Brad Dayley
Brendan Dayley
Caleb Dayley

Second Edition

# Node.js, MongoDB and Angular Web Development

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

The **Colosseum in Rome** was the largest amphitheater of the Roman Empire, and is now considered one of the greatest works of Roman architecture and engineering.

Known originally as the Flavian Amphitheater, the Colosseum was built and expanded by the three Flavian emperors, Vespasian (69-79 AD), Titus (79-81), and Domitian (81–96). The structure was given its current name from an enormous statue of the Emperor Nero that at one time stood next to the amphitheater.

It is estimated that the Colosseum could hold between 50,000 and 80,000 spectators for gladiatorial contests, animal hunts, executions, reenactments of land and sea battles, and dramas based on Roman and Greek mythology.

After the fall of Rome, the Colosseum began to fall into a state of disrepair. An earthquake caused the south side of the amphitheater to collapse, and for hundreds of years, looters and even the Church removed marble, stone, and bronze for use in other buildings.

It was the Church, however, that saved the Colosseum from complete destruction. To memorialize the early Christians believed to have died as martyrs in the Colosseum, the structure was consecrated by the Pope in 1749, putting a stop to the removal of the amphitheater's marble and ultimately leading to renovations in the 1800s.

# Node.js, MongoDB and Angular Web Development

Second Edition

# Node.js, MongoDB and Angular Web Development

Second Edition

Brad Dayley
Brendan Dayley
Caleb Dayley

Addison-Wesley

**Node.js, MongoDB and Angular Web Development, Second Edition**



ISBN-13: 978-0-13-465553-6

ISBN-10: 0-13-465553-2

Library of Congress Control Number: 2017954802

Printed in the United States of America

1 17



**Editor**
Mark Taber

**Senior Project Editor**
Tonya Simpson

**Copy Editor**
Geneil Breeze

**Indexer**
Erika Millen

**Compositor**
codeMantra

**Proofreader**
Abigail Manheim

**Technical Editor**
Jesse Smith

**Cover Designer**
Chuti Prasertsith

# Contents at a Glance

Introduction 1

**I: Getting Started**

1 Introducing the Node.js-to-Angular Stack 7

2 JavaScript Primer 15

**II: Learning Node.js**

3 Getting Started with Node.js 39

4 Using Events, Listeners, Timers, and Callbacks in Node.js 55

5 Handling Data I/O in Node.js 73

6 Accessing the File System from Node.js 95

7 Implementing HTTP Services in Node.js 115

8 Implementing Socket Services in Node.js 139

9 Scaling Applications Using Multiple Processors in Node.js 159

10 Using Additional Node.js Modules 181

**III: Learning MongoDB**

11 Understanding NoSQL and MongoDB 191

12 Getting Started with MongoDB 201

13 Getting Started with MongoDB and Node.js 221

14 Manipulating MongoDB Documents from Node.js 241

15 Accessing MongoDB from Node.js 261

16 Using Mongoose for Structured Schema and Validation 291

17 Advanced MongoDB Concepts 327

**IV: Using Express to Make Life Easier**

18 Implementing Express in Node.js 343

19 Implementing Express Middleware 367

**V: Learning Angular**

**20** Jumping into TypeScript **383**

**21** Getting Started with Angular **391**

**22** Angular Components **403**

**23** Expressions **415**

**24** Data Binding **429**

**25** Built-in Directives **441**

**VI: Advanced Angular**

**26** Custom Directives **449**

**27** Events and Change Detection **457**

**28** Implementing Angular Services in Web Applications **469**

**29** Creating Your Own Custom Angular Services **503**

**30** Having Fun with Angular **525**

Index **549**

# Contents

**Introduction 1**

**I: Getting Started**

**1 Introducing the Node.js-to-Angular Stack 7**

Understanding the Basic Web Development Framework 7

User 8

Browser 8

Webserver 10

Backend Services 10

Understanding the Node.js-to-Angular Stack Components 11

Node.js 11

MongoDB 12

Express 13

Angular 13

Summary 14

Next 14

**2 JavaScript Primer 15**

Defining Variables 15

Understanding JavaScript Data Types 16

Using Operators 17

Arithmetic Operators 17

Assignment Operators 18

Applying Comparison and Conditional Operators 18

Implementing Looping 21

`while` Loops 21

`do/while` Loops 22

`for` Loops 22

`for/in` Loops 23

Interrupting Loops 23

Creating Functions 24

Defining Functions 24

Passing Variables to Functions 24

Returning Values from Functions 25

Using Anonymous Functions 25

Understanding Variable Scope 26
Using JavaScript Objects 27
Using Object Syntax 27
Creating Custom-Defined Objects 28
Using a Prototyping Object Pattern 29
Manipulating Strings 29
Combining Strings 31
Searching a String for a Substring 31
Replacing a Word in a String 31
Splitting a String into an Array 32
Working with Arrays 32
Combining Arrays 33
Iterating Through Arrays 34
Converting an Array into a String 34
Checking Whether an Array Contains an Item 34
Adding and Removing Items to Arrays 34
Adding Error Handling 35
`try/catch` Blocks 35
Throw Your Own Errors 36
Using `finally` 36
Summary 37
Next 37

**II: Learning Node.js**

**3 Getting Started with Node.js 39**
Understanding Node.js 39
Who Uses Node.js? 40
What Is Node.js Used For? 40
What Does Node.js Come With? 40
Installing Node.js 42
Looking at the Node.js Install Location 42
Verify Node.js Executables 42
Selecting a Node.js IDE 43
Working with Node Packages 43
What Are Node Packaged Modules? 43
Understanding the Node Package Registry 43

Using the Node Package Manager 44
Searching for Node Package Modules 45
Installing Node Packaged Modules 46
Using `package.json` 47
Creating a Node.js Application 48
Creating a Node.js Packaged Module 49
Publishing a Node.js Packaged Module to the NPM Registry 50
Using a Node.js Packaged Module in a Node.js Application 52
Writing Data to the Console 53
Summary 54
Next 54

**4 Using Events, Listeners, Timers, and Callbacks in Node.js 55**
Understanding the Node.js Event Model 55
Comparing Event Callbacks and Threaded Models 55
Blocking I/O in Node.js 57
The Conversation Example 57
Adding Work to the Event Queue 59
Implementing Timers 60
Using `nextTick` to Schedule Work 63
Implementing Event Emitters and Listeners 64
Implementing Callbacks 67
Passing Additional Parameters to Callbacks 67
Implementing Closure in Callbacks 68
Chaining Callbacks 70
Summary 71
Next 71

**5 Handling Data I/O in Node.js 73**
Working with JSON 73
Converting JSON to JavaScript Objects 74
Converting JavaScript Objects to JSON 74
Using the `Buffer` Module to Buffer Data 74
Understanding Buffered Data 75
Creating Buffers 75
Writing to Buffers 76
Reading from Buffers 77
Determining Buffer Length 78

Copying Buffers 79
Slicing Buffers 80
Concatenating Buffers 81
Using the Stream Module to Stream Data 81
`Readable` Streams 82
`Writable` Streams 84
`Duplex` Streams 86
`Transform` Streams 88
Piping `Readable` Streams to `Writable` Streams 89
Compressing and Decompressing Data with `Zlib` 91
Compressing and Decompressing Buffers 91
Compressing/Decompressing Streams 92
Summary 93
Next 93

**6 Accessing the File System from Node.js 95**
Synchronous Versus Asynchronous File System Calls 95
Opening and Closing Files 96
Writing Files 97
Simple File Write 98
Synchronous File Writing 98
Asynchronous File Writing 99
Streaming File Writing 101
Reading Files 102
Simple File Read 102
Synchronous File Reading 103
Asynchronous File Reading 104
Streaming File Reading 105
Other File System Tasks 106
Verifying Path Existence 106
Getting File Info 107
Listing Files 108
Deleting Files 110
Truncating Files 110
Making and Removing Directories 111
Renaming Files and Directories 112
Watching for File Changes 112
Summary 113
Next 113

**7 Implementing HTTP Services in Node.js 115**
Processing URLs 115
Understanding the `URL` Object 116
Resolving the URL Components 117
Processing Query Strings and Form Parameters 117
Understanding Request, Response, and Server Objects 118
The `http.ClientRequest` Object 118
The `http.ServerResponse` Object 121
The `http.IncomingMessage` Object 122
The `http.Server` Object 123
Implementing HTTP Clients and Servers in Node.js 125
Serving Static Files 125
Implementing Dynamic `GET` Servers 127
Implementing `POST` Servers 130
Interacting with External Sources 132
Implementing HTTPS Servers and Clients 134
Creating an HTTPS Client 135
Creating an HTTPS Server 137
Summary 137
Next 137

**8 Implementing Socket Services in Node.js 139**
Understanding Network Sockets 139
Understanding TPC Server and `Socket` Objects 140
The `net.Socket` Object 140
The `net.Server` Object 144
Implementing TCP Socket Servers and Clients 147
Implementing a TCP Socket Client 147
Implementing a TCP Socket Server 150
Implementing TLS Servers and Clients 152
Creating a TLS Socket Client 153
Creating a TLS Socket Server 154
Summary 157
Next 157

**9 Scaling Applications Using Multiple Processors in Node.js 159**
Understanding the Process Module 159
Understanding Process I/O Pipes 159
Understanding Process Signals 160

Controlling Process Execution with the `process` Module 161
Getting Information from the `process` Module 161
Implementing Child Processes 164
Understanding the `ChildProcess` Object 164
Executing a System Command on Another Process Using `exec()` 166
Executing an Executable File on Another Process Using `execFile()` 168
Spawning a Process in Another Node.js Instance Using `spawn()` 169
Implementing Child Forks 171
Implementing Process Clusters 174
Using the Cluster Module 174
Understanding the `Worker` Object 175
Implementing an HTTP Cluster 176
Summary 179
Next 179

**10 Using Additional Node.js Modules 181**
Using the `os` Module 181
Using the `util` Module 183
Formatting Strings 183
Checking Object Types 184
Converting JavaScript Objects to Strings 184
Inheriting Functionality from Other Objects 185
Using the `dns` Module 186
Using the `crypto` Module 188
Other Node Modules and Objects 190
Summary 190
Next 190

**III: Learning MongoDB**

**11 Understanding NoSQL and MongoDB 191**
Why NoSQL? 191
Understanding MongoDB 192
Understanding Collections 192
Understanding Documents 192
MongoDB Data Types 193

Planning Your Data Model 194
Normalizing Data with Document References 195
Denormalizing Data with Embedded Documents 196
Using Capped Collections 197
Understanding Atomic Write Operations 198
Considering Document Growth 198
Identifying Indexing, Sharding, and Replication Opportunities 198
Large Collections Versus Large Numbers of Collections 199
Deciding on Data Life Cycles 199
Considering Data Usability and Performance 200
Summary 200
Next 200

**12 Getting Started with MongoDB 201**
Building the MongoDB Environment 201
Installing MongoDB 201
Starting MongoDB 202
Stopping MongoDB 203
Accessing MongoDB from the Shell Client 203
Administering User Accounts 206
Listing Users 206
Creating User Accounts 207
Removing Users 209
Configuring Access Control 209
Creating a User Administrator Account 209
Turning on Authentication 210
Creating a Database Administrator Account 211
Administering Databases 211
Displaying a List of Databases 211
Changing the Current Database 212
Creating Databases 212
Deleting Databases 212
Copying Databases 213
Managing Collections 214
Displaying a List of Collections in a Database 214
Creating Collections 214
Deleting Collections 215
Finding Documents in a Collection 216

Adding Documents to a Collection 217
Deleting Documents in a Collection 217
Updating Documents in a Collection 218
Summary 219
Next 219

**13 Getting Started with MongoDB and Node.js 221**
Adding the MongoDB Driver to Node.js 221
Connecting to MongoDB from Node.js 222
Understanding the Write Concern 222
Connecting to MongoDB from Node.js Using the `MongoClient` Object 223
Understanding the Objects Used in the MongoDB Node.js Driver 226
Understanding the `Db` Object 227
Understanding the `Admin` Object 229
Understanding the Collection Object 229
Understanding the `Cursor` Object 232
Accessing and Manipulating Databases 233
Listing Databases 233
Creating a Database 234
Deleting a Database 234
Creating, Listing, and Deleting Databases Example 234
Getting the Status of the MongoDB Server 236
Accessing and Manipulating Collections 237
Listing Collections 237
Creating Collections 237
Deleting Collections 238
Collection Creation, Listing, and Deleting Example 238
Getting Collection Information 239
Summary 240
Next 240

**14 Manipulating MongoDB Documents from Node.js 241**
Understanding Database Change Options 241
Understanding Database Update Operators 242
Adding Documents to a Collection 244
Getting Documents from a Collection 246
Updating Documents in a Collection 248

Atomically Modifying Documents in a Collection 250
Saving Documents in a Collection 252
Upserting Documents in Collection 253
Deleting Documents from a Collection 255
Removing a Single Document from a Collection 257
Summary 259
Next 259

**15 Accessing MongoDB from Node.js 261**
Introducing the Data Set 261
Understanding Query Objects 262
Understanding Query Options Objects 264
Finding Specific Sets of Documents 265
Counting Documents 268
Limiting Result Sets 270
Limiting Results by Size 270
Limiting Fields Returned in Objects 271
Paging Results 273
Sorting Result Sets 275
Finding Distinct Field Values 276
Grouping Results 277
Applying MapReduce by Aggregating Results 282
Understanding the `aggregate()` Method 283
Using Aggregation Framework Operators 283
Implementing Aggregation Expression Operators 285
Aggregation Examples 287
Summary 289
Next 289

**16 Using Mongoose for Structured Schema and Validation 291**
Understanding Mongoose 291
Additional Objects 292
Connecting to a MongoDB Database Using Mongoose 292
Defining a Schema 294
Understanding Paths 294
Creating a Schema Definition 294
Adding Indexes to a Schema 295
Implementing Unique Fields 296

Forcing Required Fields 296
Adding Methods to the Schema Model 296
Implementing the Schema on the Words Database 297
Compiling a Model 298
Understanding the Query Object 298
Setting the Query Database Operation 299
Setting the Query Database Operation Options 301
Setting the Query Operators 302
Understanding the Document Object 304
Finding Documents Using Mongoose 305
Adding Documents Using Mongoose 307
Updating Documents Using Mongoose 309
Saving Document Changes 310
Updating a Single Document 311
Updating Multiple Documents 313
Removing Documents Using Mongoose 314
Removing a Single Document 314
Removing Multiple Documents 315
Aggregating Documents Using Mongoose 317
Using the Validation Framework 320
Implementing Middleware Functions 322
Summary 325
Next 325

**17 Advanced MongoDB Concepts 327**
Adding Indexes 327
Using Capped Collections 330
Applying Replication 330
Replication Strategy 332
Deploying a Replica Set 333
Implementing Sharding 334
Sharding Server Types 335
Choosing a Shard Key 336
Selecting a Partitioning Method 337
Deploying a Sharded MongoDB Cluster 338
Repairing a MongoDB Database 341

Backing Up MongoDB 341
Summary 342
Next 342

IV: Using Express to Make Life Easier

18 Implementing Express in Node.js 343
Getting Started with Express 343
Configuring Express Settings 343
Starting the Express Server 345
Configuring Routes 345
Implementing Routes 346
Applying Parameters in Routes 347
Using Requests Objects 350
Using `Response` Objects 352
Setting Headers 352
Setting the Status 353
Sending Response 353
Sending JSON Responses 355
Sending Files 356
Sending a Download Response 359
Redirecting the Response 359
Implementing a Template Engine 360
Defining the Engine 360
Adding Locals 361
Creating Templates 361
Rendering Templates in a Response 363
Summary 365
Next 365

19 Implementing Express Middleware 367
Understanding Middleware 367
Assigning Middleware Globally to a Path 368
Assigning Middleware to a Single Route 368
Adding Multiple Middleware Functions 369
Using the `query` Middleware 369
Serving Static Files 369
Handling `POST` Body Data 371

Sending and Receiving Cookies 373
Implementing Sessions 374
Applying Basic HTTP Authentication 375
Implementing Session Authentication 377
Creating Custom Middleware 380
Summary 381
Next 382

V: Learning Angular

20 Jumping into TypeScript 383
Learning the Different Types 383
Understanding Interfaces 385
Implementing Classes 386
`Class` Inheritance 387
Implementing Modules 387
Understanding Functions 388
Summary 389
Next 389

21 Getting Started with Angular 391
Why Angular? 391
Understanding Angular 391
Modules 392
Directives 392
Data Binding 392
Dependency Injection 392
Services 393
Separation of Responsibilities 393
Adding Angular to Your Environment 393
Using the Angular CLI 394
Generating Content with the CLI 394
Creating a Basic Angular Application 395
Creating Your First Angular App 396
Understanding and Using NgModule 397
Creating the Angular Bootstrapper 398
Summary 402
Next 402

**22 Angular Components 403**
Component Configuration 403
Defining a Selector 404
Building a Template 404
Using Inline CSS and HTML in Angular Applications 405
Using Constructors 407
Using External Templates 408
Injecting Directives 410
Building a Nested Component with Dependency Injection 410
Passing in Data with Dependency Injection 412
Creating an Angular Application that Uses Inputs 413
Summary 414
Next 414

**23 Expressions 415**
Using Expressions 415
Using Basic Expressions 416
Interacting with the `Component` Class in Expressions 418
Using TypeScript in Angular Expressions 419
Using Pipes 422
Using Built-in Pipes 424
Building a Custom Pipe 426
Creating a Custom Pipe 427
Summary 428
Next 428

**24 Data Binding 429**
Understanding Data Binding 429
Interpolation 430
Property Binding 431
Attribute Binding 433
Class Binding 433
Style Binding 435
Event Binding 436
Two-Way Binding 439
Summary 440
Next 440

**25 Built-in Directives 441**
Understanding Directives 441
Using Built-in Directives 441
Components Directives 442
Structural Directives 442
Attribute Directives 445
Summary 448
Next 448

**VI: Advanced Angular**

**26 Custom Directives 449**
Creating a Custom Attribute Directive 449
Creating a Custom Directive with a Component 452
Summary 456
Next 456

**27 Events and Change Detection 457**
Using Browser Events 457
Emitting Custom Events 458
Emitting a Custom Event to the Parent Component Hierarchy 458
Handling Custom Events with a Listener 458
Implementing Custom Events in Nested Components 458
Deleting Data in a Parent Component from a Child Component 461
Using Observables 464
Creating an Observable Object 464
Watching for Data Changes with Observables 465
Summary 468
Next 468

**28 Implementing Angular Services in Web Applications 469**
Understanding Angular Services 469
Using the Built-in Services 469
Sending HTTP `GET` and `PUT` Requests with the `http` Service 470
Configuring the HTTP Request 471
Implementing the HTTP Response Callback Functions 471
Implementing a Simple JSON File and Using the `http` Service to Access It 472

Implementing a Simple Mock Server Using the `http` Service 475

Implementing a Simple Mock Server and Using the `http` Service to Update Items on the Server 481

Changing Views with the `router` Service 486

Using `routes` in Angular 488

Implementing a Simple Router 488

Implementing a Router with a Navigation Bar 492

Implementing a Router with Parameters 497

Summary 501

Next 501

**29 Creating Your Own Custom Angular Services 503**

Integrating Custom Services into Angular Applications 503

Adding an Angular Service to an Application 504

Implementing a Simple Application that Uses a Constant Data Service 505

Implementing a Data Transform Service 506

Implementing a Variable Data Service 510

Implementing a Service that Returns a Promise 515

Implementing a Shared Service 516

Summary 523

Next 523

**30 Having Fun with Angular 525**

Implementing an Angular Application that Uses the Animation Service 525

Implementing an Angular Application that Zooms in on Images 530

Implementing an Angular Application that Enables Drag and Drop 533

Implementing a Star Rating Angular Component 539

Summary 547

**Index 549**

## About the Authors

**Brad Dayley** is a senior software engineer with more than 20 years of experience developing enterprise applications and web interfaces. He has used JavaScript and jQuery for years and is the author of *Learning Angular, jQuery and JavaScript Phrasebook* and *Sams Teach Yourself AngularJS, JavaScript, and jQuery All in One*. He has designed and implemented a wide array of applications and services, from application servers to complex web applications.

**Brendan Dayley** is a web application developer who loves learning and implementing the latest and greatest technologies. He is the co-author of *Learning Angular* and *Sams Teach Yourself AngularJS, JavaScript, and jQuery All in One*. He has written a number of web applications using JavaScript, TypeScript, and Angular, and he is exploring the capabilities of new web and mobile technologies such as augmented reality and how to use them for innovative solutions.

**Caleb Dayley** is a university student studying computer science. He tries to learn all that he can and has taught himself much of what he knows about programming. He has taught himself several languages, including JavaScript, C#, and, using the first edition of this book, *NodeJS, MongoDB and Angular*. He is excited for what the future holds, and the opportunities to help design and create the next generation of innovative software that will continue to improve the way we live, work, and play.

## Acknowledgments

I'd like to take this page to thank all those who made this title possible. First, I thank my wonderful wife for the inspiration, love, and support she gives me. I'd never make it far without you. I also want to thank my boys for the help they are when I am writing. Thanks to Mark Taber for getting this title rolling in the right direction.

*—Brad Dayley*

I'd like to thank all those who helped make this book possible for me. First and foremost, my wife, who pushes me to become greater and gives me all her love. Also my father, who mentored me not just in writing and programming but in life. My mother, who has always been there for me when I need her. And finally, Mark Taber, who gave me the chance to be a part of this.

