CSE 3244 Information System Design Lab

Project Documentation

Collab.

170104029	Shafayet-Ul Islam
170104030	Shahriar Hasan Chowdhury
170104047	Rishadul Islam Khan

Collab Documentation

Collab is a platform where students and teachers around the world can communicate and share their knowledge. Collab takes a person's interests and based on those, it curates data that a person might like.

Collab uses html5, css3, sass, bootstrap, vanilla.js, Vue.js as UI tools and for backend it uses Laravel framework. These tools are very mature to develop complex web applications such as Collab.

Backend documentation is much needed to understand the core part of the project. Laravel is kind enough to give us much more flexibility to think about the core parts and isolate the boilerplate functionalities of the project that every developer needs to implement.

Laravel uses MVC (Model-View-Controller) architecture. So much of the documentation will be circled around this.

Models:

There are 7 models in our project. These models gives us the access of the data that are stored in the database. These models also handles the logic behind the data that can be fetched, updated, deleted, inserted in the database. Fortunately Laravel gives us flexibility to modify these logics.

These models can be found in /app directory.

Each of the models describes one table from the database. If we convert the model name to plural form, the table name will be shown.

These models are listed below:

- User.php
- Role.php
- Interest.php
- Newsfeed.php
- Message.php
- Project.php
- Research.php

These models can also make relations with each other as stated in the message model

Message.php

```
<?php
```

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class Message extends Model
{
    protected $guarded = [];

    public function fromContact()
    {
        return $this->hasOne(User::class, 'id', 'from');
    }
}
```

FromContact() method makes a relation between message and user model. $guarded variable expresses that all the attributes related to this model are open to change other then the primary keys.

User.php

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Laravelista\Comments\Commenter;

class User extends Authenticatable
{
    use Notifiable, Commenter;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password', 'designation', 'institution_name',
        'role_id', 'country', 'about', 'profile_image', 'cover_image',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    public function role()
    {
        return $this->hasOne('App\Role');
    }
}
```

$fillable variable is used to make use that only the attributes that are mentions in this array can be manipulated.

$hidden variable is used to hide the attributes that can cause insecure transection.

role() function is used to make a relation between Users and Role model.

Research.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Research extends Model
{
    protected $table = 'researches';
    protected $guarded = [];
}
```

$table variable gives us an option to change the name of the table that defines the research model. We needed to change the table name for the model. That's why we used it.

All the other models are coded with these type of coding techniques or options that Laravel provides. I hope explanations of these three models are adequate enough to understand the rest of the models.

Controllers:

Controllers are used to make a bridge between models are views. Controllers control all the incoming requests that need to be served.

Laravel has provided us built in controller for login and registration system. But we need to modify it to serve out purposes as we have implemented a role system to this project.

There is a folder name Auth where the built in controllers resides.

This folder can be located at /app/Http/Controllers directory of the project.

To serve our purpose, we did not need to modify the controllers but the routing. That will be discussed in routing section.

Besides this, we have eight custom controllers to fulfill our needs. These controllers are listed below:

- ContactsController.php
- HomeController.php
- InitialMessageController.php
- NewsfeedController.php
- InterestController.php
- ProfileController.php
- ProjectController.php
- ResearchController.php

ContactsController.php

```
class ContactsController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function get() { }

    public function getMessagesFor($id) { }

    public function send(Request $request) { }
}
```

public function get()

This method curates other users' data based on the authenticated user's interest. An array is made with the users id. These array of users will be shown in the message UI. the users that have sent or received message from the authenticated user will be fetched from the database also with their respective users and sent to the message UI by this method. This method is very important as it dictates the users that will be able to contact the authenticated user.

In response, this methods sends all the contacts that we need to show in the message UI.

public function getMessageFor($id)

This method receives a parameter named $id. This parameter is used to detect the user that has sent a message to the authenticated user.

This method is used to keep track of the messages that are being read and also fetch the messages that the user has sent to the authenticated user or the messages the authenticated user has sent to the user.

The $id variable contains the users id that we can use to show the messages.

In response, this methods sent all the messages that are collected from the database.

public function send(Request $request)

This method receives a parameter of Request data type. This $request parameter contains the submitted request body that is sent by the authenticated user.

In our case $request contains the user id of the authenticated user that has sent a message and the message text.

With these data, the method updates the database with the correct form and returns the message as we need to show the sent message.

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

InitiateMessage.php

```
class InitiateMessage extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }
    public function sendMessage(Request $request){ }

}
```

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

public function sendMessage(Request $request)

This method receives a parameter named $request of type Request that contains the message that the authenticated user has sent to the desired user from visiting the desired user's profile.
The $request variable contains the desired user id and the message text that need to send by the authenticated user.

Then this method save the message in the message table in the database then also broadcast the message to the desired user so that the desired user can respond quickly.

HomeController.php

