In this installment, we’ll examine a cool Laravel AJAX CRUD Tutorial. There are a few benefits to handling database interactions with ajax. We know that it helps to make page updates very quick, reduces the amount of bandwidth used, and provides for a slick user experience. We’ll make use of Laravel on the backend and jQuery on the client side to set up a fully functioning Laravel ajax crud tutorial application.

**1. Create a Model and a Migration**

First off, we are going to create a Model to represent the entities we will be creating, reading, updating and deleting. In our case we will use the concept of a Link. Let’s use artisan to create the model and migration for us in one step.  
php artisan make:model Link -m

This command creates both a Link Model and an associated migration. We can open the model found in /app/Link.php now.

**/app/Link.php**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php    namespace App;    use Illuminate\Database\Eloquent\Model;    class Link extends Model  {      //  } |

**2. Define the Fillable fields on the Model**

This allows us to mass assign data to the database when we create a new link with our ajax app.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php    namespace App;    use Illuminate\Database\Eloquent\Model;    class Link extends Model  {      protected $fillable = ['url', 'description'];  } |

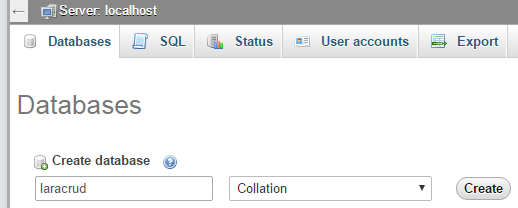
**3. Define the fields in the migration file we created**

We’ll keep this really simple. All we are adding is a field to hold the url of a link, and the description of that link as well. For good measure, we will also create 2 records in the up() method of our migration so we have just a little bit of data to work with when we first get started.

**/database/migrations/xxxx\_xx\_xx\_xxxxxx\_create\_links\_table.php**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | <?php    use Illuminate\Support\Facades\Schema;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Database\Migrations\Migration;  use App\Link;    class CreateLinksTable extends Migration  {      /\*\*       \* Run the migrations.       \*       \* @return void       \*/      public function up()      {          Schema::create('links', function (Blueprint $table) {              $table->increments('id');              $table->string('url');              $table->string('description');              $table->timestamps();          });            Link::create([              'url' => 'https://laravel.com/',              'description' => 'Laravel - The PHP framework for web artisans.',          ]);            Link::create([              'url' => 'https://laracasts.com/',              'description' => 'The best PHP and Laravel screencasts on the web.',          ]);      }          /\*\*       \* Reverse the migrations.       \*       \* @return void       \*/      public function down()      {          Schema::dropIfExists('links');      }  } |

**4. Create the database and update the .env file**

Of course before we can run any migrations, we need to have a database to connect to. In this example we have homestead running along with phpmyadmin to make working with our databases and tables super easy. Here you can see we create the database named **laracrud**.  


In the .env file we simply change the database to *laracrud* and leave the other settings at their defaults.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | DB\_CONNECTION=mysql  DB\_HOST=127.0.0.1  DB\_PORT=3306  DB\_DATABASE=laracrud  DB\_USERNAME=homestead  DB\_PASSWORD=secret |

**5. Run the migrations**

With the prior legwork of creating our migration file and database, along with specifying which database to use in our .env file, we are ready to run the migrations.  
php artisan migrate  
**Fantastic!** Let’s keep going.

**6. Create your routes**

**routes/web.php**

Within the routes file, we need to set up the endpoints that our ajax requests will be able to access. We’ll just use some simple route closures to set up loading the main page, creating a link, editing a link, as well as deleting a link.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | //--LOAD THE VIEW--//  Route::get('/', function () {      $links = Link::all();      return view('laracrud')->with('links', $links);  });    //--CREATE a link--//  Route::post('/links', function (Request $request) {      $link = Link::create($request->all());      return Response::json($link);  });    //--GET LINK TO EDIT--//  Route::get('/links/{link\_id?}', function ($link\_id) {      $link = Link::find($link\_id);      return Response::json($link);  });    //--UPDATE a link--//  Route::put('/links/{link\_id?}', function (Request $request, $link\_id) {      $link = Link::find($link\_id);      $link->url = $request->url;      $link->description = $request->description;      $link->save();      return Response::json($link);  });    //--DELETE a link--//  Route::delete('/links/{link\_id?}', function ($link\_id) {      $link = Link::destroy($link\_id);      return Response::json($link);  }); |

**7. Create the view files**

Here, we will make use of both a layout view and a partial view that simply displays our content. The app.blade.php file is our layout. It contains all the boilerplate that we would not want to have to duplicate on other pages. Note that we make use of the flexbox version of Bootstrap 4 in the head area of the layout file, and include all needed JavaScript near the end of the file. Putting JavaScript at the bottom of the page is a good practice to follow.

