@push.rocks/smartdrive

A module for managing drive data and mount points, supporting both local and cloud storage.

readme.md for @push.rocks/smartdrive

do more with local and cloud drives

Install

To install `@push.rocks/smartdrive`, you need Node.js installed on your system. If you have Node.js installed, you can simply run the following command in your terminal:

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

This will download `@push.rocks/smartdrive` and add it as a dependency to your project's `package.json` file.

Usage

`@push.rocks/smartdrive` offers an easy-to-use interface for interacting with local and cloud drives. Let's explore the capabilities of this module with TypeScript examples. To get the most out of these examples, ensure you are familiar with TypeScript and have it set up in your project.

Getting Started

Before using any functionalities, you need to import the module into your TypeScript file. Here's how you can do that using ES Module syntax:

```
import { SmartDrive } from '@push.rocks/smartdrive';
```

Listing Local Drives

One of the primary features of `@push.rocks/smartdrive` is the ability to list all local drives. This is useful for applications that need to perform operations like reading from or writing to external

drives. Here's how you can list all local drives:

```
import { SmartDrive } from '@push.rocks/smartdrive';

const mySmartDrive = new SmartDrive();

async function listLocalDrives() {
  try {
    const drives = await mySmartDrive.getLocalDriveList();
    console.log('Local Drives:', drives);
  } catch (error) {
    console.error('Error listing local drives:', error);
  }
}

listLocalDrives();
```

This function initializes a `SmartDrive` instance and calls the `getLocalDriveList` method. This method returns a Promise that resolves with an array of drives, each containing detailed information about the drive.

Mounting a Drive

Mounting a drive programmatically can be particularly useful for applications that need to manage storage devices automatically. Below is an example of how to mount a drive by its name:

```
import { SmartDrive } from '@push.rocks/smartdrive';

const mySmartDrive = new SmartDrive();

async function mountDrive(devName: string) {
  try {
    await mySmartDrive.mountDeviceByName(devName);
    console.log(`Drive ${devName} mounted successfully.`);
  } catch (error) {
    console.error(`Error mounting drive ${devName}:`, error);
  }
}

// Replace 'sdb1' with the actual device name you want to mount
```

```
mountDrive('sdb1');
```

This example demonstrates how to mount a drive specified by its device name (e.g., 'sdb1'). Note that mounting drives usually requires administrative privileges, so ensure your application has the necessary permissions to perform this action.

Drive and Mount Point Management

Managing drives and mount points is a complex operation that involves dealing with the file system and ensuring that resources are correctly handled to prevent data loss. The methods provided by `@push.rocks/smartdrive`, such as listing drives and mounting/unmounting them, serve as building blocks for more complex drive management logic.

When designing features around drive management, consider:

- Performing checks to ensure drives are not in use before unmounting.
- Providing clear feedback to users or calling applications about the success/failure of operations.
- Handling edge cases, such as attempting to mount a drive that is already mounted or has no recognizable file system.

By responsibly managing local and cloud drives, applications can safely and efficiently interact with storage devices, enhancing the user's ability to manage their data across diverse storage solutions.

Conclusion

`@push.rocks/smartdrive` offers a powerful and flexible way to interact with local and cloud storage solutions. Through its API, developers can list, mount, and manage drives within their Node.js applications. Properly leveraging these capabilities allows for the creation of robust storage management features, enhancing applications' usability and data handling efficiency.

License and Legal Information

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

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.