CREATING A SIMPLE NOTE APPLICTION USING LARAVEL ON CLOUD 9

Laravel is an open-source PHP web application framework, following the MVC architecutal pattern, today we will be building a simple notepad appliction complete with user registration. This web application will be entitled [simple-notes].

I apologize for those who will find this slow gowing, though I feel to make a pie or whatever you need to follow the steps and add ingredients at the right time in the right way, not just throw it all together at one time.

For this tutorial, the only typing need will be when creating a directory or a file, otherwise, the code or instructions canbe copied and entered or executed, but remember for terminal actions don't include ~/workspace $ when coping

In this tutorial we will cover

- Creating a workspace in Cloud9
- Setting up composer and LARAVEL
- Table migration
- Creating a simple note application

Create a workspace in Cloud9

Select the panel [*Create a new workspace*]

- Workspace Name: [*sites*]
- and select [*PHP, Apache & MySQL as the template*]
- and then click [*Create workspace*] to complete the process

We will now setup LARAVEL, with the help of COMPOSER, and customize environmental variables

This is a very simple process, weather it be at Cloud9 or some other server, first we will install Composer, LARAVEL leverages Composer to manage its dependencies.

Begin by installing Composer:

```
~/workspace $ composer global require "laravel/installer=~1.1"
```

Install our first site:

```
~/workspace $  ~/.composer/vendor/bin/laravel new simple-notes
```

With that, we are done setting up the skeleton of our site, we are now ready to build something amazing *(or at-least take baby steps to that end)*, as the message will say upon installation.

The following steps will set up the environmental variables, so we can access the database on our site. By default LARAVEL comes per-packaged with environmental variables which we will change before we go further. First we will cd into our new sites directory then we will edit the .env file, to our specific needs, to do this we will use nano. *The password will be left blank, though in a real site this is a very bad idea!*

cd into the sites directory:

```
~/workspace $ cd simple-notes
```

Open .env with nano:

```
~/workspace/simple-notes $ nano .env
```

We will change the default values of the file .env, when finished press [ctrl+x][y]and [enter] to save it

From:	To:
DB_HOST=localhost	DB_HOST=localhost
DB_DATABASE=homestead	DB_DATABASE=c9
DB_USERNAME=homestead	DB_USERNAME=[your_cloud9_user_name]
DB_PASSWORD=secret	DB_PASSWORD=

Migrations: A migration will help us to create/recreate a table(s)

A migration is a template to create a table.

We could use phpmyadmin for this purpose, though its not a good choice if we want to share our project, or put our creation on a production server, where we would have to manually, without mistakes recreate all the tables.

Migrations can be found in the directory tree @ [*simple-notes/database/migrations*], inside, Laravel comes ready with 2 tables, one called 'users' and another 'password_reset' for this lesson, we will only be working with 'users' and a custom table of our own creation called 'notes'.

We must install at-least install mysql before migrations can be done, we can install phpmyadmin too

Install mysql-ctl:

```
~/workspace/simple-notes  $ mysql-ctl install
```

Install phpmyadmin:

```
~/workspace/simple-notes  $ phpmyadmin-ctl install
```

Here we go, we will again use the terminal to create a 'notes' migration, for naming we wil stick to Laravel's convention. Which is *create_+[table name]+_table*

Create a new migration:

```
~/workspace/simple-notes  $ php artisan make:migration create_notes_table
```

Now when we look back at the directory tree (may need to refresh the tree) there will be a new file corisponding to the migration we just made which should look something like the following [*2015_8_3_000000_create_notes_table.php*], open this file to edit, and add a Schema to the method up().

Schema: def-describes an organized pattern of behavior

```
Schema::create('notes', function(Blueprint $table) {
        $table->increments('id');
        $table->integer('user_id')->unsigned()->default(0);                                   // reference to our users table
              $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');  // allows automatic deletions
        $table->string('title')->default('');
        $table->mediumText('content')->default('');
        $table->timestamps();
 });
```

The last step dealing with migrations, is we need to actually migrate it (create the tables)

Create the migration:

```
~/workspace/simple-notes  $ php artisan migrate
```

So now we can open phpmyadmin for our site, and do some tinkering or just look around. We can see what affects our code has made, as we develope.

