

Laravel

#1

Introduction to the framework

write@robertogallea.com | <http://www.robertogallea.com>

What is Laravel

- ▶ Open Source PHP-based MVC framework
- ▶ Lightweight
- ▶ Many out-of-the-box features
 - ▶ Multi-user authentication and authorization
 - ▶ Templating engine
 - ▶ Unit-Testing engine
 - ▶ Security
 - ▶ ...
- ▶ Large and active community

Workshop outline

- ▶ About Laravel
- ▶ Installation
- ▶ Laravel folder structure
- ▶ Build sample application

Installation

- ▶ Requirements (as for v5.6, latest LTS)
 - ▶ PHP $\geq 7.1.3$
 - ▶ OpenSSL PHP Extension
 - ▶ PDO PHP Extension
 - ▶ Mbstring PHP Extension
 - ▶ Tokenizer PHP Extension
 - ▶ XML PHP Extension
 - ▶ Ctype PHP Extension
 - ▶ JSON PHP Extension
- ▶ Composer
- ▶ Mysql (or other supported DBMS)

Installation

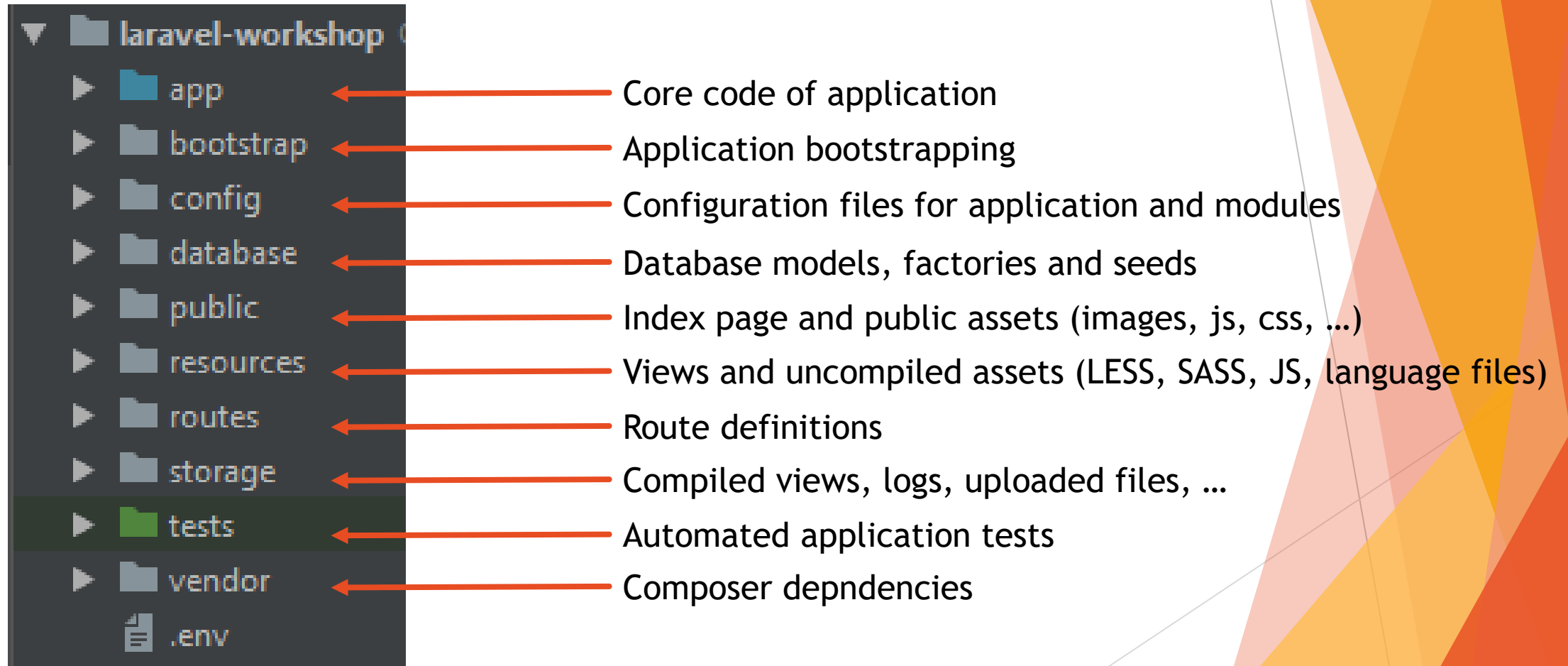
- ▶ After installing composer...

```
composer global require "laravel/installer"
```

```
laravel new blog
```

```
php artisan serve
```