**/resources/views/layouts/app.blade.php**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <!DOCTYPE html>  <html lang="en">  <head>      <!-- Required meta tags always come first -->      <meta charset="utf-8">      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">      <meta http-equiv="x-ua-compatible" content="ie=edge">        <!-- CSRF Token -->      <meta name="csrf-token" content="{{ csrf\_token() }}">        <title>Laravel AJAX CRUD</title>        <!-- Bootstrap CSS -->      <link rel="stylesheet" href="css/4.0.0-alpha.5.bootstrap-flex.min.css">        <!-- Custom styles for this template -->      <link href="css/sticky-footer-navbar.css" rel="stylesheet">  </head>    <body style="margin-top: 60px" class="container">    @yield('content')    <script src="js/3.1.1.jquery.min.js"></script>  <script src="js/1.3.7.tether.min.js"></script>  <script src="js/4.0.0-alpha.5.bootstrap.min.js"></script>  <script src="js/laracrud.js"></script>  </body>  </html> |

**/resources/views/laracrud.blade.php**

In our partial view file, we are able to set up a few things. First, we make use of a nice card, which is [**new in Bootstrap 4**](http://vegibit.com/what-is-new-in-bootstrap-4/), to display a little information about our small app, as well as to provide a button to add a new link. After this, we make use of another cool new feature in Bootstrap 4, and that is the inverse table which makes for some cool styling effects. Finally, we have a modal, which of course is not displayed until we actually need to add or edit a link. The interesting thing about this modal is that it will be massaged by jQuery to make sure it contains the right information before displaying that modal. This way we can use the same modal whether adding a link, or editing a link.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | @extends('layouts.app')    @section('content')        <div class="container">            <div class="card card-block">              <h2 class="card-title">Laravel AJAX Examples                  <small>via jQuery .ajax()</small>              </h2>              <p class="card-text">Learn how to handle ajax with Laravel and jQuery.</p>              <button id="btn-add" name="btn-add" class="btn btn-primary btn-xs">Add New Link</button>          </div>            <div>              <table class="table table-inverse">                  <thead>                  <tr>                      <th>ID</th>                      <th>Link</th>                      <th>Description</th>                      <th>Edit or Delete</th>                  </tr>                  </thead>                  <tbody id="links-list" name="links-list">                  @foreach ($links as $link)                      <tr id="link{{$link->id}}">                          <td>{{$link->id}}</td>                          <td>{{$link->url}}</td>                          <td>{{$link->description}}</td>                          <td>                              <button class="btn btn-info open-modal" value="{{$link->id}}">Edit                              </button>                              <button class="btn btn-danger delete-link" value="{{$link->id}}">Delete                              </button>                          </td>                      </tr>                  @endforeach                  </tbody>              </table>                <div class="modal fade" id="linkEditorModal" aria-hidden="true">                  <div class="modal-dialog">                      <div class="modal-content">                          <div class="modal-header">                              <h4 class="modal-title" id="linkEditorModalLabel">Link Editor</h4>                          </div>                          <div class="modal-body">                              <form id="modalFormData" name="modalFormData" class="form-horizontal" novalidate="">                                    <div class="form-group">                                      <label for="inputLink" class="col-sm-2 control-label">Link</label>                                      <div class="col-sm-10">                                          <input type="text" class="form-control" id="link" name="link"                                                 placeholder="Enter URL" value="">                                      </div>                                  </div>                                    <div class="form-group">                                      <label class="col-sm-2 control-label">Description</label>                                      <div class="col-sm-10">                                          <input type="text" class="form-control" id="description" name="description"                                                 placeholder="Enter Link Description" value="">                                      </div>                                  </div>                              </form>                          </div>                          <div class="modal-footer">                              <button type="button" class="btn btn-primary" id="btn-save" value="add">Save changes                              </button>                              <input type="hidden" id="link\_id" name="link\_id" value="0">                          </div>                      </div>                  </div>              </div>          </div>      </div>    @endsection |

**8. Create your AJAX logic**

Now that we have all of the prior steps taken care of, we can create the logic in our lararud.js file to handle the ajax logic. Check out our [**tutorial about ajax in jQuery**](http://vegibit.com/jquery-ajax-tutorial/) if you need a refresher.

