Project Title: TRexe

Developed by: Sravanthi Kapu

Introduction: In this website we can navigate to Sales page, by choosing and entering few options in a form, we can get the data related to the listing available in database. On clicking the view details button we can see the full details of the specific listing and can also contact seller by clicking the contact seller button in details page. To contact seller, we need to fill the form and by clicking the send email button an email is sent to the seller with the details we gave in the form.

Technologies:

Laravel Framework.

HTML, CSS, and PHP to develop the web pages using Laravel and MySQL database to store all required information. XAMPP server is easily managing the databases over PhpMyAdmin and PHP.

About Laravel:

Laravel is an open source, PHP web Framework based on MVC (Model-View-Controller) architectural pattern.

Blade Template Engine: Laravel has the Blade templating engine which works quite easily with typical PHP/HTML. We can use different type of widgets of JS and CSS. We can create layouts and use them where ever needed in different views easily.

As it supports MVC, our file structure and documentation becomes easy and organized.

Artisan: Laravel offers a build in the tool named as Artisan. In Laravel, developer needs to interact using a command line that handles the Laravel project environment, which allows user to perform lengthy programming tasks really quickly. It is used to create a structured code, and database structure to make it easier to manage the database system.

Apart from these Laravel have many other in built object oriented libraries and features which makes developing easy and quick.

Database:

Seller Table has all the data related to seller.

Listing Table has all the data related to the listing. As every listing has a seller and those details can be accessed from sellers table, there is a foreign key constraint between seller and listing table where seller id (Foreign key) in listing table references to seller id (Primary Key) in seller table.

Selerrev Table has all the reviews of the sellers. As every review is related to some or the other seller, there is a foreign key constraint between seller and sellerrev table where seller id (Foreign key) in sellerrev table references to seller id (Primary Key) in seller table.

The migrations are created using the below commands in command terminal:

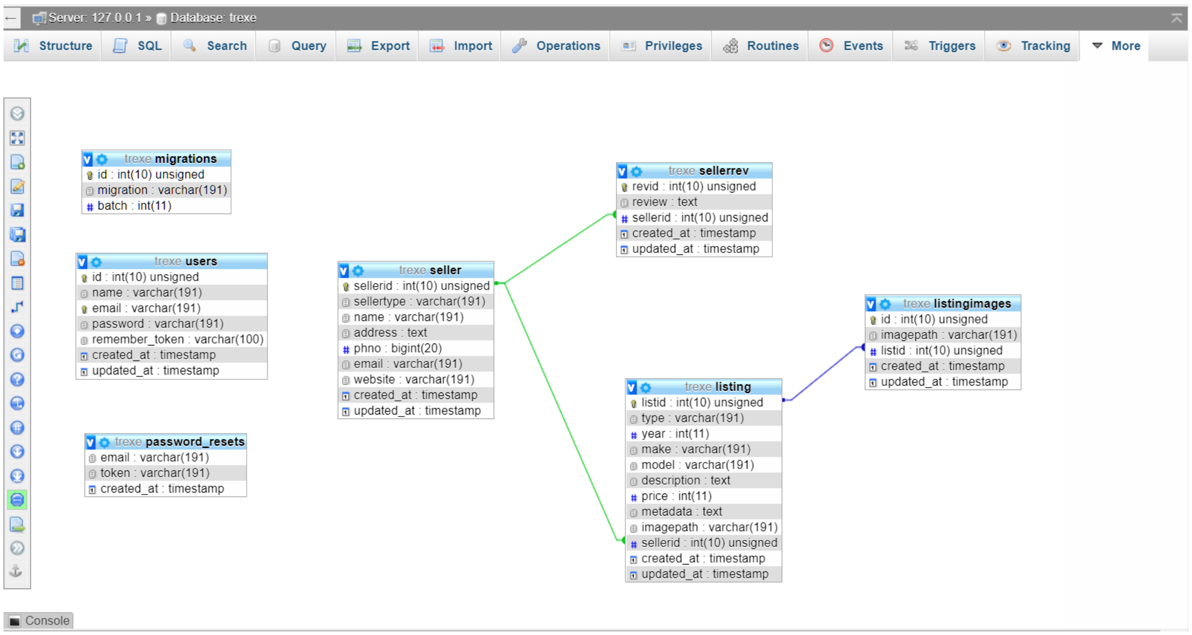
```
php artisan make:migration create_seller_table --create=seller
php artisan make:migration create_listing_table --create=listing
php artisan make:migration create_sellerrev_table --create=sellerrev
```

The timestamp of the migration is noted in migrations.

Since there are dependencies on the tables, we need to create seller migration first and the then rest so, we can avoid errors related to foreign key dependencies. Then need to define the schema of the table in each migration. Set the username, password in the .env file in order to give permissions for the user. Then by running the below command we can create tables defined in the migrations in our database.

```
php artisan migrate
```

I then populated my tables through phpmyadmin.



trexe migrations
id : int(10) unsigned
migration : varchar(191)
batch : int(11)

trexe users
id : int(10) unsigned
name : varchar(191)
email : varchar(191)
password : varchar(191)
remember_token : varchar(100)
created_at : timestamp
updated_at : timestamp

trexe password_resets
email : varchar(191)
token : varchar(191)
created_at : timestamp

trexe seller
sellerid : int(10) unsigned
sellertype : varchar(191)
name : varchar(191)
address : text
phno : bigint(20)
email : varchar(191)
website : varchar(191)
created_at : timestamp
updated_at : timestamp

trexe sellerrev
revid : int(10) unsigned
review : text
sellerid : int(10) unsigned
created_at : timestamp
updated_at : timestamp

trexe listing
listid : int(10) unsigned
type : varchar(191)
year : int(11)
make : varchar(191)
model : varchar(191)
description : text
price : int(11)
metadata : text
imagepath : varchar(191)
sellerid : int(10) unsigned
created_at : timestamp
updated_at : timestamp

trexe listingimages
id : int(10) unsigned
imagepath : varchar(191)
listid : int(10) unsigned
created_at : timestamp
updated_at : timestamp

HTML and CSS: (layouts)

In Laravel, we can use layouts. To reduce the time used for UI of the webpages, I used an existing Layout and modified it according to my design. So all the related to CSS and image files are placed in the public folder.