```
class HomeController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }
    public function index() { }

}
```

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

public function index()

This method does nothing but to return out home view.

ProfileController.php

```
class ProfileController extends Controller
{

    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index($id) { }

    public function editAbout() { }

    public function edit($id) { }

    public function updateInfo(\App\User $user) { }

    public function updateAbout(\App\User $user) { }

}
```

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

public function index($id)

This method receives a parameter $id that denotes the user id. This user id is then used for fetching the the researches, projects, interest and the basic information of the user from the database. If an authenticated user is trying to visit his own profile or some other users' profile he/she found, the $id parameter will contain the user's id. This method also return a view with the user data, researches, projects, interests of the user connected to the user id.

public function editAbout(\App\User $user)

This method does nothing but returning a view that can be used to edit the users about. Only authenticated users will be able to access this method by surfing his/her own profile.

public function updateAbout(\App\User $user)

This method accepts a parameter named $user of type User. This method is called when user has submitted a form containing user's about edit page.

The form sends a POST request to this form and the method accepts it only if the user is authenticated and update the about of the authenticated user.

After updating, this method redirects to the profile landing page with the user id.

public function edit($id)

This method accepts a parameter named $id that contains the user id. This parameter contains the user id who is going to edit his/her basic information. This method returns a view with the user's basic information so that the user can edit the information.

public function updateInfo(\App\User $user)

This method accepts a parameter $user of User type. After editing the user's basic information, the user need to submit a form. The form submit results in this method and the parameter containing the Users information.

The form sends a POST request to this method and is accepted by this only if the user is authentic and update the database with the user's edited information in the database.

After updating the database with user's information, the method redirects to the profile landing page with the user id.

InterestController.php

```
class InterestController extends Controller
{
    protected $except = [];

    public function __construct()
     {
         $this->middleware('auth');
     }

    public function index() { }

    public function post(Request $request) { }

    public function get($id) { }

}
```

InterestController contains a variable named $except. This variable is declared so that we can edit a functionality of this controller. We wanted to disable CSRF token form submission for this controller. So, we added this variable to InterestController class.

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

public function index()

This method does nothing but returning the landing page of the Interest.

public function post(Request $request)

This method accepts a parameter named $request of type Request. This method accept POST request which contains the user_id and the selected interests by the authenticated user. That means $request parameter contains the user_id and the selected interests.

After getting the user_id and the selected interest, this method saved the interests in the interests table in database using the user id.

This method redirects to the profile of the authenticated user containing the user name as the parameter.

public function get($id)

This method accepts a parameter named $id containing the user id. With this user id this method finds all the interests related to the user id and returns the data as a JSON file. If the related data is not found, this method returns 404 error, so that the client can understand it.

ResearchController.php

```
class ResearchController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function addResearch() { }

    public function postResearch() { }

    public function readResearch($user_id, $id) { }

    public function deleteResearch($user_id,$id) { }

}
```

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

public function addResearch()

This method does nothing but returning the view of the edit research page.

public function postResearch()

This method accepts POST request sent by the user by using a form declared in the addResearch view. With this method the user is able to add research to his/her profile.

This method receives the authenticated user's research data, validates it and then saves the data in researches table according to the user.

This method then redirects to the authenticated users profile with the user's id with it.

public function readResearch($user_id, $id)

This method gets two parameter named $id with the proper research id and $user_id containing the user id that has posted the research.

This method returns the view of readResearch page with the proper research data related to the research id from the database.

public function deleteResearch($user_id, $id)

This method gets two parameter named $id with the proper research id and $user_id containing the user id that has posted the research. This method finds the research related to the research id and deletes it from the database.

This method then returns the profile view page with the authenticated user id.

ProjectController.php

```
class ProjectController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function addProject() { }

    public function postProject() { }

    public function readProject($user_id, $id) { }

    public function editProject($user_id,$id) { }

    public function updateProject($id) { }

    public function deleteProject($user_id,$id) { }

}
```

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

public function addProject()

This method does nothing but returning the view of addProject page containing a form.

public function postProject()

This method accepts a post request sent by the authenticated user using the addProject view page. The method extracts the POST request and gets all the values related to the post the user wants to add, validate the incoming data and saves it in the projects table in the database according to the user.

This method redirects to the authenticated users profile with the authenticated users id.

public function readProject($user_id, $id)

This method accepts two parameters named $id containing a project id that the user wants to see and a $user_id containing the user_id that has posted the project.

This method finds the project linked to the project id and returns a view containing the project data.

public function editProject($user_id, $id)

This method accepts two parameters named $id containing a project id and a $user_id containing the user_id that has posted the project.

This method finds the project and return a view containing a form with the project data so that the user can edit the data.

public function updaeProject($user_id, $id)

This method accepts two parameters named $id containing the selected project id and a $user_id containing the user_id that has posted the project.

This method accepts POST request sent by the user by using a form declared in the editProject view. With this method the user is able to edit project to his/her profile.

This method receives the authenticated user's project data, validates it and then saves the data in researches table according to the user.

This method then redirects to the authenticated users profile with the user's id with it.

public function deleteResearch($id)

This method gets a parameter named $id with the proper project id. This method finds the project linked to the project id and deletes it from the database.

This method then returns the profile view page with the authenticated user id.

Newsfeedcontroller.php