The first section which deals with opening the modal to create a link is pretty straightforward. When a user clicks the element with an id of #btn-add, set the element with the ide of #btn-save to have a value of “add”. Next, we clear out any data that might exist in the form fields. Finally, we can actually display the modal to add a link with $('#linkEditorModal').modal('show');

The next section deals with opening a modal to update an existing link. The reason for using **$(‘body’).on(‘click’, ‘.open-modal’, function () { });** instead of *$(“.open-modal”).click( function() { });* is because this first approach will work for all tags with .open-modal in the body of the page, whether already present or dynamically added in the future by JavaScript. This is important because when we edit a link, it’s contents will be dynamically added to the page. If we use the second option mentioned here, those buttons will fail to trigger any action when clicked.

Now we need a section of logic to deal with what happens when a user clicks the save button of the open modal. This is pretty interesting because we are using the same modal, but we need to take a different action when the user clicks the save button depending on if we are creating a new link, or updating an existing link. Firstly though, you can see we set up our ajax requests to support the csrf protection Laravel provides. Then, we prevent the button on the form from submitting by itself. You’ll often see that *e.preventDefault();* syntax to denote this. Next, we use jQuery to retrieve the values set in the form and place this in the formData variable. We’ll need this data once we actually send off the http request from JavaScript because this data will be sent along with the request as the payload. Next, we grab the state of the save button. It will be either “add” or “update”. We can then make a decision on whether to send a POST or a PUT request based on this information. You’ll see we also fetch the id in question and set up the ajax url to use before making a call to $.ajax().

The ajax request itself gets just a little bit tricky here. First, we have to set the type of the request. Will it be a POST or PUT? Then we provide the url to send the ajax request to, this is built up in the logic just above. The data will contain the data from the form in the open modal window and finally the dataType will be in json format.

With all of that setup, we just need to define what happens in the case of a successful ajax request, and also one that results in an error. In our success key we store the function that will either append a new link to the list on a successful creation of a link, or replace an existing link with the updated version. We then reset the modal’s form data, and then close the modal itself. It is now ready for the next interaction with the user.

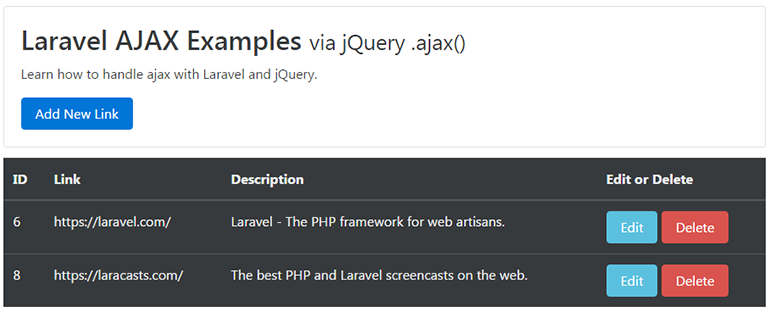
Last in our laracrud.js file is the logic to delete a link from the database and page. On success we simply delete the link from the page, and on failure we log data to the console.

