**3 Creating and setting up a MEAN project**

**This chapter covers**

* Managing dependencies by using npm and a package.json file
* Creating and configuring Express projects
* Setting up an MVC environment
* Adding Twitter Bootstrap for layout
* Publishing to a live URL and using Git and Heroku

Now we’re really ready to get underway, and in this chapter we’ll get going on building our application. Remember from chapters 1 and 2 that throughout this book we’re going to build an application called Loc8r. This is going to be a location-aware web application that will display listings near users and invite people to log in and leave reviews.

##### GETTING THE SOURCE CODE

The source code for this application is on GitHub at [github.com/simonholmes/getting-MEAN](http://www.github.com/simonholmes/getting-MEAN). Each chapter with a significant update will have its own branch. I encourage you to build it up from scratch through the course of the book, but if you wish you can get the code we’ll be building throughout this chapter from GitHub on the chapter-03 branch. In a fresh folder in terminal, the following two commands will clone it if you already have Git installed:

$ git clone -b chapter-03 <https://github.com/simonholmes/getting-MEAN-2.git>

This will give you a copy of the code that’s stored on GitHub. To run the application, you’ll need to install some dependencies with the following commands:

$ cd getting-MEAN-2

$ npm install

Don’t worry if some of this doesn’t make sense just yet, or if some of the commands aren’t working. During this chapter, we’ll install these technologies as we go.

In the MEAN stack, Express is the Node web application framework. Together Node and Express underpin the entire stack, so let’s start here. In terms of building up the application architecture, figure 3.1 shows where we’ll be focusing in this chapter. We’ll be doing two things:

1. Creating the project and encapsulating the Express application that will house everything else except the database
2. Setting up the main Express application

We’ll start with a bit of groundwork by looking at Express and seeing how we can manage dependencies and modules using npm and a package.json file. We’ll need this background knowledge to get going and set up an Express project.

Before we can really do anything, we’ll make sure that you have everything you need installed on your machine. When that’s all done, we’ll look at creating new Express projects from the command line and the various options we can specify at this point.

Express is great, but you can make it better—and get to know it better—by tinkering a little and changing some things around. This ties into a quick look at model-view-controller (MVC) architecture. Here’s where we’ll get under the hood of Express a little, and see what it’s doing by modifying it to have a very clear MVC setup.

When the framework of Express is set up as we want it, we’ll next include Twitter’s Bootstrap framework and then make the site responsive by updating the Jade templates. In the final step of this chapter we’ll push the modified, responsive MVC Express application to a live URL using Heroku and Git.

## 3.1   A brief look at Express, Node, and npm

**59**

As mentioned before, Express is a web application framework for Node. In basic terms, an Express application is simply a Node application that happens to use Express as the framework. Remember from chapter 1 that npm is a package manager that gets installed when you install Node, which gives you the ability to download Node modules or packages to extend the functionality of your application.

But how do these things work together, and how do you use them? A key piece to understanding this puzzle is the package.json file.

**3.1.1   Defining packages with package.json**

**27**

In every Node application, there should be a file in the root folder of the application called package.json. This file can contain various metadata about a project, including the packages that it depends on to run. The following listing shows an example package.json file that you might find in the root of an Express project.

{

"name": "application-name",

"version": "0.0.0",

"private": true,

"scripts": {

"start": "node ./bin/www"

},

"dependencies": {

"body-parser": "~1.15.2",

"cookie-parser": "~1.4.3",

"debug": "~2.2.0",

"express": "^4.14.0",

"morgan": "^1.7.0",

"pug": "^2.0.0-beta6" ,

"serve-favicon": "~2.3.0"

}

}

This is the file in its entirety, so it’s not particularly complex. There’s various metadata at the top of the file followed by the dependencies section. In this default installation of an Express project, there are quite a few dependencies that are required for Express to run, but we don’t need to worry about what each one does. Express itself is modular so that you can add in components or upgrade them individually.

#### Working with dependency versions in package.json

Alongside the name of each dependency is the version number that the application is going to use. Notice that they’re all prefixed with a ~.

Let’s take a look at the dependency definition for Express 4.14.0. It specifies a particular version at three levels:

* Major version (4)
* Minor version (14)
* Patch version (0)