.

isNameOrId(filters, parametersArray):

- Esta función verifica si se proporciona un filtro 'isNameOrId' en los filtros y, si es así, construye una parte de la consulta SQL basada en el valor de 'isNameOrId'.
- Si 'isNameOrId' es un número (lo que implica que se busca por ID), la función establece un parámetro en 'parametersArray' llamado 'nameOrId' y devuelve una consulta SQL que busca registros con un 'id_comercio' igual al valor proporcionado.
- Si 'isNameOrId' no es un número, la función asume que se está buscando por nombre. Se formatea el valor de 'nameOrId' con caracteres comodín '%'

para realizar una búsqueda de coincidencia parcial (ilike) en el nombre de comercio.

consultWithFilters(filters):

- Esta función se utiliza para agregar filtros adicionales a la consulta SQL en función de las opciones seleccionadas en los filtros de la solicitud.
- Por ejemplo, si el filtro mostrar es "permanentes", se agrega una condición a la consulta que busca registros con ‘tipo_permiso’ igual a "PERMANENTE".
- Si el filtro mostrar es "eventuales", se agrega una condición que busca registros con ‘tipo_permiso’ igual a "EVENTUAL".
- La función devuelve la parte de la consulta SQL que corresponde a los filtros aplicados.

getOptions(filters):

- Esta función se utiliza para agregar opciones de filtrado específicas a la consulta SQL en función de las opciones seleccionadas en los filtros de la solicitud.
- Por ejemplo, si el filtro ‘filtrarPor’ es "terceraEdad", se agrega una condición que busca registros con ‘tercera_edad’ establecido como true.
- Si el filtro ‘filtrarPor’ es "refrendados", se agrega una condición que busca registros con ‘fecha_termino’ igual o posterior a la fecha actual.
- La función devuelve la parte de la consulta SQL que corresponde a las opciones de filtrado aplicadas.

queryToDatabaseWithFilters(limitedConsult, parametersArray):

- Esta función se utiliza para ejecutar una consulta SQL en la base de datos cuando se proporcionan filtros en la solicitud.
- Toma dos parámetros:
    - **limitedConsult**: Esta es la consulta SQL que se va a ejecutar en la base de datos. Puede contener parámetros de consulta que deben reemplazarse con valores concretos.

- **parametersArray**: Un objeto que contiene los valores concretos que se utilizarán para reemplazar los parámetros de consulta en 'limitedConsult'.

- La función verifica si hay parámetros en 'parametersArray'. Si existen parámetros, significa que la consulta SQL contiene parámetros de consulta y deben ser reemplazados. La función ejecuta la consulta SQL con los parámetros y devuelve los resultados de la base de datos.
- Si no hay parámetros en 'parametersArray', la función ejecuta la consulta SQL sin ningún parámetro de consulta y devuelve los resultados de la base de datos.

queryToDatabaseWithOutFilters(limitedConsult):

- Esta función se utiliza para ejecutar una consulta SQL en la base de datos cuando no se proporcionan filtros en la solicitud.
- Toma un parámetro:
  - **limitedConsult**: Esta es la consulta SQL que se va a ejecutar en la base de datos. Puede ser una consulta que no requiere parámetros de consulta.
- La función ejecuta la consulta SQL tal como está, sin ningún parámetro de consulta, y devuelve los resultados de la base de datos.

Exportación de las funciones:

Al final del archivo, se exportan la función 'findByNameOrId' para que puedan ser utilizadas en otras partes de la aplicación.

Endpoint editCostumer

Este endpoint maneja una solicitud HTTP que realiza varias actualizaciones en una base de datos en función de los datos proporcionados en el cuerpo de la solicitud. Aquí está la explicación de este endpoint y las funciones asociadas:

editCostumer:

- Esta función toma dos objetos JSON del cuerpo de la solicitud: ‘comercio’ y ‘comerciante’. Estos objetos contienen datos para actualizar tanto el comercio como el comerciante en la base de datos.
- La función intenta realizar una serie de actualizaciones en la base de datos llamando a diferentes funciones de actualización, como ‘updateCostumer’, ‘updateAdressCostumer’, ‘updateTelefonoCostumer’, ‘updateComercio’, y ‘updateAdressComercio’.
- Después de realizar todas las actualizaciones, responde con un código de ‘status 200’ (éxito) y un mensaje JSON que indica que tanto el comerciante como el comercio se actualizaron exitosamente.
- Si se produce un error durante cualquiera de las actualizaciones, se captura en el bloque ‘catch’ y se responde con un código de ‘status 500’ (error interno del servidor) y un mensaje de error.

