Monday, 29 May 2023

Ledger Rest API

Overview

This Ledger Generator module provides a set of functions that enable generating payment lines for a lease based on the provided lease details. It handles different payment frequencies such as weekly, fortnightly, and monthly, and calculates the amount to be paid during each payment period. The generated payment lines can be used for ledger management or payment tracking purposes.

Please note that this documentation provides an overview of the functions and their purposes. For more detailed information about each function and their usage, refer to the inline comments within the code.

Installation

The provided instructions are for cloning a Git repository and running a Node.js project. Here's a breakdown of the steps:

1. Clone the Repository:
    - Open a terminal or command prompt.
    - Change to the desired directory where you want to clone the repository.
    - Run the following command to clone the repository:

      ```
      git clone https://github.com/anjulard/Ledger_Rest_API.git
      ```

2. Install Dependencies:

    - Navigate to the cloned project directory:

      `cd Ledger_Rest_API`

    - Run the following command to install the project dependencies:

      `npm install`

    - This command will read the **package.json** file and install all the required packages.

3. Build and Run the Project:
    - After the dependencies are installed, you can build and start the project:

      `npm start`

    - This command will build and run the project using the configured script in the **package.json** file.

    - The server will start running on port 4000, as indicated in the instructions.

Once the project is successfully built and running, you can access the server at **http://localhost:4000** or the specific endpoints defined in the project.

Note: Make sure you have Node.js installed on your machine before executing these commands.

Code Structure

index.js

The main entry point of the application (**index.js**).

Key Components

- **app**: The Express application instance.
- **Port**: The port number on which the server will listen.

Middleware

- **bodyParser.json()**: Middleware to parse JSON request bodies.

Routes

- **/lease**: The base URL for all lease-related routes. It is handled by the **ledgerRoutes** middleware.

Request Handlers

- **GET /**: The default route that returns a "Hello from Ledger" message.

Server Startup

- **app.listen()**: Starts the server and listens on the specified **PORT**. Prints a message to the console upon successful startup.

routes.js

This router file defines the routes for handling lease-related operations. The routes are associated with their corresponding controller functions. Here's a breakdown of each route:

- **POST /**: Creates a new lease. The **validateInputs** middleware is applied to validate the request inputs before passing it to the **createLease** controller function.
- **GET /**: Retrieves all available leases using the **getLease** controller function.

- **GET /:lease_id**: Retrieves a specific lease by providing the lease ID as a route parameter. The **fetchLease** controller function handles this operation.
- **DELETE /:lease_id**: Deletes a specific lease by providing the lease ID as a route parameter. The **deleteLease** controller function handles this operation.
- **GET /generateLedger/:lease_id**: Generates a ledger for a specific lease by providing the lease ID as a route parameter. The **fetchLedger** controller function handles this operation.

By structuring the routes in this way, you can easily handle different lease operations and delegate the actual logic to the respective controller functions.

Controllers

Provide an overview of the controllers in your application. Explain their role in handling specific HTTP requests and interacting with services to process business logic.

leaseController.js

The purpose of the lease controller is to handle lease-related operations in the Ledger Rest API. Let's break down the functions defined in the controller:

1. **createLease**:
    - This function is responsible for creating a new lease.
    - It first validates the request using **validationResult** to check for any input errors. If there are errors, it returns a response with a 400 status code and an array of errors.
    - If the validation passes, it extracts the lease data from the request body.
    - It generates a new **leaseId** using the **uuidv4** function.
    - The lease data, along with the generated **leaseId**, is pushed into the **leases** array.
    - Finally, it sends a response indicating the successful addition of the lease with the generated **leaseId**.

