PHP – Hypertext Processor and frameworks: a LARAVEL based Web application

Contents

1. Scope

This project has the scope of learning and understanding techniques and concepts especially used in web applications, as well as the way other programming languages can be adapted and used exclusively for this kind of applications.

Therefore, in the scope of creating a software web application, the following components were used: Hypertext preprocessor language (also world widely known as PHP programming language), a language-specifically framework named Laravel and one of the mostly used software architecture pattern for these types of software application, the MVC architecture (Model-View-Controller). As this application is considered to implement user-friendly interfaces, this type of architecture will help achieving this goal.

In order to understand better the interaction between al of the components previously described, in the Chapter 2 below they will be separately presented, as well as the communication flow between each one of them, in order to achieve the scope of creating a User Interface Web Application.

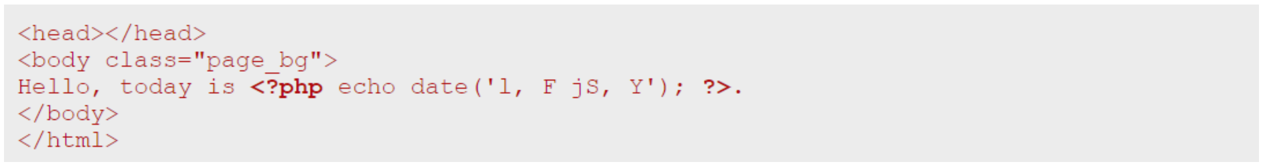
2. Theoretical concepts

As previously presented, there are three big components which this project is based on.

- Hypertext Preprocessor language (PHP)

  PHP is an HTML-embedded web scripting language. This means PHP code can be inserted into the HTML of a Web page (see Figure 1). When a PHP page is accessed, the PHP code is read or "parsed" by the server the page resides on. The output from the PHP functions on the page are typically returned as HTML code, which can be read by the browser. Because the PHP code is transformed into HTML before the page is loaded, users cannot view the PHP code on a page. This make PHP pages secure enough to access databases and other secure information. (Techterms-definitions-php)

  A lot of the syntax of PHP is borrowed from other languages such as C, Java and Perl. However, PHP has a number of unique features and specific functions as well. The goal of the language is to allow Web developers to write dynamically generated pages quickly and easily. PHP is also great for creating database-driven Web sites, as this project is based on.



```
<head></head>
<body class="page_bg">
Hello, today is <?php echo date('l, F jS, Y'); ?>.
</body>
</html>
```

Figure 1

- Model View Controller (MVC) architecture

  MVC is a software architecture pattern, commonly used to implement user interfaces, therefore it is a popular choice for architecting web applications. In general, it separates out the application logic into three separate parts, promoting modularity and ease of collaboration and reuse. It also makes applications more flexible and welcoming to iterations. (leon-earl, 2018)

  Basically, MVC enforces separation between the information (MODEL), user's interaction with information (CONTROLLER) and visual representation of information (VIEW) (see MVC flow in Figure 2).

- MODEL

The model defines what data the app should contain. If the state of this data changes, then the model will usually notify the view (so the display can change as needed) and sometimes the controller (if different logic is needed to control the updated view).

Usually, in a web application, the DATABASE represents our MODEL. Therefore, a model can add/retrieve items from the database, process the data and it communicates only with the CONTROLLERS.

The golden rule of the MVC includes the model part and it implies that the model does NOT depend on the controller or the view.

- VIEW

The view defines how the app's data should be displayed, being the only thing, the user ever sees. It can be seen as a good old-fashioned HTML/CSS.

View communicates only with the CONROLLERS, as the controllers tell the view what to do and view never response back, just listens to controllers and display the data in a user-friendly way.

- CONTROLLER

The controller contains logic that updates the model and/or view in response to input from the users of the application (e.g. button clicks).

Controllers process GET/POST/PUT/DESTROY requests with a server-side logic. A controller is also known as "The Middle Man", as it takes the information from the user, processes the information and communicates with the database (MODEL) if needed, receives the information from the database and speaks to the VIEW to explains and expose presentation to the client. In conclusion, controller depends on the view and the model

There are some cases where it is needed to just update the view to display the data in a different format, e.g., change the item order to alphabetical, or lowest to highest price. In this case the controller could handle this directly without needing to update the model. (Dalling, 2009)

o Advantages of the MVC architecture

→ Faster development process
MVC supports rapid and parallel development. With MVC, one programmer can work on the view while other can work on the controller to create business logic of the web application.

