readme.md for @push.rocks/smartocr

an ocr module using ocrmypdf

Install

To install `@push.rocks/smartocr`, use the following command with npm:

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

This module depends on a few external utilities like `ocrmypdf`, so make sure you have these installed and available in your system's PATH. Consult the `ocrmypdf` documentation for installation instructions suitable for your operating system.

Usage

This module provides a TypeScript interface for OCR processing of PDF documents using `ocrmypdf`, encapsulated in the `SmartOcr` class. Here's how to leverage it in your TypeScript project.

Preparing Your Project

First, ensure your TypeScript configuration is set up to handle ESModule syntax. You will also need to have Node.js and the external `ocrmypdf` tool installed on your system.

Basic Setup

```
import { SmartOcr } from '@push.rocks/smartocr';

async function runOcrOnPdf(pdfFilePath: string): Promise<Buffer> {
  // Initialize the SmartOcr instance
```

```
  const smartOcrInstance = await SmartOcr.createAndInit();

  // Load your PDF file into a Buffer, this can be from a file or even a remote source
  const pdfBuffer = await fs.promises.readFile(pdfFilePath);

  // Process the PDF Buffer through SmartOcr
  const processedBuffer = await smartOcrInstance.processPdfBuffer(pdfBuffer);

  return processedBuffer;
}

// Replace './path/to/your/document.pdf' with the actual path to the PDF document you want to
OCR
const ocredPdfBuffer = await runOcrOnPdf('./path/to/your/document.pdf');

// You can now save this buffer to a file, or use it as needed in your application
await fs.promises.writeFile('./path/to/output/document_ocr.pdf', ocredPdfBuffer);
```

In the example above, we import the `SmartOcr` class and use it to process a PDF by passing a `Buffer` of the PDF file to the `processPdfBuffer` method. The method returns a `Buffer` of the processed PDF which includes a text layer added by OCR.

Advanced Usage

The `SmartOcr` class maintains an internal `smartshell` instance to interface with the `ocrmypdf` command. This setup is abstracted away, ensuring you don't need to manage or understand the underlying shell commands to use OCR functionality in your application.

Handling OCR Result

The result of the `processPdfBuffer` is a `Buffer` that contains the OCR-processed PDF. This buffer can be directly written to a file system or further manipulated in memory, depending on your application's needs.

Error Handling

It's important to handle errors that may arise from reading files or the OCR process. The OCR process depends on the external `ocrmypdf` utility, so errors can occur if the utility encounters unsupported PDF structures or if there are issues with the installation of `ocrmypdf`.

```
try {
  const ocredPdfBuffer = await runOcrOnPdf('./path/to/your/document.pdf');
  await fs.promises.writeFile('./path/to/output/document_ocr.pdf', ocredPdfBuffer);
} catch (error) {
  console.error('Failed to OCR the document:', error);
}
```

Conclusion

The `@push.rocks/smartocr` library simplifies adding OCR capabilities to your TypeScript applications by abstracting away the complexity of interfacing with `ocrmypdf`. With minimal setup, you can start processing PDF documents to add searchable text layers, making this library a valuable tool for any project that requires OCR functionality.

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.

Revision #3
Created 2026-03-28 11:11:36 UTC by foss.global Team
Updated 2026-03-28 12:18:27 UTC by foss.global Team