File Structure and Functionality Implementation

Command used to create project:

```
composer create-project -prefer-dist laravel/laravel TRexe
```

Commands to run our project:

```
composer install
```

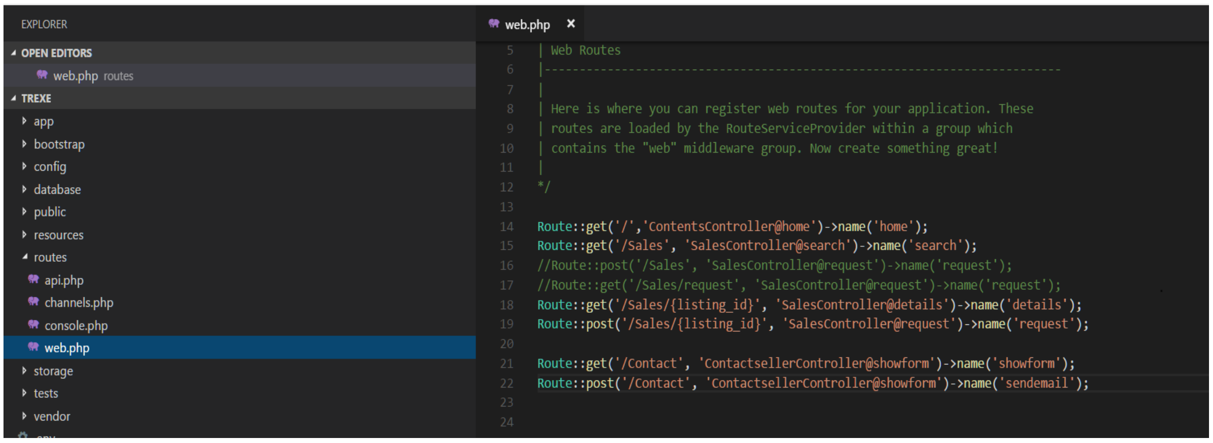
This installs the composer. (Used only when we are accessing our project later after creating)

```
php artisan serve
```
This starts the server, so we can access our webpages in the browser.

File Structure:

Routes defined in web.php in routes folder. Here the, URL to be loaded, the method to be used by specific controller for displaying a specific view are defined here.

Few of the files are created but not used. Those which are used for functionality are explained.



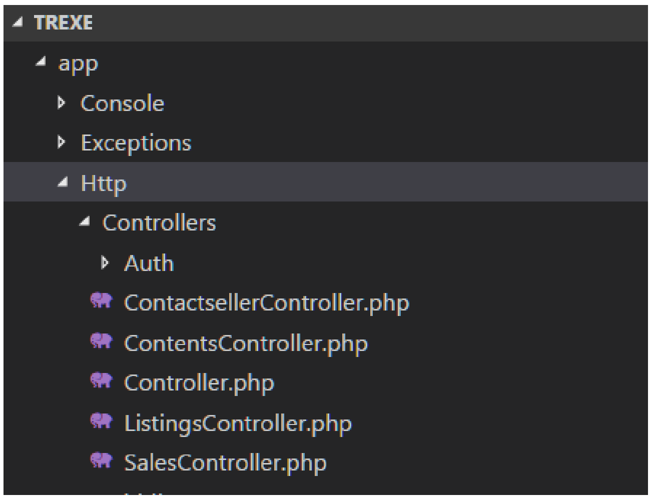
```
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!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

Route::get('/','ContentsController@home')->name('home');
Route::get('/Sales', 'SalesController@search')->name('search');
//Route::post('/Sales', 'SalesController@request')->name('request');
//Route::get('/Sales/request', 'SalesController@request')->name('request');
Route::get('/Sales/{listing_id}', 'SalesController@details')->name('details');
Route::post('/Sales/{listing_id}', 'SalesController@request')->name('request');

Route::get('/Contact', 'ContactsellerController@showform')->name('showform');
Route::post('/Contact', 'ContactsellerController@showform')->name('sendemail');
```

Controllers: Controllers can group related route logic into a class, as well as take advantage of more advanced framework features such as automatic dependency injection.

All the functions related to home page are in ContentsController.

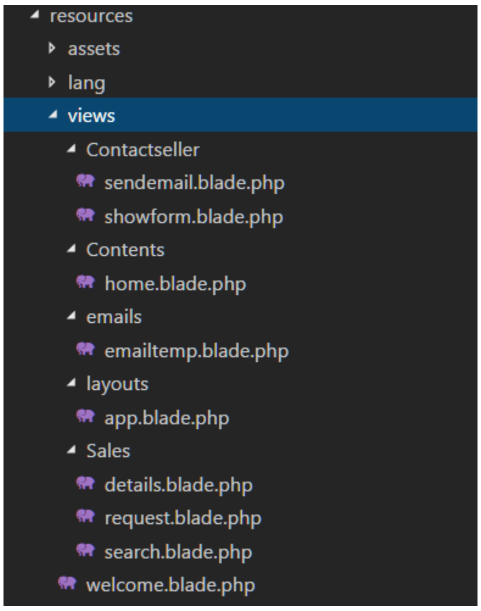
All the functions related to Sales page and view details are in SalesController.