*—Caleb Dayley*

## Accessing the Free Web Edition

Your purchase of this book in any format includes access to the corresponding Web Edition, which provides several special online-only features:

- The complete text of the book
- Updates and corrections as they become available

The Web Edition can be viewed on all types of computers and mobile devices with any modern web browser that supports HTML5.

To get access to the Web Edition of *Node.js, MongoDB and Angular Web Development* all you need to do is register this book:

1. Go to www.informit.com/register.
2. Sign in or create a new account.
3. Enter the ISBN: 9780134655536.
4. Answer the questions as proof of purchase.
5. The Web Edition will appear under the Digital Purchases tab on your Account page. Click the Launch link to access the product.

*This page intentionally left blank*

# Introduction

Welcome to *Node.js, MongoDB and Angular Web Development*. This book is designed to catapult you into the world of using JavaScript—from the server and services to the browser client—in your web development projects. The book covers the implementation and integration of Node.js, MongoDB, and Angular—some of the most exciting and innovative technologies emerging in the world of web development.

This introduction covers

- Who should read this book
- Why you should read this book
- What you will be able to achieve using this book
- What Node.js, MongoDB, and Angular are and why they are great technologies
- How this book is organized
- Where to find the code examples

Let's get started.

## Who Should Read This Book

This book is aimed at readers who already have an understanding of the basics of HTML and have done some programming in a modern programming language. Having an understanding of JavaScript will make this book easier to digest but is not required because the book does cover the basics of JavaScript.

## Why You Should Read This Book

This book will teach you how to create powerful, interactive websites and web applications—from the webserver and services on the server to the browser-based interactive web applications. The technologies covered here are all open source, and you will be able to use JavaScript for both the server-side and browser-side components.

Typical readers of this book want to master Node.js and MongoDB for the purpose of building highly scalable and high-performing websites. Typical readers also want to leverage the MVC/MVVM (Model-View-Controller/Model-View-View-Model) approach of Angular to implement

well-designed and structured webpages and web applications. Overall, Node.js, MongoDB, and Angular provide an easy-to-implement, fully integrated web development stack that allows you to implement amazing web applications.

## What You Will Learn from This Book

Reading this book will enable you to build real-world, dynamic websites and web applications. Websites no longer consist of simple static content in HTML pages with integrated images and formatted text. Instead, websites have become much more dynamic, with a single page often serving as an entire site or application.

Using Angular technology allows you to build into your webpage logic that can communicate back to the Node.js server and obtain necessary data from the MongoDB database. The combination of Node.js, MongoDB, and Angular allows you to implement interactive, dynamic webpages. The following are just a few of the things that you will learn while reading this book:

- How to implement a highly scalable and dynamic webserver, using Node.js and Express
- How to build server-side web services in JavaScript
- How to implement a MongoDB data store for you web applications
- How to access and interact with MongoDB from Node.js JavaScript code
- How to define static and dynamic web routes and implement server-side scripts to support them
- How to define your own custom Angular components that extend the HTML language
- How to implement client-side services that can interact with the Node.js webserver
- How to build dynamic browser views that provide rich user interaction
- How to add nested components to your webpages
- How to implement Angular routing to manage navigation between client application views

## What Is Node.js?

Node.js, sometimes referred to as just Node, is a development framework that is based on Google's V8 JavaScript engine. You write Node.js code in JavaScript, and then V8 compiles it into machine code to be executed. You can write most—or maybe even all—of your server-side code in Node.js, including the webserver and the server-side scripts and any supporting web application functionality. The fact that the webserver and the supporting web application scripts are running together in the same server-side application allows for much tighter integration between the webserver and the scripts.

The following are just a few reasons Node.js is a great framework:

- **JavaScript end-to-end:** One of the biggest advantages of Node.js is that it allows you to write both server- and client-side scripts in JavaScript. There have always been difficulties in deciding whether to put logic in client-side scripts or server-side scripts. With Node.js you can take JavaScript written on the client and easily adapt it for the server, and vice versa. An added plus is that client developers and server developers are speaking the same language.
- **Event-driven scalability:** Node.js applies a unique logic to handling web requests. Rather than having multiple threads waiting to process web requests, with Node.js they are processed on the same thread, using a basic event model. This allows Node.js webservers to scale in ways that traditional webservers can't.
- **Extensibility:** Node.js has a great following and an active development community. People are providing new modules to extend Node.js functionality all the time. Also, it is simple to install and include new modules in Node.js; you can extend a Node.js project to include new functionality in minutes.
- **Fast implementation:** Setting up Node.js and developing in it are super easy. In only a few minutes you can install Node.js and have a working webserver.

## What Is MongoDB?

MongoDB is an agile and scalable NoSQL database. The name Mongo comes from the word "hu**mongo**us," emphasizing the scalability and performance MongoDB provides. MongoDB provides great website backend storage for high-traffic websites that need to store data such as user comments, blogs, or other items because it is quickly scalable and easy to implement.

The following are some of the reasons that MongoDB really fits well in the Node.js stack:

- **Document orientation:** Because MongoDB is document-oriented, data is stored in the database in a format that is very close to what you deal with in both server-side and client-side scripts. This eliminates the need to transfer data from rows to objects and back.
- **High performance:** MongoDB is one of the highest-performing databases available. Especially today, with more and more people interacting with websites, it is important to have a backend that can support heavy traffic.
- **High availability:** MongoDB's replication model makes it easy to maintain scalability while keeping high performance.
- **High scalability:** MongoDB's structure makes it easy to scale horizontally by sharing the data across multiple servers.
- **No SQL injection:** MongoDB is not susceptible to SQL injection (that is, putting SQL statements in web forms or other input from the browser and thereby compromising database security). This is the case because objects are stored as objects, not using SQL strings.

## What Is Angular?

Angular is a client-side JavaScript framework developed by Google. The theory behind Angular is to provide a framework that makes it easy to implement well-designed and structured webpages and applications, using an MVC/MVVM framework.

Angular provides functionality to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. Here are some of the benefits Angular provides:

- **Data binding:** Angular has a clean method for binding data to HTML elements, using its powerful scope mechanism.
- **Extensibility:** The Angular architecture allows you to easily extend almost every aspect of the language to provide your own custom implementations.
- **Clean:** Angular forces you to write clean, logical code.
- **Reusable code:** The combination of extensibility and clean code makes it easy to write reusable code in Angular. In fact, the language often forces you to do so when creating custom services.
- **Support:** Google is investing a lot into this project, which gives it an advantage over similar initiatives that have failed.
- **Compatibility:** Angular is based on JavaScript and has a close relationship with the JavaScript standard. This makes it easier to begin integrating Angular into your environment and reuse pieces of your existing code within the structure of the Angular framework.

## How This Book Is Organized

This book is divided into six main parts:

- Part I, "Getting Started," provides an overview of the interaction between Node.js, MongoDB, and Angular and how these three products form a complete web development stack. Chapter 2 is a JavaScript primer that provides the basics of the JavaScript language that you need when implementing Node.js and Angular code.
- Part II, "Learning Node.js," covers the Node.js language platform, from installation to implementation of Node.js modules. This part gives you the basic framework you need to implement your own custom Node.js modules as well as the webserver and server-side scripts.
- Part III, "Learning MongoDB," covers the MongoDB database, from installation to integration with Node.js applications. This part discusses how to plan your data model to fit your application needs and how to access and interact with MongoDB from your Node.js applications.

- Part IV, "Using Express to Make Life Easier," discusses the Express module for Node.js and how to leverage it as the webserver for your application. You learn how to set up dynamic and static routes to data as well as how to implement security, caching, and other webserver basics.
- Part V, "Learning Angular," covers the Angular framework architecture and how to integrate it into your Node.js stack. This part covers creating custom HTML components and client-side services that can be leveraged in the browser.
- Part VI, "Advanced Angular," covers more advanced Angular development, such as building custom directives and custom services. You also learn about using Angular's built-in HTTP and routing services. This section finishes with some additional rich UI examples, such as building drag-and-drop components and implementing animations.

## Getting the Code Examples

Throughout this book, you will find code examples in listings. The title for each listing includes a filename for the source code. The source code is available for download at the book's website.

## A Final Word

We hope you enjoy learning about Node.js, MongoDB, and Angular as much as we have. They are great, innovative technologies that are fun to use. Soon, you'll be able to join the many other web developers who use the Node.js-to-Angular web stack to build interactive websites and web applications. Enjoy the book!

*This page intentionally left blank*

# 3

# Getting Started with Node.js

This chapter introduces you to the Node.js environment. Node.js is a website/application framework designed with high scalability in mind. It was designed to take advantage of the existing JavaScript technology in the browser and flow those same concepts all the way down through the webserver into the backend services. Node.js is a great technology that is easy to implement and yet extremely scalable.

Node.js is a modular platform, meaning that much of the functionality is provided by external modules rather than being built in to the platform. The Node.js culture is active in creating and publishing modules for almost every imaginable need. Therefore, much of this chapter focuses on understanding and using the Node.js tools to build, publish, and use your own Node.js modules in applications.

## Understanding Node.js

Node.js was developed in 2009 by Ryan Dahl as an answer to the frustration caused by concurrency issues, especially when dealing with web services. Google had just come out with the V8 JavaScript engine for the Chrome web browser, which was highly optimized for web traffic. Dahl created Node.js on top of V8 as a server-side environment that matched the client-side environment in the browser.

The result is an extremely scalable server-side environment that allows developers to more easily bridge the gap between client and server. The fact that Node.js is written in JavaScript allows developers to easily navigate back and forth between client and server code and even reuse code between the two environments.

Node.js has a great ecosystem with new extensions being written all the time. The Node.js environment is clean and easy to install, configure, and deploy. Literally in only an hour or two you can have a Node.js webserver up and running.

## Who Uses Node.js?

Node.js quickly gained popularity among a wide variety of companies. These companies use Node.js first and foremost for scalability but also for ease of maintenance and faster development. The following are just a few of the companies using the Node.js technology:

- Yahoo!
- LinkedIn
- eBay
- *New York Times*
- Dow Jones
- Microsoft

## What Is Node.js Used For?

Node.js can be used for a wide variety of purposes. Because it is based on V8 and has highly optimized code to handle HTTP traffic, the most common use is as a webserver. However, Node.js can also be used for a variety of other web services such as:

- Web services APIs such as REST
- Real-time multiplayer games
- Backend web services such as cross-domain, server-side requests
- Web-based applications
- Multiclient communication such as IM

## What Does Node.js Come With?

Node.js comes with many built-in modules available right out of the box. This book covers many but not all of these modules:

- **Assertion testing:** Allows you to test functionality within your code.
- **Buffer:** Enables interaction with TCP streams and file system operations. (See Chapter 5, "Handling Data I/O in Node.js.")
- **C/C++ add-ons:** Allows for C or C++ code to be used just like any other Node.js module.
- **Child processes:** Allows you to create child processes. (See Chapter 9, "Scaling Applications Using Multiple Processors in Node.js.")
- **Cluster:** Enables the use of multicore systems. (See Chapter 9.)
- **Command line options:** Gives you Node.js commands to use from a terminal.
- **Console:** Gives the user a debugging console.

- **Crypto:** Allows for the creation of custom encryption. (See Chapter 10, "Using Additional Node.js Modules.")
- **Debugger:** Allows debugging of a Node.js file.
- **DNS:** Allows connections to DNS servers. (See Chapter 10.)
- **Errors:** Allows for the handling of errors.
- **Events:** Enables the handling of asynchronous events. (See Chapter 4, "Using Events, Listeners, Timers, and Callbacks in Node.js.")
- **File system:** Allows for file I/O with both synchronous and asynchronous methods. (See Chapter 6, "Accessing the File System from Node.js.")
- **Globals:** Makes frequently used modules available without having to include them first. (See Chapter 10.)
- **HTTP:** Enables support for many HTTP features. (See Chapter 7, "Implementing HTTP Services in Node.js.")
- **HTTPS:** Enables HTTP over the TLS/SSL. (See Chapter 7.)
- **Modules:** Provides the module loading system for Node.js. (See Chapter 3.)
- **Net:** Allows the creation of servers and clients. (See Chapter 8, "Implementing Socket Services in Node.js.")
- **OS:** Allows access to the operating system that Node.js is running on. (See Chapter 10.)
- **Path:** Enables access to file and directory paths. (See Chapter 6.)
- **Process:** Provides information and allows control over the current Node.js process. (See Chapter 9.)
- **Query strings:** Allows for parsing and formatting URL queries. (See Chapter 7.)
- **Readline:** Enables an interface to read from a data stream. (See Chapter 5.)
- **REPL:** Allows developers to create a command shell.
- **Stream:** Provides an API to build objects with the stream interface. (See Chapter 5.)
- **String decoder:** Provides an API to decode buffer objects into strings. (See Chapter 5.)
- **Timers:** Allows for scheduling functions to be called in the future. (See Chapter 4.)
- **TLS/SSL:** Implements TLS and SSL protocols. (See Chapter 8.)
- **URL:** Enables URL resolution and parsing. (See Chapter 7.)
- **Utilities:** Provides support for various apps and modules.
- **V8:** Exposes APIs for the Node.js version of V8. (See Chapter 10.)
- **VM:** Allows for a V8 virtual machine to run and compile code.
- **ZLIB:** Enables compression using Gzip and Deflate/Inflate. (See Chapter 5.)

# Installing Node.js

To easily install Node.js, download an installer from the Node.js website at http://nodejs.org. The Node.js installer installs the necessary files on your PC to get Node.js up and running. No additional configuration is necessary to start creating Node.js applications.

## Looking at the Node.js Install Location

If you look at the install location, you will see a couple of executable files and a `node_modules` folder. The `node` executable file starts the Node.js JavaScript VM. The following list describes the executables in the Node.js install location that you need to get started:

- **`node`:** This file starts a Node.js JavaScript VM. If you pass in a JavaScript file location, Node.js executes that script. If no target JavaScript file is specified, then a script prompt is shown that allows you to execute JavaScript code directly from the console.
- **`npm`:** This command is used to manage the Node.js packages discussed in the next section.
- **`node_modules`:** This folder contains the installed Node.js packages. These packages act as libraries that extend the capabilities of Node.js.

## Verify Node.js Executables

Take a minute and verify that Node.js is installed and working before moving on. To do so, open a console prompt and execute the following command to bring up a Node.js VM:

```
node
```

Next, at the Node.js prompt execute the following to write `"Hello World"` to the screen.

```
>console.log("Hello World");
```

You should see `"Hello World"` output to the console screen. Now exit the console using Ctrl+C in Windows or Cmd+C on a Mac.

Next, verify that the `npm` command is working by executing the following command in the OS console prompt:

```
npm version
```

You should see output similar to the following:

```
{ npm: '3.10.5',
  ares: '1.10.1-DEV',
  http_parser: '2.7.0',
  icu: '57.1',
  modules: '48',
  node: '6.5.0',
  openssl: '1.0.2h',
```

```
uv: '1.9.1',
v8: '5.1.281.81',
zlib: '1.2.8'}
```

### Selecting a Node.js IDE

If you are planning on using an Integrated Development Environment (IDE) for your Node.js projects, you should take a minute and configure that now as well. Most developers are particular about the IDE that they like to use, and there will likely be a way to configure at least for JavaScript if not Node.js directly. For example, Eclipse has some great Node.js plugins, and the WebStorm IDE by IntelliJ has some good features for Node.js built in. If you are unsure of where to start, we use Visual Studio Code for the built-in TypeScript functionality required later in this book.

That said, you can use any editor you want to generate your Node.js web applications. In reality, all you need is a decent text editor. Almost all the code you will generate will be .js, .json, .html, and .css. So pick the editor in which you feel the most comfortable writing those types of files.

## Working with Node Packages

One of the most powerful features of the Node.js framework is the ability to easily extend it with additional Node Packaged Modules (NPMs) using the Node Package Manager (NPM). That's right, in the Node.js world, NPM stands for two things. This book refers to the Node Packaged Modules as *modules* to make it easier to follow.

### What Are Node Packaged Modules?

A Node Packaged Module is a packaged library that can easily be shared, reused, and installed in different projects. Many different modules are available for a variety of purposes. For example, the Mongoose module provides an ODM (Operational Data Model) for MongoDB, Express extends Node's HTTP capabilities, and so on.

Node.js modules are created by various third-party organizations to provide the needed features that Node.js lacks out of the box. This community of contributors is active in adding and updating modules.

Node Packaged Modules include a `package.json` file that defines the packages. The `package.json` file includes informational metadata, such as the name, version author, and contributors, as well as control metadata, such as dependencies and other requirements that the Node Package Manager uses when performing actions such as installation and publishing.

### Understanding the Node Package Registry

The Node modules have a managed location called the Node Package Registry where packages are registered. This allows you to publish your own packages in a location where others can use them as well as download packages that others have created.

The Node Package Registry is located at https://npmjs.com. From this location you can view the newest and most popular modules as well as search for specific packages, as shown in Figure 3.1.

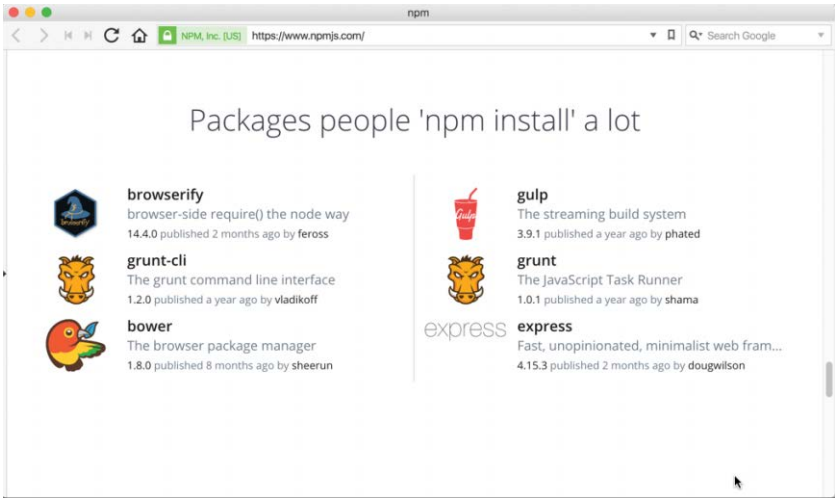


Figure 3.1 The official Node Package Modules website

## Using the Node Package Manager

The Node Package Manager you have already seen is a command-line utility. It allows you to find, install, remove, publish, and do everything else related to Node Package Modules. The Node Package Manager provides the link between the Node Package Registry and your development environment.

The simplest way to really explain the Node Package Manager is to list some of the command-line options and what they do. You use many of these options in the rest of this chapter and throughout the book. Table 3.1 lists the Node Package Manager commands.

Table 3.1 **`npm` command-line options (with `express` as the package, where appropriate)**

| Option | Description | Example |
|---|---|---|
| `search` | Finds module packages in the repository | `npm search express` |
| `install` | Installs a package either using a `package.json` file, from the repository, or a local location | `npm install`<br>`npm install express`<br>`npm install express@0.1.1`<br>`npm install ../tModule.tgz` |
| `install -g` | Installs a package globally | `npm install express -g` |

| Option | Description | Example |
|---|---|---|
| `remove` | Removes a module | `npm remove express` |
| `pack` | Packages the module defined by the `package.json` file into a .tgz file | `npm pack` |
| `view` | Displays module details | `npm view express` |
| `publish` | Publishes the module defined by a `package.json` file to the registry | `npm publish` |
| `unpublish` | Unpublishes a module you have published | `npm unpublish myModule` |
| `owner` | Allows you to add, remove, and list owners of a package in the repository | `npm add bdayley myModule`<br>`npm rm bdayley myModule`<br>`npm ls myModule` |

## Searching for Node Package Modules

You can also search for modules in the Node Package Registry directly from the command prompt using the `npm search <search_string>` command. For example, the following command searches for modules related to `openssl` and displays the results as shown in Figure 3.2:

```
npm search openssl
```

```
NAME                DESCRIPTION
bignum              Arbitrary-precision integer arithmetic using OpenSSL

certgen             Certificate generation library that uses the openssl command l
cipherpipe          Thin wrapper around openssl for encryption/decryption
csr                 Read csr file
csr-gen             Generates OpenSSL Certificate Signing Requests
dcrypt              extended openssl bindings
fixedentropy        ```js // V8 supports custom sources of entropy. // by default,

lockbox             Simple, strong encryption.

node-hardcoressl    HardcoreSSL is a package for obtaining low-level asynchronous
nrsa                OpenSSL's RSA encrypt/decrypt routines
openssl             openssl wrapper
openssl-wrapper     OpenSSL wrapper
rsa                 OpenSSL's RSA encrypt/decrypt routines
rsautl              A wrapper for OpenSSL's rsautl
selfsigned          Generate self signed certificates private and public keys
ssh-key-decrypt     Decrypt encrypted ssh private keys

ssl                 Verification of SSL certificates

ssl-keychain        OpenSSL Keychain and Key generation module
ssl-keygen          OpenSSL Key Generation module
ursa                RSA public/private key crypto

x509-keygen         node.js module to generate self-signed certificate via openssl
```

Figure 3.2 Searching for Node.js modules from the command prompt

## Installing Node Packaged Modules

To use a Node module in your applications, it must first be installed where Node can find it. To install a Node module, use the `npm install <module_name>` command. This downloads the Node module to your development environment and places it into the `node_modules` folder where the `install` command is run. For example, the following command installs the `express` module:

```
npm install express
```

The output of the `npm install` command displays the dependency hierarchy installed with the module. For example, the following code block shows part of the output from installing the `express` module.

```
C:\express\example
`-- express@4.14.0
  +-- accepts@1.3.3
  | +-- mime-types@2.1.11
  | | `-- mime-db@1.23.0
  | `-- negotiator@0.6.1
  +-- array-flatten@1.1.1
  +-- content-disposition@0.5.1
  +-- content-type@1.0.2
  +-- cookie@0.3.1
  +-- cookie-signature@1.0.6
  +-- debug@2.2.0
  | `-- ms@0.7.1 ...
```

The dependency hierarchy is listed; some of the methods Express requires are `cookie-signature`, `range-parser`, `debug`, `fresh`, `cookie`, and `send` modules. Each of these was downloaded during the install. Notice that the version of each dependency module is listed.

