



# Modern Full-Stack Development

Using TypeScript, React, Node.js,  
Webpack, and Docker

—  
Frank Zammetti

Apress®

# Modern Full-Stack Development

Using TypeScript, React, Node.js,  
Webpack, and Docker

Frank Zammetti

Apress®

## ***Modern Full-Stack Development***

Frank Zammetti  
Pottstown, PA, USA

ISBN-13 (pbk): 978-1-4842-5737-1  
<https://doi.org/10.1007/978-1-4842-5738-8>

ISBN-13 (electronic): 978-1-4842-5738-8

Copyright © 2020 by Frank Zammetti

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Louise Corrigan  
Development Editor: James Markham  
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Ekrulila from Pexels

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [rights@apress.com](mailto:rights@apress.com), or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/9781484257371](http://www.apress.com/9781484257371). For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*Dedicated to Traci, Andrew, and  
Ashley – the only people I want to share  
a shed in the woods with when the zombies come!*

*(Okay, maybe not the “only” ones...  
but they get the good canned beans.)*

# Table of Contents

|   |             |
|---|-------------|
| <b>About the Author .....</b>                                   | <b>xv</b>   |
| <b>About the Technical Reviewer .....</b>                       | <b>xvii</b> |
| <b>Acknowledgments .....</b>                                    | <b>xix</b>  |
| <b>Introduction .....</b>                                       | <b>xxi</b>  |
| <b>Chapter 1: Server-Side Action: Node and NPM .....</b>        | <b>1</b>    |
| Of JavaScript Runtimes and Building (Mostly) Servers.....       | 2           |
| First Baby Steps with Node: Installation.....                   | 5           |
| More Useful: Executing JavaScript Source Files .....            | 7           |
| Node’s Partner in Crime: NPM .....                              | 8           |
| A Few More NPM Commands.....                                    | 10          |
| Initializing a New NPM/Node Project.....                        | 11          |
| Adding Dependencies .....                                       | 13          |
| A Quick Aside: Semantic Versioning.....                         | 14          |
| Fisher Price’s “My First Node Web Server” .....                 | 15          |
| Bonus Example.....  | 18          |
| Summary.....  | 20          |
| <b>Chapter 2: A Few More Words: Advanced Node and NPM .....</b> | <b>21</b>   |
| NPM: More on package.json.....                                  | 21          |
| NPM: Other Commands .....                                       | 25          |
| Auditing Package Security .....                                 | 26          |
| Deduplication and Pruning .....                                 | 26          |
| Finding/Searching for Packages sans Browser .....               | 27          |

TABLE OF CONTENTS

- Updating Packages..... 28
- Publishing/Unpublishing Packages ..... 28
- Node: Standard Modules ..... 29**
  - File System (fs)..... 30
  - HTTP and HTTPS (http and https) ..... 32
  - OS (os)..... 34
  - Path (path)..... 35
  - Process..... 36
  - Query Strings (querystring) ..... 37
  - URL (url) ..... 38
  - Utilities (util) ..... 39
  - The Rest of the Cast ..... 40
- Summary..... 42
- Chapter 3: Client-Side Adventures: React ..... 43**
  - A Brief History of React ..... 43
  - Yeah, Okay, History Nerd, That’s All Great, but What IS React?! ..... 46
    - The Real Star of the Show: Components..... 48
    - Components Need Info: Props ..... 55
    - Components (Sometimes) Need Memory: State..... 57
    - Making Them Look Good: Style ..... 60
  - In the End, Why React?..... 62
  - Summary..... 63
- Chapter 4: A Few More Words: Advanced React ..... 65**
  - A Better Way to Write React Code: JSX ..... 65
    - Yeah, Okay, So What Does It LOOK LIKE?! ..... 66
    - A Slight Detour into Babel Land ..... 68
    - Compile JSX ..... 71
    - And Now, Put It All Together ..... 72

|   |           |
|---|-----------|
| Whither Props? .....  | 76        |
| Default Props.....  | 77        |
| Typing Props.....   | 80        |
| Component Lifecycle .....                                       | 83        |
| Summary.....  | 86        |
| <b>Chapter 5: Building a Strong Foundation: TypeScript.....</b> | <b>87</b> |
| What Is TypeScript? .....                                       | 87        |
| Jumping into the Deep End .....                                 | 89        |
| Beyond the Playground .....                                     | 91        |
| Configuring TypeScript Compilation .....                        | 92        |
| The Nitty Gritty: Types.....                                    | 93        |
| String.....   | 94        |
| Number .....  | 94        |
| Boolean .....   | 95        |
| Any .....   | 95        |
| Arrays.....   | 95        |
| Tuples.....   | 96        |
| Enums .....   | 97        |
| Function .....  | 98        |
| Object.....   | 99        |
| Null, Void, and Undefined .....                                 | 99        |
| Custom Type Aliases.....  | 101       |
| Union Types .....   | 102       |
| TypeScript == ES6 Features for “Free”! .....                    | 103       |
| The let and const Keywords .....                                | 103       |
| Block Scope .....   | 103       |
| Arrow Functions .....   | 104       |
| Template Literals.....  | 105       |
| Default Parameters .....  | 105       |

