@push.rocks/smartmoney

A library for handling monetary values effectively.

readme.md for @push.rocks/smartmoney

handle monetary values

Install

To start using `@push.rocks/smartmoney`, you'll first need to install it using npm. Run the following command in your terminal:

```
npm install @push.rocks/smartmoney --save
```

This will add `@push.rocks/smartmoney` to your project's dependencies. Now, you're ready to use it in your TypeScript projects.

Usage

The main functionality of `@push.rocks/smartmoney` is to handle monetary values effectively, focusing on common tasks such as parsing European numeric strings which may include commas for decimal points and periods for thousands separators. In financial applications, or any software that deals with monetary input in a European format, `@push.rocks/smartmoney` can be extremely useful.

Basic Example: Parsing European Number Strings

To demonstrate the basic functionality of `@push.rocks/smartmoney`, consider the task of converting a European-formatted string that represents a monetary value into a number that JavaScript can work with.

The core aspect of `@push.rocks/smartmoney` you'll be using for this is `parseEuropeanNumberString` function. Here's how you can use it:

```
import { parseEuropeanNumberString } from '@push.rocks/smartmoney';

const exampleString = '1.234,56'; // A typical European formatted number string
const resultNumber = parseEuropeanNumberString(exampleString);
console.log(resultNumber); // Logs: 1234.56
```

This simple example highlights how to convert a string like `'1.234,56'`, which is common in many European countries, into a floating-point number that JavaScript can recognize and work with (`1234.56`).

Advanced Usage: Integrating with Financial Calculations

Assuming you're building a financial application, handling user inputs in various formats might be a common task. For such scenarios, `@push.rocks/smartmoney` can help parse these inputs accurately before performing any calculations.

Let's consider you're calculating the total value of a portfolio in euros, based on user inputs that could be formatted in European style:

```
import { parseEuropeanNumberString } from '@push.rocks/smartmoney';

// User inputs, possibly fetched from a UI layer
const inputs = ['1.000,50', '2.500,00', '3.000,25'];

// Parsing and summing up the total
const total = inputs.reduce((acc, currentValue) => {
  return acc + parseEuropeanNumberString(currentValue);
}, 0);

console.log(`Total Portfolio Value: €${total.toFixed(2)}`);
// Logs: Total Portfolio Value: €6500.75
```

In this more practical example, `@push.rocks/smartmoney` ensures that each user input is correctly parsed to a float, regardless of its original format, enabling accurate and reliable financial calculations.

Conclusion and Further Applications

While the examples provided here showcase basic usage and a more integrated scenario, `@push.rocks/smartmoney` can be deepened further into any financial or data processing application requiring precise handling of monetary formats, especially those catering to users in European countries.

The clear and straightforward API of `@push.rocks/smartmoney` allows for seamless integration into existing projects, making it easier for developers to focus on the core functionality of their applications without worrying about the nuances of handling monetary values.

As part of your project's evolution, consider contributing to `@push.rocks/smartmoney`'s development or sharing your use cases, helping the library to grow and support even more scenarios beneficial to the developer community.

> Note: The usage examples provided are simplified to demonstrate the capabilities of `@push.rocks/smartmoney`. For further exploration and advanced features, dive into the source code and documentation.

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.