

Brad Dayley



# Node.js, MongoDB and AngularJS

## Web Development



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



# Node.js, MongoDB and AngularJS Web Development

---

# Developer's Library

ESSENTIAL REFERENCES FOR PROGRAMMING PROFESSIONALS

*Developer's Library* books are designed to provide practicing programmers with unique, high-quality references and tutorials on the programming languages and technologies they use in their daily work.

All books in the *Developer's Library* are written by expert technology practitioners who are especially skilled at organizing and presenting information in a way that's useful for other programmers.

Key titles include some of the best, most widely acclaimed books within their topic areas:

*PHP & MySQL Web Development*

Luke Welling & Laura Thomson

ISBN 978-0-321-83389-1

*MySQL*

Paul DuBois

ISBN-13: 978-0-321-83387-7

*Linux Kernel Development*

Robert Love

ISBN-13: 978-0-672-32946-3

*Python Essential Reference*

David Beazley

ISBN-13: 978-0-672-32978-4

*PostgreSQL*

Korry Douglas

ISBN-13: 978-0-672-32756-2

*C++ Primer Plus*

Stephen Prata

ISBN-13: 978-0321-77640-2

Developer's Library books are available at most retail and online bookstores, as well as by subscription from Safari Books Online at [safari.informit.com](http://safari.informit.com)

**Developer's  
Library**

[informit.com/devlibrary](http://informit.com/devlibrary)

# Node.js, MongoDB and AngularJS Web Development

---

Brad Dayley

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco  
New York • Toronto • Montreal • London • Munich • Paris • Madrid  
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-99578-0

ISBN-10: 0-321-99578-3

Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.

Second Printing: September 2014 with corrections December 2014

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [international@pearsoned.com](mailto:international@pearsoned.com).

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

Library of Congress Control Number: 2014936529

**Acquisitions  
Editor**

Mark Taber

**Managing  
Editor**

Kristy Hart

**Project Editor**

Elaine Wiley

**Copy Editor**

Kitty Wilson

**Indexer**

Brad Herriman

**Proofreader**

Debbie Williams

**Technical  
Reviewers**

Russell Kloepfer

Siddhartha Singh

**Cover Designer**

Chuti Prasertsith

**Senior Compositor**

Gloria Schurick



*For D!*

*A & F*



# Contents

## **I: Introduction 1**

- Who Should Read This Book 1
- Why You Should Read This Book 1
- What You Will Learn from This Book 2
- What Is Node.js? 2
- What Is MongoDB? 3
- What Is AngularJS? 4
- How This Book Is Organized 4
- Getting the Code Examples 5
- A Final Word 5

## **1 Introducing the Node.js-to-AngularJS Stack 7**

- Understanding the Basic Web Development Framework 7
- Understanding the Node.js-to-AngularJS Stack Components 11
- Summary 14
- Up Next 14

## **2 JavaScript Primer 15**

- Defining Variables 15
- Understanding JavaScript Data Types 16
- Using Operators 17
- Implementing Looping 21
- Creating Functions 24
- Understanding Variable Scope 26
- Using JavaScript Objects 27
- Manipulating Strings 30
- Working with Arrays 32
- Adding Error Handling 36
- Summary 38
- Up Next 38

## II: Learning Node.js

### 3 Getting Started with Node.js 39

- Understanding Node.js 39
- Installing Node.js 40
- Working with Node.js Packages 42
- Creating a Node.js Application 47
- Writing Data to the Console 52
- Summary 53
- Up Next 54

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

- Understanding the Node.js Event Model 55
- Adding Work to the Event Queue 59
- Implementing Callbacks 68
- Summary 72
- Up Next 72

### 5 Handling Data I/O in Node.js 73

- Working with JSON 73
- Using the `Buffer` Module to Buffer Data 74
- Using the `Stream` Module to Stream Data 83
- Compressing and Decompressing Data with `zlib` 93
- Summary 96
- Up Next 96

### 6 Accessing the File System from Node.js 97

- Synchronous Versus Asynchronous File System Calls 97
- Opening and Closing Files 98
- Writing Files 100
- Reading Files 105
- Other File System Tasks 110
- Summary 117
- Up Next 117

- 7 Implementing HTTP Services in Node.js 119**
  - Processing URLs 119
  - Processing Query Strings and Form Parameters 121
  - Understanding Request, Response, and Server Objects 122
  - Implementing HTTP Clients and Servers in Node.js 130
  - Implementing HTTPS Servers and Clients 139
  - Summary 142
  - Up Next 142
  
- 8 Implementing Socket Services in Node.js 143**
  - Understanding Network Sockets 143
  - Understanding TCP Server and Socket Objects 144
  - Implementing TCP Socket Servers and Clients 151
  - Implementing TLS Servers and Clients 157
  - Summary 162
  - Up Next 162
  
- 9 Scaling Applications Using Multiple Processors in Node.js 163**
  - Understanding the `process` Module 163
  - Implementing Child Processes 168
  - Implementing Process Clusters 178
  - Summary 183
  - Up Next 184
  
- 10 Using Additional Node.js Modules 185**
  - Using the `os` Module 185
  - Using the `util` Module 187
  - Summary 193
  - Up Next 193

### III: Learning MongoDB

- 11 Understanding NoSQL and MongoDB 195**
  - Why NoSQL? 195
  - Understanding MongoDB 196
  - MongoDB Data Types 197

Planning Your Data Model 199  
Summary 205  
Up Next 205

## **12 Getting Started with MongoDB 207**

Building the MongoDB Environment 207  
Administering User Accounts 212  
Configuring Access Control 216  
Administering Databases 218  
Managing Collections 220  
Summary 226  
Up Next 226

## **13 Getting Started with MongoDB and Node.js 227**

Adding the MongoDB Driver to Node.js 227  
Connecting to MongoDB from Node.js 228  
Understanding the Objects Used in the MongoDB  
Node.js Driver 235  
Accessing and Manipulating Databases 241  
Accessing and Manipulating Collections 245  
Summary 249  
Up Next 249

## **14 Manipulating MongoDB Documents from Node.js 251**

Understanding Database Change Options 251  
Understanding Database Update Operators 252  
Adding Documents to a Collection 254  
Getting Documents from a Collection 256  
Updating Documents in a Collection 258  
Atomically Modifying Documents in a Collection 260  
Saving Documents in a Collection 262  
Using `upsert` to Insert Documents in a Collection 263  
Deleting Documents from a Collection 265  
Removing a Single Document from a Collection 266  
Summary 268  
Up Next 268

**15 Accessing MongoDB Documents from Node.js 269**

- Introducing the Data Set 269
- Understanding Query Objects 270
- Understanding Query `options` Objects 272
- Finding Specific Sets of Documents 273
- Counting Documents 276
- Limiting Result Sets 278
- Sorting Result Sets 283
- Finding Distinct Field Values 285
- Grouping Results 286
- Applying MapReduce by Aggregating Results 289
- Summary 295
- Up Next 295

**16 Using Mongoose for Structured Schema and Validation 297**

- Understanding Mongoose 297
- Connecting to a MongoDB Database by Using Mongoose 298
- Defining a Schema 300
- Compiling a Model 304
- Understanding the `Query` Object 305
- Understanding the `Document` Object 310
- Finding Documents by Using Mongoose 312
- Adding Documents by Using Mongoose 314
- Updating Documents by Using Mongoose 316
- Removing Documents by Using Mongoose 320
- Aggregating Documents by Using Mongoose 323
- Using the Validation Framework 326
- Implementing Middleware Functions 328
- Summary 331
- Up Next 331

**17 Advanced MongoDB Concepts 333**

- Adding Indexes 333
- Using Capped Collections 336
- Applying Replication 337
- Implementing Sharding 340

Implementing a GridFS Store	348
Repairing a MongoDB Database	353
Backing Up MongoDB	354
Summary	355
Up Next	356

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

<b>18 Implementing Express in Node.js</b>	<b>357</b>
Getting Started with Express	357
Configuring Routes	359
Using Request Objects	365
Using Response Objects	366
Implementing a Template Engine	374
Summary	379
Up Next	379
<b>19 Implementing Express Middleware</b>	<b>381</b>
Understanding Middleware	381
Using the <code>query</code> Middleware	383
Serving Static Files	384
Handling POST Body Data	386
Sending and Receiving Cookies	387
Implementing Sessions	388
Applying Basic HTTP Authentication	390
Implementing Session Authentication	392
Creating Custom Middleware	395
Summary	396
Up Next	396

#### **V: Learning AngularJS**

<b>20 Getting Started with AngularJS</b>	<b>397</b>
Why AngularJS?	397
Understanding AngularJS	398
An Overview of the AngularJS Life Cycle	401
Integrating AngularJS with Existing JavaScript and jQuery	402

- Adding AngularJS to the Node.js Environment 403
- Bootstrapping AngularJS in an HTML Document 403
- Using the Global APIs 404
- Creating a Basic AngularJS Application 405
- Summary 409
- Up Next 409

**21 Understanding AngularJS Modules and Dependency Injection 411**

- Overview of Modules and Dependency Injection 411
- Defining AngularJS Modules 412
- Implementing Dependency Injection 416
- Summary 418
- Up Next 418

**22 Implementing the Scope as a Data Model 419**

- Understanding Scopes 419
- Implementing Scope Hierarchy 425
- Emitting and Broadcasting Events 428
- Summary 431
- Up Next 432

**23 Using AngularJS Templates to Create Views 433**

- Understanding Templates 433
- Using Expressions 434
- Using Filters 437
- Creating Custom Filters 443
- Summary 445
- Up Next 445

**24 Implementing Directives in AngularJS Views 447**

- Understanding Directives 447
- Using Built-in Directives 447
- Creating Your Own Directives to Extend HTML 461
- Summary 469
- Up Next 469

**25 Implementing AngularJS Services in Web Applications 471**

- Understanding AngularJS Services 471
- Using the Built-in Services 472
- Creating Custom Services 487
- Summary 489
- Up Next 490

**VI: Building Practical Web Application Components****26 Adding User Accounts to Your Website 491**

- Libraries Used 491
- Project Directory Structure 491
- Defining the User Model 492
- Creating the Server 493
- Implementing Routes 494
- Implementing the User Controller Routes 496
- Implementing the User and Authentication Views 501
- Implementing the AngularJS Module and Controller 507
- Using Social Media Accounts as Authentication Sources 508
- Summary 514
- Up Next 514

**27 Adding Comment Threads to Pages 515**

- Libraries Used 515
- Project Directory Structure 516
- Defining the Comment, Reply, Photo, and Page Models 517
- Creating the Comments Server 520
- Implementing Routes to Support Viewing and Adding Comments 520
- Implementing the Model-Based Controller Routes 521
- Implementing Photo and Comment Views 527
- Implementing the AngularJS Module and Controller to Support Comment Views 534
- Initializing the Application 540
- Summary 541
- Up Next 541

**28 Creating Your Own Shopping Cart 543**

- Project Description 543
- Libraries Used 544
- Project Directory Structure 544
- Defining the Customer, Product, and Orders Models 546
- Creating the Shopping Cart Server 550
- Implementing Routes to Support Product, Cart, and Order Requests 551
- Implementing the Model-Based Controller Routes 552
- Implementing Shopping Cart and Checkout Views 556
- Implementing the AngularJS Module and Controller to Support Shopping Cart Views 570
- Initializing the Application 579
- Summary 581
- Up Next 581

**29 Building Interactive Web 2.0 Application Components 583**

- Project Description 583
- Libraries Used 584
- Project Directory Structure 584
- Defining the Project Model 586
- Creating the Application Server 586
- Implementing Routes to Support the Views 587
- Implementing a Tabbed View 587
- Implementing a Weather Service View 592
- Implementing Draggable Elements 597
- Implementing Dynamic Data Access 600
- Initializing the Application 605
- Summary 607

**Index 609**

## 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 give me when I am writing and for making sure I still take the time to have fun.

Thanks to Mark Taber for getting this title rolling in the right direction, Russell Kloepfer and Siddhartha Singh for keeping me honest with their technical review, Kitty Wilson for turning the technical ramblings of my brain into a fine text, Tammy Graham and Laura Robbins for styling the graphics, Chuti Prasertsith for the awesome cover, and Elaine Wiley for managing the project and making sure the book is the finest quality.

## About the Author

**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 *jQuery and JavaScript Phrasebook* and *Teach Yourself jQuery and JavaScript in 24 Hours*. He has designed and implemented a wide array of applications and services, from application servers to complex Web 2.0 interfaces. He is also the author of *Python Developer's Phrasebook* and *Teach Yourself Django in 24 Hours*.

## We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author, as well as your name and phone or email address.

Email: [feedback@developers-library.info](mailto:feedback@developers-library.info)

Mail: Reader Feedback  
Addison-Wesley Developer's Library  
800 East 96th Street  
Indianapolis, IN 46240 USA

## Reader Services

Visit our website and register this book at [www.informit.com/register](http://www.informit.com/register) for convenient access to any updates, downloads, or errata that might be available for this book.

