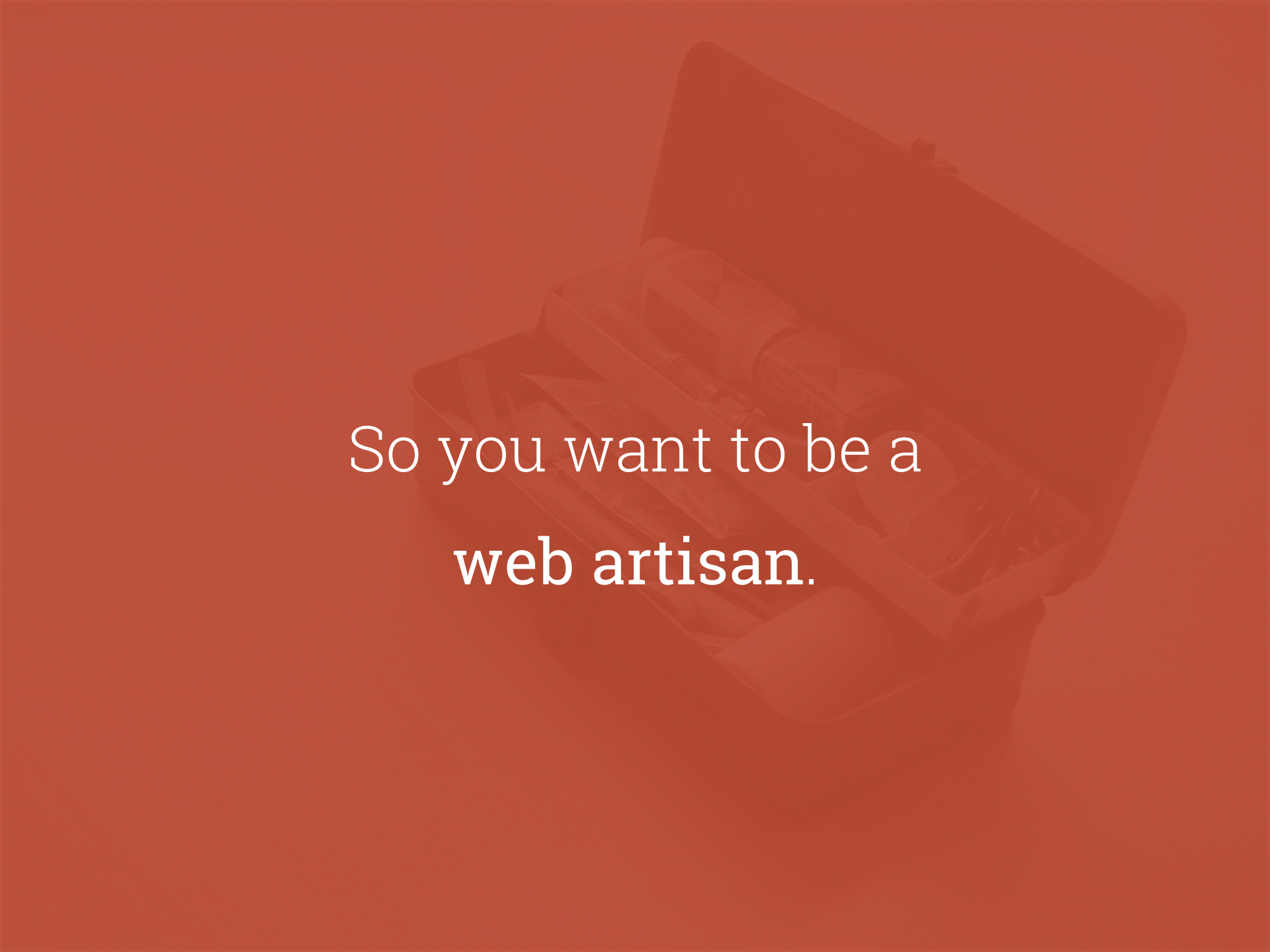
THE PHP FRAMEWORK

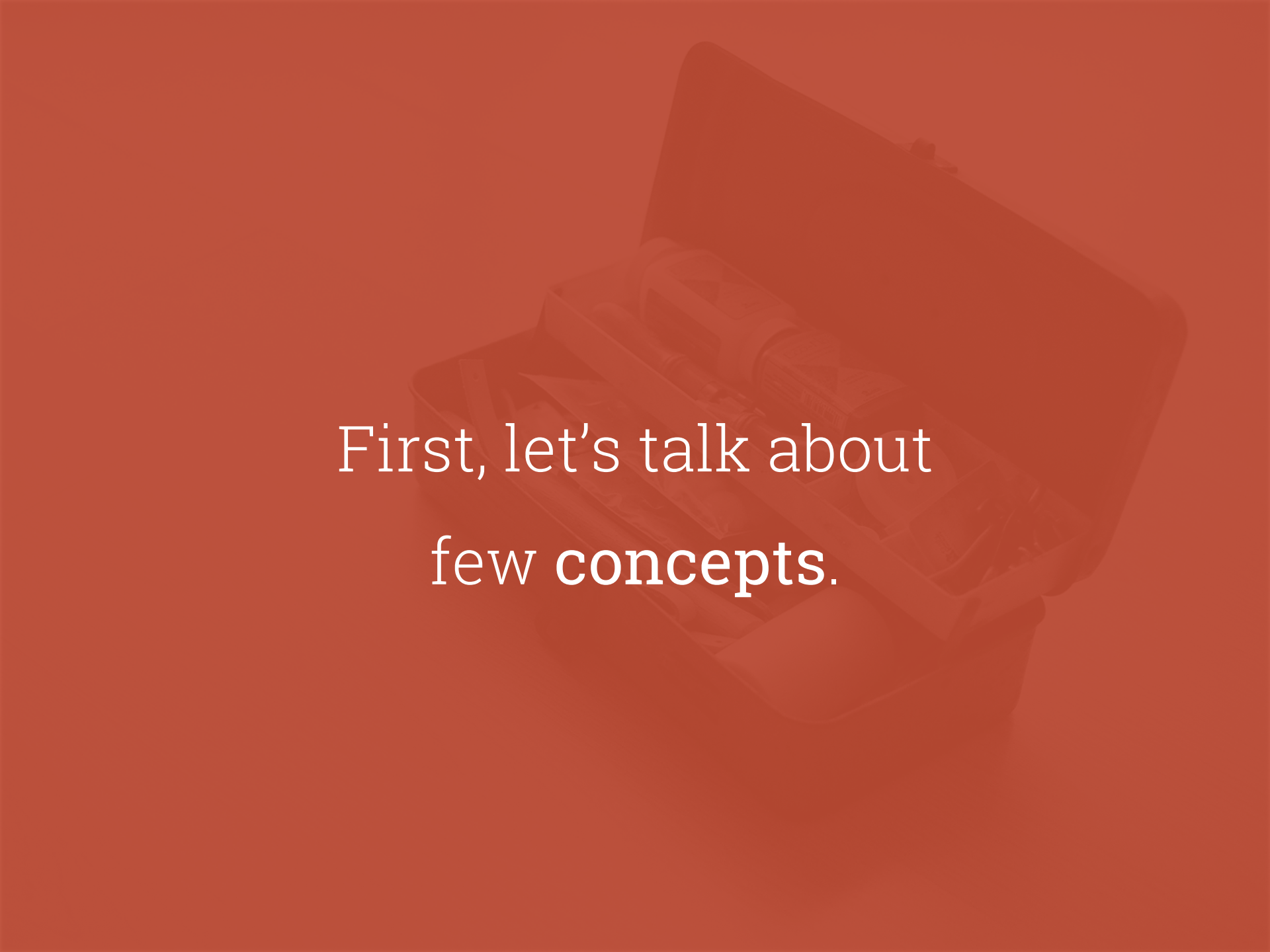
LARAVEL 5.0

WORKSHOP

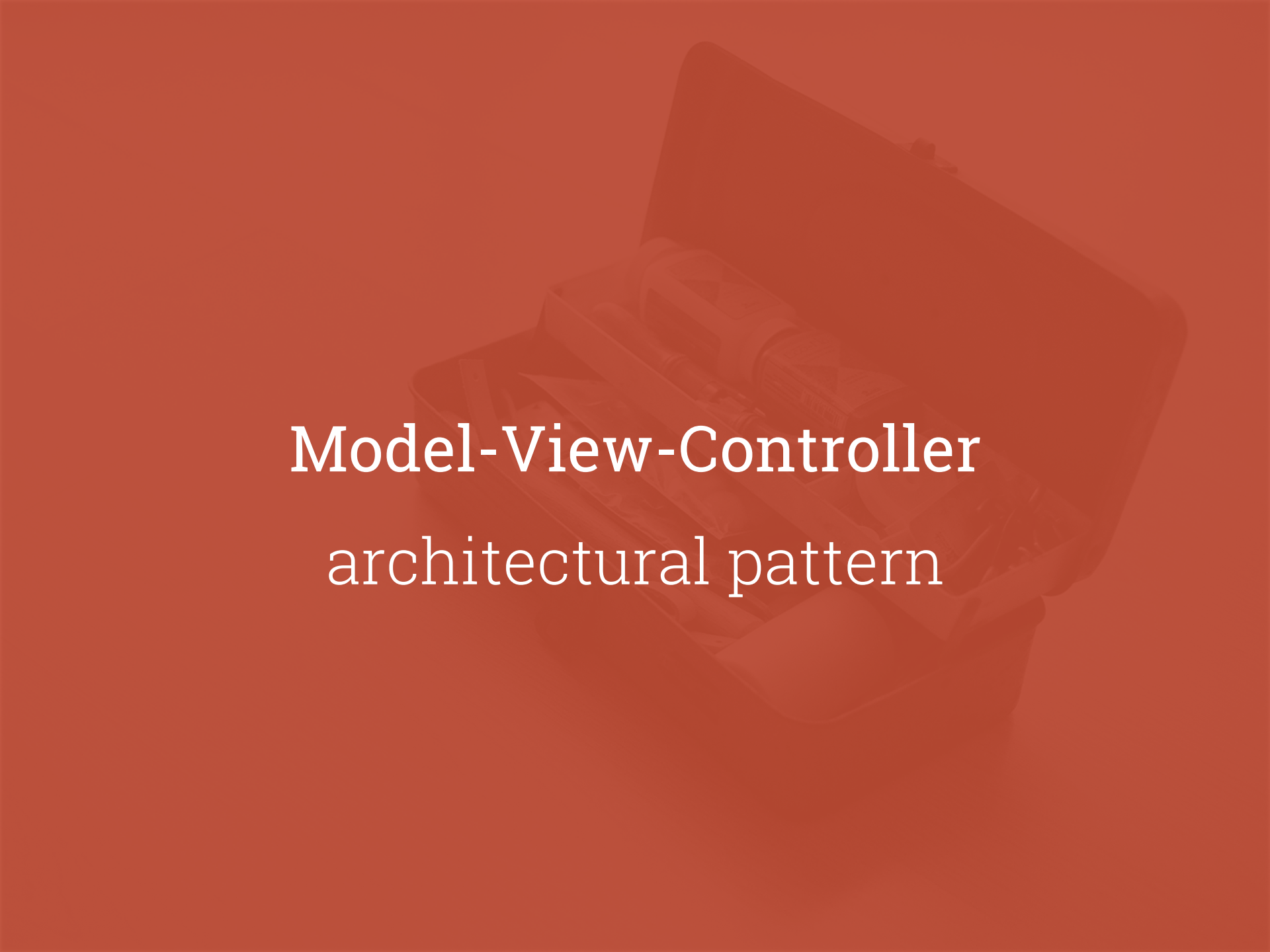

