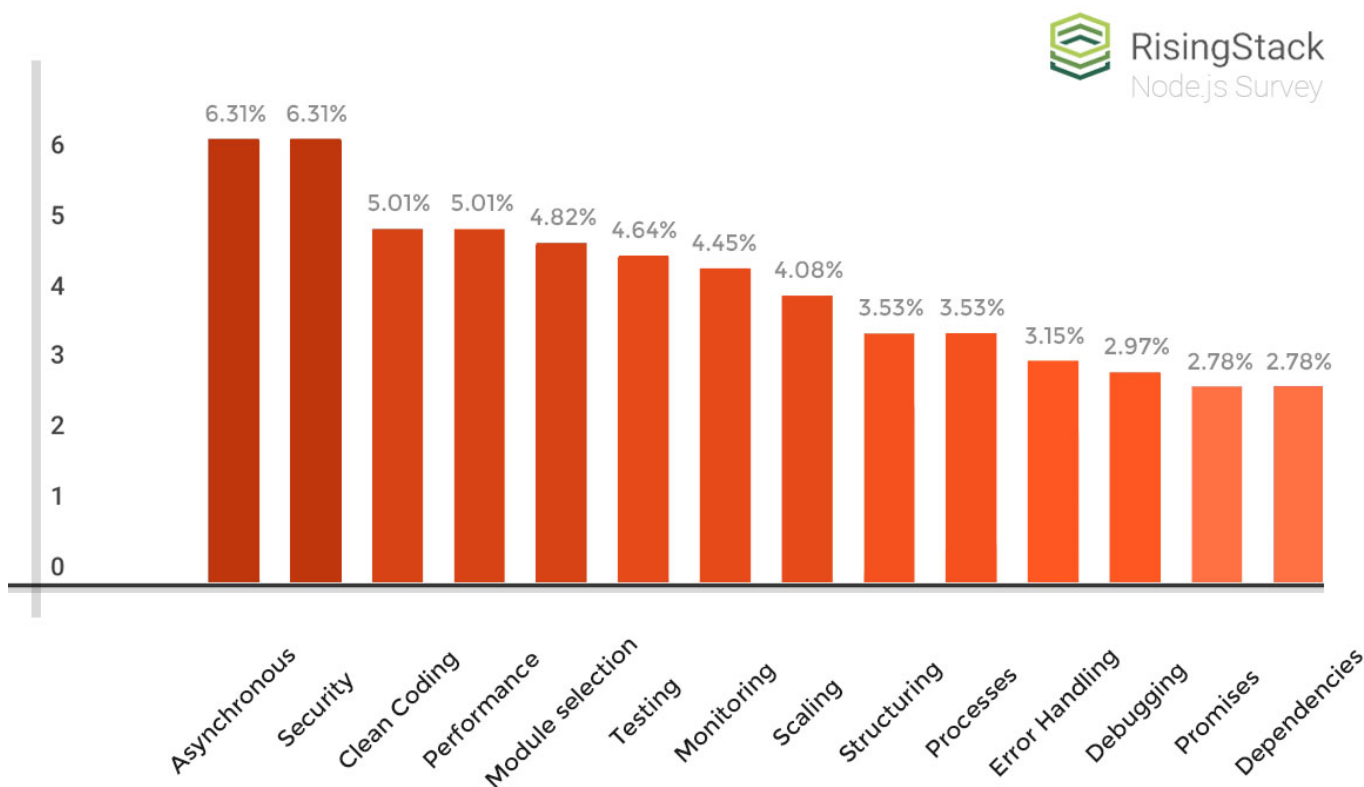


How To Use This Book

[Async/await](#) is the single most valuable feature to land in the JavaScript language spec in the last 15 years. The event loop and asynchronous programming in general are exceptional for building GUIs and servers, but callbacks make error handling tedious and code hard to read. For example, when [RisingStack](#) asked Node.js developers what they struggled with in 2017, asynchronous programming topped the list.

What's hardest to get right with Node.js at the moment?



Async/await promises to make asynchronous code as clean and easy to read as synchronous code in most use cases. Tangled promise chains and complex user-land libraries like [async](#) can be replaced with `for` loops, `if` statements, and `try/catch` blocks that even the most junior of engineers can make sense of.

