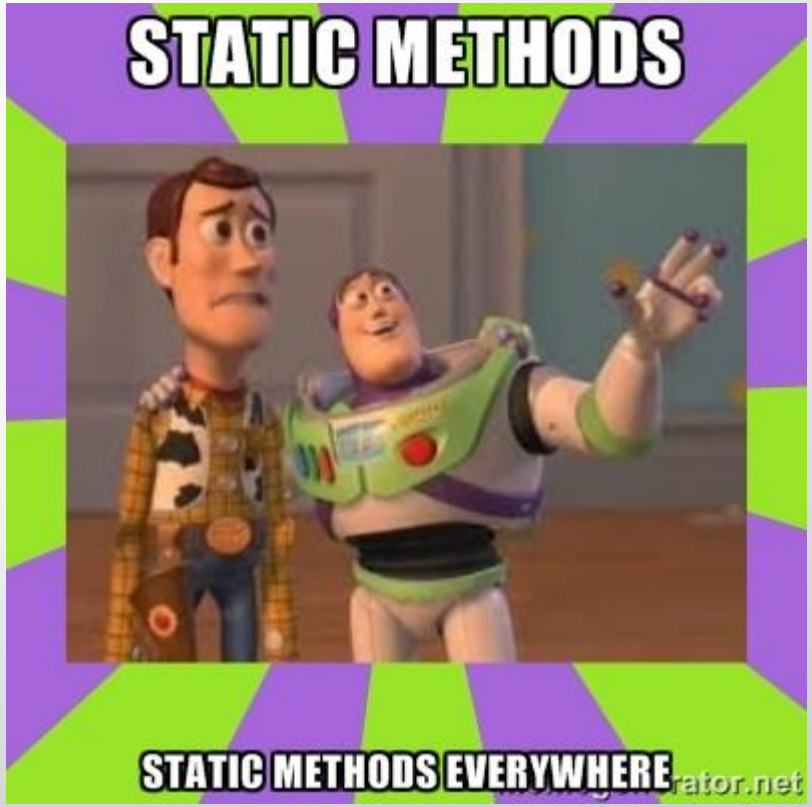
We will learn today …

- What is Laravel ?
- Install Laravel 5 with Composer
- Files structure
- What is artisan and how does it save us time?
- Routing and route types
- What is Middleware and how to use it?
- What is Blade ?
- Database and Eloquent ORM
- CRUD with validation and database connection (practical task)
- Best practices when coding in Laravel