→ Ability to provide multiple views
In the MVC Model, you can create multiple views for a model. Code duplication is very limited in MVC because it separates data and business logic from the display.

→ Support for asynchronous technique

→ Modification does not affect the entire model
Model part does not depend on the views part. Therefore, any changes in the Model will not affect the entire architecture.

→ MVC model returns the data without formatting
MVC pattern returns data without applying any formatting so the same components can be used and called for use with any interface. (mvc-advantages-disadvantages-mvc)

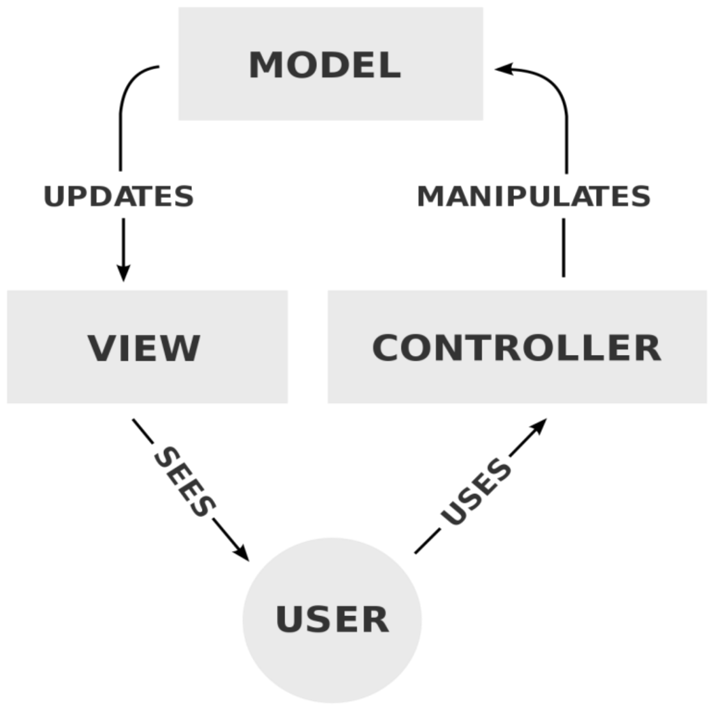


Figure 2

- <u>Laravel – PHP-based framework</u>

Laravel is a free, open-source PHP web framework created for the development of web applications following the Model-View-Controller (MVC) architectural pattern. Some of the features of Laravel are a modular packaging system with a dedicated dependency manager, ways for accessing relational-only databases and utilities for application deployment and maintenance.

Laravel is considered to be a full stack web development framework, meaning it can handle every aspect of a web application architecture – from storing and managing data using the database wrapper to displaying user interfaces using its own templating engine. Therefore, Laravel attempts to ease the common tasks used in the majority of web projects, such as authentication, routing, sessions and caching. (Surguy)

Beside the fact that Laravel is one of the popular PHP frameworks for web applications, it is also well known for its rapid development capabilities.
In Figure 3 it can be seen the File Manager which separates tasks in folders.

As Laravel is based on MVC architecture, the most used files are representing the models, views and controllers, and each one of them can be found in the following files:

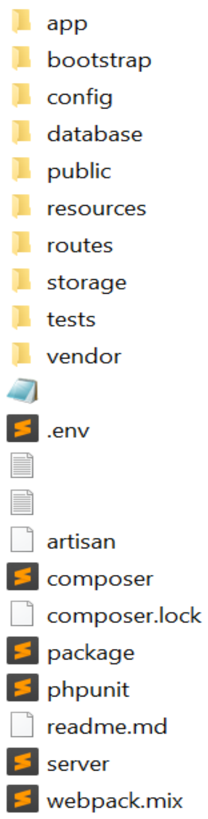
1. app → Console = create new commands for artisan to do new things in cmd
   → Events
   → Exceptions
   → HTTP → CONTROLLERS = main logic for the application
   → Middleware = for processing information before each HTTP request ; before info goes to the ROUTES file
   → Requests
   → kernel.php
   → "ModelName".php = MODELS, the interface for the database
   → routes.php/ web.php → paths to reach each view