Node.js has to be able to handle dependency conflicts. For example, the `express` module requires `cookie 0.3.1`, but another module may require `cookie 0.3.0`. To handle this situation, a separate copy for the cookie module is placed in each module's folder under another `node_modules` folder.

To illustrate how modules are stored in a hierarchy, consider the following example of how `express` looks on disk. Notice that the `cookie` and `send` modules are located under the `express` module hierarchy, and that since the `send` module requires `mime` it is located under the `send` hierarchy:

```
./
./node_modules
./node_modules/express
./node_modules/express/node_modules/cookie
./node_modules/express/node_modules/send
./node_modules/express/node_modules/send/node_modules/mime
```

## Using `package.json`

All Node modules must include a `package.json` file in their root directory. The `package.json` file is a simple JSON text file that defines the module including dependencies. The `package.json` file can contain a number of different directives to tell the Node Package Manager how to handle the module.

The following is an example of a `package.json` file with a name, version, description, and dependencies:

```
{
    "name": "my_module",
    "version": "0.1.0",
    "description": "a simple node.js module",
    "dependencies" : {
        "express"   :  "latest"
    }
}
```

The only required directives in the `package.json` file are `name` and `version`. The rest depend on what you want to include. Table 3.2 describes the most common directives:

Table 3.2 **Directives used in the `package.json` file**

| Directive | Description | Example |
|---|---|---|
| `name` | Unique name of package. | `"name": "camelot"` |
| `preferGlobal` | Indicates this module prefers to be installed globally. | `"preferGlobal": true` |
| `version` | Version of the module. | `"version": 0.0.1` |
| `author` | Author of the project. | `"author": "arthur@???.com"` |
| `description` | Textual description of module. | `"description": "a silly place"` |
| `contributors` | Additional contributors to the module. | `"contributors": [ { "name": "gwen", "email": "gwen@???.com"}]` |
| `bin` | Binary to be installed globally with project. | `"bin: { "excalibur": "./bin/excalibur"}` |
| `scripts` | Specifies parameters that execute console apps when launching node. | `"scripts" { "start": "node ./bin/excalibur", "test": "echo testing"}` |
| `main` | Specifies the main entry point for the app. This can be a binary or a .js file. | `"main": "./bin/excalibur"` |

| Directive | Description | Example |
|---|---|---|
| `repository` | Specifies the repository type and location of the package. | `"repository": { "type": "git", "location": "http://???.com/c.git"}` |
| `keywords` | Specifies keywords that show up in the `npm` search. | `"keywords": [ "swallow", "unladen" ]` |
| `dependencies` | Modules and versions this module depends on. You can use the `*` and `x` wildcards. | `"dependencies": { "express": "latest", "connect": "2.x.x, "cookies": "*" }` |
| `engines` | Version of `node` this package works with. | `"engines": { "node": ">=6.5"}` |

A great way to use `package.json` files is to automatically download and install the dependencies for your Node.js app. All you need to do is create a `package.json` file in the root of your project code and add the necessary dependencies to it. For example, the following `package.json` requires the `express` module as a dependency.

```
{
    "name": "my_module",
    "version": "0.1.0",
    "dependencies" : {
        "express"  :  "latest"
    }
}
```

Then you run the following command from root of your package, and the `express` module is automatically installed.

```
npm install
```

Notice that no module is specified in the `npm install`. That is because `npm` looks for a `package.json` file by default. Later, as you need additional modules, all you need to do is add those to the `dependencies` directive and then run `npm install` again.

## Creating a Node.js Application

Now you have enough information to jump into a Node.js project and get your feet wet. In this section, you create your own Node Packaged Module and then use that module as a library in a Node.js application.

The code in this exercise is kept to a minimum so that you can see exactly how to create a package, publish it, and then use it again.

## Creating a Node.js Packaged Module

To create a Node.js Packaged Module you need to create the functionality in JavaScript, define the package using a `package.json` file, and then either publish it to the registry or package it for local use.

The following steps take you through the process of building a Node.js Packaged Module using an example called `censorify`. The `censorify` module accepts text and then replaces certain words with asterisks:

1. Create a project folder named `.../censorify`. This is the root of the package.
2. Inside that folder, create a file named `censortext.js`.
3. Add the code from Listing 3.1 to `censortext.js`. Most of the code is just basic JavaScript; however, note that lines 18–20 export the functions `censor()`, `addCensoredWord()`, and `getCensoredWords()`. The `exports.censor` is required for Node.js applications using this module to have access to the `censor()` function as well as the other two.

Listing 3.1 `censortext.js`: **Implementing a simple** `censor` **function and exporting it for other modules using the package**

```
01 var censoredWords = ["sad", "bad", "mad"];
02 var customCensoredWords = [];
03 function censor(inStr) {
04   for (idx in censoredWords) {
05     inStr = inStr.replace(censoredWords[idx], "****");
06   }
07   for (idx in customCensoredWords) {
08     inStr = inStr.replace(customCensoredWords[idx], "****");
09   }
10   return inStr;
11 }
12 function addCensoredWord(word){
13   customCensoredWords.push(word);
14 }
15 function getCensoredWords(){
16   return censoredWords.concat(customCensoredWords);
17 }
18 exports.censor = censor;
19 exports.addCensoredWord = addCensoredWord;
20 exports.getCensoredWords = getCensoredWords;
```

4. Once the module code is completed, you need to create a `package.json` file that is used to generate the Node.js Packaged Module. Create a `package.json` file in the

`.../censorify` folder. Then add contents similar to Listing 3.2. Specifically, you need to add the `name`, `version`, and `main` directives as a minimum. The `main` directive needs to be the name of the main JavaScript module that will be loaded, in this case `censortext`. Note that the .js is not required, Node.js automatically searches for the .js extension.

Listing 3.2 `package.json`: **Defining the Node.js module**

```
01 {
02   "author": "Brendan Dayley",
03   "name": "censorify",
04   "version": "0.1.1",
05   "description": "Censors words out of text",
06   "main": "censortext",
07   "dependencies": {},
08   "engines": {
09       "node": "*"
10   }
11 }
```

5. Create a file named `README.md` in the `.../censorify` folder. You can put whatever read me instructions you want in this file.
6. Navigate to the `.../censorify` folder in a console window and run the `npm pack` command to build a local package module.
7. The `npm pack` command creates a `censorify-0.1.1.tgz` file in the `.../censorify` folder. This is your first Node.js Packaged Module.

## Publishing a Node.js Packaged Module to the NPM Registry

In the previous section you created a local Node.js Packaged Module using the `npm pack` command. You can also publish that same module to the NPM repository at http://npmjs.com/.

When modules are published to the NPM registry, they are accessible to everyone using the NPM manager utility discussed earlier. This allows you to distribute your modules and applications to others more easily.

The following steps describe the process of publishing the module to the NPM registry. These steps assume that you have completed steps 1 through 5 from the previous section:

1. Create a public repository to contain the code for the module. Then push the contents of the `.../censorify` folder up to that location. The following is an example of a Github repository URL:

   `https://github.com/username/projectname/directoryName/ch03/censorify`

2. Create an account at https://npmjs.org/signup.
3. Use the `npm adduser` command from a console prompt to add the user you created to the environment.

4. Type in the username, password, and email that you used to create the account in step 2.
5. Modify the `package.json` file to include the new repository information and any keywords that you want made available in the registry search as shown in lines 7–14 in Listing 3.3.

Listing 3.3 `package.json`: **Defining the Node.js module that includes the repository and keywords information**

```
01 {
02   "author": "Brad Dayley",
03   "name": "censorify",
04   "version": "0.1.1",
05   "description": "Censors words out of text",
06   "main": "censortext",
07   "repository": {
08     "type": "git",
09     //"url": "Enter your github url"
10   },
11   "keywords": [
12     "censor",
13     "words"
14   ],
15   "dependencies": {},
16   "engines": {
17       "node": "*"
18   }
19 }
```

6. Publish the module using the following command from the `. . ./censor` folder in the console:

   ```
   npm publish
   ```

Once the package has been published you can search for it on the NPM registry and use the `npm install` command to install it into your environment.

To remove a package from the registry make sure that you have added a user with rights to the module to the environment using `npm adduser` and then execute the following command:

```
npm unpublish <project name>
```

For example, the following command unpublishes the `censorify` module:

```
npm unpublish censorify
```

In some instances you cannot unpublish the module without using the `--force` option. This option forces the removal and deletion of the module from the registry. For example:

```
npm unpublish censorify --force
```

## Using a Node.js Packaged Module in a Node.js Application

In the previous sections you learned how to create and publish a Node.js module. This section provides an example of actually using a Node.js module inside your Node.js applications. Node.js makes this simple: All you need to do is install the NPM into your application structure and then use the `require()` method to load the module.

The `require()` method accepts either an installed module name or a path to a .js file located on the file system. For example:

```
require("censorify")
require("./lib/utils.js")
```

The .js filename extension is optional. If it is omitted, Node.js searches for it.

The following steps take you through that process so you can see how easy it is:

1. Create a project folder named `.../readwords`.
2. From a console prompt inside the `.../readwords` folder, use the following command to install the `censorify` module from the `censorify-0.1.1.tgz` package you created earlier:

   ```
   npm install .../censorify/censorify-0.1.1.tgz
   ```

3. Or if you have published the `censorify` module, you can use the standard command to download and install it from the NPM registry:

   ```
   npm install censorify
   ```

4. Verify that a folder named `node_modules` is created along with a subfolder named `censorify`.
5. Create a file named `.../readwords/readwords.js`.
6. Add the contents shown in Listing 3.4 to the `readwords.js` file. Notice that a `require()` call loads the `censorify` module and assigns it to the variable `censor`. Then the `censor` variable can be used to invoke the `getCensoredWords()`, `addCensoredWords()`, and `censor()` functions from the `censorify` module.

Listing 3.4 `readwords.js`: **Loading the** `censorify` **module when displaying text**

```
1 var censor = require("censorify");
2 console.log(censor.getCensoredWords());
3 console.log(censor.censor("Some very sad, bad and mad text."));
4 censor.addCensoredWord("gloomy");
5 console.log(censor.getCensoredWords());
6 console.log(censor.censor("A very gloomy day."));
```

7. Run the `readwords.js` application using the `node readwords.js` command and view the output shown in the following code block. Notice that the censored words are

replaced with **** and that the new censored word gloomy is added to the censorify module instance censor.

```
C:\nodeCode\ch03\readwords>node readwords
[ 'sad', 'bad', 'mad' ]
Some very *****, ***** and ***** text.
[ 'sad', 'bad', 'mad', 'gloomy' ]
A very *** day.
```

## Writing Data to the Console

One of the most useful modules in Node.js during the development process is the console module. This module provides a lot of functionality when writing debug and information statements to the console. The console module allows you to control output to the console, implement time delta output, and write tracebacks and assertions to the console. This section covers using the console module because you need to know it for subsequent chapters in the book.

Because the console module is so widely used, you do not need to load it into your modules using a require() statement. You simply call the console function using console.<*function*>(<*parameters*>). Table 3.3 lists the functions available in the console module.

Table 3.3 **Member functions of the `console` module**

| Function | Description |
|---|---|
| log([data],[...]) | Writes data output to the console. The data variable can be a string or an object that can be resolved to a string. Additional parameters can also be sent. For example:<br>`console.log("There are %d items", 5);`<br>`>>There are 5 items` |
| info([data],[...]) | Same as console.log. |
| error([data],[...]) | Same as console.log; however, the output is also sent to stderr. |
| warn([data],[...]) | Same as console.error. |
| dir(obj) | Writes out a string representation of a JavaScript object to the console. For example:<br>`console.dir({name:"Brad", role:"Author"});`<br>`>> { name: 'Brad', role: 'Author' }` |
| time(label) | Assigns a current timestamp with ms precision to the string label. |

| Function | Description |
|---|---|
| `timeEnd(label)` | Creates a delta between the current time and the timestamp assigned to `label` and outputs the results. For example:<br>`console.time("FileWrite");`<br>`f.write(data); //takes about 500ms`<br>`console.timeEnd("FileWrite");`<br>`>> FileWrite: 500ms` |
| `trace(label)` | Writes out a stack trace of the current position in code to `stderr`. For example:<br>`module.trace("traceMark");`<br>`>>Trace: traceMark`<br>`at Object.<anonymous> (C:\test.js:24:9)`<br>`at Module._compile (module.js:456:26)`<br>`at Object.Module._ext.js (module.js:474:10)`<br>`at Module.load (module.js:356:32)`<br>`at Function.Module._load (module.js:312:12)`<br>`at Function.Module.runMain(module.js:497:10)`<br>`at startup (node.js:119:16)`<br>`at node.js:901:3` |
| `assert(expression, [message])` | Writes the message and stack trace to the console if expression evaluates to `false`. |

## Summary

This chapter focused on getting you up to speed on the Node.js environment. Node.js Packaged Modules provide the functionality that Node.js does not inherently come with. You can download these modules from the NPM registry, or you can even create and publish your own. The `package.json` file provides the configuration and definition for every Node.js module.

The examples in this chapter covered creating, publishing, and installing your own Node.js Packaged Modules. You learned how to use the NPM to package a local module as well as publish one to the NPM registry. You then learned how to install the Node.js modules and use them in your own Node.js applications.

## Next

The next chapter covers the event-driven nature of Node.js. You see how events work in the Node.js environment and learn how to control, manipulate, and use them in your applications.

# Index

## Symbols


+ (addition) operator, 17
&& (and) operator, 19
= assignment operator, 18
+= assignment operator, 18
-= assignment operator, 18
/= assignment operator, 18
*= assignment operator, 18
%= assignment operator, 18
` (backquotes), 404–405
{ } (curly braces), 20, 24, 415, 416, 429
/ (division) operator, 18
$ (dollar sign), 243
== (equal) operator, 19
=== (equal) operator, 19
\' escape code, 29
\" escape code, 29
\\ escape code, 29
> (greater than) operator, 19
>= (greater than or equal to) operator, 19
++ (increment) operator, 18
< (less than) operator, 19
<= (less than or equal to) operator, 19
% (modulus) operator, 18
* (multiplication) operator, 18
! (not) operator, 19
!= (not equal) operator, 19
!== (not equal) operator, 19
|| (or) operator, 19

**( ) (parentheses), 20, 24, 436, 457**
**| (pipe symbol), 426**
**- (subtraction) operator, 18**
**~ (tilde), 404–405**

## A

**abort() method, 120, 161**
**acceptsCharset property (Request object), 351**
**accessing MongoDB**
access control, 209
authentication, 210–211
Database Administrator accounts, 211
User Administrator accounts, 209–210
from shell client, 203–204
command parameters, 205
shell commands, 204
shell methods, 205
shell scripts, 205–206
**accounts (MongoDB)**
Database Administrator accounts, 211
user accounts
creating, 206–207
listing, 206–207
removing, 209
roles, 208
User Administrator accounts, 209–210
**ActivatedRoute, importing, 488**
**$add operator, 286**
**addCensoredWords() function, 52**
**addClass() method, 446**
**$addFields operator, 285**
**addition (+) operator, 17**
**addListener() function, 65**
**AddNums interface, 385**
**address() method, 143, 146**
**addShard() method, 339**
**addShardTag() method, 340**
**addTagRange() method, 340**
**$addToSet operator, 243, 286**
**addTrailers() method, 122**
**addUser() method, 228, 229**
**admin() method, 227, 229**
**Admin object, 229**
**aggregate() method, 283, 301, 317**
**aggregating results, 282–289**
aggregate() method, 283
aggregation examples, 287–289
aggregation expression operators, 285–287
aggregation framework operators, 283–285
**aggregation expression operators, 285–287**
**aggregation framework operators, 283–285**
**AJAX (Asynchronous JavaScript and XML) requests, 9**
**alerts, asynchronous, 516**
**all() method, 304, 346**
**$all operator, 264**
**allowDrop() method, 536**
**allowHalfOpen option, 86, 145**
**and() method, 303**
**and (&&) operator, 19**
**$and operator, 263**
**Angular. *See also* Node.js-to-Angular stack**
adding to environment, 393–394
advantages of, 4, 13–14, 391
applications, creating, 395–396, 398–402
Angular bootstrapper, 399–402
animation application, 525–529
component modules, importing, 396–397

drag-and-drop application, 533–539
NgModule, 397–398
star rating application, 539–546
zoom application, 530–533
bootstrapper, 398–402
app.module.js, 401
app.module.ts, 400
first.component.js, 402
first.component.ts, 400
first.html listing, 399
browser events, 457–458
built-in directives, 441–442
attribute directives, 392, 445–448
components directives, 392, 442
structural directives, 392, 442–445
change detection, 465–468
CLI (command-line interface), 394–395
components, 396–397, 539–546
Angular bootstrapper components, 398–402
animation application components, 526–529
collections, 204
configuration, 403–404
constructors, 407
custom component directives, 452–456
deleting data in, 461–464
dependency injection, 392–393, 410–414
drag-and-drop application components, 534–538
emitting custom events from, 458
image zoom application components, 530–532
importing, 396–397
inline CSS and HTML in, 405–406
integrating custom services into, 503–504
nested components, 458–460
NgModule, 397–398
selectors, 404
separation of responsibilities, 393
star rating application components, 539–546
templates, 404–405, 408–410
custom directives, 449
custom attribute directives, 449–452
custom component directives, 452–456
custom events, 458
deleting data with, 461–464
emitting from components, 458
handling with listeners, 458
implementing in nested components, 458–460
data binding, 429
attribute binding, 433
class binding, 433–434
definition of, 392
event binding, 436–439
interpolation, 430–431
property binding, 431–433
style binding, 435–436
two-way binding, 439–440
definition of, 4
dependency injection, 392–393
event binding, 436–439
expressions, 415–416
basic expressions, 416–417
built-in pipes, 422–426
Component class interaction, 418–419
custom pipes, 426–428

pipe implementation, 422
TypeScript in, 419–422
modules, 392
observables
creating, 464–465
definition of, 464
detecting data changes with, 465–468
separation of responsibilities, 393
services
animate, 470
animation service, application using, 525–529
area-calc.service.ts, 506–510
constant data service, 505–506
definition of, 393
forms, 470
http, 470–486
integrating into applications, 503–504
mock back-end service, 540–541
purpose of, 469
ratings service, 541–542
router, 470, 487–501
service that returns a promise, 515–516
shared service, 516–523
use cases for, 503–504
UserService, 483–484
variable data service, 510–514
TypeScript, 383
in Angular expressions, 419–422
classes, 386
data types, 383–384
directives, 462–463
functions, 388–389
inheritance, 387
interfaces, 385–386
modules, 387–388

**Angular QuickStart website, 394**
**animate service, 470**
**animated component**
animated.component.css, 529
animated.component.html, 529
animated.component.ts, 527–528
**animation application**
animated.component.css, 529
animated.component.html, 529
animated.component.ts, 527–528
app.component.html, 526–527
app.component.ts, 526
app.module.ts, 525–526
folder structure, 525
**anonymous functions, 25–26**
**any data type, 384**
**app.component.css listing**
AreaCalcService, 509
custom directive with component, 454
drag-and-drop application, 535–536
RandomImageService, 513–514
router with navigation bar, 495
star rating application, 544
**app.component.html listing**
animation application, 526–527
AreaCalcService, 508–509
custom directive with component, 453–454
drag-and-drop application, 535
image zoom application, 531
PiService, 506
PromiseService, 516
RandomImageService, 513
router with navigation bar, 494
SharedService, 519
star rating application, 543–544
zoom directive, 451–452