All the functions related to contacting seller are in ContactsellerController.



The flow and the methods in controller will be explained in detail in later part of the document when explaining the functionality of each page.

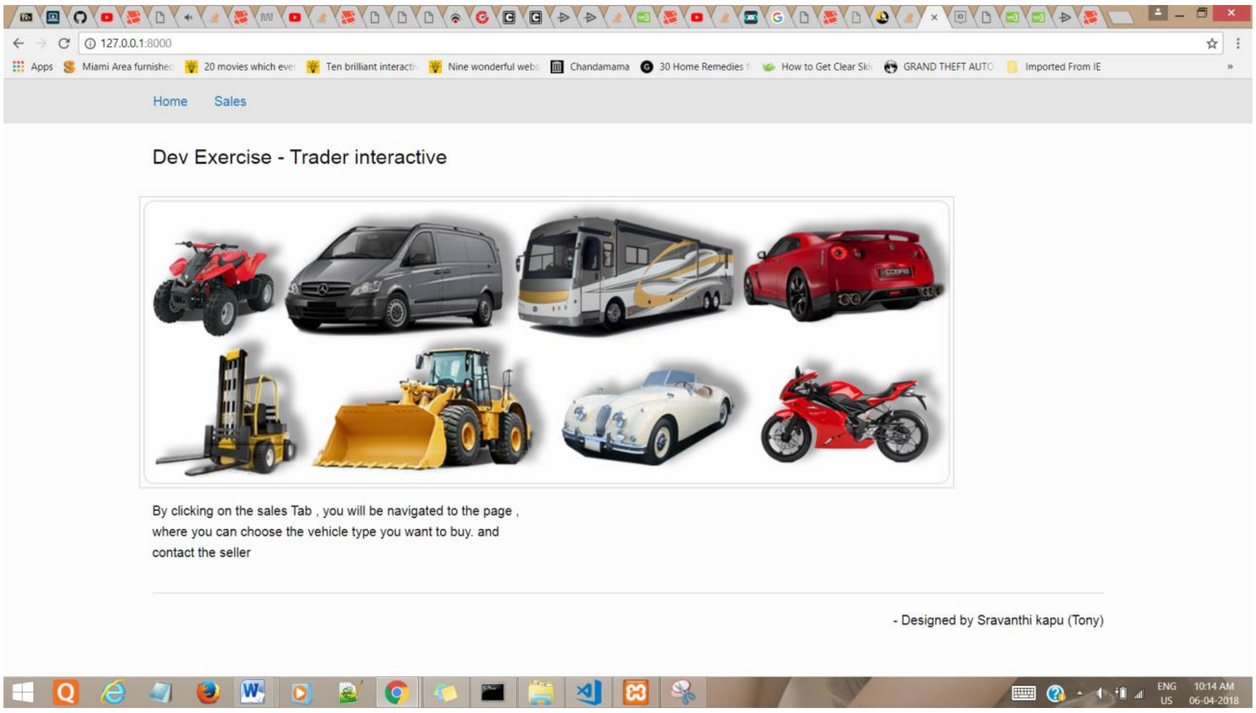
Views: Views contain the HTML served by your application and separate your controller / application logic from your presentation logic. All the layouts and the blade.php files of all the pages are in views.



```
◢ resources
  ▷ assets
  ▷ lang
  ◢ views
    ◢ Contactseller
        sendemail.blade.php
        showform.blade.php
    ◢ Contents
        home.blade.php
    ◢ emails
        emailtemp.blade.php
    ◢ layouts
        app.blade.php
    ◢ Sales
        details.blade.php
        request.blade.php
        search.blade.php
      welcome.blade.php
```

Pages and functionality:

Landing Page:



Dev Exercise - Trader interactive

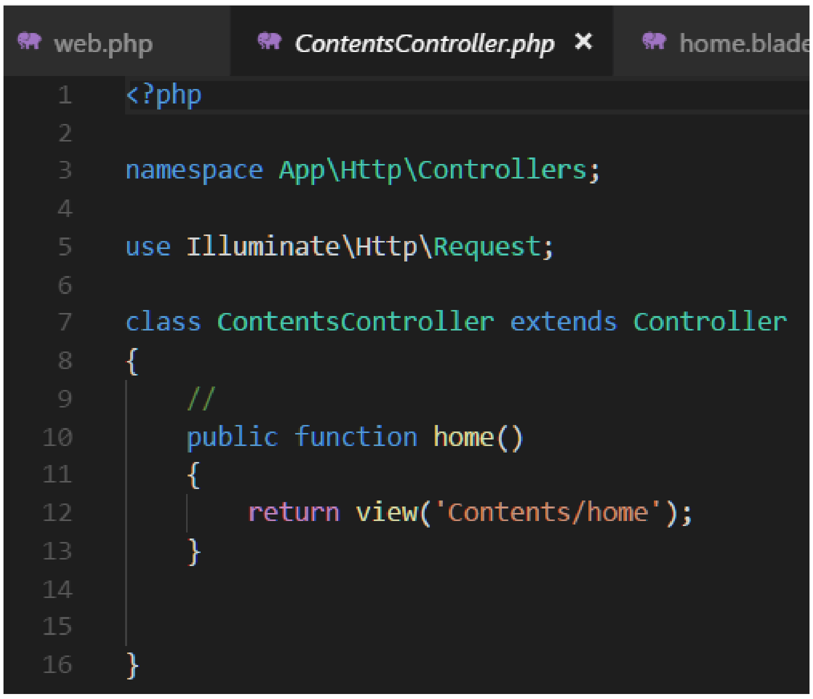
By clicking on the sales Tab , you will be navigated to the page , where you can choose the vehicle type you want to buy. and contact the seller

- Designed by Sravanthi kapu (Tony)

This html of this page is in: resources\views\contents\home.blade.php

```
Route::get('/','ContentsController@home')->name('home');
```

The route defines; the landing page should be displayed according to the method home in ContentsController.

