Designer Project Overview

Eoxys Systems India Pvt, Ltd.

Introduction:

- Designer App is used to design a lighting configuration of a commercial or residential projects by it’s plan.
- It is online lighting design tool offered by brightgreen which is used to calculate the instantaneous total power of the lights, illuminance of lights, estimation of the lights(based on amount of visible emitted light) in each and every room of the houseplan
- Initially the house plan from the owner will be uploaded to this app to design a light plan
- Using this house plan, we can generate the area of each and every room.Based on that area we can design a light plan
- Once the light plan gets completed,if necessary the estimation can be done with this app.

Designer app Structure :

Name	Date modified	Type	Size
app	17-Feb-17 7:29 PM	File folder	
bootstrap	17-Feb-17 7:29 PM	File folder	
config	21-Feb-17 10:53 AM	File folder	
database	17-Feb-17 7:29 PM	File folder	
public	17-Feb-17 9:06 PM	File folder	
resources	17-Feb-17 7:29 PM	File folder	
storage	17-Feb-17 7:30 PM	File folder	
tests	17-Feb-17 7:30 PM	File folder	
vendor	17-Feb-17 7:31 PM	File folder	
.gitattributes	05-Jan-17 9:43 PM	Text Document	1 KB
.user.ini	05-Jan-17 9:43 PM	Configuration sett...	1 KB
_ide_helper.php	05-Jan-17 9:43 PM	PHP File	313 KB
artisan	12-Jan-17 12:07 PM	File	2 KB
composer.json	28-Jan-17 8:56 PM	JSON File	2 KB
composer.lock	05-Jan-17 9:43 PM	LOCK File	120 KB
gulpfile.js	05-Jan-17 9:43 PM	JavaScript File	5 KB
package.json	05-Jan-17 9:43 PM	JSON File	1 KB
phpunit.xml	05-Jan-17 9:43 PM	XML Document	1 KB
README.md	05-Jan-17 9:43 PM	MD File	1 KB
server.php	11-Jan-17 4:15 PM	PHP File	1 KB

1. app :

The app directory, contains the core code of your application. The app directory ships with a variety of additional directories such as Console, Http, Providers, Contracts, Events, Exceptions, Extensions, Jobs, Models, Policies, Repositories, Services, Listeners.

Console:

The console directory stored all artisan commands.

Http:

This directory contains Controller, Middleware, Request.

Name	Date modified	Type	Size
Console	17-Feb-17 7:29 PM	File folder	
Contracts	17-Feb-17 7:29 PM	File folder	
Events	17-Feb-17 7:29 PM	File folder	
Exceptions	17-Feb-17 7:29 PM	File folder	
Extensions	17-Feb-17 7:29 PM	File folder	
Http	17-Feb-17 7:29 PM	File folder	
Jobs	17-Feb-17 7:29 PM	File folder	
Listeners	17-Feb-17 7:29 PM	File folder	
Models	22-Feb-17 6:52 PM	File folder	
Policies	17-Feb-17 7:29 PM	File folder	
Providers	17-Feb-17 7:29 PM	File folder	
Repositories	17-Feb-17 7:29 PM	File folder	
Services	25-Feb-17 9:05 PM	File folder	

App directory contains multiple directory like this

Providers :

This directory contains various service provider.

Contracts :

This directory contains various interface classes.

Events :

This directory stores events that your application can raise. Events may be used to alert other parts of your application that a given action has occurred, providing a great deal of flexibility and decoupling.

Exceptions :

This directory contains your application's exception handler and is also a good

Place to stick any exceptions thrown by your applications.

Extensions :

This directory contains Datasheets and SSO concept.

Jobs :

This directory contains the queueable jobs for your application.

Models :

Each database table has a corresponding “Model” which is used to interact with that table. Models allow you to query for data in your tables, as well as invert new records into the table.

Policies :

The Policies directory contains the authorization policy classes for your application. Policies are used to determine if a user can perform a given action against a resources.

Repositories :

The repository directory contains repository files of each and every controller.

Services :

This directory contains service what we have provide.

Listeners :

This directory contains the handler classes for your events. Handlers receive an event and perform logic in response to the event being fired. For example, a UserRegistered event might be handled by a SendWelcomeEmail listener.

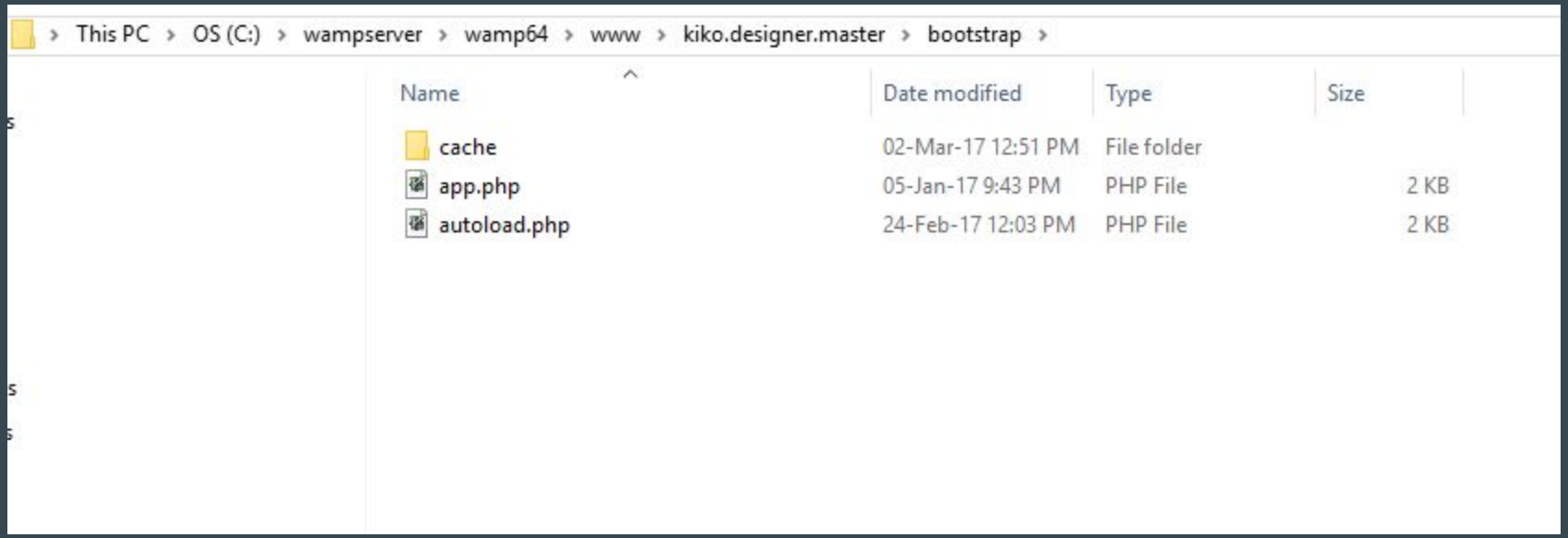
2. bootstrap :

The bootstrap directory contains a few files that bootstrap the framework and configure autoloading.