2. bootstrap = core files of the Laravel, cache, app.php, autoloadphp
3. config = select features for the application (url, timezone etc)
4. database →factories
   → migrations = allow to create DB structures to be replicated
   → seeds = seed the DB- add information for testing purposes

5. public = all elements accessible for the client (images, index.php)
6. resource → assets

   → lang

   → VIEWS = html pages
7. tests
8. vendor = source code for Laravel
9. env files = setting files for different environments (for development, testing etc)



```
app
bootstrap
config
database
public
resources
routes
storage
tests
vendor

.env


artisan
composer
composer.lock
package
phpunit
readme.md
server
webpack.mix
```

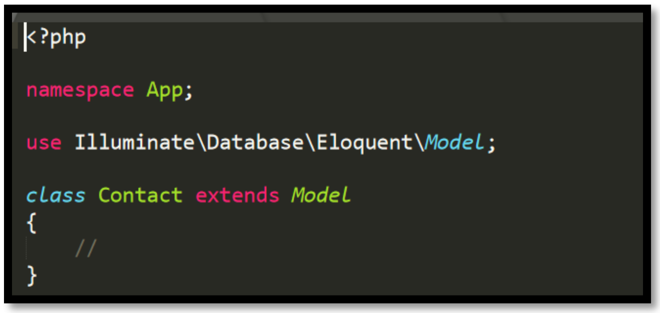
Figure 3

3. Practical examples

For showing how this MVC architecture works with Laravel, an example using the “Contact for feedback” function from the Web Application developed within this project will be presented.

- <u>MODEL part</u>

In the Laravel file manager, it has been seen that MODELS are located in the APP folder. There, it is present the Contact model, which is described in **Error! Reference source not found.** .

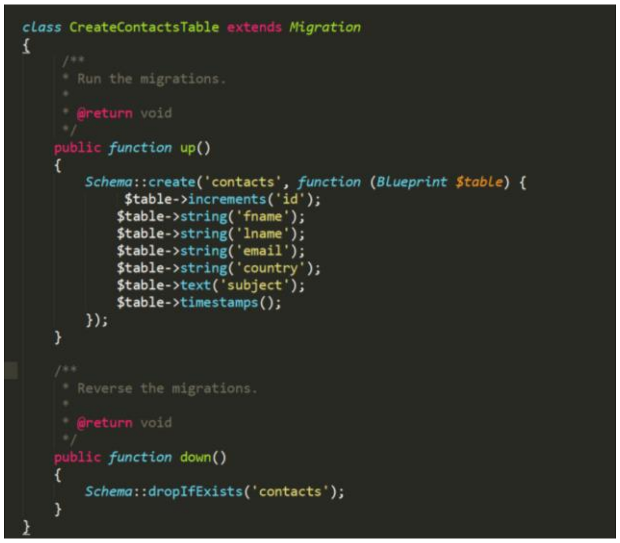


```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Contact extends Model
{
    //
}
```

Figure 4. Contact Model

The Contact model is created by the following command executed in the console:
  php artisan make: model Contact –migration.