@fntneves


So you want to be a
web artisan.

Let the **show begin**.



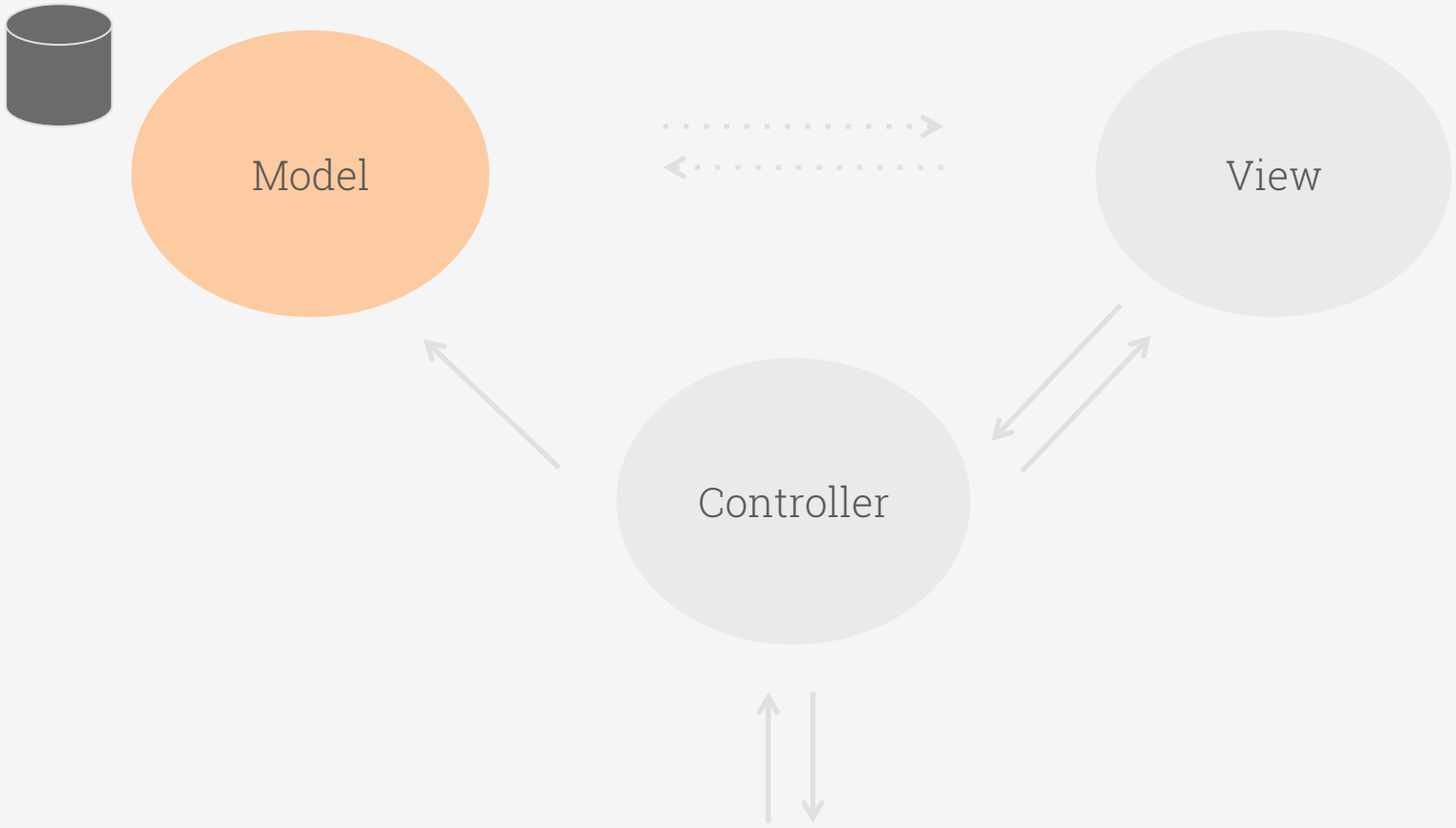
First, let’s talk about few **concepts**.