Directory structure

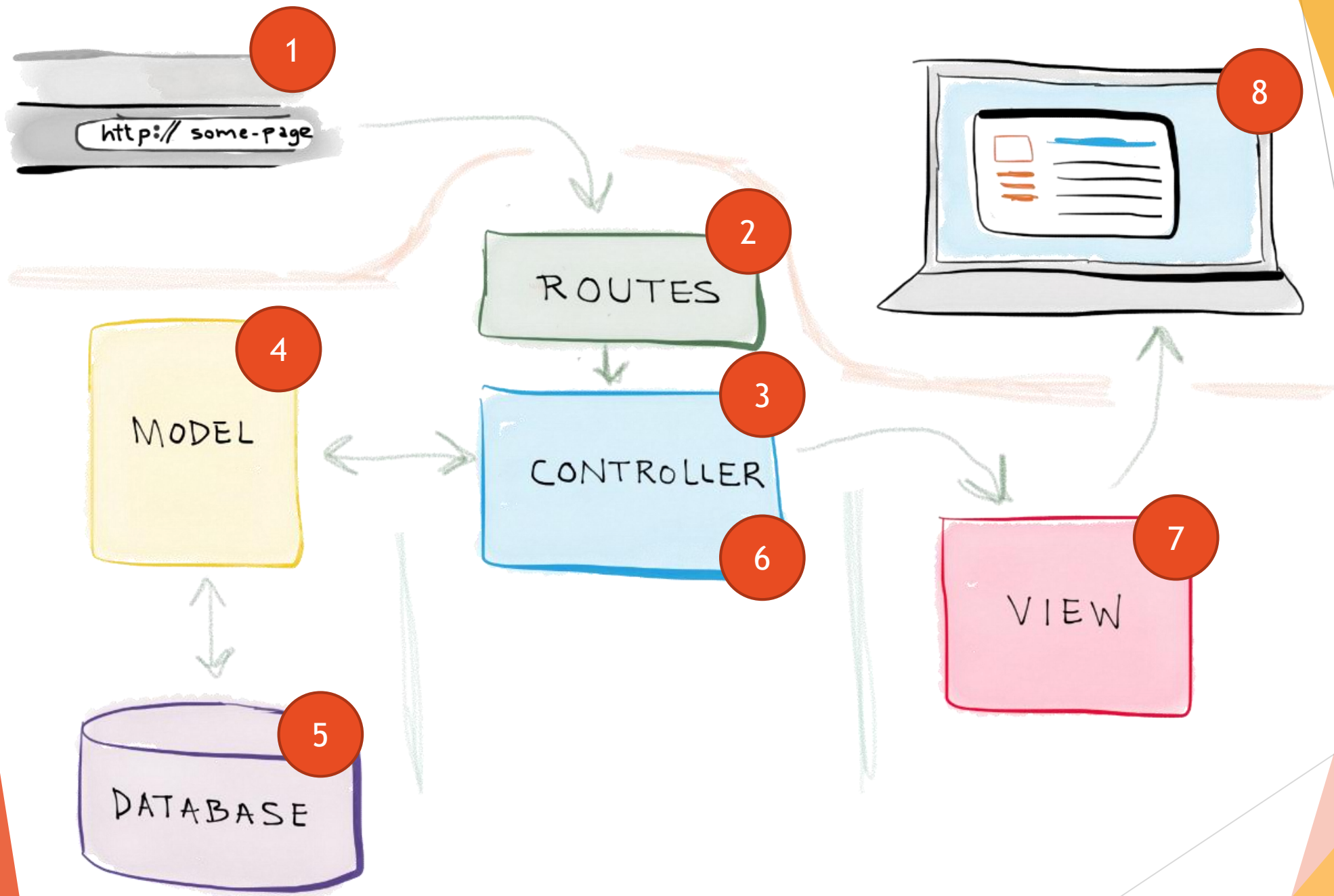


Demo application

Blog backend

First application

- ▶ Blog back-end
- ▶ Create, list and delete blog posts
- ▶ Posts have title, content, author, creation datetime



Database migrations

Database meta-model

Database setup

- ▶ Create mysql database «laravel_tutorial»

- ▶ Using phpMyAdmin

- ▶ ..or with command line

- ▶ >> mysql -u root -p

- ▶ >> create database laravel_tutorial

- ▶ Edit .env file:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=blog
DB_USERNAME=root
DB_PASSWORD=
```

What are migrations?

- ▶ Migration is like source code versioning for database model
- ▶ Used for keeping in sync database models across develop machines
- ▶ Prevent from manually adding/removing tables/columns or other database elements

Create database schema (migration)

```
php artisan make:migration create_posts_table --create=posts
```

Migration name



Table name





```
public function up( )
{
    Schema::create('posts', function (Blueprint $table) {
        $table->increments('id');
        $table->string('title',100);
        $table->string('author',100);
        $table->text('content');
        $table->timestamps();
    });
}
```

Run migration

- ▶ Each migration file created with a timestamp followed by name

`php artisan migrate`

- ▶ Executes the migrations in order
- ▶ Check the database, «posts» table has been created

Models

Objects mapping database entities

Create Model

- ▶ Laravel Object-Relational Mapping (ORM)
- ▶ Create Post model using

`php artisan make:model Post`

- ▶ `app/Post.php` created



```
<?php
```

```
namespace App;
```

```
use Illuminate\Database\Eloquent\Model;
