@fin.cx/portablefinance

Documentation for @fin.cx/portablefinance

- readme.md for @fin.cx/portablefinance
- changelog.md for @fin.cx/portablefinance

readme.md for @fin.cx/portablefinance

an interface package for the financeplus organization

Install

To install the `@fin.cx/portablefinance` package, you can use npm or yarn. Make sure you have Node.js installed on your machine.

Using npm:

```
npm install @fin.cx/portablefinance
```

Using yarn:

```
yarn add @fin.cx/portablefinance
```

Usage

```
import {
  AcCsvParser,
  IMonetaryTransaction,
  IPaymentAccount,
  IMonthlyCheckpoint,
  IVoucher,
  ICsvDescriptor
} from '@fin.cx/portablefinance';

// 1. Define a CSV Parser
class MyCsvParser extends AcCsvParser<IMonetaryTransaction> {
  public paymentProviderName = 'mybank';
```

```
  public description = 'Custom CSV Parser for MyBank';
  private transactions: IMonetaryTransaction[] = [];

  public addCsvDecriptor(csvDescriptorArg: ICsvDescriptor): void {
    // Implement the logic to parse the CSV and populate transactions array
    // For simplicity, assume the CSV is correctly formatted and contains the necessary
transaction details.
    this.transactions = [
      {
        id: 'txn1',
        data: {
          paymentAccountId: 'acc1',
          originTransactionId: 'orig1',
          originAccountId: 'origAcc1',
          additionalIds: ['add1', 'add2'],
          date: Date.now(),
          amount: 1000,
          description: 'Payment for services',
          name: 'Service Payment',
        },
      },
    ];
  }

  public async getTransactions(): Promise<IMonetaryTransaction[]> {
    return this.transactions;
  }
}

// 2. Define a Payment Account
const myVoucher: IVoucher = {
  voucherDate: new Date(),
  voucherId: 'vch1',
  voucherStatus: 'uploaded',
  voucherBinaryString: 'binarystringdata',
};

const myAccount: IPaymentAccount = {
  id: 'acc1',
  data: {
```

```
    status: 'active',
    connectionData: {
      bankAdapterType: 'mybank',
      credentials: {
        username: 'user',
        password: 'pass',
      },
    },
    currency: 'USD',
    name: 'MyBankAccount',
    checkpoints: {
      '2021': {
        1: {
          start: new Date('2021-01-01').getTime(),
          end: new Date('2021-01-31').getTime(),
          pdfVoucher: myVoucher,
        },
        2: {
          start: new Date('2021-02-01').getTime(),
          end: new Date('2021-02-28').getTime(),
          pdfVoucher: myVoucher,
        },
      },
    },
  },
};

// 3. Working with Transactions and Accounts
(async () => {
  const csvParser = new MyCsvParser();
  const csvDescriptor: ICsvDescriptor = {
    name: 'transactions.csv',
    contentString: 'dummy content', // Replace with actual CSV content
  };

  csvParser.addCsvDecriptor(csvDescriptor);

  const transactions = await csvParser.getTransactions();
  console.log('Parsed Transactions:', transactions);
```

```
  console.log('Payment Account:', myAccount);
})();
```

In this example, we demonstrate how to:

1. Define a custom CSV parser class (`MyCsvParser`) that extends the `AcCsvParser` abstract class.
2. Implement the `addCsvDecriptor` method to parse CSV content and populate transactions.
3. Define a sample payment account (`myAccount`) with necessary details.
4. Integrate the parser and account information in an asynchronous function to showcase the usage of parsed transactions and account data.

Interfaces Overview

IVoucher Interface

The `IVoucher` interface is used to represent payment vouchers, with the following fields:

- `voucherDate`: Date – The date the voucher was created.
- `voucherId`: string – Unique identifier for the voucher.
- `voucherStatus`: 'uploaded' | 'transmitted' – The status of the voucher.
- `voucherBinaryString`: string – The binary string representation of the voucher content.

IMonetaryTransaction Interface

The `IMonetaryTransaction` interface is used to represent monetary transactions, with the following fields:

- `id`: string – Unique identifier for the transaction.
- `data`: object – Contains key details about the transaction:
  - `paymentAccountId`: string – Identifier for the payment account involved in the transaction.
  - `originTransactionId`: string – Identifier for the original transaction (in case of related transactions).
  - `originAccountId`: string – Identifier for the account from where the transaction originated.
  - `additionalIds`: string[] – Additional identifiers related to the transaction.
  - `date`: number – Date of the transaction (timestamp).
  - `amount`: number – Amount involved in the transaction.
  - `description`: string – Description of the transaction.
  - `name`: string – Name associated with the transaction.
  - Optional properties for additional voucher data and metadata.

