Homework 2: Web application development exercise
Due date: 9/21/2016

Intro:

Homework 2 is divided into 2 parts:

1). Basic set up + Tutorial
2). Your assignment

We will be making a web application using Laravel framework (PHP).

Note: Part 1 contains a tutorial that can help you to get familiar with Laravel Framework. You can skip the tutorial if you want to

Let's get started!! PART 1

1. **Step 1: Download required files:**
   a. Install VirtualBox (or Virtual Machine)
      i. Link-> https://www.virtualbox.org/wiki/Downloads
   b. Download the virtual disk image (VDI) (Ubuntu 15.04)
      i. Link-> https://drive.google.com/a/usc.edu/file/d/0B8jgGk2u7XBVeUZJRXl0ZEZUeHM/view?usp=sharing

For MacOSX, I use "Archive Utility" to extract it.

** I already pre-installed Apache, MySQL, Php and other required packages for you.*

2. **Step 2: Setup virtual machine**
   a. Start VirtualBox → Click "New" → Type in "CS577HW2" in the name textbox → Choose "Linux" for "Type" → Choose → "Ubuntu 64-bit" for "Version" → Click "Continue"
   b. Slide the slider to your desired memory size (Recommended 2048mb) → Click "Continue"
   c. Select "Use an existing virtual hard disk file" → browse to the virtual disk image → Click "Create"
   d. Select "CS577HW2" virtual machine on the left panel to start up the machine.
   e. Password for the machine and all "sudo" commands is **std.**

Steps below are to be done in the Virtual Machine!!!

3. **Step 3: AllowOverwrite for Apache2**
   a. Open up Terminal
   b. Type in ➔ sudo a2enmod rewrite
   c. Type in ➔ sudo nano /etc/apache2/apache2.conf
   d. Scroll down to find this line:

```
<Directory /var/www/>
        ...
        AllowOverride None
        ...
</Directory /var/www/>
```

   e. Change **None** to **All**

f. Exit and save (In case you don't know: http://askubuntu.com/questions/477603/how-to-save-a-file-using-nano)
g. Restart Apache2 by running this command in terminal
    sudo systemctl restart apache2

4. **Create a dummy database for the tutorial**
    a. Open up Firefox → access http://localhost/phpmyadmin
    b. Username = root, password = root
    c. Create "tutorialdb" database (*or any database name that you want*)

5. **Install Composer for Laravel**
    a. Open up Terminal
    b. Type in → curl –sS https://getcomposer.org/installer | php
    c. Type in → sudo mv composer.phar /usr/local/bin/composer
    d. Type in → sudo chmod +x /usr/local/bin/composer

6. **Create a Laravel project through composer**
    a. In Terminal, type in → cd /var/www
    b. Type in → sudo chmod –R 777 html/
    c. Type in → cd html
    d. (*This step could take a while*) type in →
    composer create-project --prefer-dist laravel/laravel tutorialhw2
    *Note that "*tutorialhw2*" is your project name
    e. Go to the "*tutorialhw2*" directory (at /var/www/html/<projectname>) (<-- we will call it root)
    f. Config your Ubuntu to show hidden files
    g. Edit .env file in root
        i. Make sure that the app key is the same as the generated key listed at the end of step **d**'s result.
        ii. Config your database→ db_name = tutorialdb, user = root, pass = root.
    h. In Terminal, cd in to your tutorial directory (cd <projectname>)
    i. In Terminal, type in → php artisan migrate
        i. This should not return any errors if you set up your database information correctly.
        ii. The pre-made schemas that come with default laravel should automatically be migrated to the database (Check phpmyadmin if you can see "users" and "password_resets" tables)
    j. In Terminal, type in → sudo chmod –R 777 storage/

7. **Install Form/HTML helper**
    a. Follow the "installation" instruction: https://laravelcollective.com/docs/5.3/html

b. In the our root directory + Terminal, type in ➔ php artisan cache:clear
c. Open up firefox, access to http://localhost/<projectname>/public/
d. You should be able to see a white page with "Laravel 5" in the center as shown below.