updateCostumer(comerciante):

- Esta función se encarga de actualizar la información personal del comerciante en la base de datos.
- Esta función toma un objeto comerciante como parámetro, que contiene información sobre el comerciante a actualizar.
- La función construye una consulta SQL de actualización que actualiza los datos personales del comerciante en la tabla comerciantes de la base de datos.
- Los campos que se actualizan son: ‘apellido_paterno’, ‘apellido_materno’, ‘nombres’, ‘observaciones_comerciante’, y ‘tercera_edad’.
- La actualización se realiza utilizando el ‘id_comerciante’ proporcionado en el objeto comerciante.

updateAdressCostumer(comerciante):

- Esta función se encarga de actualizar la dirección del comerciante en la base de datos.

- Esta función también toma un objeto ‘comerciante’ como parámetro, que contiene información sobre la dirección del comerciante.
- Construye una consulta SQL de actualización que actualiza la dirección del comerciante en la tabla ‘direcciones_comerciantes’ de la base de datos.
- Los campos que se actualizan son: ‘colonia’, ‘calle’, ‘numero_exterior’, ‘numero_interior’, ‘codigo_postal’, y ‘municipio’.
- La actualización se realiza utilizando el ‘id_comerciante’ proporcionado en el objeto comerciante.

updateTelefonoCostumer(comerciante):

- Esta función toma un objeto ‘comerciante’ como parámetro, que contiene información sobre los teléfonos del comerciante.
- Esta función toma un objeto ‘comerciante’ como parámetro, que contiene información sobre los teléfonos del comerciante.
- Itera a través de los teléfonos proporcionados en el objeto ‘comerciante’.
- Para cada teléfono, llama a la función ‘foreachTelefonos’ para realizar la actualización individual de ese teléfono.

foreachTelefonos(id_comerciante, telefono):

- Esta función toma dos parámetros: ‘id_comerciante’ (identificador del comerciante) y ‘telefono’ (información del teléfono a actualizar).
- Construye una consulta SQL de actualización que actualiza el número telefónico en la tabla ‘telefonos’ de la base de datos.
- La actualización se realiza utilizando el ‘id_comerciante’ y ‘id_telefon’ proporcionados en el objeto ‘telefono’.

updateComercio(comercio):

- Esta función se encarga de actualizar la información relacionada con el comercio en la base de datos.
- Construye una consulta SQL de actualización que actualiza los datos del ‘comercio’ en la tabla comercios de la base de datos.

- Los campos que se actualizan son: ‘fecha_inicio’, ‘fecha_termino’, ‘fecha_alta’, ‘giro’, ‘metraje’, ‘horario’, ‘cancelaciones_bajas’, ‘observaciones_comercio’, ‘tipo_comercio_id_tipo_comercio’, ‘tipo_permiso’, y ‘dias_trabajados’.
- La actualización se realiza utilizando el ‘comerciante_id_comerciante’ proporcionado en el objeto comercio.

updateAdressComercio(comercio):

- Esta función toma un objeto comercio como parámetro, que contiene información sobre la dirección del comercio.
- Construye una consulta SQL de actualización que actualiza la dirección del comercio en la tabla direcciones_comercios de la base de datos.
- Los campos que se actualizan son: zona, calle, calle_colindante_uno, calle_colindante_dos, y colonia.
- La actualización se realiza utilizando el comercio_id_comercio proporcionado en el objeto comercio.

Endpoint deleteCostumer

Este endpoint permite eliminar un comerciante de la base de datos, pero solo si no está asociado a ningún comercio. A continuación, se proporciona una explicación más detallada:

Función deleteCostumer:

- Esta función maneja una solicitud HTTP de eliminación y toma un parámetro del cuerpo de la solicitud llamado ‘id_comerciante’, que identifica al comerciante que se va a eliminar.
- Comienza verificando si el comerciante tiene un comercio asociado utilizando la función ‘checkComercioAsociado’.

