# Documentación del API

## Índice

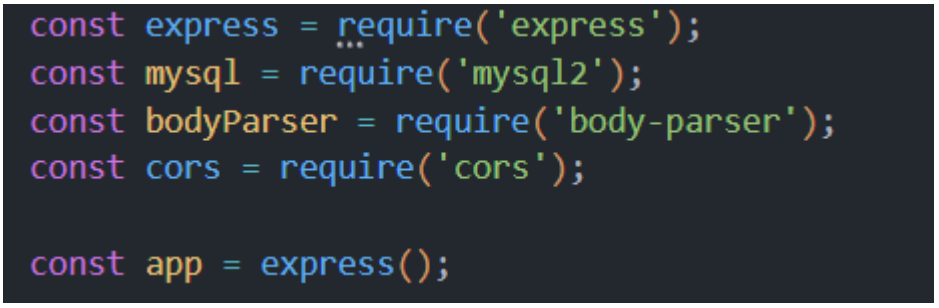
# Introducción

Este documento proporciona una descripción detallada del API desarrollado en Node.js. El API utiliza Express.js para gestionar las rutas y MySQL para la gestión de la base de datos. A continuación, se describen todas las rutas disponibles, sus propósitos, y los detalles técnicos necesarios para su utilización.

# Dependencias

El proyecto requiere las siguientes dependencias:

- `express`: Framework web para Node.js.
- `mysql2`: Cliente MySQL para Node.js.
- `body-parser`: Middleware para parsear el cuerpo de las solicitudes.
- `cors`: Middleware para habilitar CORS.




# Configuración de la conexión MySQL

Se configura la conexión a la base de datos MySQL con los detalles del host, usuario, contraseña y nombre de la base de datos.

```
// Configuración de la conexión MySQL
const connection = mysql.createConnection({
    //host: '192.168.1.58',
    host: '192.168.1.13',
    user: 'admin',
    password: '1234Qwer',
    database: 'mydb',
    connectTimeout: 300000
});

// Conexión a la base de datos
connection.connect(function(err){
    if(err){
        throw err;
    } else {
        console.log("Conexion hecha");
    }
});
```

## Middleware

Se configuran varios middleware para el manejo de las solicitudes y las respuestas:

- `bodyParser.json()`: Para parsear cuerpos de solicitudes JSON.
- `bodyParser.urlencoded({ extended: true })`: Para parsear cuerpos de solicitudes URL encoded.
- `cors()`: Para permitir el acceso a la API desde diferentes dominios.

```
// Middleware para analizar el cuerpo de las solicitudes
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Middleware para habilitar CORS
app.use(cors());
```

## Rutas de la API

### Obtener datos de la tabla 'usuarios'

Ruta: `/api/data`
Método: `GET`
Descripción: Obtiene todos los registros de la tabla `usuarios`.

```
// Definir ruta para obtener datos de la tabla 'menus'
app.get('/api/data', (req, res) => {
    // Consulta SQL para obtener los datos de la tabla 'menus'
    connection.query('SELECT * FROM usuarios', (error, results) => {
        if (error) {
            throw error;
        } else {
            res.send(results);
        }
    });
});
```

## Iniciar sesión

Ruta: `/api/iniciar-sesion`
Método: `POST`
Descripción: Verifica las credenciales del usuario y permite el inicio de sesión.

```
// Definir ruta para iniciar sesión
app.post('/api/iniciar-sesion', (req, res) => {
    const { correo, contrasena } = req.body;

    // Realizar la verificación de las credenciales en la base de datos
    connection.query('SELECT * FROM usuarios WHERE correo = ? AND contrasena = ?',
        [correo, contrasena],
        (error, results) => {
            if (error) {
                res.status(500).json({ error: error.message });
            } else {
                if (results.length > 0) {
                    res.status(200).json({ message: 'Inicio de sesión exitoso', user: results[0] });
                } else {
                    res.status(401).json({ error: 'Credenciales inválidas' });
                }
            }
        }
    );
});
```

## Registrar un nuevo usuario

Ruta: `/api/register`
Método: `POST`
Descripción: Registra un nuevo usuario en la base de datos.

```
// Definir ruta para registrar un nuevo usuario en la base de datos
app.post('/api/register', (req, res) => {
    // Obtener los datos del cuerpo de la solicitud
    const { correo, contrasena, nombre_apellido, rol, numero } = req.body;

    // Realizar la inserción en la base de datos
    connection.query('INSERT INTO usuarios (correo, contrasena, nombre_apellido, rol, numero) VALUES (?, ?, ?, ?, ?)',
        [correo, contrasena, nombre_apellido, rol, numero],
        (error, results) => {
            if (error) {
                res.status(500).json({ error: error.message });
            } else {
                res.status(201).json({ message: 'Usuario registrado exitosamente' });
            }
        }
    );
});
```

