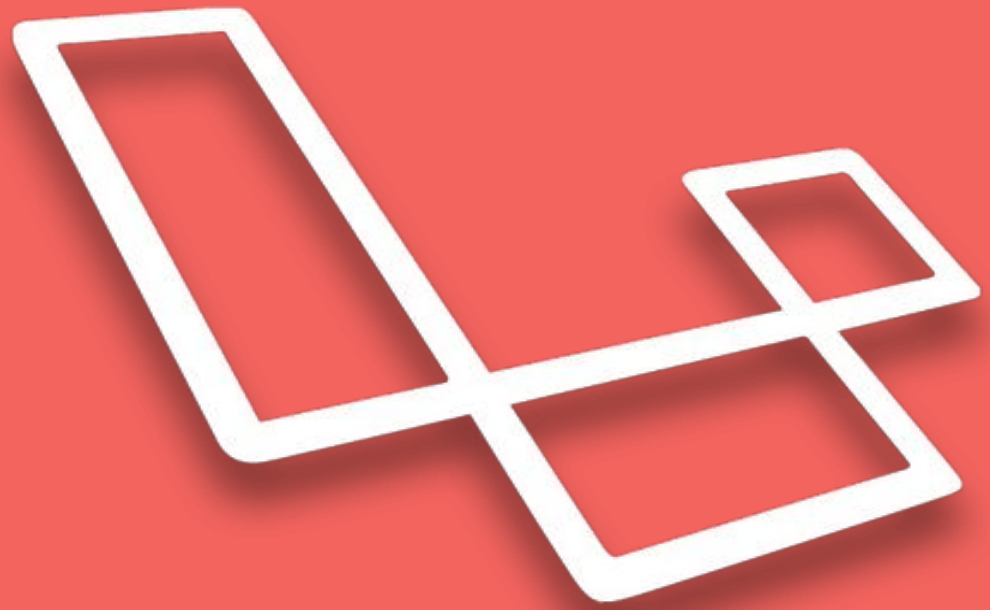
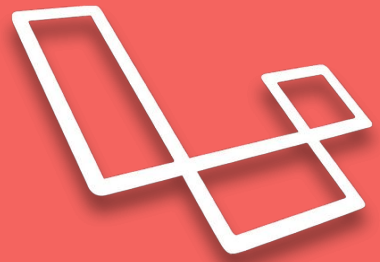


Content

- Request Validation
- Laravel Auth
- Middleware
- Service Container
- Working With Third Party Packages

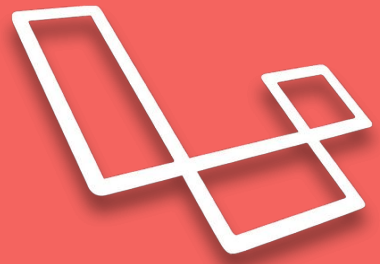


Request Validation

- **Request Life Cycle :-**
  See index.php first it loads the composers's autload file .

  Then we bootstrap laravel application container and register some basic service providers like Log Service provider look at Illuminate/Foundation/Application.php constructor method .

  Finally we create instance of kernel to register providers and take user request and process it through middlewares & handle exception then return response

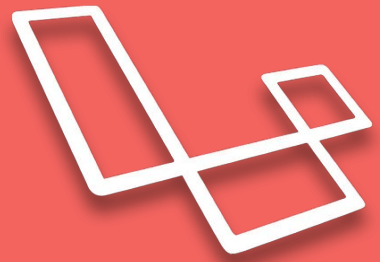


Request Validation

- In Controller :-

```
        $this->validate( $request , [
        'title ' => 'required ',
        'desc' => 'required|max:255'
    ] ,

     [
        'title.required'  => 'title is required to be filled ' ,
        'desc.max' => 'description max num of chars is 255 '
]);
```



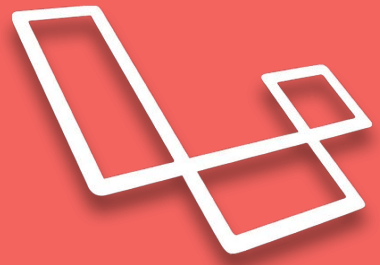
Request Validation

- Another way with request file :-
  php artisan make:request PostsStoreRequest

  Then in authorize method make it return true .