```

```
class Post extends Model  
{  
    //  
}
```

Conventions

Every column is normally accessed as `$obj->col_name` using PHP *magic methods*

Object	In DB	In Eloquent	Override
Table	«posts» assumed	class Post	protected \$table = 'my_posts'
Key	«id» assumed	\$obj->id	protected \$primaryKey = 'p_id'
Creation date	«created_at» assumed	\$obj->created_at	const CREATED_AT = 'creation_date'
Update date	«updated_at» assumed	\$obj->updated_at	const UPDATED_AT = 'update_date'

Routes

The application endpoints

Defining routes

- ▶ Define web endpoints for the application
 - ▶ GET / - lists the available posts
 - ▶ POST /post - creates a new post
 - ▶ DELETE /post/{id} - deletes a post with {id} key
- ▶ Edit file routes/web.php



```
use App\Task;
use Illuminate\Http\Request;

/**
 * Display All Posts
 */
Route::get('/', function() {
    //
});

/**
 * Add A New Task
 */
Route::post('/post', function (Request $request) {
    //
});

/**
 * Delete An Existing Task
 */
Route::delete('/post/{id}', function ($id) {
    //
});
```

Views

Presentation of content

Displaying a view

- ▶ Views are defined as files named {filename}.blade.php under /resources/views
- ▶ Display a view from the controller returning it by its name

Main page



```
Route::get('/', function () {  
    return view('posts');  
});
```

Defining a template layout

- ▶ Create the file
`resources/views/layout.blade.php`



```
<html>
<head>
    <title>Laravel tutorial</title>
</head>
<body>
<h1>Posts application</h1>
@yield( 'content' )
</body>
</html>
```

Defining a template view

- ▶ Create the file
`resources/views/posts.blade.php`

```
@extends('layout')