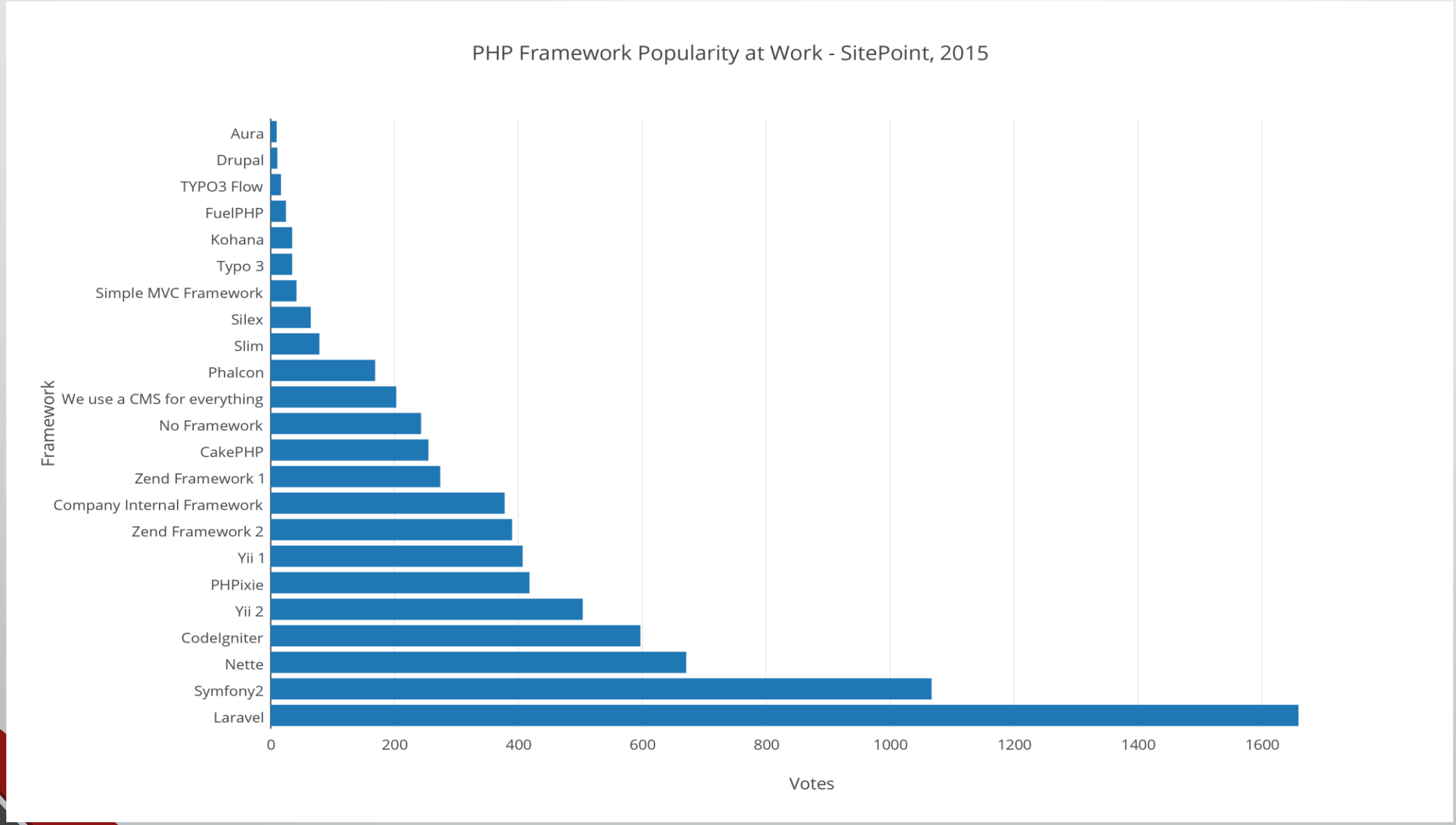


Laravel

What is Laravel ?

- Laravel is MVC PHP framework created by **Taylor Otwell** in **2011**
- Free open-source license with many contributors worldwide
- One of the best frameworks together with Symfony, CodeIgniter, Yii
- Has powerful features, saving us time
- Uses Symfony packages
- Lets see some statistics

PHP Framework Popularity at Work – SitePoint 2015



Features

- **Eloquent ORM** (object-relational mapping) – implements ActiveRecord
- **Query builder** – helps you to build secured SQL queries
- **Restful controllers** – provides a way for separating the different HTTP requests (GET, POST, DELETE, etc.)
- **Blade template engine** – combines templates with a data model to produce views
- **Migrations** – version control system for database, update your database easier
- **Database seeding** – provides a way to populate database tables with test data used for testing
- **Pagination** – easy to use advanced pagination functionalities
- **Forms security** – provides CSRF token middleware, protecting all the forms

Must have packages

- **Laravel debugbar - https://github.com/barryvdh/laravel-debugbar**

Great for debugging on local environment. Shows all the views, requests, exceptions loaded for the current page.

- **LaravelCollective – Forms & HTML - https://laravelcollective.com/docs/master/html**

Perfect for generating forms, inputs, script tags and style tags

- **Laravel IDE Helper - https://github.com/barryvdh/laravel-ide-helper**

The package helps your IDE with autocomplete and autosuggest methods, views, functions and more.

Let's install Laravel

- Laravel uses Composer to manage its dependencies
- Composer is dependency management tool for PHP, like a library full of books
- **NOT** like Yum or apt
- Per project tool (vendor folder), not per system
- Install by using the command:

composer create-project --prefer-dist laravel/laravel laravel-softuni

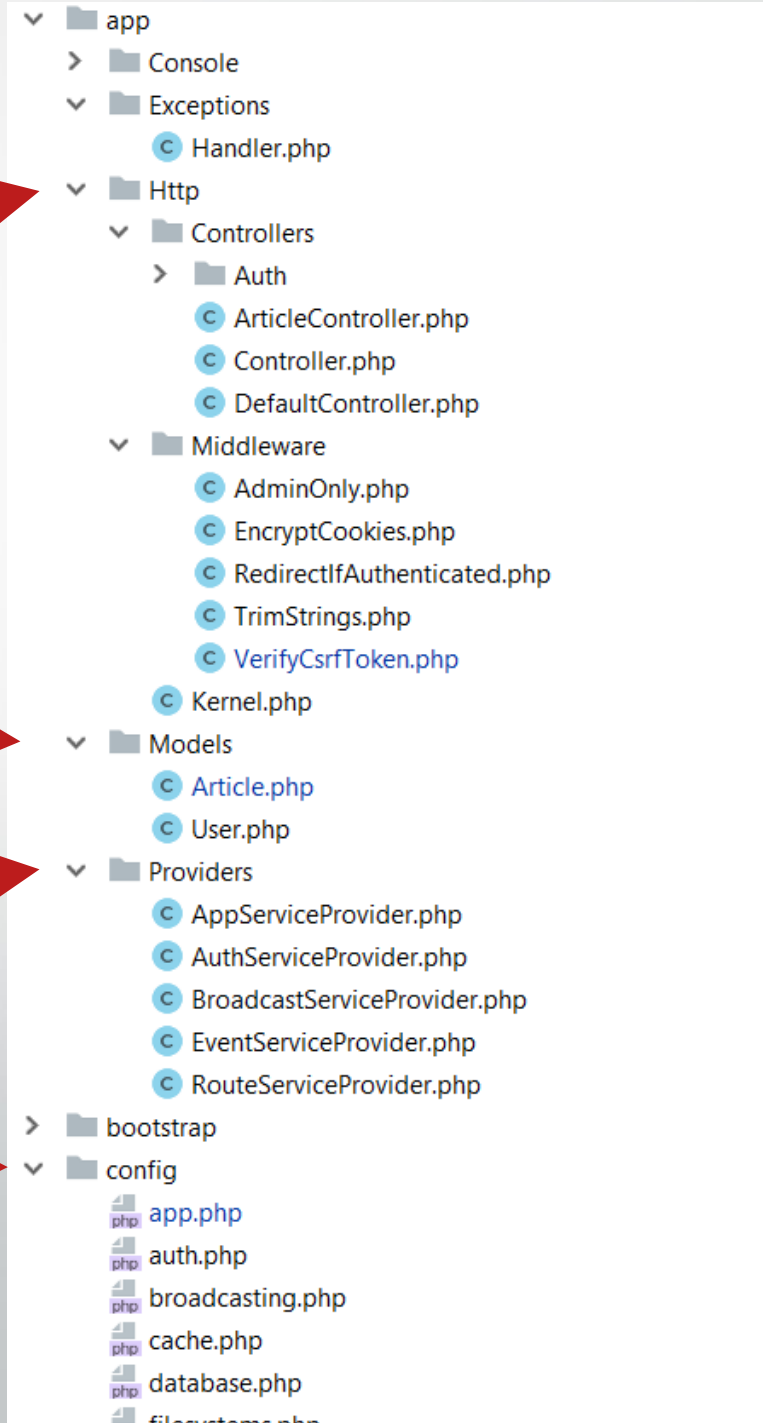
The structure

app/Http folder contains the **Controllers, Middlewares** and **Kernel** file

All the models should be located in **app/Models** folder

The service providers that are bootstrapping functions in our app are located in **app/Providers** folder

