@push.rocks/beautyfiglet

A Node.js module for creating figlet text displays.

- readme.md for @push.rocks/beautyfiglet
- changelog.md for @push.rocks/beautyfiglet

readme.md for @push.rocks/beautyfiglet

A Node.js module that facilitates the creation of ASCII art using figlet with customizable fonts and layouts.

Install

To install `@push.rocks/beautyfiglet`, ensure you have Node.js along with npm set up on your machine. Executing the following command in your terminal will install this module via npm:

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

Alternatively, you can add it as a dependency in your `package.json`:

```
{
  "dependencies": {
    "@push.rocks/beautyfiglet": "^1.0.8"
  }
}
```

Then execute:

```
npm install
```

Usage

`@push.rocks/beautyfiglet` is a comprehensive Node.js package that empowers developers to generate visually engaging figlet-style ASCII art. This section will delve into detailed use-case scenarios, ensuring you have a robust understanding of how to integrate and utilize this module effectively within your projects.

Getting Started

To begin using `@push.rocks/beautyfiglet`, import the `BeautyFiglet` class in your TypeScript file. This class serves as the gateway to generating your ASCII art.

```
import { BeautyFiglet } from '@push.rocks/beautyfiglet';


console.log('Welcome to BeautyFiglet!'); // Outputs: Welcome to BeautyFiglet!
```

Rendering Text with Figlet

The quintessential feature of this module is its ability to convert plain text into ASCII art using popular figlet formats. Below is an example of how to achieve this with `renderDefault`.

```
(async () => {
  const figletText = await BeautyFiglet.renderDefault('Hello, World!');
  console.log(figletText); // Displays "Hello, World!" in ASCII format using the default font.
})();
```

Custom Font Selection

One of the striking features of `BeautyFiglet` is the ability to select from a variety of fonts. This allows for a unique representation of your text.

```
(async () => {
  const customFiglet = await BeautyFiglet.renderText('Beautiful Text', 'Ghost');
  console.log(customFiglet); // Renders the text "Beautiful Text" with the Ghost style.
})();
```

Exploring Available Fonts

Knowing which fonts you can work with enhances your creative freedom. By listing available fonts, you can make informed choices.

```
(async () => {
  const fontsList = await BeautyFiglet.listFonts();
  console.log('Available Fonts:');
```

```
  console.log(fontsList.join(', ')); // Lists all fonts available for use in figlet.
})();
```

Custom Text Layouts

Layouts, which refer to the arrangement of text, can be customized. Modify the horizontal and vertical layouts for dynamic artistry.

```
import figlet from 'figlet';

const renderCustomLayout = (text: string, font: string, hLayout: string, vLayout: string):
Promise<string> => {
  return new Promise((resolve, reject) => {
    figlet.text(text, { font, horizontalLayout: hLayout, verticalLayout: vLayout }, (err,
data) => {
      if (err) reject(`Error: ${err.message}`);
      else resolve(data);
    });
  });
};

// Application Example
(async () => {
  try {
    const customLayoutText = await renderCustomLayout('Creative Layout', 'Ghost', 'full',
'full');
    console.log(customLayoutText); // Renders with specified layout adjustments.
  } catch (error) {
    console.error(error);
  }
})();
```

Synchronous Text Rendering

For scenarios demanding immediate rendering results, such as testing or small scripts, `textSync` is a beneficial synchronous method.

```
import { figlet } from '@push.rocks/beautyfiglet.plugins';

const artwork = figlet.textSync('Synchronicity!', {
  font: 'Standard',
  horizontalLayout: 'default',
  verticalLayout: 'default'
});

console.log(artwork); // Output is rendered immediately without asynchronous behavior.
```

Adding Color to the Output

To create more vibrant and visually appealing text displays, utilize libraries such as `chalk` to incorporate colors.

```
import figlet from 'figlet';
import chalk from 'chalk';

(async () => {
  const colorizedText = await BeautyFiglet.renderText('Color Me Beautiful', 'Standard');
  console.log(chalk.magentaBright(colorizedText)); // Renders ASCII art in bright magenta.
})();
```

Robust Error Handling

Effectively managing potential errors (e.g., when a specified font is unavailable) is essential to prevent application crashes.

```
(async () => {
  try {
    const result = await BeautyFiglet.renderText('Error Test', 'NonExistentFont');
    console.log(result);
  } catch (error) {
    console.error(`Caught an error: ${error}`); // Captures and logs errors without halting
the program.
  }
})();
```