*This page intentionally left blank*

# Introduction

Welcome to *Node.js, MongoDB, and AngularJS 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 AngularJS—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 AngularJS 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 will also want to leverage the innovative MVC approach of AngularJS to implement well-designed and structured webpages and web applications. Overall, Node.js, MongoDB, and AngularJS provide an easy-to-implement, fully integrated web development stack that allows you to implement amazing Web 2.0 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 AngularJS 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 AngularJS 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 your 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 AngularJS directives 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 authenticated user accounts to your website/web application
- How to add nested comment components to your webpages
- How to build an end-to-end shopping cart

## What Is Node.js?

Node.js 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 very active development community. People are providing new modules to extend Node.js functionality all the time. Also, it is very 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 very scalable NoSQL database. The name Mongo comes from the word “humongous,” 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 very easy to maintain scalability while keeping high performance.
- **High scalability:** MongoDB's structure makes it easy to scale horizontally by sharding 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 AngularJS?

AngularJS is a client-side framework developed by Google. It is written in JavaScript, with a reduced jQuery library. The theory behind AngularJS is to provide a framework that makes it easy to implement well-designed and structured webpages and applications, using an MVC framework.

AngularJS 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 AngularJS provides:

- **Data binding:** AngularJS has a very clean method for binding data to HTML elements, using its powerful scope mechanism.
- **Extensibility:** The AngularJS architecture allows you to easily extend almost every aspect of the language to provide your own custom implementations.
- **Clean:** AngularJS forces you to write clean, logical code.
- **Reusable code:** The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. 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:** AngularJS is based on JavaScript and has a close relationship with jQuery. This makes it easier to begin integrating AngularJS into your environment and reuse pieces of your existing code within the structure of the AngularJS framework.

## How This Book Is Organized

This book is divided into six main parts:

- Part I, “Introduction,” provides an overview of the interaction between Node.js, MongoDB, and AngularJS 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 AngularJS 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 will 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 AngularJS,” covers the AngularJS framework architecture and how to integrate it into your Node.js stack. This part covers creating custom HTML directives and client-side services that can be leveraged in the browser.
- Part VI, “Building Practical Web Application Components,” switches gears and provides some practical real-world examples of using Node.js, MongoDB, and AngularJS end-to-end to provide authenticated user accounts, comment sections, and shopping carts for web applications. This part also covers some methods to implement Web 2.0 interactions in your web applications.

## 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. To access the source code files and images used in the examples, visit:

<https://github.com/bwdbooks/nodejs-mongodb-angularjs-web-development>

## A Final Word

I hope you enjoy learning about Node.js, MongoDB, and AngularJS as much as I have. They are great, innovative technologies that are really fun to use. Soon, you’ll be able to join the many other web developers who use the Node.js-to-AngularJS web stack to build interactive websites and web applications. I also hope you enjoy this book!

*This page intentionally left blank*

# Introducing the Node.js-to-AngularJS Stack

To get you off on the right foot, this chapter focuses on the fundamental components of the web development framework and then describes the components of the Node.js-to-AngularJS stack that is the basis for the rest of the book. The first section discusses various aspects of the general website/web application development framework, from users to backend services. The purpose of first covering the web development framework components is to help you more easily understand how the components of the Node.js-to-AngularJS stack relate to the pieces of the general framework. This should help you better see the benefits of using the Node.js-to-AngularJS stack components instead of the more traditional technologies.

## Understanding the Basic Web Development Framework

To get you in the right mind-set to understand the benefits of utilizing Node.js, MongoDB, and AngularJS as your web framework, this section provides an overview of the basic components of most websites. If you are already familiar with the full web framework, then this section will be old hat, but if you only understand just the server side or client side of the web framework, then this section will give you a more complete picture.

The main components of any web framework are the user, browser, webserver, and backend services. Although websites vary greatly in terms of appearance and behavior, all have these basic components in one form or another.

This section is not intended to be in-depth, comprehensive, or technically exact but rather a very high-level perspective of the parts involved in a functional website. The components are described in a top-down manner, from user down to backend services. Then the next section discusses the Node.js-to-AngularJS stack from the bottom up, so you can get a picture of where each of the pieces fits and why. Figure 1.1 provides a basic diagram to help you visualize the components in a website/web application, which are discussed in the following sections.

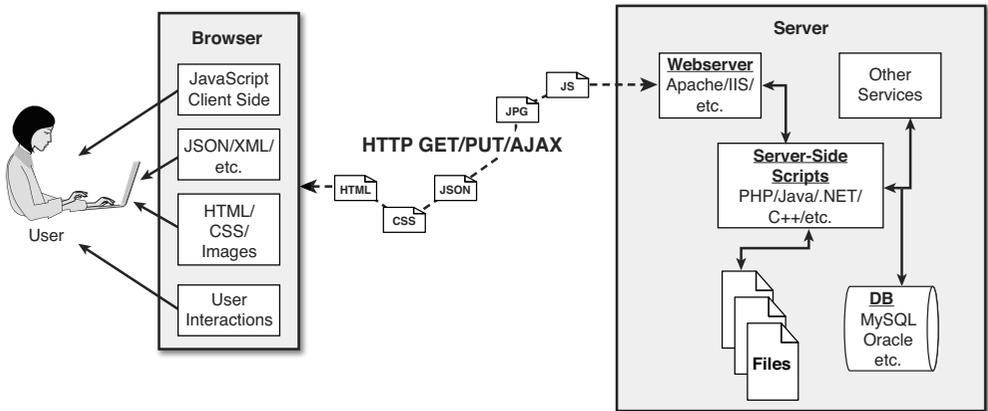


Figure 1.1 Basic diagram of the components of a basic website/web application.

## Users

Users are a fundamental part of every website; they are, after all, the reason websites exist in the first place. User expectations define the requirements for developing a good website. User expectations have changed a lot over the years. In the past, users accepted the slow, cumbersome experience of the “world-wide-wait,” but not today. They expect websites to behave much more quickly, like applications installed on their computers and mobile devices.

The user role in a web framework is to sit on the visual output and interaction input of webpages. That is, users view the results of the web framework processing and then provide interactions using mouse clicks, keyboard input, and swipes and taps.

## The Browser

The browser plays three roles in the web framework:

- Provide communication to and from the webserver
- Interpret the data from the server and render it into the view that the user actually sees
- Handle user interaction through the keyboard, mouse, touchscreen, or other input device and take the appropriate action

### Browser-to-Webserver Communication

Browser-to-webserver communication consists of a series of requests, using the HTTP and HTTPS protocols. Hypertext Transfer Protocol (HTTP) is used to define communication between the browser and the webserver. HTTP defines what types of requests can be made as well as the format of those requests and the HTTP response.

HTTPS adds an additional security layer, SSL/TLS, to ensure secure connections by requiring the webserver to provide a certificate to the browser. The user can then determine whether to accept the certificate before allowing the connection.

There are three main types of requests that a browser will make to a webserver:

- **GET:** The GET request is typically used to retrieve data from the server, such as .html files, images, or JSON data.
- **POST:** POST requests are used when sending data to the server, such as adding an item to a shopping cart or submitting a web form.
- **AJAX:** Asynchronous JavaScript and XML (AJAX) is actually just a GET or POST request that is done directly by JavaScript running in the browser. Despite the name, an AJAX request can receive XML, JSON, or raw data in the response.

## Rendering the Browser View

The screen that the user actually views and interacts with is often made up of several different pieces of data retrieved from the webserver. The browser reads data from the initial URL and then renders the HTML document to build a Document Object Model (DOM). The DOM is a tree structure object with the HTML document as the root. The structure of the tree basically matches the structure of the HTML document. For example, `document` will have `html` as a child, and `html` will have `head` and `body` as children, and `body` may have `div`, `p`, or other elements as children, like this:

```
document
+ html
  + head
  + body
    + div
      + p
```

The browser interprets each DOM element and renders it to the user's screen to build the webpage view.

The browser often gets various types of data from multiple webserver requests to build a webpage. The following are the most common types of data the browser uses to render the final user view as well as define the webpage behavior:

- **HTML files:** These provide the fundamental structure of the DOM.
- **CSS files:** These define how each of the elements on the page is to be styled, in terms of font, color, borders, and spacing.
- **Client-side scripts:** These are typically JavaScript files. They can provide added functionality to a webpage, manipulate the DOM to change the look of the webpage, and provide any necessary logic required to display the page and provide functionality.
- **Media files:** Image, video, and sound files are rendered as part of the webpage.

- **Data:** Data such as XML, JSON, or raw text can be provided by the webserver as a response to an AJAX request. Rather than send a request back to the server to rebuild the webpage, new data can be retrieved via AJAX and inserted into the webpage via JavaScript.
- **HTTP headers:** HTTP defines a set of headers that the browser can use and client-side scripts to define the behavior of the webpage. For example, cookies are contained in the HTTP headers. The HTTP headers also define the type of data in the request as well as the type of data expected to be returned to the browser.

## User Interaction

The user interacts with the browser via mice, keyboards, and touchscreens. A browser has an elaborate event system that captures user input events and then takes the appropriate actions. Actions vary from displaying a popup menu to loading a new document from the server to executing client-side JavaScript.

## Webservers

A webserver's main focus is handling requests from browsers. As described earlier, a browser may request a document, post data, or perform an AJAX request to get data. The webserver uses HTTP headers as well as a URL to determine what action to take. This is where things get very different, depending on the webserver, configuration, and technologies used.

Most out-of-the-box webservers such as Apache and IIS are made to serve static files such as .html, .css, and media files. To handle POST requests that modify server data and AJAX requests to interact with backend services, webservers need to be extended with server-side scripts.

A *server-side script* is really anything that a webserver can execute in order to perform the task the browser is requesting. These scripts can be written in PHP, Python, C, C++, C#, Perl, Java, ... the list goes on and on. Webservers such as Apache and IIS provide mechanisms to include server-side scripts and then wire them up to specific URL locations requested by the browser. This is where having a solid webserver framework can make a big difference. It often takes quite a bit of configuration to enable various scripting languages and wire up the server-side scripts so that the webserver can route the appropriate requests to the appropriate scripts.

Server-side scripts either generate a response directly by executing their code or connect with other backend servers such as databases to obtain the necessary information and then use that information to build and send the appropriate responses.

## Backend Services

Backend services are services that run behind a webserver and provide data that is used to build responses to the browser. The most common type of backend service is a database that stores information. When a request comes in from the browser that requires information from the database or other backend service, the server-side script connects to the database, retrieves

the information, formats it, and then sends it back to the browser. On the other hand, when data comes in from a web request that needs to be stored in the database, the server-side script connects to the database and updates the data.

## Understanding the Node.js-to-AngularJS Stack Components

With the basic structure of the web framework fresh in your mind, it is time to discuss the Node.js-to-AngularJS stack. The most common—and I believe the best—version of this stack is the Node.js-to-AngularJS stack comprised of MongoDB, Express, AngularJS, and Node.js.

In the Node.js-to-AngularJS stack, Node.js provides the fundamental platform for development. The backend services and server-side scripts are all written in Node.js. MongoDB provides the data store for the website but is accessed via a MongoDB driver Node.js module. The webserver is defined by Express, which is also a Node.js module.

The view in the browser is defined and controlled using the AngularJS framework. AngularJS is an MVC framework in which the model is made up of JSON or JavaScript objects, the view is HTML/CSS, and the controller is AngularJS JavaScript code.

Figure 1.2 provides a very basic diagram of how the Node.js to AngularJS stack fits into the basic website/web application model. The following sections describe each of these technologies and why they were chosen as part of the Node.js to AngularJS stack. Later chapters in the book will cover each of the technologies in much more detail.

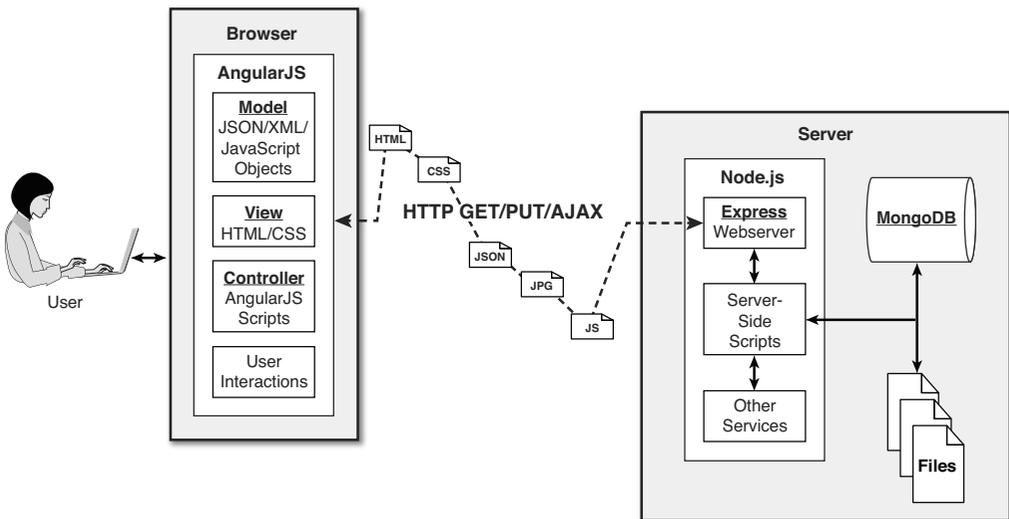


Figure 1.2 Basic diagram showing where Node.js, Express, MongoDB, and AngularJS fit in the web paradigm.

## Node.js

Node.js is a development framework that is based on Google's V8 JavaScript engine and executes it.