8. **Now everything is set up, let's start building a simple app:**

We are going to create web app that displays a celebrity and movie(s) that he/she has been in. (let's assume each movie has only one celeb)

a. Let's first create tables that we will use in this app

i. In the root directory, open terminal and type in ➔
php artisan make:migration create_celebs_table --create="celebs"
and
php artisan make:migration create_movies_table --create="movies"

ii. Go to migrations folder (root/database/migrations), you will see our newly created files.

iii. Edit "create_celeb_table.php"

1. Add name and age of our celeb in the schema and save after we are done.

```
public function up()
{
    Schema::create('celebs', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name')->default('');
        $table->integer('age')->default('0');
        $table->timestamps();
    });
}
```

iv. Edit "create_movie_table.php"

1. We are going to add a foreign key linking the celeb table to the movie table (a celeb can be in many movies, right?).

```
    */
    public function up()
    {
        Schema::create('movies', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('celeb_id')->unsigned()->default(0);
            $table->foreign('celeb_id')->references('id')->on('celebs')->onDelete('cascade');
            $table->text('name')->default('');
            $table->timestamps();
        });
    }
```

v. In Terminal, type in ➔ php artisan migrate
vi. Check phpmyadmin if you can see both “movies” and “celebs” tables.

b. Since we don’t have data in those tables yet, we need to add data in. Two ways you can do this. (add through phpmyadmin or this way below)

i. Go to root/database/seeds
ii. Create 2 files: **CelebsSeeder.php** and **MoviesSeeder.php**
iii. In CelebsSeeder.php type in,

```
<?php

use Illuminate\Database\Seeder;

class CelebsSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('celebs')->delete();
        $celebs = array(
        ["id"=>1, "name"=>"Jennifer Lawrence","created_at" => new DateTime, "updated_at"=> new DateTime],
        ["id"=>2, "name"=>"Will Smith","created_at" => new DateTime, "updated_at"=> new DateTime],
        ["id"=>3, "name"=>"Matt Damon","created_at" => new DateTime, "updated_at"=> new DateTime]
        );
        DB::table('celebs')->insert($celebs);
    }
}
```

iv. In MoviesSeeder.php, type in

```php
<?php

use Illuminate\Database\Seeder;

class MoviesSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('movies')->delete();
        $movies = array(
        ["id"=>1, "celeb_id"=>1,"name"=>"The Hunger Game","created_at" => new DateTime, "updated_at"=> new DateTime],
        ["id"=>2, "celeb_id"=>1,"name"=>"X-Men","created_at" => new DateTime, "updated_at"=> new DateTime],
        ["id"=>3, "celeb_id"=>2,"name"=>"Suicide Squad","created_at" => new DateTime, "updated_at"=> new DateTime],
        ["id"=>4, "celeb_id"=>3,"name"=>"Good Will Hunting","created_at" => new DateTime, "updated_at"=> new DateTime],
        ["id"=>5, "celeb_id"=>3,"name"=>"Jason Bourne","created_at" => new DateTime, "updated_at"=> new DateTime]
        );
        DB::table('movies')->insert($movies);
    }
}
```

v. In DatabaseSeeder.php, add in

```php
    public function run()
    {
        // $this->call(UsersTableSeeder::class);
        $this->call('CelebsSeeder');
        $this->call('MoviesSeeder');
    }
```

vi. After everything is done, run this command in Terminal
composer dump-autoload
and
php artisan db:seed

vii. Let's check phpmyadmin if you can see those data in movies and celebs tables.

c. As parts of MVC pattern, we will need Models, Views, and Controllers. Let's create them.

i. Create Controllers.
   1. In root, type in to terminal:
      php artisan make:controller CelebsController
      php artisan make:controller MoviesController

ii. Create Models
   1. In root, type in to terminal:
      php artisan make:model Celeb
      php artisan make:model Movie

iii. Add resources to route (root/routes/web.php). Notice we are creating a nested route for celebs.movies since we want a user to go to http://localhost/<root>/public/celebs/{id}/movies to be able to see what movie that celeb has been in.

