WORKSHOP

LARAVEL

A FRAMEWORK FOR WEB ARTISANS

Francisco Neves

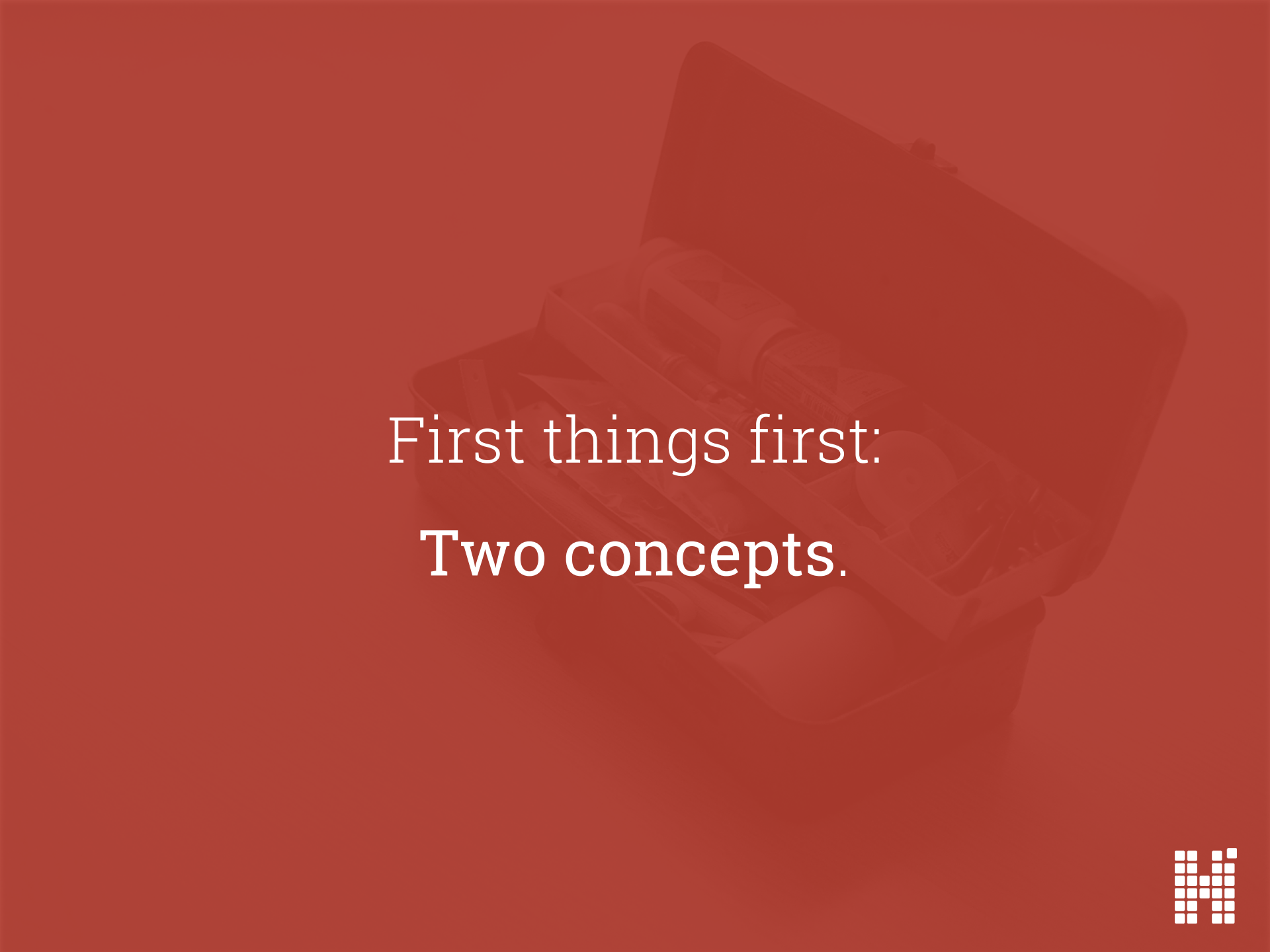
@fntneves

PhD Student at MAP-I Doctoral Programme
Researcher at HASLab, INESC TEC

FalarSobreCancro.org

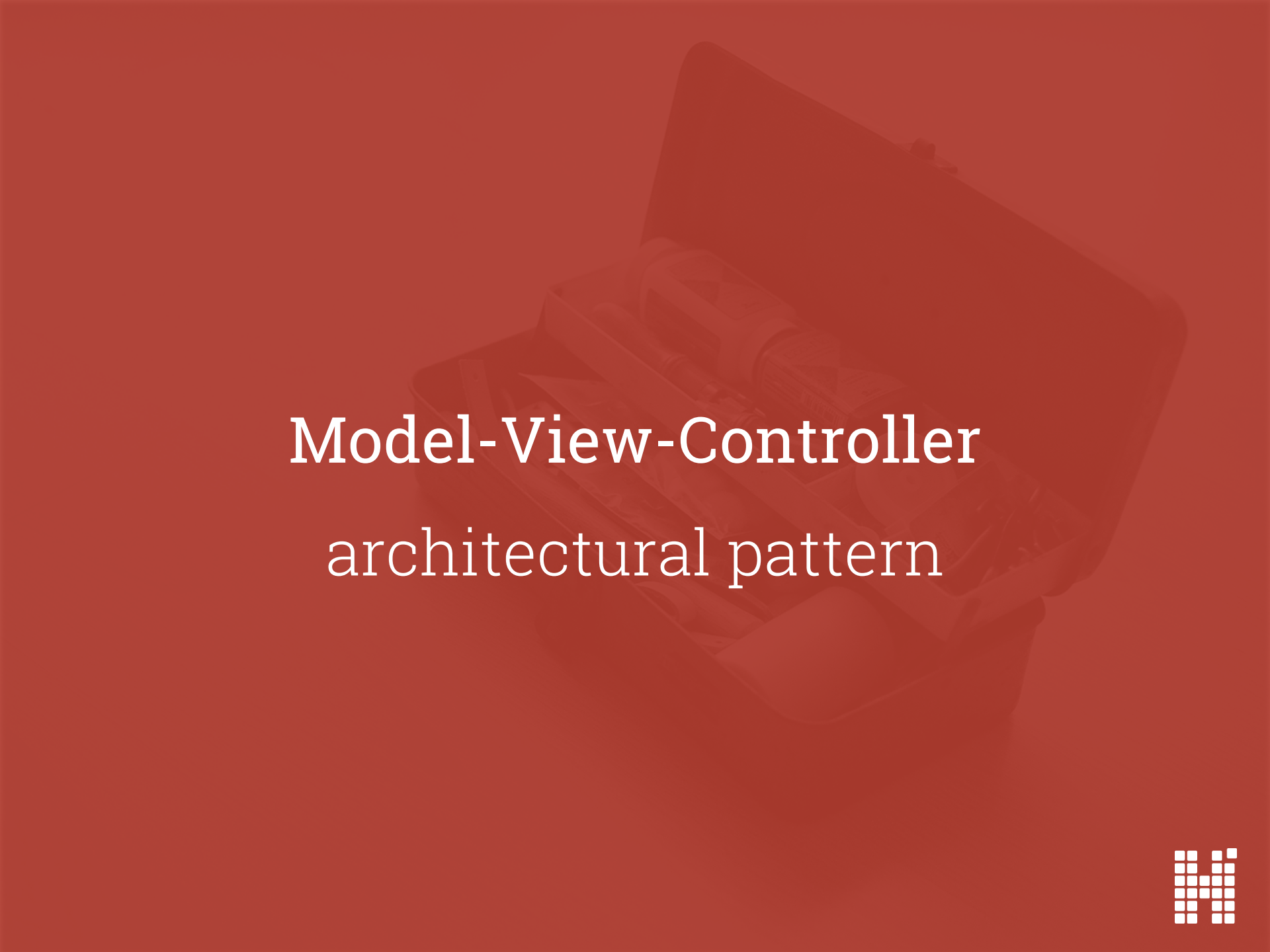
SafeCloud photos

Hackathonners



First things first:

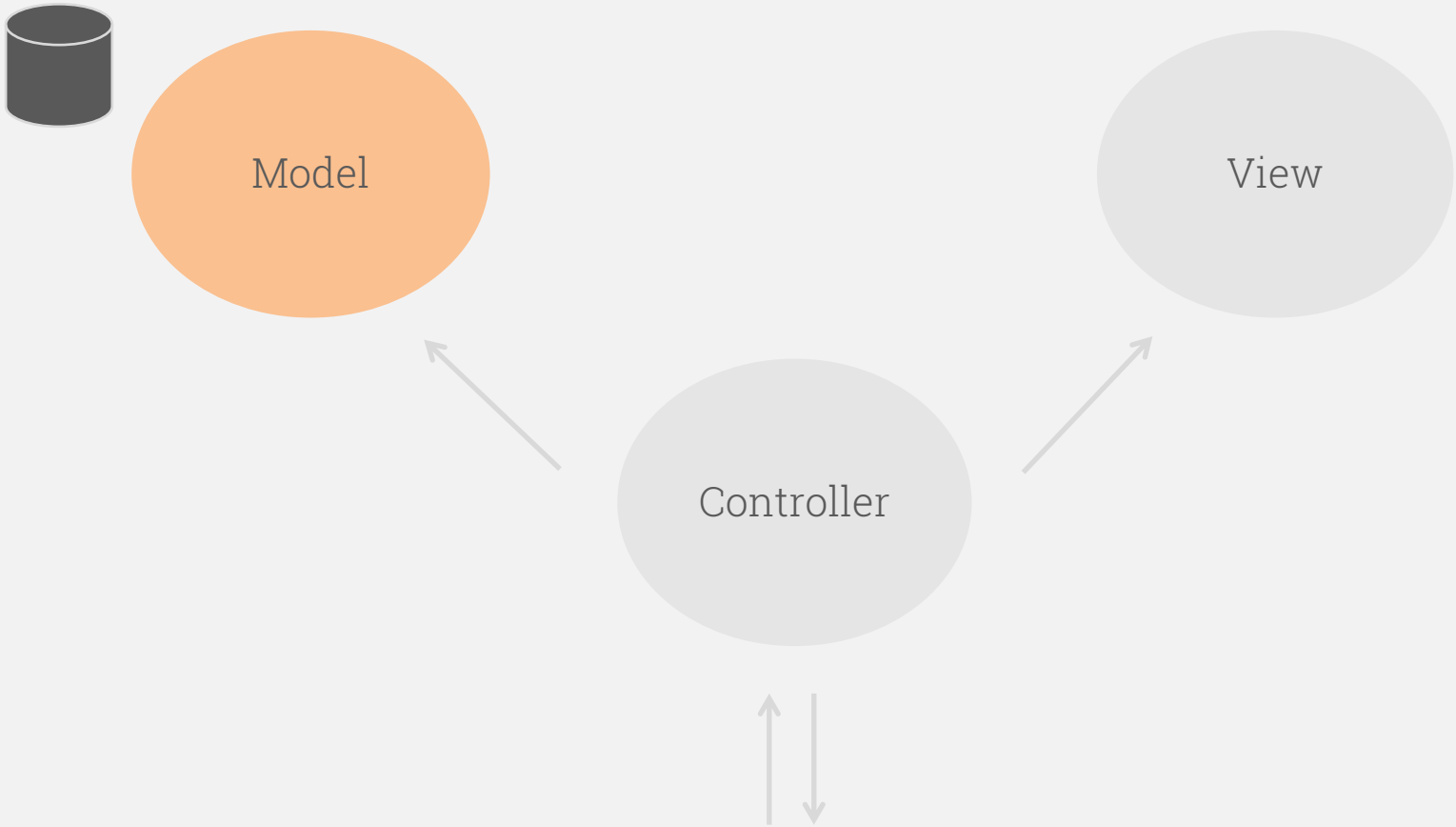
Two concepts.



