Financial Record Endpoint

Created by Brian Immel, last modified on October 10, 2024

You can use this endpoint to manage your financial records. It offers a complete set of tools for retrieving, creating, updating, and deleting records related to your financial transactions. You can use it to retrieve, create, update, and delete records related to your financial transactions.

Methods and their functions

GET: This method retrieves a list of financial records based on specific criteria. You can filter the results by providing parameters like account ID, date range, and transaction type.

POST: Use this method to create a new financial record, typically representing a transaction. You'll need to provide details such as the account it belongs to, transaction type, amount, date, and an optional description.

PUT: This method allows you to update an existing financial record. You can modify specific fields like the amount, description, or transaction type by including them in the request body.

DELETE: This method permanently removes a designated financial record.

Requirement

To ensure secure access to your financial data, this endpoint requires bearer token authentication. Include your API key in the Authorization header of your requests using the following format:

```
Authorization: Bearer YOUR_API_KEY
```

GET Method

The GET method allows you to retrieve financial records from the database by using the following request parameters: `account_id`, `start_date`, `end_date`, and `transaction_type`.

Endpoint: `/api/v1/get-records`

Request parameters

This table outlines the parameters used for querying financial records.

Parameter	Data Type	Description	Example
`account_id`	string	The ID of the account to query.	"acc12345"
`start_date`	date	The start date of the query range (YYYY-MM-DD).	"2023-01-01"
`end_date`	date	The end date of the query range (YYYY-MM-DD).	"2023-12-31"
`transaction_type`	string	The type of transaction (e.g., "deposit", "withdrawal").	"deposit"

Record retrieval using Node.js GET method

The provided code snippet demonstrates how to retrieve a specific financial record using Node.js and the Axios library. This process involves:

- Including the **Bearer Token**: Pass your token as a parameter to the `getRecord` function.
- Setting the **Authorization Header**: Within the function, add the `Authorization` header to the request configuration using the `headers` property. The header value should follow the format `Bearer <your_token>`.

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

async function getRecord(url, token, params = {}) {
     try {
          const response = await axios.get(url, {
          headers: {
               Authorization: `Bearer ${token}`
          },
          params, // Include optional query parameters here
     });
     console.log(response.data);
     } catch (error) {
          console.error('Error:', error);
     }
}

const url = 'https://your-api-endpoint.com/get-record/123';
const token = 'your_token_here';

// Example usage: Retrieve all records
getRecord('https://your-api-endpoint.com/get-record', 'your_token_here');

// Example usage: Filter by account ID
const accountId = 'acc12345';
getRecord('https://your-api-endpoint.com/get-record', 'your_token_here', { account_id:
accountId });

// Example usage: Filter by date range
const startDate = '2023-12-01';
const endDate = '2023-12-31';
getRecord('https://your-api-endpoint.com/get-record', 'your_token_here', { start_date:
startDate, end_date: endDate });

// Example usage: Filter by transaction type
const transactionType = 'deposit';
getRecord('https://your-api-endpoint.com/get-record', 'your_token_here', {
transaction_type: transactionType });
```

Explanation:

1. **Include the Axios library**: Import the axios library for Node.js.
2. **Define the `getRecord` function**: Create an asynchronous function named getRecord that accepts two parameters: the URL of the record to retrieve and the bearer token for authentication.
3. **Make the GET request**: Use axios.get to send a GET request to the specified URL. Include the bearer token in the Authorization header to authenticate the request.
4. **Handle the response**:
    - If the request is successful, the `response.data` property returns the retrieved financial record data. Log this data to the console using `console.log`.
    - If an error occurs, log the error message using `console.error`.

Addition notes:

Before running the code, replace the placeholders:

- **`https://your-api-endpoint.com/get-record/123`** with your actual API URL and record ID.
- **`your_token_here`** with your actual bearer token.

Record retrieval using cURL GET method

The code snippet provides an alternative approach to retrieving financial records using the cURL command-line tool.