TABLE OF CONTENTS

- Spread and Rest (and as an Added Bonus: Optional Arguments) ..... 106
- Destructuring ..... 107
- Classes ..... 109
- Summary ..... 116
- Chapter 6: A Few More Words: Advanced TypeScript..... 119**
- Interfaces ..... 119
- Argument/Object Interfaces ..... 120
- Methods in Interfaces ..... 122
- Interfaces and Classes ..... 123
- Extending Interfaces ..... 124
- Namespaces and Modules ..... 125
- Namespaces ..... 125
- Modules ..... 129
- Decorators ..... 131
- Decorator Factories ..... 134
- Third-Party Libraries ..... 135
- Debugging TypeScript Apps ..... 136
- Source Maps ..... 137
- Summary ..... 140
- Chapter 7: Tying It Up in a Bow: Webpack..... 141**
- What's a Bundle, and How Do I Make One?..... 142
- What's Webpack All About? ..... 143
- Dependency Graph ..... 143
- Entry ..... 144
- Output ..... 144
- Loaders ..... 145
- Plugins ..... 146
- Modes ..... 147
- Browser Compatibility ..... 148



|  |            |
|--|------------|
| Getting Started with Webpack.....                                | 148        |
| Getting More Complex.....  | 150        |
| Configuration.....   | 153        |
| Using Modules.....   | 156        |
| Wither TypeScript? .....   | 157        |
| Summary.....   | 159        |
| <b>Chapter 8: Delivering the Goods: MailBag, the Server.....</b> | <b>161</b> |
| What Are We Building? .....                                      | 161        |
| Basic Requirements .....   | 162        |
| Setting Up the Project .....                                     | 163        |
| Source File Rundown .....  | 164        |
| Adding Node Modules .....  | 165        |
| Adding Types .....   | 166        |
| A More Convenient Development Environment.....                   | 168        |
| The Starting Point: main.ts.....                                 | 170        |
| A Quick Detour: Time to Take a REST .....                        | 170        |
| Another Quick Detour: Express, for Fun and Profit.....           | 176        |
| Back to the Code! .....  | 177        |
| Gotta Know What We're Talking to: ServerInfo.ts .....            | 187        |
| Time to Send the Mail: smtp.ts.....                              | 189        |
| A Quick Detour: Nodemailer .....                                 | 190        |
| Another Quick Detour: Generics .....                             | 190        |
| Back to the Code! .....  | 193        |
| Time to Get the Mail (and Other Stuff): imap.ts .....            | 195        |
| A Quick Detour: emailjs-imap-client and mailparser .....         | 195        |
| Back to the Code! .....  | 196        |
| Reach Out and Touch Someone: contacts.ts .....                   | 204        |
| A Quick Detour: NoSQL.....                                       | 204        |
| Another Quick Detour: NeDB .....                                 | 207        |

TABLE OF CONTENTS

- Back to the Code! ..... 209
- Testing It All ..... 212
  - Optional Tooling ..... 215
- Suggested Exercises ..... 216
- Summary ..... 217
- Chapter 9: Delivering the Goods: MailBag, the Client ..... 219**
  - What Are We Building? ..... 219
    - Basic Requirements ..... 223
  - Setting Up the Project ..... 224
    - Source File Rundown ..... 228
  - The Starting Point: index.html ..... 230
  - The Starting Point, Redux: main.tsx ..... 230
    - A Quick Detour: State'ing the Obvious ..... 231
    - Back to the Code! ..... 234
  - A Bit of Configuration: config.ts ..... 237
  - A Worker for All Seasons ..... 237
    - A Quick Detour: AJAX ..... 238
    - Mirroring the Server Part 1: Contacts.ts ..... 242
    - Mirroring the Server Part 2: IMAP.ts ..... 244
    - Mirroring the Server Part 3: SMTP.ts ..... 247
  - A Cavalcade of Components ..... 247
    - A Quick Detour: Material-UI ..... 248
    - BaseLayout.tsx ..... 253
    - A Quick Detour: Functional Components ..... 260
    - Toolbar.tsx ..... 261
    - MailboxList.tsx ..... 264
    - ContactList.tsx ..... 266
    - ContactView.tsx ..... 268
    - MessageList.tsx ..... 273

