readme.md for @push.rocks/smartlifecycle

@push.rocks/smartlifecycle

a module handling lifecycle events

Install

To install `@push.rocks/smartlifecycle`, use npm:

```
npm install @push.rocks/smartlifecycle
```

This will download the package and add it to your project's `node_modules` folder. It will also add an entry to your project's `package.json` file, assuming you have one.

Usage

The `@push.rocks/smartlifecycle` module is designed to assist in handling various lifecycle events within your application, making it easier to manage startup, shutdown, and other lifecycle states cleanly and effectively. While the module contents are minimal as indicated by the current `index.ts` and `smartlifecycle.plugins.ts` files, we will explore a conceptual use case to demonstrate how you could extend and implement lifecycle management in a TypeScript application.

Getting Started

Firstly, ensure that you are importing the module in your TypeScript file:

```
import * as smartLifecycle from '@push.rocks/smartlifecycle';
```

Given the current implementation does not elaborate on specific lifecycle events, let’s conceptualize a scenario where our application needs to perform specific actions on startup and shutdown.

Implementing Lifecycle Management

Consider an application that needs to connect to a database on startup and properly disconnect when shutting down to prevent data loss or corruption. We'll also assume that logging and running some cleanup tasks are part of the requirements.

Define Lifecycle Handlers

First, we'll define our lifecycle handlers:

```
import * as fs from 'fs/promises';
import * as db from 'your-favorite-db-client'; // Placeholder for your actual DB client import

// Startup handler
async function onStart(): Promise<void> {
  console.log('Application is starting...');
  await db.connect();
  console.log('Connected to database.');
}

// Shutdown handler
async function onShutdown(): Promise<void> {
  console.log('Application is shutting down...');
  await db.disconnect();

  // Cleanup tasks
  await fs.unlink('/path/to/temporary/file');
  console.log('Disconnected from database and cleaned up temporary files.');
}

// Usage
onStart().catch(err => console.error('Startup failed:', err));
```

Handling Process Signals

To ensure your application reacts to shutdown signals (e.g., `SIGINT` from pressing Ctrl+C), you can catch these and invoke the `onShutdown` method:

```
process.on('SIGINT', () => {
  onShutdown()
    .then(() => {
      console.log('Graceful shutdown completed.');
      process.exit(0);
    })
    .catch(err => {
      console.error('Failed to shutdown gracefully:', err);
      process.exit(1);
    });
});
```

Extending the Module

While the current implementation of `@push.rocks/smartlifecycle` shown in the source code may not directly expose lifecycle events, you can extend the module by contributing to its Git repository or by wrapping it in your higher-level lifecycle management system within your application.

The conceptual implementation here demonstrates how you might use basic features of the module (as it stands) in conjunction with other application logic to manage your application's lifecycle. Adjust and expand upon these concepts based on your specific use cases and requirements.

Conclusion

`@push.rocks/smartlifecycle` provides a foundational starting point for managing lifecycle events in your TypeScript applications. By following the practices outlined above, you can ensure that your application handles startup and shutdown processes gracefully, maintaining a robust and efficient operational state. As the module evolves, look to its documentation and source code for new features and improvements that can simplify lifecycle management in your projects even further.

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.

Revision #1
Created 2026-03-28 11:40:18 UTC by foss.global Team
Updated 2026-03-28 11:40:18 UTC by foss.global Team