- Si el comerciante tiene un comercio asociado, se devuelve una respuesta de error con un código de ‘status 400’ y un mensaje que indica que no se puede eliminar el comerciante mientras tenga un comercio asociado.
- Si el comerciante no tiene un comercio asociado, se procede a eliminar al comerciante llamando a la función ‘dropCostumer’.
- Después de eliminar con éxito al comerciante, se responde con un código de ‘status 200’ y un mensaje que indica que el comerciante se ha eliminado correctamente.
- Si ocurre algún error durante el proceso, se captura en el bloque ‘catch’ y se responde con un código de ‘status 500’ (error interno del servidor) y un mensaje de error.

Función checkComercioAsociado:

- Esta función toma el ‘id_comerciante’ como parámetro y se utiliza para verificar si el comerciante tiene un comercio asociado en la base de datos.
- Realiza una consulta SQL que busca comercios con el ‘id_comerciante’ proporcionado en la tabla ‘comercios’.
- Si encuentra comercios asociados, devuelve ‘true’; de lo contrario, devuelve ‘false’.

Función dropCostumer:

- Esta función toma el ‘id_comerciante’ como parámetro y se utiliza para eliminar al comerciante de la base de datos.
- Ejecuta una consulta SQL de eliminación que elimina al comerciante de la tabla ‘comerciantes’ utilizando el ‘id_comerciante’ proporcionado.

Endpoint deleteComercio

Este endpoint se utiliza para eliminar un comercio de la base de datos. A continuación, se proporciona una explicación más detallada:

Función deleteComercio:

- Esta función maneja una solicitud HTTP de eliminación y toma un parámetro del cuerpo de la solicitud llamado 'comerciante_id_comerciante', que identifica al comercio que se va a eliminar.
- Llama a la función 'dropComercio' para realizar la eliminación del comercio.
- Después de eliminar con éxito el comercio, responde con un código de 'status 200' y un mensaje que indica que el comercio se ha eliminado correctamente.
- Si ocurre algún error durante el proceso, se captura en el bloque 'catch' y se responde con un código de 'status 500' (error interno del servidor) y un mensaje de error.

Función dropComercio:

- Esta función toma el 'comerciante_id_comerciante' como parámetro y se utiliza para eliminar un comercio de la base de datos.
- Ejecuta una consulta SQL de eliminación que busca el comercio en la tabla 'comercios' utilizando el 'comerciante_id_comerciante' proporcionado en el cuerpo de la solicitud y lo elimina.

Endpoint CrearNuevaOrdenDePago

Este endpoint se utiliza para crear una nueva orden de pago. A continuación, se proporciona una explicación más detallada:

Función CrearNuevaOrdenDePago:

- Esta función maneja una solicitud HTTP de creación y toma varios parámetros del cuerpo de la solicitud, incluyendo 'conceptoOrden', 'MontoOrdenTotal', 'idComercio', 'fechaInicio', 'fechaFin' y 'dias'.

Realiza varias operaciones para crear una nueva orden de pago en la base de datos:

- Llama a una función interna ‘internCreateNewPayOrder’ para obtener el folio de la orden de pago.
- Si el concepto de la orden no es "PESOS", actualiza la fecha de inicio, fecha final y días en la tabla de ordenes de pago.
- Consulta la referencia de la orden de pago recién creada.
- Recopila información sobre el comerciante y el comercio asociado en la base de datos.
- Formatea los datos necesarios para crear registros en una base de datos externa (UD_ORDEN_PAGO_ENC y UD_ORDEN_PAGO_DET).

Finalmente, responde con un código de ‘status 200’ y un objeto JSON que contiene la referencia de la orden de pago y el id del comercio asociado.

Si ocurre algún error durante el proceso, se captura en el bloque ‘catch’ y se responde con un código de ‘status 500’ (error interno del servidor) y un mensaje de error.