Model-View-Controller

architectural pattern

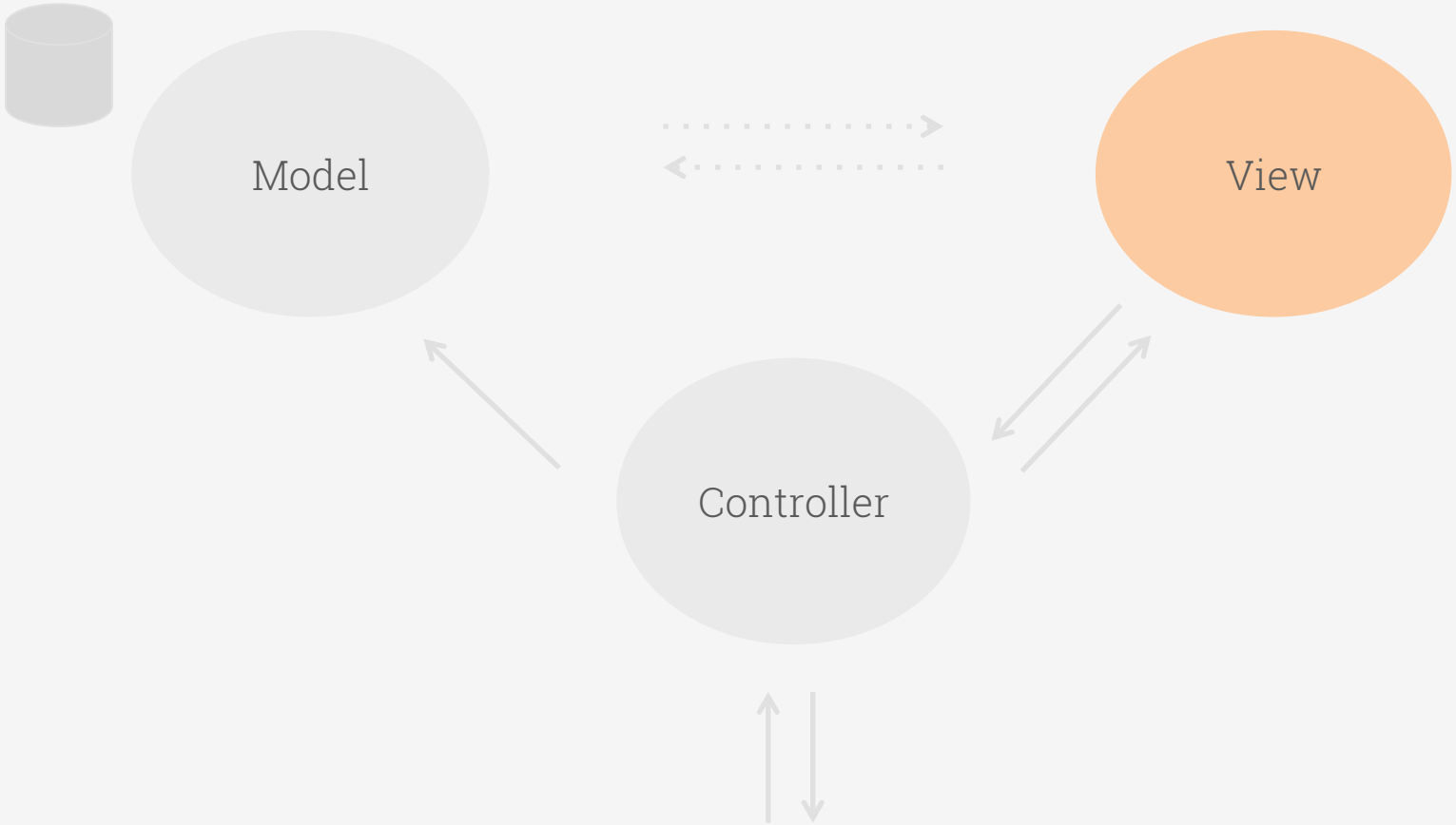
The **Model-View-Controller** pattern



Model represents status and business logic of entities.