Included in the migration is an

• id	which is unique to the entry [a primary key]
• user_id	(int) which will point to the author of the note, and allows cascading deletes
• title	(string) the title of the note
• content	(mediumString) the content of the note
• timestamps	which consist of 2 fields [created_at] and [updated_at]

Seeding the tables

Seeding the table, so we have some data to work with until we get the thing up and running

A side note before we seed our table, we will be seeding both the users and notes tables with some data, it is important that the users are seeded first, as the notes table makes references to the users table through notes.user_id. We could make our own module to contain our seeds, but since this project is very small, we will only be dealing with two tables, [users] and [notes] we will use the module provided by Laravel.

We will being by telling the **DatabaseSeeder.php** class to call the new class [*that we will create*] when seeding (located in the direcotry tree) @ [*database/seeds/DatabaseSeeder.php*]

```
class DatabaseSeeder extends Seeder
{
   public function run()
   {
      Model::unguard();

      $this->call(SimpleNoteSeeder::class);       // this is our new code

      Model::reguard();
   }
}
```

Insert a new class at the bottom of **DatabaseSeeder.php** the data provided is just example, it can be anything.

```
class SimpleNoteSeeder extends Seeder {
   public function run()   {
      DB::table('users')->delete();

      $password = Hash::make('password');

      $users_table = array(

               ['id' => 1, 'name' => 'John Smith', 'email' => 'JohnSmith@Somewhere.com', 'password' => $password,
               'created_at' => new DateTime, 'updated_at' => new DateTime],

               ['id' => 2, 'name' => 'John Doe',   'email' => 'JohnDoe@Somewhere.com',   'password' => $password,
               'created_at' => new DateTime, 'updated_at' => new DateTime],

               ['id' => 3, 'name' => 'Jane Doe',   'email' => 'JaneDoe@Somewhere.com',   'password' => $password,
               'created_at' => new DateTime, 'updated_at' => new DateTime],
      );
```

… continued

```
DB::table('users')->insert($users_table);

        DB::table('notes')->delete();

        $notes_table = array(

                        ['id' => 1, 'user_id' => 1, 'title' => 'title for notes 1',  'content' => 'content for note 1', 'created_at' => new DateTime, 'updated_at' => new
                        DateTime],

                        ['id' => 2, 'user_id' => 1, 'title' => 'title for notes 2',  'content' => 'content for note 2', 'created_at' => new DateTime, 'updated_at' => new
                        DateTime],

                        ['id' => 3, 'user_id' => 2, 'title' => 'title for notes 3',  'content' => 'content for note 3', 'created_at' => new DateTime, 'updated_at' => new
                        DateTime],

                        ['id' => 4, 'user_id' => 2, 'title' => 'title for notes 4',  'content' => 'content for note 4', 'created_at' => new DateTime, 'updated_at' => new
                        DateTime],

                        ['id' => 5, 'user_id' => 3, 'title' => 'title for notes 5',  'content' => 'content for note 5', 'created_at' => new DateTime, 'updated_at' => new
                        DateTime],

                        ['id' => 6, 'user_id' => 3, 'title' => 'title for notes 6',  'content' => 'content for note 6', 'created_at' => new DateTime, 'updated_at' => new
                        DateTime],
                            );
        DB::table('notes')->insert($notes_table);
    }
}
```

Now we can seed out tables, to do this, we will use the terminal

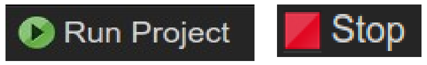
Seed the table:

```
~/workspace/simple-notes  $ php artisan db:seed
```

Now, if you installed phpmyadmin, if we look back at the tables, in phpmyadmin, we can see 4 tables have been seeded, the migrations and password_reset tables [which we won't be concerned with] the [users] table and our [notes] table

Thats it for the database setup, now we move on to creating running our site, right now we only have a splash screen, but that's okay, we will fix that shortly.