Función internCreateNewPayOrder:

- Esta función toma el ‘idComercio’, el concepto de la orden y el monto total como parámetros y se utiliza para obtener el folio de la orden de pago recién creada.
- Ejecuta una consulta SQL que llama a una función almacenada (‘crearNuevaOrdenPago’) en la base de datos, pasando los parámetros proporcionados.
- Devuelve el folio de la orden de pago obtenido de la consulta.

Creación de registros en base de datos externa:

- Se utilizan varias consultas SQL para insertar registros en una base de datos externa (‘UD_ORDEN_PAGO_ENC’ y ‘UD_ORDEN_PAGO_DET’) con información relacionada con la orden de pago, el comerciante y el comercio.
- Se formatean los datos necesarios y se crean sentencias INSERT para insertar los registros en la base de datos externa.

- Se utiliza una conexión externa (‘ExternBD’) para ejecutar estas consultas en la base de datos externa.

Endpoint getConceptosPago

Este endpoint se utiliza para obtener una lista de conceptos de pago. A continuación, se proporciona una explicación más detallada:

Función getConceptosPago:

- Esta función maneja una solicitud HTTP de consulta (GET) para obtener una lista de conceptos de pago.
- Realiza una consulta a la base de datos utilizando la función ‘pool.query’ para recuperar los conceptos de pago almacenados en la tabla ‘conceptos_pago’.
- Formatea los datos obtenidos en un formato adecuado para su presentación en el frontend, creando un array de objetos JSON que contiene la información de cada concepto.
- Responde con un código de ‘status 200’ y un objeto JSON que contiene la lista de conceptos de pago formateados.
- Si ocurre algún error durante la consulta, se captura en el bloque ‘catch’ y se responde con un código de ‘status 500’ (error interno del servidor) y un mensaje de error.

Formateo de conceptos de pago:

- Los conceptos de pago obtenidos de la base de datos se formatean en un formato que facilita su presentación en el frontend. Cada concepto se representa como un objeto con dos propiedades: ‘value’ y ‘label’.
- ‘value’ contiene toda la información del concepto de pago, incluyendo su identificador, nombre, importe, código y unidad.
- ‘label’ contiene el nombre del concepto de pago, que se mostrará en el frontend como una opción seleccionable en un elemento de lista desplegable.

Endpoint getComercianteAndcomercioInfo

Este endpoint se utiliza para obtener información detallada sobre un comerciante. A continuación, se proporciona una explicación más detallada:

Función getComercianteAndcomercioInfo:

- Esta función maneja una solicitud HTTP de consulta (GET) para obtener información sobre un comerciante y su comercio asociado.
- Recibe el número de folio como parámetro de consulta (‘folio’) desde la solicitud, que se utiliza para identificar el comercio específico que se desea consultar.
- Realiza una consulta compleja a la base de datos utilizando la función ‘pool.query’. La consulta involucra varias tablas: ‘comercios’, ‘comerciantes’, ‘direcciones_comercios’ y ‘direcciones_comerciantes’.
- Combina la información de estas tablas para obtener detalles sobre el comerciante y el comercio asociado al folio proporcionado.
- También llama a la función ‘getComerciantePhones’ para obtener la información de los teléfonos asociados al comerciante.
- Formatea los datos obtenidos en un objeto JSON que contiene la información del comerciante y su comercio, incluyendo detalles de dirección y teléfonos.
- Responde con un código de ‘status 200’ y un objeto JSON que contiene la información formateada.

Función getComerciantePhones:

- Esta función se utiliza para obtener la información de los teléfonos asociados a un comerciante en particular.
- Recibe el número de folio como parámetro (‘folio’), que se utiliza para identificar al comerciante cuyos teléfonos se desean consultar.
- Realiza una consulta a la base de datos para obtener los teléfonos relacionados con el comerciante a través de su identificador.
- Retorna un array de objetos JSON que contiene la información de los teléfonos asociados al comerciante.

Endpoint getTerceraEdadViewData

Este endpoint se utiliza para obtener información específica de un comerciante que puede estar relacionado con la tercera edad y su comercio asociado. A continuación, se proporciona una explicación más detallada:

Función getTerceraEdadViewData:

- Esta función maneja una solicitud HTTP de consulta (GET) para obtener datos relacionados con un comerciante y su comercio asociado.
- Recibe el 'id_comerciante' como parámetro de consulta desde la solicitud, que se utiliza para identificar al comerciante cuyos datos se desean consultar.
- Realiza una consulta a la base de datos utilizando la función pool.query. La consulta involucra varias tablas: 'comerciantes', 'comercios' y 'direcciones_comercios'.