Model-View-Controller

architectural pattern

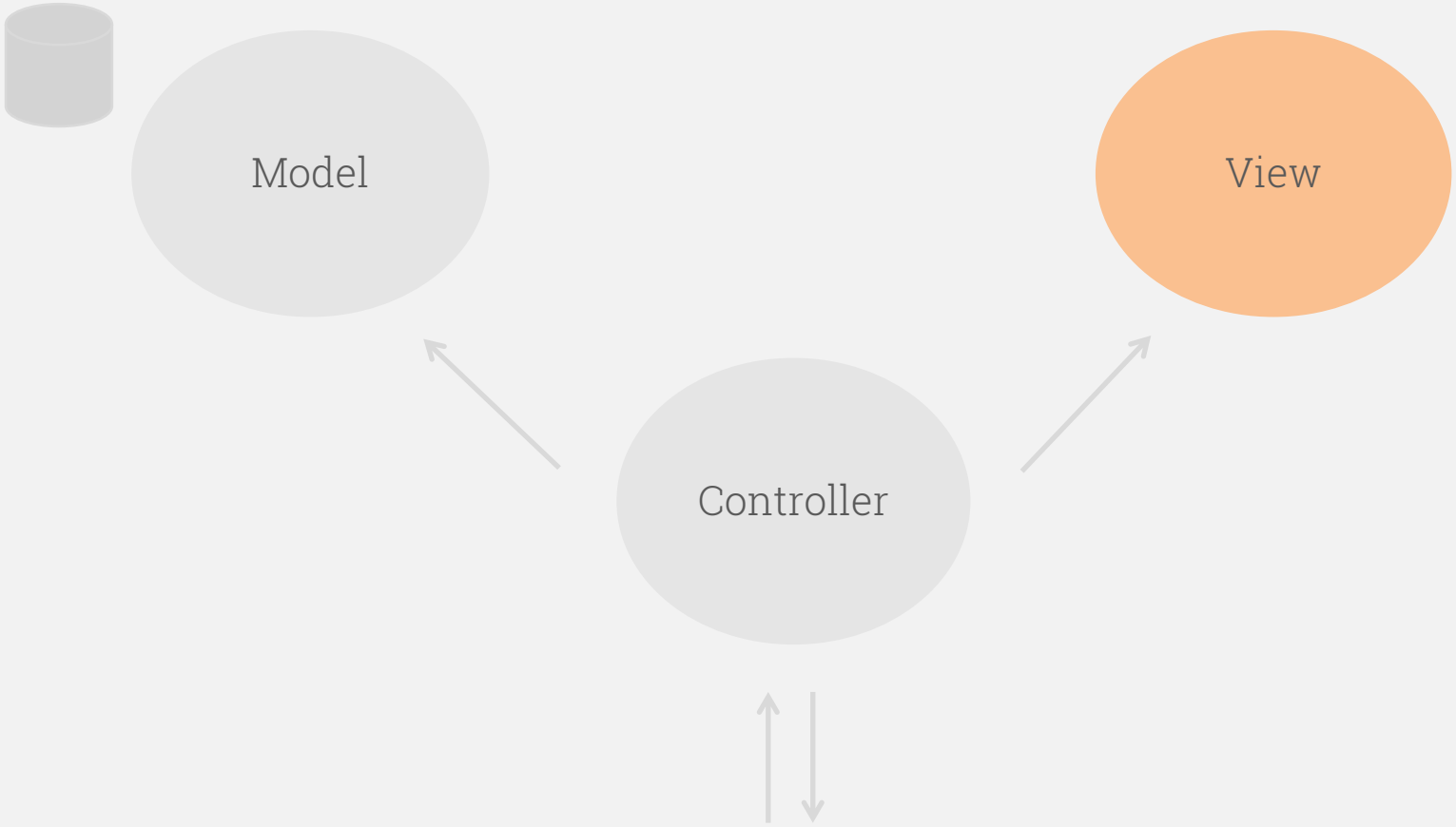
Model-View-Controller



Model accesses data and contains the business logic of each entity.

Example: Get user and check if it is registered or not.

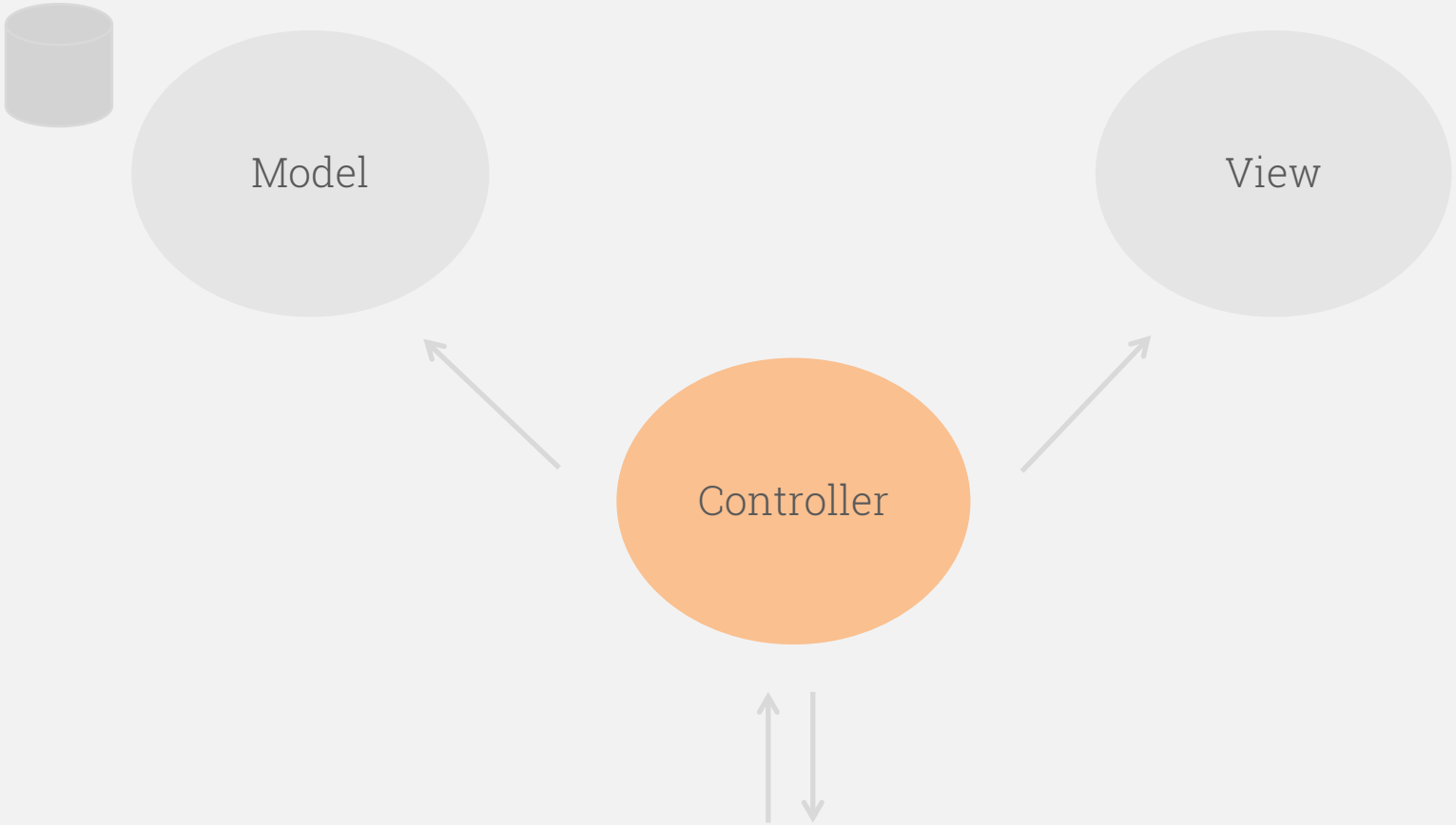
Model-View-Controller



View is the visual representation of data.

Example: Show details page of a given User ID

Model-View-Controller



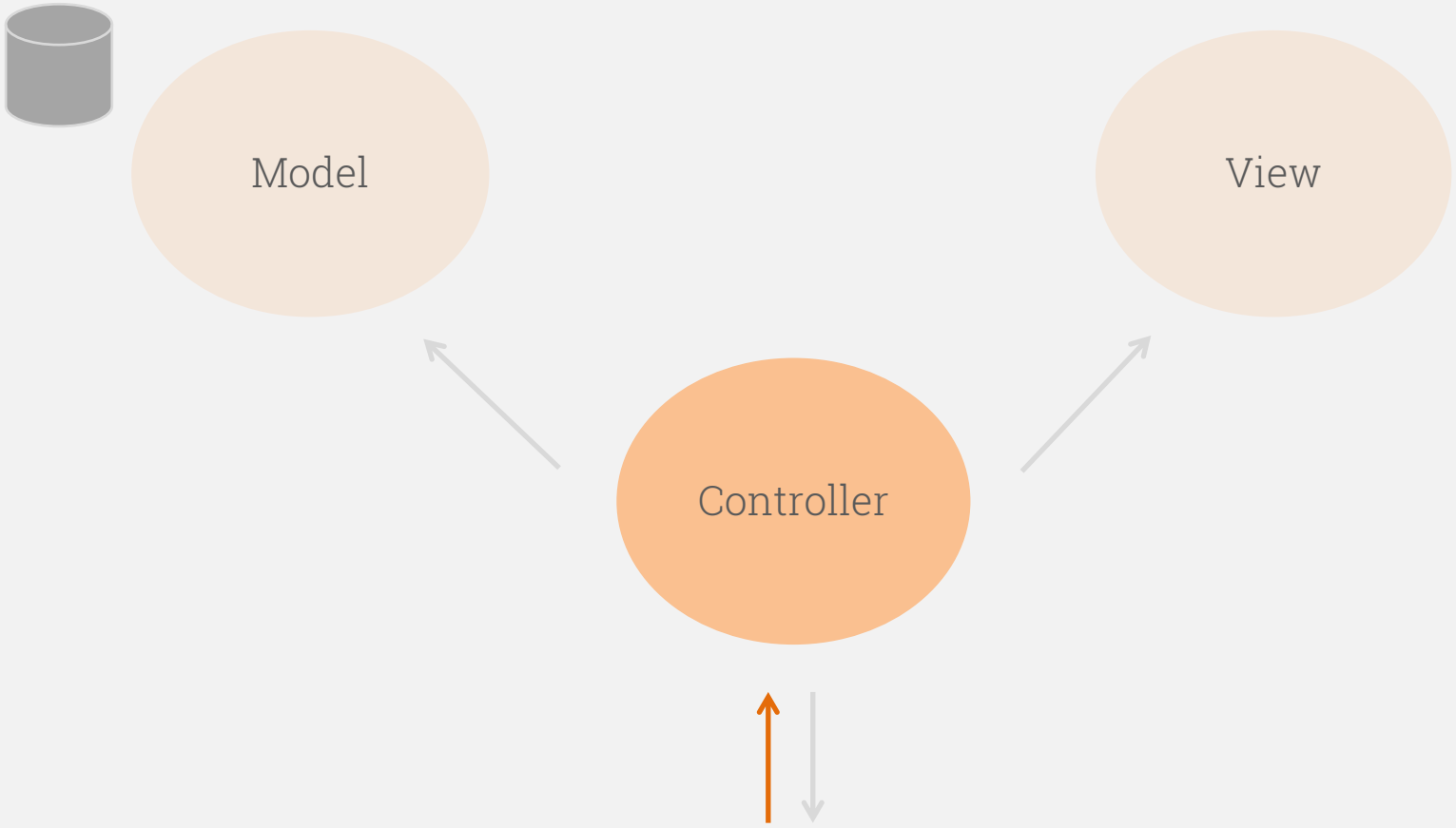
Controller translates requests to actions.

