Exercise 1 - TDD Basics

Description

Do you remember old INI files? They were especially common in older games and some applications. Basically a set of properties, values and (sometimes) sections. They tend to look like this:

```
; configuration file - edit here, or within the application

[video]
width=640
height=480
renderer=opengl
antialias=1

[sound]
music_volume=10
fx_volume=8

[network]
gateway=192.168.0.1
```

Let's write a parser, which takes text on this format, and produces an object in JavaScript with all keys and values.

Steps

- Create a new folder somewhere on your hard drive
- Inside this folder, run `npm init`. Fill out the questions (or just press enter and choose the defaults)
- We'll use mocha to run our tests. Run `npm i mocha` to install it.
- Create two files, one called `parser.js` and one called `test.js`
- Add the following to `test.js`:

```
const assert = require('assert');
const parse = require('./parser');

describe('ini parser tests', () => {
    it('should return an object with the given key and value', () => {
        const conf = parse("width=640");
        assert.strictEqual(conf.width, 640);
    });
});
```

- Add the following to `parser.js`:

```
module.exports = iniString => {
    return {};
}
```

- Open `package.json` and modify the `test` script to run `mocha test.js`. Save and exit the file.
- Make sure that everything works by running `npm t`. You should now have a failing test, since we are just returning an empty object.
- Make the test pass, by writing the simplest implementation possible (note: all you need to assume in this test, is that we have a single line on the form `a=b`. Do not try to plan more in advance than that - you'll end up writing more logic than intended').
- Add a new test, making sure that passing `"width = 640"` (with spaces) produce the same output. Make sure you have a failing test, and then write the implementation
- Add a new test (and implementation), checking that multi-line entries are OK, by passing a template string, such as:

```
// Note that we're using backticks for string literals, to pass a multiline string:
parse(`
width=640
height=480
`)
```

- Comments are generally single-line entries. Each line starting with `;` is considered a comment. Make sure that you can write comments in the text by passing something like (and of course, doing it using TDD):

```
parse(`
; This is a comment
width=640
height=480
`)
```

This should simply just be ignored (i.e., the parser should not break if you have comments). You can check and make sure code does not throw an exception using the `assert.doesNotThrow(...)` function. Alternatively, you could simply assert and make sure that the data structure just contains the correct `width` and `height` (depending on how you wrote the previous code, it might not throw at all)

Stretch task

A newer addition to `.ini` files is the concept of *sections*. Basically, by adding `[name_of_section]` in the file, it should be possible to categorize parts of the file. In our case, we'd like to translate something like this:

```
player_name=roger

[video]
width=640
height=480
renderer=opengl
antialias=1

[sound]
music_volume=10
fx_volume=8
```

... Into an object where we can access things like this:

```
const result = parse("...ini content goes here")

result.player_name // should be 'roger'
result.video.width // should be 640
result.sound.music_volume // should be 10

// ... and so on
```

Pick the simplest use-case to start with, write a test, and implement it. Work your way towards parsing sections in this way.