All the config files are located in **app/config** folder



Database folder contains the **migrations and seeds**

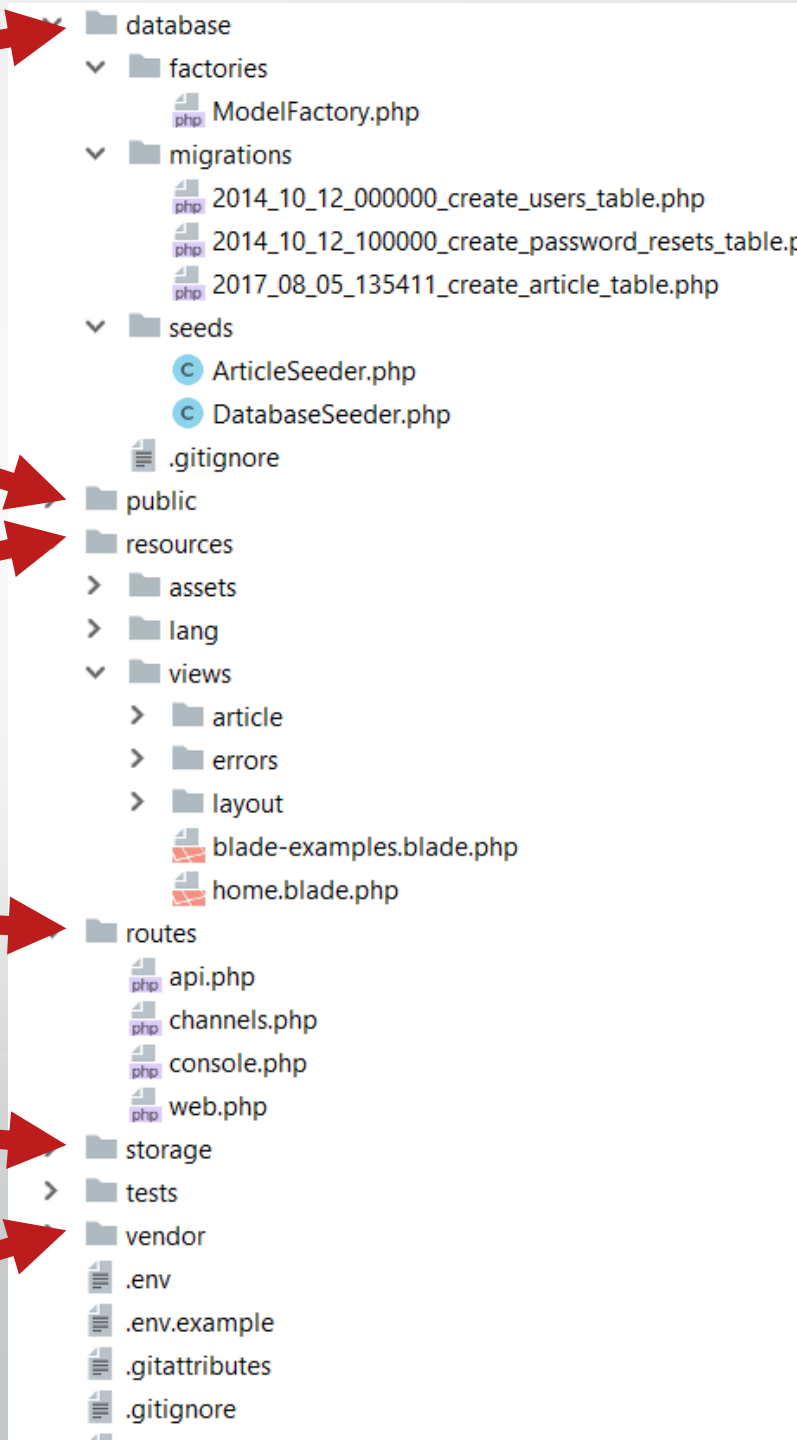
The public folder is the actual folder you are opening on the web server.
All JS / CSS / Images / Uploads are located there.

The resources folder contains all the **translations, views** and **assets** (SASS, LESS, JS) that are compiled into public folder

The routes folder contains all the routes for the project

All the **logs** / **cache** files are located in **storage** folder

The **vendor** folder contains all the composer packages (dependencies)