Example: User

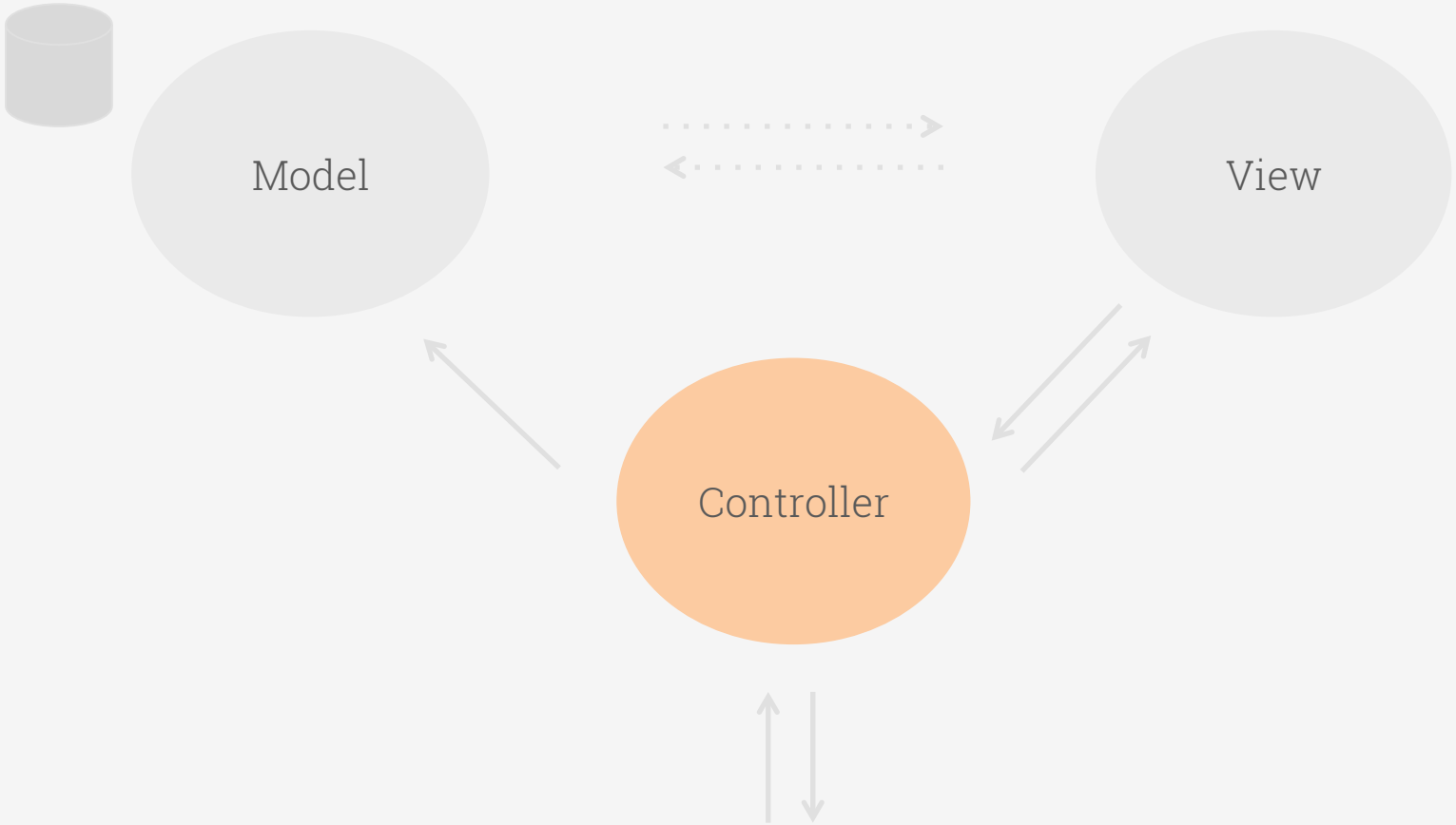
The **Model-View-Controller** pattern



View is the visual representation of data.

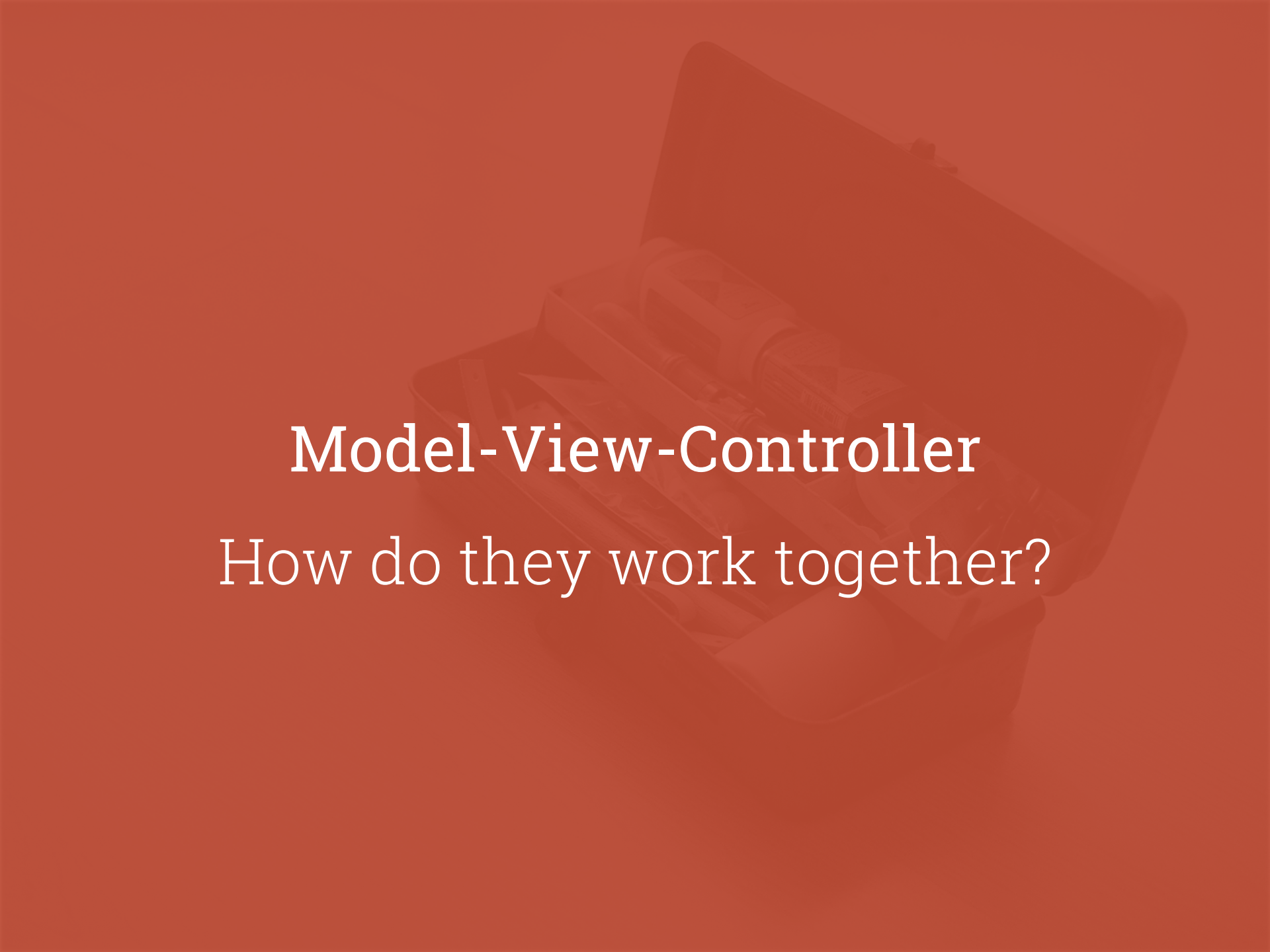
Example: Show details of a given User ID

The **Model-View-Controller** pattern



Controller translates requests in commands and actions.