```
class NewsfeedController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index() { }

    public function post() { }

    public function getPosts($id) { }

    public function edit($id) { }

    public function saveEdit($id) { }

    public function deletePosts($id) { }

}
```

public function __construct()

This is the constructor method of the Message controller. This method is used add or remove built in option provided by laravel. In our case, we have added a functionality that only the authenticated users will be served by this controller.

public function index()

This method curates other users' data based on the authenticated user's interest. An array is made with the interests of the authenticated user. Then the array is used to get selected users based on the interests from the interests table.

In this way we are getting all the users that are following the interests the authenticated user is following.

Now this method is just returning the users post that are related to the authenticated users interest and the index view of the newsfeed.

public function post()

This method accepts POST request from the authenticated user using a form that is on the newsfeed index view. with the new post related data it creates a new entry in the database with the data that is sent by the post request. This method accepts a status and a image to make a post for the authenticated user.

public function edit($id)

This method accepts one parameter named $id containing the selected post id

This method finds the post and return a view containing a form with the post data so that the user can edit that post.

This method return a view name newsfeed.editPost.

public function saveEdit($id)

This method accepts one parameter named $id containing the selected post id.

This method accepts POST request sent by the user by using a form declared in the editOwnPost view. With this method the user is able to update his/her own post.

This method redirect user to newsfeed.index view.

public function deletePost($id)

This method accepts one parameter named $id containing the selected post id.

This method accepts POST request sent by the user by using a form declared in the editOwnPost view. With this method the user is able to delete his/her own post.

This method redirect user to newsfeed.index view.

MiscController.php

```
class MiscController extends Controller
{
    public function signupPage(){
    }

    public function loginPage(){
    }

    public function checkRole($role){
```

```
    }
}
```

public function signupPage()

If the user is authenticated this method will return a view page which is provided by the index method of newsfeed controller, otherwise this method returns the signup page so user can create their account.

public function loginPage()

If the user is authenticated, this method will return a view page which is provided by the index method of newsfeed controller, otherwise this method returns the login page so user can login into his or her account.

public function checkRole($role)

This method accepts a parameter named $role that contains the role_id that the user will be assigned to.

This method will return the sign up view depending on the role that the user has assigned to.

Views

Collab web application will not function properly without the views. This property makes the web application live and interactive with it's users.

To design our views, we have used **HTML5, CSS3, SASS, Bootstrap, JavaScript etc**. There is a view folder in out app located at "/resources/views" directory of our project.

Some parts of out project like "Messenger" is built with vue.js, a front end javascript framework because messenger system is much more complicated that other parts of out web application.

These files are located at "/resources/assets/js" directory. **Vue.js** is used because we need real time update functionality which is very hard to implement in pure HTML5, CSS3 or JS.

Other assets like images, CSS files are added in the public directory of web application. All of these files are linked together with one another.

Routes

Routes are controlled by a separate file in Laravel. In this file we can link our controllers and views with the desired route. If we want to customize the default routes of our web application, this is the file to edit.

web.php

```
<?php
use Illuminate\Http\Request;
/*
|--------------------------------------------------------------------------
```

```
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', "MiscController@signupPage")->name('signup');

Route::get('/login/home', "MiscController@loginPage")->name('login_home');
Route::get('/registerpage/{role}', "MiscController@checkRole")->name('signup_valid');

Auth::routes();

Route::get('/message', 'HomeController@index')->name('home');



// Profile Controller Routes
Route::get('/profile/{id}', 'ProfileController@index')->name('profile.index');
Route::get('/profile/{id}/edit', 'ProfileController@edit')->name('profile.edit');
Route::patch('/profile/edit/{user}', 'ProfileController@updateInfo');
Route::get('/profile/{id}/edit/about', 'ProfileController@editAbout')->name('profile.edit.about');
Route::patch('/profile/edit/about/{user}', 'ProfileController@updateAbout');



// Research Controller Routes
Route::get('/profile/{id}/add/research', 'ResearchController@addResearch');
Route::post('/profile/{id}', 'ResearchController@postResearch');
Route::get('/profile/{user_id}/research/{id}', 'ResearchController@readResearch');
route::get('/profile/{user_id}/delete/research/{id}', 'ResearchController@deleteResearch');
Route::get('profile/research/download/{id}', 'ProfileController@downloadPDF')->name('downloadFile');