Web Server Integration

The functionality of this module can be extended to serve ASCII art over HTTP, allowing integration with web services. Here’s a demonstration using Express.

```
import express from 'express';
import { BeautyFiglet } from '@push.rocks/beautyfiglet';

const app = express();

app.get('/api/art/:text', async (req, res) => {
  const textToRender = req.params.text;

  try {
    const asciiArt = await BeautyFiglet.renderDefault(textToRender);
    res.send(`<pre>${asciiArt}</pre>`); // Sends the rendered ASCII art in a preformatted text block over HTTP.
  } catch (error) {
    res.status(500).send(`Error: ${error.message}`); // Manages errors with a 500 Internal Server Error status code.
  }
});

app.listen(4000, () => {
  console.log('Server running at http://localhost:4000/');
});
```

Command-Line Interface (CLI)

Creating a CLI tool enhances accessibility, allowing direct ASCII art generation from the terminal. Here's a demonstration:

```
#!/usr/bin/env node
import { BeautyFiglet } from '@push.rocks/beautyfiglet';
import { Command } from 'commander';

const program = new Command();
program.version('1.0.8');
```

```
program
  .option('-t, --text <text>', 'Text to render')
  .option('-f, --font <font>', 'Font for rendering', 'Standard')
  .action(async (cmd) => {
    try {
      const asciiArt = await BeautyFiglet.renderText(cmd.text, cmd.font);
      console.log(asciiArt); // Renders text directly in CLI using specified options.
    } catch (error) {
      console.error(`Error: ${error.message}`);
    }
  });

program.parse(process.argv);
```

CLI Tool Usage:

1. Save the script as `beautyfiglet-cli.ts`.
2. Make it executable:

```
chmod +x ./beautyfiglet-cli.ts
```

3. Establish a link via npm:

```
npm link
```

4. Run the command using:

```
beautyfiglet-cli --text "Hello World" --font "Ghost"
```

This CLI approach demonstrates seamless integration into development workflows, adding simplicity and efficiency for users.

Comprehensive Summary

Throughout this comprehensive guide, we have demonstrated the varied and extensive capabilities of the `@push.rocks/beautyfiglet` module. From initializing the module and employing basic rendering to exploring intricate font customization, layout configurations, and error handling, the guide has covered an exhaustive list of functionalities available within this versatile module.

Developers are encouraged to delve deeper into exploring font selections and engaging with additional configuration settings to thoroughly harness the potential of this tool. By integrating colorful text arrangements and utilizing robust error management, applications can balance both

aesthetic appeal and operational reliability.

This expansive exploration into the `@push.rocks/beautyfiglet` module underscores the creativity and potential this tool offers for implementing ASCII art in your Node.js projects.

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.

changelog.md for @push.rocks/beautyfiglet

2025-01-14 - 1.0.11 - fix(test)

Fix incorrect argument passed to renderDefault in test case

- Corrected the argument list for renderDefault call in test/test.ts

2025-01-14 - 1.0.10 - fix(test)

Add logging for rendered text outputs in test cases

- Added console.log statements for outputs of rendered ASCII art in test cases.
- Ensure the correctness of rendering operations by outputting results during tests.

2025-01-14 - 1.0.9 - fix(npm)

Updated package description and added more keywords for better visibility

2025-01-14 - 1.0.8 - fix(documentation)

Update README to reflect recent changes

- Updated package description for accuracy.
- Adjusted installation instructions for clarity.
- Added a new command-line interface example.
- Enhanced section on error handling with illustrative examples.

- Simplified integration guidelines with web servers using Express.

2025-01-14 - 1.0.7 - fix(core)

Improve error messages in renderText and listFonts methods

- Improved error handling in renderText and listFonts methods
- Ensure proper rejection messages during errors

2025-01-14 - 1.0.6 - fix(core)

Update .gitignore to improve ignored paths

- Ignored various build and cache directories to reduce clutter in the repository.
- Refactored .gitignore to categorize ignored paths for better readability and maintenance.

2025-01-14 - 1.0.5 - fix(package)

Fix scripts section in package.json by adding missing comma

- Corrected the JSON format in the scripts section by adding a missing comma between format and build scripts.

2025-01-14 - 1.0.4 - fix(core)

Removed unnecessary files and updated dependencies

- Removed .gitlab-ci.yml to streamline the CI/CD process
- Updated and fixed TypeScript files to align with ES Module syntax
- Removed obsolete dist files for a cleaner build
- Updated package.json to reflect current project configurations, including dependencies and scripts

2024-05-29 - 1.0.3 - Enhancements

Finalizing updates under version 1.0.3

- Updated project description

2023-07-10 to 2024-05-29 - 1.0.3 - Organizational Changes

Implemented significant organizational changes

- Switched to a new organizational scheme

2018-03-05 - 1.0.2 to 1.0.3 - Maintenance and Format Updates

Streamlined maintenance and formatting

- Updated code formatting for consistency