La consulta se estructura para obtener información específica del comerciante y su comercio asociado. Los datos incluyen:

- El nombre completo del comerciante.
- La dirección del comercio, que incluye la calle y la colonia.
- Los nombres de las calles colindantes al comercio.
- La colonia del comercio.
- El giro del comercio.
- El horario de operación del comercio.
- El metraje del comercio.
- La fecha de terminación del comercio.
- Formatea los datos obtenidos en un objeto JSON que contiene la información del comerciante y su comercio.

Responde con un código de 'status 200' y un objeto JSON que contiene la información formateada.

Endpoint getInfoOrdenPago

Este endpoint se utiliza para obtener información detallada sobre una orden de pago específica identificada por su referencia. A continuación, se proporciona una explicación más detallada:

Función getInfoOrdenPago:

- Esta función maneja una solicitud HTTP de consulta (GET) para obtener datos relacionados con una orden de pago específica.
- Recibe el parámetro de consulta 'ref' desde la solicitud, que se utiliza para identificar la orden de pago por su referencia.
- Realiza una consulta a la base de datos utilizando la función 'pool.query'. La consulta se estructura para obtener información detallada de la orden de pago y los comerciantes y comercios asociados.
- La consulta involucra varias tablas, incluyendo 'comerciantes', 'comercios', 'direcciones_comerciantes', 'direcciones_comercios', 'ordenes_pago', 'conceptos_pago' y 'telefonos'.

Los datos que se recuperan incluyen:

- El nombre completo del comerciante.
- La dirección del comerciante, que incluye calle, número exterior, colonia y municipio.
- Detalles completos de la orden de pago, como monto, fecha de inicio y finalización, entre otros.
- Detalles del concepto de pago, como concepto, importe y código.
- El tipo de comercio.
- El giro del comercio.
- El metraje del comercio.
- El horario de operación del comercio.
- La fecha actual.
- La dirección del comercio, incluyendo las calles colindantes y la colonia.
- El número telefónico del comerciante.

Formatea los datos obtenidos en un objeto JSON que contiene la información detallada de la orden de pago y los comerciantes y comercios relacionados.

Responde con un código de ‘status 200’ y un objeto JSON que contiene la información formateada.

Endpoint refrendar

Este endpoint realiza varias operaciones relacionadas con el refrendo de una orden de pago, dependiendo del tipo de concepto de pago y otras condiciones. A continuación, se proporciona una explicación más detallada:

Función refrendar:

- Esta función maneja una solicitud HTTP de consulta (GET) para realizar un refrendo en una orden de pago especificada por su número de referencia.
- Obtiene el número de referencia de la orden de pago desde el parámetro de consulta ‘numeroReferencia’ en la solicitud.
- Realiza una consulta a la base de datos utilizando la función ‘pool.query’. La consulta recupera información de la orden de pago y el concepto de pago relacionado.
- Verifica si la referencia es válida llamando a la función ‘checkIfRefIsValid’. Si no se encuentra la orden de pago o si ya ha sido utilizada (marcada como pagada), se lanzará un error.
- Llama a la función ‘checkConcepto’ para determinar cómo manejar el refrendo en función del tipo de concepto de pago.
- La función devuelve un mensaje relacionado con el resultado del refrendo y lo envía como respuesta.

Función checkIfRefIsValid:

- Esta función verifica si la referencia proporcionada es válida.

- Comprueba si la consulta a la base de datos no devolvió resultados (orden de pago no encontrada) o si la orden de pago ya ha sido marcada como pagada (campo ‘pagado’ en la tabla ‘ordenes_pago’).
- Si se encuentra un problema, lanza un error correspondiente.

Función checkConcepto:

- Esta función determina cómo manejar el refrendo según el tipo de concepto de pago.
- Recibe la orden de pago como parámetro y llama a diferentes funciones según el tipo de concepto.
- Devuelve un mensaje relacionado con el resultado del refrendo.