|   |            |
|---|------------|
| MessageView.tsx.....  | 275        |
| WelcomeView.tsx .....   | 280        |
| Suggested Exercises .....   | 280        |
| Summary .....   | 281        |
| <b>Chapter 10: Time for Fun: BattleJong, the Server .....</b>         | <b>283</b> |
| What Are We Building? .....   | 284        |
| Basic Requirements .....  | 285        |
| Setting Up the Project .....  | 286        |
| Some tsconfig.json Changes .....                                      | 287        |
| Adding Node Modules .....   | 289        |
| Adding Types .....  | 289        |
| Source File Rundown .....   | 289        |
| The Starting Point (the ONLY Point, in Fact!): server.ts.....         | 289        |
| A Quick Detour: WebSockets .....                                      | 290        |
| Back to the Code! .....   | 293        |
| Serving the Client: The Express Server .....                          | 294        |
| Handling Messages: The WebSocket Server and Overall Game Design ..... | 294        |
| Finishing Up the WebSocket Server .....                               | 299        |
| Of Tiles and Board Layouts.....                                       | 300        |
| Suggested Exercises .....   | 303        |
| Summary.....  | 304        |
| <b>Chapter 11: Time for Fun: BattleJong, the Client.....</b>          | <b>305</b> |
| What Are We Building? .....   | 305        |
| Basic Requirements .....  | 306        |
| Setting Up the Project .....  | 307        |
| Some tsconfig.json Changes .....                                      | 308        |
| Some webpack.config.js Changes .....                                  | 309        |
| Adding Libraries .....  | 311        |

TABLE OF CONTENTS

- Adding Types ..... 311
- Source File Rundown ..... 311
- The Starting Point: index.html ..... 312
- The REAL Starting Point: main.tsx ..... 313
- The Basic Layout: BaseLayout.tsx ..... 313
- Feedback and Status: ControlArea.tsx ..... 316
  - Scores ..... 317
  - Game State Messages..... 318
- Where the Action Is: PlayerBoard.tsx ..... 319
  - A Quick Detour: Custom-Type Definitions..... 320
- Back to the Code! ..... 321
  - The Render Process ..... 322
- Talking to the Server: socketComm.ts..... 326
  - Handling Server-Sent Messages ..... 327
  - Sending Messages to the Server ..... 329
- The Main Code: state.ts..... 329
  - A Few Interface for Good Measure..... 330
  - The Beginning of the State Object..... 330
  - Back to the Code! ..... 332
  - Message Handler Methods..... 334
  - The Big Kahuna: tileClick() ..... 335
- Suggested Exercises ..... 344
- Summary..... 345
- Chapter 12: Bringing the Dev Ship into Harbor: Docker ..... 347**
  - An Introduction to Containers and Containerization..... 347
  - The Star of the Show: Docker..... 349
  - Installing Docker ..... 350
  - Your First Container: “Hello, World!” of Course! ..... 352

|   |            |
|---|------------|
| A Quick Rundown of Key Docker Commands..... | 353        |
| Listing Images.....                         | 353        |
| Listing Containers.....                     | 353        |
| Starting (and Stopping) Containers.....     | 354        |
| Remove Containers and Images.....           | 355        |
| Pulling Images.....                         | 355        |
| Searching for Images.....                   | 355        |
| Attaching to a Container.....               | 356        |
| Viewing Container Logs.....                 | 356        |
| Creating Your Own Image.....                | 357        |
| Deploying to Docker Hub.....                | 362        |
| Wrapping Up MailBag and BattleJong.....     | 363        |
| Suggested Exercises.....                    | 365        |
| Summary.....                                | 365        |
| <b>Index.....</b>                           | <b>367</b> |

# About the Author

**Frank Zammetti** is an application architect for a major financial firm with nearly 25 years of professional experience (plus almost 15 years of nonprofessional experience before that). He is an author of, including this one, 12 technical books for Apress. Frank has also authored over two dozen certification exams for SHL as well as several independent articles for various publications. He is also a fiction author (shameless plug: look him up on Amazon if you like sci-fi) and a musician of some renown (and here, “some” should be taken to mean very little). Frank has been married for 25 years (to the same woman even!) and they have two children together.

# About the Technical Reviewer

**Kenneth Fukizi** is a software engineer, architect, and consultant with experience in coding on different platforms internationally. Prior to dedicated software development, he worked as a lecturer for a year and was then head of IT in different organizations. He has domain experience working with technology for companies in a wide variety of sectors. When he's not working, he likes reading up on emerging technologies and strives to be an active member of the software community.

# Acknowledgments

I'd like to acknowledge the exceptional team at Apress for allowing me to write not one but twelve books for them over the last decade or so. I've worked with so many great people, and it's virtually impossible not to forget someone in a list like this, but among the crew for sure are Ami Knox, Arockia Rajan Dhurai, Beth Christmas, Dulcy Nirmala Chellappa, Chris Mills, Christine Ricketts, Dominic Shakeshaft, Douglas Pundick, Frank Parnell, Frank Pohlmann, Gary Cornell, Jill Balzano, Julie Miller, Katie Stence, Kelly Gunther, Kelly Winkvist, Kevin Shea, Kim Wimpsett, Kimberly van der Elst, Krishnan Sathyamurthy, Laura Cheu, Laura Esterman, Leah Weissburg, Leonard Cuellar, Liz Welch, Louise Corrigan, Marilyn Smith, Michelle Lowman, Nancy Chen, Nicole Faraclas, Nirmal Selvaraj, Richard Dal Porto, Sharon Wilkey, Sofia Marchant, Stephanie Parker, Steve Anglin, Tina Nielsen, and Tracy Brown Collins.

