INTEGRATING FRONT END COMPONENTS with WEB APPLICATIONS

by Maksim Surguy

#frontendweb

Integrating Front end Components with Web Applications

Learn how to implement Bootstrap, tagging, autosuggest, spinners, date pickers, AJAX file uploaders and more in web applications

Maksim Surguy

This book is for sale at http://leanpub.com/frontend

This version was published on 2014-07-21

Tweet This Book!

Please help Maksim Surguy by spreading the word about this book on Twitter!

The suggested tweet for this book is:

If you want to be good at making cool web apps make sure you read this book by @msurguy : https://leanpub.com/frontend #frontendweb

The suggested hashtag for this book is #frontendweb.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#frontendweb

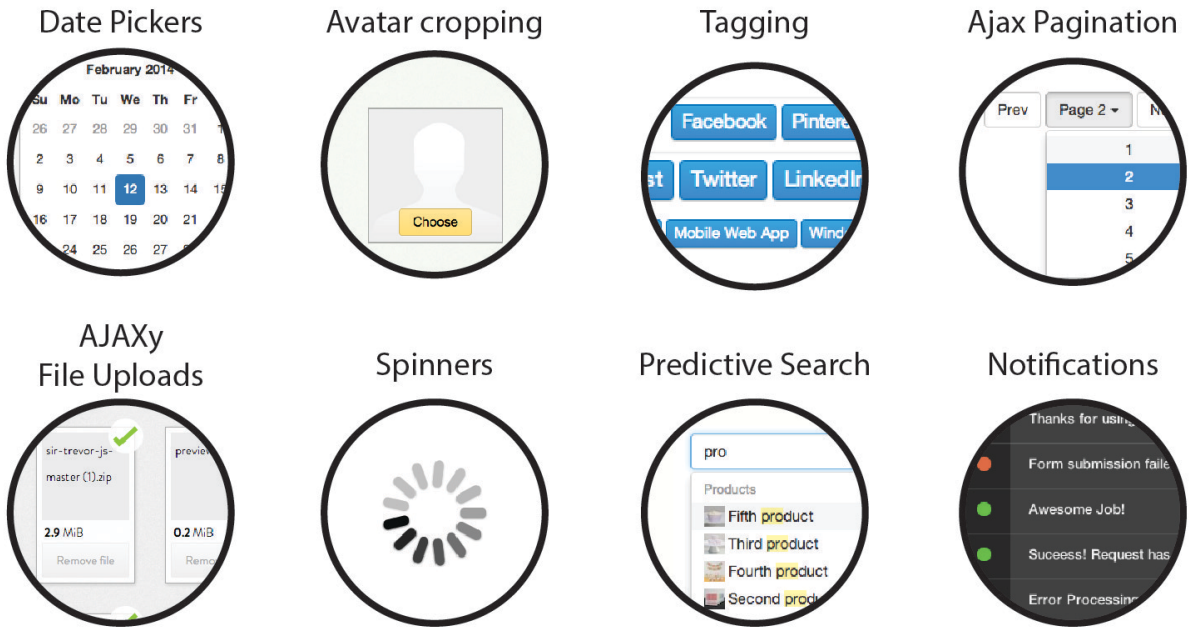
Contents

Introduction

Creating beautiful web applications when you are not a designer could be challenging. That is why frontend frameworks like Bootstrap[1] and Foundation[2] are so popular today. Responsive and user-friendly web design requires a lot of work and often we as developers are just too busy building the backends and don't have the time or necessary skills to create great web design from scratch.

The field of web development moves very quickly, no doubt about that. Technologies for both the frontend and the backend have progressed a lot in the past two/three years. At the same time there are many web developers who are generously dedicating their energy and time to move the web development industry forward by releasing multitude of open source frameworks, plugins and tools. Thanks to them there are now plenty of good solutions to common web development problems.

This book will introduce you to some of the greatest open source front end components that will greatly improve usability of your applications. Some of the components discussed in the book are:



Components discussed in this book

About the author

My name is Maksim Surguy. I am a full time web developer, part time writer and former breakdancer[3]. If you use Laravel PHP framework or Bootstrap, you might have seen some of the things I created:

[1] http://getbootstrap.com
[2] http://foundation.zurb.com
[3] https://www.youtube.com/watch?v=wEF_RHL1NFU

- Bootsnipp[4]
- Laravel-tricks, open source[5]
- Built With Laravel[6]
- Bookpag.es[7]
- Panopanda[8]
- MyMapList[9]
- Cheatsheetr[10]

I love creating new products and in the process I try to share as much as I possibly can. You can read free web development tutorials on my blog at http://maxoffsky.com and you can follow me on Twitter for various web development tips and tricks at http://twitter.com/msurguy

Prerequisites

- Knowing what HTML/JS/CSS terms mean and knowing some basics about them
- Having experience of building at least one web application of any size
- Willingness to learn and be challenged

Conventions for the terms used in the book

This books uses terms “front end” and “back end” extensively. In the context of this book they have the following meaning:

- The term “Front end” usually talks about client’s browser or HTML/CSS/JS that the browser operates with.
- The term “Back end” talks about a web application that resides on a server and potentially works with a database. The web application could be anything that processes and responds to HTTP requests, for example applications using the following technologies/languages could be called “back end”: Node, PHP, Java, Ruby, Python, .NET, etc.

Back end framework of choice for the examples in the book

While a lot of the content in this book could be adapted to applications using any backend framework, all server-side examples in this book will be done using Laravel framework. Laravel is a modern PHP full stack framework that allows you to build web applications using PHP quickly and efficiently. Some key assumptions about the server side code in this book:

[4] http://bootsnipp.com
[5] http://laravel-tricks.com
[6] http://builtwithlaravel.com
[7] http://bookpag.es
[8] http://panopanda.co
[9] http://mymaplist.com
[10] http://cheatsheetr.com

- The server-side example in this book assume that you already know how to use an **MVC framework** or similar. You are familiar with the framework's syntax, routing, ORM and a backend template engine (like Blade in Laravel or Mustache in JS frameworks). If you are not familiar with those concepts I highly encourage you to get up to speed on those topics.
- The tone of this book is geared towards developers that have built at least one application with a backend framework already. If you feel like you are not there yet, don't get discouraged and try codecademy[11] or a similar resource.
- The provided back end code examples in this book assume that you know how to create and populate a database (for example through UI-based tool like PHPMyAdmin or Sequel PRO).

PHP Version used in this book for working examples

The PHP version used in this book is **5.4** with Composer[12] installed.

Let's go over some back end conventions that this book is using.

Conventions used in the book for back end code

Scripts and Styles

In the server side examples where there is more than one front end library used, Javascript and Stylesheets of third-party libraries will be put in 'public/js/vendor' and 'public/css/vendor' folders respectively. Then the scripts and styles will be inserted into the view templates using the syntax in code listings below:

Using HTML::script helper to link to JS file in Laravel

```
1 // Generate a tag for javascript inside a Blade file
2 {{ HTML::script('js/vendor/script.js') }}
3
4 // Results in:
5 <script src="http://localhost:8000/js/vendor/script.js"></script>
```

Using HTML::style helper to link to CSS file in Laravel

```
1 // Generate a tag for javascript inside a Blade file
2 {{ HTML::style('css/vendor/style.css') }}
3
4 // Results in:
5 <link media="all" type="text/css"
6         rel="stylesheet" href="http://localhost:8000/css/vendor/style.css">
```

[11] http://www.codecademy.com/

[12] https://getcomposer.org/doc/00-intro.md#installation-nix

Design patterns and purpose of the book

While there are many great design patterns in existence for back end application architecture, this book will try to remain unbiased and provide the functionality without pushing any specific back end best-practices. This way you can adapt the functionality to your projects using whatever design pattern you prefer.

Also, there will be no Unit Tests provided because that is a topic for a separate book and there is at least one already about that: Laravel Testing Decoded[13] by Jeffrey Way.

The purpose of this book is to expose you to some of the greatest front end components, explain how they work and provide you with easy to understand guides on integrating these libraries with your web applications. Enjoy the journey and prepare to make some cool-looking, jaw-dropping, award-winning web applications!

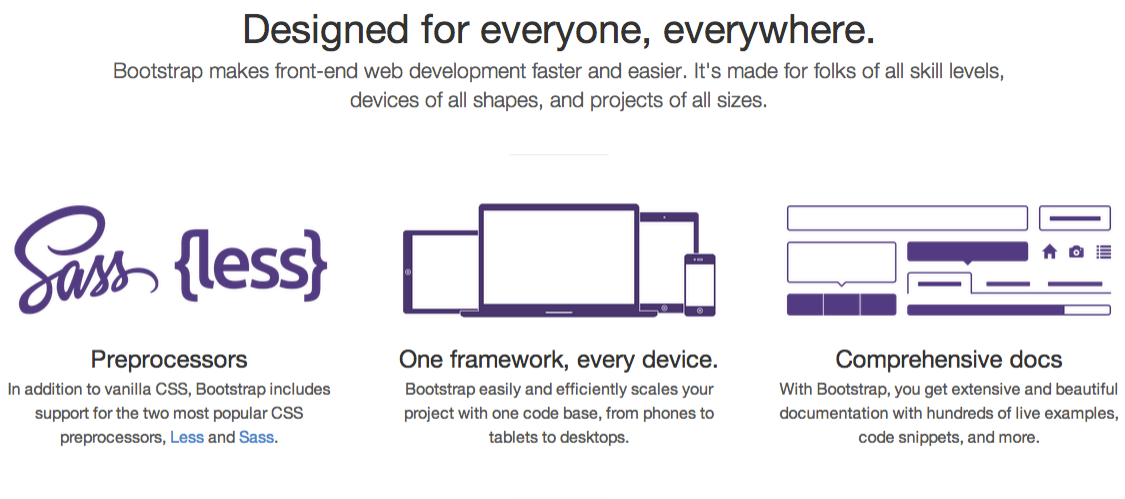
Source Code

The source code for this book is available for each chapter and is located on Github at https://github.com/msurguy/frontend-book[14]. Feel free to explore it, comment on it and improve it on Github.

[13] https://leanpub.com/laravel-testing-decoded
[14] https://github.com/msurguy/frontend-book

1 Using Bootstrap 3 HTML/CSS/JS Framework

Almost any web developer that has made a website or two has heard of Bootstrap framework. Bootstrap is a frontend framework that provides the necessary CSS and Javascript to make your websites responsive and nice looking. A clear advantage of using Bootstrap is that it accelerates web development process by letting you focus on building the application's functionality instead of tinkering with (hopefully) cross-browser load of styling rules and media queries.



Bootstrap framework

Bootstrap has been gaining a lot of popularity in the last two years which at the same time results in many websites looking alike. While it is a great framework to quickly build a prototype of a website or an admin panel for a website, it is highly encouraged to customize Bootstrap. You can do that by either using themes from Bootswatch[1], using a template from Creative Market[2] or using other tools from this list[3].

Bootstrap theme used in this book

The examples in this book will be using a Bootstrap scheme from Bootswatch called "Superhero". You can preview it here: http://bootswatch.com/superhero/[4].

To get started with using Bootstrap in your web application (not necessarily Laravel specific) you need to either:

- Use a CDN (Content Delivery Network) like Bootstrap CDN[5] to serve the Bootstrap CSS/JS files

[1] http://bootswatch.com
[2] http://bit.ly/creativemarketBS
[3] http://bootsnipp.com/resources#8
[4] http://bootswatch.com/superhero/
[5] http://www.bootstrapcdn.com/

- Download Bootstrap CSS/JS and serve it from your application's public folder

We will use Bootstrap to build some nice looking login/registration forms a bit later in this chapter, but first let's take a look at using the two methods of serving Bootstrap CSS and Javascript.

