Итак, давайте создадим шаблон для создания планет в файле app/views/planets/add.blade.php . Начнем с «каркаса»:

@extends('layout')

@section('title')

Добавление планеты

@stop

@section('content')

<div class="jumbotron">

    <div class="container">

        <h2>Добавление планеты</h2>

        {{ Form::open(array('url' => action('PlanetsController@postAdd'), 'method' => 'post', 'role' => 'form', 'class' => 'form-horizontal')) }}

        {{ Form::close() }}

    </div>

</div>

@stop

Как можно видеть, форма, пока пустая, создается вызовами  Form::open(...) и  Form::close() . С последним, думаю, все понятно: он просто генерирует закрывающий тэг  </form> . А вот Form::open(...) рассмотрим подробнее.

Синтаксис этой команды на самом деле прост: в передаваемом массиве мы передаем «настройки» формы. Есть некоторые «зарезервированные» ключи у этого массива, как например  url — он задает  action формы, то есть адрес (см. примечание), куда будет отправлена форма. Остальные же элементы массива просто определяют дополнительные атрибуты тэга  <form ...> . В нашем случае:

* 'method' => 'post' добавляет атрибут  method , указывающий, что форма должна быть отправлена методом POST
* 'role' => 'form'  и  'class' => 'form-horizontal' являются чисто украшательными, так мы создаем форму, используя Bootstrap.

**Примечание**: в данном случае адрес указывается не напрямую  /planets/add , а посредством вспомогательной функцииaction , которая генерирует этот путь исходя из переданного ей контроллера и метода. В этом есть смысл, если вдруг в определении маршрута вы решите заменить planets  на что-то другое, при этом код создания этой формы менять не придется.

Теперь, чтобы это заработало, нужно создать, пока пустой, метод postAdd  в нашем контроллере, иначе функция action  сгенерирует ошибку:

public function postAdd() {

}

Опять же, для тех, кто вдруг не полностью усвоил второй урок, напомню: мы имеем дело с двумя разными маршрутами, хоть и имеющих одинаковый URL /planets/add . При запросе адресаhttp://sbshare.localhost:8000/planets/add  из броузера (это GET  запрос), будет выполнен методgetAdd , в то время как форма будет отправлена на такой же адрес, но уже методом POST , и, следовательно, отрабатывать будет метод postAdd  контроллера.

Теперь я предлагаю сделать следующее: саму форму мы вынесем в отдельный под-шаблонform.blade.php , а в этот лишь добавим кнопку submit  (отправку формы) и инклюд самой формы. Таким образом часть шаблона add.blade.php будет такой:

 {{ Form::open(array('url' => action('PlanetsController@postAdd'), 'method' => 'post', 'role' => 'form', 'class' => 'form-horizontal')) }}

        @include('planets/form')

        <div class="form-group">

            <div class="col-sm-2">&nbsp;</div>

            <div class="col-sm-5">

                <button type="submit" class="btn btn-primary submit-button">Добавить</button>

            </div>

        </div>

        {{ Form::close() }}

Обратить внимание стоит на инклюд под-шаблона: @include('planets/form') , который будет содержать саму форму. Делаю я так потому, что для редактирования планеты у нас будет абсолютно идентичная форма. Разница лишь в открывающей инструкции — в случае с редактированием форма будет открываться посредством  Form::model(...) — что позволяет «привязать» модель планеты к форме. Но об этом позже. Сейчас давайте напишем код для генерации само формы в файле app/view/planets/form.blade.php .

Как я уже говорил, для генерации различных элементов форм у фасада Form  есть множество методов. Например, для генерации text-input ‘а в простейшей форме может использоваться такая запись:

Form::text('fieldName');

Результатом этого вызова будет такой input :

XHTML

<input type="text" name="fieldName" value="" />

Как и другие методы фасада, метод text может принимать дополнительные параметры:

* второй необязательный параметр указывает значение value это поля
* третий необязательный аргумент — это массив, описывающий дополнительные атрибуты input’а. В большинства случае он используется для указания требуемого CSS-класса input’а

Например, вызов Form::text('fieldName', 'some-value', array('class' => 'my-input-class'))  сгенерирует такой input :

<input type="text" name="fieldName" value="some-value" class="my-input-class" />

Помимо метода *text* у FormBuilder’а имеются, например, такие методы с аналогичными аргументами: *email*, *password*, *textarea*, *select*. Метод *select* немного отличается: к нему добавляется еще один аргумент (после имени поля) — массив, содержащий все опции этого выпадающего списка.

Итого, наша форма может выглядеть так:

<div class="form-group">

    <label for="sector" class="col-sm-2 control-label">Сектор</label>

    <div class="col-sm-5">

        {{ Form::select('sector', array(), null, array('class' => 'form-control')) }}

    </div>

</div>

<div class="form-group">

    <label for="level" class="col-sm-2 control-label">Уровень</label>

    <div class="col-sm-5">

        {{ Form::text('level', null, array('class' => 'form-control')) }}

    </div>

</div>

<div class="form-group">

    <label for="star" class="col-sm-2 control-label">Звезда</label>

    <div class="col-sm-5">

        {{ Form::text('star', null, array('class' => 'form-control')) }}

    </div>

</div>

<div class="form-group">

    <label for="system" class="col-sm-2 control-label">Система</label>

    <div class="col-sm-5">

        {{ Form::text('system', null, array('class' => 'form-control')) }}

    </div>

</div>

<div class="form-group">

    <label for="planet" class="col-sm-2 control-label">Планета</label>

    <div class="col-sm-5">

        {{ Form::text('planet', null, array('class' => 'form-control')) }}

    </div>

</div>

И т.д.