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. Also, the webserver can run directly on the Node.js platform as a Node.js module, which means it's much easier than using, say, Apache for wiring up new services or server-side 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 web servers to scale in ways that traditional web servers can't.
- **Extensibility:** Node.js has a great following and very active development community. People are providing new modules to extend Node.js functionality all the time. Also, it is very 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.

## MongoDB

MongoDB is an agile and very scalable NoSQL database. The name Mongo comes from the word “humongous,” emphasizing the scalability and performance MongoDB provides. It is based on the NoSQL document store model, which means data is stored in the database as basically JSON objects rather than as the traditional columns and rows of a relational database.

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. This book covers using the MongoDB driver library to access MongoDB from Node.js.

Node.js supports a variety of database access drivers, so the data store can easily be MySQL or some other database. However, the following are some of the reasons that MongoDB really fits in the Node.js stack well:

- **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 very easy to maintain scalability while keeping high performance.
- **High scalability:** MongoDB's structure makes it easy to scale horizontally by sharding the data across multiple servers.

## Express

The Express module acts as the webserver in the Node.js-to-AngularJS stack. Because it runs in Node.js, it is easy to configure, implement, and control. The Express module extends Node.js to provide several key components for handling web requests. It allows you to implement a running webserver in Node.js with only a few lines of code.

For example, the Express module provides the ability to easily set up destination routes (URLs) for users to connect to. It also provides great functionality in terms of working with HTTP request and response objects, including things like cookies and HTTP headers.

The following is a partial list of the valuable features of Express:

- **Route management:** Express makes it easy to define routes (URL endpoints) that tie directly to the Node.js script functionality on the server.
- **Error handling:** Express provides built-in error handling for “document not found” and other errors.
- **Easy integration:** An Express server can easily be implemented behind an existing reverse proxy system, such as Nginx or Varnish. This allows you to easily integrate it into your existing secured system.
- **Cookies:** Express provides easy cookie management.
- **Session and cache management:** Express also enables session management and cache management.

## AngularJS

AngularJS is a client-side framework developed by Google. It provides all the functionality needed to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. It is written in JavaScript, with a reduced jQuery library. The theory behind AngularJS is to provide a framework that makes it easy to implement web applications using the MVC framework.

Other JavaScript frameworks could be used with the Node.js platform, such as Backbone, Ember, and Meteor. However, AngularJS has the best design, feature set, and trajectory at this writing. Here are some of the benefits AngularJS provides:

- **Data binding:** AngularJS has a very clean method for binding data to HTML elements, using its powerful scope mechanism.
- **Extensibility:** The AngularJS architecture allows you to easily extend almost every aspect of the language to provide your own custom implementations.
- **Clean:** AngularJS forces you to write clean, logical code.
- **Reusable code:** The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. 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:** AngularJS is based on JavaScript and has a close relationship with jQuery. This makes it easier to begin integrating AngularJS into your environment and reuse pieces of your existing code within the structure of the AngularJS framework.

## Summary

This chapter covers the basics of the web development framework to give you a good foundation for the rest of the book. This chapter covers the basics of interaction between the webserver and browser, as well as the functionality required to make modern websites function.

This chapter also describes the Node.js-to-AngularJS stack, comprising Node.js, MongoDB, Express, and AngularJS. Node.js provides the platform for the framework, MongoDB provides the backend data store, Express provides the webserver, and AngularJS provides the client-side framework for modern web applications.

## Up Next

The next chapter provides a brief primer on the JavaScript language. Because the Node.js-to-AngularJS stack is based on JavaScript, you need to be familiar with the language to be able to follow the examples in the rest of the book.

# Index

## Symbols

---

**\$ update operator, 253**

## A

---

**a directive (AngularJS templates), 450**

**abort() method, 165**

**acceptsCharset() method, 365**

**access control, MongoDB, configuring,  
216-218**

**user accounts**

**MongoDB**

**administering, 212-215**

**assigning database roles to, 214**

**configuring access control, 216-218**

**creating, 213-215**

**listing, 212-213**

**removing, 215**

**websites, adding to, 491-514**

**\$add aggregation expression operator, 292**

**\$addToSet aggregation expression  
operator, 292**

**\$addToSet update operator, 253**

**address() method, 146, 149**

**address schema, shopping cart, defining,  
546-547**

**addToCart() function, 572**

**addUser() method, 213, 235, 237**

**admin() method, 235**

**Admin object (MongoDB driver), 237**

**administration**

- MongoDB databases, 218-220

- MongoDB user accounts, 212-215

**agent option (ClientRequest object), 123**

**aggregate() method, 289-290, 306, 325**

**aggregation**

- documents, Mongoose, 325

- MongoDB query results, 289-293

**aggregation operators, 290**

**all() method, 309**

**allowHalfOpen option**

- net.Server objects, 148

- Socket object, 145

**and() method, 309**

**AngularJS, 4, 14, 397-398, 409**

- animations, 484

- implementing in CSS, 482-484

**applications**

- code, 538-540

- creating basic, 405-409

- defining root element, 407

- initializing, 540-541

- benefits, 4, 397-398

- browser alerts, implementing, 478

- browser cookies, interacting with, 478-479

- controllers, 400

- implementing, 507-508

- data binding, 14, 400

- data models, 399

- dependency injection, 400, 411-412, 418

- implementing, 416-417

- directives, 399, 469

- adding controllers to, 466-467

- animation, 481

- binding model to page elements, 453-454

- binding page elements to controllers, 453-459

- configuring scope, 464-465

- creating custom, 467

- creating to extend HTML, 461-467

- form element extensions, 450-452

- functionality support, 448

- implementing custom, 590-591

- restricting behavior, 462-463

- expressions, 399

**filters**

- custom, 444-445

- orderBy, 442-443

- global APIs, 404

- HTML documents, bootstrapping in, 189

- HTTP Get and Put requests, sending, 473-475

- JavaScript, integration, 402-403

- jQuery, integration, 402-403

- life cycle, 401-402

- bootstrap phase, 401

- compilation phase, 402

- runtime data binding phase, 188

- link() function, manipulating DOM, 465-466

- loading library, 407

- modules, 398-399, 411-412, 418

- adding providers to, 414-416

- configuration blocks, 413-414

- creating Module object, 413

- defining, 412-416

- implementing, 507-508

- implementing to support comment views, 534-540

- run blocks, 414

- using HTML code to implement, 417

- Node.js, adding to environment, 403
- page controllers, implementing, 537-538
- scope model, 407-409
- scopes, 399, 419, 431-432
  - emitting and broadcasting events, 428-429
  - expressions, 436
  - filters, 440
  - implementing hierarchy, 425-426
  - life cycle, 424-425
  - relationship between backend server data, 423-424
  - relationship between controllers, 420-421
  - relationship between root scope and applications, 419-420
  - relationship between templates, 421-422
  - sorting and ordering, 442
- separation of responsibilities, 401
- services, 400, 471, 489
  - \$animate, 481-484
  - \$cacheFactory, 477-478
  - \$cookieStore, 478-479
  - \$http, 473-475
  - \$interval, 481
  - \$timeout, 481
  - \$window, 478
  - built-in, 472-484
  - creating constant services, 487
  - creating custom, 487-489
  - creating factory services, 487-488
  - creating service services, 488
  - creating value services, 487
  - integrating in modules, 488-489
- shopping cart, view support, 570-579
- table controller, defining, 601-603
- tables view, 603
- templates, 399, 433-434, 445
  - adding controllers to, 407
  - built-in directives, 447-459
  - creating custom filters, 443-444
  - defining directive view, 462
  - directives, 433, 447
  - expressions, 433-435
  - filters, 433, 437-441
  - modifying data displayed in rendered view, 440
  - obtaining data from scope model, 436
  - replacing elements, 463
  - tabbed view, 588-590
  - transcluding external scopes, 463-464
- timers, implementing, 481
- views, 447
  - creating with templates, 433-445
- weather controller, defining, 594
- weather view
  - defining, 595
  - styling, 596-595
- angular.module() method, 413**
- \$animate built-in service (AngularJS), 472, 481-484**
- animations**
  - elements, AngularJS, 484
  - implementing in CSS, AngularJS, 481
- anonymous functions, 26**
- APIs, global, AngularJS, 404**
- append() method, 323**
- application servers, creating, 586-587**
- applications**
  - creating basic, AngularJS, 405-409
  - initializing, 540-541

- Node.js, creating, 47-52
- RIAs (rich Internet applications), 583
- app.render() method, 378**
- arch() method, 185**
- arch property (process module), 165**
- argv property (process module), 165**
- arithmetic operators, JavaScript, 17-18**
- arrays, 32-35**
  - combining, 34
  - items
    - adding/removing, 35
    - checking for, 35
  - iterating through, 34-35
  - JavaScript, 16
  - manipulating, 33-34
  - strings
    - converting into, 35
    - splitting into, 32
- assignment operators, JavaScript, 18**
- asynchronous callback, creating a wrapper function, 70**
- asynchronous calls, file system, 97-98**
- asynchronous file reads, 108**
- asynchronous file writing, 102-103**
- atomic write operations, MongoDB, 202**
- atomically modifying documents in collections, 260-261**
- attachment() method, 367**
- auth option (ClientRequest object), 123**
- authenticate() method, 235, 237**
- authentication**
  - authenticate() method, 235, 237
  - Express sessions, 392-393
  - HTTP, applying, 390-391
  - MongoDB, 217
  - Passport NPM, 508-512
  - views, implementing, 501-507
- author directive (package.json file), 46**

- auto\_reconnect option (Server object), 229**
- autoIndex option (Schema object), 300**
- autoIndexID role (MongoDB), 221**

---

## B

- backend server data, AngularJS scopes, relation between, 423-424**
- backend services, 10-11**
- backend weather service, Node.js server, creating on, 592-593**
- backing up MongoDB, 354-355**
- base64() method, 75**
- basic-auth-connect middleware (Express), 390-391**
- billing, shopping cart, verifying, 575**
- billing schema, shopping cart, defining, 547**
- billing view, shopping cart, implementing, 563**
- bin directive (package.json file), 46**
- \$bit update operator, 253**
- blocking Node.js I/O, 57**
- blocks, try/catch, 36**
- Booleans, JavaScript, 16**
- bootstrap phase, AngularJS life cycle, 401**
- bootstrapping AngularJS in HTML documents, 189**
- broadcasting events, AngularJS scopes, 428-429**
- browsers**
  - alerts, implementing, 478
  - browser-to-webserver communication, 8-9
  - cookies, interacting with, 478-479
  - data, 9-10
  - rendering view, 9-10
  - user interaction, 10
  - web framework, 8-10

**bufferCommands** option (Schema object), 300

**buffering** data, Buffer module, 74-82

**bufferMaxEntries** option (MongoClient connections), 230

**buffers**

- concatenating, 82
- copying data from, 79-80
- creating, 76
- determining length, 79
- reading from, 77-78
- slicing, 81
- writing to, 76-77

**bufferSize** property (Socket objects), 147

**built-in directives**, AngularJS templates, 447-459

**built-in filters**, AngularJS templates, 438

**built-in services**, AngularJS, 472-484

**bytesRead** property (Socket objects), 147

**bytesWritten** property (Socket objects), 147

---

## C

**ca** option

- https.request method, 140
- tls.connect() method, 158
- tls.createServer() method, 159

**cache** property (config parameter), 473

**\$cacheFactory** built-in service (AngularJS), 472, 477-478

**callback** parameter, group() method, 287

**callbacks**

- asynchronous, creating a wrapper function, 70
- chaining, 70-71, 113
- closure, implementing in, 69
- events, 56
- implementing, 68-71
- passing additional parameters to, 68

**calls**, file system, 97-98

**capped collections** (MongoDB), 202, 336

**capped** option (Schema object), 300

**capped** role (MongoDB), 221

**cart**. See shopping cart

**cart** view, shopping cart, implementing, 560. See also shopping cart

**case sensitive routing** setting (Express), 357

**cert** option

- https.request method, 140
- tls.connect() method, 158
- tls.createServer() method, 159

**chaining** callbacks, 70-71, 113

**chdir()** method, 165

**checkKeys** option (database changes), 251

**checkout()** function, 574

**child** forks, implementing, 175-176

**child\_process** module, 168-176

**ChildProcess** object, 168-169

**chunkCollection()** method, 350

**ciphers** option (https.request method), 140

**ciphers** option (tls.createServer() method), 159

**Client** object, MongoDB, connecting to via, 230-231

**clientError** event (TLS Server objects), 159

**clients**

- HTTP, requests to test servers, 182
- HTTPS (Hypertext Transfer Protocol Secure), implementing, 139-140
- TCP socket, implementing, 151-152
- TLS (Transport Layer Security), implementing, 157-158