Cache :

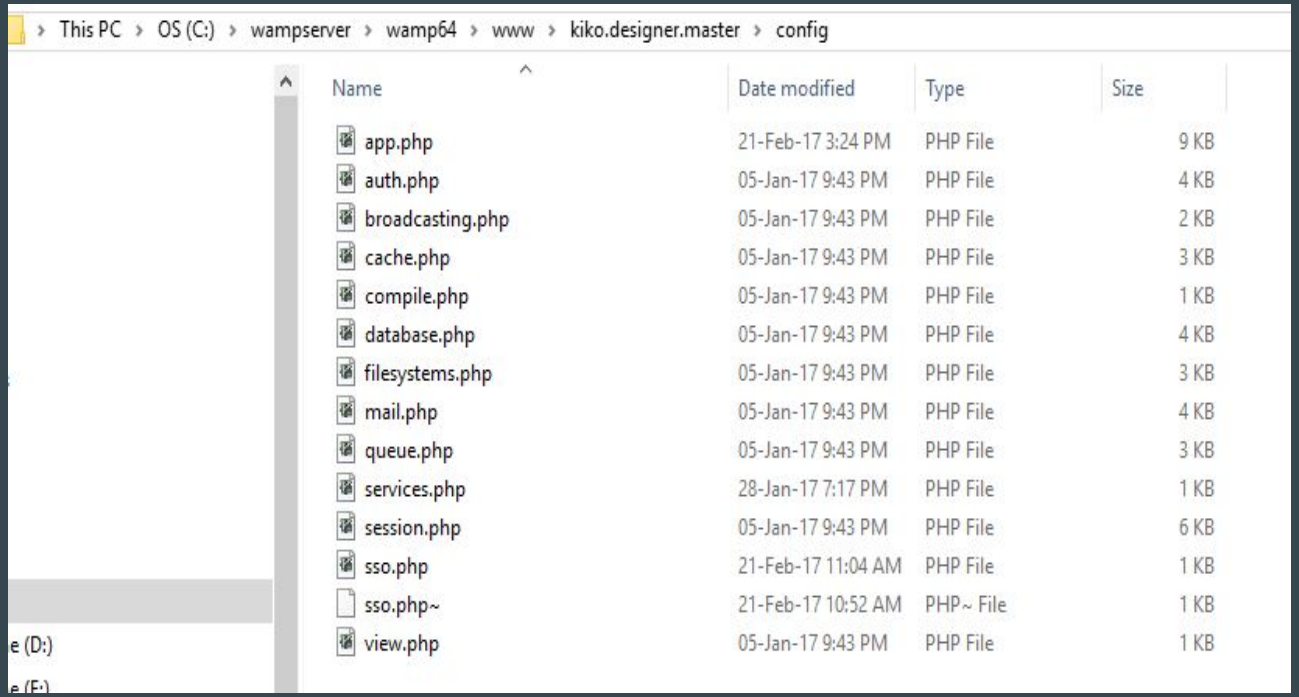
Cache directory that contains a few framework generated files for bootstrap performance optimization.

Bootstrap folder structure :



3. config :

The config directory, as the name implies, contains all of your application's configuration files.



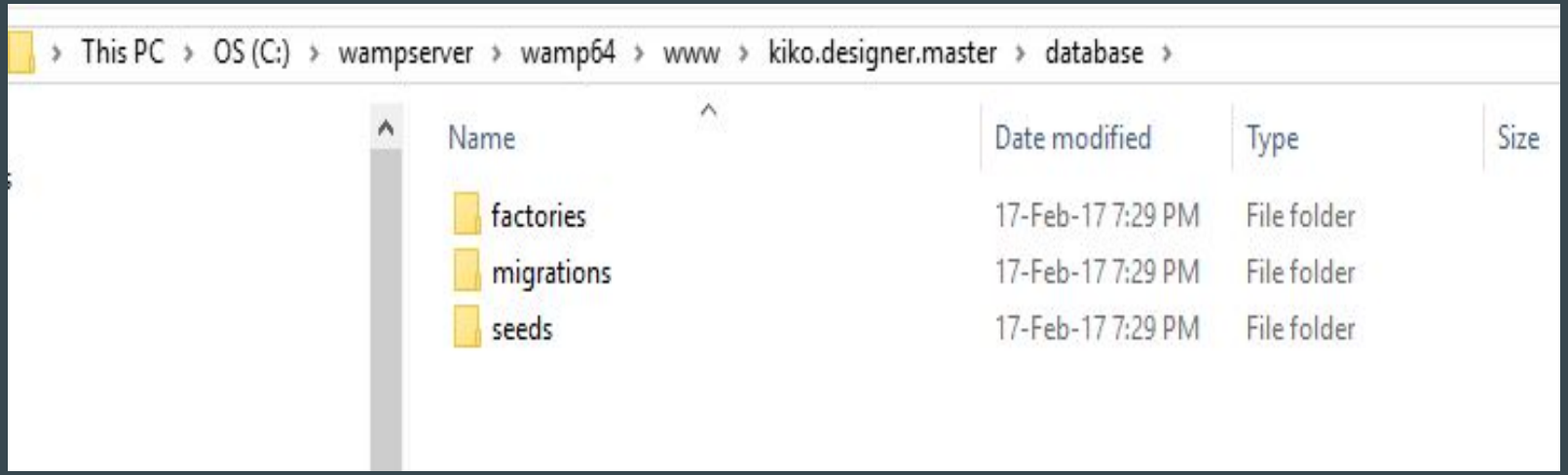
This PC › OS (C:) › wampserver › wamp64 › www › kiko.designer.master › config

Name	Date modified	Type	Size
app.php	21-Feb-17 3:24 PM	PHP File	9 KB
auth.php	05-Jan-17 9:43 PM	PHP File	4 KB
broadcasting.php	05-Jan-17 9:43 PM	PHP File	2 KB
cache.php	05-Jan-17 9:43 PM	PHP File	3 KB
compile.php	05-Jan-17 9:43 PM	PHP File	1 KB
database.php	05-Jan-17 9:43 PM	PHP File	4 KB
filesystems.php	05-Jan-17 9:43 PM	PHP File	3 KB
mail.php	05-Jan-17 9:43 PM	PHP File	4 KB
queue.php	05-Jan-17 9:43 PM	PHP File	3 KB
services.php	28-Jan-17 7:17 PM	PHP File	1 KB
session.php	05-Jan-17 9:43 PM	PHP File	6 KB
sso.php	21-Feb-17 11:04 AM	PHP File	1 KB
sso.php~	21-Feb-17 10:52 AM	PHP~ File	1 KB
view.php	05-Jan-17 9:43 PM	PHP File	1 KB

e (D:)