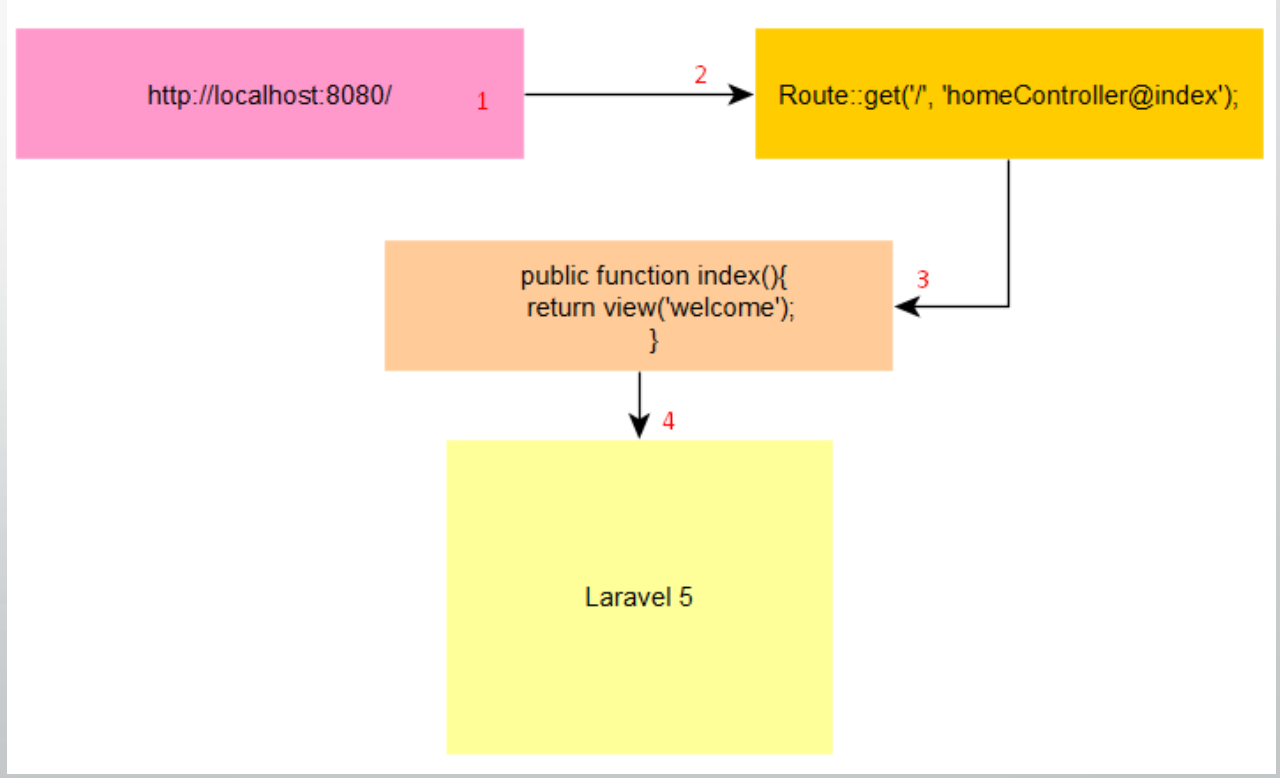
Artisan !

- **Artisan** is command-line interface for **Laravel**
- Commands that are saving time
- Generating files with artisan is **recommended**
- Run **php artisan list** in the console

```
app
 app:name              Set the application namespace
auth
 auth:clear-resets     Flush expired password reset tokens
cache
 cache:clear           Flush the application cache
 cache:forget          Remove an item from the cache
 cache:table           Create a migration for the cache database table
config
 config:cache          Create a cache file for faster configuration loading
 config:clear          Remove the configuration cache file
db
 db:seed               Seed the database with records
event
 event:generate        Generate the missing events and listeners based on registration
ide-helper
 ide-helper:generate   Generate a new IDE Helper file.
 ide-helper:meta       Generate metadata for PhpStorm
 ide-helper:models     Generate autocompletion for models
key
 key:generate          Set the application key
make
 make:auth             Scaffold basic login and registration views and routes
 make:command          Create a new Artisan command
 make:controller       Create a new controller class
 make:event            Create a new event class
 make:job              Create a new job class
 make:listener         Create a new event listener class
 make:mail             Create a new email class
 make:middleware       Create a new middleware class
 make:migration        Create a new migration file
 make:model            Create a new Eloquent model class
 make:notification     Create a new notification class
 make:policy           Create a new policy class
 make:provider         Create a new service provider class
 make:request          Create a new form request class
 make:seeder           Create a new seeder class
 make:test             Create a new test class
migrate
 migrate:install       Create the migration repository
 migrate:refresh       Reset and re-run all migrations
 migrate:reset         Rollback all database migrations
 migrate:rollback      Rollback the last database migration
 migrate:status        Show the status of each migration
notifications
 notifications:table   Create a migration for the notifications table
queue
 queue:failed          List all of the failed queue jobs
 queue:failed-table    Create a migration for the failed queue jobs database table
 queue:flush           Flush all of the failed queue jobs
```