**app.component.ts listing**
animation application, 526
AreaCalcService, 507–508
custom directive with component, 453
drag-and-drop application, 534–535
image zoom application, 530–531
PiService, 505
PromiseService, 515–516
RandomImageService, 512–513
router with navigation bar, 494
router with parameters, 498
SharedService, 519
simple router application, 490
star rating application, 543
**append() method, 318**
**applications (Angular), 395–396, 398–402.** ***See also*** **components (Angular)**
Angular bootstrapper
app.module.js, 401
app.module.ts, 400
first.component.js, 402
first.component.ts, 400
first.html, 399
animation application
animated.component.css, 529
animated.component.html, 529
animated.component.ts, 527–528
app.component.html, 526–527
app.component.ts, 526
app.module.ts, 525–526
folder structure, 525
component modules, importing, 396–397
drag-and-drop application
app.component.css, 535–536
app.component.html, 535
app.component.ts, 534–535
drag.component.css, 538
drag.component.html, 538
drag.component.ts, 538
drop.component.css, 537
drop.component.html, 537
drop.component.ts, 536
folder structure, 533–534
NgModule, 397–398
star rating application
app.component.css, 544
app.component.html, 543–544
app.component.ts, 543
app.module.ts, 540
folder structure, 539
mockbackend.service.ts, 540–541
rated-item.component.css, 545–546
rated-item.component.html, 545
rated-item.component.ts, 544–545
ratings.service.ts, 541–542
zoom application
app.component.html, 531
app.component.ts, 530–531
folder structure, 530
zoomit.component.css, 532
zoomit.component.html, 532
zoomit.component.ts, 531–532
**applications (Node.js), censorify module**
censortext.js, 49
creating, 49–50
loading into Node.js applications, 52–53
package.json, 50, 51
publishing to NPM Registry, 50–51
readwords.js, 52
**app.module.js listing, 401**
**app.module.ts listing**
Angular bootstrapper, 400
animation application, 525–526

simple mock server implementation, 473–474, 480
simple router implementation, 488–489
star rating application, 540
**AppRoutingModule, importing, 488**
**app-routing.module.ts listing**
router with navigation bar, 492–493
router with parameters, 498
simple router implementation, 489–490
**arbiter servers, 331**
**arch() method, 162, 182**
**AreaCalcService, 506–510**
app.component.css, 509
app.component.html, 508–509
app.component.ts, 507–508
area-calc.service.ts, 506–507
**area-calc.service.ts listing, 506–507**
**argv method, 162**
**arithmetic operators, 17–18**
**Array object, 32–33**
adding items to, 34–35
combining, 33–34
converting into strings, 34
iterating through, 34
manipulating, 32–33
methods, 32–33
removing items from, 34–35
searching, 34
**arrays, 16**
JavaScript
adding items to, 34–35
combining, 33–34
converting into strings, 34
iterating through, 34
manipulating, 32–33
methods, 32–33
removing items from, 34–35
searching, 34
Routes, defining, 486–487
TypeScript, 384
**assert() function, 54**
**assertion testing module, 40, 190**
**assigning Express middleware**
globally to path, 368
to single route, 368
**assignment operators, 18**
**async pipe, 424**
**asynchronous alerts, 516**
**asynchronous file reading, 104–105**
**asynchronous file system calls, 95**
**asynchronous file writing, 99–101**
**Asynchronous JavaScript and XML (AJAX), 9**
**atomic write operations, 198**
**atomically modifying documents, 250–251**
**attachment() method, 353**
**attribute binding, 433**
**attribute directives, 392, 445–448**
attribute.component.css, 448
attribute.component.html, 447
attribute.component.ts, 446–447
custom attribute directives, 449–452
definition of, 441
ngForm, 445
ngModel, 445
ngStyle, 445
**attribute.component.css listing, 448**
**attribute.component.html listing, 447**
**attribute.component.ts listing, 446–447**
**–auth parameter (mongod command), 203**
**auth property (URL object), 116**
**authenticate() method, 228, 229**

**authentication**
HTTP, 375–377
MongoDB, 210–211
session, 377–380
**author directive, 47**
**auto_reconnect option (server object), 224**
**autoIndex option (Schema object), 295**
**autoIndexID option (collections), 214**
**$avg operator, 286**

## B

**\b (backspace) escape code, 30**
**backend services, 10**
**backing up MongoDB, 341–342**
**backquotes (`), 404–405**
**backspace escape code, 30**
**Bad Guys component**
badguys.component.css, 522
badguys.component.html, 522
badguys.component.ts, 521
**base64 encoding, 75**
**basicAuth middleware, 368, 375–377**
**basicExpressions.component.ts listing, 417**
**big endian, 75**
**bin directives, 47**
**–bind parameter (mongod command), 203**
**binding. *See* data binding**
**$bit operator, 244**
**blocking I/O, 57–58**
**bodyParser middleware, 368, 371–372**
**Boolean data type**
JavaScript, 16
TypeScript, 383
**bootstrap metadata option (NgModule), 397**
**bootstrapper (Angular), 398–402**
app.module.js, 401
app.module.ts, 400
first.component.js, 402
first.component.ts, 400
first.html listing, 399
**border() function, 450**
**brackets ({ }), 415, 416**
**break statement, 23–24**
**browser events, 457–458**
**browser view, rendering, 9–10**
**BrowserAnimationsModule, loading, 526**
**browsers, 8**
browser events, 457–458
browser view, rendering, 9–10
browser-to-webserver communication, 8–9
user interaction, 10
**browser-to-webserver communication, 8–9**
**BSON, 192**
**$bucket operator, 285**
**$bucketAuto operator, 285**
**buffer module. *See* buffers**
**buffer_concat.js listing, 81**
**buffer_copy.js listing, 79**
**buffer_read.js listing, 78**
**buffer_slice.js listing, 80**
**buffer_write.js listing, 77**
**bufferCommands option (Schema object), 295**
**buffers, 74–75**
compressing/decompressing, 91–92
concatenating, 81
copying, 79–80
creating, 75–76
determining length of, 78
encoding methods, 75
reading from, 77–78
slicing, 80
writing to, 76–77

**bufferSize property (Socket object), 143**
**building templates, 404–405**
**built-in directives, 441–442**
attribute directives, 392, 445–448
attribute.component.css, 448
attribute.component.html, 447
attribute.component.ts, 446–447
definition of, 441
ngForm, 445
ngModel, 445
ngStyle, 445
components directives, 392, 441, 442
structural directives, 392, 442–445
definition of, 441
ngFor, 442, 474, 479, 484, 513
ngIf, 442–443
ngSwitch, 442–443
ngSwitchCase, 442, 443
ngSwitchDefault, 442
structural.component.ts, 443–444
**built-in events, 457–458**
**built-in pipes**
builtInPipes.component.ts, 425
table of, 422–424
**built-in services**
animate, 470
forms, 470
http, 470
GET requests, sending, 470–471
JSON file implementation and access, 472–475
PUT requests, sending, 470–471
request configuration, 471
response callback functions, 471
simple mock server implementation, 475–481
simple mock server updates, 481–486
router, 470
ActivatedRoute, importing, 488
route navigation, 488
route object parameters, 487
Router, importing, 488
router with navigation bar, 492–497
router with parameters, 497–501
Routes array, 486–487
routing module, including, 488
simple router implementation, 488–491
**builtInPipes.component.ts listing, 425**
**byteLength() method, 78**
**bytesRead property (Socket object), 144**
**bytesWritten property (Socket object), 144**

## C

**ca option**
htp.createServer(), 136
https.request(), 136
tls.connect(), 154
tls.createServer(), 155
**callback functions, 67**
applying for defined parameters, 348–349
chaining, 70
implementing closure in, 68–69
passing parameters to, 67–68
**callback_chain.js listing, 70**
**callback_closure.js listing, 69**
**callback_parameter.js listing, 67–68**
**canActivate property (route object), 487**
**canActivateChild property (route object), 487**
**canDeactivate property (route object), 487**
**canLoad property (route object), 487**
**capped collections, 197–198, 330**

**capped option**
collections, 214
Schema object, 295
**carriage return escape code, 29**
**Cascading Style Sheets. *See* CSS (Cascading Style Sheets)**
**case sensitive routing setting (Express), 344**
**catch() method, 471**
**catch statement, 35–36**
**C/C++ add-ons, 40, 190**
**censor() function, 52**
**censorify module**
censortext.js, 49
creating, 49–50
loading into Node.js applications, 52–53
package.json, 50, 51
publishing to NPM Registry, 50–51
readwords.js, 52
**censortext.js listing, 49**
**cert option**
htp.createServer(), 136
https.request(), 136
tls.connect(), 154
**certificate class, 188**
**chaining callback functions, 70**
**change detection, 465–468**
**(change) event, 457**
**changeLoop() method, 510**
**changeSize() function, 450**
**character.component.css listing, 462**
**character.component.html listing, 462**
**character.component.ts listing, 461–462**
**charAt() method, 30**
**charCodeAt() method, 30**
**charObservable, 517**
**chdir() method, 162**
**checkContinue event, 124**
**checkGoal() function, 66**
**checkKeys option, 242**
**child components, deleting parent data from, 461–464**
**child forks, 171–173**
**child processes, 159–160**
child forks, 171–173
ChildProcess object, 164–166
executable files, executing on another process, 168–169
processes, spawning, 169–171
system command, executing on another process, 166–168
**child_exec.js listing, 167–168**
**child_fork.js listing, 172–173**
**child_process module, 40, 159–160**
child forks, 171–173
ChildProcess object, 164–166
executable files, executing on another process, 168–169
processes, spawning, 169–171
system command, executing on another process, 166–168
**child_process_exec_file.js listing, 168–169**
**child_process_spawn_file.js listing, 170–171**
**ChildProcess object, 164–166**
**Children property (route object), 487**
**cipher class, 188**
**ciphers option**
htp.createServer(), 136
https.request(), 136
tls.createServer(), 155
**class binding, 433–434**
**class.component.ts listing, 434**

**classes. *See also* services (Angular)**
binding, 433–434
certificate, 188
cipher, 188
Component, 418–419
decipher, 188
defining, 386
diffieHellman, 188
@directive, 449
eCDH, 188
EventEmitter, 458
export classes
Directive, 449
ZoomDirective, 451
hash, 189
hmac, 189
inheritance, 387
MongoClient, 222
pipe, 426
SecretAgent, 387
sign, 189
verify, 189
**classExpressions.component.ts listing, 418–419**
**clearCookie() method, 373**
**clearImmediate() function, 62**
**clearInterval() function, 61**
**clearTimeout() function, 60**
**CLI (command-line interface), 394–395. *See also* commands**
**(click) event, 457**
**clientError event, 124, 156**
**ClientRequest object, 118–121**
**clients**
HTTPS clients, 135–136
TCP socket clients, 147–150
TLS socket clients, 153–154
**client-side scripts, 9**
**close event, 82, 121, 123, 124, 142, 145, 165**
**close() method, 97, 125, 146, 227, 233**
**closeSync() method, 97**
**closing files, 96–97**
**closure in callback functions, 68–69**
**cluster module, 40, 174**
events, 174
HTTP clusters, 176–179
methods, 175
properties, 175
Worker object, 175–176
**cluster_server.js listing, 177–179**
**clusterAdmin role (MongoDB), 208**
**clusters**
cluster module, 40, 174
events, 174
HTTP clusters, 176–179
methods, 175
properties, 175
Worker object, 175–176
sharded MongoDB clusters, 338
adding shards to cluster, 339
config server database instances, 338
query router servers, 338–339
**code listings. *See* listings**
**collection() method, 228, 230**
**Collection object, 229–232. *See also* collections (MongoDB)**
**collection option (Schema object), 295**
**collection_create_list_delete.js listing, 238–239**
**collection_stat.js listing, 239–240**
**collectionInfo() method, 227**
**collectionNames() method, 227**

**collections (MongoDB)**
capped collections, 197–198, 330
Collection object, 229–232
collection_create_list_delete.js application example, 238–239
creating, 214–215, 237
definition of, 192
deleting, 215–216, 238
displaying list of, 214
documents
adding, 217, 244–246, 307–309
aggregating with Mongoose, 317–320
atomically modifying, 250–251
counting, 268–269
Document object, 304–305
finding, 216, 265–268, 305–307
removing, 217–218, 255–258, 314–317
retrieving, 246–248
saving, 252–253, 310–311
updating, 218–219, 248–250, 309–314
upserting, 253–254
listing, 237
number of, 199
sharding on, 340
statistics for, 239–240
**collections command, 204**
**collections() method, 228, 237**
**$collStatus operator, 284**
**combining**
arrays, 33–34
strings, 31
**command-line interface, 394–395.** ***See also*** **commands**
**command-line options module, 40**
**commands.** ***See also*** **directives**
databases, 204
db.help, 204
dbs, 204
executing in another process, 166–168
exit, 204
help, 204
mongod, 202–203
mongodump, 342
ng eject, 395
ng generate component, 395
ng generate directive, 395
ng generate enum, 395
ng generate guard, 395
ng generate interface, 395
ng generate module, 395
ng generate pipe, 395
ng generate service, 395
ng new, 395
ng serve, 395
npm, 42, 44–45
npm adduser, 50
npm install, 46, 51
npm pack, 50
npm search, 45
npm install, 475
profile, 204
roles, 204
show, 204
use, 204
users, 204
**comment() method, 302**
**comment option (options object), 265**
**communication, browser-to-webserver, 8–9**
**comparison operators, 19–20**
**compatibility of Angular, 14**

**compiling models, 298**
**Component class, 418–419**
**component directives, 441**
**component property (route object), 487**
**components (Angular), 396–397, 539–546.** ***See also*** **services (Angular)**
Angular bootstrapper, 398–402
animation application, 526–527
animated.component.css, 529
animated.Component.html, 529
animated.component.ts, 527–528
app.component.ts, 526
app.module.ts, 525–526
folder structure, 525
collections, 204
configuration, 403–404
constructors, 407
custom component directives, 452–456
container component, 454
CSS for container component, 455
CSS for root component, 454
HTML for container component, 455
HTML for root component, 453–454
root component, 453
deleting data in, 461–464
dependency injection
building nested components with, 410–412
definition of, 392–393, 410
passing data with, 412–413
sample application with inputs, 413–414
drag-and-drop application
app.component.css, 535–536
app.component.html, 535
app.component.ts, 534–535
drag.component.css, 538
drag.component.html, 538
drag.component.ts, 538
drop.component.css, 537
drop.component.html, 537
drop.component.ts, 536
folder structure, 533–534
emitting custom events from, 458
image zoom application
app.component.html, 531
app.component.ts, 530–531
folder structure, 530
zoomit.component.css, 532
zoomit.component.html, 532
zoomit.component.ts, 531–532
importing, 396–397
inline CSS and HTML in, 405–406
integrating custom services into, 503–504
nested components, 458–460
NgModule, 397–398
selectors, defining, 404
separation of responsibilities, 393
star rating application
app.component.css, 544
app.component.html, 543–544
app.component.ts, 543
app.module.ts, 540
folder structure, 539
mockbackend.service.ts, 540–541
rated-item.component.css, 545–546
rated-item.component.html, 545
rated-item.component.ts, 544–545
ratings.service.ts, 541–542
templates
building, 404–405
external templates, 408–410

**components directives, 392, 442**
**compound indexes, 328**
**compress middleware, 368**
**compressing**
buffers, 91–92
streams, 92–93
**concat() method, 30, 31, 33–34, 81**
**$concat operator, 287**
**concatenating buffers, 81**
**conditionals**
if statements, 20
switch statements, 20–21
**config method, 161**
**–config parameter (mongod command), 202**
**config servers, creating, 338**
**configuration**
components, 403–404
selectors, 404
templates, 404–405
Express, 343–344
HTTP requests, 471
MongoDB authentication, 210–211
query database operation, 299–302
**connect event, 120, 124, 141**
**connect() method, 56, 140, 147, 154, 223, 292**
**connected property (ChildProcess object), 166**
**connecting to MongoDB**
MongoClient class, 222
MongoClient object, 223–226
Mongoose, 292–294
write concern, 222
**connection event, 124, 145**
**Connection object, 292–293**
**connectionTimeOut option (server object), 224**
**console, writing data to, 53–54**
**console module, 40, 53–54**
**constant data service, 505–506**
app.component.html, 506
app.component.ts, 505
pi.service.ts, 505
**constructor.component.ts listing, 407**
**constructors, 407**
**container component, 454**
CSS for, 455
HTML for, 455
**container directive**
container.component.css, 455
container.component.html, 455
container.component.ts, 454
**content, generating with CLI (command-line interface), 394–395**
**continue event, 120**
**continue statement, 23–24**
**contributors directive, 47**
**converting**
arrays to strings, 34
JavaScript objects to JSON, 74
JSON to JavaScript objects, 74
objects to strings, 184–185
**cookie() method, 373, 378**
**cookieParser middleware, 368, 373–374**
**cookies**
cookie sessions, 374–375
sending/receiving, 373–374
**cookieSession middleware, 368, 374–375**
**copy() method, 79**
**copyDatabase() method, 213**
**copying**
buffers, 79–80
databases, 213

**count() method, 231, 233, 268–269, 300**
**$count operator, 285**
**counting documents, 268–269**
**cpus() method, 182**
**create() method, 300, 307**
**createAlert() method, 515**
**createCollection() method, 214, 228, 234, 237, 330**
**createConnection() method, 140**
**createDb() method, 481**
**createDelete.component.CSS listing, 479–480**
**createDelete.component.html listing, 479**
**createDelete.component.ts listing, 477–478**
**createReadStream() method, 105–106**
**createServer() method, 124, 126, 128, 136, 150, 154–156, 345**
**createTimedAlert() method, 515**
**createUser() method, 207, 208, 476, 478**
**createWriteStream() method, 101–102**
**crl option**
htp.createServer(), 136
https.request(), 136
tls.createServer(), 155
**crypto module, 41, 188–190**
**csrf middleware, 368**
**CSS (Cascading Style Sheets)**
files, 9
inline CSS in Angular applications, 405–406
listings
app.component.css, 454, 495, 513–514, 535–536, 544
attribute.component.css, 448
character.component.css, 462
container.component.css, 455
details.component.css, 463
drag.component.css, 538
drop.component.css, 537
external.css, 409
http.component.CSS, 474–475
outer.css, 411
rated-item.component.css, 545–546
./static/css/static.css, 370
update.component.CSS, 485–486
zoomit.component.css, 532
**curly braces ({ }), 20, 24, 429**
**currency pipe, 423**
**current database, changing, 211–212**
**Cursor object, 232–233**
**custom Angular services, 503**
constant data service, 505–506
app.component.html, 506
app.component.ts, 505
pi.service.ts, 505
data transform service, 506–510
app.component.css, 509
app.component.html, 508–509
app.component.ts, 507–508
area-calc.service.ts, 506–507
integrating into applications, 503–504
mock back-end service, 540–541
ratings service, 541–542
service that returns a promise, 515–516
shared service, 516–523
app.component.html, 519
app.component.ts, 519
badguys.component.css, 522
badguys.component.html, 522
badguys.component.ts, 521
good-guys.component.css, 521
good-guys.component.html, 520
good-guys.component.ts, 520
shared.service.ts, 517–518

use cases for, 503–504
variable data service, 510–514
app.component.css, 513–514
app.component.html, 513
app.component.ts, 512–513
random-image.service.ts, 511–512
**custom directives, 449**
custom attribute directives, 449–452
custom component directives, 452–456
container component, 454
CSS for container component, 455
CSS for root component, 454
HTML for container component, 455
HTML for root component, 453–454
root component, 453
**custom events, 64–65, 457**
deleting data with, 461–464
emitting from components, 458
handling with listeners, 458
implementing in nested components, 458–460
**custom middleware, 380–381**
**custom pipes, 426–428**
**custom-defined objects, 28**
**customevent.component.html listing, 459–460**
**customevent.component.ts listing, 459**
**customPipes.component.ts listing, 427**
**custom.pipe.ts listing, 427**
**cwd() method, 162**
**cwd property**
exec() and execFile() methods, 166
fork() function, 172
spawn() function, 170

## D

**Dahl, Ryan, 39**
**data binding, 429**
attribute binding, 433
class binding, 433–434
definition of, 392
event binding, 436–439
interpolation, 430–431
property binding, 431–433
style binding, 435–436
two-way binding, 439–440
**data changes, detecting with observables, 465–468**
**data denormalization, 196–197**
**data event, 82, 141**
**data I/O**
buffers, 74–75
compressing/decompressing, 91–92
concatenating, 81
copying, 79–80
creating, 75–76
determining length of, 78
encoding methods, 75
reading from, 77–78
slicing, 80
writing to, 76–77
JSON (JavaScript Object Notation), 73
converting JavaScript objects to, 74
converting to JavaScript objects, 74
streams, 81
compressing/decompressing, 92–93
Duplex streams, 86–88
piping, 89–90
Readable streams, 82–84, 89–90
Transform streams, 88–89
Writable streams, 84–86, 89–90