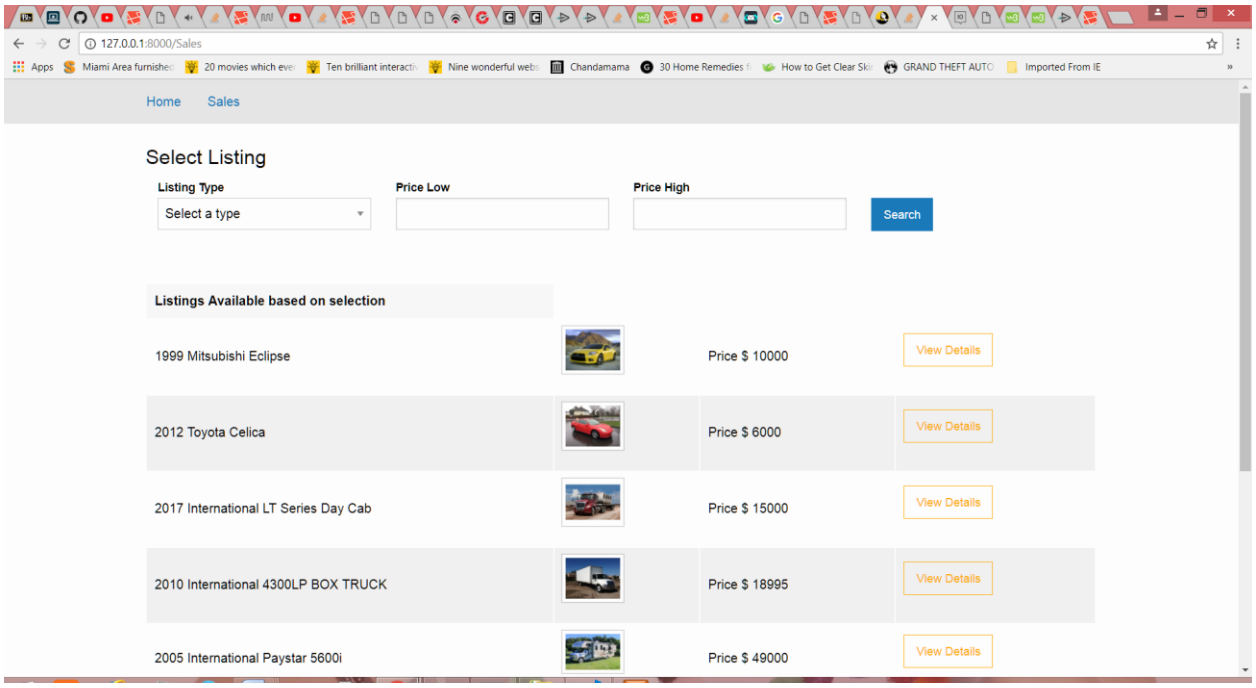


```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ContentsController extends Controller
{
    //
    public function home()
    {
        return view('Contents/home');
    }


}
```

The method in the class ContentsController returns to the view home i.e. home.blade.php. So the Landing page is displayed.

Sales Page:

Sales Page is displayed on clicking Sales tab.



Select Listing

Listing Type	Price Low	Price High	
Select a type			Search

Listings Available based on selection

1999 Mitsubishi Eclipse		Price $ 10000	View Details
2012 Toyota Celica		Price $ 6000	View Details
2017 International LT Series Day Cab		Price $ 15000	View Details
2010 International 4300LP BOX TRUCK		Price $ 18995	View Details
2005 International Paystar 5600i		Price $ 49000	View Details

When we click on Sales tab, according to the route in the app.blade.php – layout used for all the views, the below route is used as per the view:

```
<ul class="dropdown menu" data-dropdown-menu="tckp8q-dropdown-menu" role="menubar">
<li role="menuitem"><a href="{{ route('home') }}">Home</a></li>
  <li role="menuitem"><a href="{{ route('search') }}">Sales</a></li>
```

So it checks for the named Route ‘search’

```
Route::get('/Sales', 'SalesController@search')->name('search');
```

Now according to our Route, it checks for the method search in SalesController.

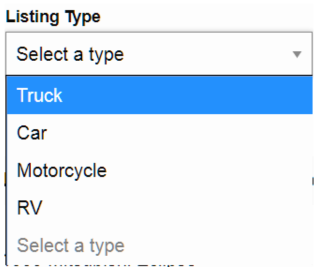
```
public function search(Request $request)
{
$data=[];


$data['listings']= $this->listing->all();

   $t=[];
   $t['listingst']=$this->listingst;
//dd($data['image']);
    return view('Sales/search',$data,$t)->with('data',$data);
}
```

According to the search method in SalesController it returns view search i.e. search.blade.php. So the Sales page is displayed.