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

Route::resource('celebs','CelebsController');
Route::resource('celebs.movies','MoviesController');
```

iv. "resource" should automatically generate 7 routes for you: "create", "edit", "destroy", "update", "show", "store" and "index".

v. Check if your route works by running this command:
php artisan route:list
You should be able to see something like this:

```
+--------+----------+-------------------------------------------+------------+
|        | GET|HEAD  | /                                         |            |
Closure                                   | web        |
|        | GET|HEAD  | celebs                                    | celebs.index          | App\Http\Controllers
\CelebsController@index   | web        |
|        | POST      | celebs                                    | celebs.store          | App\Http\Controllers
\CelebsController@store   | web        |
|        | GET|HEAD  | celebs/create                             | celebs.create         | App\Http\Controllers
\CelebsController@create  | web        |
|        | PUT|PATCH | celebs/{celebs}                           | celebs.update         | App\Http\Controllers
\CelebsController@update  | web        |
|        | DELETE    | celebs/{celebs}                           | celebs.destroy        | App\Http\Controllers
\CelebsController@destroy | web        |
|        | GET|HEAD  | celebs/{celebs}                           | celebs.show           | App\Http\Controllers
\CelebsController@show    | web        |
|        | GET|HEAD  | celebs/{celebs}/edit                      | celebs.edit           | App\Http\Controllers
\CelebsController@edit    | web        |
|        | POST      | celebs/{celebs}/movies                    | celebs.movies.store   | App\Http\Controllers
\MoviesController@store   | web        |
|        | GET|HEAD  | celebs/{celebs}/movies                    | celebs.movies.index   | App\Http\Controllers
\MoviesController@index   | web        |
|        | GET|HEAD  | celebs/{celebs}/movies/create             | celebs.movies.create  | App\Http\Controllers
\MoviesController@create  | web        |
|        | DELETE    | celebs/{celebs}/movies/{movies}           | celebs.movies.destroy | App\Http\Controllers
\MoviesController@destroy | web        |
|        | PUT|PATCH | celebs/{celebs}/movies/{movies}           | celebs.movies.update  | App\Http\Controllers
\MoviesController@update  | web        |
|        | GET|HEAD  | celebs/{celebs}/movies/{movies}           | celebs.movies.show    | App\Http\Controllers
\MoviesController@show    | web        |
|        | GET|HEAD  | celebs/{celebs}/movies/{movies}/edit      | celebs.movies.edit    | App\Http\Controllers
\MoviesController@edit    | web        |
```

d. Let's define a relationship between two models.

i. **Recall that in our scenario "a celeb can be in many movies, but a movie can only have one celeb"**

ii. Go to (root/app) you'll see two models: **Celeb.php** and **Movie.php** that we just created.

iii. In **Celeb.php**, add in movies() function (Note: because of hasMany, the name of the function is plural)

```php
class Celeb extends Model
{
    //
        public function movies(){
                return $this->hasMany('App\Movie');
        }
}
```

iv. In **Movie.php,** add in celeb() function

```
class Movie extends Model
{
    //
        public function celeb(){
                return $this->belongsTo('App\Celeb');
        }
}
```

e. Let's create a function in each Controller:

i. Go to **CelebsController.php** (root/app/Http/Controllers/) **Don't forget to add** "use App\Celeb;" before the class definition because we are using the Celeb model.

*CelebsController.php ×

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Celeb;
class CelebsController extends Controller
{
    //
        public function index()
        {
                $celebs = Celeb::all();
                return view('celebs.index',compact('celebs'));
        }
}
```

ii. This function accesses the Celeb Model and fetch all() data in the celeb table and store in the $celebs variable.

iii. It then passes the variable "celebs" to "index" (View) in the folder called "celebs".