**data life cycles, 199**
**data model, planning, 194–195**
**data normalization, 195–196**
**data passing with dependency injection, 412–413**
**Data property (route object), 487**
**data transform service, 506–510**
app.component.css, 509
app.component.html, 508–509
app.component.ts, 507–508
area-calc.service.ts, 506–507
**data types**
JavaScript, 16–17
MongoDB, 193–194
TypeScript, 383–384
**data usability, 200**
**Database Administrator accounts, 211**
**databases (MongoDB). *See also* collections (MongoDB)**
changing current, 211–212
connecting to using Mongoose, 292–294
copying, 213
creating, 212, 234
database change options, 241–242
db_create_list_delete.js sample application, 234–236
deleting, 212–213, 234
displaying list of, 211
implementing schemas on, 295–296
listing, 233
query database operation
options, 301–302
setting, 299–301
repairing, 341
update operators, 242–244
**databases command, 204**
**data.service.ts listing, 476, 481–482**
**date pipe, 423**
**db() method, 227**
**Db object, 227–228**
**db_connect_object.js listing, 226**
**db_connect_url.js listing, 225**
**db_create_list_delete.js listing, 234–236**
**db_status.js listing, 236–237**
**dbAdmin role (MongoDB), 208**
**dbAdminAnyDatabase role (MongoDB), 208**
**db.auth() method, 205**
**db.help command, 204**
**–dbpath parameter (mongod command), 203**
**dbs command, 204**
**debounce() method, 471**
**debugger module, 41**
**decipher class, 188**
**declarations metadata, 397, 410**
**decompressing**
buffers, 91–92
streams, 92–93
**decorators**
@NgModule, 397–398, 410
@pipe, 426
**defined parameters, applying route parameters with, 348**
**deflate() method, 91**
**deflateRaw() method, 91**
**delaying work**
interval timers, 61–62
timeout timers, 60–61
**delete() method, 470**
**deleteChar() method, 461**
**deleteUser() method, 476, 478**
**deleting data, 461–464**
array items, 34–35
collections, 215–216, 238
databases, 212–213

directories, 111–112
documents, 217–218
findAndRemove() method, 257–258
with Mongoose, 314–317
remove() method, 217–218, 255–256
event listeners, 65
files, 110
MongoDB user accounts, 209
**denormalizing data, 196–197**
**dependencies directive, 48**
**dependency injection**
building nested components with, 410–412
definition of, 392–393, 410
passing data with, 412–413
sample application with inputs, 413–414
**deploying**
replica sets, 333–334
sharded MongoDB clusters, 338
adding shards to cluster, 339
config server database instances, 338
query router servers, 338–339
**dereferencing timers, 63**
**description directive, 47**
**destroy() method, 142, 378**
**detached property (spawn() function), 170**
**details.component.css listing, 463**
**details.component.html listing, 463**
**details.component.ts listing, 462–463**
**detecting data changes, 465–468**
**diffieHellman class, 188**
**dir() function, 53**
**@directive class, 449**
**Directive class, 449**
**directives. *See also* commands**
author, 47
bin, 47
built-in directives, 441–442
attribute directives, 392, 445–448
components directives, 392, 442
structural directives, 392, 442–445
contributors, 47
custom directives, 449
custom attribute directives, 449–452
custom component directives, 452–456
definition of, 441
dependencies, 48
description, 47
engines, 48
keywords, 48
main, 47
name, 47
preferGlobal, 47
repository, 48
scripts, 47
version, 47
**directories**
creating, 111–112
deleting, 111–112
node_modules, 221
renaming, 112
**disable() method, 344**
**disabled() method, 344**
**disconnect event, 165, 174, 176**
**disconnect() method, 165, 175, 176, 292**
**distinct field values, finding, 276–277**
**distinct() method, 231, 276–277, 300**
**$divide operator, 286**
**division (/) operator, 18**

**dns module, 41, 186–188**
**dns_lookup.js listing, 187–188**
**doc_aggregate.js listing, 288**
**doc_count.js listing, 268–269**
**doc_delete_one.js listing, 257–258**
**doc_delete.js listing, 255–256**
**doc_distinct.js listing, 277**
**doc_fields.js listing, 271–272**
**doc_find.js listing, 247–248**
**doc_group.js listing, 279–280**
**doc_insert.js listing, 245–246**
**doc_limit.js listing, 270–271**
**doc_modify.js listing, 251**
**doc_paging.js listing, 273–274**
**doc_query.js listing, 266–268**
**doc_save.js listing, 252–253**
**doc_sort.js listing, 275–276**
**doc_update.js listing, 249–250**
**doc_upsert.js listing, 253–254**
**doCalc() function, 26**
**doCircle() method, 507**
**Document object, 292, 304–305**
**Document Object Model. *See* DOM (Document Object Model)**
**documents (MongoDB)**
- adding
  - insert() method, 217, 244–246
  - with Mongoose, 307–309
- aggregating with Mongoose, 317–320
- atomically modifying, 250–251
- counting, 268–269
- data denormalization, 196–197
- data normalization, 195–196
- embedded documents, 196–197
- finding
  - find() method, 216
  - with Mongoose, 305–307
  - sets of documents, 265–268
- getting from collections, 246–248
- growth of, 198
- Mongoose Document object, 304–305
- paths, 294
- references, 195–196
- removing
  - findAndRemove() method, 257–258
  - with Mongoose, 314–317
  - remove() method, 217–218, 255–256
- saving
  - with Mongoose, 310–311
  - save() method, 252–253
- structure of, 192–193
- TTY (time-to-live), 199
- updating
  - findAndRemove() method, 218–219
  - with Mongoose, 311–314
  - update() method, 248–250
- upserting, 253–254

**dollar sign, 243**
**DOM (Document Object Model), 9**
**doRectangle() method, 507**
**doSquare() method, 507**
**doTrapezoid() method, 507**
**doTriangle() method, 507**
**double curly braces ({{}}), 429**
**do/while loops, 22**
**download responses, sending, 359**
**downloading MongoDB, 202**
**drag-and-drop application**
- app.component.css, 535–536
- app.component.html, 535
- app.component.ts, 534–535
- drag.component.css, 538
- drag.component.html, 538
- drag.component.ts, 538
- drop.component.css, 537

drop.component.html, 537
drop.component.ts, 536
folder structure, 533–534
**drag-item component**
app.component.css, 535–536
app.component.html, 535
app.component.ts, 534–535
drag.component.css, 538
drag.component.html, 538
drag.component.ts, 538
**drain event, 84, 142**
**driver. *See* MongoDB Node.js driver**
**drop() method, 215, 232**
**dropCollection() method, 228, 238**
**dropDatabase() method, 213, 228, 234**
**drop-item component**
app.component.css, 535–536
app.component.html, 535
app.component.ts, 534–535
drop.component.css, 537
drop.component.html, 537
drop.component.ts, 536
**dropping. *See* deleting data**
**dummyDB.JSON listing, 472**
**Duplex streams, 86–88**
**dynamic GET servers, 127–129**

## E

**each() method, 232**
**$each operator, 244**
**eCDH class, 188**
**EJS template**
creating, 361–363
implementing, 363–364
**elements of arrays, adding/removing, 34–35**
**$elemMatch, 266**
**elemMatch() method, 304**
**$elemMatch operator, 264**
**embedded documents, 196–197**
**emit() method, 64, 458**
**emitter_listener.js listing, 66–67**
**emitting custom events, 458**
**enable() method, 344**
**enabled() method, 344**
**enableSharding() method, 339–340**
**encoding methods, 75**
**encoding property**
exec() method, 167
execFile() method, 167
fork() function, 172
**encrypt_password.js listing, 189**
**end event, 82, 141**
**end() method, 85, 120, 122, 127, 142**
**endian, 75**
**endianness() method, 181**
**engine() method, 360**
**engines, template, 360**
defining, 360–361
locals, adding, 361
rendered templates, sending, 363–364
template creation, 361–363
**engines directive, 48**
**ensureIndex() method, 329**
**entryComponents metadata option (NgModule), 397**
**enum data type, 384**
**env method, 162**
**env property**
exec() and execFile() methods, 166
fork() function, 172
spawn() function, 170
**env setting (Express), 344**
**environment, adding Angular to, 393–394**

**$eq operator, 263**
**equal sign (=), 18**
**equality operators, 19**
**equals() method, 304**
**error event, 82, 142, 145, 165, 176**
**error() function, 53**
**error handling, 35**
finally keyword, 36–37
throwing errors, 36
try/catch blocks, 35–36
**errors module, 41**
**errors property (Document object), 305**
**escape codes, 29–30**
**event binding, 436–439**
**event listeners**
adding to objects, 65
implementing, 65–67
removing from objects, 65
**event queue, scheduling work on, 59–60**
**event.component.ts listing, 436–438**
**EventEmitter object, 64–65, 458**
**eventHandler() method, 459**
**events, 55–56**
binding, 436–439
blocking I/O, 57–58
browser events, 457–458
callbacks, 67
chaining, 70
implementing closure in, 68–69
passing parameters to, 67–68
ClientRequest object, 120
cluster module, 174
conversation metaphor, 57–59
custom events, 64–65, 458
deleting data with, 461–464
emitting from components, 458
handling with listeners, 458
implementing in nested components, 458–460
event callbacks, 55–56
event listeners
adding to objects, 65
implementing, 65–67
removing from objects, 65
event queue, scheduling work on, 59–60
IncomingMessage object, 123
process signals, 160
Readable streams, 82
Server object, 124, 145
ServerResponse object, 121
Socket object, 141–142
threaded models, 55–56
Worker object, 176
Writable streams, 84
**events module, 41. *See also* events**
**exec() method, 166–168, 318**
**execArgv method, 162**
**execFile() method, 168–169**
**execPath method, 162**
**execPath property (fork() function), 172**
**executable files, executing on another process, 168–169**
**executables**
mongod.exe, 202
Node.js executables, verifying, 42–43
**executing**
executable files on another process, 168–169
processes, 161
system commands on another process, 166–168
**exist event, 165**
**exists() method, 106–107, 303**
**$exists operator, 263**

**existsSync() method, 106–107**
**exit command, 204**
**exit event, 174, 176**
**exit() method, 161**
**explain option (options object), 264**
**export classes**
Directive, 449
ZoomDirective, 451
**export keyword, 392**
**exporting modules, 392**
**exports metadata option (NgModule), 397**
**Express, 13, 343**
configuring, 343–344
installing, 343
middleware, 367–368
assigning globally to path, 368
assigning to single route, 368
basicAuth, 368, 375–377
bodyParser, 368, 371–372
compress, 368
cookieParser, 368, 373–374
cookieSession, 368, 374–375
csrf, 368
custom, 380–381
favicon, 367
functions, 369
logger, 367
query, 368, 369
session, 368, 377–380
static, 367, 369–371
Request objects, 350–352
Response objects, 352
download responses, 359
files in, 356–358
headers, 352–353
JSON responses, 355–357
redirection, 359
rendered templates in, 363–364
sending, 353–355
status, 353
routes
applying parameters in, 347–350
definition of, 345
implementing, 346
server, starting, 345
template engines, 360
defining, 360–361
locals, adding, 361
rendered templates, sending, 363–364
template creation, 361–363
**express_auth_one.js listing, 376**
**express_auth_session.js listing, 378–379**
**express_auth.js listing, 376**
**express_cookies.js listing, 373–374**
**express_http_https.js listing, 345**
**express_json.js listing, 356**
**express_middleware.js listing, 381**
**express_post.js listing, 372**
**express_redirect.js listing, 359**
**express_request.js listing, 351**
**express_routes.js listing, 349–350**
**express_send_file.js listing, 358**
**express_send.js listing, 354**
**express_session.js listing, 374–375**
**express_static.js listing, 370**
**express_templates.js listing, 363–364**
**expressions, 415–416**
basic expressions, 416–417
Component class interaction, 418–419
pipes
built-in pipes, 422–426
custom pipes, 426–428
definition of, 422
TypeScript in, 419–422

**extensibility**
of Angular, 14
of Node.js, 3, 12
**external sources, interacting with, 132–134**
**external templates, 408–410**
**external.component.ts listing, 409**
**external.css listing, 409**
**externalTemplate.html listing, 409**

## F

**\f (form feed) escape code, 30**
**–f parameter (mongod command), 202**
**$facet operator, 285**
**@fadeState, 527–529**
**favicon middleware, 367**
**feed() function, 386**
**fields**
distinct field values, finding, 276–277
limiting results by, 271–272
naming conventions, 193
required fields, forcing, 296
unique fields, 296
**fields option (options object), 264**
**file system, 41, 95**
asynchronous file system calls, 95
directories
creating, 111–112
deleting, 111–112
renaming, 112
files
deleting, 110
listing, 108–110
opening/closing, 96–97
reading, 102–106
renaming, 112
returning statistics about, 107–108
truncating, 110–111
verifying path of, 106–107
watching for file changes, 112–113
writing, 97–102
synchronous file system calls, 95
**file_read_async.js listing, 105**
**file_read_stream.js listing, 106**
**file_read_sync.js listing, 103–104**
**file_readdir.js listing, 109–110**
**file_read.js listing, 102–103**
**file_stats.js listing, 108**
**file_write_async.js listing, 100–101**
**file_write_stream.js listing, 101**
**file_write_sync.js listing, 99**
**file_write.js listing, 98**
**files. *See also* listings**
deleting, 110
executing in another process, 168–169
listing, 108–110
node, 42
opening/closing, 96–97
package.json file, 47–48
reading
asynchronous file reading, 104–105
simple file read, 102–103
streamed file reading, 105–106
synchronous file reading, 103–104
renaming, 112
returning statistics about, 107–108
sending in responses, 356–358
static files, serving, 125–127, 369–371
truncating, 110–111
verifying path of, 106–107
watching for file changes, 112–113
writing
asynchronous file writing, 99–101
simple file write, 98

streaming file writing, 101–102
synchronous file writing, 98–99
**fill() method, 76**
**finally keyword, 36–37**
**find() method, 216, 231, 246–248, 265–268, 299, 300, 306**
**findAndModify() method, 231, 250–251**
**findAndRemove() method, 231, 257, 300**
**findOne() method, 231, 246–248, 300, 305**
**findOneAndUpdate() method, 301**
**finish event, 84**
**$first operator, 286**
**first.component.js listing, 402**
**first.component.ts listing, 400**
**first.html listing, 399**
**flush() method, 88**
**(focus) event, 457**
**folders, node_modules, 42**
**forceServerObjectId option, 242**
**forcing required fields, 296**
**for/in loops, 23**
**fork event, 174**
**fork() method, 171–173**
**forks, 171–173**
**for loops, 22–23**
**form feed escape code, 30**
**form parameters, processing, 117–118**
**format() method, 183–184**
**formatGreeting() function, 25**
**formatting strings, 183–184**
**forms service, 470**
**frameworks.** *See* **Angular; Node.js**
**freemem() method, 182**
**fresh property (Request object), 351**
**fromCharCode() method, 30**
**fs module.** *See* **file system**
**fsync option, 242**
**functions, 24.** ***See also individual functions (for example,*** **doCalc() function*)***
anonymous functions, 25–26
callback functions, 67
applying for defined parameters, 348–349
chaining, 70
implementing closure in, 68–69
passing parameters to, 67–68
defining, 24
passing variables to, 24–25
returning values from, 25
TypeScript functions, 388–389

## G

**generating content with CLI (command-line interface), 394–395**
**$geoNear operator, 285**
**geospatial indexes, 328**
**get() method, 304, 344, 352, 470**
**GET requests**
definition of, 9
dynamic GET servers, 127–129
response callback functions, 471
sending, 470–471
**GET servers, 127–129**
**getCensoredWords() function, 52**
**getCharacters() method, 517**
**getConnections() method, 146**
**GetData requests, 55–56**
**GetFile requests, 55–56**
**getgid() method, 162**
**getgroups() method, 163**
**getHeader() method, 122**
**getItems() method, 541**
**getObservable() method, 541, 543**

**getPi() method, 505**
**getRandom() function, 510**
**getRandomImage() method, 512**
**getSiblingDB() method, 212**
**getStarClass() method, 544–545**
**getuid() method, 162**
**getUsers() method, 476, 482**
**getWeather() function, 132**
**gid property (spawn() function), 170**
**global module, 41, 190**
**Good Guys component**
good-guys.component.css, 521
good-guys.component.html, 520
good-guys.component.ts, 520
**gotoPage2() function, 498–499**
**$graphLookup operator, 285**
**greater than (>) operator, 19**
**greater than or equal to (>=) operator, 19**
**greeting() function, 25**
**group() method, 277–282, 318**
**$group operator, 284**
**grouping results, 277–282**
**growth of MongoDB documents, 198**
**gt() method, 303**
**$gt operator, 263**
**gte() method, 303**
**$gte operator, 263**
**gunzip() method, 91**
**gzip() method, 91**

## H

**handshakeTimeout option (tls.createServer), 155**
**hash class, 189**
**hash property (URL object), 117**
**hash-based sharding, 337**
**hashed indexes, 329**
**head() method, 470**
**headers, 10, 352–353**
**headers property (IncomingMessage object), 123**
**headersSent event, 121**
**help command, 204**
**–help parameter (mongod command), 202**
**Hex encoding, 75**
**high availability, 3, 13**
**hint() method, 302**
**hint option (options object), 264**
**hitCharacter() method, 517–520**
**hmac class, 189**
**home.component.html listing, 490**
**home.component.ts listing, 490**
**honorCipherOrder option (tls.createServer), 155**
**host property**
ClientRequest object, 119
URL object, 116
**hostname() method, 181**
**hostname property**
ClientRequest object, 119
Request object, 351
URL object, 116
**href property (URL object), 116**
**hrtime() method, 162**
**HTML (Hypertext Markup Language)**
events, Angular syntax for, 457–458
files, 9
inline HTML in Angular applications, 405–406
listings
animated.Component.html, 529
app.component.html. *See* app.component.html listing

app.module.ts, 480
attribute.component.html, 447
badguys.component.html, 522
character.component.html, 462
container.component.html, 455
createDelete.component.html, 479
customevent.component.html, 459–460
details.component.html, 463
drag.component.html, 538
drop.component.html, 537
externalTemplate.html, 409
first.html, 399
good-guys.component.html, 520
home.component.html, 490
http.component.html, 474
observable.component.html, 467
outer.html, 411
page1.component.html, 499
page2.component.html, 496, 500
page3.component.html, 496
rated-item.component.html, 545
route2.component.html, 491
update.component.html, 484–485
user_ejs.html, 362
zoomit.component.html, 532
router-outlet tag, 488

**HTTP (Hypertext Transfer Protocol), 8, 115. *See also* http service**
authentication, 375–377
clusters, 176–179
headers, 10
HTTPS, 8, 134–135
certificate signing request files, 135
HTTPS clients, 135–136
HTTPS servers, 137
private keys, 135
query string and form parameter processing, 117–118
requests, 9
ClientRequest object, 118–121
methods, 350–352
POST, 371–372
properties, 350–352
responses, 352
download responses, 359
files in, 356–358
headers, 352–353
IncomingMessage object, 122–123
JSON responses, 355–357
redirecting, 359
rendered templates in, 363–364
sending, 353–355
ServerResponse object, 121–122
status, 353
servers
dynamic GET servers, 127–129
external sources, interacting with, 132–134
POST servers, 130–131
Server object, 123–125
static files, serving, 125–127
URLs (Uniform Resource Locators)
resolving, 117
structure of, 115–116
URL object, 116–117

**http module. *See* HTTP (Hypertext Transfer Protocol)**

**http service, 470**
GET requests, sending, 470–471
JSON file implementation and access, 472–475
app.module.ts, 473–474
dummyDB.JSON, 472
http.component.CSS, 474–475

http.component.html, 474
http.component.ts, 473
PUT requests, sending, 470–471
request configuration, 471
response callback functions, 471
simple mock server implementation, 475–481
app.module.ts, 480
createDelete.component.CSS, 479–480
createDelete.component.html, 479
createDelete.component.ts, 477–478
creating, 475
data.service.ts, 476
user.service.ts, 478–479
simple mock server updates, 481–486
data.service.ts, 481–482
update.component.CSS, 485–486
update.component.html, 484–485
update.component.ts, 482–483
user.service.ts, 483–484
**http_client_get.js listing, 128**
**http_client_post.js listing, 131**
**http_client_static.js listing, 126–127**
**http_server_external listing, 132–133**
**http_server_get.js listing, 128**
**http_server_post.js listing, 130**
**http_server_static.js listing, 126**
**http.component.CSS listing, 474–475**
**http.component.html listing, 474**
**http.component.ts listing, 473**
**HttpModule, importing, 473–474**
**https module, 41, 134–135**
certificate signing request files, 135
HTTPS clients, 135–136
HTTPS servers, 137
private keys, 135
**httpVersion property (IncomingMessage object), 123**
**Hypertext Markup Language. *See* HTML (Hypertext Markup Language)**
**Hypertext Transfer Protocol. *See* HTTP (Hypertext Transfer Protocol)**


## I


**_id indexes, 327–328**
**id metadata option (NgModule), 397**
**id property**
Document object, 304
Schema object, 295
Worker object, 176
**IDE (Integrated Development Environment), 43**
**if statements, 20**
**imageClick() function, 532**
**images**
animation application
animated.component.css, 529
animated.Component.html, 529
animated.component.ts, 527–528
app.component.html, 526–527
app.component.ts, 526
app.module.ts, 525–526
drag-and-drop application
app.component.css, 535–536
app.component.html, 535
app.component.ts, 534–535
drag.component.css, 538
drag.component.html, 538
drag.component.ts, 538
drop.component.css, 537
drop.component.html, 537
drop.component.ts, 536
folder structure, 533–534

image zoom application
app.component.html, 531
app.component.ts, 530–531
folder structure, 530
zoomit.component.css, 532
zoomit.component.html, 532
zoomit.component.ts, 531–532
**immediate timers, 62**
**import keyword, 392**
**importing**
modules, 392
ActivatedRoute, 488
AppRoutingModule, 488
Component, 396–397
HttpModule, 473–474
InMemoryDbService, 476, 481
InMemoryWebApiModule, 480–481
NgModule, 397–398
Router, 488, 492
Observable object, 510–512
**imports metadata option (NgModule), 397**
**$inc operator, 243**
**IncomingMessage object, 122–123**
**increment (++) operator, 18**
**index() method, 295–296**
**indexed arrays, 384**
**indexes, 198–199**
adding to schemas, 295–296
creating, 327–330
**indexes() method, 296**
**indexOf() method, 30, 31, 33, 34**
**indexOptionsDefaults option (collections), 215**
**$indexStats operator, 285**
**inflate() method, 91**
**inflateRaw() method, 91**
**info() function, 53**
**inheritance, 387**
**inherits() method, 83, 185–186**
**initgroups() method, 163**
**inline CSS and HTML, 405–406**
**InMemoryDbService, importing, 476, 481**
**InMemoryWebApiModule**
implementing, 540
importing, 480–481
**in() method, 303**
**inner.component.ts listing, 412**
**$in operator, 263**
**Input decorator, 412**
**input.component.ts listing, 413–414**
**insert() method, 217, 230, 244–246**
**inspect() method, 184–185**
**installing**
Express, 343
MongoDB, 201–202
Node.js, 42
NPMs (Node Packaged Modules), 46
**Integrated Development Environment (IDE), 43**
**interface keyword, 385**
**interfaces**
Person, 385
PersonInterface, 386
RatedItem, 541
Stringy, 385
TypeScript, 385–386
**interpolation, 430–431**
**interpolation.component.ts listing, 430–431**
**interrupting loops, 23–24**
**interval timers, 61–62**
**intro.ts listing, 406**
**invalidate() method, 305**
**I/O**
blocking, 57–58
pipes, 159–160