1.1 Using a CDN to serve Bootstrap CSS/JS

Serving Bootstrap from a Content Delivery Network (CDN) could make your website perform faster. It is a good practice to serve static assets (CSS/JS files) from a CDN when you have a lot of users all over the world. Modern CDNs have servers in multipler locations around the world and so that cuts down the time that it takes to serve the CSS and Javascripts assets to the user's browser. With Bootstrap's growing popularity there are now a few CDNs that serve Bootstrap CSS and JS for **free**. The most prominent and dependable of these is Bootstrap CDN located at http://www.bootstrapcdn.com from the generous folks at MaxCDN[6].

You can use BootstrapCDN to easily add Bootstrap stylesheets and javascript into your project. By adding just two lines of code from listing 1.1 you can take advantage of using a CDN to serve Bootstrap CSS and JS to your users:

Listing 1.1 Using Bootstrap CSS and JS from CDN

```
<link href="http://netdna.bootstrapcdn.com/bootstrap/3.1.0/css/bootstrap.min.css"
      rel="stylesheet">

<script src="http://netdna.bootstrapcdn.com/bootstrap/3.1.0/js/bootstrap.min.js"></script>
```

Using a specific version of Bootstrap

Bootstrap CDN allows you to choose a specific version of Bootstrap. For the newest version be sure to go to the homepage: http://www.bootstrapcdn.com/. For the older versions check out the "legacy" tab on the Bootstrap CDN website: http://www.bootstrapcdn.com/#legacy_tab

Using a different theme for Bootstrap

Bootstrap CDN also allows you to choose a specific theme of Bootstrap from Bootswatch instead of default Bootstrap. To see the list of all themes available for use on Bootstrap CDN, check the Bootswatch tab: http://www.bootstrapcdn.com/#bootswatch_tab

Please note, Bootstrap's Javascript requires jQuery to be loaded first so if you want to use any of Bootstrap's features like modal windows, dropdowns, etc. you would have to make sure to provide jQuery before loading bootstrap.min.js file.

A page that incorporates the Bootstrap framework served from BootstrapCDN is provided in Listing 1.2 below:

[6] http://www.maxcdn.com

Listing 1.2 HTML page that uses Bootstrap from a CDN

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>View Template that use Bootstrap</title>

    <!-- Bootstrap CSS served from a CDN -->
    <link href="http://netdna.bootstrapcdn.com/bootstrap/3.1.0/css/bootstrap.min.css"
    rel="stylesheet">
  </head>

  <body>

    <div class="container">
      <div class="row">
        <h2>This template is using Bootstrap from a CDN!</h2>
      </div>
    </div>

    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.1.0/js/bootstrap.min.js">
    </script>
  </body>
</html>
```

Another option to serve Bootstrap to the user's browser instead of using a CDN is to store its stylesheets and javascript in the application's "public" directory. The public directory is the root directory that is accessible to users when they go to your website.

1.2 Using self-hosted Bootstrap CSS/JS with a back end framework

The concepts explained below could be adapted to other frameworks and not only limited to Laravel. The particular methods of each backend framework (Laravel, Symfony, Node.js, etc) are different but the concepts below apply to all MVC frameworks. To serve Bootstrap from your web application instead of using CDN you first need to download Bootstrap CSS, icon fonts and JS files, place them somewhere in the application's "public" folder and reference them in the your view templates.

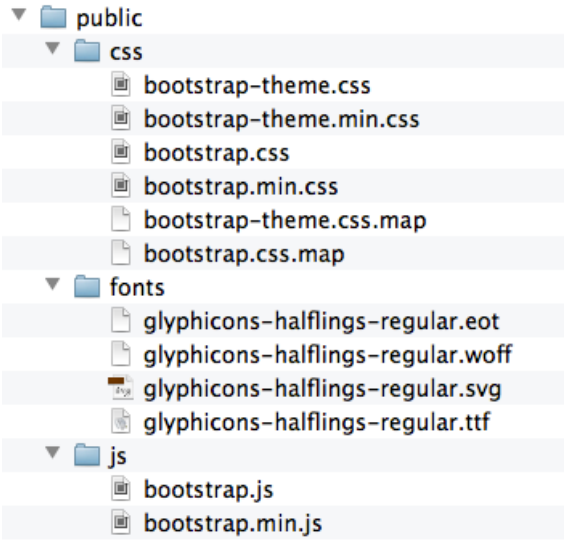
You can download Bootstrap either from http://getbootstrap.com or by using Bower[7]. When you download Bootstrap, you will find these three directories:

- css (compressed and the original css)

[7] http://bower.io/

- js (compressed and the original js)
- fonts (glyphicons, the icon font that Bootstrap uses)

Cut and paste these directories into the "public" folder of your application, so that you have the following directory structure:



Using self-hosted Bootstrap css/js/fonts

Then after you have the necessary CSS, JS and the Fonts in your application's "public" folder, you can use Bootstrap just like you would if you used a CDN. The only change that you would need would be the location of the assets. For example if you are using Laravel you'd link to the CSS and JS files by using Laravel's HTML generators in a way shown in listing 1.3 below:

Listing 1.3 Using Laravel HTML helpers to link to Bootstrap CSS and JS files

```
{{ HTML::style('css/bootstrap.min.css') }}

{{ HTML::script('js/bootstrap.min.js') }}
```

You can use these two lines of code in listing 1.3 to add Bootstrap to your Blade layouts or Blade views. For example, a complete Blade view that uses this self-hosted method of referencing to Bootstrap is provided below in listing 1.4:

Listing 1.4 Blade view that uses Bootstrap located locally.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>View Template that use Bootstrap</title>

    <!-- Referencing Bootstrap CSS that is hosted locally -->
```

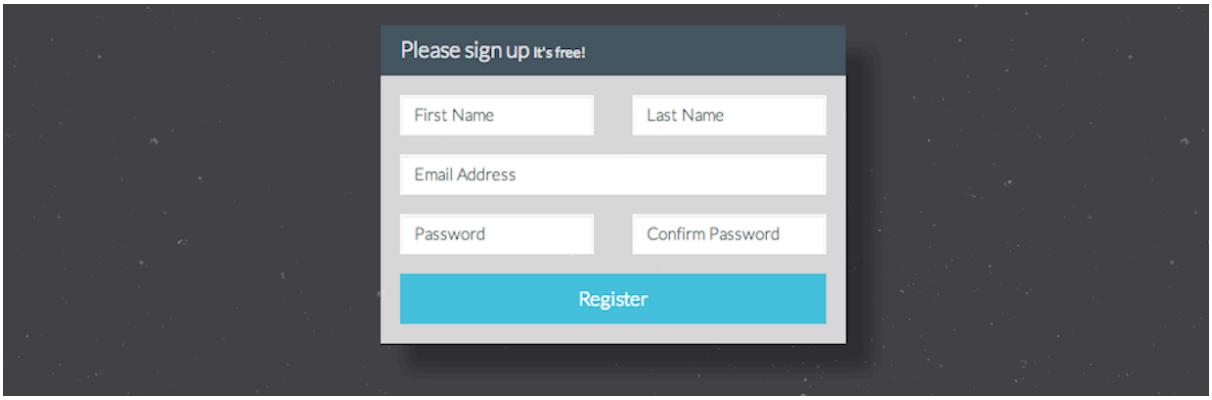
```
    {{ HTML::style('css/bootstrap.min.css') }}
  </head>

  <body>

    <div class="container">
      <div class="row">
        <h2>This template is using locally hosted Bootstrap!</h2>
      </div>
    </div>

    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>

    <!-- Referencing Bootstrap JS that is hosted locally -->
    {{ HTML::script('js/bootstrap.min.js') }}
  </body>
</html>
```

These two easy methods of referencing Bootstrap's assets (CDN and self-hosted) in Laravel allow you to use all the nifty features that come with Bootstrap: responsive layouts, good looking forms, buttons, button groups, modal windows and many many more. Let's build upon the concept you have learned so far and use Bootstrap to create some nice looking registration and login forms for our applications!

1.3 Creating HTML for the registration and login page

Every web application that asks user to register needs a nice registration page and login page. Using Bootstrap building out the pages with responsive forms is not complicated. Over the course of the next few pages you will build a registration and login forms that you then will convert to Blade templates for usage in your applications.

You will get excited when you see what you will build. Here is the resulting look of the registration page that uses Bootstrap with this book's default Bootstrap theme called "Superhero":



Registration form that you'll build

Looks good, don't you think? Let's build this registration using Bootstrap that we will serve from the Bootstrap CDN.

1.3.1 Building a registration form

First of all, we will use a nice background pattern for the login form. You can find hundreds of great looking backgrounds on Subtle Patterns[8] website. For the forms that match our "Superhero" theme the pattern called "Stardust" works very well. Download and extract that pattern from http://subtlepatterns.com/stardust/ and put the main (not @2x) file in the "img" folder inside of the "public" folder.

Getting patterns from Subtle Patterns

Subtle patterns website[9] can be incredibly helpful when you need to find a great looking repeating (seamless) pattern for your web design projects in normal and high resolution. Make sure to check out the full range of patterns that they offer and check with the site's license about using them in commercial projects.

Let's define what elements the form will be composed of. The registration form will consists of a few input fields wrapped inside Bootstrap's panel element:

- First Name (of type "text")
- Last Name ("text")
- Email ("text")
- Password (type "password")
- Password Confirmation ("password")
- Submit button (full length)

As far as sizing, the responsive registration form will be 4 columns wide on medium to large screen, 8 columns on extra small to small and 12 columns on anything less than extra small screen size. You can consult with the following Bootstrap Docs page about the different screen sizes and column naming: http://getbootstrap.com/css/#grid. To center the form on the screen we can use Bootstrap grid's offset methods such as "col-md-offset-4".

The registration form will use small input field font size (Bootstrap's "input-sm" class) so that the placeholders fit nicely inside the form. When put all together, the form's HTML will look like the listing 1.5 below:

[8] http://subtlepatterns.com/
[9] http://subtlepatterns.com/

Listing 1.5

```
<div class="row">
  <div class="col-xs-12 col-sm-8 col-md-4 col-sm-offset-2 col-md-offset-4">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Please sign up <small>It's free!</small></h3>
      </div>
      <div class="panel-body">
        <form role="form">
          <div class="row">
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="text" name="first_name" class="form-control input-sm" placeho\
lder="First Name">
              </div>
            </div>
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="text" name="last_name" class="form-control input-sm" placehol\
der="Last Name">
              </div>
            </div>
          </div>

          <div class="form-group">
            <input type="email" name="email" class="form-control input-sm" placeholder="Em\
ail Address">
          </div>

          <div class="row">
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="password" name="password" class="form-control input-sm" place\
holder="Password">
              </div>
            </div>
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="password" name="password_confirmation" class="form-control in\
put-sm" placeholder="Confirm Password">
              </div>
            </div>
          </div>

          <input type="submit" value="Register" class="btn btn-info btn-block">
```

```
        </form>
      </div>
    </div>
  </div>
</div>
```

To make the form even prettier and make it stand out, let's use CSS to add some transparency to the panel and a shadow on the right side of it. Also we would need a bit of a margin between the top of the form and top of the page. We will create a new CSS class called "centered-form" and append it to the DIV enclosing the whole form:

Listing 1.6 Addition of 'centered-form' class to the DIV enclosing the form

```
<div class="row centered-form">
  <div class="col-xs-12 col-sm-8 col-md-4 col-sm-offset-2 col-md-offset-4">
    <div class="panel panel-default">
    ...
```

Finally, create the CSS for transparency and top margin (listing 1.7):

Listing 1.7 CSS that adds top margin, transparency & shadow to the registration form

```
<style>
.centered-form .panel{
  background: rgba(255, 255, 255, 0.8);
  box-shadow: rgba(0, 0, 0, 0.3) 20px 20px 20px;
}

.centered-form{
  margin-top: 60px;
}
</style>
```