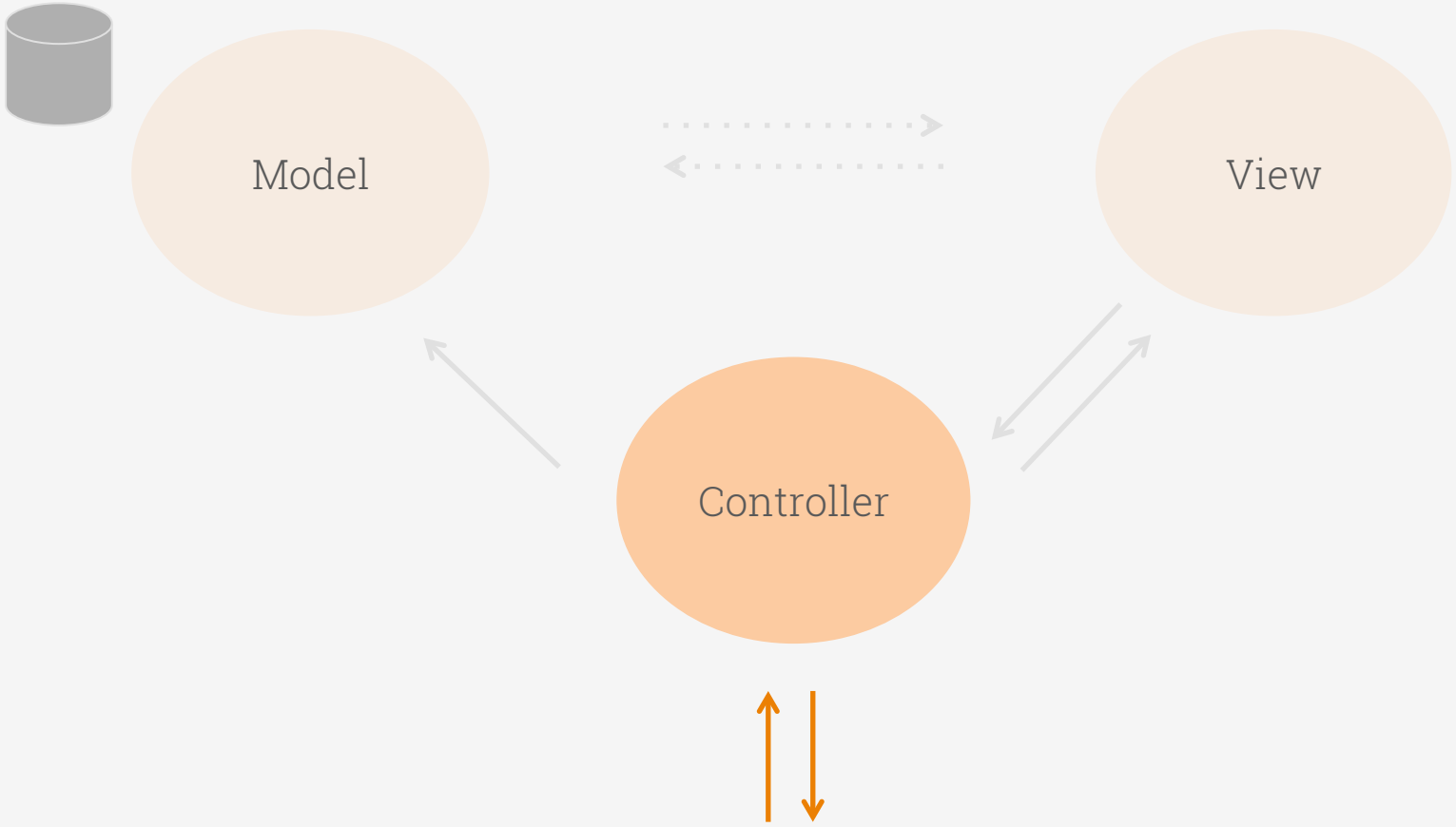
Example: UserController



Model-View-Controller

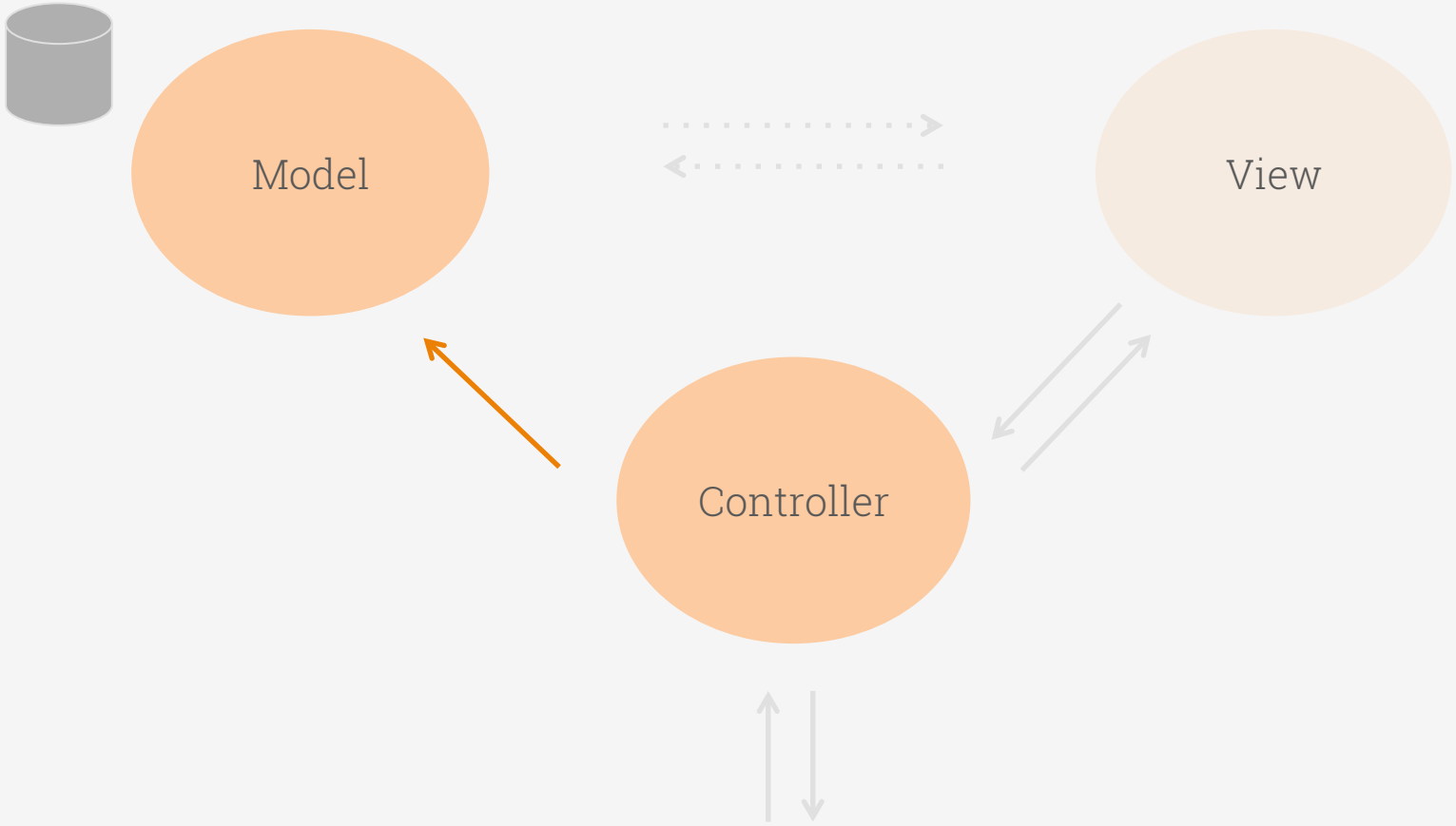
How do they work together?