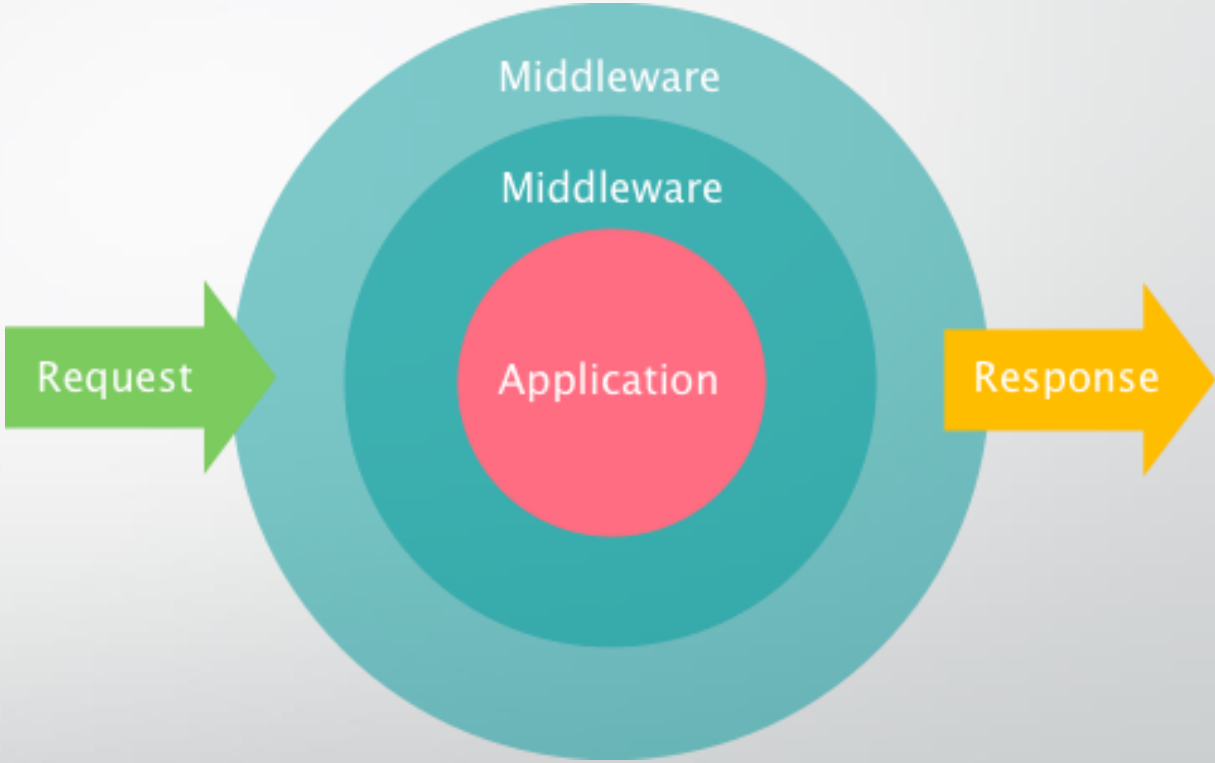
Routing

- The best and easy routing system I’ve seen
- Routing per middleware / prefix or namespace
- Routing per request method (GET, POST, DELETE, etc.)
- **ALWAYS** name your route !
- Be careful with the routing order !
- Let’s see routing examples



Middleware

- The middleware is mechanism for filtering the HTTP requests
- Laravel includes several middlewares – Authentication, CSRF Protection
- The auth middleware checks if the user visting the page is authenticated through session cookie
- The CSRF token protection middleware protects your application from cross-site request forgery attacks by adding token key for each generated form
- Let's create middleware



Blade

- Blade is the powerful template engine provided by Laravel
- All the code inside blade file is compiled to static html file
- Supports plain PHP
- Saves time
- Better components mobility, extend and include partials
- Let’s take a look at few examples