Función ordenDePagoCedula:

- Maneja el refrendo de una orden de pago con un concepto específico (concepto 32).
- Conecta a una base de datos externa y realiza una consulta para verificar si la orden de pago se ha pagado (campo ‘CONCLUIDO’).
- Verifica si el comercio asociado a la orden de pago ha sido refrendado llamando a ‘checkIfComercioHasRefrendo’.
- Actualiza la orden de pago a "pagada".
- Crea una nueva entrada en la tabla ‘cedula’.
- Devuelve un mensaje con el folio de la cédula.

Función checkIfComercioHasRefrendo:

Comprueba si el comercio asociado a la orden de pago ha sido refrendado comparando las fechas de término del comercio y la fecha actual.

Función ordenDePagoUsoDePiso:

- Maneja el refrendo de una orden de pago con un concepto específico (concepto de uso de piso).
- Realiza una consulta en una base de datos externa para verificar si la orden de pago se ha pagado.

- Actualiza el comercio asociado al refrendo llamando a ‘refrendarComercio’.
- Marca la orden de pago como "pagada".
- Devuelve un mensaje indicando que el comerciante ha sido refrendado.

Función refrendarComercio:

Actualiza la información del comercio, incluyendo la fecha de inicio, fecha de término y días trabajados, basándose en la información de la orden de pago.

Función ordenDePagoOtros:

- Maneja el refrendo de una orden de pago con otros tipos de concepto de pago.
- Simplemente marca la orden de pago como "pagada" y devuelve un mensaje.

Función setPaymentOrderToPaid:

Marca una orden de pago como "pagada" en la base de datos.

Función createNewCedula:

Crea una nueva entrada en la tabla ‘cedula’ para registrar el refrendo de un comercio. La cédula es un documento que certifica el refrendo de un comercio.

Función getFolio:

Obtiene el número de folio de la última cédula registrada en la base de datos.

Endpoint getCedulaData

Este endpoint se utiliza para obtener datos específicos relacionados con una cédula comercial en función de su número de folio. A continuación, se proporciona una explicación más detallada:

Función getCedulaData:

- Esta función maneja una solicitud HTTP de consulta (GET) para obtener datos de una cédula comercial a partir de su número de folio.

- Obtiene el número de folio de la cédula desde el parámetro de consulta ‘folio’ en la solicitud.
- Realiza una consulta a la base de datos utilizando la función ‘pool.query’. La consulta selecciona datos específicos de la cédula y relaciona información de comercios y comerciantes asociados.
- Los datos seleccionados incluyen el nombre completo del comerciante, el tipo de comercio, la fecha de término de la cédula, el número de folio, el giro del comercio, el tipo de permiso, la dirección del comercio, la zona, el metraje, el horario, las observaciones del comercio y la fecha de expedición de la cédula.
- La función devuelve los resultados de la consulta en formato JSON como respuesta HTTP con ‘status 200’.

Endpoint getDashoardinformation

Este endpoint se utiliza para obtener información estadística relacionada con el panel de control de un sistema o aplicación. A continuación, se proporciona una explicación más detallada:

Función getDashboardInformation:

- Esta función maneja una solicitud HTTP de consulta (GET) para obtener datos estadísticos del panel de control.
- Realiza múltiples consultas a la base de datos utilizando la función ‘pool.query’ para recopilar datos relevantes.
- Los datos estadísticos incluyen información como el total de usuarios registrados, el total de comercios registrados, el total de comerciantes registrados y el total de ventas realizadas hoy.
- Los resultados de estas consultas se almacenan en un objeto JSON llamado ‘data’.

Consulta queryDayGraph:

- Esta consulta se utiliza para obtener datos sobre las ventas diarias de la semana actual.
- Genera una serie de fechas para los últimos 7 días, excluyendo los domingos y sábados.
- Calcula la suma de las ventas realizadas en cada día de la semana.
- Los resultados se almacenan en el arreglo ‘dayGraph’ dentro del objeto ‘data’.

Consulta queryWeekGraph:

- Esta consulta se utiliza para obtener datos sobre las ventas semanales de las últimas 4 semanas.
- Genera una serie de fechas para las semanas anteriores y calcula la suma de las ventas en cada semana.
- Los resultados se almacenan en el arreglo ‘weekGraph’ dentro del objeto ‘data’.