Initially when the page is loaded, then all the listings are displayed on the page.

It’s implemented by sending the $data associative array to the view with all the records in listing table.

$t is sent to the view to display the types of listing in dropdown:



To get type of listing in Sales Page I added an array of listingtype in model listingtype and then by creating a constructor, and through dependency injection I used it in my search method.

listingtype class in model listingtype and inherited readlistings class.

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class listingtype extends readlistings
```

```
{
    protected $listings = ['Truck','Car','Motorcycle','RV'];
}
```

readlistings class in modal readlistings

```
class readlistings
{
    protected $listings = [];
    public function all()
    {return $this->listings;}
    public function get( $id )
    {return $this->listings[$id];}
}
```

To inject it to SalesController, added the below lines of code in controller and constructor:

```
use App\listingtype as listingtype;
```

```
public function __construct( listingtype $listingst, listing $listing)
    {
        $this->listingst = $listingst->all();
        $this->listing = $listing;
    }
```

Search Functionality:

To Display the Data related to listings based on the selection, we navigate to the route defined on the form action for search button in the search view

```
<form action="{{route('request',1)}}" method="post">
```

According to this we will check the named route request.

```
Route::post('/Sales/{listing_id}',                'SalesController@request')-
>name('request');
```

Now according to our Route, it checks for the method request in SalesController.

```
public function request(Request $request)
    {
         if($request->isMethod('post'))
        {
           $this->validate(
            $request,
            ['pricehigh'=> 'required',
                'pricelow'=>'required',
                'listtype'=>'required', ]);}
        $data=[];

       $t=[];
       $t['listingst']=$this->listingst;

       $high = request()->get('pricehigh');
        $low = request()->get('pricelow');
        $listtype = request()->get('listtype');

        $listing = new listing();

     $data['listings']=$listing->getlistings($low,$high,$listtype);
       // dd($listtype,$data['listings']);
        return view('Sales/search',$data,$t)->with('data',$data);

    }
```

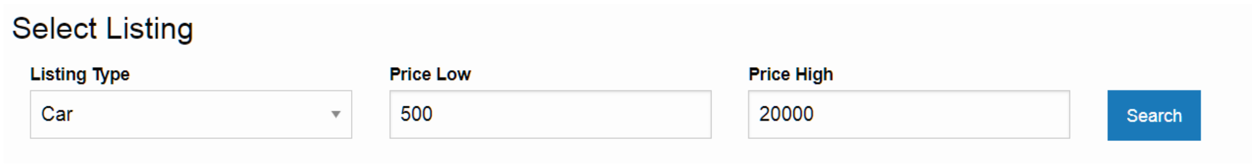
Here, we are validating the form first. Then based on the inputs given to the form by user, we get the listings which match that type and which are in that specified price range by calling the function getlisting which is in listing model in database folder.

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\DB;
class listing extends Model
{
     public $table = "listing";
     protected $primaryKey = 'listid';

     public function getlistings ($pricelow  ,$pricehigh,$type)
     {
         $availablelistings= DB::table('listing as l')
                                ->select(
'l.listid','l.year','l.make','l.model','l.description','l.price','l.imagepath'
,'l.metadata','l.sellerid')
                                ->whereRaw("
                                l.listid IN (
                                    Select v.listid from listing v
                                    WHERE (
                                        v.price > '{$pricelow}' AND
                                        v.price < '{$pricehigh}' AND
                                        v.type = '{$type}'
                                             )) ")
                                ->orderBy('l.listid')->get();
         return $availablelistings;
     }}
```

By the help of eloquent and DB Facade, we run our raw SQL query and returning the result.

By all this functions we get listings based on our search.

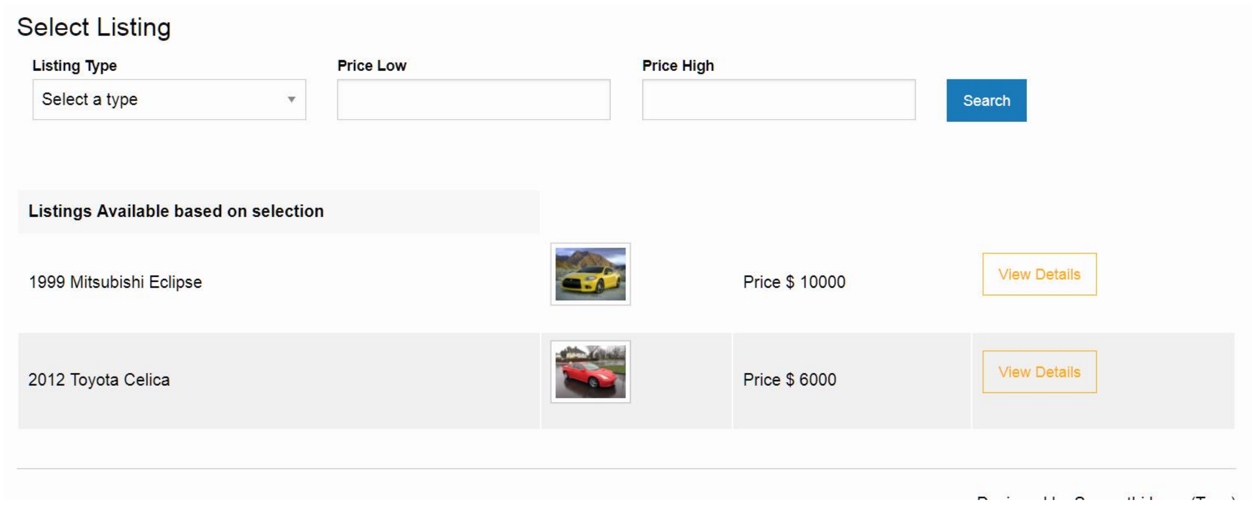
For example if we selected the below options



Select Listing

Listing Type	Price Low	Price High	
Car	500	20000	Search

Then clicked on search, we will be displayed with these details.



