@design.estate/fontloader

a minimal approach to loading fonts on the fly.

readme.md for @design.estate/fontloader

@designestate/fontloader

a fontloader for managing font loading from A to Z

Install

To install `@designestate/fontloader`, ensure that you have Node.js and npm installed, then run the following command in your project directory:

```
npm install @designestate/fontloader --save
```

This will add `@designestate/fontloader` as a dependency to your project's `package.json` file and download it to your `node_modules` folder.

Usage

The `@designestate/fontloader` package serves as a utility to manage and load fonts, specifically designed to cater to web development needs. It supports a range of fonts including 'Inter', 'MaterialIcons', and 'Courier Prime'. Below, you'll find comprehensive usage examples outlined in TypeScript to integrate and utilize the full potential offered by this font loader.

Getting Started

First, ensure you're importing the `FontLoader` class and the type `TFonts` from the package at the top of your TypeScript file:

```
import { FontLoader, TFonts } from '@designestate/fontloader';
```

Loading Fonts

To load fonts into your project effectively, `FontLoader` provides a singleton pattern ensuring that fonts are loaded once and are accessible throughout the runtime of your application or site. Here's a basic example to get you started:

```
// Define the fonts you want to load
const myFonts: TFonts[] = ['Inter', 'MaterialIcons'];

// Ensure fonts are loaded
FontLoader.ensureFonts(myFonts);

// Later in your application, you might want to explicitly load a specific font
const fontLoaderInstance = new FontLoader();
fontLoaderInstance.loadInter();
```

Note that `ensureFonts` method in this context is provided as a stub to showcase the expected functionality. It implies ensuring or checking that the desired fonts are loaded or making decisions to load them.

Advanced Usage

Depending on your application's complexity and requirements, you might want to extend or customize the behavior of the `FontLoader`. Since the provided code snippets are primarily stubs to guide the implementation, here's an extended example illustrating potential custom functionalities:

```
class AdvancedFontLoader extends FontLoader {
  constructor() {
    super();
    // Custom initialization if required
  }

  public async loadCustomFont(fontName: TFonts) {
    // Implement logic to load a custom font not available by default
    // This could involve fetching the font from an API, loading it from local storage, etc.
    console.log(`Loading custom font: ${fontName}`);
    // Placeholder logic
    return Promise.resolve();
  }
```

```
}

// Usage
const advancedFontLoader = new AdvancedFontLoader();
advancedFontLoader.loadCustomFont('Courier Prime').then(() => {
  console.log('Custom font loaded successfully');
});
```

Best Practices

- **Caching:** Implement caching mechanisms to store loaded fonts, preventing redundant fetches and enhancing performance.
- **Error Handling:** Always include error handling logic, especially when loading fonts from external sources.
- **Cross-Browser Support:** Test font loading across different browsers to ensure consistency in user experience.
- **Font Fallbacks:** Specify font fallbacks in your CSS to maintain usability and design integrity even if a font fails to load.

Conclusion

`@designestate/fontloader` offers a pragmatic foundation for managing fonts in web projects, catering to essential needs such as supporting popular fonts like 'Inter' and 'MaterialIcons'. By following the outlined examples in TypeScript, integrating and leveraging this package should streamline the font management process, allowing developers to focus on building engaging, visually consistent web experiences.

For further customization and advanced scenarios, extending the `FontLoader` class as shown in the advanced usage example provides a flexible pathway to accommodate project-specific requirements.

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.