OCR in Node with Tesseract.js

I've always wanted to do Optical Character Recognition (OCR) in Node.js but the alternatives always seemed too complicated for simple command line and browser use.

Recently I came accross Tesseract.js, a Javascript port of the Tesseract C++ library.

Tesseract (any version) will take an image as input and output any text that it finds as part of the image.

This post presents an example of how to use Tesseract.js to do OCR.

The first step is to install Tesseract.js.

```
npm install tesseract.js
```

Next, we need to import the packages we need. There are packages that are native to Node and the Tessearct.js package.

```
// Import node packages
import path from 'path';
import fs from 'path'import crypto from 'crypto';

'crypto'// Import methods from Tesseract
import {
  createWorker,
} from 'tesseract.js';
```

Take the content of the the third argument in the argv array as the name of the file to process

The image location is either the value of imagePath or a string with the path to a default image

```
const imagePath = process.argv[2];

const image = path.resolve(
    imagePath || './BrOarL6IMAA8tud.png',
);
```

Create a constant holding the hashed value of the path to the image file using the MD5 algorithm and the Hex digest.

```
const hashedValue = crypto.createHash('md5')
    .update(image)
    .digest('hex');
```

Create a Tesseract worker and assign a default logger that will log the message to the console

```
console.log(`Recognizing ${image}`);

const worker = createWorker({
  logger: (m) => {
    console.log(m);
  },
});
```

The core of the script is the async IIFE that will execute the recognition process. The steps in the IIFE are:

1. Load the worker
2. Assume that the image has English text, load and initialize the language
3. Run the recognizer on the image and store the result in the `text` constant
4. Write the text to a file with the hashed name and the extension .txt
5. Terminate the Tesseract worker

```
(async () => {
  await worker.load(); // 1
  await worker.loadLanguage('eng'); 'eng'// 2ait worker.initialize('eng')
```

```
  const {
    data: {
      text,
    },
  } = await worker.recognize(image); /'eng'// 3

  await fs.writeFile(`${image}-${hashedValue}.txt`, text, (err) => {}); `
  await worker.terminate(); // 5
})();
```

This works from the command line by running the following command:

```
node index.mjs
```

If you don’t pass a path to an image as an argument, the script will use the default image. The path to the image can also be a URL, but I chose not to use a URL for the default as I want to be able to work offline.