Request Logger Middleware Documentation

Overview

This documentation explains the functionality of a custom middleware for an Express.js application. The middleware is responsible for logging incoming requests and outgoing responses, saving them to both a local file and optionally a database. Additionally, the middleware interacts with an external API to save specific data about the requests.

Table of Contents

Key Features

- **Logs both request and response data:** Headers, body, and status codes for incoming requests and outgoing responses are logged.
- **Stores logs in files:** Logs are written to a `logs/` directory, with each day's logs stored in a separate file.
- **Optional database storage:** Commented code for logging data to a MongoDB collection (`RequestLogs`).
- **External API integration:** Logs API call data to an external system via a POST request to a predefined URL.
- **Excludes specific routes:** Requests made to `/logs/` are excluded from logging to avoid clutter.

Setup and Configuration

Dependencies

Ensure the following dependencies are installed in your project:

```
npm install express fs path dotenv axios
```

- `express`: Web framework to create APIs.
- `fs`: File system module to handle file operations.
- `path`: Module for working with file and directory paths.

- `dotenv`: Module for managing environment variables.
- `axios`: HTTP client for making requests to external APIs.

Configuration

Make sure to have a `.env` file with the following environment variable:

```
IS_READONLY=0 # Set to 1 to disable logging
```

Middleware Explanation

Request Logging

The `requestLogger` middleware function is responsible for logging requests, headers, and body data. It creates log files in the `logs/` directory, naming each log file by the current date (in ISO format), and logs the following details:

- **Request Method**: HTTP method of the request (e.g., `GET`, `POST`).
- **Request URL**: The endpoint the client is accessing.
- **Request Headers**: Includes user agent and other headers sent by the client.
- **Request Body**: If present, logs the JSON body of the request.
- **IP Address & User Agent**: Extracts IP and user agent information from the request.

```
// Log request headers and body (if available)
const headers = JSON.stringify(req.headers);
const requestBody = JSON.stringify(req.body);
```

Skipping Logs for Specific Routes

If the request URL contains `/logs/`, the middleware skips logging to avoid recursive logging of the log route.

```
if (req.url.includes('/logs/')) {
  next();
  return;
}
```

Response Interception and Logging

The middleware intercepts the response before it's sent to the client to capture the status code and the body. It overrides the default `res.send` method to log this data:

```
const originalSend = res.send;
res.send = function (body) {
  const responseBody = JSON.stringify(body);
  responseStatusCode = res.statusCode;
  originalSend.call(this, body);
};
```

If logging is enabled (`IS_READONLY=0`), the log file is created (if it doesn't already exist) and populated with the request and response data:

```
const logDir = path.join(__dirname, 'logs');
if (!fs.existsSync(logDir)) {
  fs.mkdirSync(logDir);
}
```

The log includes:

- **Response Status Code**: The HTTP status of the response.
- **Response Body**: The data returned to the client.

Example Log Entry

```
2024-09-18T10:00:00.000Z - POST /api/data
Request Headers: {"host":"localhost","user-agent":"Mozilla"}
Request Body: {"key":"value"}
Response Status Code: 200
Response Body: {"success":true}
IP Address: 192.168.1.100
User Agent: Mozilla/5.0
```

Saving API Call Data

The function `postApiCall` is responsible for logging specific API call data to an external system. It extracts the endpoint from the request URL and sends a POST request to a remote API to store this information.

```
async function postApiCall(url) {
  const endpoint = url.split('?')[0];
  const apiUrl = 'https://system.lightspeedpay.in/api/save.php';
  const endpointData = new URLSearchParams();
  endpointData.append('endpoint', endpoint);

  try {
    const response = await axios.post(apiUrl, endpointData, {
```

```
      headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    });
    console.log(response.data);
  } catch (error) {
    console.error('Error occurred while saving API call data:',
error.message);
  }
}
```

This function is called at the start of each request, ensuring that API call data is stored in a remote system for auditing purposes.

File Structure

```
project-root/
│
├── middleware/
│   └── requestLogger.js
├── models/
│   └── requestLogsModel.js (Optional: for MongoDB integration)
├── logs/
│   └── request_logs_YYYY-MM-DD.txt (Generated log files)
├── .env
└── server.js
```

- `middleware/requestLogger.js`: Contains the logging middleware.
- `models/requestLogsModel.js`: Optional Mongoose model for logging data to MongoDB.
- `logs/`: Directory where log files are generated.
- `.env`: Configuration file for environment variables.

Error Handling

Error handling is implemented for both file I/O operations and external API requests. If logging fails (e.g., due to permission issues or missing directories), errors are logged to the console:

```
fs.appendFile(logFilePath, logData, (err) => {
  if (err) {
    console.error('Error writing to log file:', err);
  }
});
```

Similarly, if the external API request fails, the error message is captured and logged:

```
catch (error) {
  console.error('Error occurred while saving API call data:',
error.message);
}
```

Conclusion

This middleware provides a robust logging mechanism for both requests and responses, offering the flexibility to log data to a file, database, or external API. By customizing it, you can monitor API usage, debug issues, or audit requests in your Express.js application.