Running our site for the first time



Hit **Run Project** at the top/center of the development screen Currently you will see a directory listing; navigate the directory tree [*simple-notes/public*] to view the site, this is the Laravel's splash screen, but we will not be using this in a few moments. You can hit **stop** to end the session, then close the imbedded browser. For me it is sort of annoying to have to navigate the directory tree to get to our site, a simple fix to this is to add an index.php to the workspaces root directory, which relocates the header to our site. We can do this by adding a file [*index.php*] to the [*root I called it (sites)*] directory. When finished open the index.php file and insert the following code.

```
<?php
   header('Location: simple-notes/public');
```

Now when the project is ran, it will go strait to Laravel's splash screen.

We are now ready to begin building the site

We begin by making some controllers, these will allow the site to change pages easily with just a simple route for example: for a home page maybe it would look like */* or */home*, for a log in page something like */login*, a profile */user_name/profile*, this is what we see in the browsers navigation bar. Laravel makes this easy, and to get started we will create a sessions controller, this sessions controller will for our site now deal with authenticating users, and what they are allowed to do on the site. We will create two controllers for this project. In the directory tree of or development environment, the controllers will be found in [simple-notes/http/controllers]. We will stick to convention when naming assets (*mostly, usually*).

Using the command make:controller, flagged plain which will give us an empty class.

To create the Sessions controller:

```
~/workspace/simple-notes  $ php artisan make:controller SessionsController --plain
```

After creating the controller, we will open it and have a look inside. We will see a namespace and several use statements, and then the class.

To demonstrate the controller and routes (*routes will be covered a little bit latter*), lets add some code to change our default home page from the LARAVEL splash to something where we can begin to see how things work.

Though, the SessionsController will not manage our home page, for demonstration, we will use it as such. Inside the SessionsController class we will add an index() method and return a simple message, as follows: insert the return statement into the index() method of our controller class, and then save the file.

```
class SessionsController extends Controller
{
   public function index()   {
      return "Hello World!!!";
   }
}
```

of cource, this is not helpful in a real world application, this is just to demonstrate the concept, latter we will point to views.

Next we will change the route of our home page to point to our new controller: to do so ...

Open the file [*app/http/routes.php*] inside we see the default route, which points to the LARAVEL splash page, this is nice, but its not helpful for our site. So we will make some changes to point to Sessions.index()

Inside routes.php we see some comments and a Route::get(), but we will comment that out for now and add our own route, we will make a new [temporary] home page for our site

```
<?php

...

Route::get('/', function () {
   return view('welcome');
});
```

```
<?php

...
/*
Route::get('/', function () {
   return view('welcome');
});
*/
Route::get('/', 'SessionsController@index');
// we could also if we feel inclined
Route::get('/home', 'SessionsController@index');
```

Before we test the new changes, we need to update our routes cache to do so we will run in the terminal the

following command:

```
~/workspace/simple-notes  $ php artisan route:cache
```

in additon we can peek at the routes:

```
~/workspace/simple-notes  $ php artisan route:list
```

route:list is useful espcially on large sites with dozens of routes to see how things are actually being routed,

Now run the site with the new changes, we should see the 'Hello World!!” message?

For our sessions controller, we will not be using the index() method, we will move on to compleing the SessionsController with some explinations along the way.

SessionsController listing

After we create this controller, we will create the views associated with the named routes, and add the appropriate routes to the routes.php file.

At the top of the file the follow will include: (*above the class definition*)

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;                    // used to extract values from input controls
use App\Http\Requests;
use App\Http\Controllers\Controller;

use Auth;                                       // used to authenticate users
use App\User;                                   // the users table, rules and details, will cover latter
use Hash;                                       // we always need to encrypt passwords
use Input;                                      // for grabbing data from an input control
```

We will add several methods to the controller, in the SessionsController class (*inside the class definition*)

- **The create() method**
  first we will check to see if there is a logged in user, if so we will redirect to a guests page, otherwise it will return a view to, with content different for those logged in or not logged in