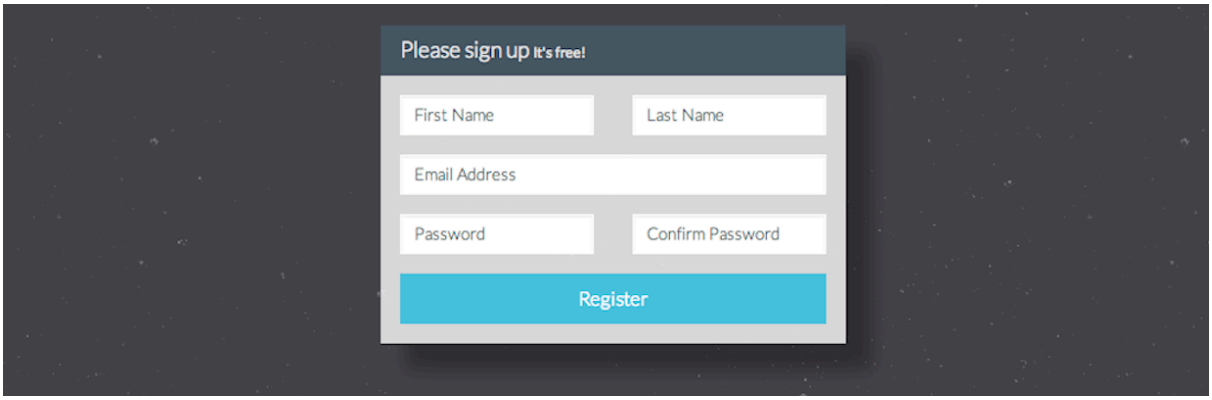
Now with this CSS in place, the full contents of the HEAD tag that enable the use of Bootstrap, add a pattern to the page and make our form a bit more better looking, is in the listing 1.8 below:

Listing 1.8 Full CSS for the registration form

```
<link href="http://netdna.bootstrapcdn.com/bootswatch/3.1.0/superhero/bootstrap.min.css"
      rel="stylesheet">
<style>
body{
  background: url("img/stardust.png");
}
.centered-form .panel{
  background: rgba(255, 255, 255, 0.8);
  box-shadow: rgba(0, 0, 0, 0.3) 20px 20px 20px;
}
```

```
.centered-form{
  margin-top: 60px;
}
</style>
```

When the HTML from listing 1.5 and CSS from listing 1.8 are put together, the registration form should look like the the one in the picture below:

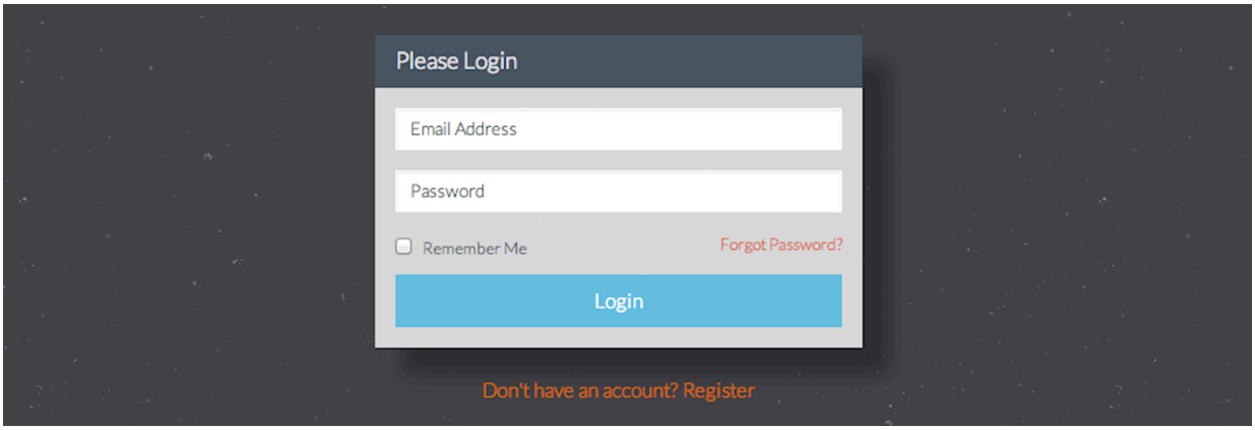


Resulting registration form

Now that we have the registration form complete, let's build a login form in a similar fashion.

1.3.2 Building a login form

The login form will now be a breeze to build, with the existing registration form HTML and CSS from the previous section. With a few minor changes in the markup of the registration form, you will have a beautiful Login form that looks like this:



Login Form that you'll build

To create this nice looking responsive form, let's define what input fields we want to have:

- Email (or username)
- Password

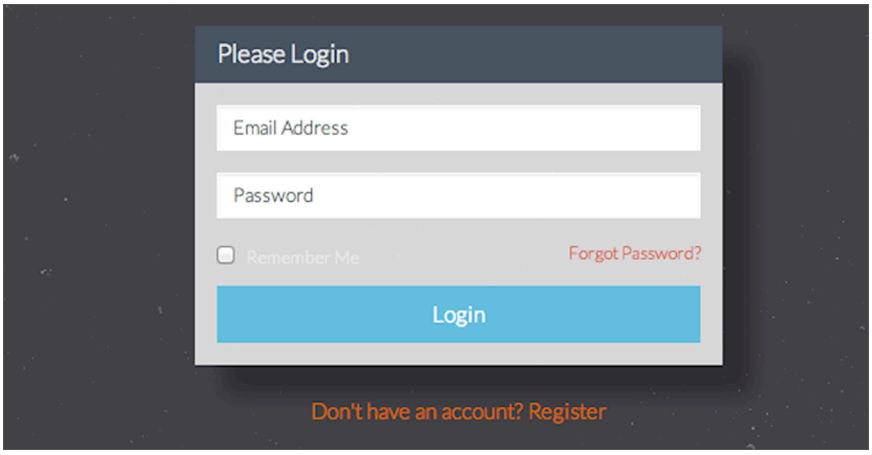
- Checkbox for “Remember me” feature
- Submit button

Also, we would like the user to be able to go to the registration form if they don’t have an account and a link to “Forgot password” feature if they don’t remember their password. With these input fields in mind, the complete HTML for the login form will look like the code in listing 1.9:

Listing 1.9 HTML of the login form

```
<div class="row centered-form">
  <div class="col-xs-12 col-sm-8 col-md-4 col-sm-offset-2 col-md-offset-4">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Please Login</h3>
      </div>
      <div class="panel-body">
        <form role="form">

          <div class="form-group">
            <input type="email" name="email" class="form-control input-sm" placeholder="Em\
ail Address">
          </div>

          <div class="form-group">
            <input type="password" name="password" class="form-control input-sm" placehold\
er="Password">
          </div>

          <div class="checkbox">
            <label>
              <input name="remember" type="checkbox" value="Remember Me"> Remember Me
              <a href="/forgot" class="pull-right">Forgot Password?</a>
            </label>
          </div>

          <input type="submit" value="Login" class="btn btn-info btn-block">

        </form>
      </div>
    </div>
    <div class="text-center">
      <a href="/register" >Don't have an account? Register</a>
    </div>
  </div>
</div>
```

When this HTML markup is rendered in the browser, the login form will look like the following image:



The look of the Login Form based on HTML from listing 1.9

Notice that the "Remember me" text is not clearly visible. Let's fix that by adding a line of "color" CSS for ".centered-form .panel" that will change the color of the text and the labels on the login form:

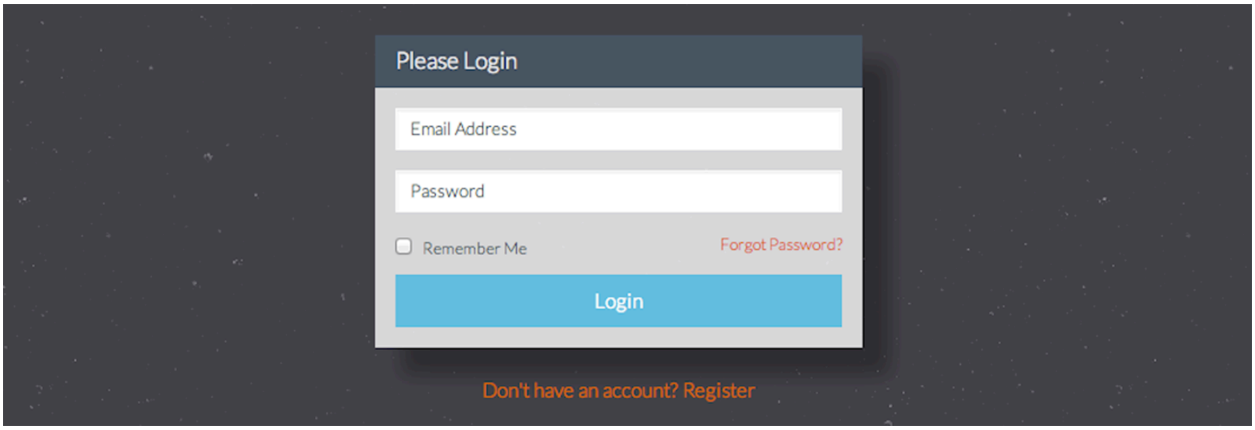
```
.centered-form .panel{
  background: rgba(255, 255, 255, 0.8);
  box-shadow: rgba(0, 0, 0, 0.3) 20px 20px 20px;
  color: #4e5d6c;
}
```

When the css is updated with that extra line, the HEAD tag of the page where this form resides will look like the listing 1.10 below:

Listing 1.10 Full CSS for the login form

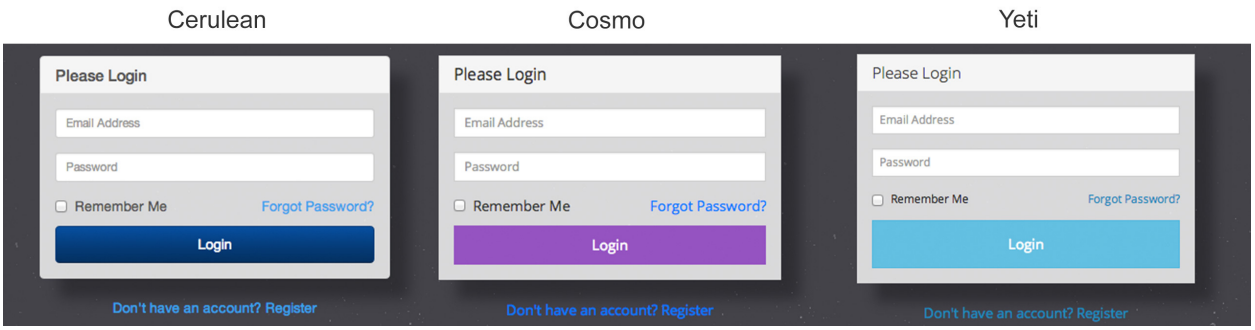
```
<link href="http://netdna.bootstrapcdn.com/bootswatch/3.1.0/superhero/bootstrap.min.css"
      rel="stylesheet">
<style>
body{
  background: url("img/stardust.png");
}
.centered-form .panel{
  background: rgba(255, 255, 255, 0.8);
  box-shadow: rgba(0, 0, 0, 0.3) 20px 20px 20px;
  color: #4e5d6c;
}
.centered-form{
  margin-top: 60px;
}
</style>
```

With that CSS in place, the login form is now complete and will render with the label text colored correctly, producing this beautiful screen that invites the user to log in into your awesome application:

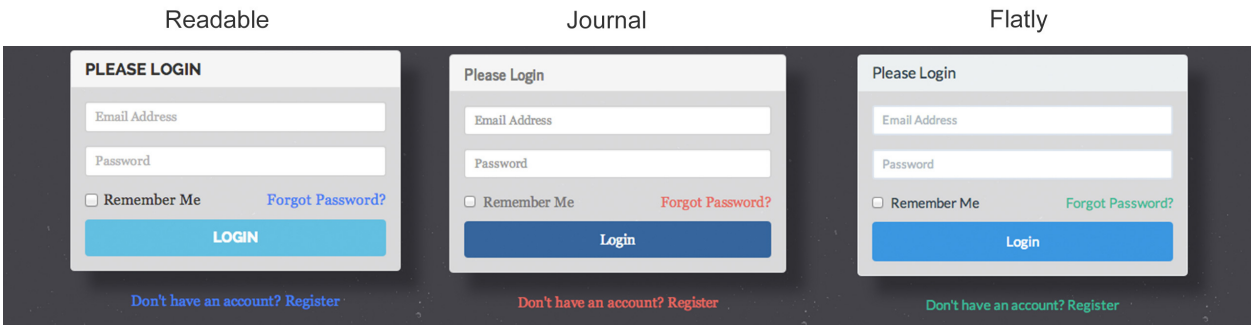


Login Form resulting from HTML in listing 1.9 and CSS in listing 1.10

Later in this chapter we will convert the registration form and the login form into Blade templates that you could easily reuse in Laravel. What if you don't like the color scheme that we are using for the projects in the book ("Superhero" theme)? Don't fear, there are so many other cool Bootstrap schemes available on Bootstrap CDN that you can use to change the look of the interface. For example, here is the same login form using Cerulean, Cosmo or Yeti styles:



And the same login form again, but using Readable, Journal or Flatly scheme:



If you wanted to use any of those different schemes for Bootstrap, all you would have to do is replace a single line in your HEAD tag requesting a different Bootstrap scheme from Bootstrap CDN (in this case, "Yeti" scheme that looks a bit like Foundation framework):

Using a different Bootstrap scheme for the whole page

```
<link href="http://netdna.bootstrapcdn.com/bootswatch/3.1.0/yeti/bootstrap.min.css"
                        rel="stylesheet">
```

Again, as mentioned before, you can take a look at different schemes provided by Bootstrap CDN at http://www.bootstrapcdn.com/#bootswatch_tab. There are plenty of choices but if you still don't like what you see there, it is encouraged to customize Bootstrap to your own liking.