**ip property (Request object), 351**
**isClosed() method, 233**
**isInit() method, 305**
**isinstanceof operator, 184**
**isMaster property (cluster module), 175**
**isModified() method, 305**
**isNew property (Document object), 305**
**isSelected() method, 305**
**isWorker property (cluster module), 175**
**iterating through arrays, 34**

## J

**JavaScript, 15. *See also* listings**
arrays
adding items to, 34–35
combining, 33–34
converting into strings, 34
iterating through, 34
manipulating, 32–33
methods, 32–33
removing items from, 34–35
searching, 34
conditionals
if statements, 20
switch statements, 20–21
data types, 16–17
error handling, 35
finally keyword, 36–37
throwing errors, 36
try/catch blocks, 35–36
functions, 24. *See also individual functions (for example, doCalc() function)*
anonymous functions, 25–26
callback functions, 67–70, 348–349
defining, 24
passing variables to, 24–25
returning values from, 25
JSON (JavaScript Object Notation), 73
converting JavaScript objects to, 74
converting to JavaScript objects, 74
keywords
export, 392
finally, 36–37
import, 392
new, 27
styles, 405
styleUrls, 408
templateUrl, 408
var, 15–16
loops, 21
do/while, 22
for, 22–23
for/in, 23
interrupting, 23–24
while, 21
methods, 27. *See also individual methods (for example, write() method)*
objects, 27
Array, 32–33
converting JSON to, 74
converting to JSON, 74
custom events, 64–65
custom-defined objects, 28
prototyping object patterns, 29
String, 29–31
syntax, 27–28
operators, 17
arithmetic operators, 17–18
assignment operators, 18
comparison operators, 19–20
isinstanceof, 184
statements
break, 23–24
catch, 35–36

continue, 23–24
if, 20
return, 25
switch, 20–21
throw, 36
try, 35–36
strings
combining, 31
converting arrays into, 34
manipulating, 29–31
replacing words in, 31
searching, 31
splitting, 32
string-related methods, 30–31
variables
defining, 15–16
passing to functions, 24–25
scope, 26–27
**JavaScript Object Notation. *See* JSON (JavaScript Object Notation)**
**join() method, 33, 34**
**journal option, 242**
**JSON (JavaScript Object Notation), 73**
converting JavaScript objects to, 74
converting to JavaScript objects, 74
file implementation and access, 472–475
app.module.ts, 473–474
dummyDB.JSON, 472
http.component.CSS, 474–475
http.component.html, 474
http.component.ts, 473
responses, sending, 355–357
**json pipe, 423**
**jsonp callback name setting (Express), 344**
**jsonp replacer setting (Express), 344**
**jsonp spaces setting (Express), 344**


## K


**keepAlive option (server object), 224**
**key option**
htp.createServer(), 135–136
https.request(), 135–136
tls.connect(), 154
tls.createServer(), 155
**(keydown) event, 458**
**(keypress) event, 458**
**keys**
private keys, generating, 135
shard keys, 336–337
x509 public key, 135, 137
**(keyup) event, 458**
**keywords. *See also* statements**
export, 392
import, 392
new, 27
styles, 405
styleUrls, 408
templateUrl, 408
this, 323
var, 15–16
**keywords directive, 48**
**kill() method, 161, 165, 176**
**killSignal property (exec() method), 167**


## L


**$last operator, 286**
**lastIndexOf() method, 30, 33**
**leave() method, 527**
**length of buffers, determining, 78**
**less than (<) operator, 19**
**less than or equal to (<=) operator, 19**
**limit() method, 301, 318**
**$limit operator, 284**

**limit option (options object), 264**
**limiting result sets, 270**
by fields, 271–272
paging results, 273–274
by size, 270–271
**listDatabases() method, 229, 233**
**listdb_connect_url.js, 225**
**listen() method, 124, 146, 345**
**listeners, 458**
adding to objects, 65
implementing, 65–67
removing from objects, 65
**listeners() function, 65**
**listening event, 145, 174**
**listing**
collections, 214, 237
databases, 211, 233
files, 108–110
MongoDB server status, 236–237
MongoDB user accounts, 206–207
**listings**
animated.component.css, 529
animated.Component.html, 529
animated.component.ts, 527–528
app.component.css
AreaCalcService, 509
custom directive with component, 454
drag-and-drop application, 535–536
RandomImageService, 513–514
router with navigation bar, 495
star rating application, 544
app.component.html
animation application, 526–527
AreaCalcService, 508–509
custom directive with component, 453–454
drag-and-drop application, 535
image zoom application, 531
PiService, 506
PromiseService, 516
RandomImageService, 513
router with navigation bar, 494
SharedService, 519
star rating application, 543–544
zoom directive, 451–452
app.component.ts
animation application, 526
AreaCalcService, 507–508
custom directive with component, 453
drag-and-drop application, 534–535
image zoom application, 530–531
PiService, 505
PromiseService, 515–516
RandomImageService, 512–513
router with navigation bar, 494
router with parameters, 498
SharedService, 519
simple router application, 490
star rating application, 543
app.module.js, 401
app.module.ts
Angular bootstrapper, 400
animation application, 525–526
simple mock server implementation, 473–474, 480
simple router implementation, 488–489
star rating application, 540
app-routing.module.ts
router with navigation bar, 492–493
router with parameters, 498
simple router implementation, 489–490

area-calc.service.ts, 506–507
attribute.component.css, 448
attribute.component.html, 447
attribute.component.ts, 446–447
badguys.component.css, 522
badguys.component.html, 522
badguys.component.ts, 521
basicExpressions.component.ts, 417
buffer_concat.js, 81
buffer_copy.js, 79
buffer_read.js, 78
buffer_slice.js, 80
buffer_write.js, 77
builtInPipes.component.ts, 425
callback_chain.js, 70
callback_closure.js, 69
callback_parameter.js, 67–68
censortext.js, 49
character.component.css, 462
character.component.html, 462
character.component.ts, 461–462
child_exec.js, 167–168
child_fork.js, 172–173
child_process_exec_file.js, 168–169
child_process_spawn_file.js, 170–171
class.component.ts, 434
classExpressions.component.ts, 418–419
cluster_server.js, 177–179
collection_create_list_delete.js, 238–239
collection_stat.js, 239–240
constructor.component.ts, 407
container.component.css, 455
container.component.html, 455
container.component.ts, 454
createDelete.component.CSS, 479–480
createDelete.component.html, 479
createDelete.component.ts, 477–478
customevent.component.html, 459–460
customevent.component.ts, 459
customPipes.component.ts, 427
custom.pipe.ts, 427
data.service.ts, 476, 481–482
db_connect_object.js, 226
db_create_list_delete.js, 234–236
db_status.js, 236–237
details.component.css, 463
details.component.html, 463
details.component.ts, 462–463
dns_lookup.js, 187–188
doc_aggregate.js, 288
doc_count.js, 268–269
doc_delete_one.js, 257–258
doc_delete.js, 255–256
doc_distinct.js, 277
doc_fields.js, 271–272
doc_find.js, 247–248
doc_group.js, 279–280
doc_insert.js, 245–246
doc_limit.js, 270–271
doc_modify.js, 251
doc_paging.js, 273–274
doc_query.js, 266–268
doc_save.js, 252–253
doc_sort.js, 275–276
doc_update.js, 249–250
doc_upsert.js, 253–254
drag.component.css, 538
drag.component.html, 538
drag.component.ts, 538
drop.component.css, 537
drop.component.html, 537
drop.component.ts, 536
dummyDB.JSON, 472
emitter_listener.js, 66–67

encrypt_password.js, 189
event.component.ts, 436–438
express_auth_one.js, 376
express_auth_session.js, 378–379
express_auth.js, 376
express_cookies.js, 373–374
express_http_https.js, 345
express_json.js, 356
express_middleware.js, 381
express_post.js, 372
express_redirect.js, 359
express_request.js, 351
express_routes.js, 349–350
express_send_file.js, 358
express_send.js, 354
express_session.js, 374–375
express_static.js, 370
express_templates.js, 363–364
external.component.ts, 409
external.css, 409
externalTemplate.html, 409
file_read_async.js, 105
file_read_stream.js, 106
file_read_sync.js, 103–104
file_readdir.js, 109–110
file_read.js, 102–103
file_stats.js, 108
file_write_async.js, 100–101
file_write_stream.js, 101
file_write_sync.js, 99
file_write.js, 98
first.component.js, 402
first.component.ts, 400
first.html, 399
good-guys.component.css, 521
good-guys.component.html, 520
good-guys.component.ts, 520
home.component.html, 490
home.component.ts, 490
http_client_get.js, 128
http_client_post.js, 131
http_client_static.js, 126–127
http_server_external, 132–133
http_server_get.js, 128
http_server_post.js, 130
http_server_static.js, 126
http.component.CSS, 474–475
http.component.html, 474
http.component.ts, 473
inner.component.ts, 412
input.component.ts, 413–414
interpolation.component.ts, 430–431
intro.ts, 406
main_pug.pug, 362–363
mockbackend.service.ts, 540–541
mongoose_aggregate.js, 319
mongoose_connect.js, 293
mongoose_create.js, 308–309
mongoose_find.js, 306–307
mongoose_middleware.js, 323–324
mongoose_remove_many.js, 316
mongoose_remove_one.js, 315
mongoose_save.js, 310
mongoose_update_many.js, 313–314
mongoose_update_one.js, 312
mongoose_validation.js, 321
nav.component.CSS, 494
nav.component.html, 493
nav.component.ts, 493
nexttick.js, 63–64
observable.component.html, 467
observable.component.ts, 466–467
os_info.js, 182–183
outer.component.ts, 411

outer.css, 411
outer.html, 411
package.json, 50, 51
page1.component.html, 499
page1.component.ts
  router with navigation bar, 495
  router with parameters, 499
page2.component.html
  router with navigation bar, 496
  router with parameters, 500
page2.component.ts
  router with navigation bar, 495
  router with parameters, 500
page3.component.html, 496
page3.component.ts, 496
person.component.ts, 413
pi.service.ts, 505
process_info.js, 163–164
promise.service.ts, 515
property.component.ts, 432
random-image.service.ts, 511–512
rated-item.component.css, 545–546
rated-item.component.html, 545
rated-item.component.ts, 544–545
ratings.service.ts, 541–542
readwords.js, 52
route2.component.html, 491
route2.component.ts, 491
shared.service.ts, 517–518
simple_interval.js, 61–62
simple_timer.js, 60–61
socket_client.js, 148–149
socket_server.js, 151–152
./static/css/static.css, 370
stream_duplex.js, 87
stream_piped.js, 90
stream_read.js, 83–84
stream_transform.js, 88–89
stream_write.js, 85
structural.component.ts, 443–444
style.component.ts, 435
twoWay.component.ts, 439–440
typescriptExpressions.component.ts, 421
update.component.CSS, 485–486
update.component.html, 484–485
update.component.ts, 482–483
user_ejs.html, 362
user.service.ts, 478–479, 483–484
util_inherit.js, 185–186
word_schema.js, 297
zlib_buffers.js, 91–92
zlib_file.js, 93
zoom.component.ts, 450–452
zoom.directive.ts, 450–451
zoomit.component.css, 532
zoomit.component.html, 532
zoomit.component.ts, 531–532
**little endian, 75**
**load() method, 205, 206**
**loadavg() method, 182**
**loadChildren property (route object), 487**
**local template variables, adding, 361**
**localAddress property**
  ClientRequest object, 119
  Socket object, 143
**localPort property (Socket object), 144**
**location() method, 353**
**log() function, 53**
**logCar() function, 67–70**
**logColorCar() function, 67**
**logger middleware, 367**
**logout() method, 228, 229**
**–logpath parameter (mongod command), 203**
**lookup() method, 186**

**$lookup operator, 285**
**loops, 21**
do/while, 22
for, 22–23
for/in, 23
interrupting, 23–24
while, 21
**lowercase pipe, 423**
**lt() method, 303**
**$lt operator, 263**
**$lte operator, 263**

## M

**main directive, 47**
**main_pug.pug listing, 362–363**
**manipulating**
arrays, 32–33
strings, 29–31
**map() method, 471**
**MapReduce, 282.** ***See also*** **aggregating results**
**markModified() method, 305**
**match() method, 30, 318**
**$match operator, 283, 284**
**$max operator, 286**
**max option (collections), 214**
**maxBuffer property (exec() method), 167**
**–maxConns parameter (mongod command), 203**
**maxScan option (options object), 265**
**maxTickDepth() method, 162**
**media files, 9**
**memoryUsage() method, 162**
**message event, 165, 176**
**method property**
ClientRequest object, 119
IncomingMessage object, 123
Request object, 351
**methods.** ***See also individual methods (for example,*** **write() method*)***
adding to schemas, 295–296
definition of, 27
**methods property (Schema object), 296**
**middleware (Express), 367–368**
assigning globally to path, 368
assigning to single route, 368
basicAuth, 368, 375–377
bodyParser, 368, 371–372
compress, 368
cookieParser, 368, 373–374
cookieSession, 368, 374–375
csrf, 368
custom, 380–381
favicon, 367
functions, 369
logger, 367
Mongoose middleware functions, 322–324
query, 368, 369
session, 368, 377–380
static, 367, 369–371
**middleware functions (Mongoose), 322–324**
**$min operator, 286**
**mkdir() method, 111**
**mkdirSync() method, 111**
**mock back-end service, 540–541**
**mock web servers**
simple JSON file implementation, 472–475
app.module.ts, 473–474
dummyDB.JSON, 472
http.component.CSS, 474–475
http.component.html, 474
http.component.ts, 473
simple mock server implementation, 475–481

app.module.ts, 480
createDelete.component.CSS, 479–480
createDelete.component.html, 479
createDelete.component.ts, 477–478
creating, 475
data.service.ts, 476
user.service.ts, 478–479
simple mock server updates, 481–486
data.service.ts, 481–482
update.component.CSS, 485–486
update.component.html, 484–485
update.component.ts, 482–483
user.service.ts, 483–484
**mockbackend.service.ts listing, 540–541**
**mod() method, 303**
**$mod operator, 263, 287**
**model() method, 298**
**Model object, 292, 298**
**models, compiling, 298**
**modifiedFields() method, 310**
**modifiedPaths() method, 305**
**modules (Angular), 392**
importing
AppRoutingModule, 488
BrowserAnimationsModule, 526
Component, 396–397
HttpModule, 473–474
InMemoryDbService, 476, 481
InMemoryWebApiModule, 480–481
NgModule, 397–398
Router, 492
routing module, 488
TypeScript, 387–388
**modules (Node.js)**
assertion testing, 190
buffer. *See* buffers
C/C++ add-ons, 190
child_process, 159–160
child forks, 171–173
ChildProcess object, 164–166
executable files, executing on another process, 168–169
processes, spawning, 169–171
system command, executing on another process, 166–168
cluster, 174
events, 174
HTTP clusters, 176–179
methods, 175
properties, 175
Worker object, 175–176
console, 53–54
creating, 49–50
crypto, 188–190
definition of, 43
dns, 186–188
events. *See* events
express. *See* Express
fs. *See* file system
global, 190
http. *See* HTTP (Hypertext Transfer Protocol)
https, 134–135
certificate signing request files, 135
HTTPS clients, 135–136
HTTPS servers, 137
private keys, 135
installing, 46
loading into Node.js applications, 52–53
mongodb. *See* MongoDB Node.js driver

mongoose. *See* Mongoose
net, 41. *See also* socket services
Server object, 144–147
Socket objects, 140–144
Node Package Manager, 44–45
Node Package Registry, 43–44
publishing modules to, 50–51
viewing, 43–44
os, 181–183
overview of, 40–41
package.json file, 47–48
process, 159
process execution, 161
process I/O pipes, 159–160
process signals, 160
returning information from, 161–164
publishing to NPM Registry, 50–51
REPL (Read Event Print Loop), 190
searching for, 45
stream. *See* streams
tls, 139–140, 152–153
TLS socket clients, 153–154
TLS socket servers, 154–156
util, 183–186
format() method, 183–184
inherits() method, 185–186
inspect() method, 184–185
isinstanceof operator, 184
V8, 190
Zlib, 91
compressing/decompressing buffers, 91–92
compressing/decompressing streams, 92–93

**modules module, 41**

**modulus (%) operator, 18**

**MongoClient object, 222, 223–226**

**mongod command, 202–203**

**MongoDB, 192. *See also* MongoDB Node.js driver; Node.js-to-Angular stack**

access control, 209
authentication, 210–211
Database Administrator accounts, 211
User Administrator accounts, 209–210
access from shell client, 203–204
command parameters, 205
shell commands, 204
shell methods, 205
shell scripts, 205–206
advantages of, 3, 12–13
atomic write operations, 198
backing up, 341–342
collections
accessing statistics for, 239–240
adding documents to, 217, 244–246, 307–309
atomically modifying documents in, 250–251
capped collections, 197–198, 330
collection_create_list_delete.js application example, 238–239
creating, 214–215, 237
definition of, 192
deleting, 215–216, 238
deleting documents in, 217–218, 255–256
displaying list of, 214
enabling sharding on, 340
finding documents in, 216, 305–307
getting documents from, 246–248
listing, 237
number of, 199

removing documents from, 314–317
removing single document from, 257–258
saving documents in, 252–253, 310–311
updating documents in, 218–219, 248–250, 309–310
upserting documents in, 253–254
connecting to
MongoClient object, 222, 223–226
Mongoose, 292–294
write concern, 222
data life cycles, 199
data model, planning, 194–195
data types, 193–194
data usability and performance, 200
databases
changing current, 211–212
connecting to using Mongoose, 292–294
copying, 213
creating, 212, 234
database change options, 241–242
db_create_list_delete.js sample application, 234–236
deleting, 212–213, 234
displaying list of, 211
listing, 233
repairing, 341
update operators, 242–244
definition of, 3, 12
distinct field values, finding, 276–277
documents
adding, 217, 244–246, 307–309
aggregating with Mongoose, 317–320
atomically modifying, 250–251
counting, 268–269
data denormalization, 196–197
data normalization, 195–196
embedded documents, 196–197
finding, 216, 265–268, 305–307
getting from collections, 246–248
growth of, 198
paths, 294
references, 195–196
removing, 217–218, 255–256, 257–258, 314–317
saving, 252–253, 310–311
structure of, 192–193
TTY (time-to-live), 199
updating, 218–219, 248–250, 309–314
upserting, 253–254
downloading, 202
indexing, 198–199, 327–330
installing, 201–202
MapReduce, 282
middleware functions, 322–324
Mongoose, 291–292
adding documents with, 307–309
aggregating documents with, 317–320
Document object, 304–305
finding documents with, 305–307
middleware functions, 322–324
models, compiling, 298
objects, 292
Query object, 298–305
removing multiple documents with, 315–317
removing single documents with, 314–315
schemas, 294–298
updating multiple documents with, 313–314

updating single documents with, 311–313
validation framework, 320–322
objects
Admin, 229
Collection, 229–232
Cursor, 232–233
Db, 227–228
MongoClient, 222, 223–226
options, 264–265
query, 262–264
replication, 199
applying, 330–332
replica sets, 333–334
strategy, 332–333
result sets, 270
aggregating, 282–289
grouping results, 277–282
limiting by size, 270–271
limiting fields returned in, 271–272
paging results, 273–274
sorting, 275–276
server status, displaying, 236–237
sharding, 199
definition of, 334
enabling on collections, 340
enabling on databases, 339–340
hash-based sharding, 337
partitioning methods, 337
range-based sharding, 337
shard keys, 336–337
shard tag ranges, 340–341
sharded MongoDB clusters, 338–339
sharding server types, 335
starting, 202–203
stopping, 203
user accounts
creating, 206–207
listing, 206–207
removing, 209
roles, 208