```
 public function create() {

        if (Auth::check()) {
           return redirect('guests');
        }
        return view('sessions.create');
}
```

- **The store() method**
  This method is called, when a new user is being logged in, first we extract the email and password, and assign them to $credentials, then we check the hashed password (done automatically by Laravel) and email, we redirect to login, else for now, just display a message

```
public function store(Request $request)   {

       $credentials = $request->only('email', 'password');

        if (Auth::attempt(['email' => $credentials['email'], 'password' => $credentials['password']])) {

            return redirect('login');
        }

        return 'attempted login failed';
    }
```

- **The Destroy() method**
  The destroy method, clears the authentication and redirects to the guest page, where an option to log in is made avalible

```
 public function destroy()  {

        Auth::logout();

        return redirect('guests');
    }
```

- **The Register() method**
  Shows the registration page, so someone can …

```
  public function register() {
        return view('sessions.register');
    }
```

- **The newUser() method** // the name for method is not convention, but sometimes I get lazy
  First the input of the new register will be validated, emails need to contain a name, an at symbol [@] a host and a dot something someone@gmail.com or anotherperson@somewhere.net, also the password and user name need to be at least 6 character long, and the name must be unique within the database. In addition there are 2 password fields match.

```
 public function newUser(Request $request) {
        // validate user inputs
        $this->validate($request, [
                'email'     => 'required|min:6|unique:users',
                'password'  => 'required|confirmed|min:6',
                'name' => 'required|min:6|unique:users',
            ]);
        // extract the parts we need to insert into the database
         $input = $request->only(
            'name',
            'email',
            'password',
            'password_confirmation'
        );
        // create the new user
         User::create([
            'name' => Input::get('name'),
            'email' => Input::get('email'),
            'password' => Hash::make(Input::get('password')),
        ]);
        return redirect('login');
}
```

Save the changes. And for our beginner site, thats all we will use for authentication, that was not too bad huh?

**complete file listing can be found at the end of this document*

Now we will update our routes.php file so they will point to our SessionsController class, and then add some views.

Update to routes.php, below we will notice the routes definitin match up with the methods provided, posts are when a submitt button is pressed, and gets are to load views, ommited is the home page that will be corrected when it comes up

```
<?php
    // the following two will be changed latter in the tutorial, but for
    // now they can stay so we can test changes thus far
    Route::get('/', 'SessionsController@index');
    // we could also if we feel inclined
    Route::get('/home', 'SessionsController@index');

    Route::get('/login', 'SessionsController@create');
    Route::get('logout', 'SessionsController@destroy');
    Route::post('login', 'SessionsController@store');
    Route::get('register', 'SessionsController@register');
    Route::post('register', 'SessionsController@newUser');
    Route::resource('sessions', 'SessionsController');
```

finally update the routes cache:

```
~/workspace/simple-notes  $ php artisan route:cache
```

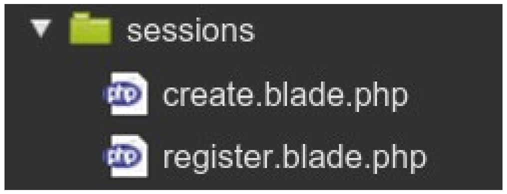
if the application is now ran we will still see our “Hello World!!” message, but if we try to navigate to . https://sites[user_name].c9.io/simple-notes/public/{login or logout or register} we get page not found, thats because we have not created the views yet.

Creating the session views

With views, we add a [.blade] in the name of the file, which allows us to use some fancy coding avalible to us provided by Laravel, more on that in a moment. But for now we will create a folder, and 2 php files inside that folder for our views.

In the directory tree locate the views folder [*resources/views*], right click and add a new folder named [*sessions*] recall all our calls in the SessionsController were prepended with *sessions* this is the name of the sub folder inside the views folder.

In the sessions folder add two php files [*create.blade.php* and *register.blade.php*]



While we are here, we will also add another file, to the [*views*] folder this file will be used as a template for all the others, using the features of blade, we don't need to retype the HTML/HEADER/BODY tags or remember to link to our style sheet, since its already there. The file added to [*views*] is [*layout.blade.php*]. With the following code. Notice the 9th line, which is @yield('content') we, could have several @yields depending on what we are building, this is where blade will insert our code, so whatever code we insert will be as if it were @yield.

The design I used is not nessisary to follow, this is just a quick out of the box, to get our site up and running, the details are in the imagination. I am going to speed up a bit as most of the following code should be clear, we will also include a styles.css before completion of the application, and a couple more views.

views/Layout.blade.php