1.4 Converting the registration and login forms to Laravel Blade templates

What do you need to do in order to convert the registration and login forms to templates compatible with Laravel or other backend framework? First, you will want to create a layout, like Blade layout[10] to store the HTML that is common to more than one page of your application. Then you will convert the form input elements to your framework's form methods. Laravel has "Form" methods (http://laravel.com/docs/html) that make that process simple. Last step is to add some highlighting for the errors that might arise when the validation of the forms doesn't pass after submittal.

If you are using any backend framework, creating HTML layout template helps you separate the code that is common to two or more pages of your application. Laravel's Blade templating makes it easy to then use that layout in other pages that will apply the layout as their "skin". Using a common layout is incredibly beneficial when you want to make updates to all pages without going through all of the view files.

Let's start the form conversion process by creating a layout template that will store the common HTML/CSS of both, registration and login forms. From the two HTML pages we can identify that the following areas of both forms have the same code:

Listing 1.11 Layout template starter

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Using a Blade layout</title>

    <!-- Bootstrap CSS served from a CDN -->
    <link
      href="http://netdna.bootstrapcdn.com/bootswatch/3.1.0/superhero/bootstrap.min.css"
       rel="stylesheet">

    <style>
    body{
      background: url("img/stardust.png");
    }
```

[10] http://laravel.com/docs/templates#blade-templating

```
    .centered-form .panel{
      background: rgba(255, 255, 255, 0.8);
      box-shadow: rgba(0, 0, 0, 0.3) 20px 20px 20px;
      color: #4e5d6c;
    }
    .centered-form{
      margin-top: 60px;
    }
    </style>
  </head>

  <body>

    <div class="container">
      ...
    </div>

    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.1.0/js/bootstrap.min.js">
    </script>
  </body>
</html>
```

The section of the code with "..." shows a placeholder for the forms themselves. In Laravel, to use a layout placeholder you use "@yield('someSection')" to specify a part of the layout that will be injected with content placed in a Blade template that is rendered. Let's put a placeholder for "content" section and save the code listed below in a file "layout.blade.php" in the "app/views" folder:

Listing 1.12 Contents of app/views/layout.blade.php

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Using a Blade layout</title>

    <!-- Bootstrap CSS served from a CDN -->
    <link
      href="http://netdna.bootstrapcdn.com/bootswatch/3.1.0/superhero/bootstrap.min.css"
            rel="stylesheet">
    <style>
    body{
      background: url("img/stardust.png");
    }
    .centered-form .panel{
```

```
      background: rgba(255, 255, 255, 0.8);
      box-shadow: rgba(0, 0, 0, 0.3) 20px 20px 20px;
      color: #4e5d6c;
    }
    .centered-form{
      margin-top: 60px;
    }
    </style>
  </head>

  <body>

    <div class="container">
      @yield('content')
    </div>

    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.1.0/js/bootstrap.min.js">
    </script>
  </body>
</html>
```

With this layout template in place, we can easily create the templates for registration and login forms without repeating this code over and over again. Let's create a Blade template for the registration form and save it as "registration.blade.php" file in "app/views" folder. To apply the layout template we created earlier, we will just use "@extends('layout')" statement and specify which part of the template will be acting as "content" that will be inserted in place of "@yield('content')":

Listing 1.13 Contents of app/views/registration.blade.php

```
@extends('layout')

@section('content')

<div class="row">
  <div class="col-xs-12 col-sm-8 col-md-4 col-sm-offset-2 col-md-offset-4">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Please sign up <small>It's free!</small></h3>
      </div>
      <div class="panel-body">
        <form role="form">
          <div class="row">
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="text" name="first_name" class="form-control input-sm" placeho\
lder="First Name">
```

```
            </div>
            </div>
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="text" name="last_name" class="form-control input-sm" placehol\
der="Last Name">
              </div>
            </div>
          </div>

          <div class="form-group">
            <input type="email" name="email" class="form-control input-sm" placeholder="Em\
ail Address">
          </div>

          <div class="row">
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="password" name="password" class="form-control input-sm" place\
holder="Password">
              </div>
            </div>
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                <input type="password" name="password_confirmation" class="form-control in\
put-sm" placeholder="Confirm Password">
              </div>
            </div>
          </div>

          <input type="submit" value="Register" class="btn btn-info btn-block">

        </form>
      </div>
    </div>
  </div>
</div>

@stop
```

Great! Now to see this in the browser, you will need to define a route that will render the registration page template applying the layout. As a basic example, let's define the following route in the "app/routes.php" file:

Listing 1.14 Route that renders the registration form

```
Route::get('register', function()
{
  return View::make('registration');
});
```

With this route in place, the registration page will be rendered when the user goes to "/register" URL relative to the application.

While the form is displaying correctly in the browser, it doesn't have proper methods that integrate with Laravel's Input/Validation. Let's fix that by converting form elements to Laravel's Form methods. To begin, we will first convert the "form" tag to use "Form::open()" / "Form::close()". Then we will do a replacement of all input tags with Laravel's corresponding form methods:

- <input type="text"> tag with "Form::text()"
- <input type="email"> tag with "Form::email()"
- <input type="password"> tag with "Form::password()"
- <input type="checkbox"> tag with "Form::checkbox()"
- <input type="submit"> tag with "Form::submit()"

Listing 1.15 shows the Blade file for the registration form (app/views/registration.blade.php) with the input methods replaced with Laravel's form generators:

Listing 1.15 Registration form using Laravel's form methods

```
@extends('layout')

@section('content')

<div class="row centered-form">
  <div class="col-xs-12 col-sm-8 col-md-4 col-sm-offset-2 col-md-offset-4">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Please sign up <small>It's free!</small></h3>
      </div>
      <div class="panel-body">
        {{ Form::open() }}
          <div class="row">
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                {{ Form::text('first_name', null, array('class'=>'form-control input-sm','\
placeholder'=>'First Name')) }}
              </div>
            </div>
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                {{ Form::text('last_name', null, array('class'=>'form-control input-sm','p\
```

```
laceholder'=>'Last Name')) }}
              </div>
            </div>
          </div>

          <div class="form-group">
            {{ Form::email('email', null, array('class'=>'form-control input-sm','placehol\
der'=>'Email Address')) }}
          </div>

          <div class="row">
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                {{ Form::password('password', array('class'=>'form-control input-sm','plac\
eholder'=>'Password')) }}
              </div>
            </div>
            <div class="col-xs-6 col-sm-6 col-md-6">
              <div class="form-group">
                {{ Form::password('password_confirmation', array('class'=>'form-control in\
put-sm','placeholder'=>'Confirm Password')) }}
              </div>
            </div>
          </div>

          {{ Form::submit('Register', array('class'=>'btn btn-info btn-block')) }}

        {{ Form::close() }}
      </div>
    </div>
  </div>
</div>

@stop
```

When the plain HTML input methods are replaced with Laravel's form methods, the rendered form by default will automatically have it's submission method assigned to type "POST" and the URL of the action will match the URL of the currently displayed page (in this case "/register" relative to the application URL). Also, with this change, the inputs will be working correctly with Laravels Input methods and validation.

Let's say you had a route that would process the registration form, executed upon POSTing to the "/register" URI. You would have some validation rules defined for each of the form's fields and you would use Laravel's "Validator::make" to check that the input matches the validation rules. If the validation fails, take the user back to the registration form and pass the validation errors through the session under "errors" session variable, otherwise tell the user that the form was validated (and create the user in the DB, etc). The code in listing 1.16 defines such a route with validation:

Listing 1.16 Validation for the registration form

```
Route::post('register', function()
{
  $rules = [
    'first_name'  =>  'required',
    'last_name'   =>  'required',
    'email'       =>  'required|email',
    'password'    =>  'required|confirmed'
  ];

  $validator = Validator::make(Input::all(), $rules);

  if ($validator->fails())
  {
      return Redirect::to('register')->withInput()->withErrors($validator);
  }

  return 'Form passed validation!';
});
```

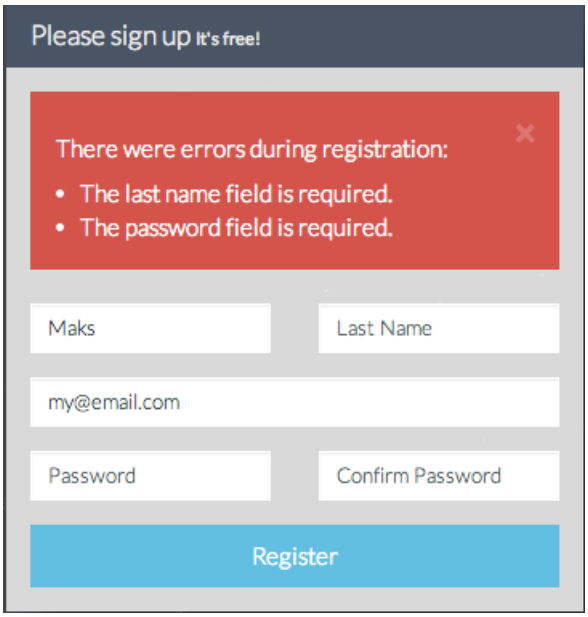
Laravel provides you with full-stack solution for handling rendering of the form, its submittal, remembering the old values and highlighting the fields that are incorrectly filled out by the user. For example if you wanted to show a list of validation errors when the user submits incorrect form values, you could use the "errors" variable that Laravel's validator sets in the session after the form submittal. Insert the following Blade code right above the form's opening tag to display a list of all validation errors that occur during form submission:

Listing 1.17 Outputting validation errors in the registration form

```
...
@if(Session::get('errors'))
  <div class="alert alert-danger alert-dismissable">
    <button type="button" class="close" data-dismiss="alert" aria-hidden="true">&times;</b\
utton>
    <h5>There were errors during registration:</h5>
    @foreach($errors->all('<li>:message</li>') as $message)
      {{$message}}
    @endforeach
  </div>
@endif

{{ Form::open() }}
...
```

With this addition to the registration form and the addition from listing 1.17 to the routing file, upon incorrect input entry and form submittal the registration page would show the validation errors wrapped in

Bootstrap's "alert" class. Each error that occurred would be listed and previous input values would be persisted for the user:



Listing validation errors in the form

This completes the conversion of the registration form HTML to a template compatible with Laravel's Blade, with all benefits that Blade brings. This form in not perfect but it is enough to get the user to register and show what errors occur on registration. The login form needs to be converted in a similar way. Now that you know the process, you can convert the login form by yourself. In fact it has less fields so the resulting Blade template would be much shorter.