Example: PostsController handles requests related to posts

Model-View-Controller

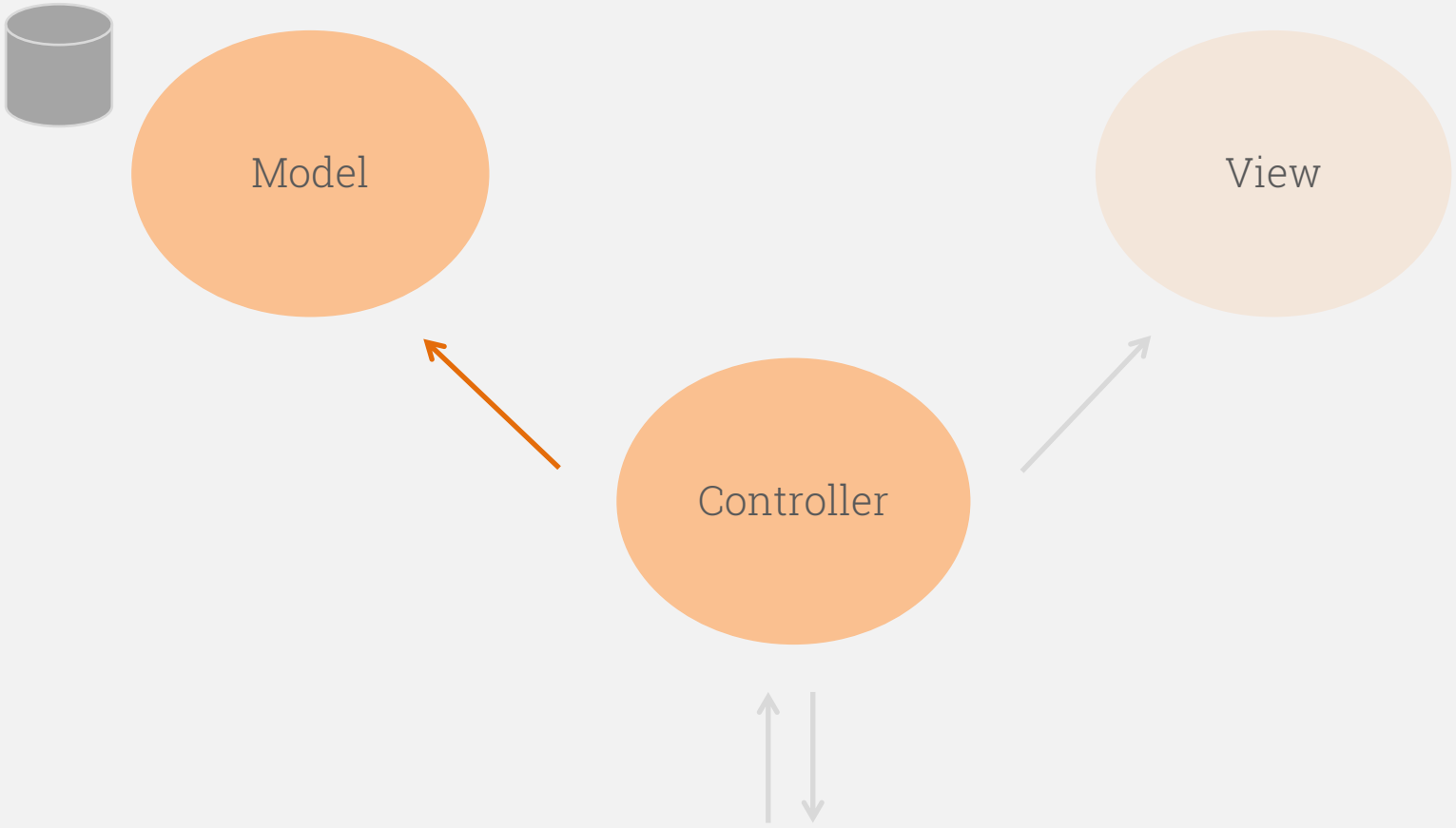
How do they work together?

Model-View-Controller



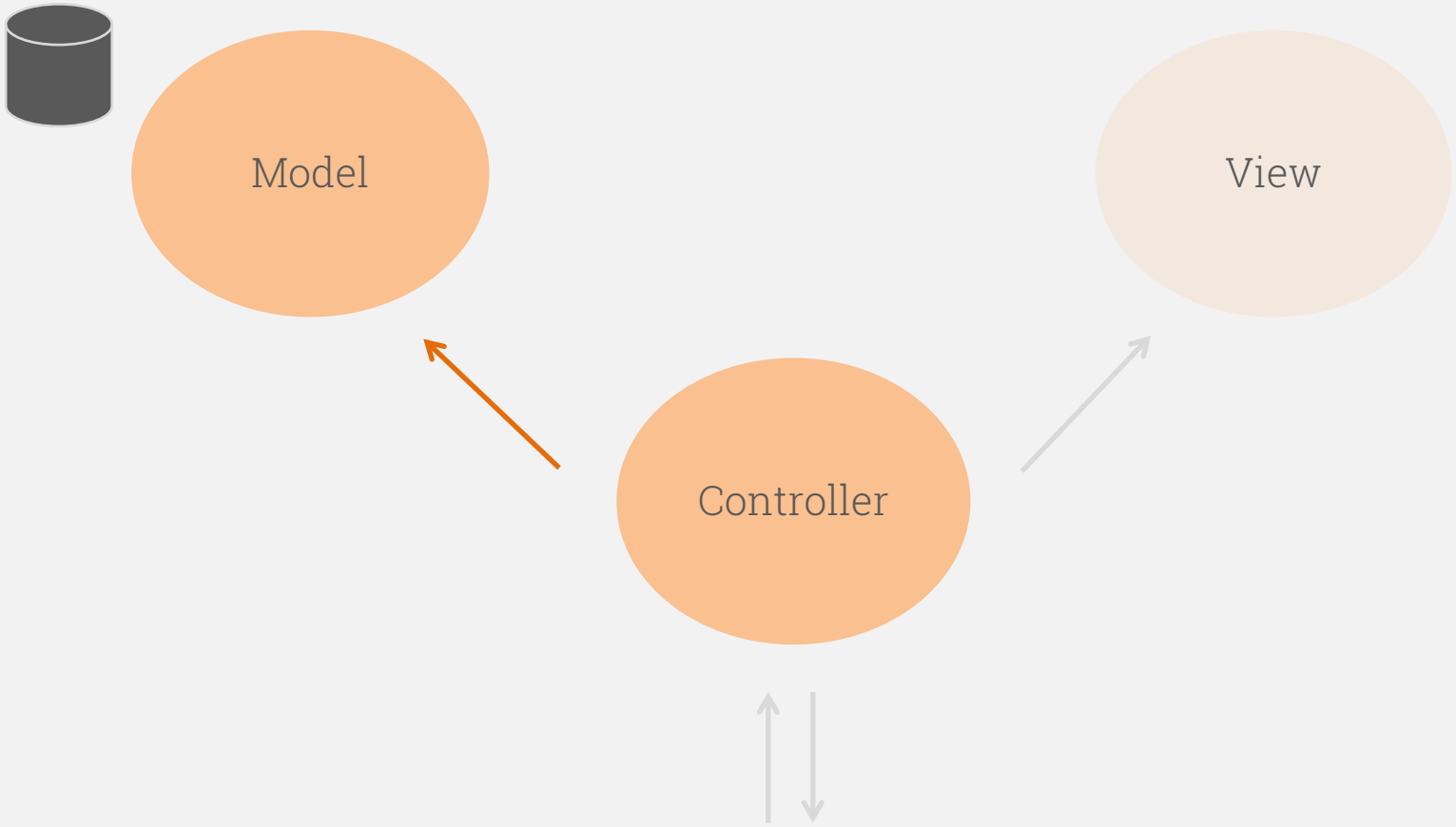
Controller is the entry point. Request is translated to action.

Model-View-Controller



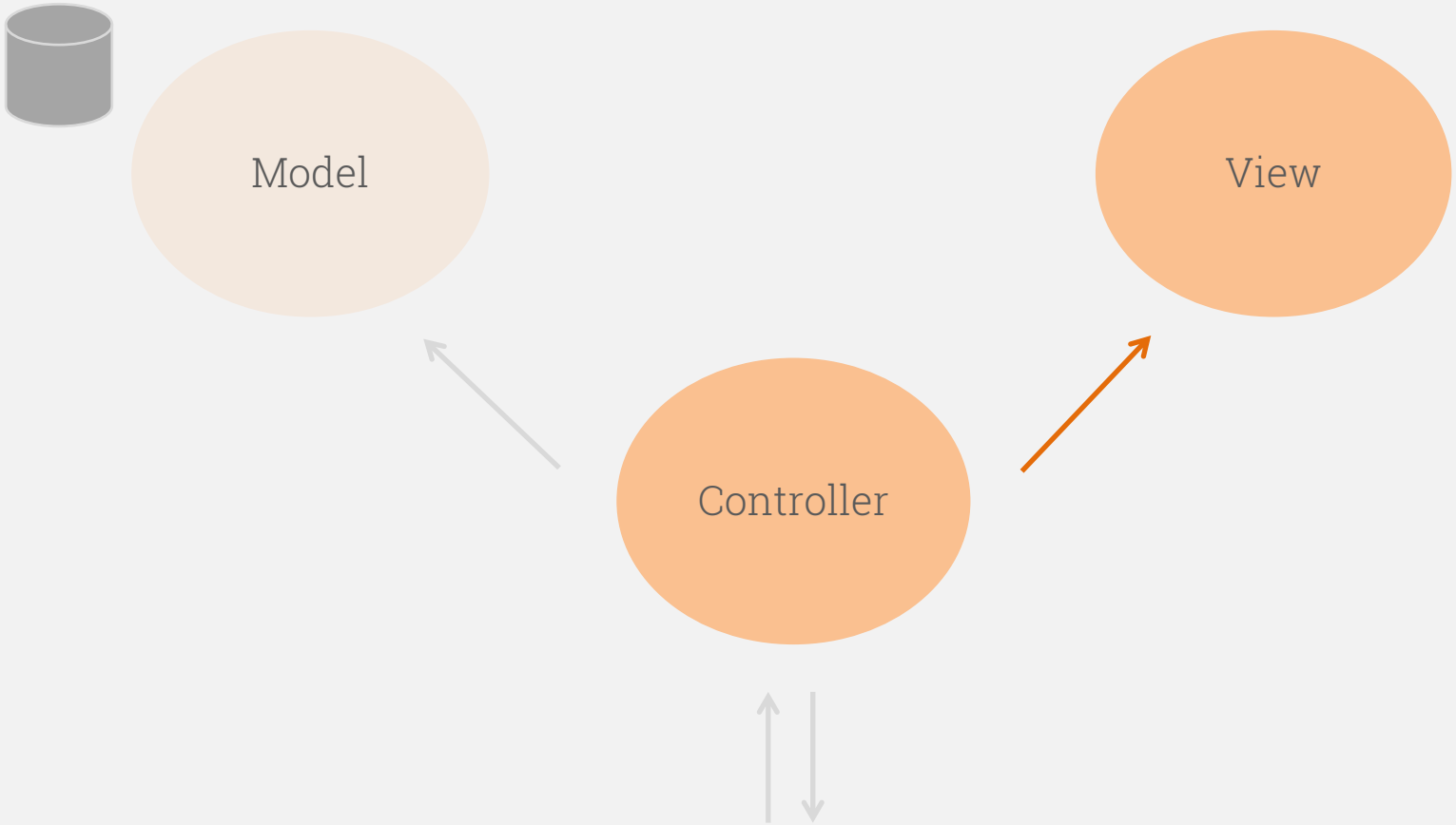
A **Controller** may call one or many **Models** to fetch data.