The following [JavaScript from a 2012 blog post](#) is a typical example of where code goes wrong with callbacks. This code works, but it has a lot of error handling boilerplate and deeply nested `if` statements that obfuscate the actual logic. Wrapping your mind around it takes a while, and proper error handling means copy/pasting `if (err != null)` into every callback.

```

function getWikipediaHeaders() {
  // i. check if headers.txt exists
  fs.stat('./headers.txt', function(err, stats) {
    if (err != null) { throw err; }
    if (stats == undefined) {
      // ii. fetch the HTTP headers
      var options = { host: 'www.wikipedia.org', port: 80 };
      http.get(options, function(err, res) {
        if (err != null) { throw err; }
        var headers = JSON.stringify(res.headers);
        // iii. write the headers to headers.txt
        fs.writeFile('./headers.txt', headers, function(err) {
          if (err != null) { throw err; }
          console.log('Great Success!');
        });
      });
    } else { console.log('headers already collected'); }
  });
}

```

Below is the same code using async/await, assuming that `stat()`, `get()`, and `writeFile()` are properly promisified.

```

async function getWikipediaHeaders() {
  if (await stat('./headers.txt') != null) {
    console.log('headers already collected');
  }
  const res = await get({ host: 'www.wikipedia.org', port: 80 });
  await writeFile('./headers.txt', JSON.stringify(res.headers));
  console.log('Great success!');
}

```

You might not think async/await is a big deal. You might even think async/await is a bad idea. I've been in your shoes: when I first learned about async/await in 2013, I thought it was unnecessary at best. But when I started working with generator-based coroutines (the 2015 predecessor to async/await), I was shocked at how quickly server crashes due to `TypeError: Cannot read property 'x' of undefined` vanished. By the time async/await became part of the JavaScript language spec in 2017, async/await was an indispensable part of my dev practice.

Just because async/await is now officially part of JavaScript doesn't mean the world is all sunshine and rainbows. Async/await is a new pattern that promises to make day-to-day development work easier, but, like any pattern, you need to understand it or you'll do more harm

than good. If your `async/await` code is a patchwork of copy/pasted StackOverflow answers, you're just trading callback hell for the newly minted [async/await hell](#).

The purpose of this book is to take you from someone who is casually acquainted with promises and `async/await` to someone who is comfortable building and debugging a complex app whose core logic is built on `async/await`. This book is only 52 pages and is meant to be read in about 2 hours total. You may read it all in one sitting, but you would be better served reading one chapter at a time, studying the exercises at the end, and getting a good night's sleep in between chapters to really internalize the information.

This book is broken up into 4 chapters. Each chapter is 12 pages, including exercises at the end of each chapter that highlight key lessons from the chapter. The exercises require more thought than code and should be easy to answer within a few minutes.

The first 3 chapters are focused on promise and `async/await` fundamentals, and strive to avoid frameworks and outside dependencies. In particular, the code samples and exercises are meant to run in Node.js 8.x and will **not** use any transpilers like Babel.

In the interest of providing realistic examples, the code samples will use the `superagent` module for making HTTP requests. The 4th chapter will discuss integrating `async/await` with some common npm modules.

If you find any issues with the code samples or exercises, please report them at github.com/vkarpov15/mastering-async-await-issues.

Are you ready to master `async/await`? Let's get started!

Async/Await: The Good Parts

The `async` and `await` keywords are new additions to JavaScript as part of the 2017 edition of the language specification. The `async` keyword modifies a function, either a normal `function() {}` or an arrow function `() => {}`, to mark it as an *async function*. In an async function, you can use the `await` keyword to pause the function's execution until a promise settles. In the below function, the `await` keyword pauses the function's execution for approximately 1 second.

Example 1.1

```
async function test() {  
  // This function will print "Hello, World!" after 1 second.  
  await new Promise(resolve => setTimeout(() => resolve(), 1000));  
  console.log('Hello, World!');  
}  
  
test();
```

You can use the `await` keyword anywhere in the body of an async function. This means you can use `await` in `if` statements, `for` loops, and `try/catch` blocks. Below is another way to pause an async function's execution for about 1 second.

Example 1.2

```
async function test() {  
  // Wait 100ms 10 times. This function also prints after 1 sec.  
  for (let i = 0; i < 10; ++i) {  
    await new Promise(resolve => setTimeout(resolve, 100));  
  }  
  console.log('Hello, World!');  
}  
  
test();
```

There is one major restriction for using `await`: you can only use `await` within the body of a function that's marked `async`. The following code throws a `SyntaxError`.

Example 1.3

```
function test() {  
  const p = new Promise(resolve => setTimeout(resolve, 1000));  
  // SyntaxError: Unexpected identifier  
  await p;  
}  
  
test();
```