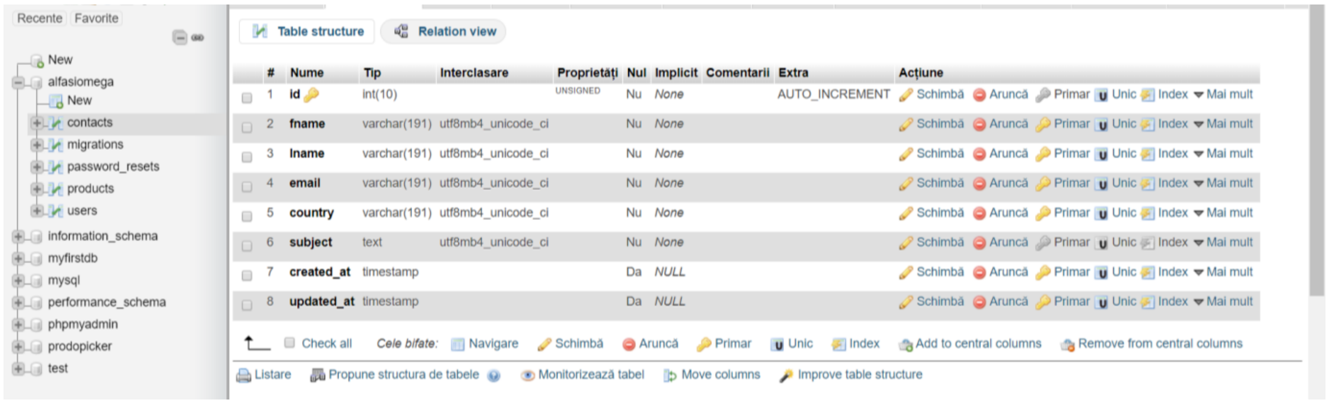
After executing this command, in the database file should appear a PHP file in which we configure the fields that are requested and needed for the Contacts table and all the information regarding the table Contacts will be migrated in order to create the table (Figure 5).



```
class CreateContactsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('contacts', function (Blueprint $table) {
             $table->increments('id');
            $table->string('fname');
            $table->string('lname');
            $table->string('email');
            $table->string('country');
            $table->text('subject');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('contacts');
    }
}
```

Figure 5. Contacts table to be migrated

Launching the following command in the console, it will create the table in project's database (mysql selected as default) with all the fields configured previously file: php artisan migrate. Now we have set the MODEL part in which the data will be stored (Figure 6).



Recente Favorite

New
alfasiomega
New
contacts
migrations
password_resets
products
users
information_schema
myfirstdb
mysql
performance_schema
phpmyadmin
prodopicker
test

Table structure | Relation view

#	Nume	Tip	Interclasare	Proprietăți	Nul	Implicit	Comentarii	Extra	Acțiune
1	id	int(10)		UNSIGNED	Nu	*None*		AUTO_INCREMENT	Schimbă Aruncă Primar Unic Index Mai mult
2	fname	varchar(191)	utf8mb4_unicode_ci		Nu	*None*			Schimbă Aruncă Primar Unic Index Mai mult
3	lname	varchar(191)	utf8mb4_unicode_ci		Nu	*None*			Schimbă Aruncă Primar Unic Index Mai mult
4	email	varchar(191)	utf8mb4_unicode_ci		Nu	*None*			Schimbă Aruncă Primar Unic Index Mai mult
5	country	varchar(191)	utf8mb4_unicode_ci		Nu	*None*			Schimbă Aruncă Primar Unic Index Mai mult
6	subject	text	utf8mb4_unicode_ci		Nu	*None*			Schimbă Aruncă Primar Unic Index Mai mult
7	created_at	timestamp			Da	*NULL*			Schimbă Aruncă Primar Unic Index Mai mult
8	updated_at	timestamp			Da	*NULL*			Schimbă Aruncă Primar Unic Index Mai mult

Check all *Cele bifate:* Navigare Schimbă Aruncă Primar Unic Index Add to central columns Remove from central columns

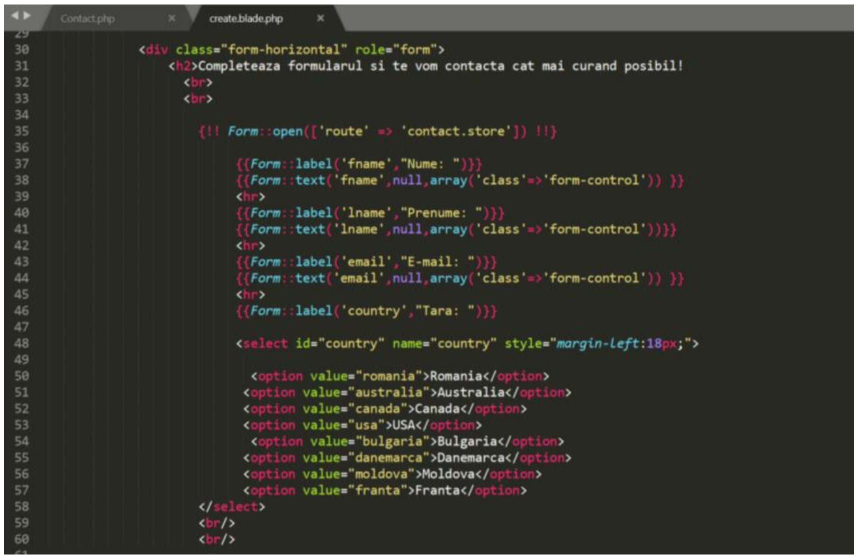
Listare Propune structura de tabele Monitorizează tabel Move columns Improve table structure