**public/js/laracrud.js**

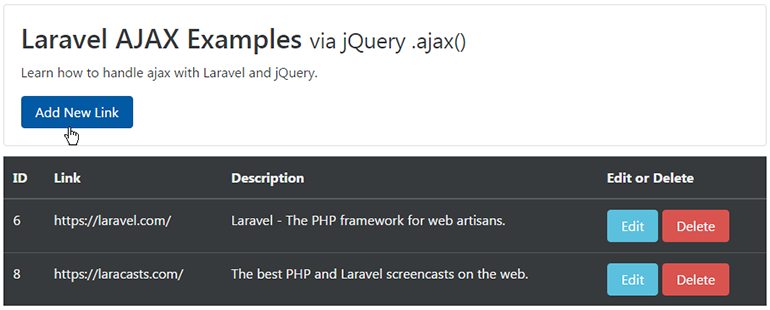
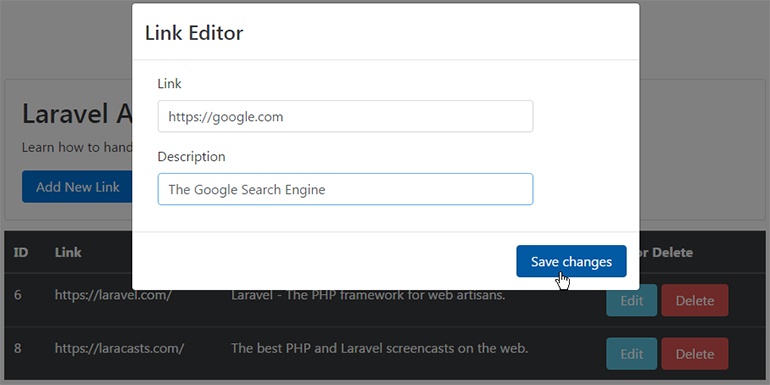
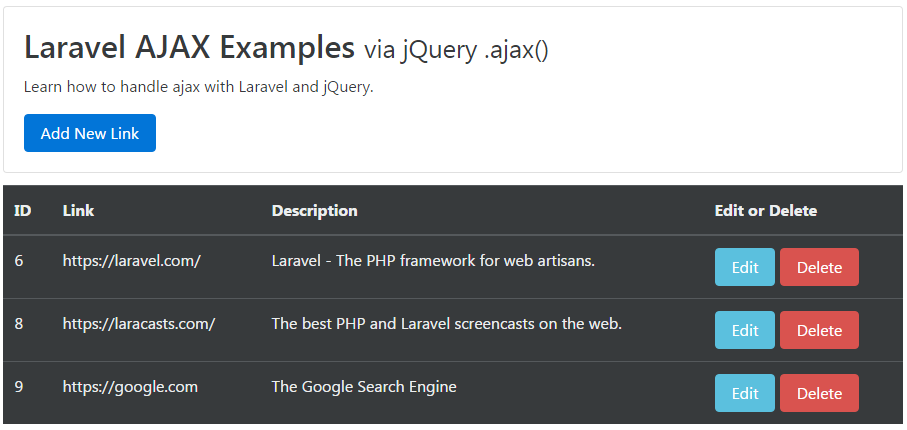
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | $(document).ready(function () {      ////----- Open the modal to CREATE a link -----////      $('#btn-add').click(function () {          $('#btn-save').val("add");          $('#modalFormData').trigger("reset");          $('#linkEditorModal').modal('show');      });        ////----- Open the modal to UPDATE a link -----////      $('body').on('click', '.open-modal', function () {          var link\_id = $(this).val();          $.get('links/' + link\_id, function (data) {              $('#link\_id').val(data.id);              $('#link').val(data.url);              $('#description').val(data.description);              $('#btn-save').val("update");              $('#linkEditorModal').modal('show');          })      });        // Clicking the save button on the open modal for both CREATE and UPDATE      $("#btn-save").click(function (e) {          $.ajaxSetup({              headers: {                  'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')              }          });          e.preventDefault();          var formData = {              url: $('#link').val(),              description: $('#description').val(),          };          var state = $('#btn-save').val();          var type = "POST";          var link\_id = $('#link\_id').val();          var ajaxurl = 'links';          if (state == "update") {              type = "PUT";              ajaxurl = 'links/' + link\_id;          }          $.ajax({              type: type,              url: ajaxurl,              data: formData,              dataType: 'json',              success: function (data) {                  var link = '<tr id="link' + data.id + '"><td>' + data.id + '</td><td>' + data.url + '</td><td>' + data.description + '</td>';                  link += '<td><button class="btn btn-info open-modal" value="' + data.id + '">Edit</button> ';                  link += '<button class="btn btn-danger delete-link" value="' + data.id + '">Delete</button></td></tr>';                  if (state == "add") {                      $('#links-list').append(link);                  } else {                      $("#link" + link\_id).replaceWith(link);                  }                  $('#modalFormData').trigger("reset");                  $('#linkEditorModal').modal('hide')              },              error: function (data) {                  console.log('Error:', data);              }          });      });        ////----- DELETE a link and remove from the page -----////      $('.delete-link').click(function () {          var link\_id = $(this).val();          $.ajaxSetup({              headers: {                  'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')              }          });          $.ajax({              type: "DELETE",              url: 'links/' + link\_id,              success: function (data) {                  console.log(data);                  $("#link" + link\_id).remove();              },              error: function (data) {                  console.log('Error:', data);              }          });      });  }); |