IPaymentAccount Interface

The `IPaymentAccount` interface represents a payment account with various properties, organized into nested structures.

- `id`: string – Unique identifier for the payment account.
- `data`: object – Contains the core data for the payment account:
  - `status`: 'active' | 'inactive' | 'deleted' – Status of the account.
  - `connectionData`: object – Holds connection-related data such as type and credentials.
  - `currency`: 'EUR' | 'USD' – Currency of the account.
  - `name`: string – Name of the account.
  - `checkpoints`: object – Monthly checkpoint data structured by year.

ICsvDescriptor Interface

The `ICsvDescriptor` interface describes a CSV file, consisting of:

- `name`: string – Name of the CSV file.
- `contentString`: string – The content of the CSV file as a string.

Abstract Class AcCsvParser

The `AcCsvParser` abstract class serves as a blueprint for creating specific CSV parsers for different payment providers. It includes:

- `paymentProviderName`: Abstract property for the name of the payment provider.
- `description`: Abstract property for a description of the parser.
- `addCsvDecriptor`: Abstract method accepting a `ICsvDescriptor` object to process CSV data.
- `getTransactions`: Abstract method returning a promise which resolves to an array of `IMonetaryTransaction` objects.

Advanced Usage

Beyond the basic usage demonstrated above, the `@fin.cx/portablefinance` package can be utilized for more advanced financial data management scenarios:

Handling Multiple CSV Files

```
// Define multiple CSV descriptors
const csvDescriptors: ICsvDescriptor[] = [
  {
    name: 'transactions_jan.csv',
    contentString: 'dummy content January', // Replace with actual content
  },
```

```
  {
    name: 'transactions_feb.csv',
    contentString: 'dummy content February', // Replace with actual content
  },
];

// Add and parse all CSV files
csvDescriptors.forEach((descriptor) => {
  csvParser.addCsvDecriptor(descriptor);
});

// Retrieve and process all transactions
const allTransactions = await csvParser.getTransactions();
console.log('All Transactions:', allTransactions);
```

Transaction Filtering and Aggregation

```
// Filter transactions by date or amount
const filteredTransactions = allTransactions.filter((transaction) => {
  return transaction.data.amount > 500; // Example: transactions greater than 500
});

console.log('Filtered Transactions:', filteredTransactions);

// Aggregate transaction amounts
const totalAmount = allTransactions.reduce((acc, transaction) => acc +
transaction.data.amount, 0);
console.log('Total Transaction Amount:', totalAmount);
```

Integration with Payment Accounts

```
// Assume we have multiple accounts
const accounts: IPaymentAccount[] = [myAccount]; // Add more accounts as needed

// Link transactions to payment accounts
const transactionsByAccount = accounts.map((account) => {
  return {
    accountId: account.id,
    transactions: allTransactions.filter((txn) => txn.data.paymentAccountId === account.id),
  };
```

```
});


console.log('Transactions by Account:', transactionsByAccount);
```

This comprehensive usage example covers the complete set of features provided by the `@fin.cx/portablefinance` package, demonstrating how to extend the abstract CSV parser, work with interfaces, handle multiple CSV files, filter and aggregate transactions, and integrate them with payment accounts. Feel free to customize the example based on your specific use cases and workflows.

For further information, consult the linked documentation at the top of this README.

Happy coding! □□

changelog.md for @fin.cx/portablefinance

2024-07-05 - 1.0.28 - fix(documentation)

Updated documentation in readme.md

- Fixed and elaborated on the installation steps for npm and yarn.
- Added usage examples and interfaces overview sections.
- Included sections on advanced usage scenarios like handling multiple CSV files, transaction filtering and aggregation.

2024-07-02 - 1.0.27 - fix(core)

Fix npm package availability links and status badges in readme.md

- Updated npm, GitLab, and GitHub links in readme.md.
- Enhanced status badges for clearer visibility.

2024-07-02 - 1.0.26 - fix(core)

Update package.json dependencies and metadata

- Updated devDependencies versions in package.json

2023-11-17 - 1.0.25 - core

Fixes and updates to the core functionalities.

- update

2023-11-16 - 1.0.15 to 1.0.24 - core

Series of fixes and updates to the core functionalities.

- update

2021-04-16 - 1.0.11 to 1.0.15 - core

Series of fixes and updates to the core functionalities.

- update

2019-07-07 - 1.0.8 - core

Fixes and added new features related to CSV parsing.

- add Base class for CSV Parsers

2019-06-24 - 1.0.5 to 1.0.7 - core

Series of fixes and updates to the core functionalities.

- update

2019-05-23 - 1.0.1 to 1.0.4 - core

Initial series of updates and fixes following project launch.

- update