Figure 6. Contacts table in DB

- <u>VIEW part</u>

For the view part, it has been created 2 files regarding the Contact component, create.blade.php and show.blade.php, containing HTML components, but including also styles and features by importing CSS, JS and bootstrap files.

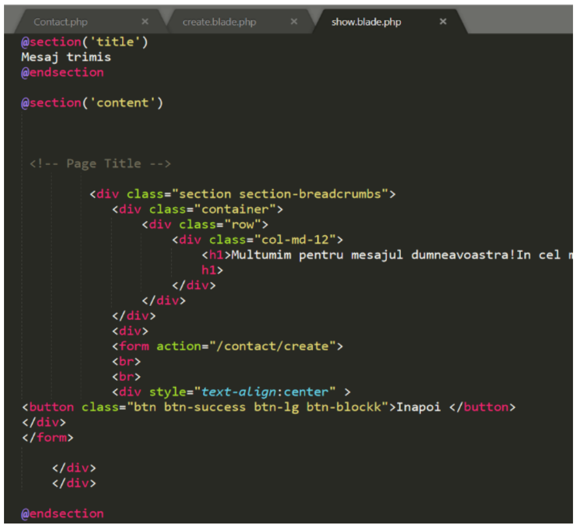
Create.blade.php contains the form which must be completed in order to send the information to the database (figure 7).



```
Contact.php   ×   create.blade.php   ×
            <div class="form-horizontal" role="form">
                <h2>Completeaza formularul si te vom contacta cat mai curand posibil!
                  <br>
                  <br>

                    {!! Form::open(['route' => 'contact.store']) !!}

                         {{Form::label('fname',"Nume: ")}}
                         {{Form::text('fname',null,array('class'=>'form-control')) }}
                         <hr>
                         {{Form::label('lname',"Prenume: ")}}
                         {{Form::text('lname',null,array('class'=>'form-control'))}}
                         <hr>
                         {{Form::label('email',"E-mail: ")}}
                         {{Form::text('email',null,array('class'=>'form-control')) }}
                         <hr>
                         {{Form::label('country',"Tara: ")}}

                         <select id="country" name="country" style="margin-left:18px;">

                           <option value="romania">Romania</option>
                          <option value="australia">Australia</option>
                          <option value="canada">Canada</option>
                          <option value="usa">USA</option>
                           <option value="bulgaria">Bulgaria</option>
                          <option value="danemarca">Danemarca</option>
                          <option value="moldova">Moldova</option>
                          <option value="franta">Franta</option>
                    </select>
                    <br/>
                    <br/>
```

Figure 7. create.blade.php

Show.blade.php contains the page with the message displayed after the form was submitted correctly (figure 8).



```
Contact.php   ×   create.blade.php   ×   show.blade.php   ×
@section('title')
Mesaj trimis
@endsection

@section('content')



 <!-- Page Title -->

        <div class="section section-breadcrumbs">
           <div class="container">
               <div class="row">
                   <div class="col-md-12">
                       <h1>Multumim pentru mesajul dumneavoastra!In cel m
                       h1>
                   </div>
               </div>
           </div>
           <div>
           <form action="/contact/create">
           <br>
           <br>
           <div style="text-align:center" >
<button class="btn btn-success btn-lg btn-blockk">Inapoi </button>
</div>
</form>

    </div>
    </div>

@endsection
```

