@fin.cx/csvparser

Documentation for @fin.cx/csvparser

readme.md for @fin.cx/csvparser

```
# @fin.cx/csvparser
a csvparser for fin2021

## Install

To install the `@fin.cx/csvparser` module, you can use npm. Simply run the following command:

```bash
npm install @fin.cx/csvparser
```

Usage

The `@fin.cx/csvparser` module provides tools to parse CSV files from various financial institutions such as Commerzbank, Fidor Bank, PayPal, and Spendesk. This helps in extracting and managing financial transaction data efficiently.

Here's a guide to showcase various functionalities along with comprehensive TypeScript code examples.

Getting Started

First, make sure to import the necessary components from the `@fin.cx/csvparser` module.

```
import { CsvSpendesk, CsvPayPal, CsvFidor, CsvCommerzbank } from '@fin.cx/csvparser';
```

Spendesk CSV Parsing

Parsing Spendesk CSV from File

The `CsvSpendesk` class provides methods to parse Spendesk CSV files.
```

```
import { CsvSpendesk } from '@fin.cx/csvparser';

async function parseSpendeskFile(filePath: string) {
  try {
    const spendeskData = await CsvSpendesk.fromFile(filePath);
    const transactions = await spendeskData.getTransactions();
    console.log(transactions);
  } catch (error) {
    console.error('Error parsing Spendesk file:', error);
  }
}

// Usage example
parseSpendeskFile('./path/to/spendesk.csv');
```

Parsing Spendesk CSV from Directory

You might have multiple Spendesk CSV files in a directory. The `fromDir` method helps in parsing all of them.

```
import { CsvSpendesk } from '@fin.cx/csvparser';

async function parseSpendeskDirectory(dirPath: string) {
  try {
    const spendeskData = await CsvSpendesk.fromDir(dirPath);
    const transactions = await spendeskData.getTransactions();
    console.log(transactions);
  } catch (error) {
    console.error('Error parsing Spendesk directory:', error);
  }
}

// Usage example
parseSpendeskDirectory('./path/to/spendesk_dir');
```

PayPal CSV Parsing

The `CsvPayPal` class is used to parse PayPal CSV files.

```
import { CsvPayPal } from '@fin.cx/csvparser';

async function parsePayPalFile(filePath: string) {
  try {
    const payPalData = new CsvPayPal();
    const csvDescriptor = {
      filename: 'paypal.csv',
      contentString: await fs.promises.readFile(filePath, 'utf8')
    }
    payPalData.addCsvDecriptor(csvDescriptor);
    const transactions = await payPalData.getTransactions();
    console.log(transactions);
  } catch (error) {
    console.error('Error parsing PayPal file:', error);
  }
}

// Usage example
parsePayPalFile('./path/to/paypal.csv');
```

Fidor Bank CSV Parsing

The `CsvFidor` class is used to parse Fidor Bank CSV files.

```
import { CsvFidor } from '@fin.cx/csvparser';

async function parseFidorFile(filePath: string) {
  try {
    const fidorData = new CsvFidor();
    const csvDescriptor = {
      filename: 'fidor.csv',
      contentString: await fs.promises.readFile(filePath, 'utf8')
    }
    fidorData.addCsvDecriptor(csvDescriptor);
    const transactions = await fidorData.getTransactions();
    console.log(transactions);
  } catch (error) {
    console.error('Error parsing Fidor file:', error);
  }
```

```
}

// Usage example
parseFidorFile('./path/to/fidor.csv');
```

Commerzbank CSV Parsing

The `CsvCommerzbank` class is used to parse Commerzbank CSV files.

```
import { CsvCommerzbank } from '@fin.cx/csvparser';

async function parseCommerzbankFile(filePath: string) {
  try {
    const commerzbankData = new CsvCommerzbank();
    const csvDescriptor = {
      filename: 'commerzbank.csv',
      contentString: await fs.promises.readFile(filePath, 'utf8')
    }
    commerzbankData.addCsvDecriptor(csvDescriptor);
    const transactions = await commerzbankData.getTransactions();
    console.log(transactions);
  } catch (error) {
    console.error('Error parsing Commerzbank file:', error);
  }
}

// Usage example
parseCommerzbankFile('./path/to/commerzbank.csv');
```

Advanced Usage

Merging Spendesk Data

You may need to merge multiple Spendesk CSV data instances.

```
import { CsvSpendesk } from '@fin.cx/csvparser';

async function mergeSpendeskData(filePaths: string[]) {
```

```
  try {
    const spendeskInstances = await Promise.all(filePaths.map(CsvSpendesk.fromFile));
    const mergedInstance = spendeskInstances.reduce(
      (merged, currentInstance) => merged.concat(currentInstance),
      spendeskInstances[0]
    );
    console.log(await mergedInstance.getTransactions());
  } catch (error) {
    console.error('Error merging Spendesk data:', error);
  }
}

// Usage example
mergeSpendeskData([
  './path/to/spendesk1.csv',
  './path/to/spendesk2.csv'
]);
```