Model-View-Controller



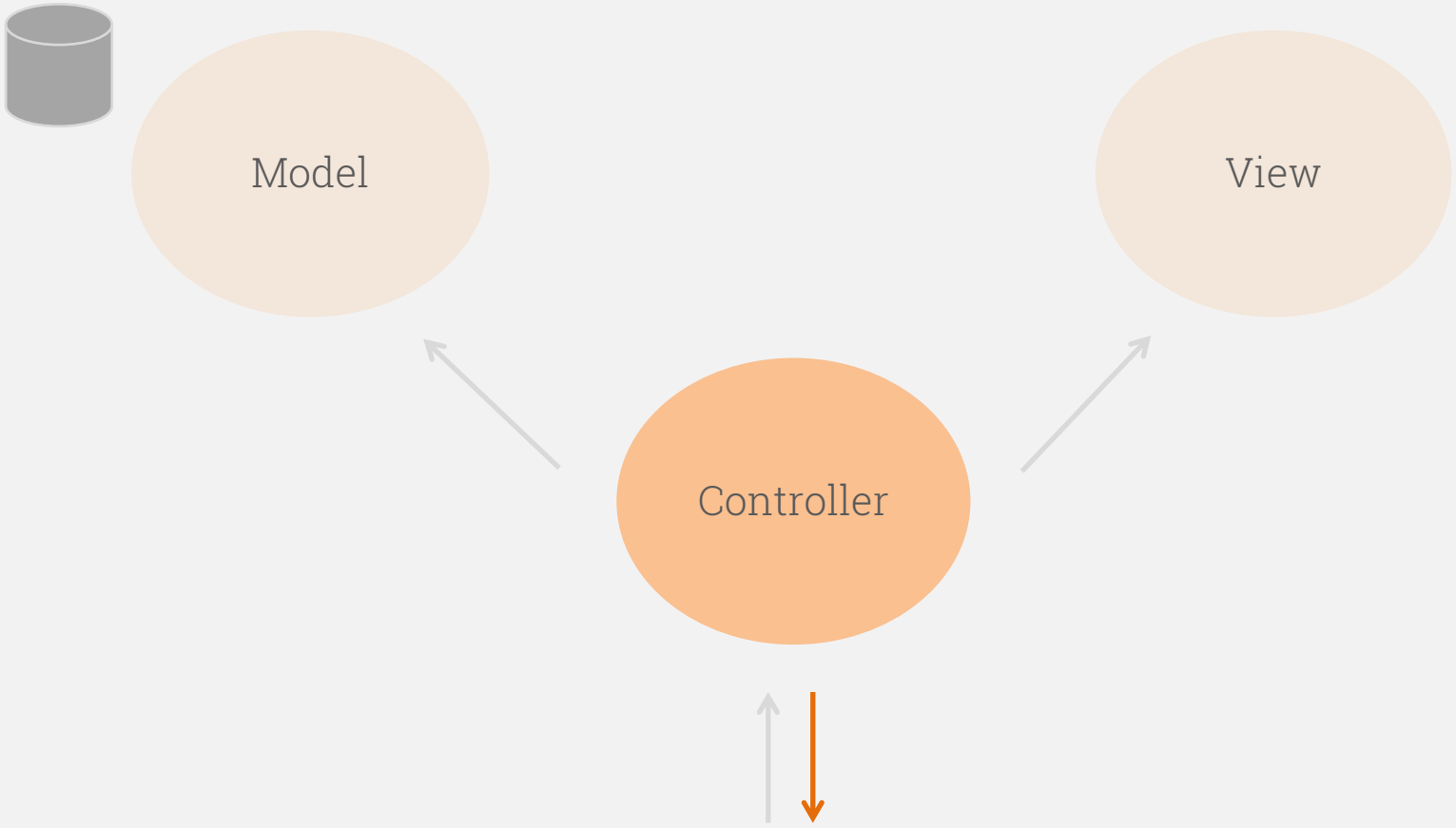
Model fetches data from database and returns it to **Controller**.

Model-View-Controller



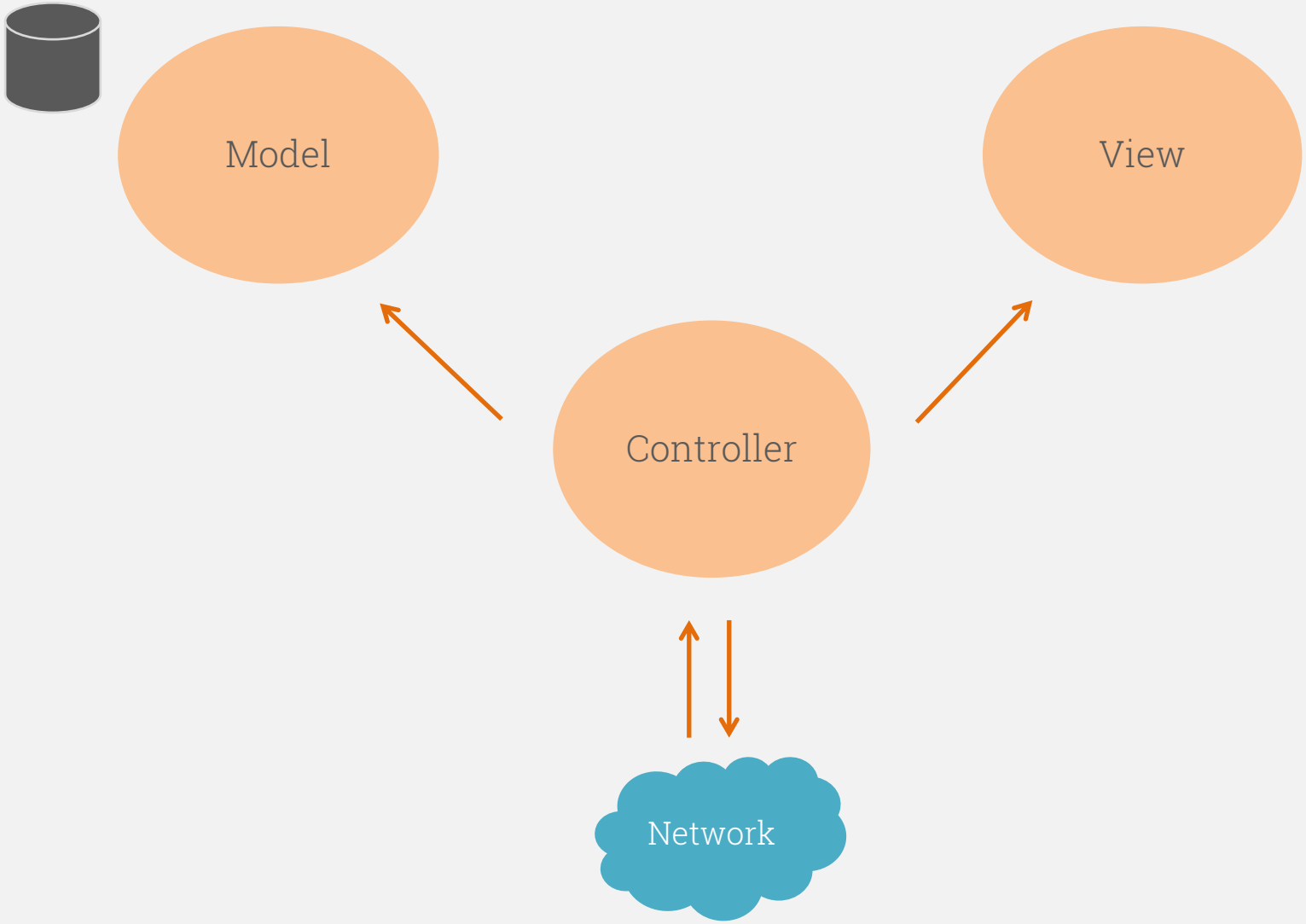
Controller passes the fetched data to a given **View**.

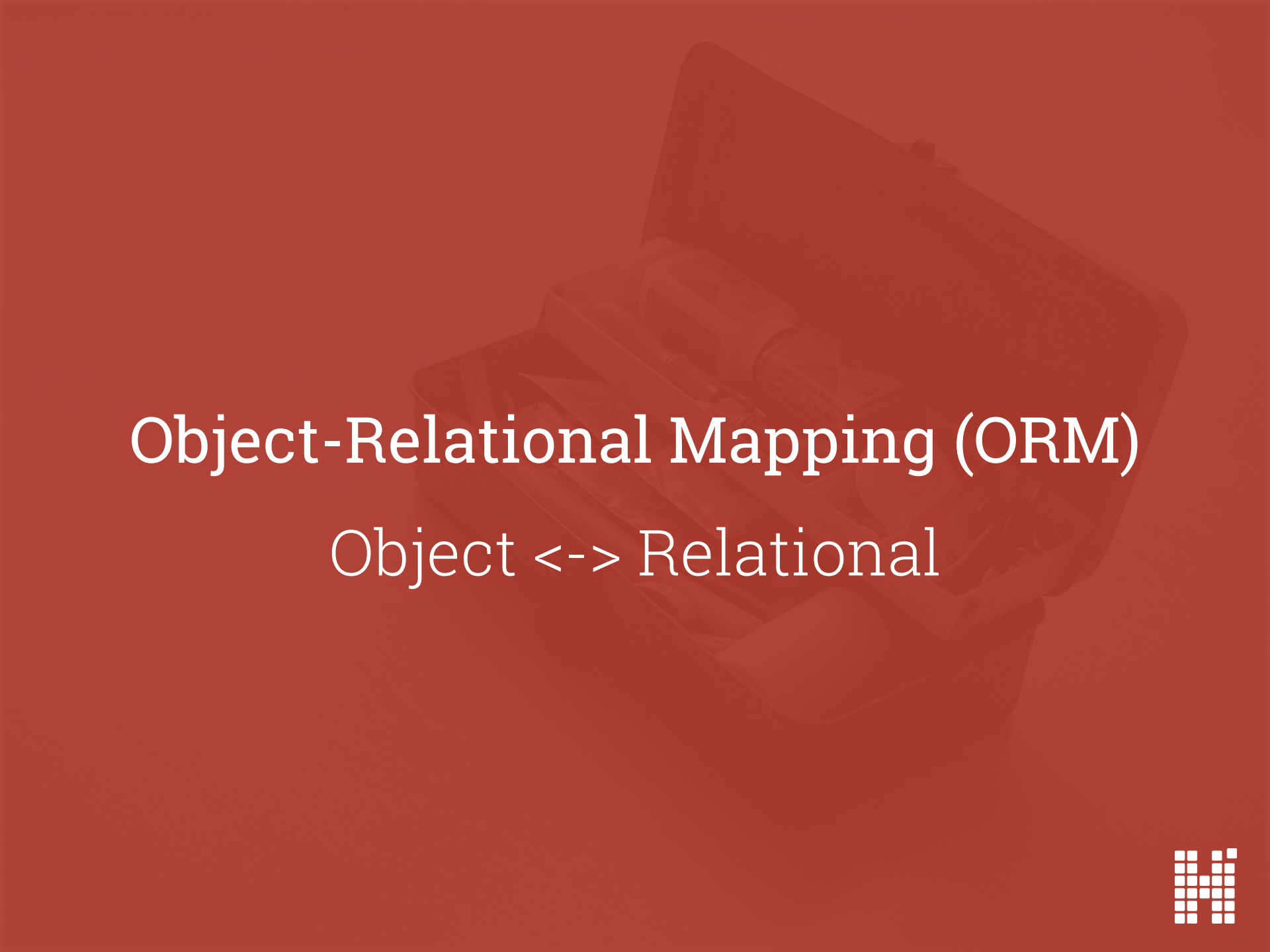
Model-View-Controller



Controller returns the **View** to the request's origin.

Model-View-Controller





Object-Relational Mapping (ORM)

Object <-> Relational

Object-Relational Mapping

PHP Class

```
class Tweet
{
    private $id;

    private $text;

    private $created_at;

}
```

Relational Table

tweets
id: *integer*
text: *varchar(120)*
created_at: *datetime*

Object's attributes are mapped to **columns of a relational table**.


Let me present you...


Laravel
A framework for web artisans

What is Laravel?

A PHP framework released in 2012

The most used PHP framework

Laravel hit the top in Dec 2013

The **Philosophy**

"**Laravel is a web application framework with expressive, elegant syntax**. We believe development must be an enjoyable, creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.

Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, we've attempted to combine the very best of what we have seen in other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Laravel is accessible, yet powerful, providing powerful tools needed for large, robust applications. A superb inversion of control container, expressive migration system, and tightly integrated unit testing support give you the tools you need to build any application with which you are tasked."