```
<!doctype html>
<html lang="en">
    <head>
        <meta name="viewport" content="width=device-width">                    <!-- for handsets -->
        <link rel="stylesheet" href="/simple-notes/public/css/styles.css">     <!-- css we will add -->
    </head>
      <body>
        <div class="content">
            @yield('content')
        </div>
    </body>
</html>
```

views/sessions/create.blade.php
notice @section('content'), this file and all the other views extents layout, so we don't have to retype, as the project gets larger, it also helps to maintain consistancy from page to page

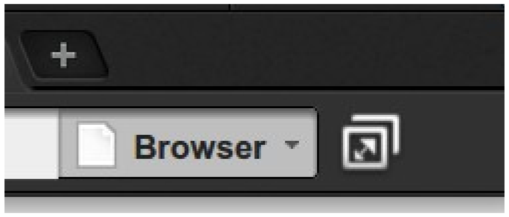
```
@extends('layout')
@section ('content')
   <h2>Login</h2>
    <form method="POST" action="login">
          {!! csrf_field() !!}

          <input type="hidden" name="_token" value="{{ csrf_token() }}">
        <div class="input_form">
                              <label>E-Mail Address</label>
                              <input type="email" name="email" value="{{ old('email') }}">
                              <label>Password</label>
                              <input type="password" name="password">
                                        <button type="submit">Login</button>
                                        <a href="register">not a member?</a>
                    </div>
          </form>
@stop
```

views/sessions/register.blade.php

this is a little more involved than is create.blade.php

```
@extends('layout')
    @if($errors->any())
            <div class='error_msg'>
                @foreach ($errors->all() as $error)
                    <p>{{$error}}</p>
                @endforeach
            </div>
        @endif

@section ('content')
   <h2>Register</h2>
    <form method="POST" action="register">

          {!! csrf_field() !!}

          <input type="hidden" name="_token" value="{{ csrf_token() }}">
        <div class="input_form">
                              <label>E-Mail Address</label>
                              <input type="email" name="email" value="{{ old('email') }}"  placeholder="email">
                              <label>Password</label>

                              <input type="password" name="password" placeholder="password">
                              <input type="password" name="password_confirmation"  placeholder="confirm password">

                              <label>User Name</label>
                              <input type="text" name="name" value="{{ old('name') }}"  placeholder="user name">


                              <button type="submit">Register</button>
                              <a href="login">already a member?</a>
                    </div>
          </form>
@stop
```

Now if we run the app (*hint: pop the web app out of the developmental window, so we can go back and forth quicker*)



I will use the url on my site, yours will be a different Cloud9 user_name.

If you navigate to *https://demo-johnsmithsvsu.c9.io/simple-notes/public/login* you will see a bold Login text some input boxes and a submit, there will not be any styling yet, because we have not yet created the styles.css. Also there is an anchor href to a registration page. Thats nice, but it does not do anything yet, and it looks fine, but we can do better, so lets continue.

Before tackling the second controller, we can create a css file, and make the site a little more professional looking, again though this is a simple style sheet just for demonstration.

We will add the CSS in the directory tree at [*public/css/styles.css*] create the folder [*/css*] and add the file [*styles.css*], which is linked to in the [*layout.blade.php*] file.

After this, we should beable to create a new user, and log in, although nothing else.

public/css/styles.css

```css
body {
   background-color: #222;
   font-size: 18px;
   font-family: Verdana, Geneva, sans-serif;
}

.content {
   border: 1px solid #aaa;
   background-color: #eee;
   border-radius: 5px;
   padding: 1em;
   max-width: 800px;
   margin: 0 auto;
}