  After that define your rules in rules method .

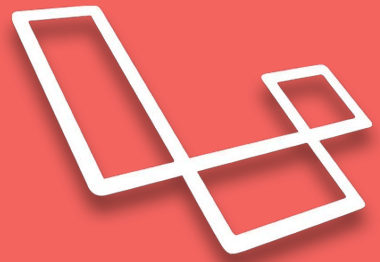
- validation rules :-
https://laravel.com/docs/master/validation#available-validation-rules
- custom validation messages in request file :-
https://laravel.com/docs/master/validation#customizing-the-error-messages



Laravel Auth

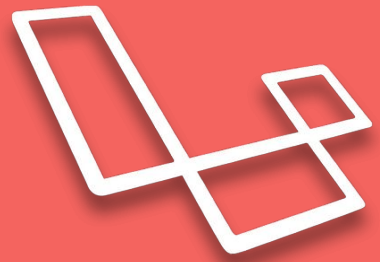
- **Read laravel docs , where it illustrates all you need better**

  https://laravel.com/docs/master/authentication#authentication-quickstart

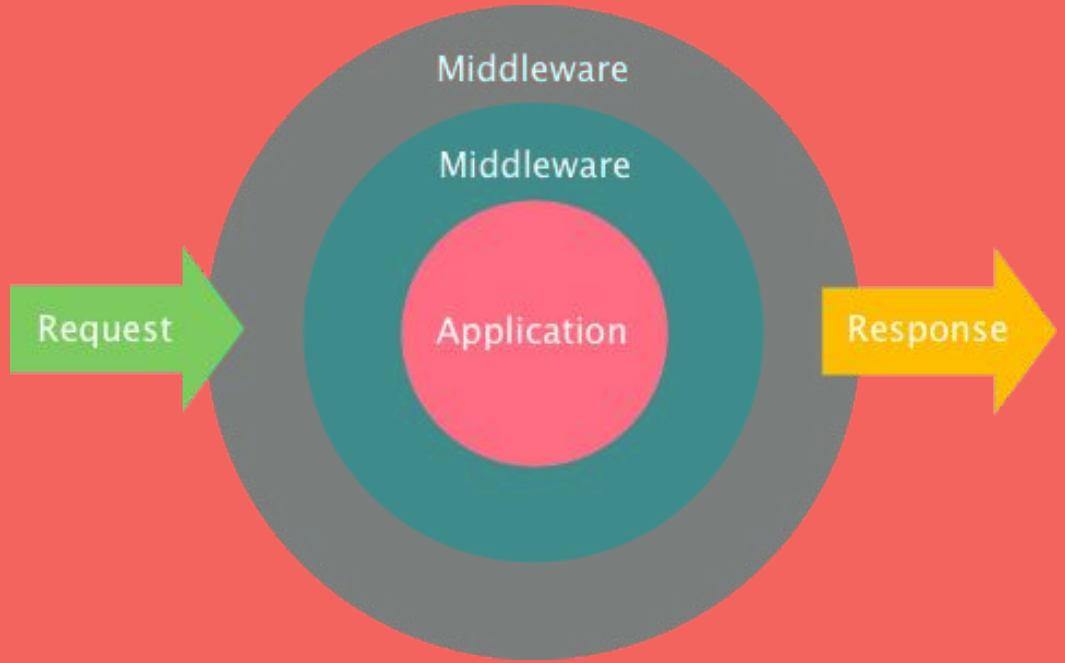


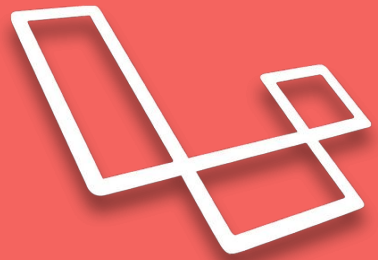
Middleware

- It's a series of layers around the application , every request passes through middlewares .
- Middlewares can inspect the request to decorate or reject it
- Also middlewares can decorate the response .
- register the middlewares in app/Http/Kernel.php
- handle($request , ..) method where you handle the request and choose to pass it for the next middleware or not .