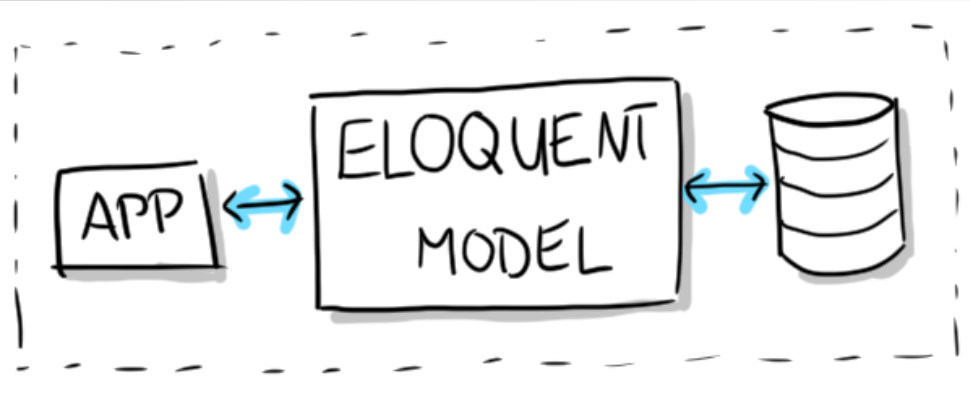
Eloquent & Database

- The Eloquent ORM (Object-relational mapping) provides simple ActiveRecord implementation for working with the database

```
$article = new Article();
$article->title = 'Article title';
$article->description = 'Description';
$article->save();
```

```
INSERT INTO `article` (`title`, `description`) VALUES ('Article title', 'Description');
```

- Each table has its own “Model”. You can use the model to read, insert, update or delete row from the specific table
- Let’s check one model



Laravel model

ClassName	Singular of table name	`protected $table = 'custom_name'`
Primary key	id	`protected $primaryKey`
Timestamp	created_at, updated_at	`protected $timestamp = false`
Guarded	array of fields name	`protected $guarded = array('id', 'password')`
Fillable	array of fields name	`protected $fillable = array('id', 'password')`

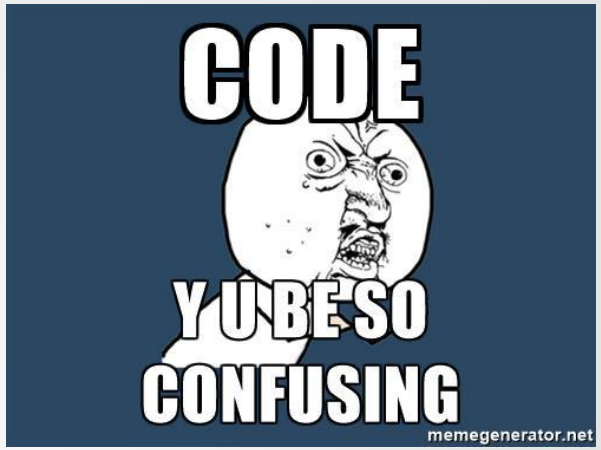
Practical task

We will play with Laravel and create CRUD for recipes (Create, Read, Update, Delete).

The recipe will have the following columns / fields :

- Id – primary key – not null
- Title – varchar – 255 length – not null
- Description – text – nullable
- Status – enum [active / inactive] – not null – defaults to active
- Created At – datetime – not null
- Updated At – datetime – not null

Best practices in Laravel



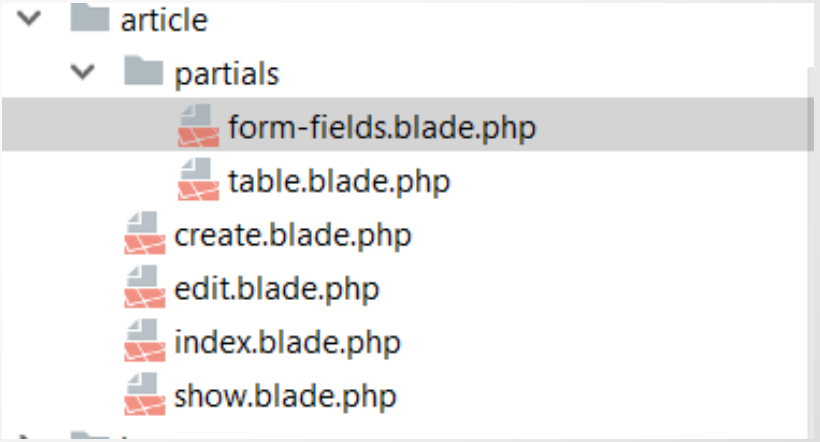
NEVER write queries or model logic inside the controller! The controller job is to communicate with the model and pass data to the view.

```php
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    $goodArticles = Article::latestPaginatedArticles( limit: 6);

    $badArticles = Article::orderBy( column: 'created_at', direction: 'DESC')->paginate( perPage: 6);
```