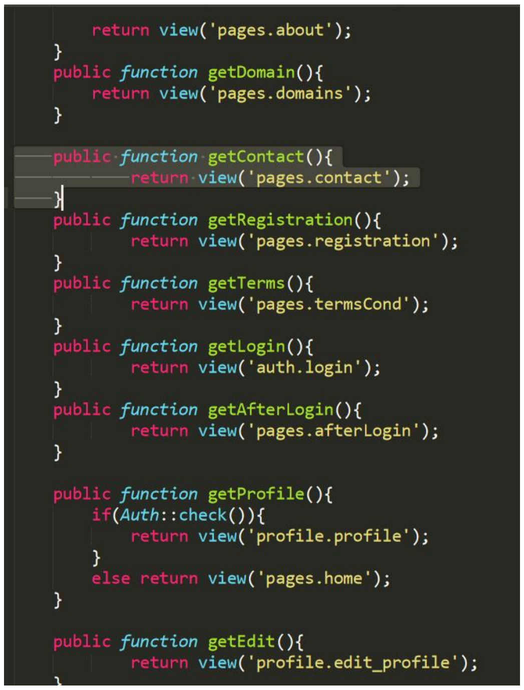
Figure 8. show.blade.php

- <u>CONTROLLER part</u>

The controller part implies also 2 files, one according strictly to the Contact pages and the logic that is implemented for the contact side, and another one regarding the general way of accessing and general functionalities for all the pages included in the web Application.

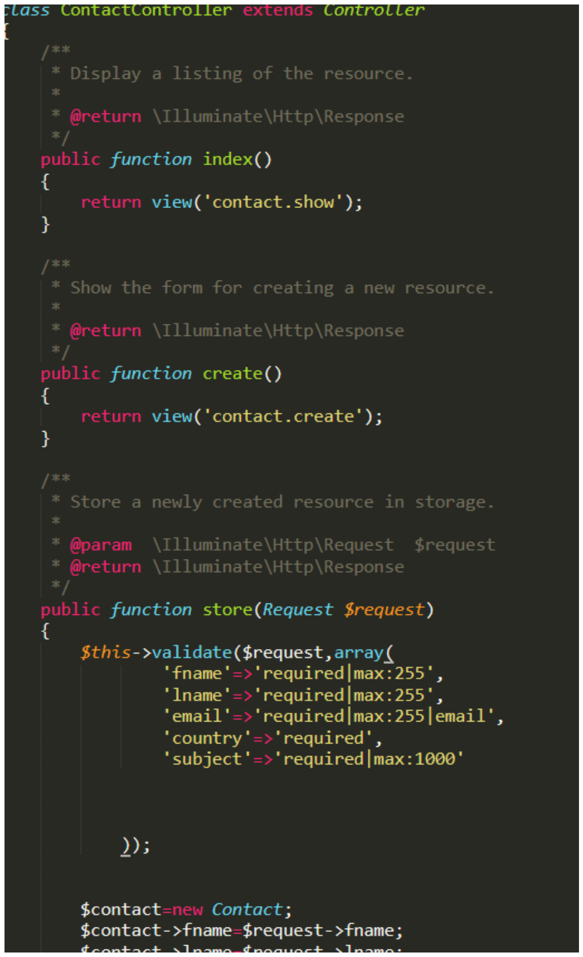
In the pagesController there are implemented the functions in which paths to the views are configured and set (Figure 10).

In the ContactController there are implemented functions regarding the operation which is going to be run in order to make the connection between the Model and the View (Figure 10).



```
        return view('pages.about');
    }
    public function getDomain(){
        return view('pages.domains');
    }

    public function getContact(){
            return view('pages.contact');
    }
    public function getRegistration(){
            return view('pages.registration');
    }
    public function getTerms(){
            return view('pages.termsCond');
    }
    public function getLogin(){
            return view('auth.login');
    }
    public function getAfterLogin(){
            return view('pages.afterLogin');
    }

    public function getProfile(){
        if(Auth::check()){
            return view('profile.profile');
        }
        else return view('pages.home');
    }

    public function getEdit(){
            return view('profile.edit_profile');
```