```
curl -X GET -H "Authorization: Bearer your_token_here" -G \
-d "account_id=acc12345" \
-d "start_date=2023-12-01" \
-d "end_date=2023-12-31" \
-d "transaction_type=deposit" \
https://your-api-endpoint.com/get-record
```

Explanation:

- **`-X GET`**: Specifies the HTTP method (GET) for retrieving data.
- **`-H "Authorization: Bearer your_token_here"`**: Sets the Authorization header with your bearer token. Replace your_token_here with your actual token.
- **`-G`**: Indicates that the data should be sent as query parameters.
- **`-d "account_id=acc12345"`**: Sets the account_id query parameter. Replace `acc12345` with your desired value.
- **`-d "start_date=2023-12-01"`**: Sets the `start_date` query parameter (adjust the date as needed).
- **`-d "end_date=2023-12-31"`**: Sets the `end_date` query parameter (adjust the date as needed).
- **`-d "transaction_type=deposit"`**: Sets the `transaction_type` query parameter (adjust the type as needed).
- **`https://your-api-endpoint.com/get-record`**: The URL of the API endpoint.

Additional notes:

- Remember to replace placeholders with your actual values.
- You can use query parameters appended to the URL to filter retrieved records (e.g., by account ID, date range).
- For advanced scenarios, consider using additional cURL options or tools like `jq` to process the response data further.

Successful response for GET method

Upon successful retrieval, the API returns a JSON response with a "success" flag and an array containing the matching financial records.

Example JSON response:

```
{
     "success": true,
     "data": [
          {
               "id": "txn1234",
               "amount": 100.00,
               "transaction_date": "2024-07-03",
               "transaction_type": "deposit",
               "description": "July bonus"
          },
          // ... other transactions
     ]
}
```

POST Method

You can create a new financial record within the database using the POST method by using the following parameters: `account_id`, `transaction_type`, `amount`, `transaction_date`, and `description`.

Endpoint: `/api/v1/create-records`

Request parameters

This table outlines the parameters used for creating a new financial record.

Field	Data Type	Description	Required
`account_id`	string	The ID of the account to associate the record with.	Yes
`transaction_type`	string	The type of transaction (e.g., "deposit", "withdrawal").	Yes
`amount`	decimal	The amount of the transaction.	Yes
`transaction_date`	date	The date of the transaction (YYYY-MM-DD).	Yes
`description`	string	A description of the transaction.	Optional

Create a new record using Node.js

This section provides a Node.js code example demonstrating how to create a new financial record within the API. It utilizes the Axios library for making HTTP requests.

- Including the **Bearer Token**: Pass your token as a parameter to the `createFinancialRecord` function.
- Setting the **Authorization Header**: Within the function, add the Authorization header to the request configuration using the headers property. The header value should follow the format Bearer <your_token>.

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

async function createFinancialRecord(url, token, data) {
     try {
          const response = await axios.post(url, data, {
               headers: {
                    Authorization: `Bearer ${token}`
               }
          });
     console.log(response.data);
     } catch (error) {
          console.error('Error:', error);
     }
}


const url = 'https://your-api-endpoint.com/api/v1/create-record';
const token = 'your_token_here';
const data = {
     "account_id": "acc12345",
     "transaction_type": "deposit",
     "amount": 100.00,
     "transaction_date": "2023-12-25",
     "description": "Christmas bonus"
};

createFinancialRecord(url, token, data);
```

Explanation:

1. **Import the `axios` library**: This library is commonly used for making HTTP requests in Node.js.
2. **Define the `createFinancialRecord` function**: This function takes the URL, bearer token, and data to be sent as parameters.
3. **Make the POST request**: Use `axios.post` to send a POST request to the specified URL.
    - The `data` parameter contains the JSON object representing the financial record to be created.
    - The `headers` object includes the Authorization header with the bearer token.
4. **Handle the response**: If the request is successful, the response data contains the newly created record. Otherwise, an error is logged.