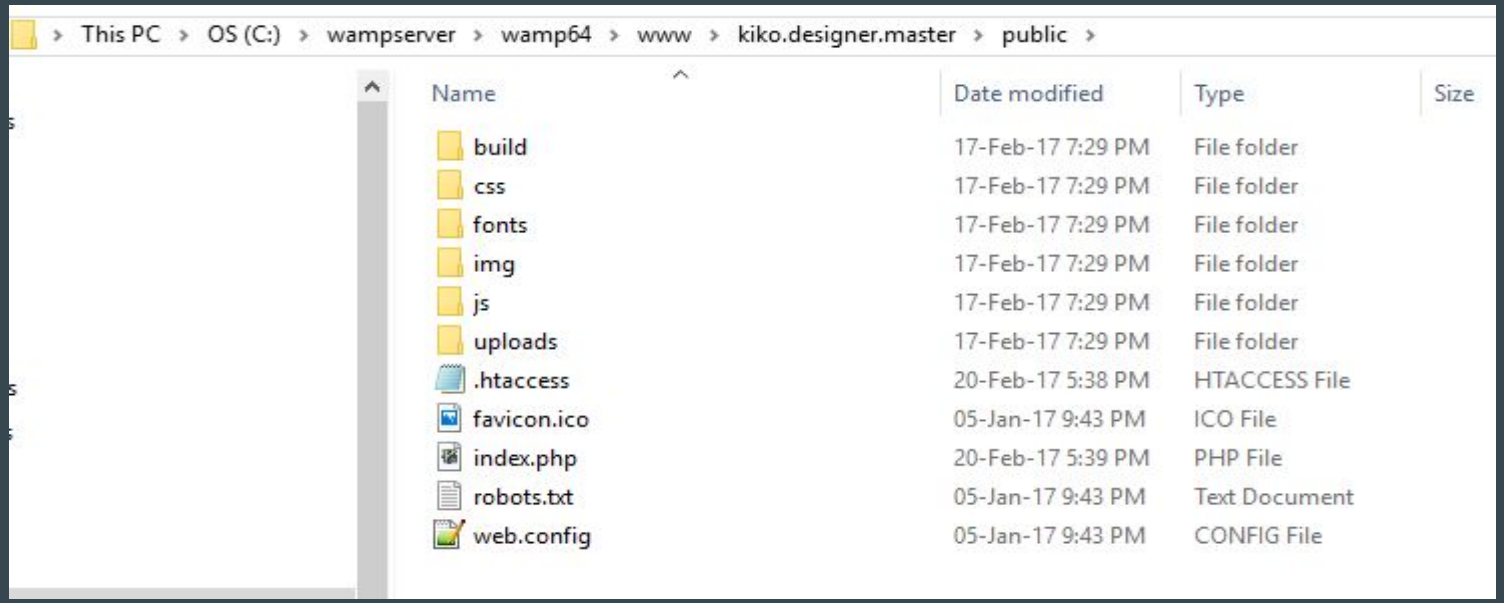
4. database :

The database folder contains your database migration and seeds. If you wish, you may also use this folder to hold an SQLite database.



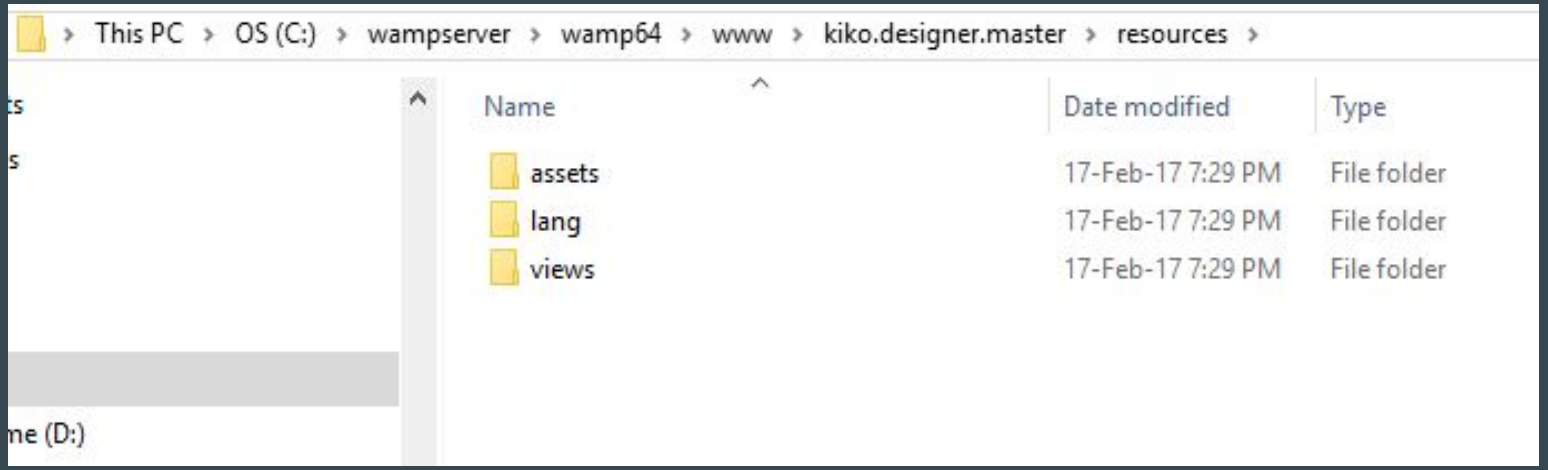
5. public :

This is the application's document root. It starts the Laravel application. It also contains the assets of the application like JavaScript, CSS, Images, etc.



6. resources :

This directory contains raw assets such as the LESS & Sass files, localization and language files, and Templates that are rendered as HTML.



7. storage :

The storage directory contains compiled Blade templates, file based sessions, file caches, and other files generated by the framework. This folder is segregated into app, framework, and logs directories. The app directory may be used to store any files utilized by your application. The framework directory is used to store framework generated files and caches. Finally, the logs directory contains your application's log files.

This PC › OS (C:) › wampserver › wamp64 › www › kiko.designer.master › storage ›

Name	Date modified	Type
clockwork	17-Feb-17 7:29 PM	File folder
framework	17-Feb-17 7:29 PM	File folder
logs	27-Feb-17 9:32 PM	File folder
pdf	02-Mar-17 8:02 PM	File folder
uploads	17-Feb-17 7:30 PM	File folder

8. test :

The tests directory contains your automated tests.

9. vendor:

The vendor directory contains your Composer dependencies

Lifecycle Overview :

First Things

The entry point for all requests to a Laravel application is the public/index.php file. All requests are directed to this file by your web server (Apache / Nginx) configuration. The index.php file doesn't contain much code. Rather, it is simply a starting point for loading the rest of the framework.