iv. Now let's edit **MoviesController.php** (root/app/Http/Controllers/). Since we already declared the relationship between the two models. Fetching celeb data will automatically fetch the celeb's movie data (see "**f**" part for how to get that movie data).

```php
use App\Celeb;

class MoviesController extends Controller
{
    //
        public function index($id){
                $celeb_name = Celeb::find($id);
                return view('movies.index',compact('celeb_name'));
        }
}
```

f. **Let's create Views (root/resources/views)**
   i. Create a folder called "celebs"
   ii. Create a folder called "movies"
   iii. Within each folder, create a file called "**index.blade.php**"

index.blade.php of celebs may contain something like this:

index.blade.php ×

```
<!DOCTYPE html>
<html>
    <head>
        <title>Laravel</title>
    </head>
    <body>
        <div class="container">
            <div class="content">
                <h2>A List of Celebs</h2>
                        <ul>
                                @foreach($celebs as $celeb)
                                    <li>
{{link_to_route('celebs.movies.index',$celeb->name,[$celeb->id])}}
</li>
                                @endforeach
                        </ul>
            </div>
        </div>
    </body>
</html>
```

link_to_route is used to create a link to "celebs.movies.index" route (see also route:list command). It shows $celeb->name and also passes an array containing $celeb->id to the "celebs.movies.index" route.
Basically it creates this link "http://localhost/<root>/public/celebs/{id}/movies"