Figure 9.pagesController

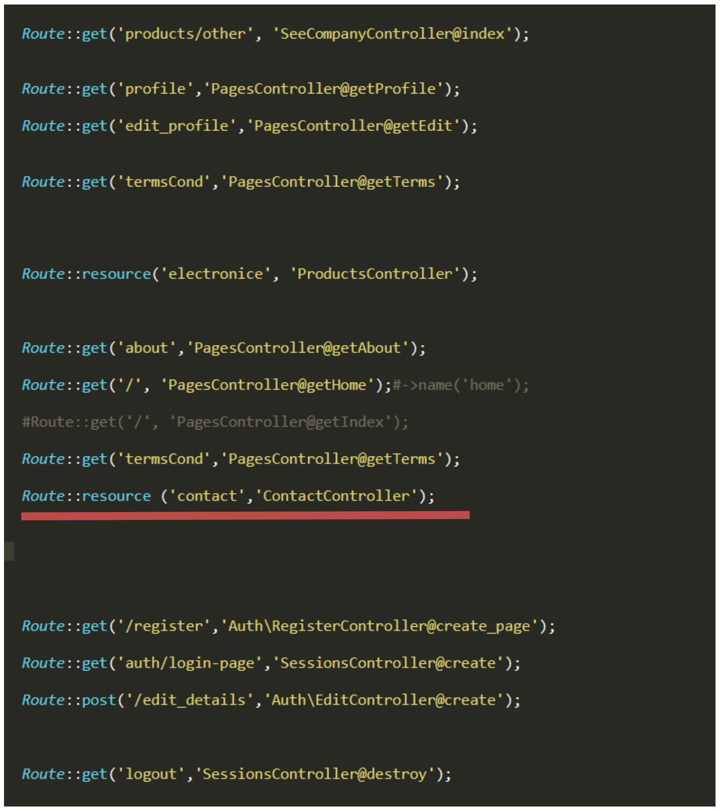


```
class ContactController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        return view('contact.show');
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        return view('contact.create');
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $this->validate($request,array(
                'fname'=>'required|max:255',
                'lname'=>'required|max:255',
                'email'=>'required|max:255|email',
                'country'=>'required',
                'subject'=>'required|max:1000'



            ));


        $contact=new Contact;
        $contact->fname=$request->fname;
```

Figure 10. ContactController

- <u>COMMUNICATION part</u>

In order these three parts to communicate, a routing file is needed to be configured. Therefore, it exists the web.php file in which the routes and paths for getting to each page is configured (Figure 11).



```
Route::get('products/other', 'SeeCompanyController@index');

Route::get('profile','PagesController@getProfile');
Route::get('edit_profile','PagesController@getEdit');

Route::get('termsCond','PagesController@getTerms');


Route::resource('electronice', 'ProductsController');


Route::get('about','PagesController@getAbout');
Route::get('/', 'PagesController@getHome');#->name('home');
#Route::get('/', 'PagesController@getIndex');
Route::get('termsCond','PagesController@getTerms');
Route::resource ('contact','ContactController');



Route::get('/register','Auth\RegisterController@create_page');
Route::get('auth/login-page','SessionsController@create');
Route::post('/edit_details','Auth\EditController@create');


Route::get('logout','SessionsController@destroy');
```

Figure 11. web.php

4. Conclusions

Combining the PHP language with Laravel framework based on MVC architecture, the project created represents a Web Application with an User-friendly interface whose scope is to interact with the users and offer them a better view of an Electronics Components Store, where they can register, authenticate, compare prices and other offers regarding electronics from a virtual Catalog and also to send feedback or any other suggestions regarding the offers.

Using the Model-View-Controller architecture not only helped the User visualize in a friendly and fast way the pages of the web site, but also helps the development side to handle the application in a much easier way and also it gives less harder times when it comes to maintaining the project or to test it.

Bibliography

1. Dalling, T. (2009, may 31). *model-view-controller-explained.* Retrieved from software-design: https://www.tomdalling.com/blog/software-design/model-view-controller-explained/

2. leon-earl, j. R. (2018, June 8). *Modern_web_app_architecture.* Retrieved from MVC_architecture: https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture

3. *mvc-advantages-disadvantages-mvc.* (n.d.). Retrieved from Interserver: https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/

4. Surguy, M. (2014). *Laravel - my first framework.* Lean Publishing.

5. *Techterms- definitions-php.* (n.d.). Retrieved from https://techterms.com/definition/php