2. **getLease**:
    - This function retrieves all the available leases.
    - It simply sends the **leases** array as the response, which contains all the leases.
3. **fetchLease**:
    - This function retrieves a specific lease based on the provided **lease_id**.
    - It extracts the **lease_id** from the request parameters.
    - It uses the **find** method on the **leases** array to search for a lease that matches the provided **lease_id**.
    - If a matching lease is found, it sends that lease as the response.
    - If no matching lease is found, it returns an empty response.
4. **deleteLease**:
    - This function deletes a specific lease based on the provided **lease_id**.
    - It extracts the **lease_id** from the request parameters.
    - It uses the **find** method on the **leases** array to search for a lease that matches the provided **lease_id**.
    - If a matching lease is found, it retrieves its index using **indexOf**.
    - It removes the lease from the **leases** array using **splice**.
    - Finally, it sends a response indicating the successful deletion of the lease with the provided **lease_id**.
    -

The **leases** array is exported to make it accessible to other parts of the application, such as other controllers or services that may need access to the lease data.

ledgerController.js

The **fetchLedger** function in the ledger controller is used to retrieve the ledger for a specific lease. Let's break down its usage:

1. It first finds the lease with the provided **lease_id** from the **leases** array using the **find** method.
2. Next, it performs validation on the ledger parameters using the **validateLedger** function. If there are any errors returned from the validation, it sends a response with a 400 status code and an array of errors.
3. If the validation passes, it calls the **generateLedger** function, passing in the necessary parameters: **start_date**, **end_date**, **frequency**, and **weekly_rent** from the lease.

4. The **generateLedger** function calculates the payment lines based on the provided parameters and returns an array of line items.
5. The **prepareResponse** function is then called to format the line items into a desired response format.
6. Finally, the response is sent back to the client using **res.send**.

Overall, the purpose of the **fetchLedger** function is to fetch the ledger information for a specific lease by calling the necessary functions (**validateLedger**, **generateLedger**, and **prepareResponse**) and returning the formatted response.

Dependencies

The **dependencies** listed are the required dependencies for the project. These dependencies are essential for the application to function properly. Here's an explanation of each dependency:

1. **express**: Express is a fast and minimalist web application framework for Node.js. It provides a robust set of features for building web applications and APIs, handling HTTP requests and responses, managing routes, and more.
2. **express-validator**: Express Validator is a set of middleware functions for request validation in Express. It simplifies the process of validating and sanitizing user input data, preventing common security vulnerabilities like cross-site scripting (XSS) and SQL injection.
3. **iso-datestring-validator**: iso-datestring-validator is a package that provides a validator function for ISO 8601 date strings. It can be used to validate and parse date strings in the ISO format, ensuring they are in the correct format and represent valid dates.
4. **moment**: Moment.js is a popular JavaScript library for parsing, validating, manipulating, and formatting dates and times. It provides a simple and powerful API for working with dates, timezones, and durations.
5. **moment-timezone**: Moment Timezone is an extension for Moment.js that adds support for working with timezones. It allows converting dates between different timezones, handling daylight saving time changes, and performing timezone-aware calculations.
6. **uuid**: UUID is a package that generates universally unique identifiers (UUIDs). UUIDs are often used to assign unique identifiers to resources or entities in distributed systems to avoid conflicts.

7. **validate-date**: Validate Date is a package that provides a simple function for validating date strings. It checks if a given string represents a valid date using JavaScript's built-in **Date** object and returns a boolean indicating the validity.

These dependencies are crucial for the functionality and proper operation of the application. They provide necessary features for web development, request validation, date manipulation, and unique identifier generation.

Change log

Version 1.0.0 (2023-05-29)

- Added a new API endpoint for creating a new lease.
- Implemented functionality to automatically generate a unique lease ID for each new lease.
- Enhanced error handling and input validation for the create lease endpoint.
- Implemented API endpoint for fetching lease details using the lease ID.
- Added a new API endpoint for generating a ledger based on a given lease.
- Improved error handling for the fetch lease endpoint to handle invalid or non-existent lease IDs.
- Added support for validating the start_date and end_date parameters in ISO string format.
- Implemented validation for the payment frequency parameter to ensure it is one of the allowed values: WEEKLY, FORTNIGHTLY, or MONTHLY.
- Implemented validation for the weekly_rent parameter to ensure it is a valid number.
- Added validation for the timezone parameter to ensure it is a valid TZ database name.
- Refactored the code to handle errors gracefully and provide appropriate error responses.
- Improved code efficiency and optimized data processing for generating line items.
- Included proper documentation and usage instructions for the ledger API endpoint.