The index.php file loads the Composer generated autoloader definition, and then retrieves an instance of the Laravel application from bootstrap/app.php script. The first action taken by Laravel itself is to create an instance of the application / service container.

HTTP / Console Kernels :

Next, the incoming request is sent to either the HTTP kernel or the console kernel, depending on the type of request that is entering the application. These two kernels serve as the central location that all requests flow through. For now, let's just focus on the HTTP kernel, which is located in app/Http/Kernel.php.

The HTTP kernel extends the Illuminate\Foundation\Http\Kernel class, which defines an array of bootstrappers that will be run before the request is executed. These bootstrappers configure error handling, configure logging, detect the application environment, and perform other tasks that need to be done before the request is actually handled.

The HTTP kernel also defines a list of HTTP middleware that all requests must pass through before being handled by the application. These middleware handle reading and writing the HTTP session, determine if the application is in maintenance mode, verifying the CSRF token, and more.

The method signature for the HTTP kernel's handle method is quite simple: receive a Request and return a Response. Think of the Kernel as being a big black box that represents your entire application. Feed it HTTP requests and it will return HTTP responses.

Service Providers :

One of the most important Kernel bootstrapping actions is loading the service providers for your application. All of the service providers for the application are configured in the config/app.php configuration file's providers array. First, the register method will be called on all providers, then, once all providers have been registered, the boot method will be called.

Service providers are responsible for bootstrapping all of the framework's various components, such as the database, queue, validation, and routing components. Since they bootstrap and configure every feature offered by the framework, service providers are the most important aspect of the entire Laravel bootstrap process.

Dispatch Request :

Once the application has been bootstrapped and all service providers have been registered, the Request will be handed off to the router for dispatching. The router will dispatch the request to a route or controller, as well as run any route specific middleware.

SSO CONCEPT :

Single sign-on (SSO) is a session and user authentication service that permits a user to use one set of login credentials (e.g., name and password) to access multiple applications. The service authenticates the end user for all the applications the user has been given rights to and eliminates further prompts when the user switches applications during the same session. On the back end, SSO is helpful for logging user activities as well as monitoring user accounts.

Here we are using custom sso concept. For that we have separate SSO modules and middleware and some configuration files

1. When you are hitting the root url (/). It will redirects the index .php
2. The index.php file loads the Composer generated autoloader definition, and then retrieves an instance of the Laravel application from bootstrap/app.php script.

3. The incoming request is sent to the HTTP kernel. The HTTP kernel also defines a list of HTTP middleware that all requests must pass through before being handled by the application. These middleware handle reading and writing the HTTP session, determine if the application is in maintenance mode, verifying the CSRF token

4. It receives a Request and return a Response via HTTP kernel . The HTTP kernel redirect the URI to routes and All Laravel routes are defined in the app/Http/routes.php .The most basic Laravel routes simply accept a URI

5. Now app check the particular URI is present in the routes if it is there then redirect to the relevant controller's function.

6. The particular controller's function construct 2 services and 1 middleware initially.

- PDFService
- FloorplanService
- SSO middleware

Location : \app\Http\routes.php

```php
<?php
Route::group(['middleware' => ['web']], function() {

    Route::get('/', 'LightplanController@index')->name('home');
    //Route::get('/app', 'LightplanController@index');
    // SSO
    Route::group(['prefix' => 'sso'], function() {
        Route::get('/', 'LightplanController@index');
        Route::get('/logout', 'SSOController@logout');
        Route::get('/{token}', 'SSOController@index');
    });

    //Route::get('/proxy', 'LightplanController@proxy');
    Route::post('/upload', 'LightplanController@upload');
    Route::post('/image_crop', 'LightplanController@image_crop');
    Route::post('/pdf', 'LightplanController@pdf');

    // API
    Route::group(['prefix' => 'api'], function() {

        Route::get('/projects', 'ProjectsController@all');
        Route::get('/project/{id}', 'ProjectsController@show');
        Route::post('/project', 'ProjectsController@store');
        Route::put('/project', 'ProjectsController@update');
        Route::delete('/project/{id}', 'ProjectsController@destroy');
```

If you are hitting (/),it redirects to index.php and the index.php have a http kernel and it call the routes.php finally it call the relevant URI through that we reach the LightPlanController's index function

Location : \app\Http\Controllers\LightplanController.php

```
private $PDFService;
private $FloorplanService;

public function __construct(
    PDFService $pdfService,
    FloorplanService $floorplanService
) {
    $this->middleware('sso');
    $this->PDFService = $pdfService;
    $this->FloorplanService = $floorplanService
}

public function index()
{
    $user = session('user');
```

Here we construct sso middleware , PDF service, FloorplanService after constructing these service then call index function

In construct function first we will define about “ **this-> middleware(“sso”) “**;

When we are requesting on that time it loads the all class in “**\app\Http\kernel.php**”.In that PHP we have a assigned the middleware for “sso” here we get it that middleware class.

```
 *
 * These middleware may be assigned to groups or used individually.
 *
 * @var array
 */
protected $routeMiddleware = [
    //'auth' => \App\Http\Middleware\Authenticate::class,
    //'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth:
    //'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'sso' => \SSO\Middleware\SSOMiddleware::class,
];
```

Then we enter into that class

\SSO\Middleware\SSOMiddleware.php

```
     */

    public function handle($request, Closure $next)

    {

        if (!session()->has('user')) {

            return redirect()->away(SSO::getSSOUrl());

        }

return $next($request);

    }

}
```

If session having user information then it redirects to the relevant request otherwise it will redirects to (sso::getSSOUrl()).

SSO:: getSSOUrl() is located in " \app\Extensions\SSO\SSOManager.php"

```
public function __construct($app)
{
    $this->app = $app['config']['sso'];

    $this->Encrypter = new Encrypter($this->app['secret'], 'AES-256-CBC');
}

/**
 * Return URL of SSO service
 *
 * @return string  URL of SSO
 */
public function getSSOUrl()
{
return $this->app['url'] . '/sso/' . $this->createToken();
}
```

App stores the value from app/config/sso.php

Here it makes the redirect url from sso.php

\config\sso.php

```php
<?php

return [

    'url' => 'https://' . env('SSO_HOST', 'my.ssokiko.com'),

    'clientId' => env('SSO_CLIENT_ID', 'Nwjeeu23uiS8en3mfeefoIJfw4ni73kt'),

    'secret' => env('SSO_SECRET', '65bKfwekl0Nflsmy4pMVldrgoVNUJEnn'),

    'responseTimeLimit' => 10, // seconds for valid response
];
```

Here we have to set the redirecting host, clientId, secret key of redirecting host

The getSSOUrl() function return the URL string. This url string contains sso app url and its token. This token having client-Id and encrypted secret key