@section('content')
    @include('errors')
    <!-- TODO Posts table -->

    <!-- New post form -->
    <form method="post" action="/post">
        {{ csrf_field() }}
        <table>
            <tr>
                <td><label for="title">Title</label></td>
                <td><input type="text" id="title" name="title"/><br/></td>
            </tr>
            <tr>
                <td><label for="author">Author</label></td>
                <td><input type="text" id="author" name="author"/><br/></td>
            </tr>
            <tr>
                <td><label for="content">Content</label></td>
                <td><textarea rows="10" cols="100" id="content" name="content"></textarea><br/></td>
            </tr>
            <tr>
                <td></td>
                <td><button type="submit">Create post</button></td>
            </tr>
        </table>
    </form>
@endsection
```

Define a template sub-view

- ▶ Create the file
`resources/views/errors.blade.php`



```
<!-- error page -->
@if (count($errors) > 0)
<h2>Errors!</h2>
<ul>
    @foreach($errors->all() as $error)
        <li>{{ $error }}</li>
    @endforeach
</ul>
@endif
```

Layout.blade.php

<h1>abc</h1>

@yield('content')

posts.blade.php

@extends('layout')

@section('content')

@include('errors')

<div>content</div>

@endsection

errors.blade.php

<p>print errors</p>

Rendered view

<h1>abc</h1>


<p>print errors</p>

<div>content</div>

Post creation (input validation)

```
Route::post('/post', function (Request $request) {  
    $validator = Validator::make($request->all(), [  
        'title' => 'required|max:100',  
        'author' => 'required|max:100',  
        'content' => 'required',  
    ]);  
  
    if ($validator->fails()) {  
        return redirect('/')  
            ->withInput()  
            ->withErrors($validator);  
    }  
  
    // create post  
  
    return redirect('/');  
});
```

Post creation (object creation)



```
$post = new Post();  
$post->title = $request->title;  
$post->author = $request->author;  
$post->content = $request->content;  
$post->save();
```

Loading data

- ▶ Laravel uses Eloquent engine
- ▶ Example: load blog posts ordered by ascending creation date

```
$posts = Post::orderBy('created_at', 'asc')->get();
```

Passing data to view



```
Route::get('/', function () {  
    $posts = Post::orderBy('created_at', 'asc')->get();  
  
    return view('posts', [  
        'posts' => $posts  
    ]);  
});
```

Pass \$posts variable to view
and name it «posts»



```
@extends('layout')

@section('content')
    @include('errors')
    <!-- Posts table -->
    <table style="width: 100%; border: 1px solid black;">
        <tr>
            <th>Title</th>
            <th>Author</th>
            <th>Content</th>
            <th>Creation date</th>
            <th>Actions</th>
        </tr>
        @foreach($posts as $post)
            <tr>
                <td>{{ $post->title }}</td>
                <td>{{ $post->author }}</td>
                <td>{{ $post->content }}</td>
                <td>{{ $post->created_at }}</td>
                <td></td>
            </tr>
        @endforeach
    </table>

    <!-- New post form -->

@endsection
```

Deleting posts



```
Route::delete('/post/{id}', function ($id) {  
    Post::findOrFail($id)->delete();  
  
    return redirect('/');  
});
```

```
@extends('layout')

@section('content')
    @include('errors')
    <!-- Posts table -->
    <table style="width: 100%; border: 1px solid black;">
        <tr>
            <th>Title</th>
            <th>Author</th>
            <th>Content</th>
            <th>Creation date</th>
            <th>Actions</th>
        </tr>
        @foreach($posts as $post)
            <tr>
                <td>{{ $post->title }}</td>
                <td>{{ $post->author }}</td>
                <td>{{ $post->content }}</td>
                <td>{{ $post->created_at }}</td>
                <td>
                    <form method="post" action="/post/{{ $post->id }}">
                        {{ csrf_field() }}
                        {{ method_field('DELETE') }}
                        <button type="submit">Delete post</button>
                    </form>
                </td>
            </tr>
        @endforeach
    </table>