Working with PayPal Linked Transactions

PayPal CSV data contains linked transactions, which you may need to handle specifically.

```
import { CsvPayPal } from '@fin.cx/csvparser';

async function handlePayPalLinkedTransactions(filePath: string) {
  try {
    const payPalData = new CsvPayPal();
    const csvDescriptor = {
      filename: 'paypal.csv',
      contentString: await fs.promises.readFile(filePath, 'utf8')
    }
    payPalData.addCsvDecriptor(csvDescriptor);
    const transactions = await payPalData.getTransactions();

    const linkedTransactions = transactions.filter(tx => tx.linkedTransactionCode);
    console.log(linkedTransactions);
  } catch (error) {
    console.error('Error handling linked transactions:', error);
  }
}
```

```
// Usage example
handlePayPalLinkedTransactions('./path/to/paypal.csv');
```

Parsing Multiple File Types

You may need to parse CSV files from different financial service providers.

```
import { CsvSpendesk, CsvPayPal, CsvFidor, CsvCommerzbank } from '@fin.cx/csvparser';

async function parseMultipleFiles(filePaths: {spendesk: string, paypal: string, fidor: string,
commerzbank: string}) {
  try {
    const spendeskData = await CsvSpendesk.fromFile(filePaths.spendesk);
    const payPalData = new CsvPayPal();
    const csvDescriptorPayPal = {
      filename: 'paypal.csv',
      contentString: await fs.promises.readFile(filePaths.paypal, 'utf8')
    }
    payPalData.addCsvDecriptor(csvDescriptorPayPal);

    const fidorData = new CsvFidor();
    const csvDescriptorFidor = {
      filename: 'fidor.csv',
      contentString: await fs.promises.readFile(filePaths.fidor, 'utf8')
    }
    fidorData.addCsvDecriptor(csvDescriptorFidor);

    const commerzbankData = new CsvCommerzbank();
    const csvDescriptorCommerzbank = {
      filename: 'commerzbank.csv',
      contentString: await fs.promises.readFile(filePaths.commerzbank, 'utf8')
    }
    commerzbankData.addCsvDecriptor(csvDescriptorCommerzbank);

    console.log(await spendeskData.getTransactions());
    console.log(await payPalData.getTransactions());
    console.log(await fidorData.getTransactions());
    console.log(await commerzbankData.getTransactions());
```

```
  } catch (error) {
    console.error('Error parsing multiple files:', error);
  }
}

// Usage example
parseMultipleFiles({
  spendesk: './path/to/spendesk.csv',
  paypal: './path/to/paypal.csv',
  fidor: './path/to/fidor.csv',
  commerzbank: './path/to/commerzbank.csv'
});
```

By covering multiple use cases and providing comprehensive documentation and code samples, this readme should help developers integrate and utilize the `@fin.cx/csvparser` module effectively.

```
undefined
```

changelog.md for @fin.cx/csvparser

2024-07-05 - 1.1.0 - feat(core)

Add support for CI workflows and update gitignore

- Added `.gitea/workflows/default_nottags.yaml` for non-tag workflows.
- Added `.gitea/workflows/default_tags.yaml` for tag workflows.
- Updated `.gitignore` to include standard and custom ignore rules.
- Added `.gitlab-ci.yml` for GitLab CI/CD integration.
- Added `.npmrc` with a custom registry URL.
- Added VSCode configuration files for debugging and settings.
- Added `changelog.md` to document project changes.
- Added `license` file with MIT License.
- Updated scripts in `package.json` for building, testing, and documentation.
- Added test file `test/test.ts`.
- Added core CSV parsing files and structures for Spendesk, PayPal, Fidor, and Commerzbank.

2024-07-02 - 1.0.9 - fix(core)

No changes detected

2023-11-17 - 1.0.8 - Core

Main version update

- Miscellaneous fixes

2023-11-17 - 1.0.7 - Core

Minor fixes

- Core update

2023-11-17 - 1.0.6 - Core

Minor fixes

- Core update

2022-01-03 - 1.0.5 to 1.0.6 - Core

Multiple minor fixes and updates

- Core updates

2022-01-03 - 1.0.4 - Core

Minor fixes

- Core update

2022-01-03 - 1.0.3 - Commerzbank

Commerzbank specific fix

- Fixed Commerzbank to comply with spec