Consulta queryMonthGraph:

- Esta consulta se utiliza para obtener datos sobre las ventas mensuales de los últimos 6 meses.
- Genera una serie de fechas para los meses anteriores y calcula la suma de las ventas en cada mes.
- Los resultados se almacenan en el arreglo ‘monthGraph’ dentro del objeto ‘data’.

Respuesta HTTP:

Una vez que se han recopilado todos los datos estadísticos, la función devuelve una respuesta HTTP con ‘status 200’ que incluye el objeto JSON data. Esta respuesta proporciona a los usuarios del panel de control una visión general de la actividad reciente y pasada del sistema.

Endpoint getIncome

Este endpoint se utiliza para obtener información sobre los ingresos generados en un sistema o aplicación. A continuación, se proporciona una explicación más detallada:

Función getIncome:

- Esta función maneja una solicitud HTTP de consulta (GET) para obtener datos de ingresos.
- Realiza una consulta a la base de datos utilizando la función 'pool.query' para recopilar información sobre los ingresos en diferentes intervalos de tiempo.
- Los datos de ingresos incluyen el total pagado en el año actual, el total pagado en el mes actual, el total pagado en la semana actual y el total pagado hoy.

Consulta SQL (query):

- La consulta SQL utiliza funciones de agregación condicional para calcular los totales de ingresos en diferentes intervalos de tiempo, como el año actual, el mes actual, la semana actual y el día actual.
- La condición 'pagado = true' se utiliza para considerar solo las órdenes de pago que han sido pagadas.

Obtención de datos adicionales:

- La función realiza una solicitud HTTP a otro endpoint ('http://${process.env.API_HOST}:4000/admin/getDashboardInformation') para obtener información adicional relacionada con el panel de control del sistema. Esto incluye datos como el total de usuarios, comercios, comerciantes y ventas hoy.
- Los resultados de esta solicitud se almacenan en la variable 'dashboardInfo'.

Respuesta HTTP:

Una vez que se han recopilado los datos de ingresos y la información del panel de control, la función devuelve una respuesta HTTP con ‘status’ 200 que incluye un objeto JSON llamado data. Este objeto contiene tanto la información de ingresos como la información del panel de control, lo que permite a los usuarios obtener una visión completa de la actividad y el rendimiento del sistema.

Endpoint checkIfCommerceIsAvailableToLeave

Este endpoint se utiliza para verificar si un comercio está disponible para darse de baja en un sistema o aplicación. A continuación, se proporciona una explicación más detallada:

Parámetro id_comercio:

Recibe el parámetro ‘id_comercio’ de la solicitud HTTP. Este parámetro es esencial ya que se utiliza para identificar el comercio que se está verificando.

Obtención de Fechas y Folio:

Utiliza dos funciones auxiliares para obtener información importante sobre el comercio:

- **getFechas(id_comercio):** Esta función realiza una consulta a la base de datos para obtener las fechas de inicio y término del comercio, así como la fecha actual.
- **getFolio(id_comercio, fechaInicio, fechaTermino):** Verifica si el comercio tiene una cédula de operación asociada durante el período definido por las fechas de inicio y término.

Comprobaciones:

Una vez que se obtiene la información necesaria, la función realiza las siguientes comprobaciones:

- Verifica si el comercio existe en la base de datos. Si no existe, devuelve una respuesta de error 404 con el mensaje "El comercio no existe".

- Comprueba si la fecha de término del comercio es anterior a la fecha actual. Si es así, significa que el comercio no ha sido refrendado y, por lo tanto, no puede darse de baja. Devuelve una respuesta de error 400 con el mensaje "El comercio no tiene refrendo".
- Luego, verifica si el comercio tiene una cédula de operación asociada. Si no la tiene, devuelve una respuesta de error 400 con el mensaje "El comercio tiene refrendo pero no cédula".
- Si todas las comprobaciones son exitosas, significa que el comercio cumple con los requisitos para darse de baja y devuelve una respuesta de éxito (código de 'status 200') con el mensaje "El comercio puede darse de baja".

