# Views:

**Login:**

* Input for user and password + login button
* LINK : “Forgot Password?” password reset
* LINK : Signup

**Signup:**

* Pass

**Forgot Password:**

* pass

**User Home:**

* view the skills graph
* view skill analysis
* view recommended events

**User Profile:**

* be able to change user f

**User Home:**

**User Home:**

# Controllers:

d

# Models (Database Tables):

* User table : need this to store basic user info, such as username and password. Probably also some sort of id.
* User history table of some sort, that constantly tracks updates to a

Important commands:

Composer dump-autoload

Php artisan make:model -mc (makes a migration and controller… followed by adding field in migration file and doing “php artisan migrate”)

# Laracasts Overview:

First we have something called the web routes file. This is located in the “routes/web.php”. The route essentially maps urls to views. Basically we say: “if the user visits ‘/tasks/poop’, show them *this* page. If they visit ‘/’, show them *this other* page. We specify what page by referencing a view file to show, as well as the controller that prepare the view.

Then we have the views themselves, which are located in *resources > views*. Views are literally HTML files with PHP code inserted in, which is used to dynamically generate certain parts. Laravel has a custom syntax for this, because it uses something called the blade templating engine (all views are of type “blade.php”). The idea behind this templating engine is that you can use php variables in the blade.php file. But you need to pass these variables in when serving up the view to the client.

We pass variables to the view using the Controller, which are located in app>http>Controllers. These controllers interact with a model, gather the required arguments and then return a view and the needed arguments.

11: We set up the create post page. To do this, we make a view called create.blade.php. Then, we add a form to it, bootstrap style. What it has are the following: two text areas, with name=”body”, name=”title”, method = “POST” and action=”/posts” and also a button with type submit. The idea behind it falls back to MVC: We must set a web route that responds to POST requests from /post, specifying a controller and a method that handles it. In that method (it’s called store() by convention), we create a new record, accepting the title and body (as specified by the name fields from the inputs), and then saving them, and then redirecting them to another page.

Now some security considerations that Laravel handles.

To protect against cross site request forgery (when someone else performs actions on behalf of an authenticated user), we add {{csrf\_field()}} to the top of the form, which generate a token and stuff. Handled by Laravel though, no need to worry about the details.

Consider the fact someone could alter the HTML and add an input field like timestamp. On submit, this will now

12: Still on the form. To prevent the user from submitting null in forms and just clicking submit, we must use HTML5 browser validation (use “required” key word in inputs). Obviously we need some serverside validation too.

For this, in our store() method that creates the record, we validate the post by using the building validation system: you call $this->validate(request(), [ ‘title’ => ‘required | max:10’ ]);. As you can see, for each field in the In the input, you can add rules separate by pipes. See the docs for more rules. Now, if an invalid input is provided, we just redirect to the same page, (a refresh, basically). It also creates a populated errors variable, $errors. This is available for use in the view, which we can foreach loop through to display the errors in a div, if there are any. In the video, he adds a red colored div around this div which persists even If he refreshes. To get rid of this, we add @if(count($errors)) … @endif, putting the printed errors div in the middle there.

13: Here, we load the posts from the database and display them in a wall. Luckily, this is as easy as fetching all the posts from the db ($post = POST::all()) and then returning a view with the $posts as a parameter.

Here we use the partial from resources to create a post, post.blade.php. Then we went through it and used the passed in $post variable as we should: for the title and body e.g. {{$post->body}}

We also used Carbon to format the timestamp method: {{$post->created\_at->toFormattedDateString()}}.

19 Associating with Users:

* We want to make login and register pages, so we create routes for them. Specifically, we make a controller for each, and add create() methods:  
  Route::get(“/register”, “RegistrationController@create”)   
  Route::get(“/login”, “SessionsController@create”)
* Then we actually make the controllers:   
  php artisan make:controller SessionsController  
  php artisan make:controller RegistrationController
* tro