## Confirmar contraseña

Ruta: `/api/confirm-password`
Método: `POST`
Descripción: Confirma si la contraseña proporcionada coincide con la registrada para el usuario.

```
// Confirmar contraseña
app.post('/api/confirm-password', (req, res) => {
    const { idUsuario, password } = req.body;

    connection.query('SELECT contrasena FROM usuarios WHERE idUsuario = ?', [idUsuario], (error, results) => {
        if (error) {
            res.status(500).json({ error: error.message });
        } else if (results.length === 0 || results[0].contrasena !== password) {
            res.status(401).json({ success: false, message: 'Contraseña incorrecta' });
        } else {
            res.status(200).json({ success: true, message: 'Contraseña confirmada' });
        }
    });
});
```

## Actualizar datos de usuario

Ruta: `/api/update-user`
Método: `PUT`
Descripción: Actualiza los datos de un usuario en la base de datos.

```
// Definir ruta para actualizar datos de usuario
app.put('/api/update-user', (req, res) => {
    const { idUsuario, correo, contrasena, nombre_apellido, rol, numero, newPassword } = req.body;

    const updatedData = [correo, newPassword || contrasena, nombre_apellido, rol, numero, idUsuario];
    connection.query(
        'UPDATE usuarios SET correo = ?, contrasena = ?, nombre_apellido = ?, rol = ?, numero = ? WHERE idUsuario = ?',
        updatedData,
        (updateError, updateResults) => {
            if (updateError) {
                res.status(500).json({ error: updateError.message });
            } else {
                res.status(200).json({ message: 'Datos actualizados correctamente' });
            }
        }
    );
});
```

## Publicar un nuevo comentario

Ruta: `/api/comentarios`
Método: `POST`
Descripción: Publica un nuevo comentario en la base de datos.

```
// Definir ruta para publicar un nuevo comentario
app.post('/api/comentarios', (req, res) => {
    // Obtener los datos del cuerpo de la solicitud
    const { titulo, comentario, estrellas, usuarioId } = req.body;

    // Verificar si el usuario está autenticado
    if (!usuarioId) {
        return res.status(401).json({ error: 'Usuario no autenticado' });
    }

    // Realizar la inserción en la base de datos
    connection.query(
        'INSERT INTO comentarios (titulo, comentario, estrellas, Usuarios_idUsuario) VALUES (?, ?, ?, ?)',
        [titulo, comentario, estrellas, usuarioId],
        (error, results) => {
            if (error) {
                res.status(500).json({ error: error.message });
            } else {
                res.status(201).json({ message: 'Comentario publicado exitosamente' });
            }
        }
    );
});
```

## Obtener todos los comentarios

Ruta: `/api/comentarios`
Método: `GET`
Descripción: Obtiene todos los comentarios registrados en la base de datos.

```
// Definir ruta para obtener todos los comentarios
app.get('/api/comentarios', (req, res) => {
    connection.query('SELECT * FROM comentarios', (error, results) => {
        if (error) {
            res.status(500).json({ error: error.message });
        } else {
            res.json(results);
        }
    });
});
```

## Registrar una nueva reserva

Ruta: `/api/reservar` Método: `POST` Descripción: Registra una nueva reserva en la base de datos.

```
// Definir ruta para registrar una nueva reserva
app.post('/api/reservar', (req, res) => {
    const { numPersonas, fecha_hora, numMesa, usuarioId } = req.body;

    // Verificar si el usuario está autenticado
    if (!usuarioId) {
        return res.status(401).json({ error: 'Usuario no autenticado' });
    }

    // Realizar la inserción en la base de datos
    connection.query(
        'INSERT INTO reservas (numPersonas, fecha_hora, numMesa, Usuarios_idUsuario) VALUES (?, ?, ?, ?)',
        [numPersonas, fecha_hora, numMesa, usuarioId],
        (error, results) => {
            if (error) {
                res.status(500).json({ error: error.message });
            } else {
                res.status(201).json({ message: 'Reserva registrada exitosamente' });
            }
        }
    );
});
```