**client-side** scripts, 9

**close** event

- ChildProcess object, 168
- net.Socket object, 149
- Socket objects, 145

**close() method, 149, 235, 240, 350**

**closing files, 98-99**

**closure, callbacks, implementing in, 69**

**cluster module, 178-179**

**clusterAdmin database role (MongoDB), 214**

**clusters**

HTTP, implementing, 181

process, implementing, 178-181

sharded MongoDB, 344-347

**collection() method, 235, 350**

**Collection object, MongoDB driver, 238**

**collection option (Schema object), 300**

**collectionInfo() method, 235**

**collectionNames() method, 235**

**collections (MongoDB), 196**

aggregating query results, 289-293

capped, 202, 336

creating, 221, 246-247

deleting, 222, 247

displaying in databases, 220-221

documents

adding to, 223, 254-255

atomically modifying in, 260-261

counting specific sets of in  
collections, 277-278

deleting from, 224, 265

finding, 222, 257, 274-275

finding distinct field values in a  
specific set of, 286

grouping sets of documents by  
specific fields in, 288, 294

inserting in, 255, 263

limiting specific sets of, 279

obtaining from, 256-257

paging results, 282

removing single from, 266-267

saving in, 262

updating in, 224-225, 258-259

field values, finding, 285

fields

forcing required, 302

implementing unique, 302

indexing, 203

large, 203-204

listing, 246-247

managing, 220-225

obtaining information, 248

replications, 203

sharding, 203, 340-347

enabling on, 347

shared tag ranges, setting up, 347

statistics, retrieving and displaying, 248

word\_stats, implementing schema, 303

**collections() method, 235**

**command parameter, group() method, 287**

**command services, creating, 534-535**

**commands**

mongod, 345-346

spawning in another process, 174-175

**comment() method, 307**

**comment option (options object), 272**

**comment thread view, implementing, 530-531**

**comment threads, webpages, adding to, 515-541**

**comment view, implementing, 532**

**comments**

adding to servers, 535

comment application server routes for  
Express, 521

partial comment template, 531-532

photo comments portion of view,  
537-538

- retrieving from servers, 535
- server, creating, 520
- webpage support, 538-540
- CommentThread model (Mongoose)**
  - defining, 518-519
  - implementing controller, 523-526
- comparison operators, JavaScript, 19-21**
- \$compile built-in service (AngularJS), 472**
- compilation phase, AngularJS life cycle, 402**
- compiling Mongoose models, 304**
- compound index (MongoDB), 334**
- compression, Zlib module, 93-95**
- \$concat aggregation expression operator, 292**
- concatenating buffers, 82**
- conditional operators, JavaScript, 20**
- config property (process module), 165**
- config server database instances, creating, 345**
- configuration**
  - Express, 357
  - routes, 359-363
  - MongoDB access control, 216-218
- configuration blocks, AngularJS modules, 413-414**
- connect event (Socket objects), 145**
- connect option (ClientRequest object), 124**
- connected property (ChildProcess object), 169**
- connection event (net.Socket object), 149**
- connection strings, MongoDB, connecting to, 232-234**
- console module (Node.js), writing data to, 52**
- constant() method, 415**
- constant services (AngularJS), creating, 487**
- continue option (ClientRequest object), 124**
- contributors directive (package.json file), 46**
- controller property (AngularJS templates), 461**
- controllers**
  - AngularJS, 400
  - adding to templates, 407
  - binding page elements to, 453-459
  - directives, adding to, 466-467
  - implementing, 507-508
  - scopes, relationship between, 420-421
  - page, implementing, 537-538
  - photo, implementing, 535-537
  - shopping cart
    - customer model, 554-555
    - order model, 553-554
    - product model, 552
  - weather, defining, 594
- cookie-parser middleware (Express), 387-388**
- cookies**
  - Express, sending and receiving, 387-388
  - interacting with, AngularJS, 478-479
- \$cookies built-in service (AngularJS), 472**
- cookie-session middleware (Express), 388-389**
- \$cookieStore service (AngularJS), 478-479**
- copy() method, 404**
- copying**
  - buffers, 79-80
  - MongoDB databases, 220
- count() method, 238, 240, 306**
- counting MongoDB documents, 276-277**
- cpus() method, 185**
- create() method, 306**
- createCollection() method, 235, 336**
- creation phase (AngularJS scope), 424**

**curl option**`https.createServer()` method, 140`tls.createServer()` method, 159**CSS (cascading style sheets)**

adding to style views, 507

files, 9

views, adding to, 533-534

**currency[:symbol] filter (AngularJS templates), 438****Cursor object, MongoDB driver, 240****custom defined objects, creating, 28-29****custom directives, AngularJS, 590-591**

creating, 467

**custom Express middleware, creating, 395****custom filters, AngularJS, creating, 444-445****custom services, AngularJS, creating, 487-489****customer model controllers, shopping cart, implementing, 554-555****customer schema, shopping cart, defining, 549-550****`cwd()` method, 165****cwd option**`exec()` function, 170`fork()` function, 175`spawn()` function, 173

---

**D****Dahl, Ryan, 39-40****data**

buffering, Buffer module, 74-82

streaming, 83-92

**data binding, AngularJS, 14, 400****data event (Socket objects), 145****data life cycles, MongoDB, 204****data models**

AngularJS, 399

MongoDB, 199-204

**data normalization, MongoDB, 199-200****data property (config parameter), 473****data sets, MongoDB, 269-270****data types**

JavaScript, 16-17

MongoDB, 198

**data usability, MongoDB, 204****databases (MongoDB). See also collections**

administering, 212-215, 218-220

applying replication, 337-340

building environment, 207-212

change options, 251

configuring access control, 216-218

connecting to via Client object, 231-232

copying, 220

creating, 242-243

deleting, 219, 242-243

displaying collections in, 220-221

enabling sharding on, 346

listing, 242-243

repairing, 353-354

update operators, 252-253

**date[:format] filter (AngularJS templates), 438****`db()` method, 235****Db object (MongoDB driver), 235****dbAdmin database role (MongoDB), 214****dbAdminAnyDatabase database role (MongoDB), 214****decompression, Zlib module, 93-95****defining**

address schema, shopping cart, 546-547

AngularJS modules, 412-416

- billing schema, shopping cart, 547
- CommentThread model, 518-519
- customer schema, shopping cart, 549-550
- Express template engine, 375
- functions, 24-25
- order schema, shopping cart, 548-549
- Page model, 517
- Photo model, 518
- product schema, shopping cart, 547-548
- quantity schema, shopping cart, 548
- table controller, 601-603
- user model, 492-493
- variables, 15-16
- weather controller, 594
- weather view, 595
- deleteFromCart() function, 573-574**
- deleteUser route, implementing, 499**
- deleting files, 114**
- denormalizing data, MongoDB, 200-201**
- dependencies directive (package.json file), 46**
- dependency injection, AngularJS, 400, 411-412, 418**
  - implementing, 416-417
- deployment, sharded MongoDB clusters, 344-347**
- dereferencing timers, 63**
- description directive (package.json file), 46**
- destroy() method, 146, 392**
- detached option, spawn() function, 173**
- directive() method, 461-467**
- directives (AngularJS), 399, 469**
  - adding controllers to, 466-467
  - configuring scope, 464-465
  - creating custom, 467
  - implementing custom, 590-591
  - restricting behavior, 462-463
  - templates, 433, 447
    - binding model to page elements, 453-454
    - binding page elements to controllers, 453-459
    - built-in, 447-459
    - creating to extend HTML, 461-467
    - form element extensions, 450-452
    - functionality support, 448
- directories**
  - creating and removing, 115-116
  - renaming, 116
- disconnect event**
  - ChildProcess object, 168
  - cluster module, 178
  - Worker object, 180
- disconnect() method, 169, 179-180, 299**
- distinct() method, 238, 285, 306**
- \$divide aggregation expression operator, 292**
- dns module, 191-192**
- Document object, 298, 310**
- \$document built-in service (AngularJS), 472**
- documents**
  - HTML, bootstrapping AngularJS in, 189
  - MongoDB, 196-198. *See also* collections (MongoDB)
    - adding to collections, 223, 254-255
    - adding via Mongoose, 314
    - aggregating via Mongoose, 323-325
    - atomically modifying in collections, 260-261
    - counting, 276-277
    - counting specific sets of in collections, 277-278
    - deleting from collections, 224, 265

- document references, 199-200
- embedding, 200-201
- finding distinct field values in a specific set of, 286
- finding in collections, 222
- finding specific sets of, 273-274
- finding via Mongoose, 312
- grouping sets of by specific fields in collections, 288, 294
- growth, 203
- inserting in collections, 263
- limiting fields returned by, 280
- limiting result sets, 278-282
- obtaining from collections, 256-257
- querying, 270-272
- removing single from collection, 266-267
- removing via Mongoose, 320-322
- saving in collections, 262
- updating in collections, 224-225, 258-259
- updating via Mongoose, 316-319
- domains, performing lookups and reverse lookups, 193
- do/while loops, 22
- download responses, sending, 373
- draggable views, 607
  - implementing, 597-599
- drain event (Socket objects), 145
- drop() method, 238
- dropCollection() method, 235
- dropDatabase() method, 235
- Duplex streams, 88-89
- dynamic data access, 600-603
- dynamic GET servers, implementing, 132-133

---

## E

- each() method, 240
- \$each update operator, 253
- element() method, 404
- elemMatch() method, 309
- emitters, event, 65-66
- emitting events, AngularJS scopes, 428-429
- encoding option
  - exec() function, 170
  - fork() function, 175
- end event (Socket objects), 145
- end() method, 146
- endian, 75
- endianness() method, 185
- engines directive (package.json file), 46
- env option
  - exec() function, 170
  - fork() function, 175
  - spawn() function, 173
- env property (process module), 165
- env setting (Express), 357
- environments, MongoDB, building, 207-212
- eof() method, 350
- equals() method, 310, 404
- error event
  - ChildProcess object, 168
  - net.Socket object, 149
  - Socket objects, 145
  - Worker object, 180
- error handling, JavaScript, adding, 36-38
- errors method, 310
- event queue, Node.js, adding work to, 59-66
- EventEmitter object, creating custom, 66-67

**events**

AngularJS scopes, emitting and broadcasting, 428-429

callbacks, 56

ChildProcess object, 168

cluster module, 178

creating anonymous function for additional parameters, 68-69

emitters, 65-66

EventEmitter object, creating custom, 66-67

listeners, 65-66

Node.js

model, 55-59

processes, 164

Socket objects, 145

net.Socket, 149

Worker object, 180

**exec() function, processes, executing system commands on, 170**

**exec() method, 323**

**execArgv property (process module), 165**

**execFile() function, 170**

processes, executing executable files on another, 171-172

**execPath option (fork() function), 175**

**execPath property (process module), 165**

**executable files, executing on another process, 171-172**

**executables, Node.js, verifying, 41**

**exists() method, 309**

**exit event**

ChildProcess object, 168

cluster module, 178

Worker object, 180

**exit() method, 165**

**explain option (options object), 272**

**Express, 357, 379**

authentication, 390-391

HTTP, 391

session, 393-394

configuring, 357

cookies, sending and receiving, 387-388

HTTP authentication, applying, 390-391

middleware, 381-382, 396

adding multiple functions, 383

assigning globally to paths, 382-383

assigning to a single route, 383

basic-auth-connect, 390-391

cookie-parser, 387-388

cookie-session, 388-389

creating custom, 395

POST parameters, 386

query, 383-384

session, 392-393

static, 384

Request object, 365

accessing properties, 365

redirecting, 374

Response object, 366-374

redirecting responses, 373-374

sending download responses, 373

sending files, 371-372

sending JSON responses, 369-370

sending responses, 368

setting headers, 366-367

setting status, 367-368

route controller, creating for/words route, 600-601

routes

applying parameters in, 361-363

configuring, 359-363

## servers

- creating, 493-494
- implementing HTTP and HTTPS, 359
- implementing routes, 494-496
- implementing user controller routes, 496-501
- starting, 358-359

## sessions

- authentication, 392-393
- implementing, 388-389

## template engines

- adding locals, 375-376
- defining, 375
- implementing, 374-378

## templates

- creating, 376-377
- rendering in responses, 378
- webservers, implementing application database connection, 494

**Express module, Node.js-to-AngularJS stack, 13****expressions, AngularJS, 399**

- templates, 433-435

**extend() method, 404**


---

**F**
**factory() method, 415****factory services (AngularJS), creating, 487-488****field values, MongoDB, finding, 285****fields, collections (MongoDB)**

- forcing required, 302
- implementing unique, 302

**fields option (options object), 272****file system**

- calls, 97-98
- directories, creating and removing, 115-116

**files**

- allowing streaming writes to, 104
- asynchronous reads, performing, 108
- asynchronous writes, performing to, 103
- closing, 98-99
- CSS (cascading style sheets), 9
- deleting, 114
- executable, executing on another process, 171-172
- HTML, 9
- information, obtaining, 111-112
- JSON (JavaScript Object Notation) strings, writing to, 100
- listing, 113
- media, 9
- monitoring changes, 117
- opening, 98-99
- renaming, 116
- sending, Response object, 371-372
- serving static, Express, 384
- static, serving, 130-131
- streaming reads, 110
- synchronous reads, performing, 107
- synchronous writes, performing to, 101-102
- truncating, 114-115
- verifying path existence, 110-111
- writing, 100-104

**filter:exp:compare filter (AngularJS templates), 438****filters, AngularJS**

- custom, 444-445
- orderBy, 442-443
- scopes, 440
- templates, 433, 437-441
  - creating custom filters, 443-444
  - implementing ordering and filtering, 441