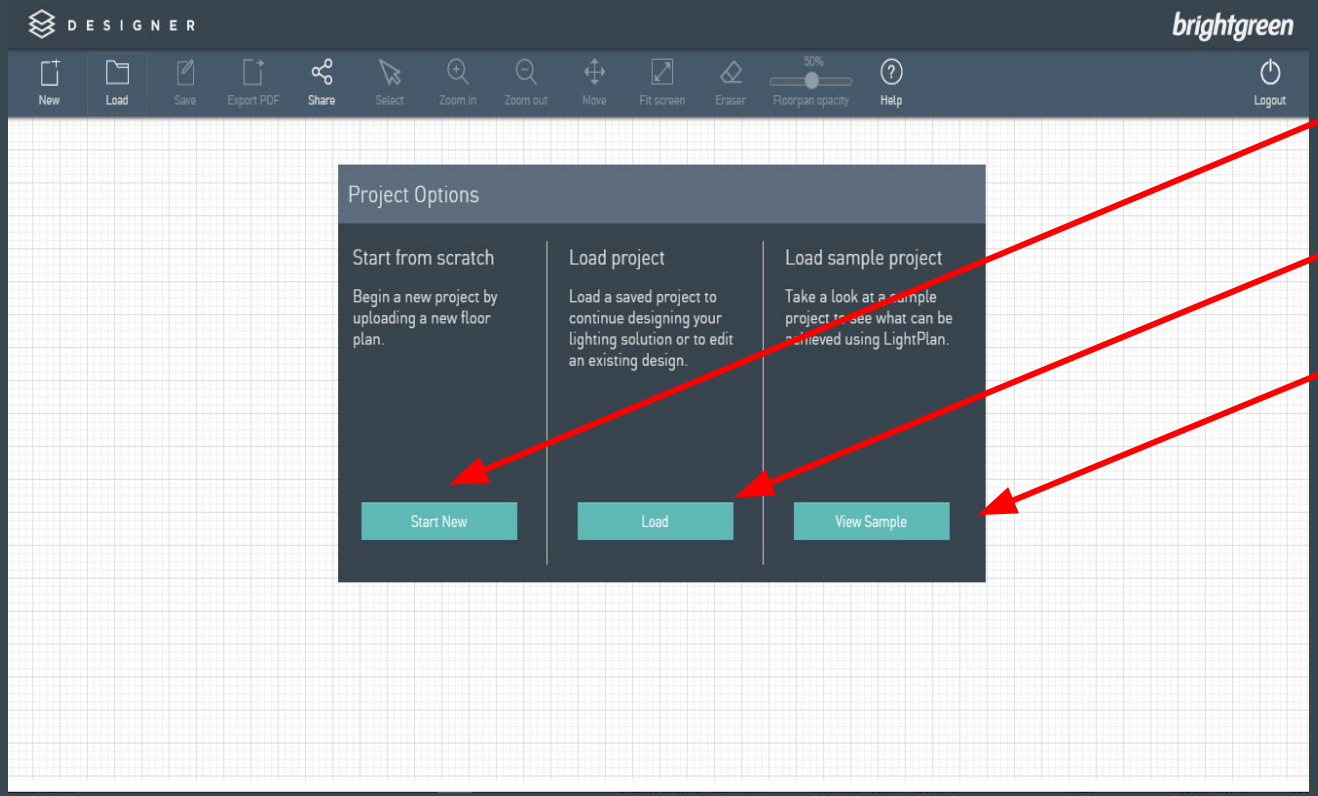
Example : https://your-SSO-domain/sso/token

token: client_id . encrypted secret key

```
public function getSSOUrl()
{
return $this->app['url'] . '/sso/' . $this->createToken();
}
```

Finally sso app check your client -id to its database and get the appropriate url for that client-id from database and redirects to the appropriate url