Views mobility

Extend and include partials. For example share the same form fields on 2 pages – add and edit




```
{!! Form::open(['route' => 'article.store', 'class' => 'form-horizontal']) !!}
    @include('article.partials.form-fields')

    <div class="form-group">
        <div class="col-sm-2 col-sm-offset-10">
            <button class="btn btn-success btn-block" type="submit">
                Add
            </button>
        </div>
    </div>
{!! Form::close() !!}
```

Forms security

Always use the CSRF token protection that Laravel provides in forms you create, the hackers will not be able to spam your forms and database

```
<form method="POST">
    {{ csrf_field() }}

    <button type="submit" class="btn btn-success">Submit</button>
</form>
```

```
▼<form method="POST">
    <input type="hidden" name="_token" value=
    "XOJeeHtwqR62hmUD0Ed1vruMdvq9jmdUNeWWoUH1">
    <button type="submit" class="btn btn-success">Submit</button>
  </form>
```

Database architecture

Be careful with the database architecture, always use the proper length for specific column and never forget the indexes for searchable columns

```
Schema::create('article', function (Blueprint $table) {
    $table->increments( column: 'id');
    $table->string( column: 'title',  length: 200);
    $table->text( column: 'description')->nullable();
    $table->enum( column: 'status', ['active', 'inactive'])->defaults('active');
    $table->timestamps();
    $table->index(['title', 'status']);
});
```

Big query

- Avoid the big query unless you really have to do it. The big query is hard to debug and understand.
- You can merge the small queries into one to save the CPU time on server, but sometimes the query becomes way too big.


```
->leftJoin('post', 'post.id', '=', 'notification.post_id')->whereNull('post.deleted_at')
->leftJoin('user', 'user.id', '=', 'post.user_id')
->leftJoin('post_comment', 'post_comment.id', '=', 'notification.comment_id')
->where(function ($query) use ($loggedUser) {
    $query->where('notification.type', '!=', 'video_completed')
        ->where('notification.by_user_id', '!=', $loggedUser->id)
        ->whereNotIn('notification.by_user_id', Magic::getBlockedUserIds())
        ->where(function ($query) use ($loggedUser) {
            $query->where(function ($query) use ($loggedUser) {
                $query->where('post.user_id', $loggedUser->id)
                    ->where(function ($query) use ($loggedUser) {
                        $query->where('notification.type', '!=', 'post_tag')
                            ->where('notification.type', '!=', 'comment_tag');
                    });
            })->orWhere(function ($query) use ($loggedUser) {
                $query->where('notification.user_id', $loggedUser->id)
                    ->where(function ($query) use ($loggedUser) {
                        $query->where('notification.type', 'post_tag')
                            ->orWhere('notification.type', 'comment_tag');
                    })
                    ->where('post.status', 'active')
                    ->where('post.completed', 'yes');
            })->orWhere(function ($query) use ($loggedUser) {
                $taggedInPosts = $loggedUser->taggedInPosts();
                $query->where(function ($query) use ($loggedUser) {
                    $query->where('notification.type', 'post_like')
                        ->orWhere('notification.type', 'post_comment');
                })
                    ->whereIn('notification.post_id', $taggedInPosts);
            });
        })
        ->whereNull('user.deleted_at');
})
->orWhere(function ($query) use ($loggedUser) {
    $query->where('notification.type', 'video_completed')
        ->where('notification.by_user_id', $loggedUser->id)
        ->whereNull('user.deleted_at');
})
```

Don’t forget the PHPDoc

Don’t forget to write comments for each method or complicated logic. The PHPDoc comments are helping the IDE to autosuggest easier and the developers to understand the piece of code

```php
/**
 * Get the latest articles
 *
 * @param int $limit
 *
 * @return \Illuminate\Support\Collection
 */
public static function latestArticles(int $limit)
{
    return self::limit($limit)
        ->orderBy( column: 'created_at',  direction: 'DESC')
        ->get();
}
```