Middleware

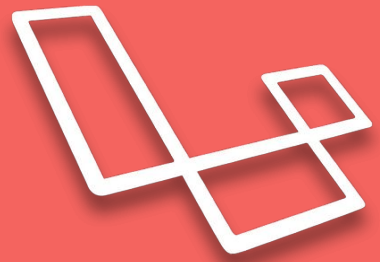




Service Container

- Other names for service container
  (IOC container , DI container , Application container)

- Dependency Injection (DI) :-
  Instead to make object instantiate it's dependencies internally , it will be passed from outside .



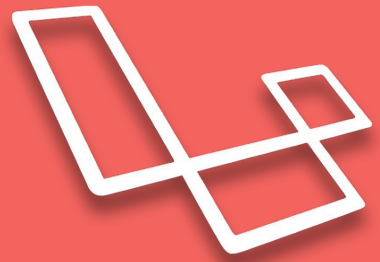
Service Container

- Class User {

```
    function __construct ()
    {
        $mysqlConnection = new Mysql();
    }

}
```

$user = new User(); // in this way object instantiated it's dependencies internally



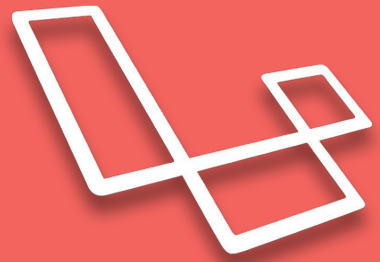
Service Container

- Class User {
  protected $dbConnection;
  //DB is an interface to easily switch between different db drivers
  function __construct (DB $db)
  {
  $this->dbConnection = $db;
  }

  }

$mysqlConnection = new Mysql();

$user = new User($mysqlConnection); // the user object dependencies are passed from outside .



Service Container

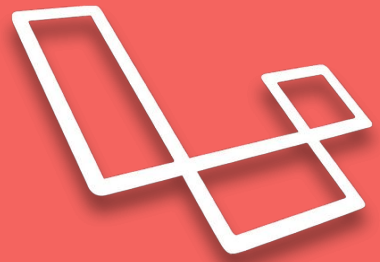
- Service container in laravel responsible for binding & resolving & AutoWiring

- Try this in controller :-

```
public function index (Request $request)

{
        dd($request);
}
```

// so how does the $request prints an object without instantiating it !!?

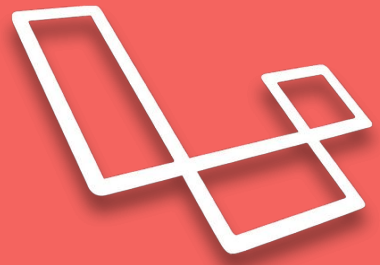


Service Container

- In the previous example , the illuminate container see if the class or method needs a dependencies it will make AutoWiring

- AutoWiring :- means if the class or method have dependencies type hinted , then it will use reflection to get the type hinted instance .

- what is reflection :- http://php.net/manual/en/book.reflection.php



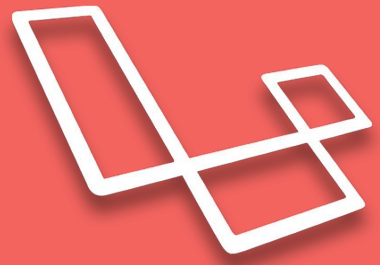
Service Container

- Binding

```
app()->bind( Car::class ,  function() {
            return  new LamboCar();
        }
);
```

- Resolving

```
app(Car::class) // means when someone asks for instance
from Car::class it will return new LamboCar instance .
```



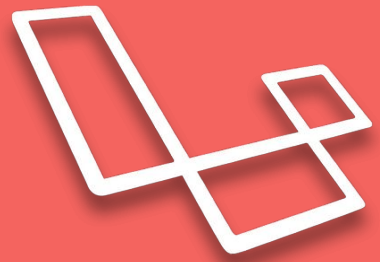
Service Container

- Service Providers :-
  This is the class where you bind services to the laravel app.

- php artisan make:provider TestProvider

- register() :- this is where you bind things into container

- boot() :- called after all register methods in other providers have been called . //this is where you register events listeners or routes and do any behaviour .