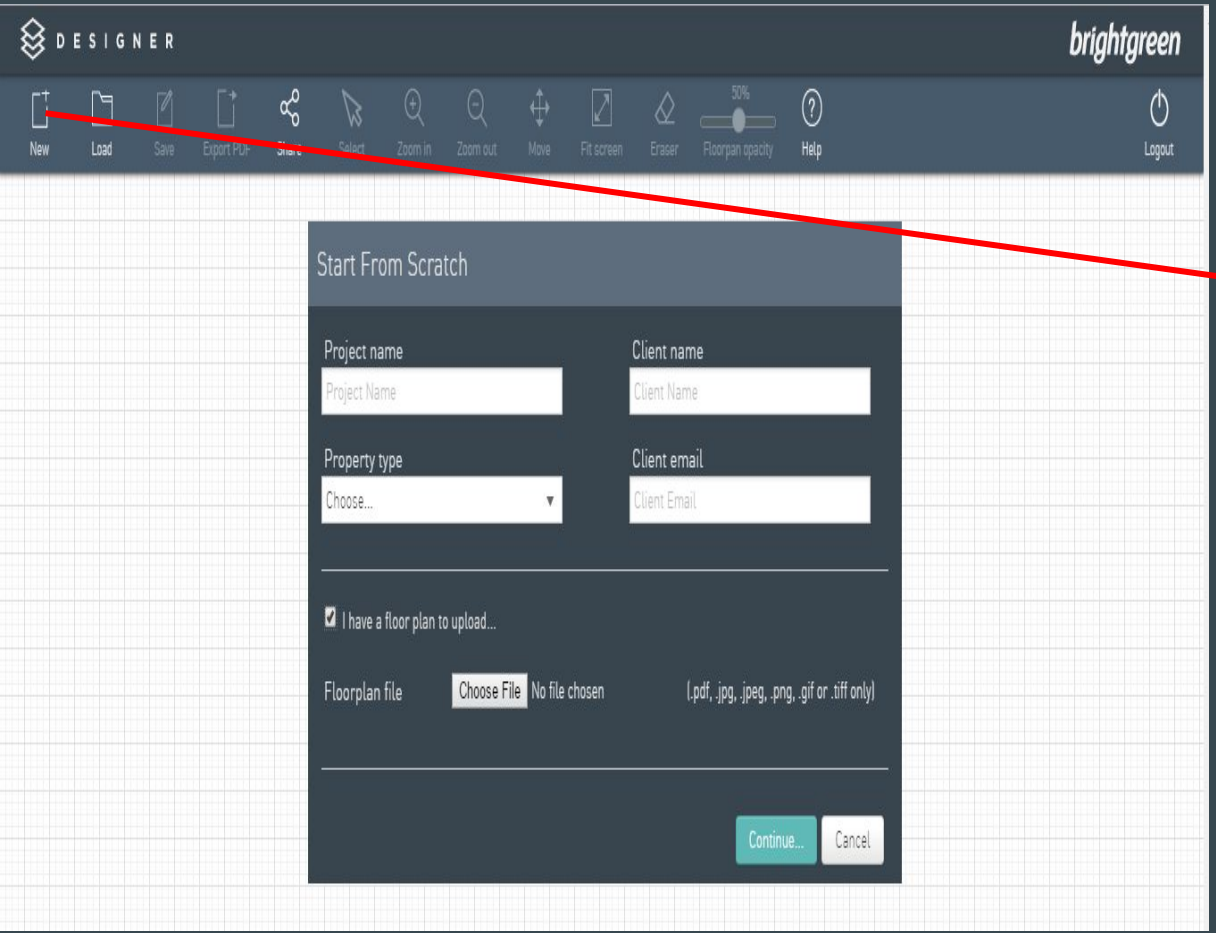
Our Index page look like this



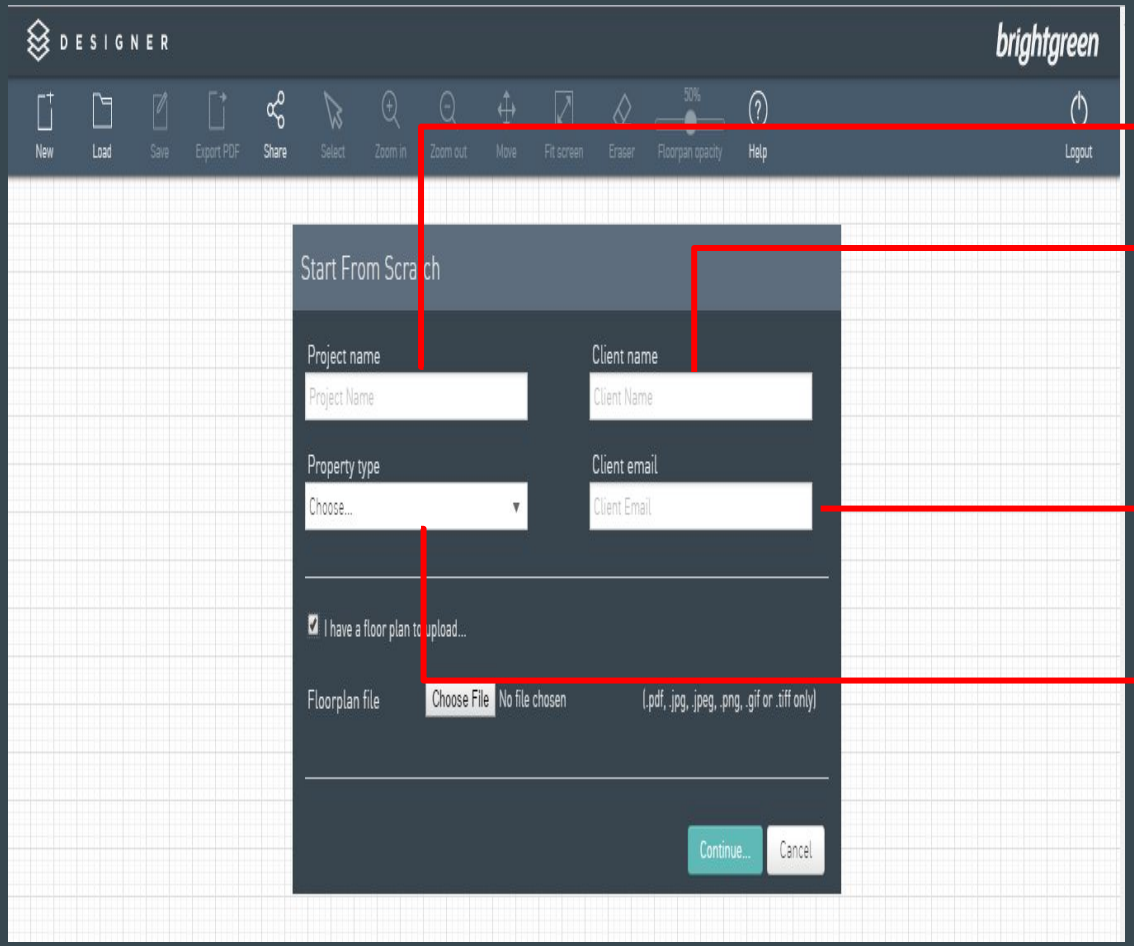
To create a new project

To load the all old project.

To view the sample project.