**mongodb module. *See* MongoDB Node.js driver**

**MongoDB Node.js driver. *See also* Mongoose**
adding to project, 221
collections
accessing statistics for, 239–240
collection_create_list_delete.js application example, 238–239
creating, 237
deleting, 238
listing, 237
database change options, 241–242
database update operators, 242–244
databases
creating, 234
db_create_list_delete.js sample application, 234–236
deleting, 234
listing, 233
distinct field values, finding, 276–277
documents
adding to collections, 244–246
atomically modifying, 250–251
counting, 268–269
deleting, 255–258
finding specific sets of, 265–268
getting from collections, 246–248
saving in collections, 252–253
updating in collections, 248–250
upserting, 253–254

objects
Admin, 229
Collection, 229–232
Cursor, 232–233
Db, 227–228
query objects, 262–265
result sets, 270
aggregating, 282–289
grouping results, 277–282
limiting by size, 270–271
limiting fields returned in, 271–272
paging results, 273–274
sorting, 275–276
**mongod.exe, 202**
**mongodump command, 342**
**Mongoose, 291–292**
connecting to MongoDB with, 292–294
Document object, 304–305
documents
adding, 307–309
aggregating, 317–320
finding, 305–307
removing multiple, 315–317
removing single, 314–315
saving, 310–311
updating multiple, 313–314
updating single, 311–313
models, compiling, 298
objects, 292
Query object, 298–299
operators, 302–304
query database operation, 299–302
schemas
defining, 294–295
implementing on database, 295–296
indexes, 295–296
methods, 295–296
paths, 294
required fields, forcing, 296
unique fields, 296
value types, 294–298
validation framework, 320–322
**mongoose_aggregate.js listing, 319**
**mongoose_connect.js listing, 293**
**mongoose_create.js listing, 308–309**
**mongoose_find.js listing, 306–307**
**mongoose_middleware.js listing, 323–324**
**mongoose_remove_many.js listing, 316**
**mongoose_remove_one.js listing, 315**
**mongoose_save.js listing, 310**
**mongoose_update_many.js listing, 313–314**
**mongoose_update_one.js listing, 312**
**mongoose_validation.js listing, 321**
**(mouseover) event, 458**
**multi option, 242**
**multikey indexes, 328**
**multiple documents**
removing with Mongoose, 315–317
updating with Mongoose, 313–314
**multiplication (*) operator, 18**
**$multiply operator, 287**
**myCustomEvent, 459**

## N

**\n (new line) escape code, 29**
**name directive, 47**
**naming conventions, 193, 404**
**nav.component.CSS listing, 494**
**nav.component.html listing, 493**
**nav.component.ts listing, 493**
**navigating routes, 488**

**navigation bar, router with, 492–497**
app.component.CSS, 495
app.component.html, 494
app.component.ts, 494
app-routing.module.ts, 492–493
nav.component.CSS, 494
nav.component.html, 493
nav.component.ts, 493
page1.component.ts, 495
page2.component.html, 496
page2.component.ts, 495
page3.component.html, 496
page3.component.ts, 496
**ne() method, 303**
**$ne operator, 263**
**nested components**
building with dependency injection, 410–412
custom events in, 458–460
**net module, 41. *See also* socket services**
Server object, 144–147
Socket objects
creating, 140–141
data flow across, 144
events, 141–142
methods, 142–143
properties, 143–144
**networkInterfaces() method, 182**
**new keyword, 27**
**new line escape code, 29**
**new option, 242**
**newSession event, 156**
**nextObject() method, 232**
**nextTick() function, 63–64, 161**
**nexttick.js listing, 63–64**
**ng eject command, 395**
**ng generate component command, 395**
**ng generate directive command, 395**
**ng generate enum command, 395**
**ng generate guard command, 395**
**ng generate interface command, 395**
**ng generate module command, 395**
**ng generate pipe command, 395**
**ng generate service command, 395**
**ng new command, 395**
**ng serve command, 395**
**ng-content directive, 452**
**ngFor directive, 442, 474, 479, 484, 513**
**ngForm directive, 445**
**ngIf directive, 442–443**
**ngModel directive, 445**
**NgModule, 397–398, 410**
**ngOnInit() method, 465, 482, 505, 531, 543**
**ngStyle directive, 445**
**ngSwitch directive, 442–443**
**ngSwitchCase directive, 442, 443**
**ngSwitchDefault directive, 442**
**nin() method, 303**
**$nin operator, 263**
**node file, 42**
**Node Package Manager. *See* NPM (Node Package Manager)**
**Node Package Registry**
publishing modules to, 50–51
viewing, 43–44
**Node Packaged Modules. *See* NPMs (Node Packaged Modules)**
**node_modules directory, 221**
**node_modules folder, 42**
**Node.js. *See also* modules (Node.js); MongoDB Node.js driver**
advantages of, 2–3, 11–12
buffers, 74–75

compressing/decompressing, 91–92
concatenating, 81
copying, 79–80
creating, 75–76
determining length of, 78
encoding methods, 75
reading from, 77–78
slicing, 80
writing to, 76–77
child processes, 164
child forks, 171–173
ChildProcess object, 164–166
executable files, executing on another process, 168–169
processes, spawning, 169–171
system command, executing on another process, 166–168
companies using, 40
connecting to MongoDB from
MongoClient object, 223–226
write concern, 222
definition of, 2, 11, 39
development of, 39
directories
creating, 111–112
deleting, 111–112
renaming, 112
events, 55–56
blocking I/O, 57–58
callbacks, 67–70
conversation metaphor, 57–59
custom events, 64–65
event callbacks, 55–56
event listeners, 65–67
event queue, scheduling work on, 59–60
threaded models, 55–56
executables, verifying, 42–43
files
asynchronous file system calls, 95
deleting, 110
listing, 108–110
opening/closing, 96–97
reading, 102–106
renaming, 112
returning statistics about, 107–108
synchronous file system calls, 95
truncating, 110–111
verifying path of, 106–107
watching for file changes, 112–113
writing, 97–102
HTTP services, 115
ClientRequest object, 118–121
dynamic GET servers, 127–129
external sources, interacting with, 132–134
IncomingMessage object, 122–123
POST servers, 130–131
query string and form parameter processing, 117–118
Server object, 123–125
ServerResponse object, 121–122
static files, serving, 125–127
URLs (Uniform Resource Locators), 115–117
HTTPS services, 134–135
certificate signing request files, 135
HTTPS clients, 135–136
HTTPS servers, 137
private keys, 135
IDE (Integrated Development Environment), 43
install location, 42
installing, 42

JSON (JavaScript Object Notation), 73
converting JavaScript objects to, 74
converting to JavaScript objects, 74
manipulating documents from. *See* documents (MongoDB)
MongoClient object, 222
nextTick() function, 63–64
objects
checking type of, 184
converting to strings, 184–185
EventEmitter, 64–65
Stats, 107
process clusters, 174
events, 174
HTTP clusters, 176–179
methods, 175
properties, 175
Worker object, 175–176
processes, 159
executable files, executing on another process, 168–169
process execution, 161
process I/O pipes, 159–160
process signals, 160
returning information about, 161–164
spawning, 169–171
system command, executing on another process, 166–168
socket services, 139–140
net.Server object, 144–147
net.Socket object, 140–144
TCP socket clients, 147–150
TCP socket servers, 150–152
TLS socket clients, 153–154
TLS socket servers, 154–156
TLS/SSL, 152–153
streams, 81
compressing/decompressing, 92–93
Duplex streams, 86–88
piping, 89–90
Readable streams, 82–84, 89–90
Transform streams, 88–89
Writable streams, 84–86, 89–90
strings
converting objects to, 184–185
formatting, 183–184
timers, 60
dereferencing, 63
immediate timers, 62
interval timers, 61–62
timeout timers, 60–61
use cases for, 40
Zlib, 91
compressing/decompressing buffers, 91–92
compressing/decompressing streams, 92–93
**Node.js modules. *See* modules (Node.js)**
**Node.js-to-Angular stack, 7**
components of, 11–14
web development framework
backend services, 10
browsers, 8–10
diagram of, 7–8
users, 8
webservers, 10
**nodelay option (server object), 224**
**–nohttpinterface parameter (mongod command), 203**
**–nojournal parameter (mongod command), 203**
**–noprealloc parameter (mongod command), 203**

**nor() method, 303**
**$nor operator, 263**
**normalizing data, 195–196**
**NoSQL, 191–192. *See also* MongoDB**
**not (!) operator, 19**
**not equal (!=) operator, 19**
**$not operator, 263**
**NPM (Node Package Manager), 44–45**
**npm command, 42, 44–45**
npm adduser, 50
npm install, 46, 51
npm pack, 50
npm search, 45
**npm install command, 475**
**NPMs (Node Packaged Modules). *See* modules (Node.js)**
**NPNProtocols option (tls.createServer), 155**
**null data type**
JavaScript, 17
TypeScript, 384
**number data type**
JavaScript, 16
TypeScript, 383
**number of MongoDB collections, 199**
**number pipe, 423**
**numberOfRetries option (options object), 265**

## O

**Object Document Model (ODM). *See* Mongoose**
**object literals, 17**
**objects, 27**
Admin, 229
Array, 32–33
adding items to, 34–35
combining, 33–34
converting into strings, 34
iterating through, 34
manipulating, 32–33
methods, 32–33
removing items from, 34–35
searching, 34
checking type of, 184
ChildProcess object, 164–166
ClientRequest, 118–121
Collection. *See* collections (MongoDB)
Connection, 292–293
converting JSON to, 74
converting to JSON, 74
converting to strings, 184–185
Cursor, 232–233
custom events, 64–65
custom-defined objects, 28
Db, 227–228
Document, 292, 304–305
event listeners
adding, 65
implementing, 65–67
removing, 65
EventEmitter, 64–65
IncomingMessage, 122–123
inheriting from, 185–186
Model, 292, 298
MongoClient, 222, 223–226
Observable
creating, 464–465
detecting data changes with, 465–468
importing, 510–512
options, 264–265, 275–276
properties, 27
prototyping object patterns, 29
Query, 262–264, 298–299
operators, 302–304
query database operation, 299–302

Request, 350–352
Response, 352
download responses, 359
files in, 356–358
headers, 352–353
JSON responses, 355–357
redirection, 359
rendered templates in, 363–364
sending, 353–355
status, 353
route, 487
Schema, 292, 294–295
Server, 123–125, 144–147
ServerResponse, 121–122
Socket
creating, 140–141
data flow across, 144
events, 141–142
methods, 142–143
properties, 143–144
Stats, 107
String, 29–31
combining, 31
converting arrays into, 34
escape codes, 29–30
manipulating, 29–31
methods, 30–31
replacing words in, 31
searching, 31
splitting, 32
syntax, 27–28
URL, 116–117
Worker, 175–176

**Observable object**
creating, 464–465
detecting data changes with, 465–468
importing, 510–512

**observable.component.html listing, 467**
**observable.component.ts listing, 466–467**
**observables**
definition of, 464
Observable object
creating, 464–465
detecting data changes with, 465–468
importing, 510–512

**ODM (Object Document Model). *See* Mongoose**
**once() function, 65**
**onDrop() method, 536**
**on() function, 65, 126, 160**
**online event, 174**
**open() method, 56, 96–97, 227**
**opening files, 96–97**
**openSync() method, 96–97**
**operators**
JavaScript, 17
arithmetic operators, 17–18
assignment operators, 18
comparison operators, 19–20
isinstanceof, 184
MongoDB
$add, 286
$addFields, 285
$addToSet, 243, 286
$all, 264
$and, 263
$avg, 286
$bit, 244
$bucket, 285
$bucketAuto, 285
$collStatus, 284
$concat, 287
$count, 285

$divide, 286
$each, 244
$elemMatch, 264, 266
$eq, 263
$exists, 263
$facet, 285
$first, 286
$geoNear, 285
$graphLookup, 285
$group, 284
$gt, 263
$gte, 263
$inc, 243, 263
$indexStats, 285
$last, 286
$limit, 284
$lookup, 285
$lt, 263
$lte, 263
$match, 283, 284
$max, 286
$min, 286
$mod, 263, 287
$multiply, 287
$ne, 263
$nin, 263
$nor, 263
$not, 263
$or, 263
$out, 285
$pop, 243
$project, 284
$pull, 244
$pullAll, 243
$push, 244, 286
$redact, 284
$regex, 264
$rename, 243
$replaceRoot, 285
$sample, 285
$set, 243
$setOnInsert, 243
$size, 264
$skip, 284
$slice, 244
$sort, 244, 284
$sortByCount, 285
$strcasecmp, 287
$substr, 287
$subtract, 287
$sum, 286
$toLower, 287
$toUpper, 287
$type, 263
$unset, 243
$unwind, 284
**options object, 264–265, 275–276**
**or() method, 303**
**or (||) operator, 19**
**$or operator, 263**
**originalUrl property (Request object), 351**
**os module, 41, 181–183**
**os_info.js listing, 182–183**
**$out operator, 285**
**outer.component.ts listing, 411**
**outer.css listing, 411**
**outer.html listing, 411**
**outlet property (route object), 487**

## P

**package.json file, 47–48, 50, 51**
**packages. *See* modules (Node.js)**
**page1.component.html listing, 499**

**page1.component.ts listing**
router with navigation bar, 495
router with parameters, 499
**page2.component.html listing**
router with navigation bar, 496
router with parameters, 500
**page2.component.ts listing**
router with navigation bar, 495
router with parameters, 500
**page3.component.html listing, 496**
**page3.component.ts listing, 496**
**paging results, 273–274**
**param() method, 349**
**parameters**
MongoDB shell commands, 205
passing to callback functions, 67–68
route parameters, applying, 347
callback functions, 348–349
with defined parameters, 348
express_routes.js example, 349–350
with query strings, 347–348
with regex, 348
router with parameters, 497–501
app.component.ts, 498
app-routing.module.ts, 498
page1.component.html, 499
page1.component.ts, 499
page2.component.html, 500
page2.component.ts, 500
**parent components, deleting data in, 461–464**
**parentheses ( ), 20, 24, 436, 457**
**parse() method, 74, 116, 117–118, 347**
**parseWeather() function, 132**
**parsing query strings, 117–118**
**partial option (options object), 265**
**partitioning methods, 337**
**passing data**
with dependency injection, 412–413
parameters to functions, 67–68
variables to functions, 24–25
**passphrase option**
htp.createServer(), 136
https.request(), 136
tls.connect(), 154
tls.createServer(), 155
**patch() method, 470**
**path module, 41**
**path property**
ClientRequest object, 119
Request object, 351
route object, 487
URL object, 117
**pathname property (URL object), 117**
**paths**
assigning Express middleware to, 368
document paths, 294
verifying, 106–107
**pause() method, 82, 142**
**pauseOnConnect option, 145**
**performance, MongoDB, 200**
**periodic work, scheduling**
with immediate timers, 62
with interval timers, 61–62
**Person interface, 385**
**person.component.ts listing, 413**
**PersonInterface, 386**
**pfx option**
htp.createServer(), 136
https.request(), 136
tls.connect(), 154
tls.createServer(), 155
**pid method, 162**
**pid property (ChildProcess object), 166**

**ping() method, 229**
**pipe class, 426**
**@pipe decorator, 426**
**pipe event, 84**
**pipe() method, 83, 89–90**
**pipe symbol (|), 426**
**pipes**
  built-in pipes
    builtInPipes.component.ts, 425
    table of, 422–424
  custom pipes, 426–428
  definition of, 422
  piping Readable streams to Writable streams, 89–90
  process I/O pipes, 159–160
**PiService, 505–506**
  app.component.html, 506
  app.component.ts, 505
  pi.service.ts, 505
**pi.service.ts listing, 505**
**platform() method, 162, 182**
**playLoop() function, 465**
**poolSize option (server object), 224**
**pop() method, 33**
**$pop operator, 243**
**–port parameter (mongod command), 202**
**port property**
  ClientRequest object, 119
  URL object, 116
**post() method, 470**
**post middleware functions, 322–324**
**POST requests**
  body data, handling, 371–372
  definition of, 9
**POST servers, 130–131**
**pre middleware functions, 322–324**
**preferGlobal directive, 47**
**preventDefault() method, 536**
**primary servers, 330**
**private keys, 135**
**process module, 41, 159**
  process execution, 161
  process I/O pipes, 159–160
  process signals, 160
  returning information from, 161–164
**process property (Worker object), 176**
**process signals, 160**
**process_info.js listing, 163–164**
**processes, 159**
  child processes, 159–160
    ChildProcess object, 164–166
    executable files, executing on another process, 168–169
    processes, spawning, 169–173
    system command, executing on another process, 166–168
  process clusters, 174
    events, 174
    HTTP clusters, 176–179
    methods, 175
    properties, 175
    Worker object, 175–176
  process execution, 161
  process I/O pipes, 159–160
  process signals, 160
  returning information about, 161–164
**profile command, 204**
**project() method, 318**
**$project operator, 284**
**PromiseService, 515–516**
**promise.service.ts listing, 515**
**properties**
  binding, 431–433
  ClientRequest object, 119

cluster module, 175
definition of, 27
Document object, 304–305
IncomingMessage object, 123
net.Socket object, 143–144
Request object, 351
URL object, 116–117
Worker object, 176
**property binding, 431–433**
**property.component.ts listing, 432**
**protocol property**
Request object, 351
URL object, 116
**prototyping object patterns, 29**
**providers metadata option (NgModule), 397**
**publishing modules, 50–51**
**$pull operator, 244**
**$pullAll operator, 243**
**push() method, 33**
**$push operator, 244, 286**
**put() method, 470**
**PUT requests**
response callback functions, 471
sending, 470–471

## Q

**queries**
MongoDB query object, 262–264
Mongoose Query object, 298–299
operators, 302–304
query database operation, 299–302
options objects, 264–265
query strings, 41
applying route parameters with, 347–348
processing, 117–118
result sets, 270
aggregating, 282–289
grouping results, 277–282
limiting by size, 270–271
limiting fields returned in, 271–272
paging results, 273–274
sorting, 275–276
**query middleware, 368, 369**
**query object (MongoDB), 262–264**
**Query object (Mongoose), 298–299**
operators, 302–304
query database operation
options, 301–302
setting, 299–302
**query property**
Request object, 351
URL object, 117
**query router servers, 335, 338–339**
**query strings, 41**
applying route parameters with, 347–348
processing, 117–118
**queryRemover() function, 380**
**–quiet parameter (mongod command), 202**

## R

**\r (carriage return) escape code, 29**
**RandomImageService, 510–514**
app.component.css, 513–514
app.component.html, 513
app.component.ts, 512–513
random-image.service.ts, 511–512
**random-image.service.ts listing, 511–512**
**range-based sharding, 337**
**rated-item component, 543–544**
rated-item.component.css, 545–546
rated-item.component.html, 545
rated-item.component.ts, 544–545

RatedItem interface, 541
ratings.service.ts listing, 541–542
Read Event Print Loop (REPL) module, 41, 190
read() method, 82, 104–105, 302, 318
read option (Schema object), 295
read role (MongoDB), 208
readable event, 82
Readable streams, 82–84, 89–90
readAnyDatabase role (MongoDB), 208
readConcern option (server object), 224
readdir() method, 109
readdirSync() method, 109
readFile() method, 102–103
reading files
  asynchronous file reading, 104–105
  simple file read, 102–103
  streamed file reading, 105–106
  synchronous file reading, 103–104
readInt8() method, 77
readInt16BE() method, 77
readInt16LE() method, 77
readline module, 41
readPreference option
  options object, 265
  server object, 224
readSync() method, 103–104
readwords.js listing, 52
readWrite role (MongoDB), 208
readWriteAnyDatabase role (MongoDB), 208
receiving cookies, 373–374
ReconnectInterval option (server object), 224
reconnectTries option (server object), 224
$redact operator, 284
redirect() method, 359
redirecting responses, 359
redirectTo property (route object), 487
ref() method, 63, 143, 146
references, MongoDB documents, 195–196
regenerate() method, 378
regex, applying route parameters with, 348
regex() method, 303
$regex operator, 264
Registry, Node Package Registry
  publishing modules to, 50–51
  viewing, 43–44
rejectUnauthorized option
  htp.createServer(), 136
  https.request(), 136
  tls.connect(), 154
  tls.createServer(), 155
release() method, 182
remoteAddress property (Socket object), 143
remoteFamily property (Socket object), 143
remotePort property (Socket object), 143
remove() method, 217, 230, 255, 301, 305, 314–316
removeHeader() method, 122
removeListener() function, 65
removeShardTag() method, 341
removeUser() method, 209, 228, 229
removing data. *See* deleting data
rename() method, 112, 230
$rename operator, 243
renameCollection() method, 228
renameSync() method, 112
renaming files/directories, 112
render() method, 363–364
rendered templates, sending, 363–364
rendering browser view, 9–10
–repair parameter (mongod command), 203
repairDatabase() method, 341
repairing databases, 341
REPL (Read Event Print Loop) module, 41, 190