// Project Controller Routes
Route::get('/profile/{user_id}/add/project', 'ProjectController@addProject');
Route::post('/profile/{user_id}/add/project', 'ProjectController@postProject');
Route::get('/profile/{user_id}/project/{id}', 'ProjectController@readProject');
Route::get('/profile/{user_id}/edit/project/{id}', 'ProjectController@editProject');
Route::patch('/profile/edit/project/{id}', 'ProjectController@updateProject');
Route::get('/profile/{user_id}/delete/project/{id}', 'ProjectController@deleteProject');


//Interest Controller Routes
//Route::get("/api/interest", "InterestController@index");
Route::post("/api/interest/post", "InterestController@post");
Route::get("/api/interest/get/{id}", "InterestController@get");
Route::get("interest", "InterestController@index");


// Newsfeed
Route::get("newsfeed", "NewsfeedController@index")->name('newsfeed.index');
Route::post("/{user}/post", "NewsfeedController@post")->name('save_status');
// Route::get("/{user}/post", "InterestController@index")->name('save_status');
Route::post("saveEdit/{postID}", "NewsfeedController@saveEdit")->name('newsfeed.saveEdit');
Route::get("edit/{user}", "NewsfeedController@edit")->name('newsfeed.edit');
Route::get("posts/{user}", "NewsfeedController@getPosts")->name('newsfeed.getPosts');
Route::get("delete/posts/{postID}", "NewsfeedController@deletePosts")->name('newsfeed.deletePosts');




/// chat
Route::get('/contacts', 'ContactsController@get');
Route::post('/conversation/start', 'InitiateMessage@sendMessage')->name('initiate.send');
```

```
Route::get('/conversation/{id}', 'ContactsController@getMessagesFor');
Route::post('/conversation/send', 'ContactsController@send');
```

This piece of the code shows us that we can add all the routes and their respective controllers in this file. Even which route is going to serve which HTTP request can re added here.

We can even pass variables in the routes so that when the route sends request to the corresponding controller described in the file can receive this variables as parameters. With these parameters we can curate data for each users.

All the routes that are added here can be discovered by user. Any routes other than these are going to send a HTTP code **404 error**.

This is the file where we can bring out the functionality of our code with the proper routes accessible by the users.

Broadcast System

Messenger system need to have a broadcast system to live update the other users' messenger view. That's why we need to add a broadcast channel in our application.

NewMessage.php

```
class NewMessage implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $message;

    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct(Message $message)
    {
        $this->message = $message;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return \Illuminate\Broadcasting\Channel|array
     */
    public function broadcastOn()
    {
        return new PrivateChannel('messages.' . $this->message->to);
    }

    public function broadcastWith()
    {
        $this->message->load('fromContact');

        return ["message" => $this->message];
    }
}
```

This class is located at "/app/events" directory. This class initiates the Message broadcast system.

All the methods and other elements are provided by Laravel framework.

BroadcastServiceProvider.php

```
class BroadcastServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Broadcast::routes();

        require base_path('routes/channels.php');
    }
}
```

To provide a broadcast, we need to make a broadcast service provider that can provide it.

This method returns the base routes that Laravel controls internally and the condition that it will be broadcasted or not.

Channels.php

```
Broadcast::channel('messages.{id}', function ($user, $id) {
    return $user->id === (int) $id;
});
```

This static method returns the condition of the broadcast to send the data across other users.

Broadcasting needs live servers that we need to configure. In this case, we need to use some third party server.

In this case, we have used pusher as our broadcast driver.

To add this funtionaliy we need to edit the .env file in out web application.

.env

```
BROADCAST_DRIVER=pusher

PUSHER_APP_ID={app_id}
PUSHER_APP_KEY={app_key}
PUSHER_APP_SECRET={secret_key}
PUSHER_APP_CLUSTER={cluster}
```

Third bracket included keys can be added by visiting the pusher website. After creating an account and also an app, these credentials can be obtained.

After all of this, these lined of code should be added to the .env file with the correct credentials.

Broadcast system will work if all of this are done one by one.

Miscellaneous things

To develop Collab, we have used Xampp as php environment and Mysql for our database drivers. **Matching drivers and version number is needed to run this project on local machine**.

This version numbers can be found in **composer.json** file in the root directory.

For our javascript dependencies, **Node and NPM** must be installed in the machine.

At last Xampp, Mysql, Node and NPM must be present in the local machine.

Conclusion

Collab is a platform that was designed for sharing knowledge and ideas. Our team strongly believe that, this platform can create a bridge between students and teachers around the globe with its highly interactive user experience, beautiful graphical interface and breathtaking fully functional features.