Funciones auxiliares

- **getInfoToSend({id_comercio}):** Esta función se utiliza para obtener información adicional sobre el comercio que se podría utilizar al procesar una solicitud de baja. Devuelve un objeto con detalles como el nombre del comerciante, el giro del comercio y la dirección.
- **getFolio(id_comercio, fechaInicio, fechaTermino):** Verifica si el comercio tiene una cédula de operación asociada durante el período definido por las fechas de inicio y término. Si encuentra un folio asociado, lo devuelve; de lo contrario, devuelve 'null'.

Respuestas HTTP:

La función devuelve respuestas HTTP adecuadas según el resultado de las comprobaciones. Puede devolver respuestas de éxito (código de 'status 200') o respuestas de error (códigos de 'status 404 y 400') junto con mensajes descriptivos.

Endpoint getFormatoBajaInfo

Este endpoint se utiliza para obtener información específica necesaria para generar un formato de baja de un comercio. A continuación, se proporciona una explicación más detallada:

Parámetro id_comercio:

Recibe el parámetro ‘id_comercio’ de la solicitud HTTP. Este parámetro es esencial ya que se utiliza para identificar el comercio del cual se desea obtener información para el formato de baja.

Obtención de Información Principal:

Utiliza la función ‘getInfoToSend({id_comercio})’ para obtener información principal sobre el comercio, como el nombre del comerciante, el giro del comercio y la dirección. Esta información se almacena en la variable ‘infoToSend’.

Obtención de Fechas y Folio:

Luego, utiliza las funciones auxiliares para obtener información sobre las fechas relacionadas con el comercio y el folio de la cédula de operación, si está disponible:

- **getFechas(id_comercio):** Obtiene las fechas de inicio y término del comercio y la fecha actual.
- **getFolio(id_comercio, fechaInicio, fechaTermino):** Verifica si el comercio tiene una cédula de operación asociada durante el período definido por las fechas de inicio y término. El folio se almacena en la variable folio.

Añadir Folio a la Información:

Agrega el folio obtenido de la cédula de operación a la información principal almacenada en ‘infoToSend’. Esto permite incluir el folio en el formato de baja que se generará más tarde.

Respuesta HTTP:

Finalmente, responde con un código de ‘status 200’ (éxito) y envía la información completa necesaria para generar el formato de baja en formato JSON. Esta información incluye detalles del comercio (nombre, giro, dirección, etc.) y el folio de la cédula de operación si está disponible

Endpoint baja

Este endpoint se utiliza para realizar la solicitud de baja de un comercio. A continuación, se proporciona una explicación más detallada:

Parámetro folio:

Recibe el parámetro ‘folio’ desde el cuerpo de la solicitud HTTP. Este parámetro generalmente se utiliza para identificar el comercio que se solicita dar de baja.

Proceso de Baja:

Llama a la función ‘queryToDatabase(folio)’ para realizar la operación de baja en la base de datos. Esta función se encarga de actualizar un campo en la tabla de comercios para marcar el comercio como dado de baja. La actualización se basa en el folio proporcionado.

Respuesta HTTP Exitosa:

Si la operación de baja se completa con éxito, responde con un código de ‘status HTTP 200’ (éxito) y envía el mensaje "Baja exitosa".

Manejo de Errores:

En caso de que ocurra algún error durante la operación de baja o en la consulta a la base de datos, se captura la excepción y se responde con un código de ‘status HTTP 500’ (error interno del servidor) junto con un mensaje de error "Error en el registro".

Función queryToDatabase:

Esta función es responsable de realizar la consulta SQL necesaria para marcar un comercio como dado de baja en la base de datos. Aquí está su funcionamiento:

Parámetro folio:

Recibe el parámetro ‘folio’, que se utiliza para identificar el comercio que se solicita dar de baja.

Consulta SQL de Actualización:

Construye una consulta SQL que actualiza el campo ‘cancelaciones_bajas’ en la tabla de comercios, marcando el comercio como dado de baja. La actualización se basa en el folio proporcionado.

Ejecución de la Consulta SQL:

Utiliza la función ‘pool.query’ para ejecutar la consulta SQL en la base de datos.