As I said, I'm sure I've forgotten someone, but rest assured it was not on purpose! Thank you all for giving me a shot and allowing me to continue this journey. I most definitely could not have done it alone and I thank you all, unreservedly!



# Introduction

You know, when I started learning how to program, it was a piece of cake!

You'd turn on the computer and be greeted by a nice little "Ready" prompt. You'd start typing in some code (BASIC), and eventually, you'd type run, hit Enter, and watch whatever it was you put in there spit back something (my first program was a man drawn with various keyboard characters doing jumping jacks). You might save that program to a cassette – yes, kids, a *cassette!* – and hand it to your friends if you wanted to share.

But that was it. It was just that easy.

Nowadays, though, the story is *very* different.

Writing even a trivial application now involves layers upon layers of abstractions and complexities that you must mix together, like baking the world's most complicated cake, hoping it all works in the end. Then, should you want to distribute the technological terror you've constructed (sorry, Aldearan), you've got even more challenges to overcome.

How *anyone* learns to program from scratch these days, I'm not sure!

But I'm hoping to help there!

With this book, I'm going to look at the ingredients that go into baking a cake – err, building an application – these days. To be sure, it won't cover everything. And no one recipe is necessarily the same anyway – there are lots of choices available to a developer now. But I believe I've chosen the ones most commonly used to build modern full-stack applications.

What exactly is a full-stack application anyway? Well, simply put, it's an application that includes both a front-end "client," like a web site, and a back-end "server," like, well, a *server!* We're talking about building an application that combines those two halves into a coherent whole. Most application development these days is web-based in some way (where "web" doesn't have to mean something available on the public Internet, but something built with web technologies like HTML, JavaScript, and CSS), so that's what we're going to be doing in this book.

To do this, we're going to use React, which is one of the most popular libraries for building clients out there today. And we'll use Node.js, which is a popular choice for back-end development. We're also going to use TypeScript, a language that enhances

## INTRODUCTION

JavaScript on both sides of the fence to make our coding lives better. We're going to touch on several other tools that relate to all of this including Babel and Webpack. We'll talk about some strategies for connecting the client to the server including REST and WebSockets. Finally, you'll learn about packaging up applications using the very popular Docker.

All this will be combined to build two full, real applications. This way, it's not just simple, contrived examples. No, it'll be real code, practical solutions to real problems encountered in building them, and real techniques for putting all these pieces together and making sense of all this complexity.

In the end, you'll have a solid foundation for building modern full-stack applications that you can go forward with on your own to create greatness.

I mean it'll never be as great as my guy doing jumping jacks written in BASIC and loaded off a cassette, but you gotta have goals. 😊

So let's get to it. There's work to be done, learning to be accomplished, and, I hope, fun to be had!

## CHAPTER 1

# Server-Side Action: Node and NPM

Welcome to the book! I hope you've got a comfy chair under you, a tasty drink on the table next to you and perhaps a light snack (may I suggest some dark chocolate biscotti?), and your brain ready to soak up a ton of knowledge on modern web development, 'cause that's what the show is all about and the curtains are about to be drawn!

In this book, we'll be building two full apps that will demonstrate all the concepts that we'll be discussing along the way in a practical manner. Far from being just simple, contrived bits of code, these are two full apps which are functional and useful (and even *fun*, given that one of them is a game, which will provide you a whole new way of looking at coding). As we do so, you'll get insight into the thinking that went into them, their design and architecture, so you get a holistic picture of what's involved in building something like these two apps. You will even, here and there, get some notes about issues I faced and how I resolved them, things that will almost certainly help you achieve your goals as you charge onward into your own projects.

To start, we'll look at what is most usually (though not exclusively, as you'll learn!) the purview of the server side. Remember that we're talking "full-stack" development here, which means you'll be learning about coding clients as well as the server code they make use of in order to form a cohesive, whole application. In this chapter, we'll begin by looking at two extremely popular tools for developing servers: Node.js and NPM.

# Of JavaScript Runtimes and Building (Mostly) Servers

Ryan Dahl – that cat has some talent, I tell ya!

Ryan is the creator of a fantastic piece of software called Node.js (or just plain Node, as it is often written, and as I’ll write it from here on out). Ryan first presented Node at the European JSConf in 2009, and it was quickly recognized as a potential game-changer, as evidenced by the standing ovation his presentation received (I presume Ryan is an excellent presenter generally as well).