— Taylor Otwell *in Laravel.com (2014)*

The **Philosophy**

"Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable, creative experience to be truly fulfilling. Laravel **attempts to take the pain out of development by easing common tasks** used in the majority of web projects, **such as authentication, routing, sessions, and caching.**

Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, we've attempted to combine the very best of what we have seen in other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Laravel is accessible, yet powerful, providing powerful tools needed for large, robust applications. A superb inversion of control container, expressive migration system, and tightly integrated unit testing support give you the tools you need to build any application with which you are tasked."

— Taylor Otwell *in Laravel.com (2014)*

The **Philosophy**

"Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable, creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.

Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, **we've attempted to combine the very best of what we have seen in other web frameworks**, including frameworks implemented in other languages, **such as Ruby on Rails, ASP.NET MVC, and Sinatra**.

Laravel is accessible, yet powerful, providing powerful tools needed for large, robust applications. A superb inversion of control container, expressive migration system, and tightly integrated unit testing support give you the tools you need to build any application with which you are tasked."

— Taylor Otwell *in Laravel.com (2014)*

The **Philosophy**

"Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable, creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.

Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, we've attempted to combine the very best of what we have seen in other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Laravel is accessible, yet powerful, **providing powerful tools needed for large, robust applications.** A superb inversion of control container, expressive migration system, and tightly **integrated unit testing support** give you the tools you need to build any application with which you are tasked."

— Taylor Otwell *in Laravel.com (2014)*

Project folder structure

Knowing your workspace

The **Folder Structure**

app	Our application workspace: controllers, models, ...
bootstrap	Bootstrap files like classes auto-loading
config	Application configuration: authentication, database, ...
database	Database migrations and seeds
public	Public items, like *robots.txt* and *index.php*
resources	Views, template's assets and i18n files
routes	Application's routes: web, api, ...
storage	Storage folder for sessions, files, etc.
tests	Application tests: unit, integration, ...
.env.example	Environment parameters
artisan	Swiss army knife for multiple purposes
composer.json	Project dependencies (PHP packages)
gulpfile.js	Task-manager
package.json	Project dependencies (Javascript packages)
phpunit.xml	Configuration for testing environment
server.php	Local server for development only

The swiss army knife

Artisan CLI

The **Artisan CLI**

Artisan is the name of the command-line interface included with Laravel.
It provides helpful commands for your use while developing your application.

```
$ php artisan list                      # List available commands

$ php artisan help migrate              # Show help screen for “migrate”

$ php artisan migrate --env=testing     # Specify configuration environment

$ php artisan --version                 # Show version of Artisan
```

Examples from: http://laravel.com/docs/5.3/artisan

Controllers

Endpoints of the application

Basic Controller

You define everything

The Basic Controller

```
$ php artisan make:controller TweetsController
```

app/Http/Controllers/TweetsController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TweetsController extends Controller
{
    public function showHello()
    {
        return "Hello";
    }
}
```

RESTful Resource Controller

Controller represents resources

The RESTful Resource Controller

Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Table from: http://laravel.com/docs/5.3/controllers

The RESTful Resource Controller

```
$ php artisan make:controller TweetsController --resource
```

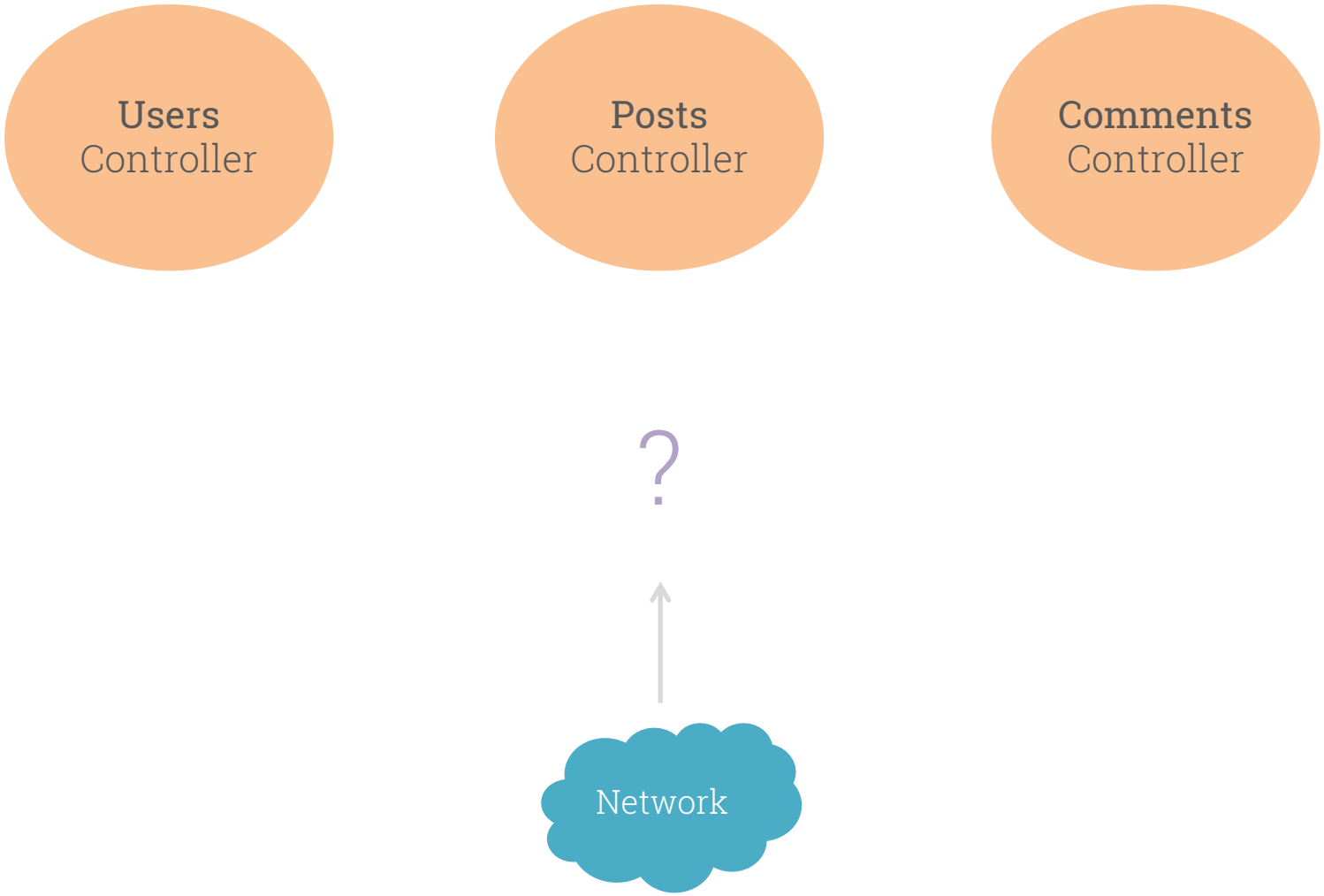
app/Http/Controllers/TweetsController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TweetsController extends Controller
{
    public function index() {}
    public function create() { }
    public function store(Request $request) { }
    public function show($id) { }
    public function edit($id) { }
    public function update(Request $request, $id) { }
    public function destroy($id) { }
}
```

Applications have several controllers

How to know which we want?

The **Router** component



Router Component

knows how to deal with it.

The **Router** component - Foundations

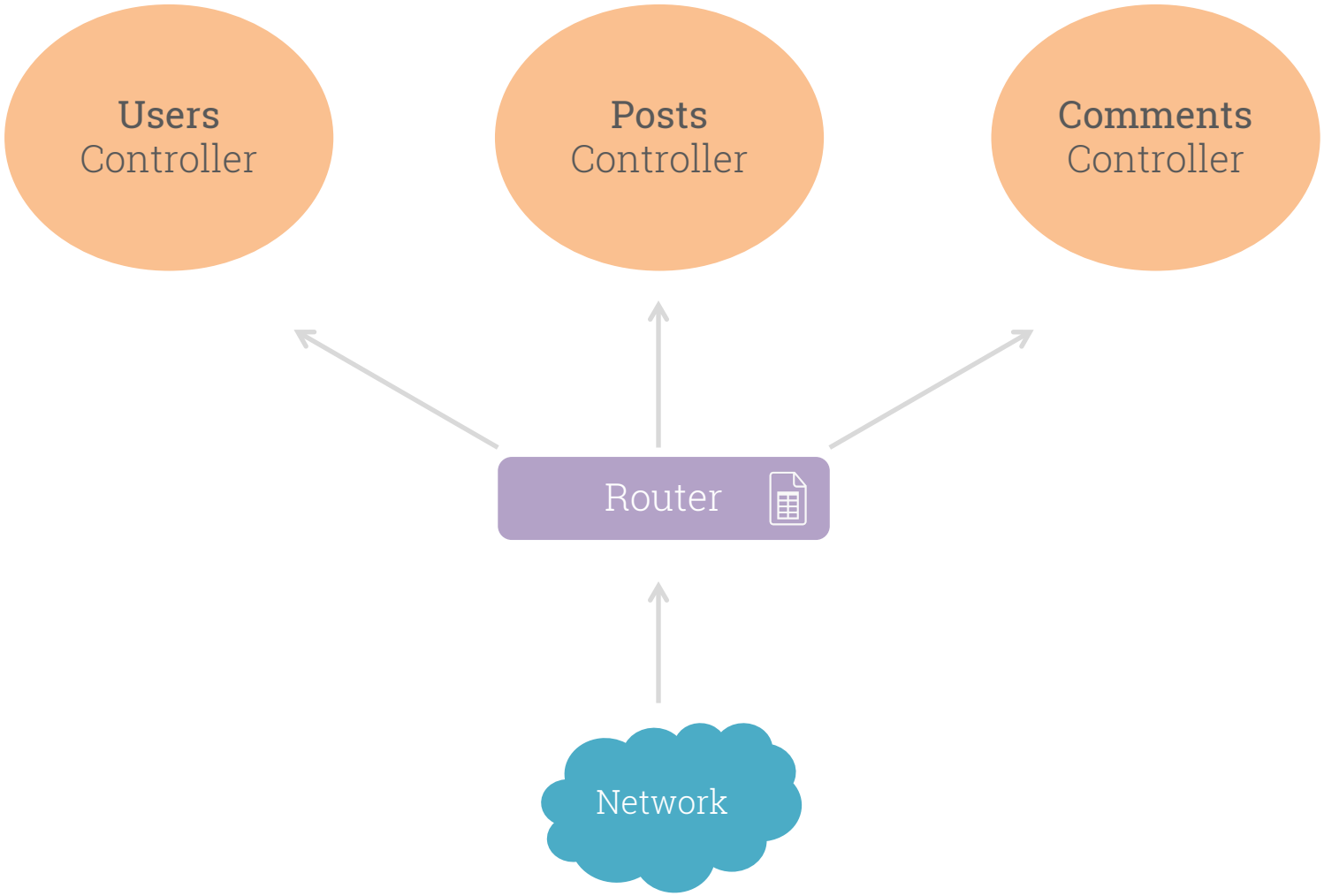
http://domain.dev/users/create

URL is composed of a **protocol**, **domain** and a **URI**

The **Router** component - Foundations

A **HTTP Request** contains a method: GET, POST, HEAD...