**finalize parameter, group() method, 287**  
**finally keyword, error handling, 36**  
**find() method, 238, 256, 278-279, 306**  
**findAndModify() method, 238, 260-261**  
**findAndRemove() method, 238, 267**  
**findOne() method, 238, 256, 306**  
**findOneAndRemove() method, 306**  
**findOneAndUpdate() method, 306**  
**\$first aggregation expression operator, 292**  
**for loops, 22-23**  
**forceServerObjectId option (database changes), 251**  
**forEach() method, 404**  
**for/in loops, 23**  
**fork event (cluster module), 178**  
**fork() function, 175-176**  
**form parameters, processing, 121-122**  
**formatting strings, 187-188**  
**form/ngform directive (AngularJS templates), 450**  
**forms, element extension, AngularJS directives, 450-452**  
**freemen() method, 185**  
**fresh property (Request object), 365**  
**fromJson() method, 404**  
**fs module, 97-98**  
**fs.stats() method, 112**  
**fsync option**

- database changes, 251
- MongoClient connections, 230

**functionality**

- AngularJS templates, directives, 448
- objects, inheriting from other, 190

**functions, 24. See also methods**

- addToCart(), 572
- anonymous, 26
- checkout(), 574

**console module (Node.js), 52**  
**defining, 24-25**  
**deleteFromCart(), 573-574**  
**exec(), 170**  
**execFile(), 170-172**  
**Express middleware, adding, 383**  
**findAndModify(), 260-261**  
**fork(), 175-176**  
**getSortObj(), 600-601**  
**hashPW(), 496**  
**link(), 465-466**  
**makePurchase(), 575-576**  
**passing variables to, 25**  
**returning values from, 25-26**  
**setShipping(), 574-575**  
**spawn(), 173-174**  
**verifyBilling(), 575**

---

## G

**geospatial index (MongoDB), 334**  
**get() method, 310, 349, 365, 367**  
**Get requests, sending, AngularJS, 473-475**  
**GET servers, implementing dynamic, 132-133**  
**getc() method, 350**  
**getConnections() method, 149**  
**getgid() method, 165**  
**getSortObj() function, 600-601**  
**getuid() method, 165**  
**getUserProfile routes, implementing, 498**  
**gid option, spawn() function, 173**  
**global APIs, AngularJS, 404**  
**global variables, defining, 27**  
**globally assigning Express middleware to paths, 382-383**  
**Google, authentication, Passport NPM, 511-512**

**Grid object, 349**

**GridFS (MongoDB)**

Grid object, implementing from  
Node.js, 348-349

GridStore object, implementing from  
Node.js, 350-352

implementing, 348-349  
from Node.js, 350-352

**group() method, 286-288, 323**

**\$group operator, aggregation() method,  
290**

**gt() method, 309**

**gte() method, 309**

---

## H

**handshakeTimeout option,  
tls.createServer() method, 159**

**hash-based sharding, MongoDB, 344**

**hashed index (MongoDB), 334**

**hashPW() function, 496**

**headers**

HTTP, 10

Response object, setting, 366-367

**headers option (ClientRequest object), 123**

**headers property**

config parameter, 473

Request object, 365

**helper functions, shopping cart, 572**

**Hex() method, 75**

**hierarchy, AngularJS scope, implementing,  
425-426**

**hint() method, 307**

**hint option (options object), 272**

**honorCipherOrder option, tls.createServer()  
method, 159**

**host option**

ClientRequest object, 123

Socket object, 145

**host property (Request object), 365**

**hostname() method, 185**

**hostname option (ClientRequest object),  
123**

**hrtime() method, 165**

**HTML documents, 9**

AngularJS, bootstrapping in, 189

**\$http built-in service (AngularJS), 472-475**

**HTTP (HyperText Transfer Protocol)**

authentication, applying, Express,  
390-391

clusters, implementing, 181

headers, 10

service URLs (uniform resource  
locators), processing, 119-121

**http.ClientRequest object, 122-124**

**http.IncomingMessage object, 126**

**httpOnly property (options parameter), 387**

**HTTPS (Hypertext Transfer Protocol Secure)**

clients, implementing, 139-140

servers, implementing, 139-140, 142

**https.createServer() method, 140**

**http.ServerResponse object, 125**

**https.request() method, 140**

---

## I

**\_id index (MongoDB), 334**

**\_id option (Schema object), 300**

**id() method, 310**

**id option (Schema object), 300**

**id property (Worker object), 180**

**if statements, 20**

**immediate timers, 62-63**

**in() method, 309**

**\$inc update operator, 253**

**indexes**

- MongoDB, adding, 333-336
- schemas, adding to, 301-302
- view, implementing, 503

**indexing, MongoDB collections, 203****inheritance, functionality, from other objects, 190****initgroups() method, 165****initial parameter, group() method, 287****initializing shopping cart, 579-581****input directive (AngularJS templates), 450****input.checkbox directive (AngularJS templates), 450****input.email directive (AngularJS templates), 450****input.number directive (AngularJS templates), 450****input.radio directive (AngularJS templates), 450****input.text directive (AngularJS templates), 450****input.url directive (AngularJS templates), 450****insert() method, 238, 255****installation**

- MongoDB, 207-208
- Node.js, 40-41
- NPMs (Node Packaged Modules), 44-45

**\$interval built-in service (AngularJS), 472, 481****interval timers, 61****invalidate() method, 310****I/O (input/output), Node.js, blocking, 57****IP addresses, performing lookups and reverse lookups, 193****ip property (Request object), 365****IPC (interprocess communication), 143****isArray() method, 404****isClosed() method, 240****isDate() method, 404****isDefined() method, 404****isElement() method, 404****isFunction() method, 404****isInit() method, 310****isMaster property (cluster module), 179****isModified() method, 310****isNew() method, 310****isNumber() method, 404****isObject() method, 404****isSelected() method, 310****isString() method, 404****isUndefined() method, 404****isWorker property (cluster module), 179****iterating through arrays, 34-35**

---

**J**

---

**JavaScript, 15, 38****AngularJS, integration, 402-403****arrays, 16, 32-35**

- adding/removing items, 35
- checking item contents, 35
- combining, 34
- converting into strings, 35
- iterating through, 34-35
- manipulating, 33-34

**Booleans, 16****data types, 16-17****error handling, adding, 36-38****functions, 24**

- anonymous, 26
- defining, 24-25
- passing variables to, 25
- returning values from, 25-26

**if statements, 20**

JSON (JavaScript Object Notation),  
 converting to objects, 74

loops, 21

- for/in, 23
- do/while, 22
- for, 22-23
- interrupting, 23-24
- while, 21-22

null, 17

numbers, 16

object literal, 17

objects, 27

- converting to JSON, 74
- creating custom defined, 28-29
- prototyping patterns, 29-30
- syntax, 27-28

operators, 17-21

- arithmetic, 17-18
- assignment, 18
- comparison, 19-21
- conditional, 20

strings, 16

- combining, 31-32
- manipulating, 30-32
- replacing a word in, 32
- splitting into an array, 32

switch statements, 20-21

variables

- defining, 15-16, 27
- scoping, 26-27

**journal option**

- database changes, 251
- MongoClient connections, 230

**jQuery, AngularJS, integration, 402-403**

**JSON (JavaScript Object Notation), 73-74**

- converting JavaScript objects to, 74
- converting to JavaScript objects, 74

- reading string files to objects, 106
- responses, sending, 369-370

**json filter (AngularJS templates), 438**

**json replacer setting (Express), 357**

**json spaces setting (Express), 357**

**jsonp callback name setting (Express), 357**

---

## K

### key option

- https.request method, 140
- tls.connect() method, 158
- tls.createServer() method, 159

**keys parameter, group() method, 287**

**keywords directive (package.json file), 46**

**kill() method, 165, 169, 180**

**killSignal option (exec() function), 170**

---

## L

**\$last aggregation expression operator, 292**

**length, buffers, determining, 79**

**life cycles, AngularJS scopes, 401-402, 424-425**

- bootstrap phase, 401
- compilation phase, 402
- runtime data binding phase, 188

**limit() method, 307, 323**

**\$limit operator, aggregation() method, 290**

**limit option (options object), 272**

**limitTo:limit filter (AngularJS templates), 438**

**link() function, manipulating DOM, 465-466**

**link property (AngularJS templates), 461**

**listDatabases() method, 237**

**listen() method, 149**

**listeners, event, 65-66**

**listening event**

- cluster module, 178
- net.Socket object, 149

**listing files, 113****listing users, MongoDB, 212-213****loadavg() method, 185****local variables, defining, 27****localAddress option**

- ClientRequest object, 123
- Socket object, 145

**localAddress property (Socket objects), 147****\$locale built-in service (AngularJS), 472****localPort property (Socket objects), 147****locals, Express template engine, adding, 375-376****\$location built-in service (AngularJS), 472****location() method, 367****login routes, implementing, 497****login view, implementing, 504****logout() method, 235, 237****lookup() method, 191****loops, JavaScript, 21**

- for/in, 23
- do/while, 22
- for, 22-23
- interrupting, 23-24
- while, 21-22

**lowercase filter (AngularJS templates), 438****lowercase() method, 404****lt() method, 309****lte() method, 309**


---

## M

**main directive (package.json file), 46****makePurchase() function, 575-576****MapReduce, aggregating results, 289-293****markModified() method, 310****match() method, 323****\$match operator, aggregation() method, 290****\$max aggregation expression operator, 292****max role (MongoDB), 221****maxAge property (options parameter), 387****maxBuffer option, exec() function, 170****maxScan option (options object), 272****maxTickDepth property (process module), 165****media files, 9****memoryUsage() method, 165****message event**

- ChildProcess object, 168
- Worker object, 180

**method option (ClientRequest object), 123****method property**

- config parameter, 473
- Request object, 365

**methods**

- in(), 309
- or(), 309
- and(), 309
- abort(), 165
- acceptsCharset(), 365
- address(), 146, 149
- addUser(), 213, 235, 237
- admin(), 235
- aggregate(), 289-290, 306, 325
- all(), 309
- angular.module(), 413
- append(), 323
- app.render(), 378
- arch(), 185
- attachment(), 367
- authenticate(), 235, 237

- chdir(), 165
- ChildProcess objects, 169
- chunkCollection(), 350
- close(), 149, 235, 240, 350
- cluster module, 179
- collection(), 235, 350
- collectionInfo(), 235
- collectionNames(), 235
- collections(), 235
- comment(), 307
- constant(), 415
- copy(), 404
- count(), 238, 240, 306
- cpus(), 185
- create(), 306
- createCollection(), 235, 336
- cwd(), 165
- db(), 235
- destroy(), 146, 392
- directive(), 461-467
- disconnect(), 169, 179-180, 299
- distinct(), 238, 285, 306
- drop(), 238
- dropCollection(), 235
- dropDatabase(), 235
- each(), 240
- element(), 404
- elemMatch(), 309
- encoding between strings and binary buffers, 75
- end(), 146
- endianness(), 185
- eof(), 350
- equals(), 310, 404
- errors, 310
- exec(), 323
- exists(), 309
- exit(), 165
- extend(), 404
- factory(), 415
- find(), 238, 256, 278-279, 306
- findAndModify(), 238
- findAndRemove(), 238, 267
- findOne(), 238, 256, 306
- findOneAndRemove(), 306
- findOneAndUpdate(), 306
- forEach(), 404
- freemem(), 185
- fromJson(), 404
- fs.stats(), 112
- get(), 310, 349, 365, 367
- getc(), 350
- getConnections(), 149
- getgid(), 165
- getuid(), 165
- group(), 286-288, 323
- gt(), 309
- gte(), 309
- hint(), 307
- hostname(), 185
- hrtime(), 165
- id(), 310
- initgroups(), 165
- insert(), 238, 255
- invalidate(), 310
- isArray(), 404
- isClosed(), 240
- isDate(), 404
- isDefined(), 404
- isElement(), 404
- isFunction(), 404
- isInit(), 310
- isModified(), 310
- isNew(), 310

- isNumber(), 404
- isObject(), 404
- isSelected(), 310
- isString(), 404
- isUndefined(), 404
- kill(), 165, 169, 180
- limit(), 307, 323
- listDatabases(), 237
- listen(), 149
- loadavg(), 185
- location(), 367
- logout(), 237
- logout() method, 235
- lookup(), 191
- lowercase(), 404
- lt(), 309
- lte(), 309
- manipulating arrays, 33-34
- manipulating string objects, 30
- markModified(), 310
- match(), 323
- memoryUsage(), 165
- mod(), 309
- model(), 304
- modifiedPaths(), 310
- ne(), 309
- net.Server object, 149
- networkInterfaces(), 185
- nextObject(), 240
- nextTick(), 63-64, 165
- nin(), 309
- nor(), 309
- open(), 235, 350
- os module, 185
- platform(), 185
- passport.authenticate(), 510
- pause(), 146
- ping(), 237
- process execution, 165
- process module, 165-167
- project(), 323
- provider(), 416
- put(), 348
- puts(), 350
- queryRemover(), 395
- read(), 307, 323, 350
- readlines(), 350
- ref(), 146, 149
- regenerate(), 392, 497
- regex(), 309
- release(), 185
- remove(), 238, 306, 310, 320-322, 499
- removeUser(), 235, 237
- rename(), 238
- renameCollection(), 235
- req.logout(), 511
- res.cookie(), 387-388
- resolve(), 191
- resume(), 146
- reverse(), 191
- rewind(), 240, 350
- safe(), 307
- save(), 238, 262, 310, 392, 498
- schema, 310
- Schema object, adding to, 302-303
- seek(), 350
- select(), 307
- send(), 169, 180
- serializeUser(), 510
- serverStatus(), 237, 244
- service(), 415
- set(), 310, 367
- setEncoding(), 146
- setgid(), 165