Node is a platform for running primarily, though not exclusively, server-side code that has high performance and is capable of handling large request loads with ease. It is based on the most widely used language on the planet today: JavaScript. It’s straightforward to get started with and understand, yet it puts tremendous power in the hands of developers, in large part thanks to its asynchronous and event-driven model of programming. In Node, almost everything you do is nonblocking, meaning code won’t hold up the processing of other request threads. Most types of I/O, which is where blocking comes into play most, are asynchronous in Node, whether it’s network calls or file system calls or database calls. This, plus the fact that to execute code, Node uses Google’s popular and highly tuned V8 JavaScript engine, the same engine that powers its Chrome browser, makes it very high performance and able to handle a large request load (assuming that you as the developer don’t botch things of course!).

It’s also worth noting that, as weird as it may sound, Node is single-threaded. It at first seems like this would be a performance bottleneck, but in fact, it’s a net benefit because it avoids context-switching. However, this is a little bit of a misnomer in that it’s more correct to say that Node is event-driven and single-threaded with background workers. When you fire off some type of I/O request, Node will generally spawn a new thread for that. But, while it’s doing its work, that single event-driven thread continues executing your code. All of this is managed with an event queue mechanism so that the callbacks for those I/O operations are fired, back on that single thread, when the responses come back. All of this means that there is no (or at least minimal) context-switching between threads but also that the single thread is never sitting idle (unless there is *literally* no work to do of course), so you wind up with that net positive benefit I mentioned.

**Note** In later chapters, you'll see that Node isn't specific to the server side of the equation, and in fact, you don't always build apps with Node; sometimes you use it to install and execute tools for various purposes on your own development machine. Hold on to that thought; we'll be coming back to before long a few chapters from now.

---

None of these technical details are especially important to use as a Node developer, but the performance it yields is what makes it no wonder that so many significant players and sites have adopted Node to one degree or another. These aren't minor outfits we're talking about, we're talking names you doubtless know, including DuckDuckGo, eBay, LinkedIn, Microsoft, Netflix, PayPal, Walmart, and Yahoo, to name just a few examples. These are large businesses that require top-tier performance, and Node can deliver on that promise (again, with the caveat that you as the developer don't mess things up, because that's always possible).

Node is a first-class runtime environment, meaning that you can do such things as interacting with the local file system, access relational databases, call remote systems, and much more. In the past, you'd have to use a "proper" runtime, such as Java or .Net to do all this; JavaScript wasn't a player in that space. With Node, this is no longer true. It can compete not only on performance but also in terms of what capabilities it provides to developers. If you can think of it, chances are you can do it with Node, and that wasn't always the case with JavaScript.

To be clear, Node isn't in and of itself a server. You can't just start up Node and make HTTP requests to it from a web browser. It won't do anything in response to your requests by default. No, to use Node as a server, you must write some (straightforward and concise, as you'll see) code that then runs on the Node "runtime." Yes, you effectively write your own web server and app server, if you want to split hairs (or potentially FTP, Telnet, or any other type of server you might wish to). That's a very odd thing to do as a developer - we usually apply the "don't reinvent the wheel" mantra for stuff like that and pull one of the hundreds of existing options off the shelf. Plus, writing such servers sounds (and probably actually *is*) daunting to most developers, and for good reason! To be sure, it absolutely would be if you tried to write a web server from scratch in many other languages, especially if you want it to do more than just serve static content files. But not with Node!

But remember, acting as a server is just one capability that Node provides as a JavaScript runtime, and it can provide this functionality only if you, as a developer, feed it the code it needs to do so! In fact, a great many developer tools, and other types of apps, use Node as their runtime nowadays. Node really is all over the place!

---

**Note** As you'll see, React, Webpack, and TypeScript, three things that are primary focuses of this book (Docker being the outlier), use Node to run and/or to be installed (well, NPM is used to install them if we're being accurate, but we'll get to NPM in just a moment). These are tools, not servers, which is the main point: Node is useful for much more than just creating servers!

---

Node allows you to use the same language and knowledge on both client and server, something that was difficult to accomplish before. In fact, aside from Java and some Microsoft technologies (see project Blazor, which seeks to do the same thing with C#, if you're curious), there never has really been an opportunity to do so until Node came along. It's a pretty compelling opportunity.

Another critical aspect of Node is a driving design goal of the project, which is keeping its core functionality to an absolute minimum and providing extended functionality by way of APIs (in the form of JavaScript modules) that you can pick and choose from as needed. Node gets out of your way as much as possible and allows you only to introduce the complexity you really need, when you need it. Node ships with an extensive library of such modules, but each must be imported into your code, and then there are literally thousands of other modules that you can bring in as needed, some of which you'll see as we progress throughout this book.

In addition to all of this, getting, installing, and running Node are trivial exercises, regardless of your operating system preference. There are no complicated installs with all sorts of dependencies to manage, nor is there a vast set of configuration files to mess with before you can bring up a server and handle requests. It's a five-minute exercise, depending on the speed of your Internet connection and how fast you can type! There is also no required tooling to work with Node. In fact, a simple text editor is enough, in simplest terms (though that isn't to say you won't want a robust IDE with Node support later, but at least for this book I won't be assuming anything other than Notepad or some equivalent text editor).