Converting the login form

As an exercise, please try converting the login form to Laravel's Blade syntax by yourself.

1.5 Helpful Resources and Packages

The community around Bootstrap framework is amazingly big and creative. With framework's growing popularity many tools and helpful utilities start to appear on the scene. There are many extensions of Bootstrap, such as:

- Websites loaded with Bootstrap snippets (http://bootsnipp.com and http://bootply.com)
- Date pickers
- Theme builders
- Drag and drop editors like http://divshot.com
- Button builder/Form Builder (http://bootsnipp.com/buttons and http://bootsnipp.com/forms)
- Ready to go templates, such as the ones on Creative Market[11]

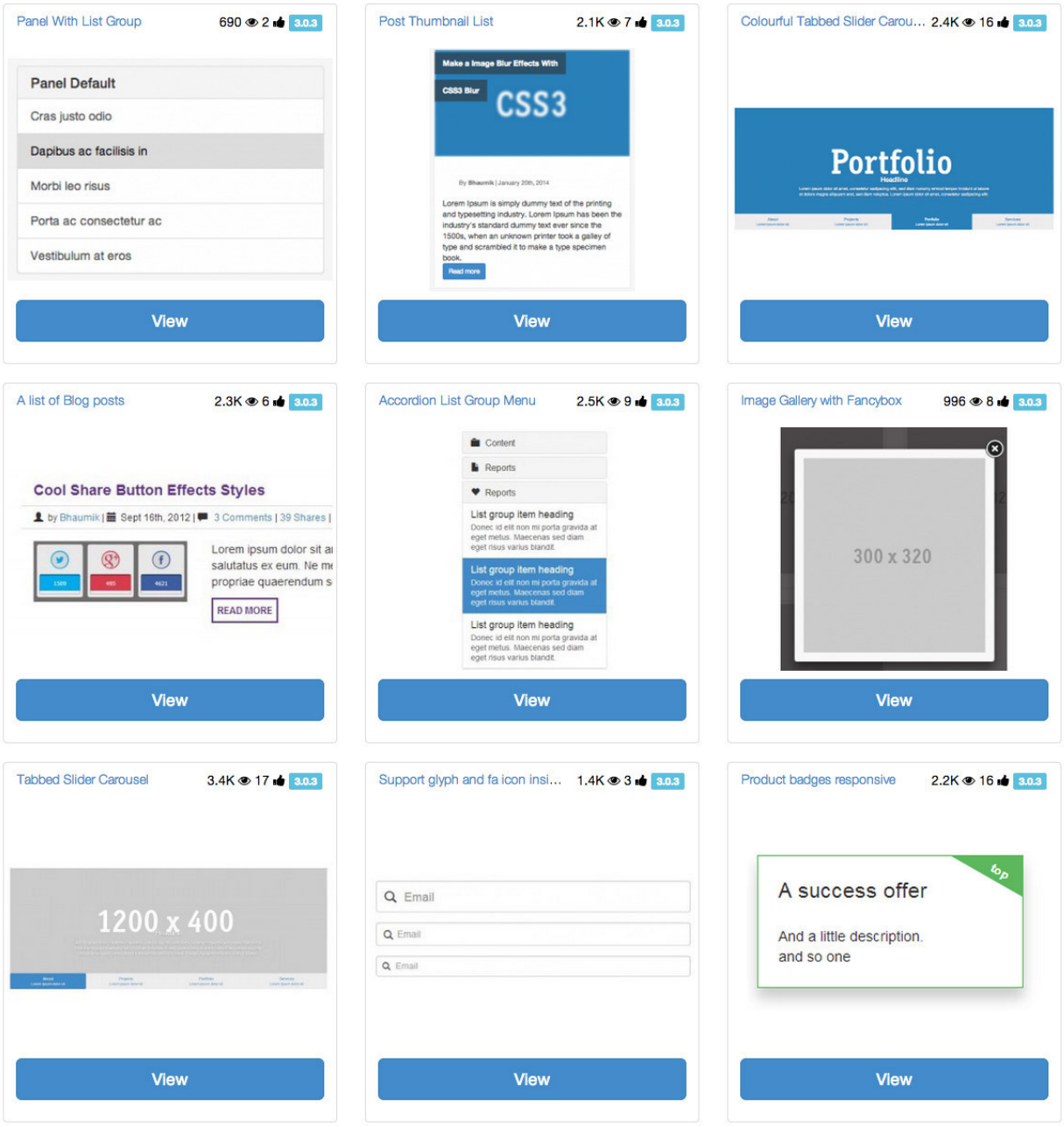
[11] http://bit.ly/creativemarketBS

Some of these resources will be discussed in this book guiding you through the projects of the Bootstrap community. Let's take a look at Bootsnipp and then learn about helpful PHP packages that make creation of Bootstrap compatible code easier.

1.5.1 Using Bootsnipp to get awesome Bootstrap code

You might have heard of a website called Bootsnipp[12]. This website provides web developers with hundreds of Bootstrap code snippets ready to be copy/pasted to your application. Login forms, lists, panels, navigation and menu examples, sidebars, Bootstrap plugins, all of these could be found on http://bootsnipp.com and reused in your projects (featured snippets are MIT license). Here is a small example of what kind of snippets you might find on Bootsnipp:

[12] http://bootsnipp.com



Bootsnipp

To use any of the snippets on http://bootsnipp.com just copy and paste snippet's HTML and CSS into your template to get the snippet to work the same way as on Bootsnipp preview tab. Simple as that! Enjoy using it to find some great looking Bootstrap code snippets!

1.5.2 More resources on integrating Bootstrap and Laravel

- A starter site made with Laravel and Bootstrap : https://github.com/andrew13/Laravel-4-Bootstrap-

Starter-Site

- Former, a package to make building forms easier (http://anahkiasen.github.io/former/)
- A blog post[13] from Elena Kolevska on using Grunt, Bower, Laravel and Bootstrap together

1.6 Summary

By now you have a good idea on how to download Bootstrap or use Bootstrap CDN to serve Bootstrap HTML/CSS/JS framework to your users. You learned how to create an awesome looking registration and login forms by using the CDN, changing the look of those forms in an instant and making them responsive. Then you have learned the process of converting Bootstrap forms into forms compatible with Laravel's Blade templating engine. Finally, you have looked at various resources that introduced you to the growing Bootstrap community.

In the next chapter you will learn how to implement a component that is necessary to give feedback about the status of user's actions, especially for AJAX actions, a spinner! Ready?

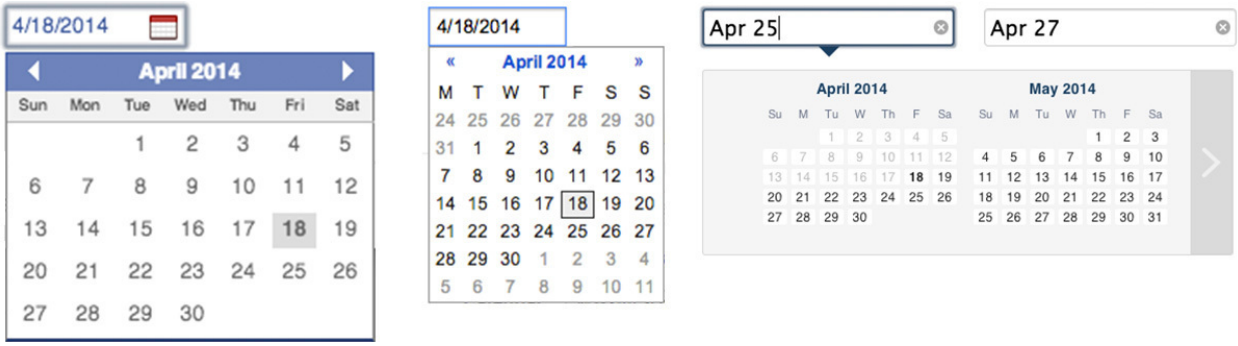
[13] http://blog.elenakolevska.com/using-grunt-with-laravel-and-bootstrap/

2 Integrating Date Pickers

This chapter covers:

- Theory behind date pickers
- Practical examples of date pickers in action
- Using Pickadate open source library for a date picker
- Integrating date pickers with the back end

Date pickers play an important part of many modern web applications. Choosing the period of hotel stay, booking flights, specifying the length of events, reserving rental cars, selecting the range of admin reports, all of these are examples of website functionality that requires the user to choose a certain date or a range of dates.



Date pickers used on popular websites

If the website is used by only one person - there is probably no need to integrate a date picker, just let the user type the date in an input field like this:

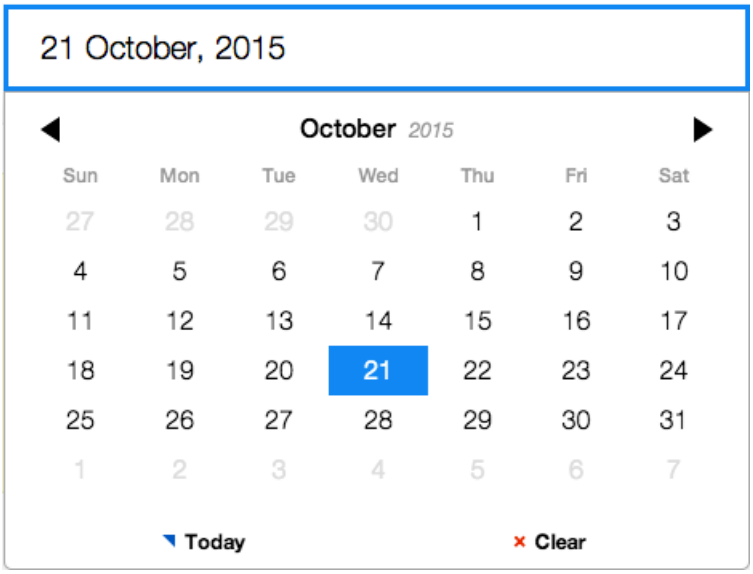
Simplest date picker - text input field

However, usually websites are used by more than one person visiting from more than one country and a need for a bit more complex in nature yet easy to use solution arises. (drum roll please…) The solution is to use date pickers. Date pickers make it painless and simple to navigate through a calendar to choose a date that the user has in mind. When the user is presented with choosing a date on a website, he or she will have the following questions that a date picker should be able to answer:

- In what format should I enter the date? (taking into account the local preferences of the user)
- Is next Friday the twelfth or the thirteenth? (visual representation of the calendar)
- What is today's date? (clear hint of current date)

A good date picker front end solution offers compatibility with a wide range of languages, date formats, screen sizes and browsers. This chapter will focus on integrating one of the best open source plugins solving

the problem of integrating a date picker, called Pickadate[1]. When integrated with a web application, it looks like this, clean and minimal:



Full featured date picker with Pickadate plugin

You'll gradually learn about date pickers and about integrating them in your applications as you read through this chapter. First you'll learn about the basic theory behind date pickers, then you'll be introduced to some existing open source projects providing date picker functionality, specifically you'll meet Pickadate plugin in its full glory. Finally, you'll integrate a date picker with a back end framework in a small application. Let's get started by understanding the theory behind date pickers.