If you are clicking New it will show “start from scratch” dialogue box

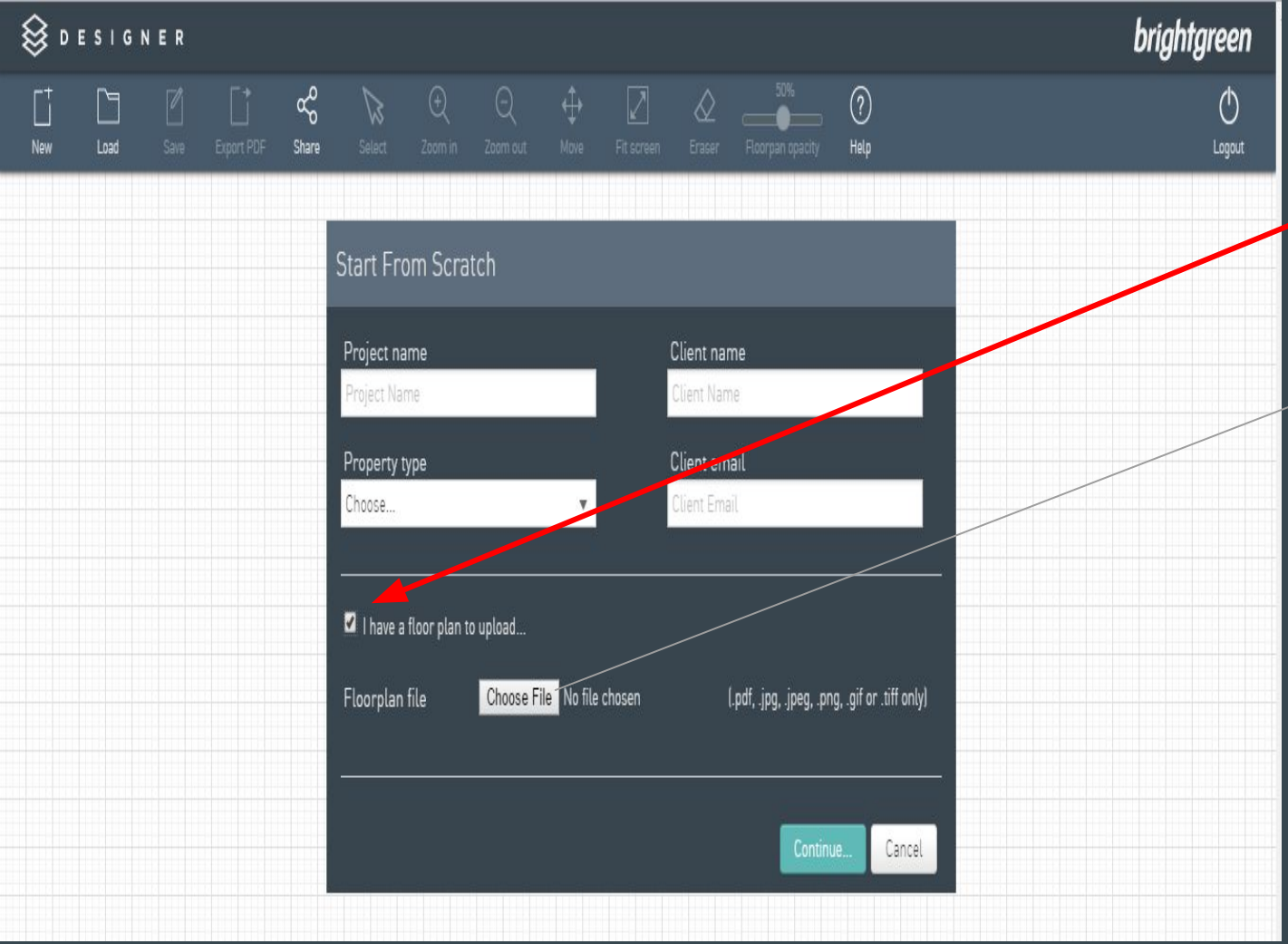


Enter the project name

Enter the client name

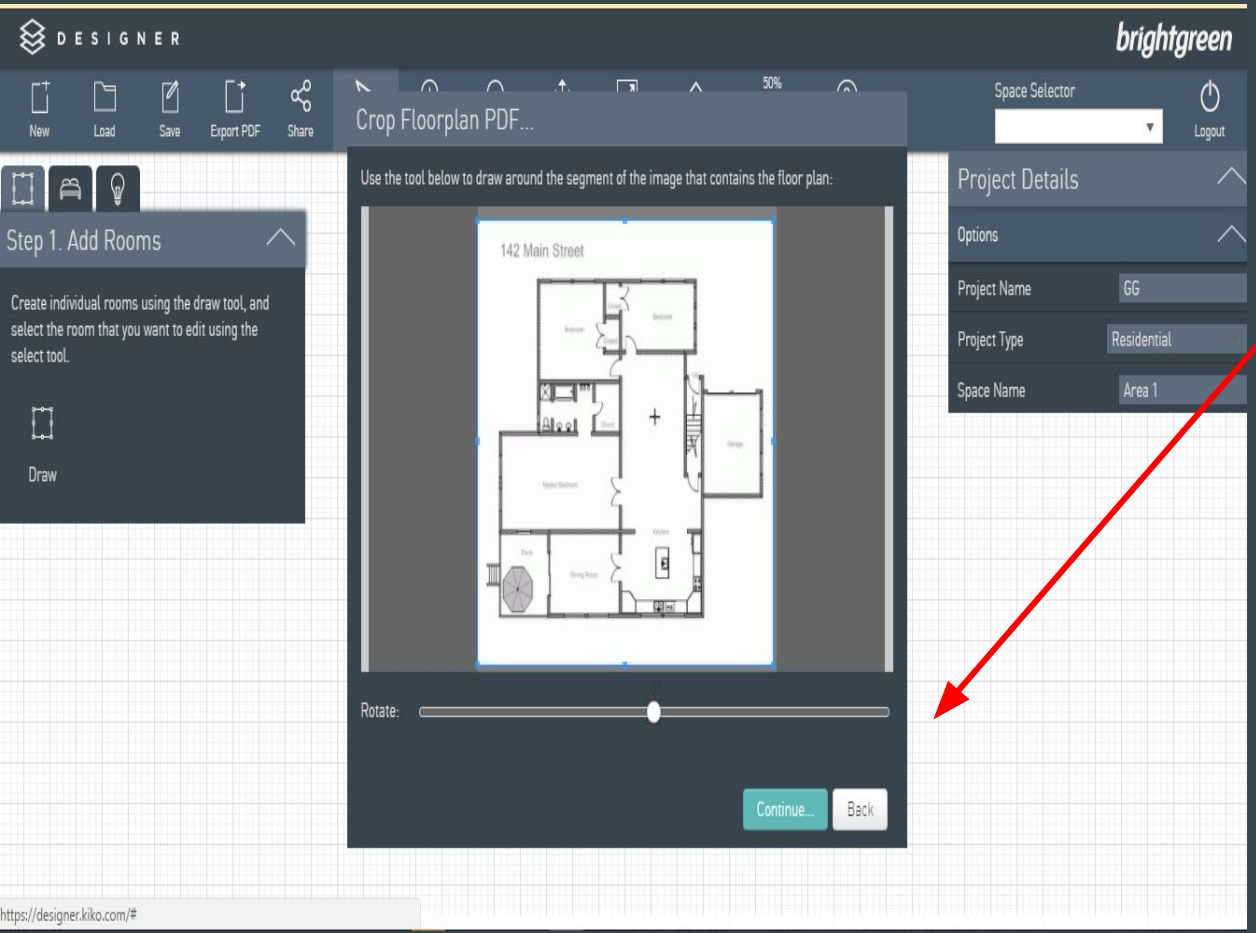
Enter the client email

Enter the project_type



If you have a plan check this box

Choose your file hear



Through **build/js/app-9628e456c2.js** file we have load the plan to floorplan container and also crop the plan limit whatever you want.

After clicking “continue” button it directly goes to “public/build/js/**app-9628e456c2**.js”. Here it will feed the trim file to relevant form and submit.

```
        })
    },
    hide: function() {
        this.dialog && $(this.dialog).dialog("close")
    },
    on_crop_rotate: function(e, t) {
        this.img.cropper("rotateTo", t), $("#crop_rotate_slider a span").html(t + "&deg;")
    },
    crop_image: function() {
        var e = this.img.cropper("getData");
        $("#crop_data").val(JSON.stringify(e)), app.set_loading($(".crop_dialog"), !0), $("#crop_form").submit()
    },
    crop_image_callback: function(e) {
        if (app.set_loading($(".crop_dialog"), !1), e.success) app.ui.image_crop.hide(), $("#crop_rotate_slider a span").html("0&deg;")
            url: e.result
        });
        else {
            var t = e.result || "An error has occured, please check the information and try again";
            alert(t)
```

Here it feed the cropped image data to the particular form id and also submit the form

Form-id =”#crop_data”

Location : \resources\views\includes\ui\image_crop.php

After submit the form it will call the form action (/image_crop) in routes.php

Post ------>method

/image_crop---->url

```
<div id="image_crop_dialog" class="crop_dialog dialog_pad" style="display:none">
    <form id="crop_form" method="post" action="/image_crop" target="image_crop_frame">
        <div>
            Use the tool below to draw around the segment of the image that contains the floor plan:
            <br/><br/>
        </div>
```