All of this makes working with Node so much more straightforward than many competing options while providing you with top-notch performance and load handling capabilities. Moreover, it does so with a consistent technological underpinning as that which you develop your client applications.

That's Node in a nutshell!

Next, let's see about getting it onto your machine so that you can start playing with some code together and we can look at Node in a little more depth.

---

**Note** If you aren't a JavaScript expert, don't worry, we won't be getting too fancy. Even when we get to the apps, I'll consciously keep things as simple as possible. It *is* expected that you have *some* experience with JavaScript though, but you don't need to be Brendan Eich or Doug Crockford (but if you have no experience with TypeScript, that's fine; we'll start from square one with it later).

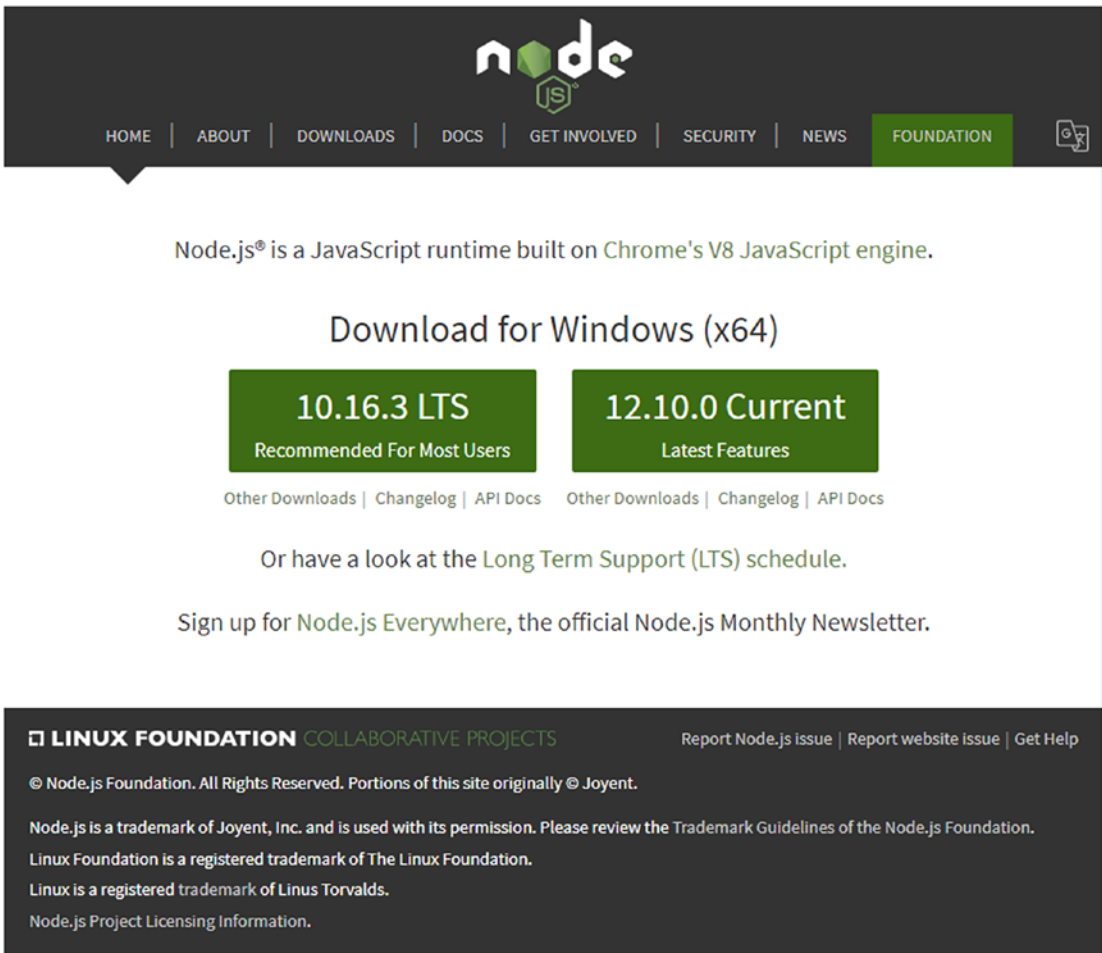
---

## First Baby Steps with Node: Installation

To get started, there's only one address to remember:

`http://nodejs.org`

That's your one-stop shop for all things Node, beginning, right from the front page, with downloading it, as you can see in Figure 1-1.



**Figure 1-1.** Node has a simple web site, but it gets the job done!

Usually, I would tell you to install the latest version available, but in this case, it might be better to choose a long-term support (LTS) version, because they tend to be more stable. However, it shouldn't (he said, with fingers crossed) matter which you choose, for the purposes of this book. For the record, however, I developed all the code using version 10.16.3, so if you encounter any problems, I would suggest choosing that version, which you can get from the Other Downloads link and then the Previous Releases link (you'll be able to download any past version you like from there).