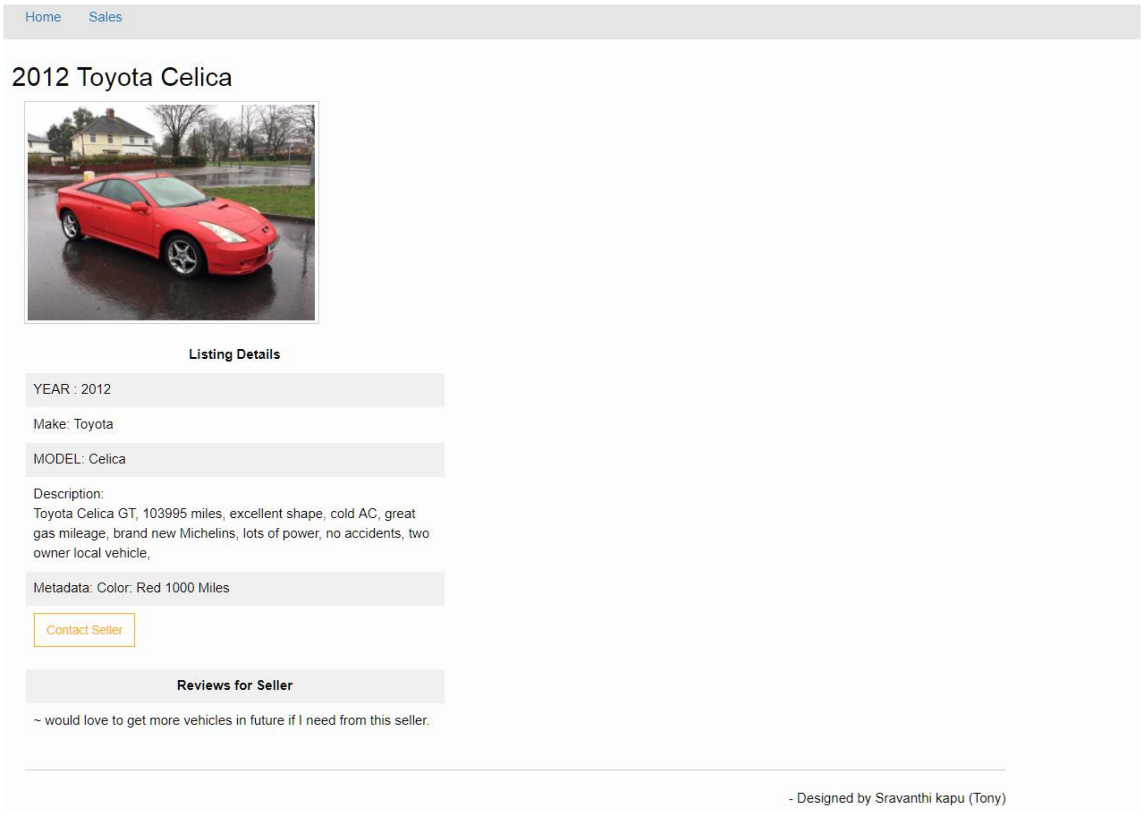
Select Listing

Listing Type	Price Low	Price High	
Select a type			Search

Listings Available based on selection			
1999 Mitsubishi Eclipse		Price $ 10000	View Details
2012 Toyota Celica		Price $ 6000	View Details

View Details Functionality:

When we click on view details button for a specific listing, we will be navigated to details page.



Home Sales

2012 Toyota Celica

Listing Details

YEAR : 2012

Make: Toyota

MODEL: Celica

Description:
Toyota Celica GT, 103995 miles, excellent shape, cold AC, great gas mileage, brand new Michelins, lots of power, no accidents, two owner local vehicle,

Metadata: Color: Red 1000 Miles

Contact Seller

Reviews for Seller

~ would love to get more vehicles in future if I need from this seller.

- Designed by Sravanthi kapu (Tony)

When we click on view details button, according to the route in the search.blade.php view, the below route is used:

```
<a class="hollow button warning"
href="{{route('details',['listing_id'=>$listings->listid])}}">View Details</a>
```

We are sending the selected listing id to the named route details in web.php.

```
Route::get('/Sales/{listing_id}', 'SalesController@details')->name('details');
```

Now, we will call the function details in SalesController, by sending the listing id.

```
public function details($listid)
    {
        $data =[];
        $listdata = $this->listing->find($listid);
        $data['type']= $listdata->type;
        $data['year']= $listdata->year;
        $data['make']= $listdata->make;
        $data['model']= $listdata->model;
        $data['description']= $listdata->description;
        $data['price']= $listdata->price;
        $data['metadata']= $listdata->metadata;
        $data['imagepath']= $listdata->imagepath;
        $data['sellerid']= $listdata->sellerid;

        $i = $data['sellerid'];
        $seller = new seller();
        $selectedseller= $seller->find($i);
        $data['reviews']=$selectedseller->reviews()->get();
        //dd($data['reviews']);
        return view('Sales/details',$data)->with('data',$data);    }
```

In this function details, in SalesController Class, with the help of find function in eloquent, we are finding all the details of the listing from listing table, with the specific list id received by this function.