The **Model-View-Controller** pattern



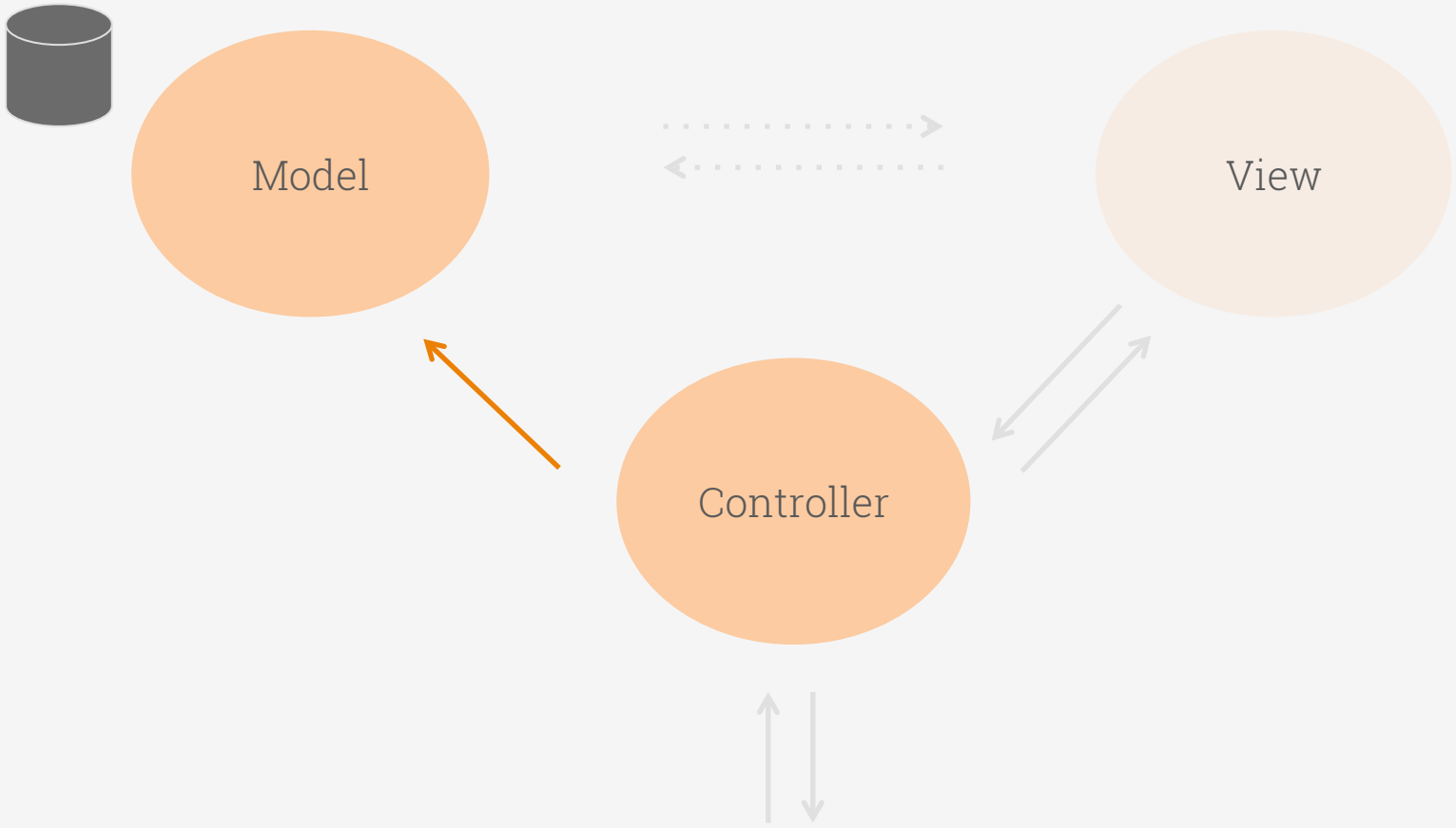
Controller is the entry point. Request is translated to action.

The **Model-View-Controller** pattern



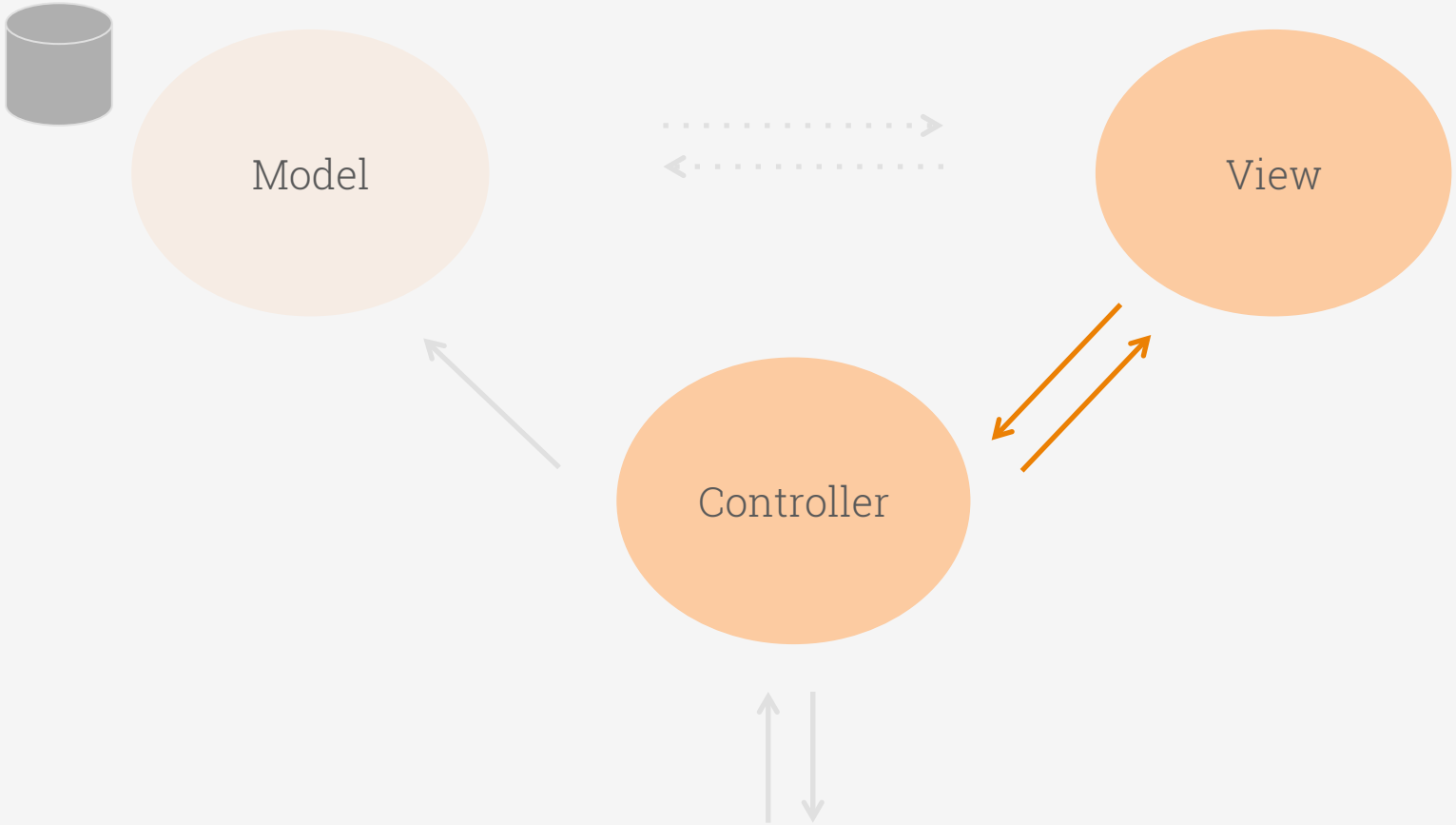
Controller may call one or many **Models** to fetch data.