View application in action

Fully functional demo application that you will build throughout this chapter is available at this URL: http://books.maxoffsky.com/frontend/ch8

2.1 Theory behind date pickers

Most date pickers store user's choice in a hidden input field. The reason for that is so that when the form containing this input is submitted to the server it would send the chosen date in a format that the back end would be able to understand. Consider asking the user to choose any date of the year. If instead of using a date picker the user would be required to type the answer in a text input field, some of the possible entries would be:

- April 18, 2014 (US format)
- 9 June, 2014 (European format)
- 04/18/2014 (US format)
- 18/04/2014 (European format)

[1] http://amsul.ca/pickadate.js/index.htm

- Next Monday (my favorite, lazy person format)
- A month from now ("I'm not Rainman[2], I don't keep a calendar in my head!!!")
- Today ("I am unemployed and have no idea what date it is")

As you can see, there are many different formats that users around the world might use to choose a date. Without using a date picker this inconsistency could be disastrous when the form is submitted and the back end application tries to process the input. To alleviate this problem, most date pickers usually consist of two parts, an interface and a hidden input field. When the user interacts with the interface (opens a date picker, chooses a date, etc.), the hidden input field will contain the choice in a consistent format that could be understood by the back end. Figure 8.1 illustrates the interaction of the two components of a date picker, the interface and the hidden input field:

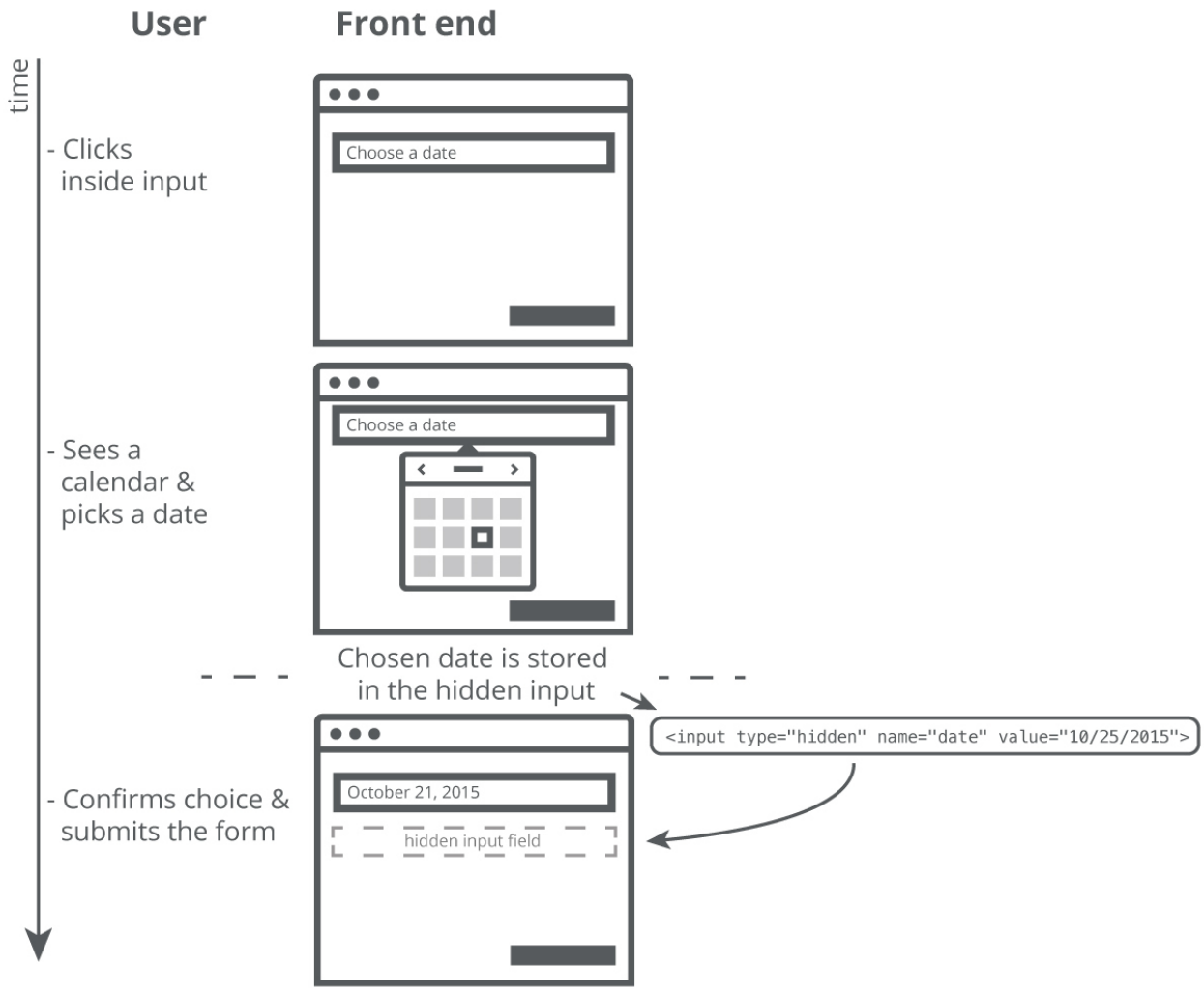


Figure 8.1 Interaction of the interface and the hidden input field

It will be important to double check that the value in the hidden input field is indeed a date in the desired format. When the form containing the hidden input field is submitted to the back end, the date input should

[2] http://en.wikipedia.org/wiki/Kim_Peek#Rain_Man

be parsed to make sure that the user didn't enter any malicious input and then the parsed/verified value could be stored in the database for further operations. Not having this verification could lead to security problems.

Can you make a date picker without hidden input field?

Some date pickers store the value in a text input field instead of using a hidden input field. While it is possible to do this, it is not as flexible and it is harder to constrain the format of user's input.

There are many existing open source date picker libraries that make integrating a date picker into a straight forward process. In this chapter we will look at using just one library called "Pickadate", but using any other library will follow similar concepts so you could easily pick up another one if for some reason Pickadate does not satisfy your requirements. Here is a list of some of the best, well-designed and well-built date picker libraries:

- jQuery UI datepicker[3]
- Bootstrap datepicker[4] by Andrew Rowls
- Zebra Datepicker[5]
- Datepicker[6] by Stefan Petre
- Pickadate[7] by Amsul

When choosing a date picker library it is important that it works with many different date formats, supports many languages, is easy to skin, and works with many screen sizes (responsive). Pickadate library was chosen for this book because it satisfies all of these requirements while staying very small in size.

Decode date pickers on the web

For the next week try to inspect date pickers on various websites by launching Chrome Developer Tools and looking at the HTML markup behind the date picker interface. Find the hidden input field that is affected when you choose different date and check the value it contains after your interaction with the date picker.

Now that you know the theory behind date pickers and know where to find great libraries providing this functionality, let's start integrating a date picker in your web application.

2.2 Integrating Pickadate library

Pickadate is a very compact yet powerful date picker library. It weights only 8Kb in total including one of the two skins provided and has support for 38 languages. What makes this library different from the rest open source date pickers is its modern look (adapting to any screen size), great modular architecture and incredible user experience. It is hard to underestimate the importance of responsive design because so many websites need to be mobile-friendly and Pickadate gracefully fills this void present in other solutions. Enough talking, let's start doing!

[3] http://jqueryui.com/datepicker/
[4] https://github.com/eternicode/bootstrap-datepicker
[5] http://stefangabos.ro/jquery/zebra-datepicker/
[6] http://www.eyecon.ro/datepicker/#about
[7] http://amsul.ca/pickadate.js/date.htm

Due to its modular design, Pickadate library is broken up in a few Javascript components, a picker core (**picker.js**), a date picker (**picker.date.js**) and a time picker (**picker.time.js**). To integrate the date picker functionality of Pickadate library we need to include its core script - **picker.js** (also called **base** script in this library), its date picker script - **picker.date.js** and one of the provided themes.

The themes are also broken up in a few parts. There is a base CSS file for each theme, a date picker CSS file and time picker CSS file matching the theme. When the library is loaded on the page we will transform a simple input field into a great-looking date picker. You can download this library from one of the releases on Github: https://github.com/amsul/pickadate.js/releases

Whoa, time picker too?

While Pickadate library provides both, a time picker and a date picker, we will only use the date picker functionality in this chapter. When you are comfortable integrating the date picker integrating its time picker component will be trivial because these components share a lot of similar concepts.

2.2.1 Including Pickadate and its themes in your page

After the library is downloaded[8], let's include it by loading it's Javascript and one of the included themes. At the time of this writing there are two themes provided with the library: default and classic. Default theme uses a full-screen popup to show the date picker while the classic theme shows the date picker in a drop down menu that slides from under the input field. We will use the **classic** theme for this chapter. The image in figure 8.2 shows the visual difference between the two themes:

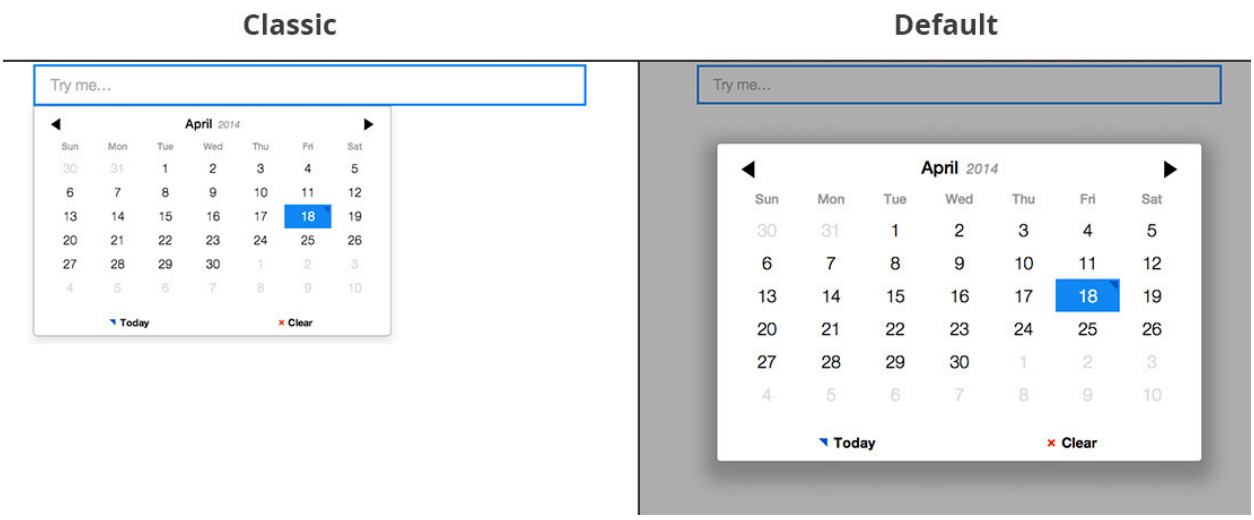


Figure 8.2 Themes included with Pickadate

Let's include the compressed assets of Pickadate library. Include its CSS in the HEAD tag of your page and its JS before the closing BODY tag. Code in listing 8.1 shows the library's assets included on a page and ready for use.

[8] https://github.com/amsul/pickadate.js/releases

Listing 8.1 Including Pickadate library

```
<!DOCTYPE html>
<html>
  <head>
    ...
    <!-- Include library's base theme and date picker theme files -->
    <link href="vendor/pickadate/compressed/themes/classic.css" rel="stylesheet">
    <link href="vendor/pickadate/compressed/themes/classic.date.css" rel="stylesheet">
    ...
  </head>
  <body>
    ...
    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
    <!-- Include library's JS files -->
    <script src="vendor/pickadate/compressed/picker.js"></script>
    <script src="vendor/pickadate/compressed/picker.date.js"></script>
  </body>
</html>
```

Dependencies and browser support of Pickadate

Pickadate plugin requires jQuery to be present on your page in order to work. The plugin works with all modern browsers - and even IE8+. For IE8 support include **legacy.js** file that comes with the library.

Now that the library and one of its themes are included, we can use it on an input field and explore the different options available.