In order to get all the seller reviews of the seller of this listing, I used the seller id from the listings details and by using find; I got the seller details from seller table.

Then as there is hasMany relation between sellerrev and seller, I have defined a function in seller model.

```
class seller extends Model
{//
    public $table = "seller";
    protected $primaryKey = 'sellerid';

    public function reviews()
     {
         return $this->hasMany('App\sellerrev','sellerid');
     }}
```

By using $data['reviews']=$selectedseller->reviews()->get() , all reviews based on the seller id are obtained.

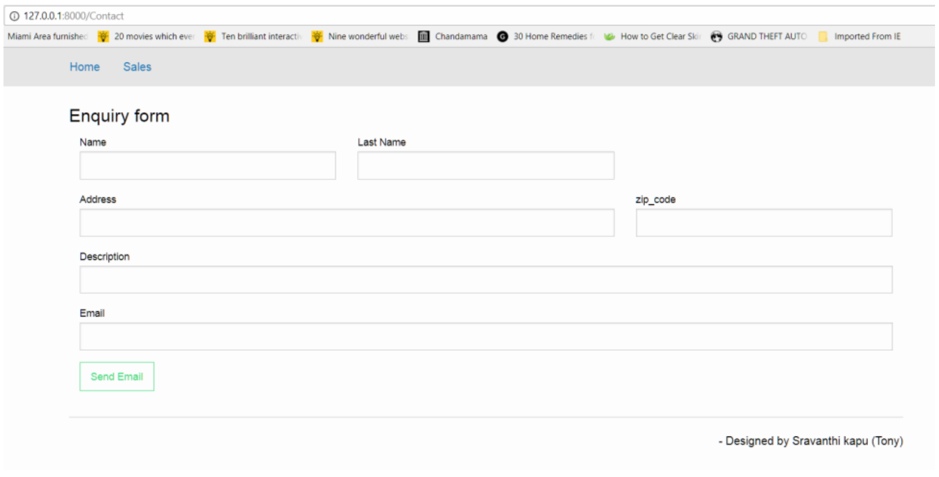
Approach to display seller details: (not implemented in the current code)

By passing $selectedseller array, the details of the seller can be sent to the view, in order to display the seller details.

Approach to display other images of listing: (not implemented in the current code)

As per database design, all the images of listing can be stored in listingimages table. Similar to reviews and seller, images and listing also have hasMany relation. By defining a function in listing model and then by using get method, we can get all the images for the specific listing and then by passing that to the view, we can display the images.

Contact Seller Functionality: When we click on the contact seller button, we will be navigated to the contact page.



When we click on contact seller button, according to the route in the details.blade.php, the below route is used as per the view:

```
<a    class="hollow    button    warning"    href="{{route('showform')}}">Contact
Seller</a>
```

It checks for the named route showform in web.php.

```
Route::get('/Contact', 'ContactsellerController@showform')->name('showform');
```

Now, we will call the function showform in ContactsellerController.

```
public function showform(Request $request)
    {
        $data =[];

        $data['name'] = $request->input('name');
        $data['last_name'] = $request->input('last_name');
        $data['address'] = $request->input('address');
        $data['zip_code'] = $request->input('zip_code');
        $data['Description'] = $request->input('Description');
        $data['email'] = $request->input('email');

        if($request->isMethod('post'))
        {
           $this->validate(
            $request,
            [
                'name'=> 'required',
                'last_name'=>'required',
                'address'=> 'required',
                'zip_code'=>'required|digits:5',
                'email'=>'required|email',
            ]
            );
            Mail::send('emails/emailtemp',$data,function($m)use ($data){
                $m->from($data['email']);
                $m->to('skapu001@odu.edu');
            }
             );
        }
        return view('Contactseller/showform'); }
```

According to this function it returns to the view showform.blade.php.

On clicking submit, as per the route in the view showform.blade.php we will go to named routed sendemail.

```
<form action="{{route('sendemail')}}" method="post">
```

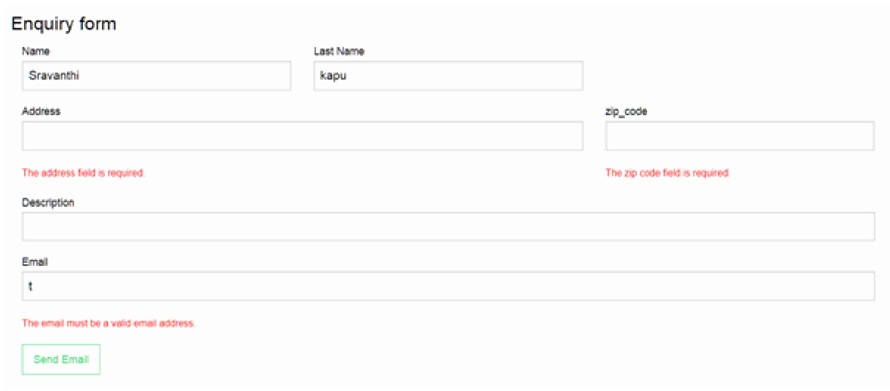
According to the named route sendemail:

```
Route::post('/Contact', 'ContactsellerController@showform')>name('sendemail');
```

We will check for the method showform in Contactseller class.

When the form is submitted, then the method will be post, then all the operations inside if are also executed for the method showform.

First the validations are done in this function, when the form is submitted.


Enquiry form

Name: Sravanthi

Last Name: kapu

Address:

zip_code:

The address field is required.

The zip code field is required.

Description:

Email: t

The email must be a valid email address.