.input_form {
   display: block;
   border: 1px solid #08a;
   padding: .5em 1em;
   border-radius: 5px;
   background-color: #e0e0ef;
   margin: 1em 0;
}

.input_form * {
   display: block;
   margin: 0 0 .5em 0;
}

.input_form input[type='text'],
input[type='password'],
input[type='email'], textarea {

   width: 100%;
   font-size: 1.25em;
}

.input_form label {

   font-size: 1.25em;
   font-weight: bold;
}
.input_form input[type="submit"] {
   font-size: 1.25em;
}

.code_block {

   border: 2px solid #558;
   background-color: #E3E4FA;
   font-size: 1em;
   padding: 5px 5px 20px 5px;
   overflow: auto;
   display:block;
}

.sub_block {
   margin: 1em;
   background-color: #F3F4FF;
   padding: 5px 10px 20px 10px;
}

.sub_block > * {

   margin: .25em 10px .25em 40px;
}

.sub_block h1,
.sub_block h2,
.sub_block h3,
.sub_block h4,
.sub_block h5,
.sub_block h6 {
   margin-left: 20px;
}

.sub_block > p {
   font-style: italic;
   font-weight: bold;
}
.time_stamps {
   font-size: .75em;
   font-style: italic;
}

.owners_select {
   background-color: #BCC6CC;
}

.error_msg {
   display: block;
   border: 1px solid black;
   background-color: red;
   color: white;
   padding: 3px;
   font-style: italic;
}

.error_msg > * {
   margin: 2px 0;
}

.sub_block input[type='text'],
input[type='password'],
input[type='email'],
textarea {
   font-size: 1.25em;
}

.sub_block label {

   font-size: 1.25em;
   font-weight: bold;
   margin-bottom: .5em;
}

.sub_block input[type="submit"],
button {
   font-size: 1.25em;
   padding: .5em 1em;
}
```

Now if we update our login page, *https://demo-johnsmithsvsu.c9.io/simple-notes/public/login we can see that we now have style, some people like that sort of thing.*

Moving on to the public controller

Back in the terminal we will create our final controller, so we can code it, as I was codeing this, I grew lazy, I mixed members in with a pages controller, and just called it PublicController, though they probably should be seperated.

To create the public controller:

```
~/workspace/simple-notes  $ php artisan make:controller PublicController --plain
```

In the controller will be added several methods: create(), post(), store(), update() and delete(); we will add views and update the route.php file before this can take effect with the final product being

```
<?php
/* the typical headers */
namespace App\Http\Controllers;
use Illuminate\Http\Request;                    // so we can grab data from input forms
use App\Http\Requests;
use App\Http\Controllers\Controller;
use Auth;                                       // for authentication

use DB;

use App\Note;                                   // our note table for eloquent, to enhance database operations
                                                // we will add Note.php shortly
use Illuminate\Support\Facades\Input;           // for input command, allowing us to grab input values imlicitly

class PublicController extends Controller {

   /* this function is called when the user hits the home page */
   public function create() {
      // authenticate the user, currently this app loads all the notes, this will not be ideal for large repos,
      if (Auth::check()) {
         $notes = DB::select('SELECT * FROM  `notes` ORDER BY created_at DESC, user_id');
          return view('public.index', compact('notes'));
      }
           return view('public.index');
   }
```

```
/* this will add a note for a user */
public function post(Note $note, Request $request) {
      // always (most of the time) authenticate the user
      if (Auth::check()) {                                    // check if post is from a logged in user
          $this->validate($request, [                         // validate the input, there is no point in adding a note with no
                  'title'   => ['required', 'min:3'],         // title or not content
                  'content' => ['required', 'min:3'],
              ]);

          $input = $request->only('title', 'content');
          $input = array_add($input, 'user_id', Auth::user()->id);
          $note->create($input);
          return back();
      } else {