The **Router** component

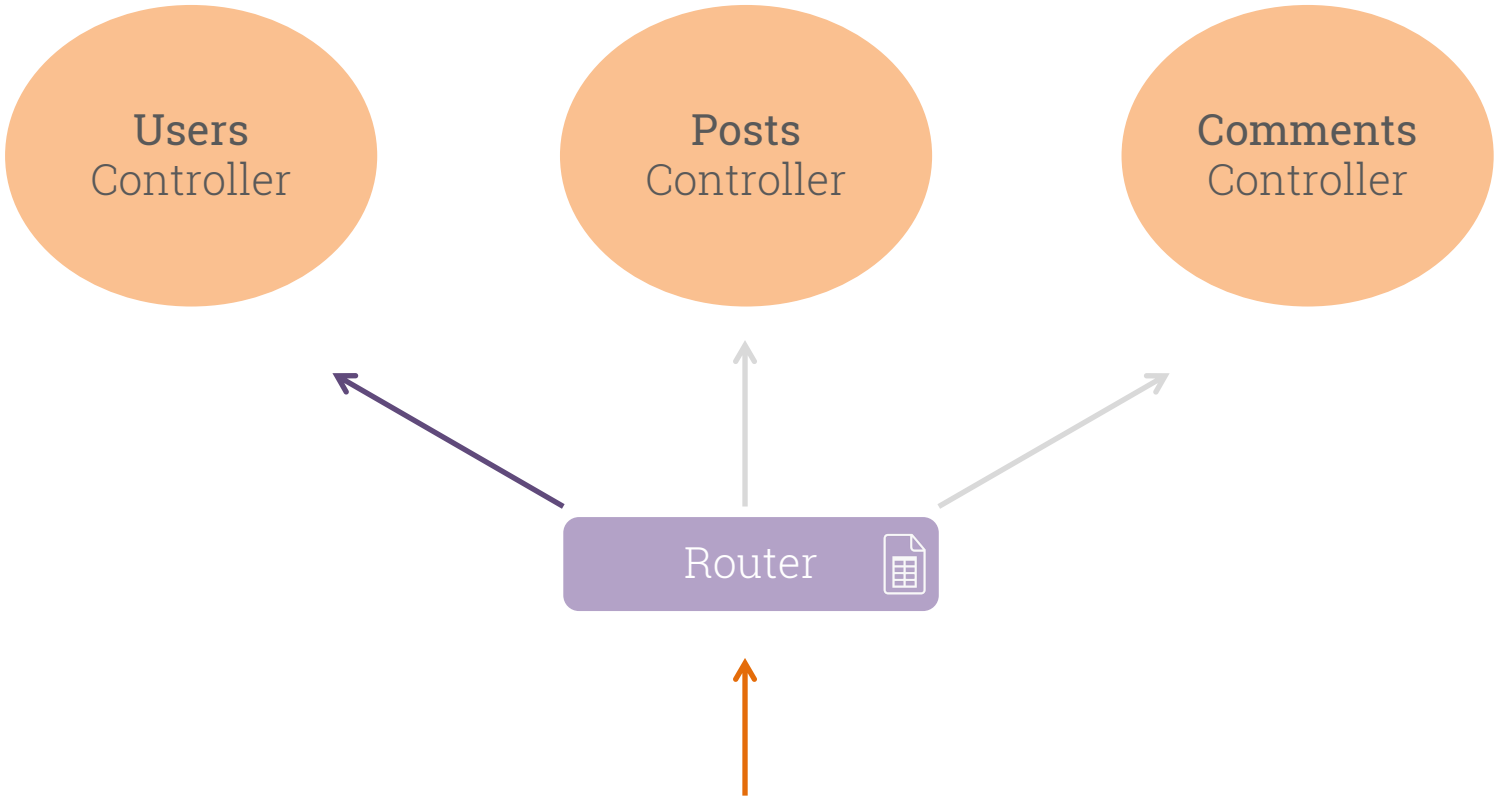


The **Router** component

http://domain.dev**/users**/create

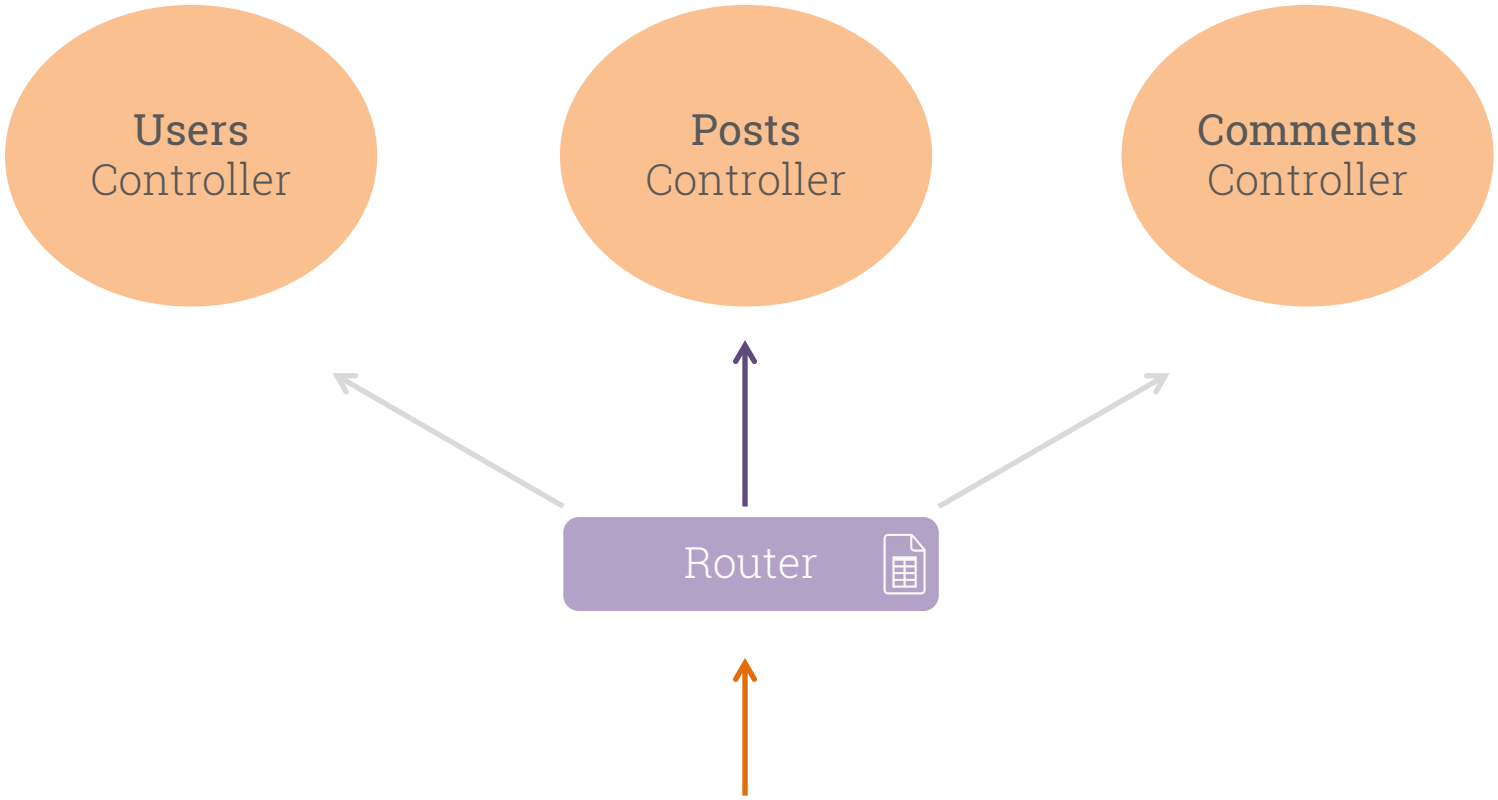
UsersController

The **Router** component



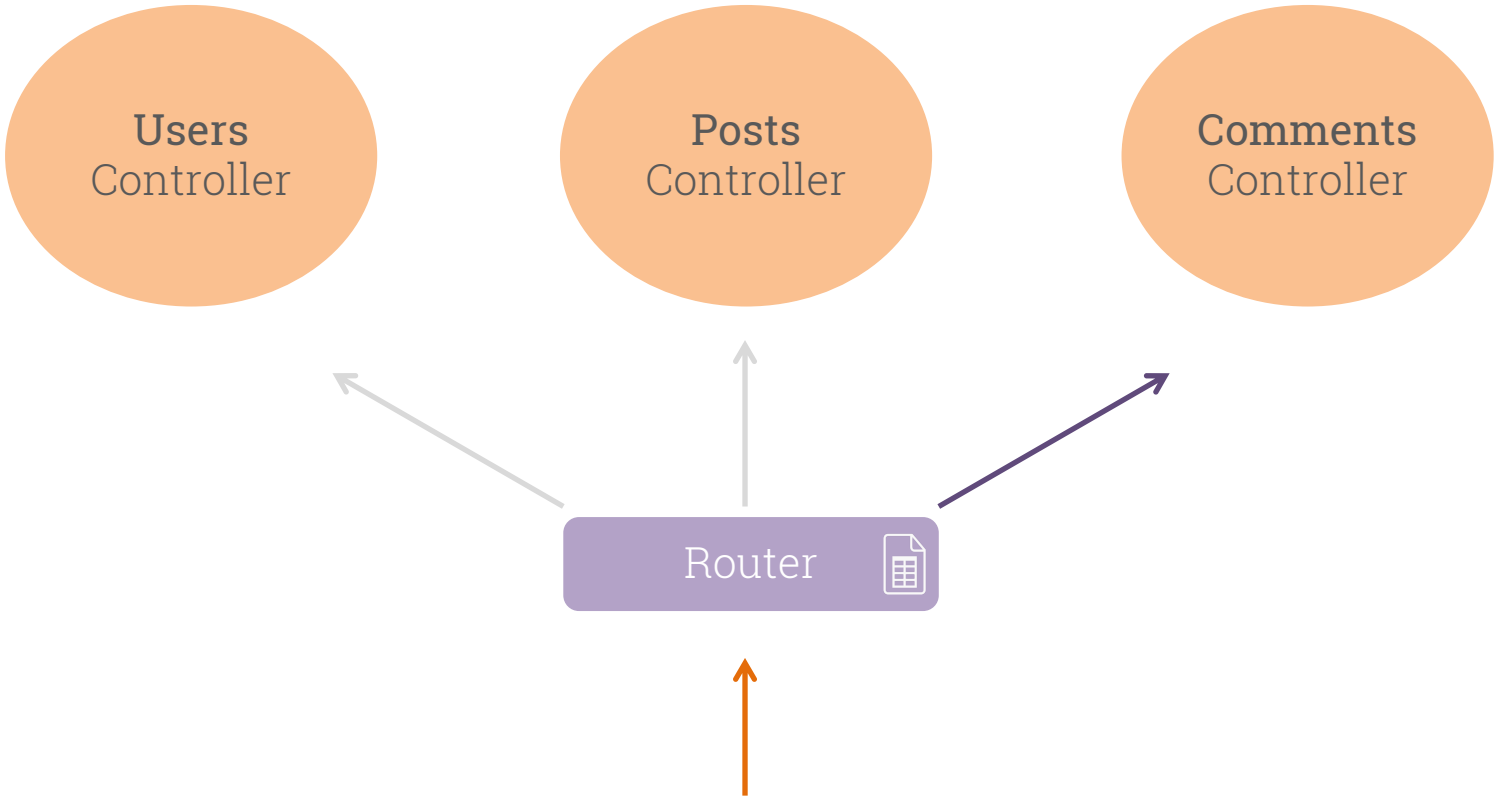
Request **/users** will be routed to **UsersController**

The **Router** component



Request **/posts/create** will be routed to **PostsController**

The **Router** component



Request **/comments/1** will be routed to **CommentsController**

The **Router** component

Method	URI	Name	Action
GET|HEAD	api/v1/events	events.index	App\Http\Controllers\Api\EventsController@index
POST	api/v1/events	events.store	App\Http\Controllers\Api\EventsController@store
GET|HEAD	api/v1/events/create	events.create	App\Http\Controllers\Api\EventsController@create
GET|HEAD	api/v1/events/{event}	events.show	App\Http\Controllers\Api\EventsController@show
PUT|PATCH	api/v1/events/{event}	events.update	App\Http\Controllers\Api\EventsController@update
DELETE	api/v1/events/{event}	events.destroy	App\Http\Controllers\Api\EventsController@destroy
GET|HEAD	api/v1/events/{event}/edit	events.edit	App\Http\Controllers\Api\EventsController@edit

Router redirects requests considering the **request's parameters**.