## Obtener las reservas del usuario

Ruta: `/api/reservas`
Método: `GET`
Descripción: Obtiene las reservas del usuario autenticado.

```
// Definir ruta para obtener las reservas del usuario
app.get('/api/reservas', (req, res) => {
    // Verificar si el usuario está autenticado
    const usuarioId = req.query.usuarioId;
    if (!usuarioId) {
        return res.status(401).json({ error: 'Usuario no autenticado' });
    }

    // Consulta SQL para obtener las reservas del usuario
    const currentDate = new Date().toISOString().slice(0, 19).replace('T', ' ');
    connection.query('SELECT * FROM reservas WHERE Usuarios_idUsuario = ? AND fecha_hora > ? ORDER BY fecha_hora DESC', [usuarioId, currentDate], (error, results) => {
        if (error) {
            res.status(500).json({ error: error.message });
        } else {
            res.json(results);
        }
    });
});
```

## Eliminar una reserva

Ruta: `/api/reservas/:idReserva`
Método: `DELETE`
Descripción: Elimina una reserva especificada por su ID.

```
app.delete('/api/reservas/:idReserva', (req, res) => {
    const idReserva = req.params.idReserva;

    connection.query('DELETE FROM reservas WHERE idReserva = ?', [idReserva], (error, results) => {
      if (error) {
        res.status(500).json({ error: error.message });
      } else {
        res.status(200).json({ message: 'Reserva eliminada exitosamente' });
      }
    });
  });
```

## Actualizar una reserva

Ruta: `/api/reservas/:idReserva`
Método: `PUT`
Descripción: Actualiza los datos de una reserva específica.

```
// Ruta para actualizar una reserva
app.put('/api/reservas/:idReserva', (req, res) => {
  const idReserva = req.params.idReserva;
  const { numPersonas, fecha_hora, numMesa } = req.body;

  connection.query(
    'UPDATE reservas SET numPersonas = ?, fecha_hora = ?, numMesa = ? WHERE idReserva = ?',
    [numPersonas, fecha_hora, numMesa, idReserva],
    (error, results) => {
      if (error) {
        res.status(500).json({ error: error.message });
      } else {
        res.status(200).json({ message: 'Reserva actualizada exitosamente' });
      }
    }
  );
});
```

## Obtener todos los menús

Ruta: `/api/menus`
Método: `GET`
Descripción: Obtiene todos los registros de la tabla `menus`.

```
// Definir ruta para obtener datos de la tabla 'menus'
app.get('/api/menus', (req, res) => {
    // Consulta SQL para obtener los datos de la tabla 'menus'
    connection.query('SELECT * FROM menus', (error, results) => {
        if (error) {
            res.status(500).json({ error: 'Error al obtener las publicaciones' });
        } else {
            res.json(results);
        }
    });
});
```

## Obtener un menú específico y sus platos

Ruta: `/api/menu/:idMenus`
Método: `GET`
Descripción: Obtiene los datos de un menú específico y sus platos asociados.

```
// Definir ruta para obtener datos de un menú específico y sus platos
app.get('/api/menu/:idMenus', (req, res) => {
    const menuId = req.params.idMenus;
    // Consulta SQL para obtener los datos del menú y sus platos asociados
    const query = `
        SELECT m.idMenus, m.nombre AS nombreMenu, m.precio, m.descripcion AS descripcionMenu, m.img, m.categorias,
               p.idPlato, p.nombre AS nombrePlato, p.descripcion AS descripcionPlato, p.precio AS precioPlato
        FROM menus m
        LEFT JOIN platos p ON m.idMenus = p.idMenus
        WHERE m.idMenus = ?
    `;
    connection.query(query, [menuId], (error, results) => {
        if (error) {
            res.status(500).send({ error: 'Error al obtener el menú' });
        } else {
            // Organizar los resultados para tener un objeto de menú con su lista de platos asociados
            const menu = {
                idMenus: results[0].idMenus,
                nombre: results[0].nombreMenu,
                precio: results[0].precio,
                descripcion: results[0].descripcionMenu,
                img: results[0].img,
                cate[illegible]ias,
                plat (property) idPlato: any
                    idPlato: row.idPlato,
                    nombre: row.nombrePlato,
                    descripcion: row.descripcionPlato,
                    precio: row.precioPlato
                }))
            };
            res.json(menu);
        }
    });
});
```