The **Model-View-Controller** pattern



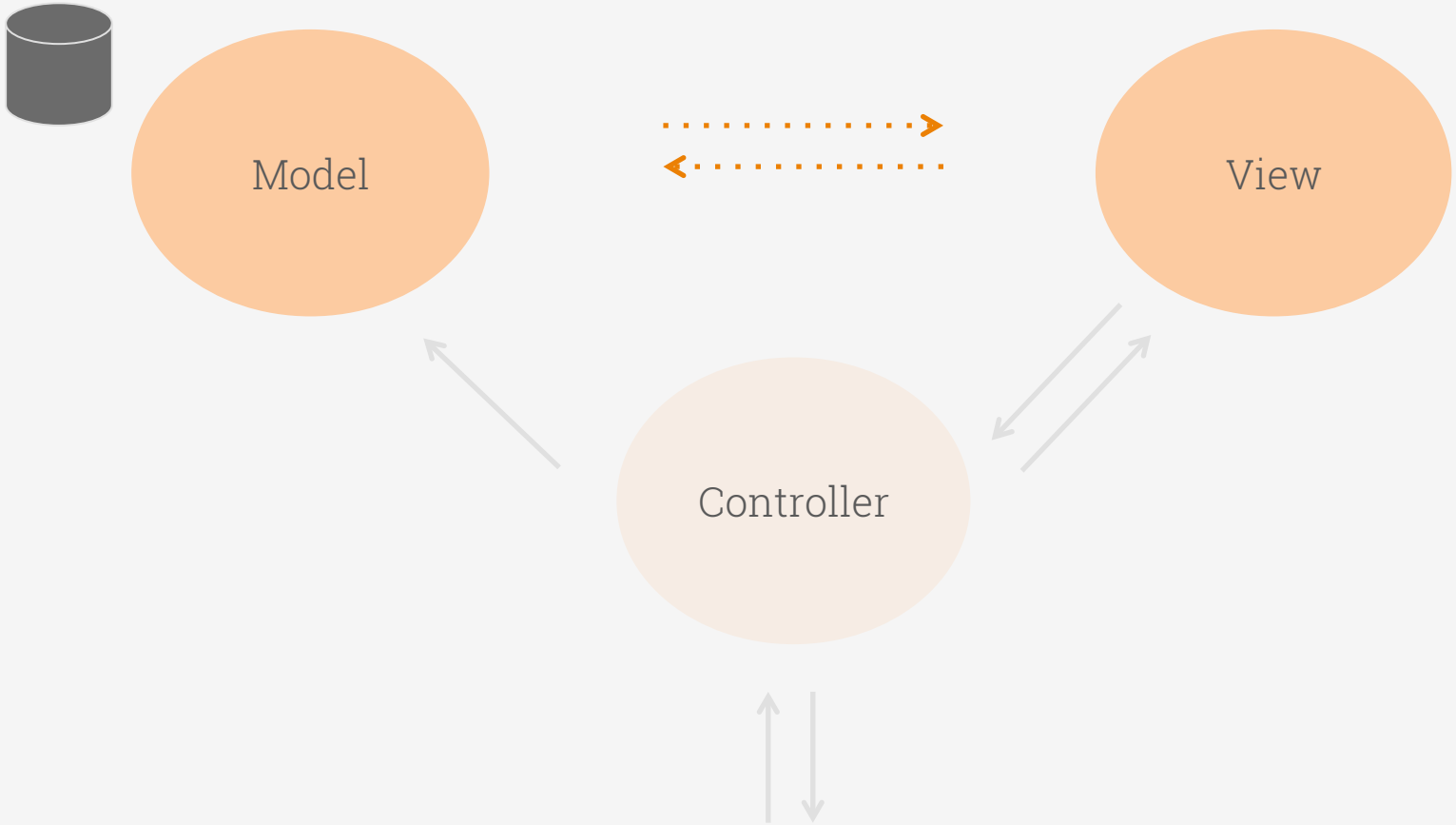
Model fetches data from database and returns it to **Controller**.

The **Model-View-Controller** pattern



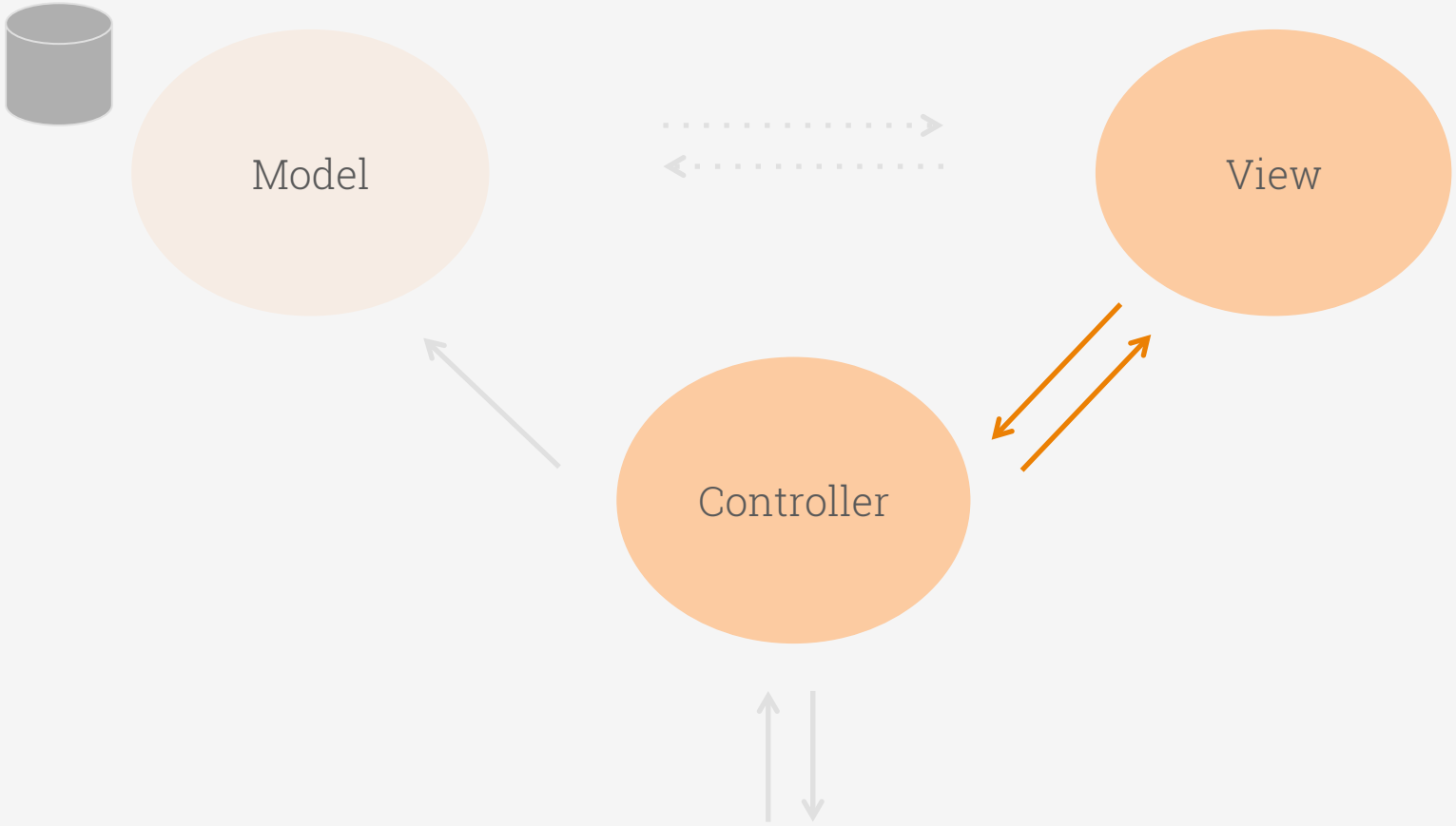
Controller calls **View** and passes the fetched data to it.