If all the validations are passed then an email is sent.

Email Functionality:

Code to send email:

```
Mail::send('emails/emailtemp',$data,function($m)use ($data){
                $m->from($data['email']);
                $m->to('skapu001@odu.edu');
```

In order to send email:

The Mail Facade is used. So we need to include below line in our controller.

```
use Mail;
```

The layout of the email is defined in view/emails/emailtemp.blade.php

The data is sent to the send method in mail Façade and the passing the values with the variable inside the function, we can send the email. Once email is sent, it redirects to the contact page.

I hardcoded the value of emailed sent to “to” function.

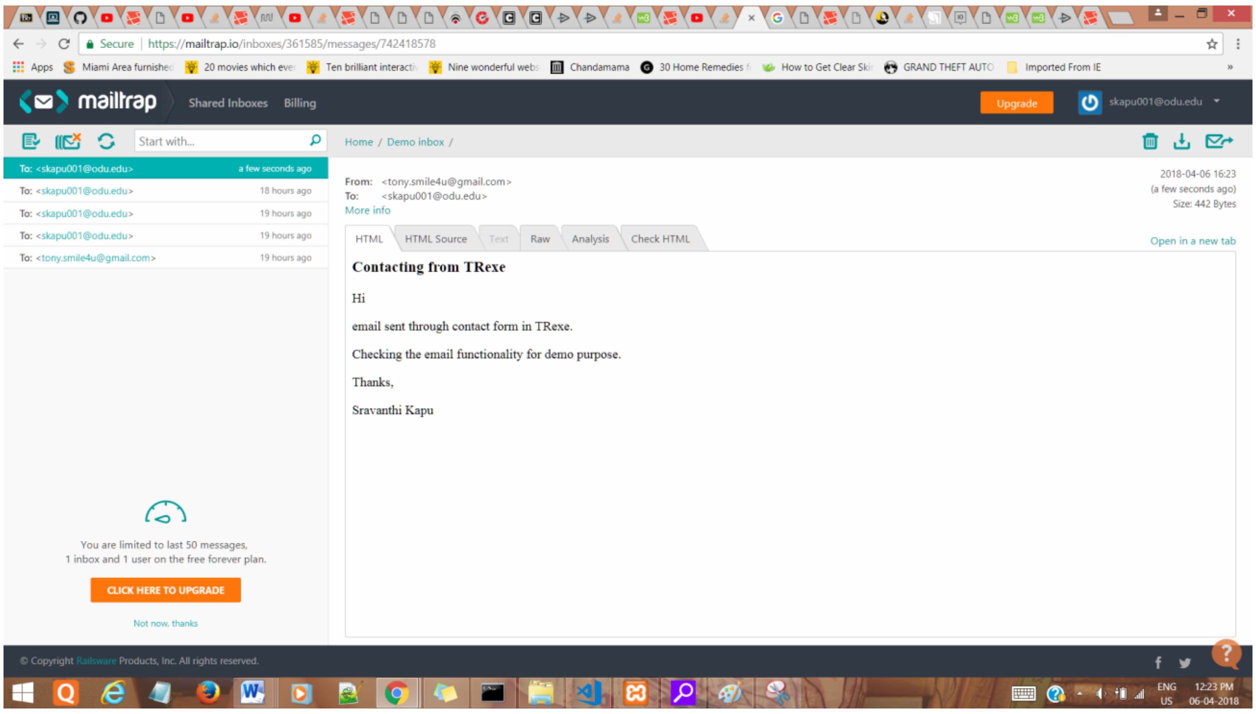
Configurations done in .env file in order to send mail:

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=569a804afbb1b2
MAIL_PASSWORD=35df597b998869
MAIL_ENCRYPTION=null
```

The username and password details can be obtained by creating an account in mailtrap.io.

Al the emails sent / received can be check in that inbox.

The sent/received emails in mailtrap.io:



https://mailtrap.io/inboxes/361585/messages/742418578

To: <skapu001@odu.edu> a few seconds ago
To: <skapu001@odu.edu> 18 hours ago
To: <skapu001@odu.edu> 19 hours ago
To: <skapu001@odu.edu> 19 hours ago
To: <tony.smile4u@gmail.com> 19 hours ago

From: <tony.smile4u@gmail.com>
To: <skapu001@odu.edu>
More info

2018-04-06 16:23
(a few seconds ago)
Size: 442 Bytes

Contacting from TRexe

Hi

email sent through contact form in TRexe.

Checking the email functionality for demo purpose.

Thanks,

Sravanthi Kapu

You are limited to last 50 messages,
1 inbox and 1 user on the free forever plan.

© Copyright Railsware Products, Inc. All rights reserved.

Approach to send email to specific seller: (not implemented in current code)

In order to send an email to seller, in details page we have seller details.

By sending the email of the seller, when we are routing to the contact page, we will have seller email. So by providing that variable to to() an email can be sent to the seller.

Few errors faced:

1. When implementing email functionality:
   Error: Address in mailbox given does not comply with RFC 2822, 3.6.2

   Solution: Whenever we change the config files, we always need to execute the below command in order to make sure Laravel uses the new config properties:
   `php artisan config:cache`

2. When displaying images in views:
   Error: The listing images are displayed in one page and in the other page a blank no image icon is displayed. The image path picked is correct when verified it using inspect element in the browser, the image didn't load.

   Solution: we need to use the helpers – [**"{{ asset( ]** or [**"{{ URL::to(']** before our file path in order to get the actual path of the images in public folder and then all the images in any view are displayed properly.