    <!-- New post form -->

@endsection
```

Using Controllers

- ▶ Till now we used closures for managing routes
- ▶ Delegate actions to **Controllers**
- ▶ Run *php artisan make:controller PostController*
 - ▶ Creates Http/Controllers/PostController.php
- ▶ Edit web.php



Controller Method



```
Route::get('/', 'PostController@index');  
Route::post('/post', 'PostController@store');  
Route::delete('/post/{post}', 'PostController@destroy');
```

```
public function index()
{
    $posts = Post::orderBy('created_at','desc')->get();

    return view('posts')->with([
        'posts' => $posts,
    ]);
}

public function store(Request $request) {
    $validator = Validator::make($request->all(), [
        'title' => 'required|max:100',
        'content' => 'required',
    ]);

    if ($validator->fails()) {
        return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }

    $post = new Post();
    $post->title = $request->title;
    $post->content = $request->content;
    $post->user_id = Auth::user()->id;
    $post->save();

    return redirect('/');
}

public function destroy($post) {
    $post = Post::findOrFail($post);
    $post->delete();

    return redirect('/');
}
```

Authentication scaffolding

Leveraging built-in authentication layer

Auth Scaffolding

- ▶ Add Auth scaffolding

`php artisan make:auth`

- ▶ Adds:
 - ▶ `Auth::routes();` in `web.php`. Try launching *php artisan route:list*
 - ▶ Four controllers under `Controllers/Auth`
 - ▶ Two views under `resources/views`
 - ▶ Two views under `resources/views/password`

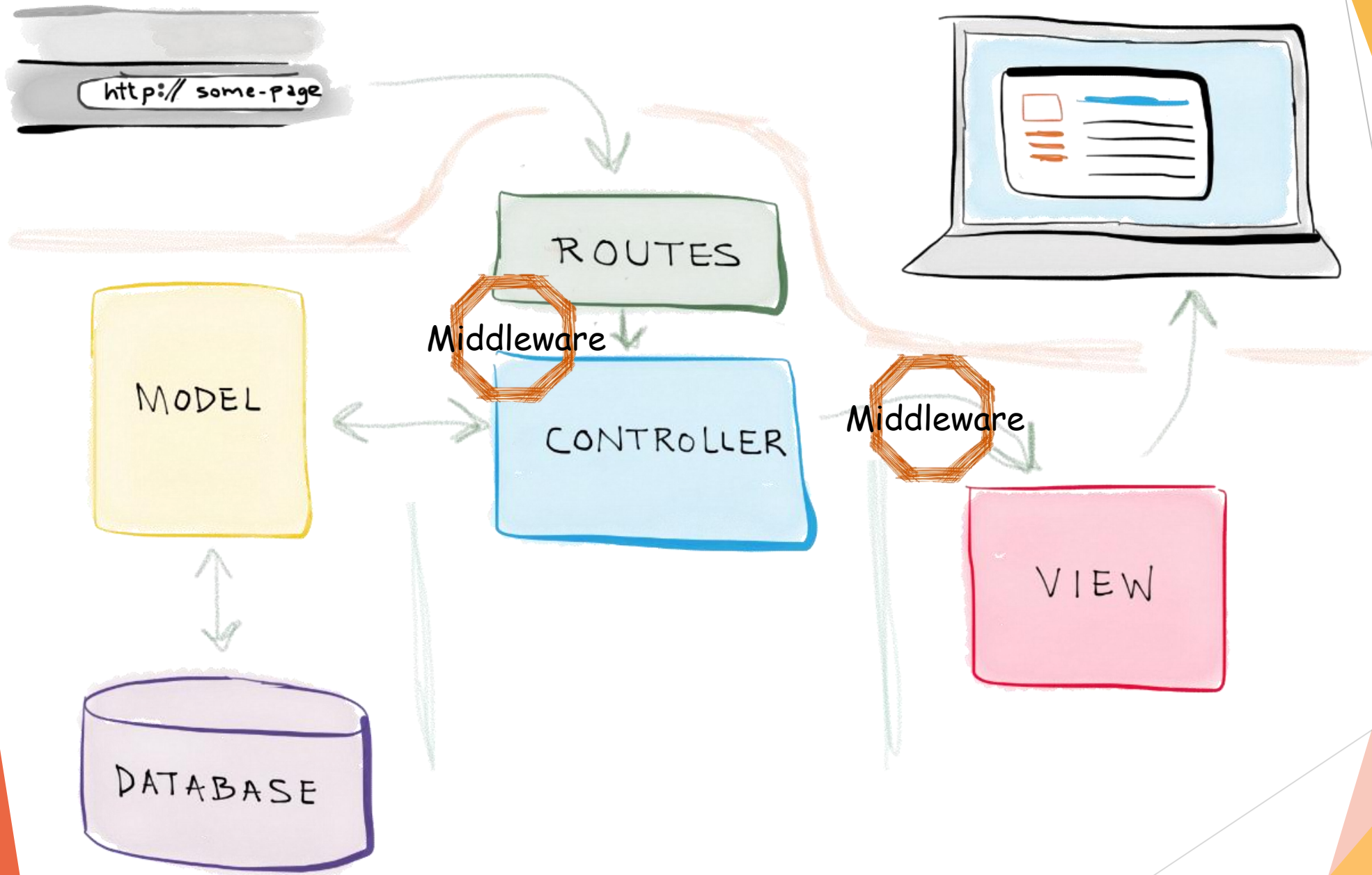
Protecting resources

- ▶ Use Auth Middleware
- ▶ It redirects user to login if not authenticated

Middleware: *Piece of code that filters request **before** or **after** the request is executed*

- ▶ Inside PostController.php →