(It also considers domain)

The Basic Controller

We **must** add TweetsController to our web routes table.

routes/web.php

```php
<?php

/*
 * Now, we can access
 * GET http://domain.com/hello
 *
 * "/hello" will be routed to action "showHello"
 * of TweetsController controller.
 */

Route::get('/hello', 'TweetsController@showHello');
```

The **Router** component – Basic Controller

```
+----------+-------+------+-----------------------------------------------+
| Method   | URI   | Name | Action                                        |
+----------+-------+------+-----------------------------------------------+
| GET|HEAD | hello |      | App\Http\Controllers\TweetsController@showHello |
+----------+-------+------+-----------------------------------------------+
```

The RESTful Resource Controller

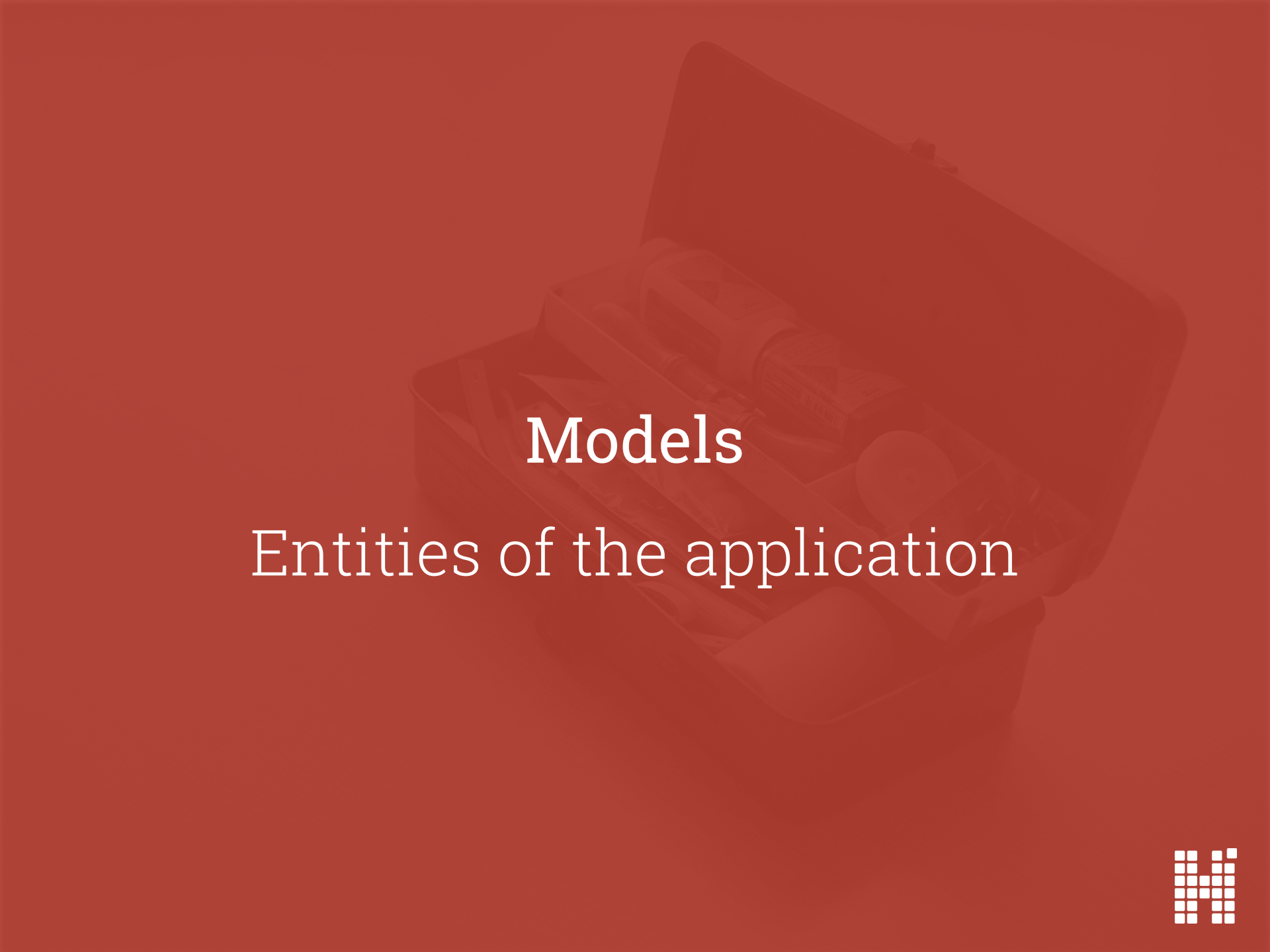
We **must** add TweetsController to our web routes table.

routes/web.php

```php
<?php

/*
 * Now, we can access
 * GET http://domain.com/hello
 *
 * "/hello" will be routed to action "showHello"
 * of TweetsController controller.
 */

Route::resource('tweets', 'TweetsController');
```

The **Router** component – Resource Controller

Method	URI	Name	Action
GET|HEAD	tweets	tweets.index	App\Http\Controllers\TweetsController@index
POST	tweets	tweets.store	App\Http\Controllers\TweetsController@store
GET|HEAD	tweets/create	tweets.create	App\Http\Controllers\TweetsController@create
GET|HEAD	tweets/{tweet}	tweets.show	App\Http\Controllers\TweetsController@show
PUT|PATCH	tweets/{tweet}	tweets.update	App\Http\Controllers\TweetsController@update
DELETE	tweets/{tweet}	tweets.destroy	App\Http\Controllers\TweetsController@destroy
GET|HEAD	tweets/{tweet}/edit	tweets.edit	App\Http\Controllers\TweetsController@edit



Models

Entities of the application

The Model

```
$ php artisan make:model Tweet
```

app/Tweet.php

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Tweet extends Model
{
    //
}
```

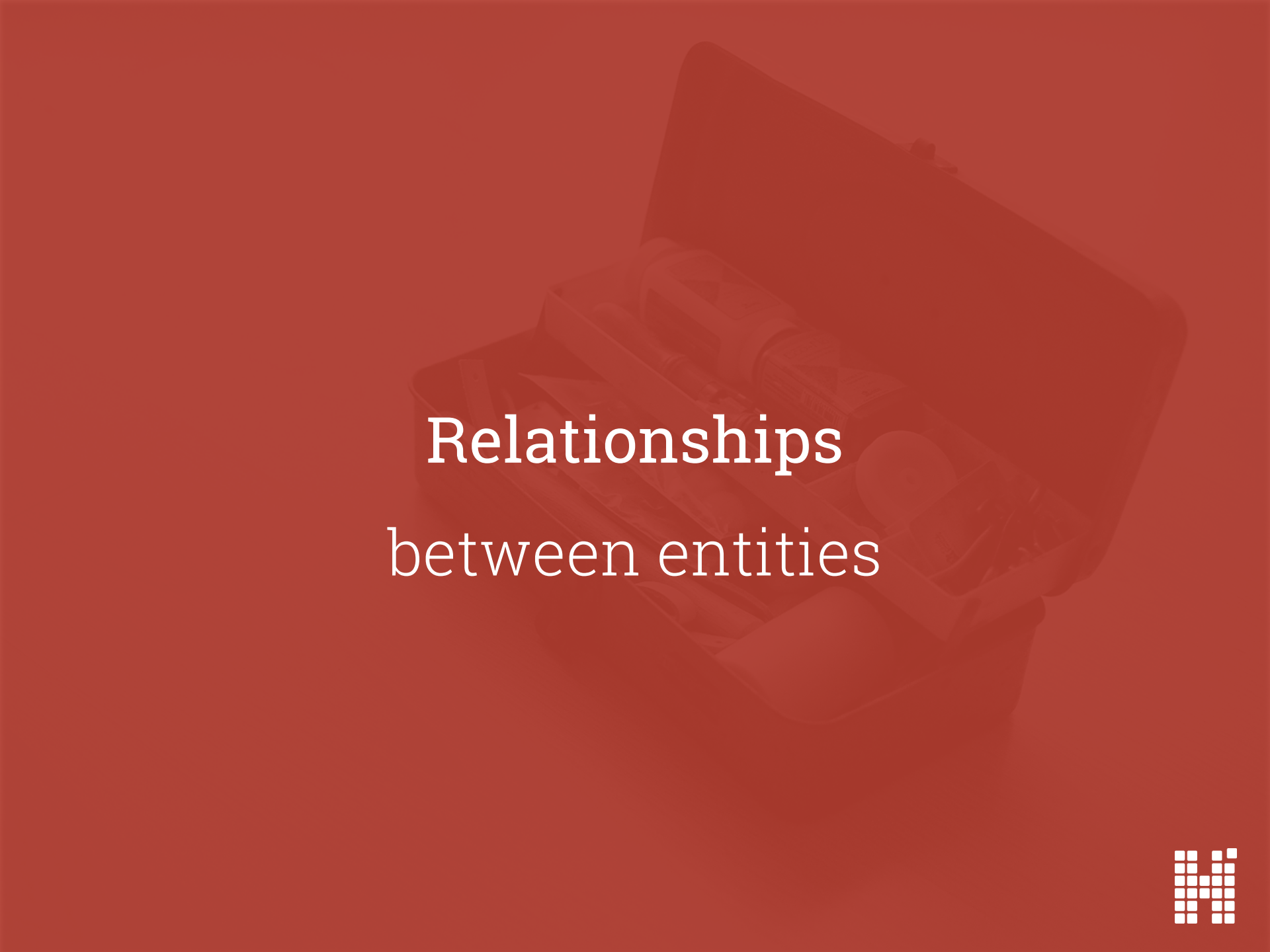
The **Model**

Tweet provides out of the box methods of **Eloquent ORM**.

app/Http/Controllers/TweetsController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Tweet; // Import the Tweet model.

class TweetsController extends Controller
{
    public function index()
    {
        // Return all tweets in database.
        return Tweet::all();
    }
}
```

Take a look to other methods: https://laravel.com/api/5.3/Illuminate/Database/Eloquent/Model.html



Relationships

between entities

The One-One Relationship

A one-to-one relationship is a **very basic relation**.

A User belongs to **one IdentityCard.**
An IdentityCard belongs to **one User**.

IdentityCards table contains a foreign key column **user_id.**
Users table contains a foreign key column **identity_card_id.**

Description from: http://laravel.com/docs/5.3/eloquent#relationships

The One-One Relationship

We specify that **one Tweet belongs to one User**

app/Tweet.php

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Tweet extends Model
{
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

The One-Many Relationship

A one-to-many relationship is how we say that:

A Tweet belongs to **one User.**
A User has **many Tweets**.

Tweets table contains a foreign key column **user_id.**

Description from: http://laravel.com/docs/5.3/eloquent#relationships

The One-Many Relationship

Specify that **one User has many Tweets**

app/User.php

```php
<?php

namespace App;

class User extends Model
{
    public function tweets()
    {
        return $this->hasMany(Tweet::class);
    }
}
```

The Many-Many Relationship

A many-to-many relationship is how we say that:

A User has **many Jobs (or belongs to many).**
A Job has **many Users (or belongs to many**.

A pivot table *job_user* that relates both models must exist with the following foreign keys: **job_id**, **user_id.**

Description from: http://laravel.com/docs/5.3/eloquent#relationships

The Many-Many Relationship

Specify that **one User belongs to many Jobs**

app/User.php

```php
<?php

namespace App;

class User extends Model
{
    public function jobs()
    {
        return $this->belongsToMany(Job::class);
    }
}
```

The Many-Many Relationship

Specify that **one Job belongs to many Users**

app/Job.php

```php
<?php

namespace App;

class Job extends Model
{
    public function users()
    {
        return $this->belongsToMany(User::class);
    }
}
```

What about Views?

You may want to show data.

The **Blade Templating**

Blade is a powerful **templating engine** provided with Laravel.

Blade is **driven by template inheritance and sections**.

All Blade files **should end with the .blade.php extension**.