Addition notes:

Before running the code, replace the placeholders:

- **`https://your-api-endpoint.com/api/v1/create-record`** with your actual API URL and record ID.
- **`your_token_here`** with your actual bearer token.

Create a new record using cURL

This section outlines how to use cURL to create a new financial record within the API.

```
curl -X POST -H "Authorization: Bearer your_token_here" -H "Content-Type:
application/json" -d '{ \
     "account_id": "acc12345", \
     "transaction_type": "deposit", \
     "amount": 100.00, \
     "transaction_date": "2023-12-25", \
     "description": "Christmas bonus" \
}' \
https://your-api-endpoint.com/api/v1/create-record
```

Explanation:

- **`-X POST`**: Specifies the HTTP method as POST.
- **`-H "Authorization`**: Bearer your_token_here": Sets the Authorization header with the bearer token. Replace your_token_here with your actual token.
- **`-H "Content-Type`**: application/json": Indicates that the request body is in JSON format.
- **`-d '{ ... }'`**: Sets the request body as a JSON object. Replace the placeholder values with your desired data.
- **`https://your-api-endpoint.com/api/v1/create-record`**: The URL of the API endpoint.

Remember to replace:

- **`your_token_here`** with your actual bearer token.
- **`https://your-api-endpoint.com/api/v1/create-record`** with the correct URL of your API.

Successful response for POST method

Upon successful creation, the API returns a 201 status code and a JSON response containing the newly created record, including its unique ID.

Example JSON response:

```
{
     "id": "txn123",
     "account_id": "acc12345",
     "transaction_type": "deposit",
     "amount": 100.00,
     "transaction_date": "2023-12-25",
     "description": "Christmas bonus"
}
```

PUT Method

The PUT method allows you to modify existing financial records within the database. You can update specific fields like the amount, description, or transaction type while leaving other fields unchanged.

Endpoint: `/api/v1/update-records/{record_id}`

Request parameters

This table outlines the optional parameters that can be included in the request body when updating a financial record.

Note: Although all fields are optional, you must specify at least one field.

Field	Data Type	Description	Required
`account_id`	string	The ID of the account to associate the record with.	Optional
`transaction_type`	string	The type of transaction (e.g., "deposit", "withdrawal").	Optional
`amount`	decimal	The amount of the transaction.	Optional
`transaction_date`	date	The date of the transaction (YYYY-MM-DD).	Optional
`description`	string	A description of the transaction.	Optional

Update a record using Node.js

This code snippet showcases a Node.js function named `updateRecord` that utilizes Axios to modify an existing financial record within the database.

Node.js and the Axios library. This process involves:

- Including the **Bearer Token**: Pass your token as a parameter to the `updateRecord` function.
- Setting the **Authorization Header**: Within the function, add the Authorization header to the request configuration using the headers property. The header value should follow the format Bearer <your_token>.

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

async function updateRecord(url, token, data) {
     try {
          const response = await axios.put(url, data, {
               headers: {
                    Authorization: `Bearer ${token}`
               }
          });
          console.log(response.data);
     } catch (error) {
          console.error('Error:', error);
     }
}

const url = 'https://your-api-endpoint.com/update-record/123';
const token = 'your_token_here';
const data = {
     // Update data here
     "amount": 200.00,
     "description": "Revised transaction"
};

updateRecord(url, token, data);
```

Explanation:

1. **Include the Axios library**: Import the axios library for Node.js.
2. **Define `updateRecord` function**: Takes URL, token, and update data as parameters.
3. **Make PUT request**: Uses `axios.put` with URL, data, and headers.
4. **Set Authorization header**: Includes bearer token for authentication.
5. **Handle response**: Logs the response data or an error.

Additional notes:

Remember to replace the placeholder values (`https://your-api-endpoint.com/update-record/123` and `your_token_here`) with your actual API URL, record ID, and bearer token before running this code.

Update a record using cURL

This section demonstrates how to update an existing financial record within the API using cURL.