- setgroups(), 165
- setKeepAlive(), 146
- setNoDelay(), 146
- setOptions(), 307
- setTimeout(), 146
- setuid(), 165
- setupMaster(), 179
- size(), 309
- skip(), 307, 323
- snapshot(), 307
- Socket objects, 146-147
- sort(), 240, 307, 323
- stream(), 350
- tell(), 350
- tls.connect(), 158
- tls.createServer(), 159
- tmpdir(), 185
- toArray(), 240
- toJSON(), 310
- toJson(), 404
- toObject(), 310
- toString(), 310
- totalmen(), 185
- touch(), 392
- type(), 185, 367
- unlink(), 350
- unref(), 146, 149
- unwind(), 323
- update(), 238, 306, 310
- uppercase(), 404
- uptime(), 165, 185
- validate(), 310
- value(), 415
- where(), 309
- write(), 146, 350
- writeFile(), 350

### **middleware, Express, 381-382, 396**

- adding multiple functions, 383
- assigning globally to paths, 382-383
- assigning to a single route, 383
- basic-auth-connect, 390-391
- cookie-parser, 387-388
- cookie-session, 388-389
- creating custom, 395
- POST parameters, 386
- query, 383-384
- session, 392-393
- static, 384

### **middleware framework, Mongoose, 328-329**

**\$min aggregation expression operator, 292**

**\$mod aggregation expression operator, 292**

**mod() method, 309**

**model() method, 304**

**model mutation phase (AngularJS scope), 425**

### **model-based controller routes**

- implementing, 521-526
- shopping cart, 552-555

**models, Mongoose, compiling, 304**

**modifiedPaths() method, 310**

**Module object, creating, 413**

### **modules, 185**

- AngularJS, 398-399, 411-412, 418
  - adding providers to, 414-416
  - configuration blocks, 413-414
  - creating Module object, 413
  - defining, 412-416
  - implementing, 507-508
  - implementing to support comment views, 534-540
  - run blocks, 414
  - service integration, 488-489

## Node.js

- child\_process, 168-176
- cluster, 178-179
- console, writing data to, 52
- dns, 191-192
- fs, 97-98
- NPMs (Node Packaged Modules), 42-47, 491, 508-512
- os, 185-186
- Packaged Modules, 48-52
- process, 163-167
- Stream, 83-92
- util, 187-192
- Zlib, 93-95

**mongod command-line, 345-346**

- parameters, 208

**MongoDB, 3-4, 12-13, 195-196, 207, 226, 227, 295, 355**

- accessing from shell client, 209-212
- atomic write operations, 202
- authentication, 217
- backing up, 354-355
- benefits, 3-4
- building environment, 207-212
- Client object, connecting to via, 230-231
- collections, 196
  - adding documents to, 223, 254-255
  - atomically modifying documents in, 260-261
  - capped, 202, 336
  - counting specific sets of documents in, 277-278
  - creating, 221, 246-247
  - deleting, 222, 247
  - deleting documents from, 224, 265
  - displaying in databases, 220-221
  - finding documents in, 222, 257, 274-275

- forcing required fields, 302
- grouping sets of documents by specific fields in, 288, 294
- implementing unique fields, 302
- indexing, 203
- inserting documents into, 255, 263
- large, 203-204
- limiting specific sets of documents, 279
- listing, 246-247
- managing, 220-225
- obtaining documents from, 256-257
- obtaining information, 248
- paging results, 282
- removing single document from, 266-267
- replications, 203
- retrieving and displaying stats, 248
- saving documents in, 262
- sharding, 203, 340-347
- updating documents in, 224-225, 258-259
- word\_stats, 303
- commands, parameters and results, 211
- connecting to
  - connection strings, 232-234
  - MongoClient object instance, 231-232
- data life cycles, 204
- data models, 199-204
- data normalization, 199-200
- data set, 269-270
- data types, 198
- data usability, 204
- databases
  - administering, 218-220
  - applying replication, 337-340
  - change options, 251

- connecting to, 298-299
- copying, 220
- creating, 242-243
- deleting, 219, 242-243
- displaying collections in, 220-221
- listing, 242-243
- repairing, 353-354
- update operators, 252-253
- denormalizing data, 200-201
- documents, 196-198
  - adding to collections, 223, 254-255
  - adding via Mongoose, 314
  - aggregating via Mongoose, 323-325
  - atomically modifying in collections, 260-261
  - counting, 276-277
  - deleting from collections, 224, 265
  - document references, 199-200
  - embedding, 200-201
  - finding distinct field values in a specific set of, 286
  - finding in collections, 222
  - finding specific sets of, 273-274
  - finding via Mongoose, 312
  - growth, 203
  - inserting in collections, 263
  - limiting fields returned by, 280
  - limiting result sets, 278-282
  - obtaining from collections, 256-257
  - querying, 270-272
  - removing single from collection, 266-267
  - removing via Mongoose, 320-322
  - saving in collections, 262
  - updating in collections, 224-225, 258-259
  - updating via Mongoose, 316-319
- driver, objects, 235-240
- field values, finding, 285
- GridFS
  - implementing, 348-352
  - implementing Grid object from Node.js, 348-349
  - implementing GridStore object from Node.js, 350-352
- indexes, adding, 333-336
- installing, 207-208
- methods, 211
- Node.js
  - adding driver to, 227-228
  - connecting from, 228-234
- query objects, 270-271
  - options, 272
- query results
  - aggregating, 289-293
  - grouping, 286-288
  - paging, 281-282
  - sorting sets, 283
- scripting shell, 212
- server, obtaining status, 244
- Server object, 229
- shared keys, choosing, 341
- shell commands, 210
- starting, 208
- stopping, 209
- user accounts
  - administering, 212-215
  - assigning database roles to, 214
  - configuring access control, 216-218
  - creating, 213-215
  - creating database administrator accounts, 217-218
  - creating user administrator accounts, 216

- listing, 212-213
- removing, 215
- write concern levels, 228
- MongoDB Node.js driver, 249, 267, 269, 295. See also MongoDB**
  - adding documents to collections, 254-255
  - aggregating query results, 289-293
  - atomically modifying documents in collections, 260-261
  - counting documents, 276-277
  - creating collections, 246-247
  - databases
    - change options, 251
    - creating, 242-243
    - deleting, 242-243
    - update operators, 252-253
  - deleting collections, 246-247
  - deleting documents from collections, 265
  - driver objects, 235-240
  - finding field values, 285
  - finding specific sets of documents, 273-274
  - grouping results, 286-288
  - inserting documents in collections, 263
  - limiting result sets, 278-282
  - listing collections, 246-247
  - listing databases, 242-243
  - Mongoose, 297-298
  - obtaining documents from collections, 256-257
  - obtaining server status, 243
  - query objects, 270-271
  - query options objects, 272
  - querying documents, 270-272
  - removing a single document from a collection, 266-267
  - sorting result sets, 283
  - updating documents in collections, 258-259
- Mongoose, 297-298, 331**
  - CommentThread model, defining, 518-519
  - Document object, 310
  - documents
    - adding, 314
    - aggregating, 323-325
    - finding, 312
    - removing, 320-322
    - updating, 316-319
  - middleware framework, 328-329
  - models, compiling, 304
  - MongoDB databases, connecting to, 298-299
  - objects, 298
  - Page model, defining, 517
  - paths, 300
  - Photo model, defining, 518
  - Query object, 305-309
    - setting query database operation, 306
    - setting query operators, 308-309
  - Schema object, 300-301
    - adding methods to, 302-303
  - schemas, defining, 300-303
  - user model, defining, 492-493
  - validation framework, 326-327
- multi option (database changes), 251**
- multikey index (MongoDB), 334**
- multiple documents, updating, via Mongoose, 316-319**
- \$multiply aggregation expression operator, 292**
- mutation observation phase (AngularJS scope), 425**

## N

---

- name directive (package.json file), 46
- ne() method, 309
- net.Server object, 148-151
- net.Socket object, 144-148
- network sockets, 143-144
- networkInterfaces() method, 185
- new option (database changes), 251
- newSession event (TLS Server objects), 159
- nextObject() method, 240
- nextTick() method, 165
  - scheduling work, 63-64
- ngApp directive (AngularJS templates), 448
- ngBind directive (AngularJS templates), 453
- ngBindHtml directive (AngularJS templates), 453
- ngBindTemplate directive (AngularJS templates), 453
- ngBlur directive (AngularJS templates), 458
- ngChange directive (AngularJS templates), 458
- ngChecked directive (AngularJS templates), 458
- ngClass directive (AngularJS templates), 453, 481
- ngClassEvent directive (AngularJS templates), 453
- ngClassOdd directive (AngularJS templates), 453
- ngClick directive (AngularJS templates), 458
- ngCloak directive (AngularJS templates), 448
- ngController directive (AngularJS templates), 448
- ngCopy directive (AngularJS templates), 458
- ngCut directive (AngularJS templates), 458
- ngDbClick directive (AngularJS templates), 458
- ngDisabled directive (AngularJS templates), 453
- ngFocus directive (AngularJS templates), 458
- ngHide directive (AngularJS templates), 453, 481
- ngHref directive (AngularJS templates), 448
- ngIf directive (AngularJS templates), 453, 481
- ngInclude directive (AngularJS templates), 448, 481
- ngInit directive (AngularJS templates), 453
- ngKeyDown directive (AngularJS templates), 458
- ngKeyPress directive (AngularJS templates), 458
- ngKeyUp directive (AngularJS templates), 458
- ngList directive (AngularJS templates), 448
- ngModel directive (AngularJS templates), 453
- ngMouseDown directive (AngularJS templates), 458
- ngMouseenter directive (AngularJS templates), 458
- ngMouseleave directive (AngularJS templates), 458
- ngMouseMove directive (AngularJS templates), 458
- ngMouseover directive (AngularJS templates), 458
- ngMouseup directive (AngularJS templates), 458
- ngNonBindable directive (AngularJS templates), 448
- ngOpen directive (AngularJS templates), 448
- ngOptions directive (AngularJS templates), 450

- ngPaste directive (AngularJS templates), 458
- ngPluralize directive (AngularJS templates), 448
- ngReadonly directive (AngularJS templates), 448
- ngRepeat directive (AngularJS templates), 453, 481
- ngRequired directive (AngularJS templates), 448
- ngSelected directive (AngularJS templates), 448
- ngShow directive (AngularJS templates), 453, 481
- ngSrc directive (AngularJS templates), 448
- ngSrcset directive (AngularJS templates), 448
- ngStyle directive (AngularJS templates), 453
- ngSubmit directive (AngularJS templates), 458
- ngSwipeLeft directive (AngularJS templates), 458
- ngSwipeRight directive (AngularJS templates), 458
- ngSwitch directive (AngularJS templates), 453, 481
- ngTransclude directive (AngularJS templates), 448
- ngValue directive (AngularJS templates), 453
- ngView directive (AngularJS templates), 448, 481
- nin() method, 309
- Node Package Registry, 42
- Node.js, 2-3, 12, 39-40, 53-55, 72-73, 96-97, 117, 183-184, 193
  - AngularJS, adding to environment, 403
  - applications, creating, 47-52
  - benefits, 3
  - buffering data, Buffer module, 74-82
  - callbacks, implementing, 68-71
  - clusters, HTTP, 181
  - event queue, adding work to, 59-66
  - events
    - callbacks, 56
    - emitters, 65-66
    - listeners, 65-66
    - model, 55-59
  - Express, 357
    - configuring, 357
    - configuring routes, 359-363
    - implementing template engines, 374-378
    - Request object, 365
    - Response object, 366-374
    - starting servers, 358-359
  - file paths, verifying existence, 110-111
  - file system
    - calls, 97-98
    - creating and removing directories, 115-116
    - renaming directories, 116
  - files
    - deleting, 114
    - listing, 113
    - monitoring changes, 117
    - obtaining information, 111-112
    - opening and closing, 98-99
    - renaming, 116
    - truncating, 114-115
    - writing, 100-104
  - HTTP services, 119, 142
    - http.ClientRequest object, 122-124
    - http.IncomingMessage object, 126
    - http.ServerResponse object, 125
    - implementing clients and servers, 130-137

- implementing HTTPS servers and clients, 139-142
  - processing query strings and form parameters, 121-122
  - processing URLs (uniform resource locators), 119-121
  - Server object, 128-130
- IDEs, selecting, 41
- installing, 40-41
- I/O (input/output), blocking, 57
- JSON (JavaScript Object Notation), 73-74
- modules, 68-69, 185
  - child\_process, 168-176
  - cluster, 178-179
  - console, writing data to, 52
  - dns, 191-192
  - fs, 97-98
  - NPMs (Node Packaged Modules), 42-47, 491, 508-512
  - os, 185-186
  - Packaged Modules, 48-52
  - process, 163-167
  - Stream, 83-92
  - util, 187-192
  - Zlib, 93-95
- MongoDB
  - adding driver to, 227-228
  - connecting to from, 228-234
  - implementing Grid object from, 348-349
  - implementing GridStore object from, 350-352
  - obtaining collection information, 248
  - saving documents in collections, 262
  - write concern levels, 228
- MongoDB driver, 249, 267, 269, 295
  - adding documents to collections, 254-255
  - aggregating query results, 289-293
  - atomically modifying documents in collections, 260-261
  - counting documents, 276-277
  - creating collections, 246-247
  - creating databases, 242-243
  - database change options, 251
  - database update operators, 252-253
  - deleting collections, 246-247
  - deleting databases, 242-243
  - deleting documents from collections, 265
  - driver objects, 235-240
  - finding field values, 285
  - finding specific sets of documents, 273-274
  - grouping results, 286-288
  - inserting documents in collections, 263
  - limiting result sets, 278-282
  - listing collections, 246-247
  - listing databases, 242-243
  - obtaining documents from collections, 256-257
  - obtaining server status, 243
  - query objects, 270-271
  - query options objects, 272
  - querying documents, 270-272
  - removing a single document from a collection, 266-267
  - sorting result sets, 283
  - updating documents in collections, 258-259
- Mongoose, 297-298
  - compiling models, 304
  - connecting to MongoDB databases, 298-299