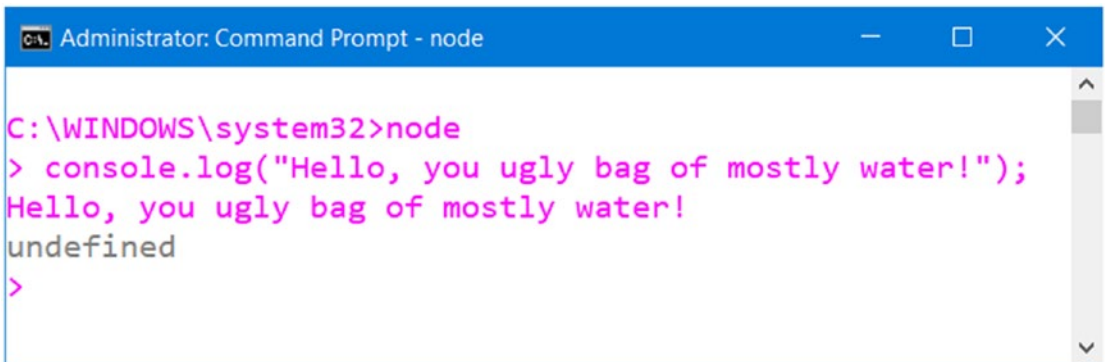
The download will install in whatever fashion is appropriate for your system, and I leave this as an exercise for the reader. For example, on Windows, Node provides a perfectly ordinary and straightforward installer that will walk you through the necessary (and extremely simple) steps. On macOS X, a typical install wizard will do the same.



Once the install completes, you will be ready to play with Node. The installer should have added the Node directory to your path. So, as a first simple test, go to a command prompt or console prompt, type `node`, and press Enter. You should be greeted with a `>` prompt. Node is now listening for your commands in interactive mode. To confirm, type the following:

```
console.log("Hello, you ugly bad of mostly water!");
```

Press Enter, and you should be greeted with something like what you see in Figure 1-2 (platform differences excepted, I'm a Windows guy myself, unashamedly, so that's where the screenshots throughout this book will be from, perhaps with a few exceptions later).



```
C:\WINDOWS\system32>node
> console.log("Hello, you ugly bag of mostly water!");
Hello, you ugly bag of mostly water!
undefined
>
```

**Figure 1-2.** *The rather uppity (though technically accurate) first greeting, proving Node is alive*

If you find that this doesn't work, please take a moment and ensure that Node is indeed in your path. It will make things a lot easier going forward.

## More Useful: Executing JavaScript Source Files

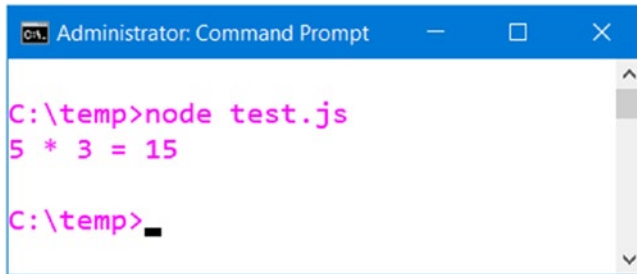
Interacting with Node in CLI (Command-Line Interface) mode like this is fine and dandy, but it's limited. What you really want to do is execute a saved JavaScript file using Node. As it happens, that's easy to do. Create a text file named `test.js` (it could be anything, but that's a pretty good choice at this point), and type the following code into it (and, of course, save it):

```
let a = 5;
let b = 3;
let c = a * b;
console.log(`${a} * ${b} = ${c}`);
```

To execute this file, assuming you are in the directory in which the file is located, you simply must type this:

```
node test.js
```

Press Enter after that, and you should be greeted with an execution, such as the one you see in Figure 1-3.



**Figure 1-3.** *It ain't much, but it's a real program running with Node!*

Clearly, this little bit of code is unexceptional, but it does demonstrate that Node can execute plain old JavaScript just fine. It demonstrates that we're dealing with at least the ES2015 specification as well, being more specific, thanks to the use of let and template literals (or string interpolation if you prefer). You can experiment a bit if you like, and you will see that Node should run any basic JavaScript that you care to throw at it like this.

## Node's Partner in Crime: NPM

NPM, which stands for Node Package Manager, is a companion app that installs alongside Node (though it is developed separately and can be updated on a different schedule than Node). With it, you can download packages, which are reusable JavaScript modules (and any supporting stuff they might need) from a central package registry (or a private repository if you have one). The central repository you can find at

[www.npmjs.com](http://www.npmjs.com)

You can visit it through a web browser and look through all the packages available, which makes finding exactly what you need easy.

Using NPM is simple: it's merely another command to run from a command prompt, just like Node is. For example, let's say you create a directory named `MyFirstNodeProject`. In it, you execute the following:

```
npm install express
```

Here, `npm` is the CLI program that is NPM itself, and `install` is one command you can issue to it. Then, `express` is an argument to that command, and this is the general form that most of your interactions with NPM will take.