          $errors = ['ERROR: You are not logged in, and are not authorized to add posts.'];
          return back()->withErrors(compact('errors'));
      }
   }
/* this function will update a note from a patch */
public function store(Request $request)  {
        // always (most of the time) authenticate the user
        if (Auth::check()) {
                $this->validate($request, [
                        'title'   => ['required', 'min:3'],
                        'content' => ['required', 'min:3'],
                ]);
                $input = $request->only('title', 'content', 'id');                  // grab the input data
                $input = array_add($input, 'user_id', Auth::user()->id);        // add the user id to the array

                DB::table('notes')                                              // using mass injection to update the table
                        ->where('id', $input['id'])
                        ->update($input);

                 return redirect('guests');                                     // redirect to guests AKA: home page
        } else {
                $errors = ['ERROR: You are not logged in, and are not authorized to add posts.'];
                return back()->withErrors(compact('errors'));
        }
   }
```

```
/* this function is called for an update, it retrieves data from the table, to fill the form with current data */
public function update($id) {
        // get the relevent note from the id selected by the authorized user
        $notes = DB::select('SELECT * FROM  `notes` WHERE id = '. $id);
        if ($notes) {
                $note = $notes[0];                              // the query returns an array, but we only have 1 note
                return view('public.edit', compact('note'));    // show the note edit/update form
        } else {
                $errors = ['ERROR: Post not found!'];           // unlikely to have not found a note with $id, but just in-case
                return back()->withErrors(compact('errors'));   // we will kick back to the calling page with an error
        }
    }
/* oh, crud the thing needs to be able to be delted too */
public function delete($id) {
        // only a user may delete a note
        if (Auth::check()) {
                $notes = DB::select("SELECT * FROM `notes` WHERE id = $id");
                // can only delete a note which exists
                if ($notes) {
                        $note = $notes[0];
                        // but only the author/owner of that note may delete it
                        If (Auth::user()->id == $note->user_id) {
                                // delete the note
                                DB::delete('DELETE from `notes` WHERE id ='.$note->id);
                                return back();                  //  this will also call create, so the data will be updated
                        } else {
                                // someone, though unlikely since there is no button, though a url could be entered, don't want
                                // one user to be able to delete anothers note
                                $errors = ['ERROR: You are not the owner of this note, you are not authorized to delete this post.'];
                                return back()->withErrors(compact('errors'));   // displays an error message to user
                        }
                } else {
                        $errors = ['ERROR: Post not found.'];
                        return back()->withErrors(compact('errors'));
                }
        } else {
                return back();
        }

    }
}
```

Now we can create a Note.php so we can use eloquent

Add a new file [***app/Note.php***], and enter the following code, ensure the $table name matches the table in the database in this case its 'notes'

```
<?php namespace App;
    use Illuminate\Database\Eloquent\Model as Eloquent;
    class Note extends Eloquent {
        protected $table = 'notes';
        // this allows exceptions for mass injections, which LARAVEL denies by default,
        // since mass injection of data can change data which should not be, such as passwords
        protected $fillable = [
            'title', 'content', 'user_id'
        ];

    }
```

Again update the **routes.php** file, so we can actually navigate to the views we will create in a moment, this will be the final edit of this file, the file should now contain: the routes containing { ??? } are used to pass request data to the controller.

```
<?php
    Route::get('/', 'PublicController@create');
    Route::get('guests', 'PublicController@create');

    Route::get('/notes/delete/{data}', 'PublicController@delete');
    Route::get('/notes/edit/{data}', 'PublicController@update');
    Route::patch('guests', 'PublicController@store');
    Route::post('/guests', 'PublicController@post');

    Route::get('/login', 'SessionsController@create');
    Route::get('logout', 'SessionsController@destroy');
    Route::post('login', 'SessionsController@store');
    Route::get('register', 'SessionsController@register');
    Route::post('register', 'SessionsController@newUser');
    Route::resource('sessions', 'SessionController');
    Route::patch('notes/edit/{data}', 'PublicController@store');
```

Our work is almost finished

All thats left is to update the routes and finish adding a couple more views, one for our home page AKA: guest page, and a page to edit notes. The guest page is also the place where the notes can be viewed and added

Before continuing make sure to update the route cache

update the routes cache:

```
~/workspace/simple-notes  $ php artisan route:cache
```

Create a folder in the views directory named [*public*] [*resources/views/public*] and add two files [*edit.blade.php* and *index.blade.php*]

views/public/index.blade.php

Create the home page/Guest page view

```
@extends('layout')

        @if($errors->any())
           <div class='error_msg'>
              @foreach ($errors->all() as $error)
                 <p>{{$error}}</p>
              @endforeach
           </div>

       @endif

@section ('content')

       <?php if (Auth::check()) { ?>

           <h2>Welcome <strong>{{ Auth::user()->name }}</strong></h2>

           <div class="code_block">
              <a href="/simple-notes/public/logout">log out</a>

                 @foreach($notes as $note)
                  <div class="code_block sub_block {{ (Auth::user()->id == $note->user_id)?'owners_select':'' }}">
```

```
                <h2>{{ $note->title }}</h2>
                <p>{{ $note->content }}</p>
                <div class='time_stamps'>
                    created:<span> {{ $note->created_at }}&nbsp;</span>
                    updated:<span > {{ $note->updated_at }} </span>
                     <?php if (Auth::check()  && Auth::user()->id == $note->user_id) { ?>
                       <span>&nbsp;[&nbsp;<a href="notes/delete/{{$note->id}}">delete</a>&nbsp;]</span>
                       <span>&nbsp;[&nbsp;<a href="notes/edit/{{$note->id}}">edit</a>&nbsp;]</span>

                    <?php } ?>
                </div>
            </div>
        @endforeach

         <div class="code_block sub_block owners_select">
              <form method="POST" action="" >
                   <input type=hidden name="_token" value="{{ csrf_token() }}">
               <p>
                   <h2 style="margin-left:-20px;">Create a new note</h2>
                   <input type="text" id="this_title" name="title" placeholder="new note title">
               </p>
               <p>
                   <textarea rows="6" name="content"  placeholder="new note content"></textarea>
               </p>
               <input type="submit" value="Submit">
            </form>
        </div>

    <?php } else { ?>

        <h2>Welcome <strong>Guest</strong></h2>

            <div class="code_block">

                     This is an academic site, creating a simple note-pad;
                     To edit the notepad you must be <a href="/simple-notes/public/login">logged in</a>

            <?php } ?>
    </div>

@stop
```

We can test the site now, it should show a new homescreen with a public message

To log in, we can use either a member which we included when we seeded the database tables, or create a new user. I chose to register a new user, to test the registration form at the same time.

When a new user is registered successfully, the user will be redirected to the login page, where he/she can add the newly created credentials to login. Upon a successful login, the user will be redirected to the guest AKA: home page, where we will see some pre-seeded notes from the three users already present.

In addition, a new note can be entered, and it will appear at the top of the page, as it is the most recent addition to the database. Though we delete it, we can not edit the note yet, because we have not created the view for it. We can create the final view, and our project is done for now....

views/public/edit.blade.php

Edits/updates the content of a selected note

```
@extends('layout')

@section ('content')

    <?php if (Auth::check()) { ?>
        <div class="code_block">
            <h1> EDIT! {{ $note->title }} </h1>
            <p1> {{ Auth::user()->name }} </p1>

          <form method="POST" action="">
               <div class="code_block sub_block {{ (Auth::user()->id == $note->user_id)?'owners_select':'' }}">
                      <input name="_method"type="hidden" value="PATCH">
                      <input type=hidden name="_token" value="{{ csrf_token() }}">
                      <input hidden name="id" value="{{ $note->id }}">
                  <p>
                      <label for="name">title:</label>
                      <input type="text" id="this_title" name="title" value="{{ $note->title }}">
                  </p>
                  <p>
                      <textarea rows="6" name="content">{{ $note->content }}</textarea>
                  </p>
                  <input type="submit" value="Update">
                  <a href="{{ URL::previous() }}">Go Back</a>
               </div>
          </form>
        </div>
    <?php } ?>
@stop
```

HU-RA!! WE ARE DONE!!!