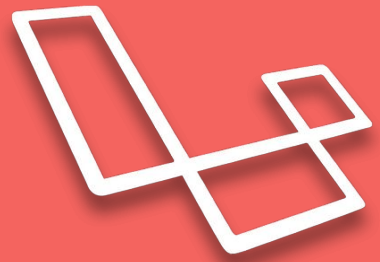


Service Container

- In config/app.php see the providers array .

- Now try to play around with this package :- https://github.com/barryvdh/laravel-debugbar

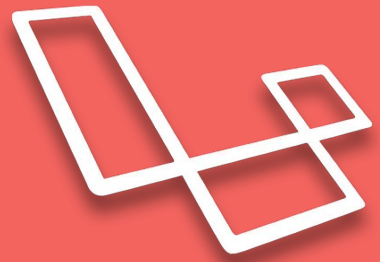
- To know more about illuminate container check Matt Stuffer talk https://laracon.net/2017



Lab 2

- Add validation using form request files on Create & Update https://laravel.com/docs/5.6/validation#creating-form-requests
- Title & description are required , minimum length for title is 3 chars and unique, for description the minimum length is 10 chars ,make sure when updating post without changing Title it still works
  Also make sure that no one hacks you and send an id of post creator that doesn't exist in the database

- Make sure to display error messages of failed validation https://laravel.com/docs/5.6/validation#quick-displaying-the-validation-errors

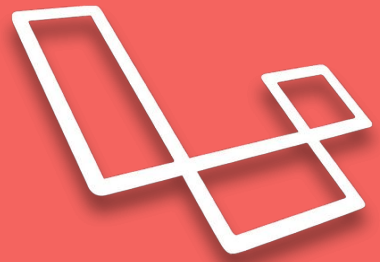


Lab 2

- Use php artisan:make auth , to scaffold the auth page

- Modify on the current navbar , and make it use the laravel default auth navbar , and also we need the link to All Posts (Do whatever you see suits the case)

- Add Authentication middleware on all posts routes , and make anyone who isn't authenticated to redirect back to login page

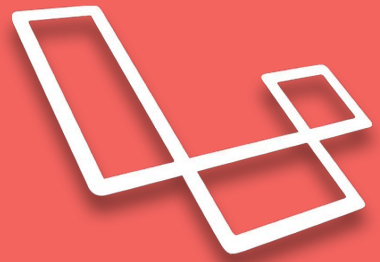


Lab 2

- Make our post have slug , using this package (the slug will be generated from the post title , users aren't allowed to fill slug or send it in the request , search for $request->validated()or $request->only())
  (Read the package documentation carefully)
  https://github.com/cviebrock/eloquent-sluggable

- Show the slug column In Index page

For Bonus Choose one

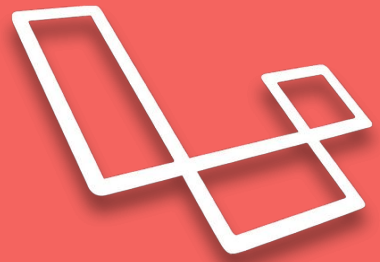


Lab 2 (Bonus 1)

- Upload image to post , and validate extensions are only (.jpg, .png) , and use Storage to store and show images also when updating post we remove the old image, and when deleting we remove the image also
  Hint:- see if Mutators can make your work easier
  https://laravel.com/docs/5.6/filesystem#file-uploads

- Add comments to post using polymorphic relation and don't use any packages , also we need to show who commented on post
  https://laravel.com/docs/5.6/eloquent-relationships#polymorphic-relations

- Use this package to add Tags to post , the user will enter comma separated tags
  https://github.com/spatie/laravel-tags



Lab 2 (Bonus 2)

- Make custom validation rule , that makes sure the user is only allowed to create 3 posts and if he exceeded this number we show a validation error message https://laravel.com/docs/5.6/validation#custom-validation-rules
- Add View Ajax Button that opens Bootstrap Modal , showing post info (title , description , slug , username, useremail) using ajax request (check laravel responsable interface in docs to make your code cleaner)

- Add restore button on index page to restore deleted posts you will need to use soft delete https://laravel.com/docs/5.6/eloquent#soft-deleting