```
curl -X PUT -H "Authorization: Bearer your_token_here" -H "Content-Type:
application/json" -d '{ \
     "amount": 200.00, \
     "description": "Revised transaction" \ }' \
https://your-api-endpoint.com/api/v1/update-records/123
```

Explanation:

- **`-X PUT`**: Specifies the HTTP method as PUT.
- **`-H "Authorization`**: Bearer your_token_here": Sets the Authorization header with the bearer token. Replace your_token_here with your actual token.

- **`-H "Content-Type`**: application/json": Indicates that the request body is in JSON format.
- **`-d '{ ... }'`**: Sets the request body as a JSON object. Replace the placeholder values with your desired update data.
- **`https://your-api-endpoint.com/api/v1/update-records/123`**: The URL of the API endpoint, including the specific record ID you want to update.

Remember to replace:

- **`your_token_here`** with your actual bearer token.
- **`https://your-api-endpoint.com/api/v1/update-records/123`** with the correct URL of your API and the desired record ID.

Additional notes:

- You can update any or all of the fields in the request body.
- For more complex JSON payloads, you might consider using a tool like `jq` to format and manipulate the data.
- Always ensure that your bearer token is kept secure and confidential.

Successful response for POST method

Upon successful update, the API returns a 200 status code and the updated financial record.

Example JSON response:

```
{
     "id": "txn123",
     "account_id": "acc12345",
     "transaction_type": "deposit",
     "amount": 200.00,
     "transaction_date": "2023-12-25",
     "description": "Revised transaction"
}
```

DELETE Method

The DELETE method allows you to permanently remove existing financial record from the database.

Endpoint: `/api/v1/delete-record/{record_id}`

Request parameters

No request body is required for the DELETE method.

Additional notes:

- The `record_id` in the URL path specifies the unique identifier of the record to be deleted.
- The DELETE method is typically idempotent, meaning it can be safely executed multiple times without changing the result.

Delete a record using Node.js

The provided code snippet demonstrateshow to permanently remove a financial record from the database using Node.js and Axios.

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

async function deleteRecord(url, token) {
     try {
          const response = await axios.delete(url, {
               headers: {
                    Authorization: `Bearer ${token}`
               }
          });
          console.log(response.data);
     } catch (error) {
          console.error('Error:', error);
     }
}

const url = 'https://your-api-endpoint.com/delete-record/123';
const token = 'your_token_here';

deleteRecord(url, token);
```

Explanation:

1. **Include the Axios library**: Import the axios library for Node.js.
2. **Define `deleteRecord` function**: Takes URL and token as parameters.
3. **Make DELETE request**: Uses axios.delete with URL and headers.
4. **Set Authorization header**: Includes bearer token for authentication.
5. **Handle response**: Logs the response data or an error.

Addition notes:

Before running the code, replace the placeholders:

- **`https://your-api-endpoint.com/delete-record/123`** with your actual API URL and record ID. **Note**: ensure you have the correct record id before executing this request.
- **`your_token_here`** with your actual bearer token.

Delete a record using cURL

This section outlines how to use cURL to permanently delete a financial record from the database.

```
curl -X DELETE -H "Authorization: Bearer your_token_here" \
https://your-api-endpoint.com/api/v1/delete-record/123
```

Explanation:

- **`-X DELETE`**: Specifies the HTTP method as DELETE.
- **`-H "Authorization: Bearer your_token_here"`**: Sets the Authorization header with the bearer token. Replace your_token_here with your actual token.

- `https://your-api-endpoint.com/api/v1/delete-record/123`: The URL of the API endpoint, including the specific record ID you want to delete.

Remember to replace:

- `your_token_here` with your actual bearer token.
- `https://your-api-endpoint.com/api/v1/delete-record` with the correct URL of your API.

Error Codes

This table lists the potential error codes that may be returned by the API, along with their corresponding descriptions:

Error Code	Description
400	Bad Request (invalid parameters or missing required fields).
401	Unauthorized (authentication failed).
403	Forbidden (access denied).
404	Not Found (record not found).
500	Internal Server Error.