**replace() method, 30, 31**
**$replaceRoot operator, 285**
**replacing words in strings, 31**
**replica sets, deploying, 333–334**
**replication**
applying, 330–332
replica sets, 333–334
strategy, 332–333
**replications, 199**
**repository directive, 48**
**request event, 124**
**request() method, 118, 136**
**Request objects, 350–352**
**requestCert option (tls.createServer), 155**
**requests, 9**
ClientRequest object, 118–121
configuration, 471
GET requests, 470–471
GetData, 55–56
GetFile, 55–56
POST, 371–372
PUT requests, 470–471
request event, 124
Request objects, 350–352
response callback functions, 471
**require() method, 52**
**required fields, forcing, 296**
**requiredPaths() method, 296**
**resolve() method, 117, 186, 515**
**Resolve property (route object), 487**
**resolve4() method, 187**
**resolve6() method, 187**
**resolveCname() method, 187**
**resolveMx() method, 187**
**resolveNs() method, 187**
**resolveSrv() method, 187**
**resolveTxt() method, 187**
**resolving URLs (Uniform Resource Locators), 117**
**response callback functions (HTTP), 471**
**response event, 120**
**Response objects, 352**
download responses, 359
headers, 352–353
JSON responses, 355–357
redirection, 359
sending, 353–355
sending files in, 356–358
sending rendered templates in, 363–364
status, 353
**responses**
IncomingMessage object, 122–123
response event, 120
Response objects, 352
download responses, 359
headers, 352–353
JSON responses, 355–357
redirection, 359
sending, 353–355
sending files in, 356–358
sending rendered templates in, 363–364
status, 353
ServerResponse object, 121–122
**responsibilities, separation of, 393**
**result sets, 270**
aggregating, 282–289
aggregate() method, 283
aggregation examples, 287–289
aggregation expression operators, 285–287
aggregation framework operators, 283–285

grouping results, 277–282
limiting by size, 270–271
limiting fields returned in, 271–272
paging results, 273–274
sorting, 275–276
**resume() method, 82, 142**
**resumeSession event, 156**
**return statement, 25**
**return values, 25**
**reusable code, 14, 361**
**reverse() method, 33, 187**
**rewind() method, 233**
**rmdir() method, 111**
**rmdirSync() method, 111**
**roles (MongoDB), 208**
**roles command, 204**
**root component (custom directive), 453**
CSS for, 454
HTML for, 453–454
**route object, 487**
**route2.component.html listing, 491**
**route2.component.ts listing, 491**
**Router, importing, 488**
**Router module, importing, 492**
**router service, 470**
ActivatedRoute, importing, 488
route navigation, 488
route object parameters, 487
Router, importing, 488
router with navigation bar, 492–497
app.component.CSS, 495
app.component.html, 494
app.component.ts, 494
app-routing.module.ts, 492–493
nav.component.CSS, 494
nav.component.html, 493
nav.component.ts, 493
page1.component.ts, 495
page2.component.html, 496
page2.component.ts, 495
page3.component.html, 496
page3.component.ts, 496
router with parameters, 497–501
app.component.ts, 498
app-routing.module.ts, 498
page1.component.html, 499
page1.component.ts, 499
page2.component.html, 500
page2.component.ts, 500
Routes array, defining, 486–487
routing module, including, 488
simple router implementation, 488–491
app.component.ts, 490
app.module.ts, 488–489
app-routing.module.ts, 489–490
home.component.html, 490
home.component.ts, 490
route2.component.html, 491
route2.component.ts, 491
**router-outlet tag, 488**
**routes**
applying parameters in, 347
callback functions, 348–349
with defined parameters, 348
express_routes.js example, 349–350
with query strings, 347–348
with regex, 348
assigning Express middleware to, 368
definition of, 345
implementing, 346
**Routes array, defining, 486–487**
**routing module, including, 488**
**runGaurdsAndResolvers property (route object), 487**

## S

**safe() method, 302**
**safe option (Schema object), 295**
**$sample operator, 285**
**save() method, 217, 218–219, 230, 252–253, 305, 308, 310, 378**
**saving documents, 252–253, 310–311**
**scalability**
of MongoDB, 3, 13
of Node.js, 3, 12
**scheduling work**
nextTick() function, 63–64
timers, 60
dereferencing, 63
immediate timers, 62
interval timers, 61–62
timeout timers, 60–61
**Schema object, 292, 294–295**
**schema property (Document object), 305**
**schemas**
defining, 294–295
implementing on database, 295–296
indexes, adding, 295–296
methods, adding, 295–296
paths, 294
required fields, forcing, 296
Schema object, 292, 294–295
unique fields, 296
value types, 294–298
**schemas metadata option (NgModule), 397**
**scope of variables, 26–27**
**scripts, 205–206. *See also* listings**
**scripts directive, 47**
**search() method, 30**
**search property (URL object), 117**
**searching**
arrays, 34
collections, 216
documents, 216, 265–268
field values, 276–277
for NPMs (Node Packaged Modules), 45
strings, 31
**secondary servers, 331**
**SecretAgent class, 387**
**secure property (Request object), 351**
**secureConnection event, 156**
**secureProtocol option**
htp.createServer(), 136
https.request(), 136
tls.connect(), 154
tls.createServer(), 155
**select() method, 301**
**selectCharacter() function, 461**
**selector option (components), 403**
**selectors, 404**
**selectUser() method, 482**
**send() method, 165, 176, 353–355**
**sendData event, 121**
**sendfile() method, 356–358**
**sending**
cookies, 373–374
rendered templates, 363–364
requests, 470–471
responses
download responses, 359
JSON responses, 355–357
redirection, 359
send() method, 353–355
**separation of responsibilities, 393**
**serializeFunctions option, 242**
**Server object, 123–125, 144–147**
**server status, displaying, 236–237**

**servername option (tls.connect), 154**
**ServerResponse object, 121–122**
**servers. *See also* mock web servers**
arbiter, 331
config servers, 335, 338
Express server, starting, 345
HTTP (Hypertext Transfer Protocol)
dynamic GET servers, 127–129
external sources, interacting with, 132–134
POST servers, 130–131
Server object, 123–125
static files, serving, 125–127
HTTPS, 137
MongoDB server status, displaying, 236–237
primary servers, 330
query router servers, 335, 338–339
secondary servers, 331
shard servers, 335
TCP socket servers, 150–152
TLS socket servers, 154–156
**server-side programs, 10**
**serverStatus() method, 229**
**services (Angular)**
animate, 470
animation service, application using
animated.component.css, 529
animated.component.html, 529
animated.component.ts, 527–528
app.component.html, 526–527
app.component.ts, 526
app.module.ts, 525–526
folder structure, 525
constant data service, 505–506
app.component.html, 506
app.component.ts, 505
pi.service.ts, 505
data transform service, 506–510
app.component.css, 509
app.component.html, 508–509
app.component.ts, 507–508
area-calc.service.ts, 506–507
definition of, 393
forms, 470
http, 470
GET requests, sending, 470–471
JSON file implementation and access, 472–475
PUT requests, sending, 470–471
request configuration, 471
response callback functions, 471
simple mock server implementation, 475–481
simple mock server updates, 481–486
integrating into applications, 503–504
mock back-end service, 540–541
purpose of, 469
ratings service, 541–542
router, 470
ActivatedRoute, importing, 488
route navigation, 488
route object parameters, 487
Router, importing, 488
router with navigation bar, 492–497
router with parameters, 497–501
Routes array, 486–487
routing module, including, 488
simple router implementation, 488–491
service that returns a promise, 515–516
shared service, 516–523
app.component.html, 519
app.component.ts, 519
badguys.component.css, 522

badguys.component.html, 522
badguys.component.ts, 521
good-guys.component.css, 521
good-guys.component.html, 520
good-guys.component.ts, 520
shared.service.ts, 517–518
use cases for, 503–504
UserService, 483–484
variable data service, 510–514
app.component.css, 513–514
app.component.html, 513
app.component.ts, 512–513
random-image.service.ts, 511–512
**session middleware, 368, 377–380**
**sessionIdContext option (tls.createServer), 155**
**sessions**
implementing, 374–375
session authentication, 377–380
session middleware, 368
**set() method, 304, 344, 352**
**$set operator, 243**
**setEncoding() method, 82, 142**
**setgid() method, 162**
**setgroups() method, 163**
**setHeader() method, 122**
**setImmediate() function, 62**
**setInterval() function, 61–62**
**setKeepAlive() method, 143**
**setMaxListeners() function, 65**
**setNoDelay() method, 121, 143**
**$setOnInsert operator, 243**
**setOptions() method, 301**
**setRating() method, 544**
**sets of documents, finding, 265–268**
**setSocketKeepAlive() method, 121**
**setTimeout() function, 60–61, 120, 122, 123, 143, 510**
**settings property (cluster module), 175**
**setuid() method, 162**
**setup event, 174**
**setupMaster() method, 175**
**shard keys, 336–337**
**shard tag ranges, 340–341**
**sharding, 199**
definition of, 334
enabling on collections, 340
enabling on databases, 339–340
hash-based sharding, 337
partitioning methods, 337
range-based sharding, 337
shard keys, 336–337
shard tag ranges, 340–341
sharded MongoDB clusters, 338
adding shards to cluster, 339
config server database instances, 338
query router servers, 338–339
sharding server types, 335
**shared service, 516–523**
app.component.html, 519
app.component.ts, 519
badguys.component.css, 522
badguys.component.html, 522
badguys.component.ts, 521
good-guys.component.css, 521
good-guys.component.html, 520
good-guys.component.ts, 520
shared.service.ts, 517–518
**SharedService. *See* shared service**
**shared.service.ts listing, 517–518**
**shell clients, accessing MongoDB from, 203–204**
command parameters, 205
shell commands, 204

shell methods, 205
shell scripts, 205–206
**shift() method, 33**
**show command, 204**
**SIGBREAK event, 160**
**SIGHUP event, 160**
**SIGINT event, 160**
**SIGKILL event, 160**
**sign class, 189**
**signals, process, 160**
**SIGPIPE event, 160**
**SIGSTOP event, 160**
**SIGTERM event, 160**
**SIGUSR1 event, 160**
**SIGWINCH event, 160**
**silent property (fork() function), 172**
**simple_interval.js listing, 61–62**
**simple_timer.js listing, 60–61**
**single documents**
removing with Mongoose, 314–315
updating with Mongoose, 311–313
**single field indexes, 328**
**size, limiting results by, 270–271**
**size() method, 304**
**$size operator, 264**
**size option (collections), 214**
**skip() method, 302, 318**
**$skip operator, 284**
**skip option (options object), 264**
**slice() method, 30, 33, 80**
**$slice operator, 244**
**slice:start:end pipe, 423**
**slicing buffers, 80**
**snapshot() method, 302**
**snapshot option (options object), 265**
**SNICallback option (tls.createServer), 155**
**socket event, 120**
**Socket objects**
creating, 140–141
data flow across, 144
events, 141–142
methods, 142–143
properties, 143–144
**socket property (IncomingMessage object), 123**
**socket services, 139–140**
net.Server object, 144–147
net.Socket object
creating, 140–141
data flow across, 144
events, 141–142
methods, 142–143
properties, 143–144
TCP socket clients, 147–150
TCP socket servers, 150–152
TLS socket clients, 153–154
TLS socket servers, 154–156
**socket_client.js listing, 148–149**
**socket_server.js listing, 151–152**
**socketPath property (ClientRequest object), 119**
**socketTimeOut option (server object), 224**
**soldierOfGondor() function, 388**
**sort() method, 33, 233, 302, 318**
**$sort operator, 244, 284**
**sort option (options object), 264, 275–276**
**$sortByCount operator, 285**
**sorting result sets, 275–276**
**spare property (indexes), 329**
**spawn() method, 169–171**
**spawning processes, 169–171**
**splice() method, 33**
**split() method, 30, 32**

**splitting strings, 32**
**SQL injection, 13**
**ssl option (server object), 224**
**stale property (Request object), 351**
**star rating application**
app.component.css, 544
app.component.html, 543–544
app.component.ts, 543
app.module.ts, 540
folder structure, 539
mockbackend.service.ts, 540–541
rated-item.component.css, 545–546
rated-item.component.html, 545
rated-item.component.ts, 544–545
ratings.service.ts, 541–542
**starting**
Express server, 345
MongoDB, 202–203
query router servers, 338–339
**statements. *See also* keywords**
break, 23–24
catch, 35–36
continue, 23–24
finally, 36–37
if, 20
return, 25
switch, 20–21
throw, 36
try, 35–36
**static files, serving, 125–127, 369–371**
**static middleware, 367, 369–371**
**./static/css/static.css listing, 370**
**statistics, accessing**
collection statistics, 239–240
file statistics, 107–108
**stats() method, 107–108, 232**
**Stats object, 107**
**statsSync() method, 107**
**status of Response objects, 353**
**statusCode event, 121, 123**
**stderr, 159, 166**
**stdin, 159, 166**
**stdio property (spawn() function), 170**
**stdout, 159, 166**
**stopping MongoDB, 203**
**strategy, replication, 332–333**
**$strcasecmp operator, 287**
**stream module. *See* streams**
**stream_duplex.js listing, 87**
**stream_piped.js listing, 90**
**stream_read.js listing, 83–84**
**stream_transform.js listing, 88–89**
**stream_write.js listing, 85**
**streams, 81**
compressing/decompressing, 92–93
Duplex streams, 86–88
piping, 89–90
Readable streams, 82–84, 89–90
streamed file reading, 105–106
streamed file writing, 101–102
Transform streams, 88–89
Writable streams, 84–86, 89–90
**strict option (Schema object), 295**
**strict routing setting (Express), 344**
**string decoder module, 41**
**String object, 29–31**
combining, 31
converting arrays into, 34
escape codes, 29–30
manipulating, 29–31
methods, 30–31
replacing words in, 31
searching, 31
splitting, 32

**stringify() method, 74, 118**
**strings, 16**
converting objects to, 184–185
formatting, 183–184
JavaScript
combining, 31
converting arrays into, 34
escape codes, 29–30
manipulating, 29–31
methods, 30–31
replacing words in, 31
searching, 31
splitting, 32
query strings, 41
applying route parameters with, 347–348
processing, 117–118
TypeScript, 383
**Stringy interface, 385**
**structural directives, 392, 442–445**
definition of, 441
ngFor, 442
ngIf, 442–443
ngSwitch, 442–443
ngSwitchCase, 442, 443
ngSwitchDefault, 442
structural.component.ts, 443–444
**structural.component.ts listing, 443–444**
**style binding, 435–436**
**style.component.ts listing, 435**
**styles keyword, 405**
**styles option (components), 404**
**stylesUrls option (components), 404**
**styleUrls keyword, 408**
**(submit) event, 457**
**subscribe() method, 464**
**substr() method, 31**
**$substr operator, 287**
**substrings, searching for, 31**
**$subtract operator, 287**
**subtraction (-) operator, 18**
**suicide property (Worker object), 176**
**$sum operator, 286**
**switch statements, 20–21**
**synchronous file reading, 103–104**
**synchronous file system calls, 95**
**synchronous file writing, 98–99**
**system command, executing on another process, 166–168**

## T

**\t (tab) escape code, 30**
**tab escape code, 30**
**tags, router-outlet, 488**
**TCP (Transmission Control Protocol)**
socket clients, 147–150
socket servers, 150–152
**template engines, 360**
defining, 360–361
locals, adding, 361
rendered templates, sending, 363–364
template creation, 361–363
**template option (components), 404**
**templates**
building, 404–405
creating, 361–363
external templates, 408–410
template engines, 360
defining, 360–361
locals, adding, 361
rendered templates, sending, 363–364
template creation, 361–363
**templateUrl keyword, 408**
**templateUrl option (components), 404**
**text indexes, 328**

**then() method, 472–473**
**this keyword, 323**
**threaded event models, 55–56**
**throw statement, 36**
**throwing errors, 36**
**tilde (~), 404–405**
**time() function, 53**
**timeEnd() function, 54**
**timeout event, 141**
**timeout property**
exec() method, 167
options object, 265
**timeout timers, 60–61**
**timers, 41, 60**
dereferencing, 63
immediate timers, 62
interval timers, 61–62
timeout timers, 60–61
**time-to-live (TTY) values**
documents, 199
indexes, 329
**title method, 162**
**tls module, 41, 139–140, 152–153. *See also* socket services**
TLS socket clients, 153–154
TLS socket servers, 154–156
**tmpdir() method, 181**
**toArray() method, 232**
**toJSON() method, 305**
**$toLower operator, 287**
**toLowerCase() method, 31**
**toObject() method, 305**
**toPromise() method, 471, 472–473**
**toString() method, 33, 77, 305**
**totalmem() method, 182**
**touch() method, 378**
**$toUpper operator, 287**
**toUpperCase() method, 31**
**trace() function, 54**
**trailers property (IncomingMessage object), 123**
**transform() method, 88**
**Transform streams, 88–89**
**Transmission Control Protocol. *See* TCP (Transmission Control Protocol)**
**truncate() method, 110–111**
**truncateSync() method, 110–111**
**truncating files, 110–111**
**trust proxy setting (Express), 344**
**try statement, 35–36**
**TTY (time-to-live) values**
documents, 199
indexes, 329
**two-way binding, 439–440**
**twoWay.component.ts listing, 439–440**
**type() method, 181, 353**
**$type operator, 263**
**types. *See* data types**
**TypeScript, 383. *See also* listings**
in Angular expressions, 419–422
classes
defining, 386
inheritance, 387
data types, 383–384
directives, 462–463
functions, 388–389
interfaces, 385–386
modules, 387–388
**typescriptExpressions.component.ts listing, 421**

## U

**ucs2 encoding, 75**
**uid property (spawn() function), 170**

Uniform Resource Locators. *See* URLs (Uniform Resource Locators)
unique fields, 296
unique property (indexes), 329
unlink() method, 110
unlinkSync() method, 110
unpipe event, 85
unpipe() method, 83, 89–90
unref() method, 63, 143, 146
$unset operator, 243
unshift() method, 33
unwind() method, 318
$unwind operator, 284
update() method, 218–219, 231, 248–250, 301, 304, 311, 313
update operators, 242–244
update.component.CSS listing, 485–486
update.component.html listing, 484–485
update.component.ts listing, 482–483
updateRating() method, 541–542
updateUser() method, 482, 483
updating
  documents
    findAndRemove() method, 218–219
    with Mongoose, 309–314
    update() method, 248–250
  mock web server items, 481–486
    data.service.ts, 481–482
    update.component.CSS, 485–486
    update.component.html, 484–485
    update.component.ts, 482–483
    user.service.ts, 483–484
upgrade event, 120, 124
uppercase pipe, 423
upsert option, 242
upserting documents, 253–254
uptime() method, 162, 182
URL object, 116–117
url property (IncomingMessage object), 123
URLs (Uniform Resource Locators), 41
  resolving, 117
  structure of, 115–116
  URL object, 116–117
use command, 204
use() method, 368
user accounts (MongoDB)
  creating, 206–207
  listing, 206–207
  removing, 209
  roles, 208
User Administrator accounts, 209–210
user_ejs.html listing, 362
userAdmin role (MongoDB), 208
userAdminAnyDatabase role (MongoDB), 208
users, 8, 10
users command, 204
UserService, 483–484
user.service.ts listing, 478–479, 483–484
utf8 encoding, 75
utf16le encoding, 75
util module, 183
  format() method, 183–184
  inherits() method, 185–186
  inspect() method, 184–185
  isinstanceof operator, 184
util_inherit.js listing, 185–186
utilities module, 41
UUID() method, 205

## V

V8 module, 41, 190
validate() method, 305, 320
validation framework (Mongoose), 320–322

**validationAction option (collections), 215**
**validationLevel option (collections), 215**
**validator option (collections), 214**
**valueOf() method, 31, 33**
**var keyword, 15–16**
**variable data service, 510–514**
app.component.css, 513–514
app.component.html, 513
app.component.ts, 512–513
random-image.service.ts, 511–512
**variables**
defining, 15–16
local template variables, 361
passing to functions, 24–25
scope, 26–27
**–verbose parameter (mongod command), 202**
**verify class, 189**
**verifying**
file path, 106–107
Node.js executables, 42–43
**version directive, 47**
**version method, 161**
**–version parameter (mongod command), 202**
**versions method, 161**
**view cache setting (Express), 344**
**view engine setting (Express), 344**
**viewProviders option (components), 404**
**views (browser), rendering, 9–10**
**views setting (Express), 344**
**Visual Studio Code, 394**
**VM module, 41**
**void data type, 384**

## W

**w option, 224, 241**
**warn() function, 53**
**watchFile() method, 112–113**
**watching for file changes, 112–113**
**web applications. *See* components (Angular); services (Angular)**
**web browsers. *See* browsers**
**web development framework**
backend services, 10
browsers, 8
browser view, rendering, 9–10
browser-to-webserver communication, 8–9
user interaction, 10
diagram of, 7–8
users, 8
webservers, 10
**web servers. *See* mock web servers**
**webservers, 10**
**where() method, 302**
**while loops, 21**
**word_schema.js listing, 297**
**words, replacing in strings, 31**
**work, scheduling**
adding to event queue, 59–60
nextTick() function, 63–64
timers, 60
dereferencing, 63
immediate timers, 62
interval timers, 61–62
timeout timers, 60–61
**Worker object, 175–176**
**worker property (cluster module), 175**
**workers property (cluster module), 175**
**Writable streams, 84–86, 89–90**
**write concern, 222**
**write() method, 76, 84–85, 118, 120, 122, 128, 142, 151**
**write operations, 198**
**writeContinue() method, 122**
**writeFile() method, 98**

**writeInt8() method, 76**
**writeInt16BE() method, 76**
**writeInt16LE() method, 76**
**writeSync() method, 98–101**
**writing**
to console, 53–54
files
asynchronous file writing, 99–101
simple file write, 98
streaming file writing, 101–102
synchronous file writing, 98–99
**wtimeout option, 241**
**wTimeOut option (server object), 224**

## X-Y-Z

**x509 public key, 135, 137**
**Zlib module, 41, 91**
compressing/decompressing buffers, 91–92
compressing/decompressing streams, 92–93
**zlib_buffers.js listing, 91–92**
**zlib_file.js listing, 93**
**zoom application**
app.component.html, 531
app.component.ts, 530–531
folder structure, 530
zoomit.component.css, 532
zoomit.component.html, 532
zoomit.component.ts, 531–532
**zoom directive, 449–452**
**ZoomDirective export class, 451**
**zoom.directive.ts listing, 450–451**
**zoomit component**
zoomit.component.css, 532
zoomit.component.html, 532
zoomit.component.ts, 531–532