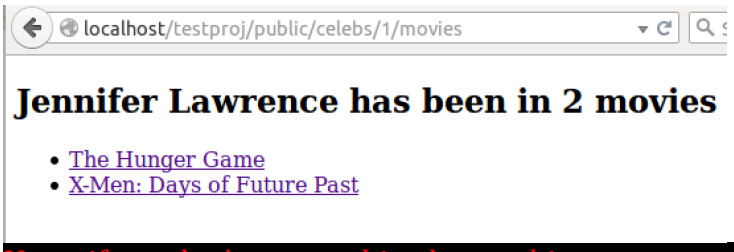
Index.blade.php of **movies** may contain something like this:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Laravel</title>
    </head>
    <body>
        <div class="container">
            <div class="content">
                <h2>{{$celeb_name->name}} has been in {{$celeb_name->movies->count()}}
movies</h2>
                <ul>@foreach($celeb_name->movies as $movie)
                        <li><a href="">{{$movie->name}}</a></li>
                @endforeach
                </ul>
            </div>
        </div>
    </body>
</html>
```

As you can see, we can get movie data for a certain celeb by calling "->movies" from the celeb data (i.e., $celeb_name->movies). If you want to fetch the data in the controller, see: https://laravel.com/docs/5.3/eloquent-relationships#one-to-many

Now, if you go to the site:
http://localhost/tutorialhw2/public/celebs
You should see something like this:

If you click at Jennifer Lawrence, you'll see something like this:



Note: if you don't see anything but a white page, not even an error message, try "**sudo chmod –R 777 storage/**" in terminal

Now that you have been exposed to Laravel, it's time to start on our actual homework.
Let's get started. PART 2.

::Now it's time for you to create a new project::

Scenario:

Due to the increase in the number of Pokémon trainers in your area, nurse Joy from the Pokémon center has contacted USC CSSE asking for help. She told us that she needs a Pokémon trainer management system. Would you be willing to help Joy? She gave us the requirements below:

Requirements:

1). A trainer should be able to login/logout/register to the system.

2). Nurse Joy, Professor Oak and other professors should be able to log in as an admin. (Note: *Admin should be able to do everything a normal user can*)

3). Once logged in:

a. A trainer should be able to see other trainers' *name*, *hometown*, *total numbers of Pokémon they have*, *names of Pokémon they have*, and *whether or not they are an admin*. This should be shown in the "Home" page. If the logged-in user is an admin, there should be an edit button for them so that they can do part **c**.
   i. If they don't have a profile yet, the "edit" for an admin button should not be shown.
   ii. If they don't have a profile yet, "N/A" should be shown in those related cells.

b. A trainer should be able to create their profile. (*Apart from Hint #3, you are not allowed to edit the pre-made User table.*)
   i. The link to "my profile" should be on top of "log out" button (under your name in the NavBar or Menu bar – as part of the dropdown menu)
   ii. Once in the profile page:
      1. If a trainer does not have a profile yet, the system should let them know.
      2. If a trainer has already created a profile, the system should show their information accordingly.
      3. The trainers should be able to edit their profile information (including name and email they input from the sign up).
   iii. The trainer should not be able to edit other trainer's profile.

c. Admin should be able to edit everyone's profile.

d. Admin should see the "admin" button by the Navbar (menu bar) so that he can access the admin page.
e. A trainer should not be able to access admin page, an error should be shown if they attempt to access the page.
f. Admin should be able to add/delete Pokémon to the system database in the admin page + an appropriate error/success message should be shown.
g. If a Pokémon already exists in the system, an admin should not be able to add it again (case-insensitive) + an appropriate error/success message should be shown.
h. In the admin page, the system should show a total numbers of Pokémon in the system, a list of all Pokémon, and how many trainers in the system have that particular Pokémon.
i. In the profile page, a trainer should be able to add Pokémon based on a list of Pokémon registered in the system by an admin (part f). (Preferably a drop down menu to select. The names on the drop down list should be in alphabetical order). They should also be able to remove their Pokémon.
    i. If they already added the Pokémon, they shouldn't be able add the same Pokémon again. An error message should be shown.
    ii. A success message should be shown if added/deleted successfully.
j. There should be a search bar in the homepage that can be used to search for Pokémon. If an entry exists, a trainer should be able to see names of trainers (Or just number of trainers) who have that Pokémon as well.

Hint:

1. See: https://laravel.com/docs/5.3/authentication for authentication scaffolding. One command and you're good to go with login/logout/registration.
2. See https://laravel.com/docs/5.0/templates for blade templates after you run authentication command. – Master page layout.
3. We do not expect you to implement roles/permission functionality for admin. Simply add "isAdmin" as a Boolean type in the User or Profile table and check for it somehow is acceptable.
4. You don't have to use "web" middleware, it has been applied by default for the new Laravel. Check php artisan route:list to see.
5. Don't assume that there's only one admin for the whole system. We will register another admin and try on your system. (i.e., **manually change the isAdmin data in the DB**)
6. You should not be using *raw query builder* in your controller.
7. See http://laravel-recipes.com/recipes/280/interacting-with-your-application if you want to make sure the data is returned or you fetch it correctly

8. Note that once you define the relationship between models, you can make use of simple query to get the data of one model from another model. See: https://www.laravel.com/docs/5.3/eloquent-relationships#one-to-many as an example.
9. See "Laravel flash message" on how to display err/success messages.
10. You can use Visual Paradigm (http://greenbay.usc.edu/csci577/fall2016/tools) or https://www.draw.io/ , or other tools that you are familiar with to generate your ER-Diagram.

Useful Resources:

Model Relationship: https://laravel.com/docs/5.3/eloquent-relationships
HTML and Form syntax: https://laravelcollective.com/docs/5.3/html
Routing: https://laravel.com/docs/5.2/routing
Task Example (with Create, delete): https://laravel.com/docs/5.2/quickstart
Authentication: https://laravel.com/docs/5.3/authentication

Submission guideline:

What you should submit:

1. ** **Please put your first name, last name, USC ID in the welcome page (welcome.blade.php)**
2. Source code (the whole project).
3. README.txt (You should include all your assumptions, external resources that you use, and/or your design decision)
4. Your ER-Diagram.
5. Screenshots of your website.
6. Your DB (export the db as "***HW2.sql***")

Submit everything in a .zip file on D2L (**LASTNAME_FIRSTNAME_HW2.zip**). Sign up for an in-person grading slot with the TAs or graders (*we will let you know*). Bring your own laptop, make sure your application is ready to go. During the grading session, we may ask you a couple of questions and have you modify some codes.

Grading Criteria:

- -50% if you do not use Laravel framework.
- Functionalities
- All forms and link should be in a correct format (https://laravelcollective.com/docs/5.3/html for link and form)
- Have a clear understanding of MVC (we can judge it by your code)
- Correctly define the relationship between models.
- ER diagram. (If hand-drawn, it should be legible)
- Screenshots.
- -15% penalty per late day.

Note: Though you will not be graded based on the look of your website, I will give extra points if you make it look good (e.g., add images, change theme, use jquery, use ajax, etc.).