**9. Test Drive Your Laravel AJAX CRUD**

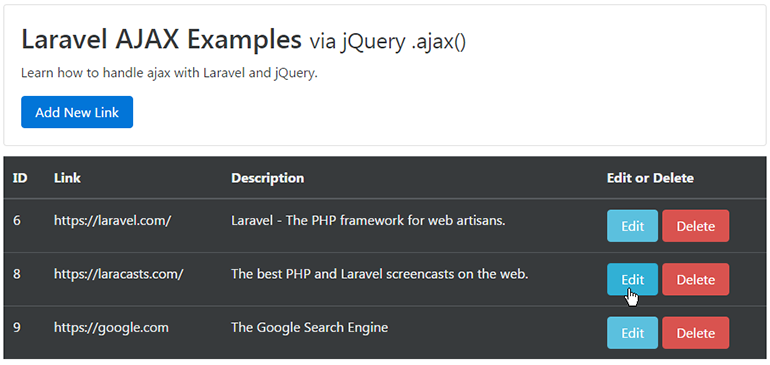
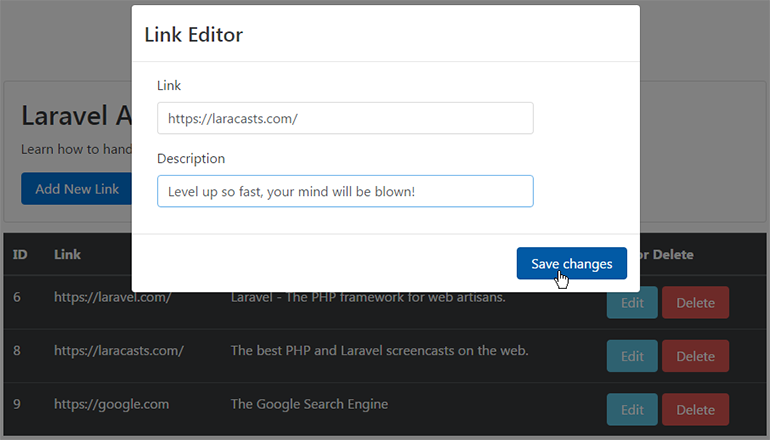
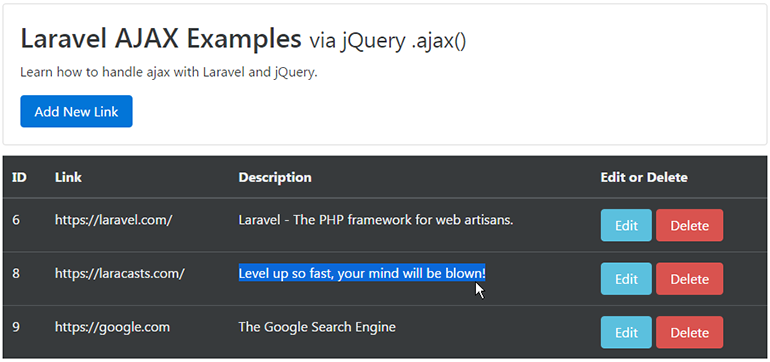
**Loading the default view.**



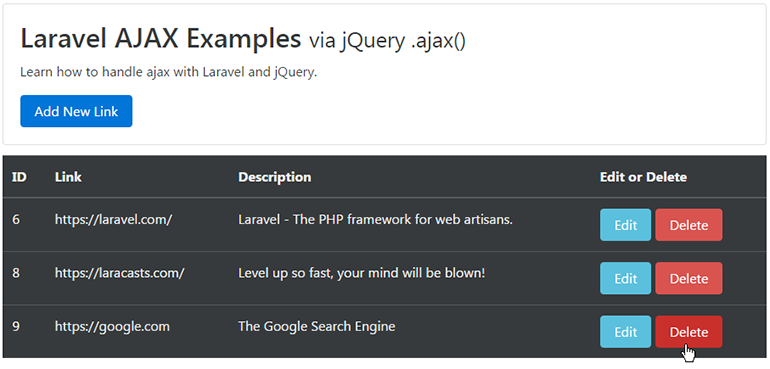
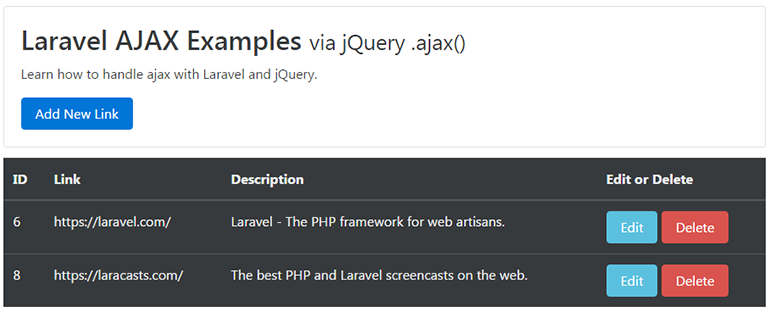
**Adding a new link**

**Editing an existing link**

**Delete a link**

**Laravel AJAX CRUD Tutorial Summary**

What a great series of examples of how to set up some basic ajax crud functionality in your Laravel app! We covered a fair amount of work. Most of the Laravel side is pretty straightforward. A lot of the logic happens in the JavaScript file via [**jQuery**](http://vegibit.com/tag/jquery/).