- defining schemas, 300-303
- Document object, 310
- objects, 298
- Query object, 305-309
- NPMs (Node Packaged Modules), 42-47
  - 491
  - creating, 48-49
  - installing NPMs (Node Packaged Modules), 44-45
  - Passport, 508-512
  - publishing, 49-50
  - searching for, 44
  - using in applications, 51-52
- packages, 42
  - installing NPMs (Node Packaged Modules), 44-45
  - Node Package Registry, 42
  - NPM (Node Package Manager), 43
  - NPMs (Node Packaged Modules), 42
  - package.json file, 46-47
- process clusters, implementing, 178-181
- reading files, 105-109
- scaling applications, 163
- socket services, 143-144, 162
  - implementing TCP socket clients, 151-152
  - implementing TCP socket servers, 154-155
  - implementing TLS clients, 157-158
  - implementing TLS servers, 159
  - Socket objects, 144-151
    - TCP Server, 144-151
- timers, implementing, 60-63
- verifying executables, 41
- Worker object, 179-180

## **Node.js-to-AngularJS stack, 7, 11, 14**

- AngularJS, 14
- Express module, 13
- MongoDB, 12-13
- Node.js, 12

## **nor() method, 309**

## **NoSQL, MongoDB, 195-196**

- collections, 196
- data models, 199-204
- data types, 198
- documents, 196-198

## **NPM (Node Package Manager), 43**

## **NPMs (Node Packaged Modules), 42-47, 491**

- creating, 48-49
- installing NPMs (Node Packaged Modules), 44-45
- Passport, 508-512
- publishing, 49-50
- searching for, 44
- using in applications, 51-52

## **NPNProtocols option, tls.createServer() method, 159**

## **number[:fraction] filter (AngularJS templates), 438**

## **numberOfRetries option (MongoClient connections), 230**

## **numberOfRetries option (options object), 272**

## **numbers, JavaScript, 16**

---

## O

## **object literal, JavaScript, 17**

## **objects, 27**

- Buffer
  - concatenating, 82
  - copying data from one another, 80
  - creating and manipulating slices, 81

- reading from, 78
  - writing to, 77
- checking types, 188
- ChildProcess, 168-169
- Client, connecting to MongoDB via, 230-231
- creating custom defined, 28-29
- Document, 310
- EventEmitter, creating custom, 66-67
- Grid, 348-349
- GridStore object, implementing from Node.js, 350-352
- inheriting functionality from other, 190
- JavaScript
  - converting JSON to, 74
  - converting to JSON, 74
  - converting to strings, 189
- JSON string files, reading to, 106
- limiting fields returned in, 280
- Model, 304
- Module, creating, 413
- MongoDB driver, 235-240
  - Admin object, 237
  - Collection object, 238
  - Cursor object, 240
  - Db object, 235
- Mongoose, 298
- prototyping patterns, 29-30
- Query, 305-309
- query, 270-271
  - options, 272
- Request, 365
- Response, 366-374
- Schema, 300-301
  - adding methods to, 302-303
- Server, MongoDB, 229

- Socket
  - events, 145
  - net.Server, 148-151
  - net.Socket, 144-148
  - properties, 147
  - syntax, 27-28
  - Worker, 179-180
- online event (cluster module), 178**
- open() method, 235, 350**
- opening files, 98-99**
- operators**
  - aggregation, 290
  - JavaScript, 17-21
    - arithmetic, 17-18
    - assignment, 18
    - comparison, 19-21
    - conditional, 20
    - update, databases, 252-253
- options object, 272**
- options parameter, group() method, 287**
- or() method, 309**
- order model controllers, shopping cart, implementing, 553-554**
- order schema, shopping cart, defining, 548-549**
- orderBy filters, AngularJS, 442-443**
- orderBy:exp:reverse filter (AngularJS templates), 438**
- ordering AngularJS filters, 441**
- originalUrl property (Request object), 365**
- os module, 185-186, 193**
- otherDBRoles field, addUser() method, 213**

---

## P

- Packaged Modules. See NPMs (Node Packaged Modules)**
- package.json file, 46-47**

**packages (Node.js), 42**

- Node Package Registry, 42
- NPM (Node Package Manager), 43
- NPMs (Node Packaged Modules), 42-47, 491
  - creating, 48-49
  - installing, 44-45
  - Passport, 508-512
  - publishing, 49-50
  - searching for, 44
  - using in applications, 51-52

**package.json file, 46-47****page controllers, AngularJS, implementing, 537-538****Page model**

- defining, 517
- implementing controller, 521-522

**parameters**

- callbacks, passing additional to, 68
- Express routes, applying in, 361-363
- form, processing, 121-122

**params property (config parameter), 473****partial option (options object), 272****partitioning methods, MongoDB, selecting, 343-344****passphrase option**

- https.request method, 140
- tls.connect() method, 158
- tls.createServer() method, 159

**Passport NPM, 508-512, 514****passport.authenticate() method, 510****path option (ClientRequest object), 123****path property**

- options parameter, 387
- Request object, 365

**paths**

- Express middleware, assigning globally to, 382-383

files, verifying existence, 110-111

Mongoose, 300

**patterns, objects, prototyping, 29-30****pause() method, 146****px option**

- https.request method, 140
- tls.connect() method, 158
- tls.createServer() method, 159

**photo controllers, implementing, 535-537****Photo model**

- defining, 518
- implementing controller, 522

**photo view, implementing, 527-528****pid property**

- ChildProcess object, 169
- process module, 165

**ping() method, 237****pipes, process I/O, 163-164****platform() method, 185****platform property (process module), 165****poolSize option (Server object), 229****\$pop, 253****port option**

- ClientRequest object, 123
- Socket object, 145

**POST parameters, Express middleware, 386****POST servers, implementing, 135-136****preferGlobal directive (package.json file), 46****process execution, controlling, 165****process module, 163-167**

- controlling process execution, 165
- I/O (input/output) pipes, 163-164
- obtaining information from, 165-167
- signals, 164

**process property (Worker object), 180**

**processes**

- clusters, implementing, 178-181
- executable files, executing on another process, 171-172
- executing executable files on another, 172
- executing system commands in another, 171
- spawning command in another, 174-175
- spawning in another instance, 173-174
- system commands, executing on another, 170
- worker, 181-182

**processing query strings and form parameters, 121-122****product model controllers, shopping cart, implementing, 552****product page view, shopping cart, implementing, 559****product schema, shopping cart, defining, 547-548****products view, shopping cart, implementing, 558****project() method, 323****\$project operator, aggregation() method, 290****properties**

- ChildProcess object, 169
- cluster module, 179
- process module, 165-167
- Socket objects, 147

**protocol property (Request object), 365****provider() method, 416****providers, AngularJS modules, adding to, 414-416****publishing Packaged Modules, 49-50****\$pull update operator, 253****\$pullAll update operator, 253****purchases, shopping cart, 575-576****\$push aggregation expression operator, 292****\$push update operator, 253****put() method, 348****Put requests, sending, AngularJS, 473-475****puts() method, 350****pwd field, addUser() method, 213**


---

## Q

**quantity schema, shopping cart, defining, 548****queries, MongoDB**

- aggregating results, 289-293
- documents, 270-272
- grouping results, 286-288
- objects, 270-271
  - options, 272
- paging results, 281-282
- sorting result sets, 283

**query middleware (Express), 383-384****Query object, 305-309**

- query operators, setting, 308-309

**query parameter, group() method, 287****query property (Request object), 365****query router servers, starting, 345-346****query strings, processing, 121-122****queryRemover() method, 395**


---

## R

**range-based sharding, MongoDB, 344****read database role (MongoDB), 214****read() method, 307, 323, 350****read option (Schema object), 300****Readable streams, 83-85**

- piping to Writable streams, 92

- readAnyDatabase database role (MongoDB), 214
- reading from buffers, 77-78
- readlines() method, 350
- readPreference option
  - options object, 272
  - Server object, 229
- readWrite database role (MongoDB), 214
- readWriteAnyDatabase database role (MongoDB), 214
- redirecting responses, 373-374
- reduce parameter, group() method, 287
- ref() method, 146, 149
- regenerate() method, 392, 497
- regex() method, 309
- rejectUnauthorized option
  - https.request method, 140
  - tls.connect() method, 158
  - tls.createServer() method, 159
- release() method, 185
- remoteAddress property (Socket objects), 147
- remotePort property (Socket objects), 147
- remove() method, 238, 306, 310, 320-322, 499
- removeUser() method, 235, 237
- rename() method, 238
- \$rename update operator, 253
- renameCollection() method, 235
- renaming files, 116
- repairing MongoDB databases, 353-354
- replace property (AngularJS templates), 461
- replication
  - MongoDB collections, 203
  - MongoDB databases, applying, 337-340
- repository directive (package.json file), 46
- req.logout() method, 511
- Request object, 365
- requestCert option (tls.createServer() method), 159
- requests, routes, shopping cart, 551
- require property (AngularJS templates), 461
- required fields, MongoDB collections, forcing, 302
- res.cookie() method, 387-388
- resolve() method, 191
- \$resource built-in service (AngularJS), 472
- Response object (Express), 366-374
  - download responses, sending, 373
  - redirecting responses, 373-374
  - sending files, 371-372
  - sending JSON responses, 369-370
  - sending responses, 368
  - setting headers, 366-367
  - setting status, 367-368
- response option (ClientRequest object), 124
- responses, Express templates, rendering in, 378
- responseType property (config parameter), 473
- restrict property (AngularJS templates), 461
- result sets, MongoDB queries, 278-283
- resume() method, 146
- resumeSession event (TLS Server objects), 159
- retryMilliseconds option (MongoClient connections), 230
- reverse() method, 191
- review view, shopping cart, implementing, 565
- rewind() method, 240, 350
- RIAs (rich Internet applications), 583

- roles field, addUser() method, 213
- root scope, AngularJS applications, relationship between, 419-420
- \$rootElement** built-in service (AngularJS), 472
- \$rootScope** built-in service (AngularJS), 472
- \$route** built-in service (AngularJS), 472
- routes
  - comments, 520-521
  - Express
    - applying parameters in, 361-363
    - configuring, 359-363
  - model-based controller, implementing, 521-526
  - servers
    - implementing, 494-496
    - implementing user controller, 496-501
  - shopping cart
    - model-based controller, 552-555
    - request, 551
    - view support, 587
- run blocks, AngularJS modules, 414
- runtime data binding phase, AngularJS life cycle, 188

## S

---

- safe() method, 307
- safe option (Schema object), 300
- save() method, 238, 262, 310, 392, 498
- \$sce** built-in service (AngularJS), 472
- schema method, 310
- Schema object, 298, 300-301
  - adding methods to, 302-303
- schemas
  - adding indexes to, 301-302
  - defining, Mongoose, 300-303
  - word\_stats collection (MongoDB), implementing on an, 303

- \$set** update operator, 253
- \$setOnInsert** update operator, 253
- scope destruction phase (AngularJS scope), 425
- scope property (AngularJS templates), 461
- scopes
  - AngularJS, 399, 419, 431-432
    - configuring, 464-465
    - emitting and broadcasting events, 428-429
    - expressions, 436
    - filters, 440
    - implementing hierarchy, 425-426
    - life cycle, 424-425
    - relationship between backend server data, 423-424
    - relationship between controllers, 420-421
    - relationship between root scope and applications, 419-420
    - relationship between templates, 421-422
    - sorting and ordering, 442
    - shopping cart, initializing, 571
- scoping variables, 26-27
- script directive (AngularJS templates), 448
- scripts directive (package.json file), 46
- secure property (Request object), 365
- secureConnection event (TLS Server objects), 159
- secureProtocol option
  - https.request method, 140
  - tls.connect() method, 158
  - tls.createServer() method, 159
- security, authentication, Express sessions, 392-393
- seek() method, 350
- select directive (AngularJS templates), 450
- select() method, 307