## Crear un nuevo menú

Ruta: `/api/menus`
Método: `POST`
Descripción: Crea un nuevo menú en la base de datos.

```
// Ruta para crear un nuevo menú
app.post('/api/menus', (req, res) => {
    const { nombre, precio, descripcion, img, categoria } = req.body; // Obtener los datos del cuerpo de la solicitud

    // Consulta SQL para insertar un nuevo menú con la ruta de la imagen
    //const query = 'INSERT INTO menus (nombre, precio, descripcion, img, categoria) VALUES (?, ?, ?, ?, ?)';

    // Ejecutar la consulta SQL con los datos proporcionados
    connection.query(
        'INSERT INTO menus (nombre, precio, descripcion, img, categoria) VALUES (?, ?, ?, ?, ?)',
        [nombre, precio, descripcion, img, categoria],
        (error, results) => {
        if (error) {
            res.status(500).json({ error: error.message });
        } else {
            res.status(201).json({message:'Publicacion insertada correctamente'});
        }
    });
});
```

## Obtener reservas filtradas

Ruta: `/api/reservas`
Método: `GET`

Descripción: Obtiene las reservas filtradas por usuario y/o fecha, solo accesible para administradores.

```
app.get('/api/reservas', isAdmin, (req, res) => {
  // Verificar si se proporcionó un ID de usuario y una fecha en los parámetros de la consulta
  const usuarioId = req.query.usuarioId;
  const fecha = req.query.fecha;

  // Verificar si se proporcionó al menos un parámetro de filtro
  if (!usuarioId && !fecha) {
      return res.status(400).json({ error: 'Se requiere al menos un parámetro de filtro (usuarioId o fecha)' });
  }

  // Consulta SQL base para obtener reservas
  let query = 'SELECT * FROM reservas';

  // Parámetros para la consulta SQL
  const queryParams = [];

  // Agregar condiciones de filtrado según los parámetros proporcionados
  if (usuarioId) {
      query += ' WHERE Usuarios_idUsuario = ?';
      queryParams.push(usuarioId);
  }

  if (fecha) {
      if (queryParams.length > 0) {
          query += ' AND';
      } else {
          query += ' WHERE';
      }
      query += ' fecha_hora >= ?';
      queryParams.push(fecha);
  }

  // Ordenar las reservas por fecha_hora en orden descendente
  query += ' ORDER BY fecha_hora DESC';

  // Ejecutar la consulta SQL con los parámetros
  connection.query(query, queryParams, (error, results) => {
      if (error) {
          res.status(500).json({ error: error.message });
      } else {
          res.json(results);
      }
  });
);
```

## Crear una nueva publicación

Ruta: `/api/publicaciones`
Método: `POST`
Descripción: Crea una nueva publicación en la base de datos.

```
// Definir ruta para crear una nueva publicación
app.post('/api/publicaciones', (req, res) => {
    const { titulo, publicacion, usuarioId } = req.body;
    // Verificar si el usuario está autenticado
    if (!usuarioId) {
        return res.status(401).json({ error: 'Usuario no autenticado' });
    }
    // Consulta SQL para insertar una nueva publicación
    //const query = 'INSERT INTO publicaciones (titulo, publicacion, Usuarios_idUsuario) VALUES (?, ?, ?)';

    // Ejecutar la consulta SQL con los datos proporcionados
    connection.query(
        'INSERT INTO publicaciones (titulo, publicacion, Usuarios_idUsuario) VALUES (?, ?, ?)',
        [titulo, publicacion, usuarioId],
        (error, results) => {
        if (error) {
            res.status(500).json({ error: error.message});
        } else {
            res.status(201).json({message: 'Publicación insertada correctamente'});
        }
    });
});
```

**Obtener todas las publicaciones**

Ruta: `/api/publicaciones`
Método: `GET`
Descripción: Obtiene todas las publicaciones de la base de datos.

```
// Definir ruta para obtener todas las publicaciones
app.get('/api/publicaciones', (req, res) => {
    // Consulta SQL para obtener todas las publicaciones
    connection.query('SELECT * FROM publicaciones', (error, results) => {
        if (error) {
            res.status(500).json({ error: 'Error al obtener las publicaciones' });
        } else {
            res.json(results);
        }
    });
});
```

# Ejecución del Servidor

El servidor se inicia en el puerto 3000, permitiendo que la API esté disponible para recibir solicitudes.

```
const port = 3000;
app.listen(port, () => {
    console.log(`Servidor escuchando en el puerto ${port}`);
});
```

## Conclusión

Este documento proporciona una guía completa para entender y utilizar el API desarrollado. Cada endpoint está diseñado para interactuar con la base de datos MySQL y realizar diversas operaciones necesarias para el funcionamiento de la aplicación. Para cualquier consulta o problema, se recomienda revisar los mensajes de error devueltos por el servidor y los logs generados durante su ejecución.