NOTE: This form id and previous slide submitting form id is same

Location : app\http\routes.php

```php
<?php
Route::group(['middleware' => ['web']], function() {

    Route::get('/', 'LightplanController@index')->name('home');
    //Route::get('/sso', 'LightplanController@index');
    // SSO
    Route::group(['prefix' => 'sso'], function() {
        Route::get('/', 'LightplanController@index');
        Route::get('/logout', 'SSOController@logout');
        Route::get('/{token}', 'SSOController@index');
    });

    //Route::get('/proxy', 'LightplanController@proxy');
    Route::post('/upload', 'LightplanController@upload');
    Route::post('/image_crop', 'LightplanController@image_crop');
    Route::post('/pdf', 'LightplanController@pdf');
```

Method : Post

Url : /image_crop

It routes to LightPlanController 's image_crop function

This request directly calls the LightPlanController's particular function

Location : \app\Http\Controllers\LightplanController.php

“ Route::post('/image_crop', 'LightplanController@image_crop'); ”

```
public function image_crop()
{
$result = $this->FloorplanService->crop_image(Request::all());
$data = json_encode([
'success' => isset($result[0]) ? $result[0] : false,
'result' => isset($result[1]) ? $result[1] : null
]);
return view('lightplan.image_crop', [
'data' => $data
]);
}
```

This controller function call the FloorplanService -> image_crop function

Location : \app\Services\FloorplanService.php

Here we get the thumb image of plan and save into location same location and return result array with image path and success msg

```
    }
        public function crop_image($request)
{
$cropImage = $request['crop_image_path'];
$image = $request['full_image_path'];
$output = str_replace(('.' . $this->imageOutputType), ('.crop.' . $this->imageOutputType), $image);
 echo '<script type="text/javascript">alert("' . $output . '")</script>';
list($width, $height, $type, $attr) = getimagesize($image);
$ratio = $width / $this->thumbnailWidth;
$cropData = json_decode($_POST['crop_data'], true);
$x = floatval($cropData['x']) * $ratio;
$y = floatval($cropData['y']) * $ratio;
$w = floatval($cropData['width']) * $ratio;
$h = floatval($cropData['height']) * $ratio;
$rotate = floatval($cropData['rotate']);
$greyscale = "-set colorspace Gray -separate -average";
$command = "convert -rotate {$rotate} +repage -crop {$w}x{$h}+{$x}+{$y} +repage {$greyscale} {$image} {$output}";
// $result = $this->convert('crop', $command);
$result = $this-> make_crop($image, $output,$x,$y,$w,$h);
$success = $result[0];
if ($success) {
// Remove previous image…
unlink($image);
unlink($cropImage);
rename($output, $image);
return [true, $image];
}
return $result;
}
```

```php
public function image_crop()
{
$result = $this->FloorplanService->crop_image(Request::all());
$data = json_encode([
'success' => isset($result[0]) ? $result[0] : false,
'result' => isset($result[1]) ? $result[1] : null
]);
return view('lightplan.image_crop', [
'data' => $data
]);
}
```

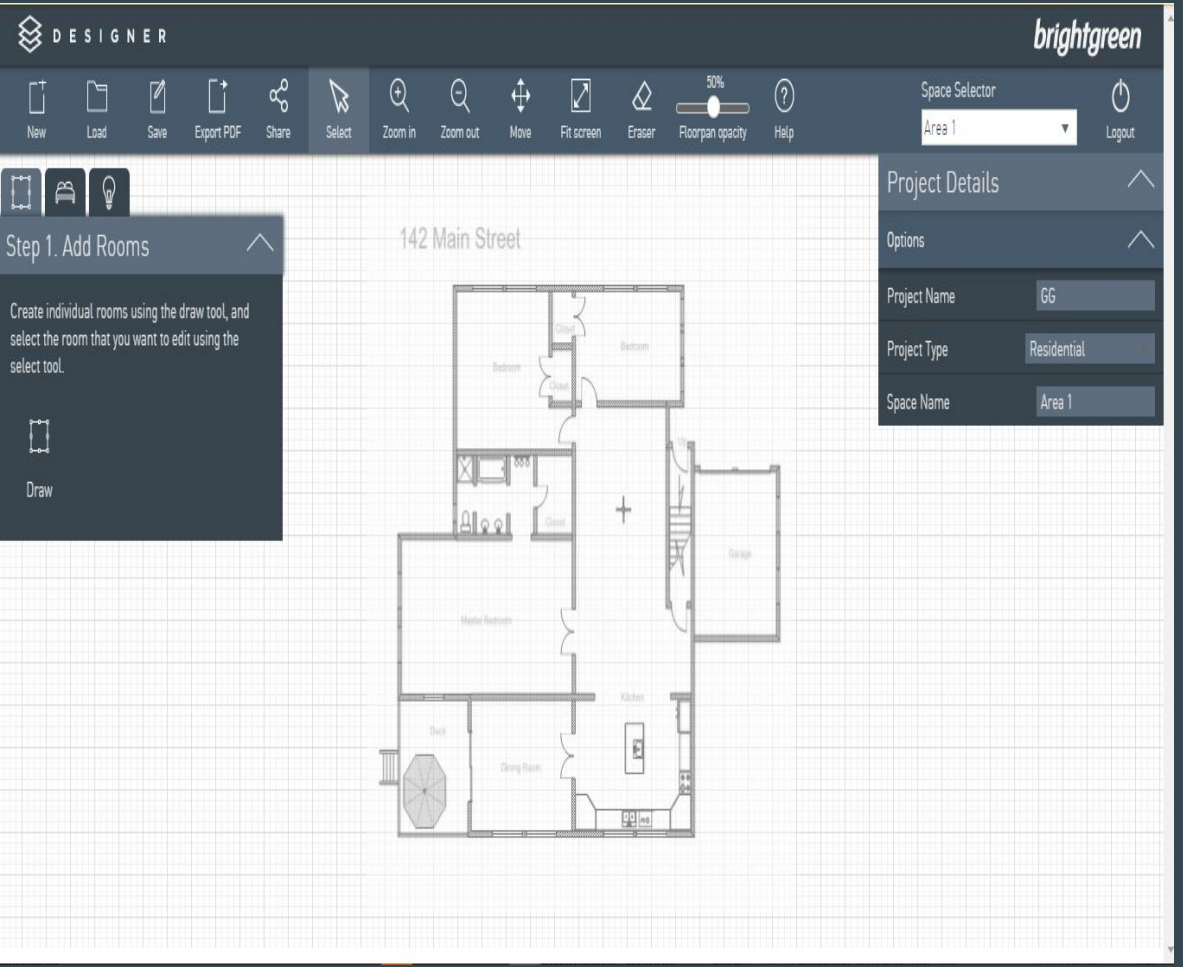
Result of array having image path and success msg

Return to corresponding view

Finally it redirects to this view page

\resources\views\lightplan\image_crop.blade.php

```php
<script type="text/javascript">
    window.parent.app.ui.image_crop.crop_image_callback(<?php echo $data; ?>);
</script>
```



After clicking continue button build/js/app-9628e456c2.js file having image crop function it just trim the image and submit the crop data to form and feed the value to crop-data