---

**Note** Most NPM commands have a shorter form as well. For example, rather than type `install`, you can just type `i`, and it will do the same thing. Consult the NPM docs for these shortcuts, or be pedantic like your favorite author and always type it long-form, err, for clarity or something!

---

If you execute that, you'll find that a directory called `node-modules` has been created, and inside it will be a lot of...well, a lot of stuff you typically don't need to worry about too much! In short, though, it's all the code that makes up the Express module (which doesn't matter right now, but is a JavaScript module, or package if you prefer, which we'll be using in the MailBag app a few chapters hence... but we'll get to that app in due time, we've got a fair bit of ground to cover before then, so for now suffice it to say it's one of the two apps we're going to be building with the technologies discussed over the first six chapters), plus whatever modules Express itself depends on (and whatever they might depend on, and so on). NPM takes care of fetching all those dependencies for you. You'll also notice a file named `package-lock.json` has been created, and for our purposes here, you don't need to worry about that except to know not to delete it as NPM needs it to do its job.

When you use the `install` command like this, the modules you name are installed in the current directory, and this is referred to as the *local cache*, or *project cache*. You can also install the module into what's called the *global cache* by adding an argument to the command:

```
npm install -g express
```

Now, Express will be installed in a location outside the current directory and will be shared by all Node projects (or, more precisely, it will be available to all projects, because, of course, a project won't use a globally installed module unless you tell it to). Most usually, you will want to install dependencies in the project cache so that different projects can use different version of a given module than other projects (there is always a single version of a given module in the global cache, if any are present at all).

## A Few More NPM Commands

Aside from `install`, there are many other NPM commands, but you'll probably only use a subset most of the time. For example, to find out what modules are installed in your project, you can issue this command:

```
npm ls
```

Like on a \*nix system, `ls` is short for list, and that's what it does: lists the installed modules. What you'll see is a textual tree that shows the modules and then the modules they depend on. In other words, more will likely be shown than just the modules you explicitly installed (some modules don't have dependencies, but in the NPM ecosystem, those tend to be the exception rather than the rule).

---

**Tip** One very helpful tip I can give is that if you want to see just the top-level modules, whether in the global or local cache, you can add `--depth=0` to the `ls` command.

---

If you want to see what's installed in global cache instead, you can do

```
npm -g ls
```

In fact, keep that `-g` option in mind because you can add that to most NPM commands to differentiate between the local and global caches.

You can also update a given module:

```
npm update express
```

Just provide the name of the module to update, and NPM will take care of it, updating to the latest available version. If you don't provide a package name, then NPM will dutifully update *all* packages. And yes, you can drop a `-g` on it either way to target the global cache.

You can, of course, uninstall packages too:

```
npm uninstall express
```

Execute that and NPM will wipe Express from the local cache, along with its transient dependencies (so long as nothing else that remains that depends on them).

These few commands represent likely the majority of what you'll need to interact with NPM. I refer you to the NPM docs for other commands (and note that just typing `npm` and hitting Enter at a command prompt will show you a list of available commands, and you can then type `npm help <command>` to get information about each).

## Initializing a New NPM/Node Project

Now, in all of this, I did skip one step that clearly is optional but is, in fact, typical, and that's initializing a new project. With most Node/NPM projects, you'll also have a file named `package.json` in the root directory of the project. This file is the project manifest file, and it provides metadata information to NPM (and Node, at least indirectly) about your project that it needs to do certain things. It will tell NPM what modules to install if they haven't been installed yet for one thing (which makes giving a project to another developer very easy!). It will also contain information like the name and version of the project, its main entry point, and lots of other information (most of which is optional, but we'll look at that a bit more in the next chapter).

While you can write this file by hand or even go entirely without it, it's a good idea to have it, and it's a good idea to let NPM create it for you, which you can do by executing this command:

```
npm init
```

If you are following along, please make sure the directory you run this from is empty (delete `node_modules` and `package-lock.json` if present, both of which will be described later). This will trigger an interactive process that walks you through the creation of the `package.json` file, something like you see in Figure 1-4.

```

Administrator: Command Prompt

C:\temp>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (temp)
version: (1.0.0)
description: Init'ing a project
entry point: (test.js)
test command:
git repository:
keywords:
author: Frank W. Zammetti
license: (ISC)
About to write to C:\temp\package.json:
{
  "name": "temp",
  "version": "1.0.0",
  "description": "Init'ing a project",
  "main": "test.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Frank W. Zammetti",
  "license": "ISC"
}

Is this OK? (yes)

C:\temp>

```

**Figure 1-4.** *Initializing a project with NPM*

This will walk you through an interactive, step-by-step process wherein you can enter whichever information is relevant to your project, if any. You can just hit Enter on each option to use the default (or a blank value, whichever is applicable), or you can enter the values that are appropriate to you. For our purposes here though, you indeed can and should simply hit Enter on each prompt in the process.