Description from: http://laravel.com/docs/5.3/templates#blade-templating

The **View** with **Blade Templating**

With **Blade Templating**, create a master layout.

resources/views/layouts/master.blade.php

```
<html>
    <head>
        ...
    </head>
    <body>
        @section('header')
            This is the header.
        @stop

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

The **View** with **Blade Templating**

Controller passes data to view **resources/views/tweets/index.blade.php**

app/Http/Controllers/TweetsController.php

```php
<?php

...
use App\Tweet;

class TweetsController extends Controller
{
    public function index()
    {
        $tweets = Tweet::all();
        // Pass data to view "tweets/index.blade.php" and return
        return view('tweets.index', ['tweets' => $tweets]);
    }
}
```

More about passing data to views: https://laravel.com/docs/5.3/views

The **View** with **Blade Templating**

View extends sections of master layout and override content

resources/views/tweets/index.blade.php

```
@extends('layouts.master')

@section('header')
    <p>This is appended to the master header.</p>
@endsection

@section('content')
    <p>This is my body content, where will be shown my tweets.</p>

    @foreach ($tweets as $tweet)
        <p>This is tweet "{{ $tweet->id }}"</p>
    @endforeach
@endsection
```

Take a look to other Blade Templating features: https://laravel.com/docs/5.3/blade

Database migrations

Version control for databases

The Database Migrations

Migrations are a type of version control for your database.

Allow to **modify the database schema**.

Migrations are typically paired with the Schema Builder.

Note: We need to create a database before calling migrations.

Description from: http://laravel.com/docs/5.3/migrations

The Schema Builder

Provides a **database agnostic way of manipulating tables**.

All of the databases supported by Laravel, with an **unified API**.

More about Schema Builder: http://laravel.com/docs/5.3/schema

The Database Migration

```
$ php artisan make:migration create_tweets_table
```

database/migrations/xxx_create_tweets_table.php

```php
class CreateTweetsTable extends Migration
{
    public function up()
    {
        Schema::create('tweets', function(Blueprint $table)
        {
            $table->increments('id');
            $table->integer('user_id')->unsigned(); // Owner of tweet
            $table->string('text');

            $table->foreign('user_id')->references('id')->on('users');
        });
    }
    public function down()
    {
        Schema::drop('tweets');
    }
}
```

The Database Migration

Migrations can be rolled back, too.

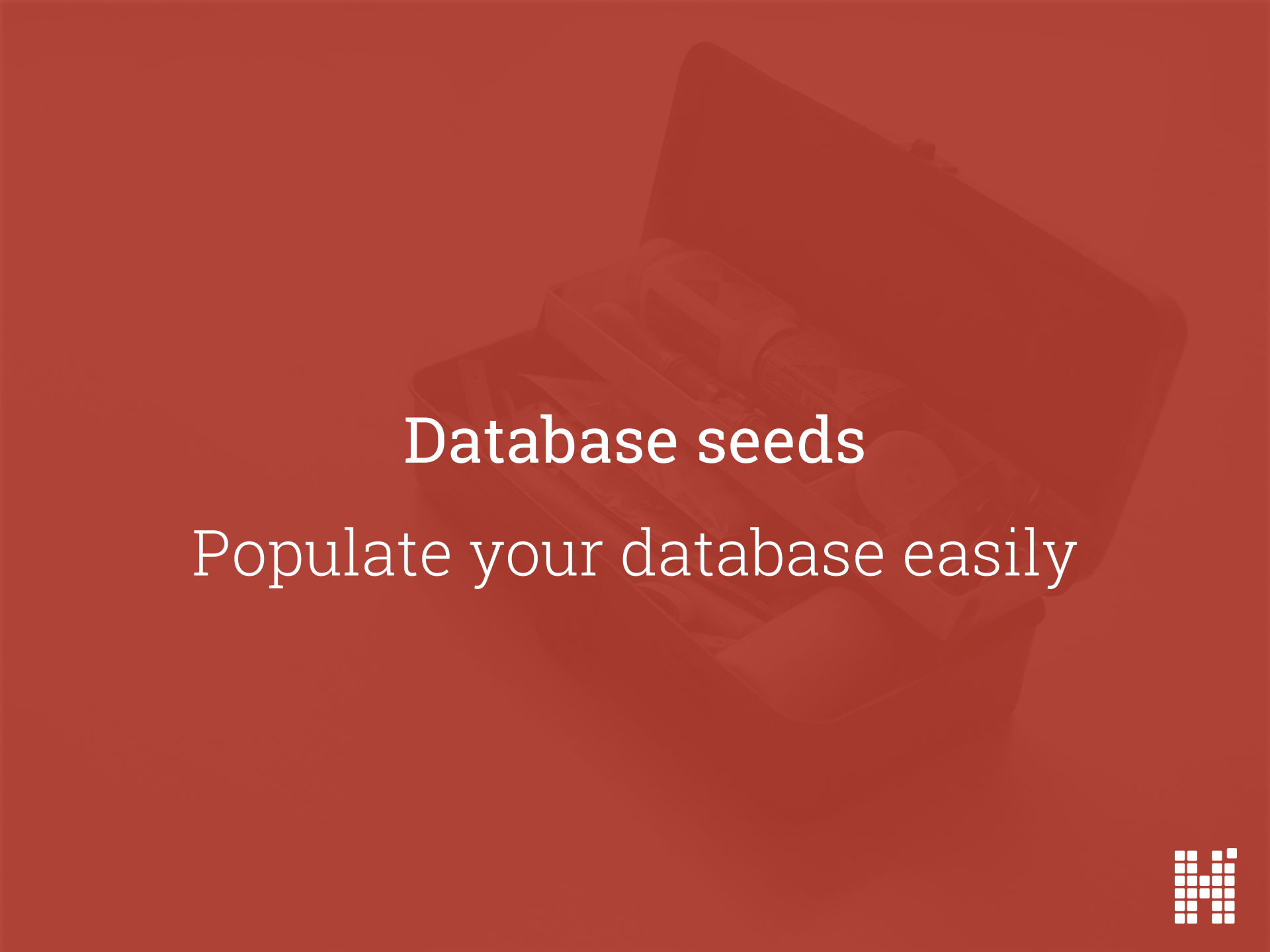
It's like a version control for your database. Reset everything is possible, too.

```
$ php artisan migrate:install        # Create "migrations" table

$ php artisan migrate                # Migrate a database to next version

$ php artisan migrate:rollback       # Rollback database version

$ php artisan migrate:status         # Show current status of database
```

Examples from: Artisan CLI

Database seeds

Populate your database easily



The Database Seeder

Laravel also includes a **simple way to seed your database**.

All seed classes are stored in **database/seeds**.

They are useful for testing or first time deployment, i.e, to create administrator accounts.

Use **DatabaseSeeder class**, to run other seed classes.

Description from: http://laravel.com/docs/5.3/migrations#database-seeding

The Database Seed

Create a **Seeder** class for each persisted entity.

database/seeds/UsersTableSeeder.php

```php
<?php

use Illuminate\Database\Seeder;
use App\User;

class UsersTableSeeder extends Seeder
{
    public function run()
    {
        User::create([
            'name' => 'First User',
            'email' => 'speaker@enei.pt',
            'org' => 'Hackathonners',
        ]);
    }
}
```

The Database Seed

Update **DatabaseSeeder**. You can choose the order of seeding.

database/seeds/DatabaseSeeder.php

```php
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(UsersTableSeeder::class);
    }
}
```

The Database Seed

Database may be seeded or reset

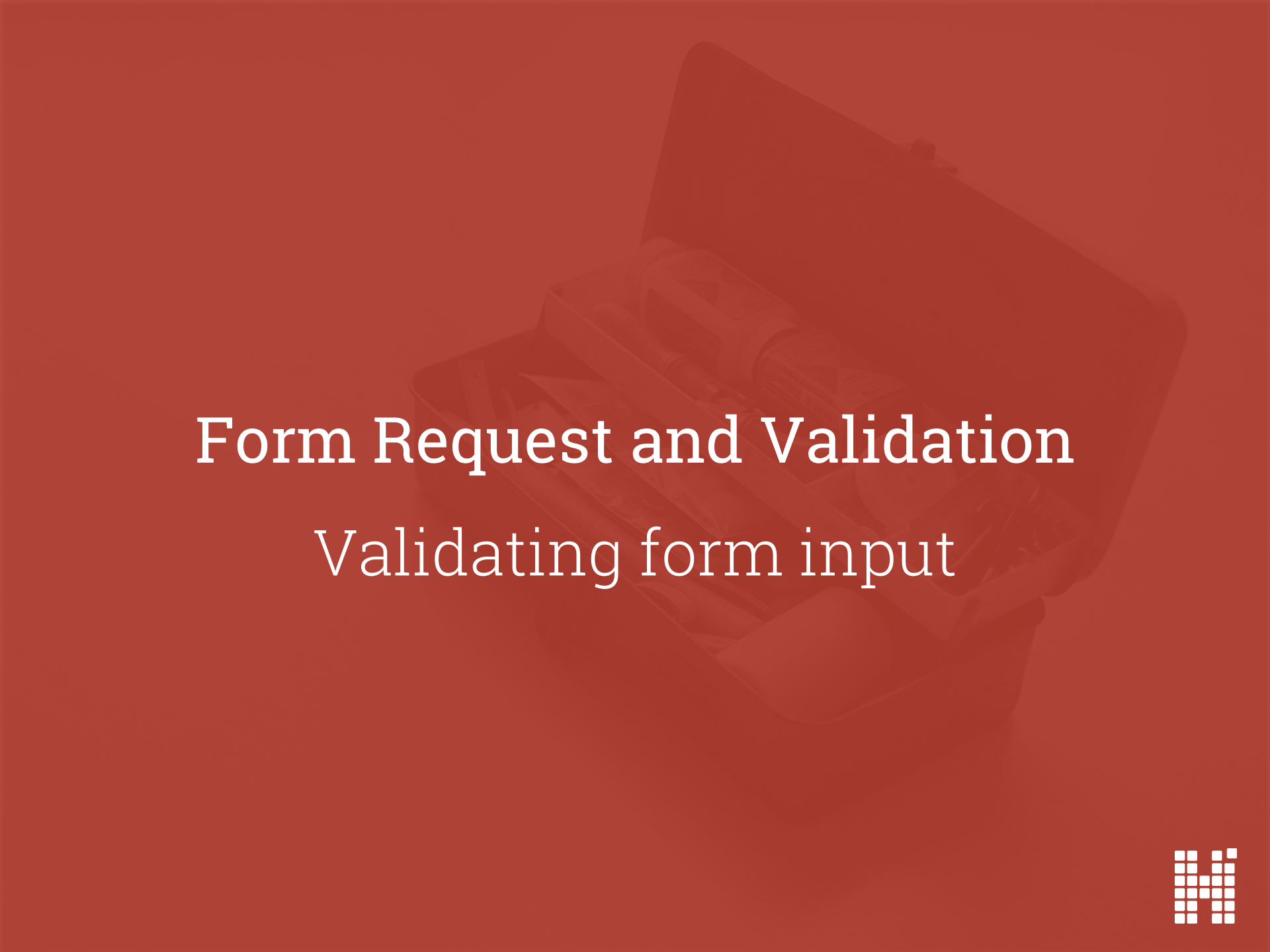
Use seeders when you're testing you app or need administrators in application deployment.

```
$ php artisan db:seed                    # Seed database
```

Examples from: Artisan CLI

Form Request and Validation

Validating form input



Validating Form Requests

```
$ php artisan make:request CreateTweetRequest
```

app/Requests/CreateTweetRequest.php

```php
<?php

namespace App\Http\Requests\Registration;

use App\Http\Requests\Request;

class CreateTweetRequest extends Request
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'text' => 'required|min:1|max:120'
        ];
    }
}
```

Validating Form Requests

When method is called, then **validation succeeded**

app/Controllers/TweetsController.php

```php
<?php

namespace App\Http\Controllers;

use App\Requests\CreateTweetRequest;
use App\Tweet;

class TweetsController extends Controller
{
    public function store(CreateTweetRequest $request)
    {
        // Laravel injects the dependency (object) in this method
        // Form is valid when this method is called
        // Save tweet to database.
        Tweet::create([
            'text' => $request->input('text')
        ]);
    }
}
```

Community

Pointers to learn more

The **Community**

Laravel Official Website
http://laravel.com

Laravel IO Forum
http://laravel.io

Laravel Screencasts
http://laracasts.com

Laravel Podcasts
http://podbay.fm/show/653204183

Questions?

That’s all folks.

contact@francisconeves.com

@fntneves

{Github, Twitter, LinkedIn}