```
public function __construct()  
{  
    $this->middleware('auth');  
}
```



Model relations

One-to-many example

Post-User Relation

- ▶ Each post is written by one user
- ▶ Each user writes many posts
- ▶ One-to-many relation
- ▶ Use of model relations

Required modifications

- ▶ Modify field in database model (migration)
- ▶ Add relation to User model in Post model
- ▶ Add relation to Post model in User model
- ▶ Modify PostController@store to save current user id as author
- ▶ Modify view to...
 - ▶ Not ask for author since author is current user
 - ▶ In posts list, retrieve author name from author relation

Implementing relation in blog application

- ▶ Remove «author» string field from create_posts_table migration
- ▶ Add «user_id» integer field in create_posts_table migration
 - ▶ `$table->integer('user_id')->index();` // Good idea to set it as index
- ▶ Rerun migration
 - ▶ `php artisan migrate:fresh`
- ▶ In Post.php add fillable fields
 - ▶ Only Fillable attributes can be mass assigned (more on this later)
 - ▶ `protected $fillable = ['title', 'content'];` // in Post.php

hasMany relation

- ▶ Access posts by writing
- ▶ `$user = User::find(1);`
- ▶ `$user->posts;`



```
// Inside User.php  
public function posts()  
{  
    return $this->hasMany(Post::class);  
}
```

belongsTo relation

- ▶ Access user by writing
- ▶ `$post = Post::`
- ▶ `$post->user;`



```
// Inside Post.php  
public function user()  
{  
    return $this->belongsTo(User::class);  
}
```

Editing view to reflect relations



```
// Inside posts.blade.php  
<td class="table-text">  
    <div>{{ $post->user->name }}</div>  
</td>
```

Setting post user as the current user

- ▶ Remove author field inside posts.blade.php
- ▶ Update PostController@store

```
public function store(Request $request)
{
    $validator = Validator::make($request->all(), [
        'title' => 'required|max:100',
        'content' => 'required',
    ]);

    if ($validator->fails()) {
        return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }

    $post = new Post();
    $post->title = $request->title;
    $post->user_id = Auth::user()->id;
    $post->content = $request->content;
    $post->save();

    return redirect('/');
}
```

What's next

- ▶ Model binding
- ▶ Authorization scaffolding
- ▶ Custom Requests
- ▶ Using middlewares
- ▶ Defining middlewares
- ▶ Event dispatching, observing and listening
- ▶ ...