2.2.2 Enabling Pickadate

Let's enable Pickadate on an input field that will serve as a date picker. Imagine that you have a form containing many different input fields and one of the necessary fields in the form is a text input field that should allow the user to choose a date on a calendar. Enabling Pickadate on this text field will do two things: create the date picker interface (initially hidden) and attach events that control the interface when the user clicks inside of the input field. Code in listing 8.2 below shows how to enable Pickadate date picker on a text input field inside of a form:

Listing 8.2 Enabling Pickadate on an input field

```
...
<form>
  ...
  <input id="date" name="date" type="text" placeholder="Choose a date">

  <input type="submit" value="Submit" class="btn btn-info btn-block">
</form>
...

<!-- After including the date picker scripts -->
<script>
  $(function() {
    // Enable Pickadate on an input field
    $('#date').pickadate();
  });
</script>
```

When Pickadate is enabled on the input field, you can see the date picker in action by clicking inside of that input field. Screenshot in figure 8.3 below shows the effect of enabling Pickadate on the field:

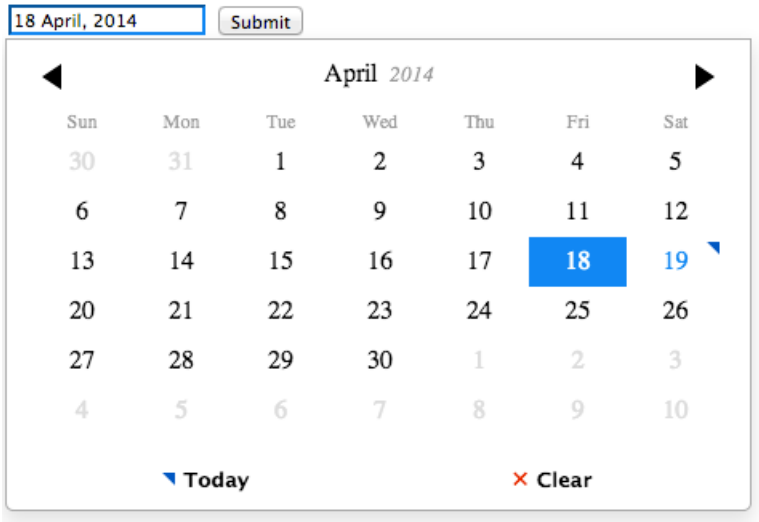


Figure 8.3 Pickadate enabled on an input field

There's something interesting about invoking Pickadate this way, without any additional parameters. No hidden input will be created, but instead the chosen date will be stored as the actual value of the input field. Because the format of the chosen date (e.g. "18 April, 2014") is standard enough to be submitted to the server there is no need to duplicate the input fields. The need for the hidden input field comes when you want the displayed date format to be different from the format of the value that is submitted to application's back end. Let's take a look at changing the date format of the submitted value.

2.3 Changing the date format

Most often, the date format of the submitted date picker value will be dictated by the back end requirements depending on what needs to be done with the chosen date. By default, the format of the date picker is set to 'd mmmm, yyyy', where 'd' is the date of the month (1-31), 'mmmm' is the full name of the month (e.g. January) and 'yyyy' is the year in full form (e.g. 2014). If you want to change the format of the value that will be submitted to the server you can pass that as '`formatSubmit`' option to `pickadate()`. When this option is used, a hidden input field is created and it's name is set to the name of the initial input field plus suffix of '_submit'. Code in listing 8.3 shows how instead of default date format you can set a format of 'yyyy/mm/dd' on the hidden input field:

Listing 8.3 Setting the date format of hidden input field

```
$(function() {
  // Enable Pickadate on an input field and
  // specifying date format for hidden input field
  $('#date').pickadate({
    formatSubmit : 'yyyy/mm/dd'
  });
});
```

When the form is submitted, it will now contain two input fields, one named 'date' and another one (hidden): 'date_submit'. The format of the 'date' field will be the same as it was before, 'd mmmm, yyyy', but the format of 'date_submit' hidden input field will be 'yyyy/mm/dd', e.g '2014/04/18'. If you want to format the hidden input field differently, use the table 8.1 below (from Pickadate documentation) to understand how to format different components of the date: days, months and years:

Table 8.1 Formatting rules of Pickadate

Rule	Description	Range
d	Date of the month	1-31
dd	Date of the month with a leading zero	01-31
ddd	Day of the week in short form	Sun - Sat
dddd	Day of the week in full form	Sunday - Saturday
m	Month of the year	1 - 12
mm	Month of the year with a leading zero	01 - 12
mmm	Month name in short form	Jan - Dec
mmmm	Month name in full form	January - December
yy	Year in short form	00 - 99
yyyy	Year in full form	2000 - 2999

For example setting 'formatSubmit' option to MySQL datetime format would look like this:

```
// Setting format of the submitted value to conform to MySQL datetime format
formatSubmit : 'yyyy-mm-dd 00:00:00'
```

Now that you know how to change the date format of the hidden input field submitted to the back end, let's take a look at changing the date on the front end.

2.3.1 Changing the date format of displayed value

To specify the format of the value that is displayed to the user after he or she chooses a date you can set the 'format' option to a desired date format. For example if you wanted to say "You chose Oct-25-2015" after the user chose a date, you would do it by passing 'You chose mmm-dd-yyyy' as the format option as you can see from code in listing 8.4:

Listing 8.4 Customizing format of displayed date

```
format : 'You chose mmm-dd-yyyy'
```

Escaping 'd','m' and 'y' characters

If you need to use 'd', 'm', 'y' characters in the format string other than for the date format, you need to escape them by prepending an exclamation mark like this: 'The !date is mm-dd-yyyy'. If you just put 'The date is mm-dd-yyyy' you will see something like 'The 25ate is 10-25-2015' which is probably not what you intend.

Now that you know how to set formats for both, the visible and the hidden input fields, let's explore another area of Pickadate customization, changing the names of the input fields that will be sent to the server when the form containing the date picker is submitted.

2.4 Modifying input fields

There are a many options that Pickadate provides when it comes to naming the input fields that will be sent to the server. You can set an option to send only the hidden field (instead of the input field + hidden field), otherwise you can give the hidden field a custom name using a suffix and a prefix options. Let's take a look at submitting just the hidden input first and then let's learn how to customize hidden input name if it is sent as an addition to the actual input field.

2.4.1 Making hidden input the only value submitted to the server

To send only the value contained in the hidden input you can set 'hiddenName' to true. Doing this will basically replace the original text input with the hidden input and will give the name of the original input to the hidden input. Code in listing 8.5 below shows how to enable this option along with custom format for the hidden field and the effect that it has on the HTML of the date picker:

Listing 8.5 Substituting real input field with the hidden field

```
$(function() {
  $('#date').pickadate({
    formatSubmit : 'yyyy-mm-dd 00:00:00',
    hiddenName : true
  });
});

// The HTML of the input fields when the form is submitted will contain:
<input id="date" name="" type="text" placeholder="Choose a date">
<input type="hidden" name="date" id="date_hidden" value="2015-10-25 00:00:00">
```

Use this by default

In my opinion, submitting only the formatted value is necessary for a date picker since what is displayed is not necessarily easy to parse by the server (especially when it comes to localization issues). Using these two options in listing 8.5 is a good start for integrating Pickadate.

2.4.2 Renaming the hidden input field

In those cases when you need to submit both values, the displayed and the hidden value, you can customize the name of the hidden input field by using '`hiddenPrefix`' and '`hiddenSuffix`' options. The '`hiddenPrefix`' option sets a text string that will be prepended to the name of the hidden input. The '`hiddenSuffix`' option sets a text string that will be appended to the name of the hidden input, as specified above this option is set to '_submit' by default. For example if you wanted to have the name of the hidden input be "date_myfield", you would name the original input field "date" and add the `hiddenSuffix` option with a value of '_myfield', like it is shown in listing 8.6 below:

Listing 8.6 Changing the name of the hidden input field

```
$(function() {
  $('#date').pickadate({
    formatSubmit : 'yyyy-mm-dd 00:00:00',
    hiddenSuffix : '_myfield'
  });
});

// The HTML of the input fields when the form is submitted will contain:
<input id="date" name="" type="text" placeholder="Choose a date">
<input type="hidden" name="date_myfield" id="date_myfield" value="2015-10-25 00:00:00">
```

Great! You now are able to control the names and formats of the input fields that will be submitted with your form containing the date picker. We'll see these concepts in action later on when we build the integration of a date picker with the back end so keep them as a reference. For now let's take a look at setting a default value in the date picker upon page load.

2.5 Setting default value of the date picker

Setting the default value of the date picker could be done in a few ways depending on the formatting of the submitted date. If the date that will be displayed is the same as the date that will be submitted, nothing special needs to be done, just set the '`value`' attribute of the input field, e.g. 'April 18, 2014' and you are good to go. In case when the submitted (hidden) value differs in formatting from the displayed value, you can set a default value of the input field by adding '`data-value`' HTML attribute to the original input field and format its value according to the format of the submitted value ('`formatSubmit`'). Code in listing 8.7 shows the latter, most common case when the default value needs to use the same format as the submitted value:

Listing 8.7 Setting the default value formatted as the submitted value

```
...
<input id="date" name="date" type="text" data-value="2015-10-25 00:00:00"
  placeholder="Choose a date">
...

<script>
$(function() {
  // Enable Pickadate on an input field using a custom format for submitted value
  $('#date').pickadate({
    formatSubmit : 'yyyy-mm-dd 00:00:00'
  });
});
</script>
```

Now when you load the page and click on the date picker you will see that by default it will be set to October 25, 2015 just like we wanted it to be (figure 8.4):

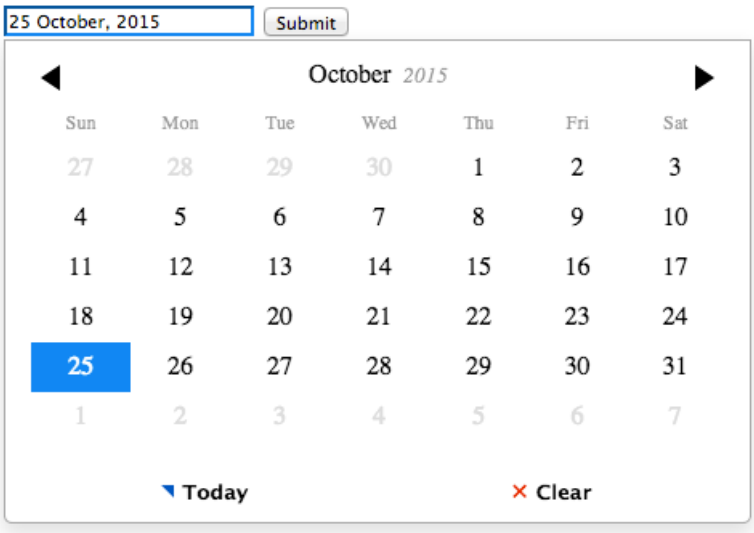


Figure 8.4 Setting data-value results in the pre-defined calendar setting

2.6 Changing the language used in the date picker

Pickadate comes with an easy way to switch the language and the defaults used to name days, months and to format dates to accomodate all users of your application. By including one of the provided language files (get them here[9]) in the page where you use this date picker you can instantly change the locale used in the date picker.