- send() method, 169, 180**
- sending responses, Express, 368-370**
- serializeFunctions option (database changes), 251**
- serializeUser() method, 510**
- Server object, 128-130**
  - MongoDB, 229
- server types, sharding, 341**
- servername option, tls.connect() method, 158**
- servers**
  - application, creating, 586-587
  - comments, creating, 520
  - creating, 493-494
  - Express, starting, 358-359
  - GET, implementing dynamic, 132-133
  - HTTPS (Hypertext Transfer Protocol Secure), implementing, 139-140, 142
  - MongoDB driver, obtaining status, 244
  - POST, implementing, 135-136
  - query router, starting, 345-346
  - routes
    - implementing, 494-496
    - implementing user controller, 496-501
  - shopping cart, creating, 550-551
  - TCP socket, implementing, 154-155
  - TLS (Transport Layer Security), implementing, 159
- serverStatus() method, 237, 244**
- service() method, 415**
- service services (AngularJS), creating, 488**
- services**
  - AngularJS, 400, 471, 489
    - \$animate, 481-484
    - \$cacheFactory, 477-478
    - \$cookieStore, 478-479
    - \$http, 473-475
    - \$interval, 481
    - \$timeout, 481
    - \$window, 478
  - built-in, 472-484
  - creating constant services, 487
  - creating custom, 487-489
  - creating factory services, 487-488
  - creating service services, 488
  - creating value services, 487
  - integrating in modules, 488-489
  - command, creating, 534-535
- session middleware (Express), 392-393**
- sessionIdContext option (tls.createServer() method), 159**
- sessions, Express**
  - authentication, 392-393
  - implementing, 388-389
- set() method, 310, 367**
- setEncoding() method, 146**
- setgid() method, 165**
- setgroups() method, 165**
- setKeepAlive(), 146**
- setNoDelay() method, 146**
- setOptions() method, 307**
- sets of documents, MongoDB, finding specific, 273-274**
- setShipping() function, 574-575**
- setTimeout() method, 146**
- settings property (cluster module), 179**
- setuid() method, 165**
- setup event (cluster module), 178**
- setupMaster() method, 179**
- sharding MongoDB collections, 203, 340-347**
- shared keys, MongoDB, choosing, 341**
- shared tag ranges, MongoDB collections, setting up, 347**

**shell client, MongoDB, accessing from, 209-212**

**shell commands, MongoDB, 210**

**shipping information, shopping cart, setting, 574-575**

**shipping view, shopping cart, implementing, 561**

**shopping cart, 543-546, 581**

adding items to, 572-573

AngularJS modules and controllers, view support, 570-579

billing schema, defining, 547

billing view, implementing, 563

cart view, implementing, 560

checkout, 574

customer model controllers, implementing, 554-555

customer schema, defining, 549-550

defining address schema, 546-547

deleting items from, 573-574

helper functions, 572

initializing, 579-581

making purchases, 575-576

order model controllers, implementing, 553-554

order schema, defining, 548-549

product model controllers, 552

product page view, implementing, 559

product schema, defining, 547-548

products view, implementing, 558

quantity schema, defining, 548

request routes, implementing, 551

review view, implementing, 565

routes, model-based controller, 552-555

servers, creating, 550-551

shipping information, setting, 574-575

shipping view, implementing, 561

shopping scope, initializing, 571

shopping view, implementing, 556

verifying billing, 575

views, implementing, 556-570

**signals, process, 164**

**signed property (options parameter), 387**

**signup routes, implementing, 496-497**

**signup view, implementing, 502**

**silent option (fork() function), 175**

**single field index (MongoDB), 334**

**single routes, assigning Express middleware, 383**

**size, limiting result sets by, 278**

**size() method, 309**

**size role (MongoDB), 221**

**skip() method, 307, 323**

**\$skip operator, aggregation() method, 290**

**skip option (options object), 272**

**\$slice update operator, 253**

**slicing buffers, 81**

**snapshot() method, 307**

**snapshot option (options object), 272**

**SNICallback option (tls.createServer() method), 159**

**social media accounts, using as authentication sources, 508-512**

**Socket objects**

events, 145

methods, 146-147

net.Server, 148-151

net.Socket, 144-148

properties, 147

**socket option (ClientRequest object), 124**

**socketOptions option (Server object), 229**

**socketPath option (ClientRequest object), 123**

**sockets, 143-144**

clients, implementing TCP, 151-152

servers, implementing TCP, 154-155

- sort() method, 240, 307, 323**
- \$sort operator, aggregation() method, 290**
- \$sort update operator, 253**
- sort option (options object), 272**
- sorting, MongoDB query result sets, 283**
- sparse property (MongoDB indexes), 335**
- spawn() function, spawning processes in another instances, 173-174**
- splitting strings into arrays, 32**
- ssl option (Server object), 229**
- stale property (Request object), 365**
- statements**
  - if, 20
  - switch, 20-21
- static files, serving, 130-131**
- static middleware (Express), 384**
- stdin property (ChildProcess object), 169**
- stdio option, spawn() function, 173**
- stdout property (ChildProcess object), 169**
- stderr property (ChildProcess object), 169**
- \$strcasecmp aggregation expression operator, 292**
- stream() method, 350**
- Stream module, 83-92**
- streaming data, Stream module, 83-92**
- streaming reads, files, 110**
- streaming writes, files, allowing to, 104**
- streams**
  - compressing/decompressing, 95
  - Duplex streams, 88-89
  - file writing, 102-103
  - output, synchronous writing to, 188-189
  - Readable streams, 83-85
    - piping to Writable streams, 92
  - Transform streams, 90
  - Writable streams, 86-87
- strict option (Schema object), 300**
- strict routing setting (Express), 357**
- strings**
  - arrays
    - converting into, 35
    - splitting into, 32
  - combining, 31-32
  - connection, components, 233
  - converting JavaScript objects to, 189
  - encoding between binary buffers, 75
  - escape codes, 30
  - formatting, 187-188
  - JavaScript, 16
  - manipulating, 30-32
  - MongoDB, connecting to via, 232-234
  - query, processing, 121-122
  - replacing a word in, 32
  - searching for substrings, 32
- styling**
  - draggable views, 599
  - tables view, 603
  - view HTML files, 507
  - weather view, 596-595
- substrings, searching for, 32**
- \$substr aggregation expression operator, 292**
- \$subtract aggregation expression operator, 292**
- suicide property (Worker object), 180**
- \$sum aggregation expression operator, 292**
- switch statements, 20-21**
- synchronous calls, file system, 97-98**
- synchronous reads, files, performing, 107**
- synchronous writes, files, performing to, 101**
- synchronous writing to output streams, util module, 188-189**

**syntax, objects, 27-28**

**system commands, processes, executing  
on another, 170**

## T

---

**tabbed view templates (AngularJS),  
588-590**

**tabbed views, implementing, 587-591**

**table controller, AngularJS, defining,  
601-603**

**tables view, 603**

initializing words database for, 605-607

**TCP (Transmission Control Protocol), 143**

clients, implementing, 151-152

servers, implementing, 154-155

**tell() method, 350**

**template engines, Express**

adding locals, 375-376

defining, 375

implementing, 374-378

**template property (AngularJS templates),  
461**

**\$templateCache built-in service  
(AngularJS), 472**

**templates**

AngularJS, 399, 433-434, 445

adding controllers to, 407

creating custom filters, 443-444

defining directive view, 462

directives, 433

expressions, 433-435

filters, 437-441

modifying data displayed in  
rendered view, 440

obtaining data from scope model,  
436

replacing elements, 463

scopes, relationship between,  
421-422

transcluding external scopes,  
463-464

views, 469

EJS, implementing user and  
authentication views, 501-507

Express

creating, 376-377

rendering in responses, 378

partial billing, 563-564

tabbed view, 588-590

**templateURL property (AngularJS  
templates), 461**

**text index (MongoDB), 334**

**textarea directive (AngularJS templates),  
450**

**threaded web model versus event  
callbacks, 56-59**

**\$timeout built-in service (AngularJS), 472,  
481**

**timeout event (Socket objects), 145**

**timeout option**

exec() function, 170

options object, 272

**timeout property (config parameter), 473**

**timeout timers, 60**

**timers**

dereferencing, 63

immediate, 62-63

implementing, 60-63

AngularJS, 481

interval, 61

timeout, 60

**title property (process module), 165**

**TLS (Transport Layer Security)**

clients, implementing, 157-158

servers, implementing, 159

**tls.connect()** method, options, 158  
**tls.createServer()** method, 159  
**tmpdir()** method, 185  
**toArray()** method, 240  
**toJSON()** method, 310  
**toJson()** method, 404  
**\$toLower** aggregation expression operator, 292  
**toObject()** method, 310  
**toString()** method, 310  
**totalmen()** method, 185  
**touch()** method, 392  
**\$ToUpper** aggregation expression operator, 292  
**transclude** property (AngularJS templates), 461  
**Transform** streams, 90  
**transformRequest** property (config parameter), 473  
**transformResponse** property (config parameter), 473  
**truncating**, files, 114-115  
**trust proxy** setting (Express), 357  
**try/catch** blocks, error handling, 36  
**TTL** property (MongoDB indexes), 335  
**type()** method, 185, 367

---

## U

**update()** method, 238, 306, 310, 316-319  
**update operators**, databases, 252-253  
**updateUser** route, implementing, 498  
**updating**

- documents in MongoDB collections, 224-225
- documents in Mongoose, 316-319

**upgrade option** (ClientRequest object), 124  
**uppercase filter** (AngularJS templates), 438  
**uppercase()** method, 404  
**upsert option** (database changes), 251  
**uptime()** method, 165, 185  
**URL** object, 120  
**url** property (config parameter), 473  
**URLs** (uniform resource locators)
 

- processing, 119-121
- resolving components, 121

**user accounts**

- MongoDB
  - administering, 212-215
  - assigning database roles to, 214
  - configuring access control, 216-218
  - creating, 213-215
  - listing, 212-213
  - removing, 215
  - websites, adding to, 491-514

**user controller** routes, servers, implementing, 496-501  
**user field**, addUser() method, 213  
**user login** routes, implementing, 497  
**user model**, defining, 492-493  
**user signup** routes, implementing, 496-497  
**user views**, implementing, 501-507  
**userAdmin** database role (MongoDB), 214  
**userAdminAnyDatabase** database role (MongoDB), 214  
**users, websites**, 8

**ucs2()** method, 75  
**uid** option, spawn() function, 173  
**unique** property (MongoDB indexes), 335  
**unlink()** method, 350  
**unref()** method, 146, 149  
**\$unset** update operator, 253  
**unwind()** method, 323  
**\$unwind** operator, aggregation() method, 290

**userSource field, addUser() method, 213**

**utf8() method, 75**

**utf16le() method, 75**

**util module, 187-193**

object types, checking, 188

objects, inheriting functionality from  
other, 190

streams, synchronous writing to  
output, 188-189

strings

converting JavaScript objects to,  
189

formatting, 187-188

## V

---

**validate() method, 310**

**validation framework, Mongoose, 326-327**

**value() method, 415**

**value services (AngularJS), creating, 487**

**values, functions, returning from, 25-26**

**variables**

defining, 15-16, 27

functions, passing to, 25

scoping, 26-27

**verifyBilling() function, 575**

**version directive (package.json file), 46**

**version property (process module), 165**

**versions property (process module), 165**

**view cache setting (Express), 357**

**view engine setting (Express), 357**

**views**

AngularJS templates, 433-434, 445, 447,  
469

built-in directives, 447-459

creating custom filters, 443-444

creating directives to extend  
HTML, 461-467

directives, 433, 447

expressions, 433-435

filters, 433, 437-441

authentication, implementing, 501-507

browsers, rendering, 9-10

comment, implementing, 532

comment thread, implementing,  
530-531

CSS, adding to, 533-534

draggable, 607

implementing, 597-599

index, implementing, 503

login, implementing, 504

photo, implementing, 527-528

routes, support, 587

shopping cart

billing view, 563

cart view, 560

implementing, 556-570

product page view, 559

products view, 558

review view, 565

shipping view, 561

shopping view, 556

signup, implementing, 502

style, adding CSS code to, 507

tabbed, implementing, 587-591

tables

AngularJS, 603

initializing words database for,  
605-607

user, implementing, 501-507

weather

defining, 595

implementing, 592-595

styling, 596-595

**views setting (Express), 357**

---

## W

**w option**

- database changes, 251
- MongoClient connections, 230

**watcher registration phase (AngularJS scope), 424-425****weather controller, AngularJS, defining, 594****weather service view, implementing, 592-595****weather view, AngularJS**

- defining, 595
- implementing, 592-595
- styling, 596-595

**web development framework, 7-11**

- backend services, 10-11

**webpages**

- comment thread view, implementing, 530-531
- comment threads, adding to, 515-541
- comment view, implementing, 532
- photo view, implementing, 527-528

**webservers, 10**

- creating, Express, 493-494
- implementing application database connection, 494
- routes, implementing, 494-496

**websites, users, 8**

- adding to, 491-514

**where() method, 309****while loops, 21-22****\$window built-in service (AngularJS), 472, 478****withCredentials property (config parameter), 473****word\_stats collection (MongoDB), implementing schema on, 303****work**

- adding to event queue, 59-66
- scheduling, nextTick() method, 63-64

**Worker object, 179-180****worker processes, 181-182****worker property (cluster module), 179****workers property (cluster module), 179****Writable streams, 86-87**

- piping from Readable streams, 92

**write concern levels, MongoDB, 228****write() method, 146, 350****writeFile() method, 350****writing files, 100-104****writing to buffers, 76-77****wtimeout option**

- database changes, 251
- MongoClient connections, 230

---

## X-Y-Z

**xsrCookieName property (config parameter), 473****xsrHeaderName property (config parameter), 473****Zlib module, data compression/decompression, 93-95**