As a fun example, let's change the names of the months used in the calendar to Pirate-themed expressions. To do that, we'll extend the defaults of the date picker and set the names of the months as an array passed

[9] https://github.com/amsul/pickadate.js/tree/dev/lib/translations

to 'monthsFull' option. Code in listing 8.8 shows how with this simple addition before initializing the date picker you can instantly change the locale used in the date picker:

Listing 8.8 Changing the language used in the date picker

```
...
<form>
  <input id="date" name="date" type="text" placeholder="Choose a date, pirate">
  <input type="submit" value="Submit" class="btn btn-info btn-block">
</form>
...

<script>
  // Add custom names for the months
  jQuery.extend( jQuery.fn.pickadate.defaults, {
      monthsFull: [ 'Januarr!', 'Februahoy', 'Marr!ch', 'Ayepril', 'Mayday',
      'Jolly June', 'Julaye', 'Arr!gust', 'Septembrum', 'Octobrum', 'Novembrum', 'Decembru\
m' ]
  });

  $(function() {
    // Enable Pickadate on an input field
    $('#date').pickadate();
  });
</script>
```

With this simple addition, the date picker suddenly has a nice pirate feel to it, as you can see in the figure 8.5 below:

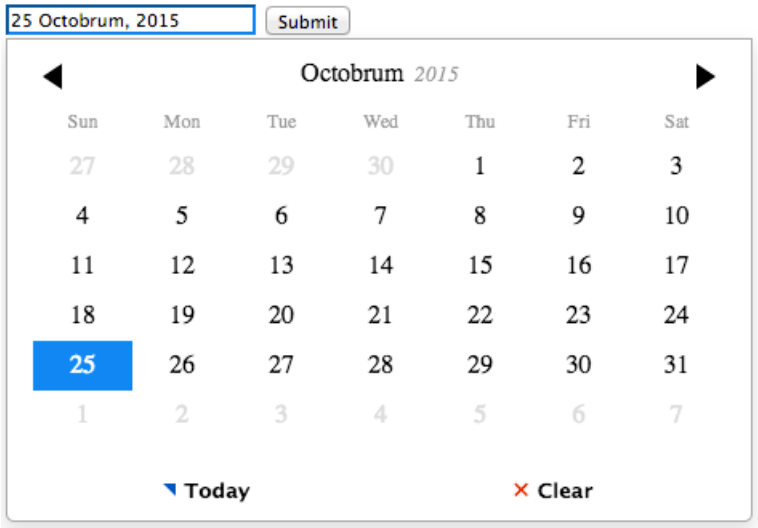


Figure 8.5 Pirate themed datepicker

The full list of options that you can customize in the plugin's defaults is shown below in listing 8.9:

Listing 8.9 Localization options of Pickadate

```
jQuery.extend( jQuery.fn.pickadate.defaults, {
  monthsFull: [ ],   // Array of full month names, like 'January' ...
  monthsShort: [ ],  // Array of short month names, like 'Jan', 'Feb' ...
  weekdaysFull: [ ], // Array of full day names, like 'Monday', 'Tuesday' ...
  weekdaysShort: [ ],// Array of short day names, like 'Mon', 'Tue' ...
  today: '',         // String to be used to mark 'Today' button
  clear: '',         // String to be used to mark 'Clear' button
  firstDay: 1,       // First day of the week
  format: '',        // Default format of the displayed value
  formatSubmit: ''   // Default format of the submitted value
});
```

With this easy way to change localization options, adding your own languages to the plugin should now be trivial. Have fun creating your own date picker localizations! Now that you know how to set and change various options, set locale of the plugin and the default values, we can proceed with this chapter and implement a date picker in a small web application. With the knowledge you have gained up to this point it will not be a problem.

2.7 Building the back end for a date picker

Let's build a complete application that will use a date picker to demonstrate the concepts you have learned in this chapter. The back end of this application will process the submission of a form containing a date picker and will do the following:

- If the date submitted is a valid date (i.e. the user didn't put any malicious input), store the entry in the database
- If the date submitted is not a valid date, show an error message to the user.

Get the full application

Fully functional application source code is available on Github at https://github.com/msurguy/frontend-book/tree/master/ch8/complete-application. Feel free to install it and see it in action on your machine.

The algorithm of the whole application is illustrated in the diagram in figure 8.6 below:

Figure 8.6 Diagram of the application

Using this application flow, let's create simple route definitions, the DB structure of the application, view templates containing the date picker and finally, connect everything together into an application. Let's get started by creating two simple routes that will contain all logic of the application.

2.7.1 Creating the route definitions

Like you've seen in other chapters of this book, the application containing a form will have a minimal amount of routes. Let's create two routes, one that will show the form to the user and the other will process form

submission. Code in listing 8.10 shows route definitions for these two application endpoints:

Listing 8.10 Route definitions for the application

```
// routes.php

// Show a page with a datepicker
Route::get('/', function()
{
  ...
});

// Process submission of the form containing the date picker
Route::post('/', function()
{
  ...
});
```

When the user navigates to the index route ("/") of the application, we'll display the date picker using Pickadate plugin. Submission of the form containing the date picker will execute the POST index route. All logic for the validation and processing of the form will be placed there. With these route placeholders in place, let's now create the DB structure that will store user's submissions.

2.7.2 Creating the database structure and the model

The DB structure for this application will be incredibly simple. We'll only store the date that the user chooses and no other information beside the fields required by the back end framework (Laravel). The database structure for our table called "dates" will be as follows:

- id (autoincrementing integer, primary key)
- date (datetime)
- created_at (datetime)
- updated_at (datetime)

Let's create a migration for this simple table. In the terminal run the following command while you are in the application's root folder:

```
php artisan migrate:make create_dates_table
```

Running this command should create a file in "app/database/migrations" folder with a name "create_-dates_table" in it. Open up this file and add the following migration definition to it (listing 8.11):

Listing 8.11 Migration for the 'dates' table

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateDateTable extends Migration {

        public function up()
        {
                Schema::create('dates', function($table)
                {
                    $table->increments('id');
                    $table->dateTime('date');
                    $table->timestamps();
                });
        }

        public function down()
        {
                Schema::dropIfExists('dates');
        }
}
```

With this migration in place, let's run the migration. Running the migration will create the DB structure for our application. Run the migration with the following command in the Terminal:

```
php artisan migrate
```

Configure DB settings

Before running the migration make sure you configured the DB settings in "app/config/-database.php" file

Creating a model will allow us to work with the "dates" table as we would with any other objects in PHP. Let's go ahead and create "Date" model by placing the contents of code in listing 8.12 into a file called "Date.php" in "app/models" folder:

Listing 8.12 Model for the 'dates' table

```
// app/models/Date.php
<?php

class Date extends Eloquent{}
```

With the database configured, the table created and the model defined, we can now create the templates that will make up the user interface (front end) of the application.

2.7.3 Creating the front end

There will be two view templates, one that shows a date picker (and an error if there is any) and another one that shows the results of the form submission. Let's create the first template by using Pickadate plugin. We'll set the format of the submitted value to MySQL datetime-friendly format and will assign a default date to show in the date picker to be May 1st, 2014. Code in listing 8.13 shows the Blade template for the index page (store it in "app/views/index.blade.php"):

Listing 8.13 Index page view template

```
@extends('layout')

@section('styles')
  {{ HTML::style('vendor/pickadate/compressed/themes/classic.css') }}
  {{ HTML::style('vendor/pickadate/compressed/themes/classic.date.css') }}
@endsection

@section('content')
<div class="row centered-form">
  <div class="col-xs-12 col-sm-12 col-md-4 col-sm-offset-2 col-md-offset-4">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Integrating a date picker</h3>
      </div>
      <div class="panel-body">

        {{ Form::open() }}

          <div class="form-group">
            <label for="date">Please choose your favorite date</label>

            <input id="date" name="date" type="text" data-value="2014-05-01 00:00:00" clas\
s="form-control" placeholder="Pick one...">
          </div>

          <input type="submit" value="Submit" class="btn btn-info btn-block">

        {{ Form::close() }}
      </div>
    </div>
  </div>
</div>
@stop

@section('scripts')
  {{ HTML::script('vendor/pickadate/compressed/picker.js') }}
  {{ HTML::script('vendor/pickadate/compressed/picker.date.js') }}
  <script type="text/javascript">
```

```
    $( '#date' ).pickadate({
      formatSubmit : 'yyyy-mm-dd 00:00:00',
      hiddenName: true
    });
  </script>
@stop
```

Layout template used in the application

The layout Blade template for the application ("app/views/layout.blade.php") can be retrieved from the source on Github here[10]

With the index page complete, let's switch to the results page. When the form is submitted and validated properly, we would like to show the value that user has submitted along with a list of all previously submitted values. Let's call this template "results.blade.php" and place it in "app/views" folder. We will place a loop iterating over an array of objects containing previous user submissions along with the value that the user just chose. Code in listing 8.14 shows the complete Blade template for the results page:

Listing 8.14 Results page view template

```
@extends('layout')

@section('content')
<div class="row centered-form">
  <div class="col-xs-12 col-sm-12 col-md-4 col-sm-offset-2 col-md-offset-4">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Results</h3>
      </div>
      <div class="panel-body">
        You have chosen: {{ $date }}<br>

        Previously chosen entries:<br>
        @foreach($dates as $entry)
          {{ $entry->date }}</br>
        @endforeach
        <br>

        <a href="{{ url('/') }}" class="btn btn-default btn-block">Go back to the date pic\
ker</a>
      </div>
    </div>
  </div>
</div>
@stop
```

[10] https://github.com/msurguy/frontend-book/blob/master/ch8/complete-application/app/views/layout.blade.php

With the view templates created we can now focus on adding actual logic in the application. Let's place that in the routes that we defined previously.

2.7.4 Adding the logic

The GET index route should just show us the index page. The POST route will retrieve the form submission, confirm that the date submitted is indeed a date and then store the submission in the database. Let's complete the GET index route first. Code in listing 8.15 shows that we only need to render the "index.blade.php" template when the user navigates to the index page of the application:

Listing 8.15 GET Index route completed

```
// Show a page with a datepicker
Route::get('/', function()
{
  return View::make('index');
});
```

The POST index route is where the magic of date validation and processing happens. This route will contain the following logic (listing 8.16):

Listing 8.16 POST Index route

```
// Process submission of the form containing the date picker
Route::post('/', function()
{
  // Get user's input

  // Validate input
  if(/* date isn't empty && date is valid */){
    // Create a new entry in the DB using Eloquent ORM
    // Retrieve all previously submitted dates
    // Display the results page, passing the date that the user has submitted
    // and all dates previously submitted
  } else {
    // Redirect to index page with an error notification when input isn't valid
  }
});
```

Great. The skeleton of the route tells us that we should check if the date is valid but there is no built-in method in PHP to do that if we provide a string formatted as MySQL datetime field. To validate the date in PHP and make sure it is in MySQL datetime format let's use this little function (place it in routes.php for example):

Listing 8.17 Date validator

```
// A helper function to determine if date input is valid or not
function validateDate($date, $format = 'Y-m-d H:i:s')
{
  $d = DateTime::createFromFormat($format, $date);
  return $d && $d->format($format) == $date;
}
```

Now we can use this function to determine if the input that the user provided is indeed a date and not some malicious SQL injection. Let's put this all together and complete the POST index route. Code in listing 8.18 shows the POST index route containing the logic to gather and verify user's input. When the input validates, the application will create a new entry in the "dates" table, otherwise an error message will be passed to the index page of the application:

Listing 8.18 Complete route definition for the POST index route

```
1  // Process submission of the form containing the date picker
2  Route::post('/', function()
3  {
4    // Get user's input
5    $date = Input::get('date');
6
7    // Validate input
8    if($date != '' && validateDate($date)){
9      // Create a new entry in the DB using Eloquent ORM
10     $newDate = new Date;
11     $newDate->date = $date;
12     $newDate->save();
13
14     // Retrieve all previously submitted dates
15     $dates = Date::all();
16
17     return View::make('result', ['date' => $date, 'dates' => $dates]);
18   } else {
19     // Redirect to index page with an error notification when input isn't valid
20     return Redirect::to('/')->with('invalid_date', true);
21   }
22 });
```

The last step that is required is to actually show the error message to the user. Let's add the following code in the "index.blade.php" file before opening the form tag (listing 8.19):

Listing 8.19 Showing the error messages to the user

```
...
<div class="panel-body">

  @if(Session::has('invalid_date'))
    <div class="alert alert-warning alert-dismissable">
      You have not chosen a date or have entered invalid input. Please try picking a date \
again.
    </div>
  @endif

{{ Form::open() }}
...
```

That's it! With this code in place we now have a fully functional application that uses a date picker to assist the user with choosing a particular date. You can now take advantage of all features of Pickadate plugin and modify its options to suit your application.

Check out completed application

The complete application is hosted online and is available at this URL: http://books.maxoffsky.com/frontend/ch8. Enjoy!

2.8 Summary

In this chapter you have learned a great deal about date pickers. You have learned the theory behind using a date picker to help the user choose a date and store it in a format that your application will understand. In the process you have met a great responsive library for integrating date pickers called "Pickadate" and have explored its many options of formatting the date, changing the looks and localizing it to fit your application's audience. Finally, you have applied the skills from this chapter to build a fully functional back end application that integrates a date picker. With the knowledge acquired it will be now easy to use other date pickers and integrate a time picker that comes with Pickadate.

In the next chapter of the book we will shift the gears a bit and get into more advanced topics such as AJAX file uploads. You'll meet one of the greatest libraries providing AJAX file upload functionality and photo cropping/resizing.