The **Model-View-Controller** pattern



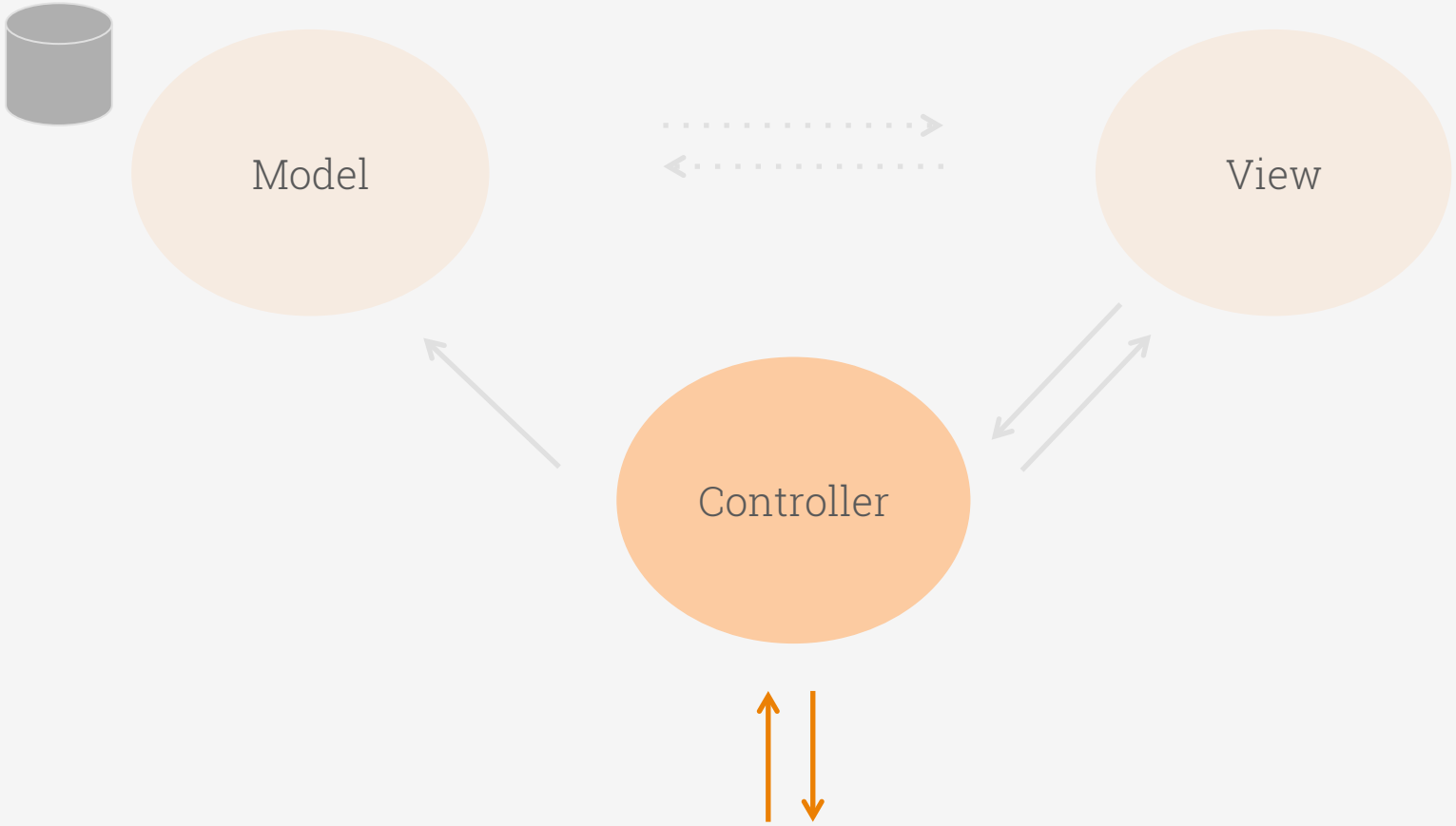
View may fetch specific data from **Model**.

The **Model-View-Controller** pattern



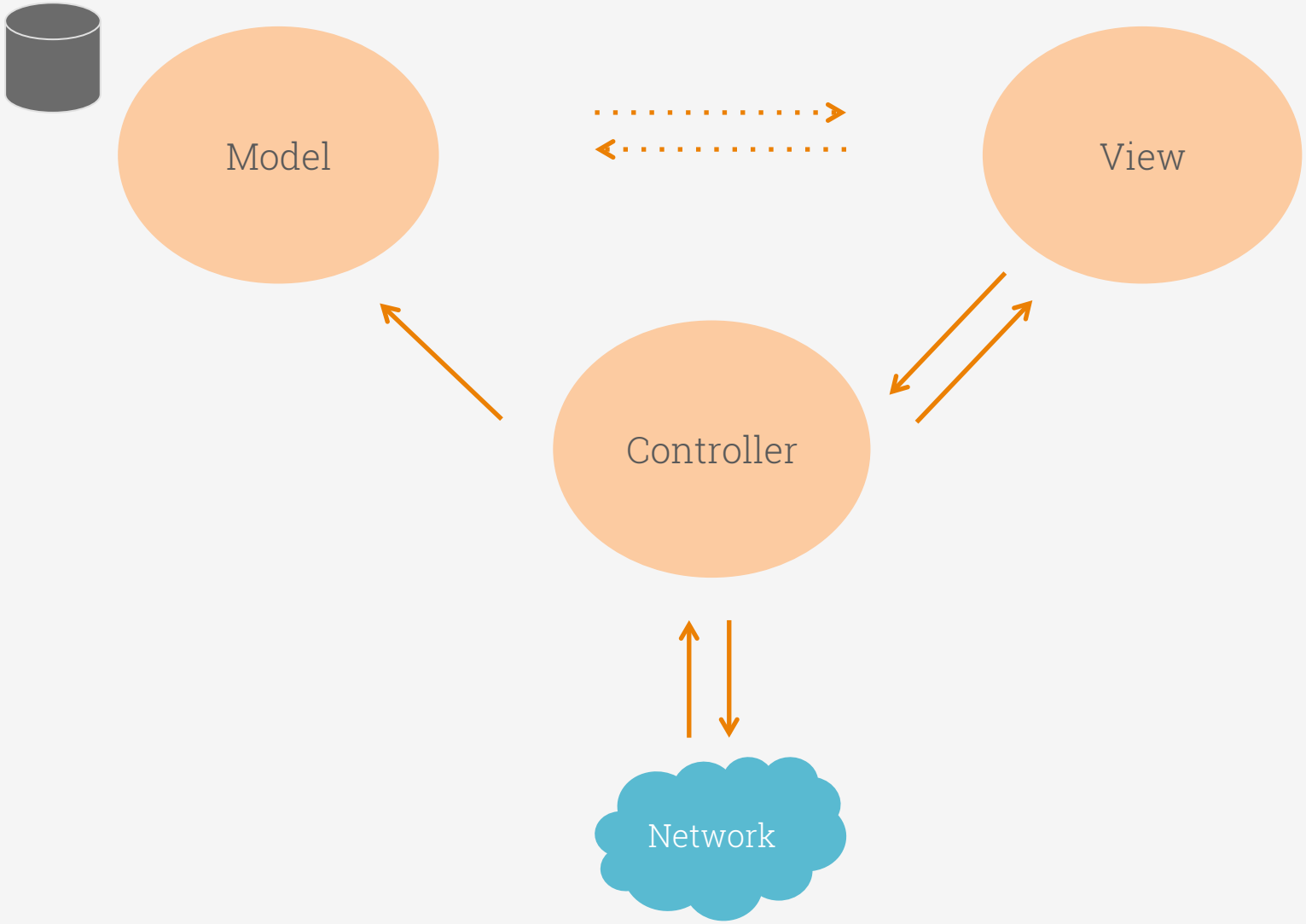
Once **View** is built, then it's returned to **Controller**.

The **Model-View-Controller** pattern



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

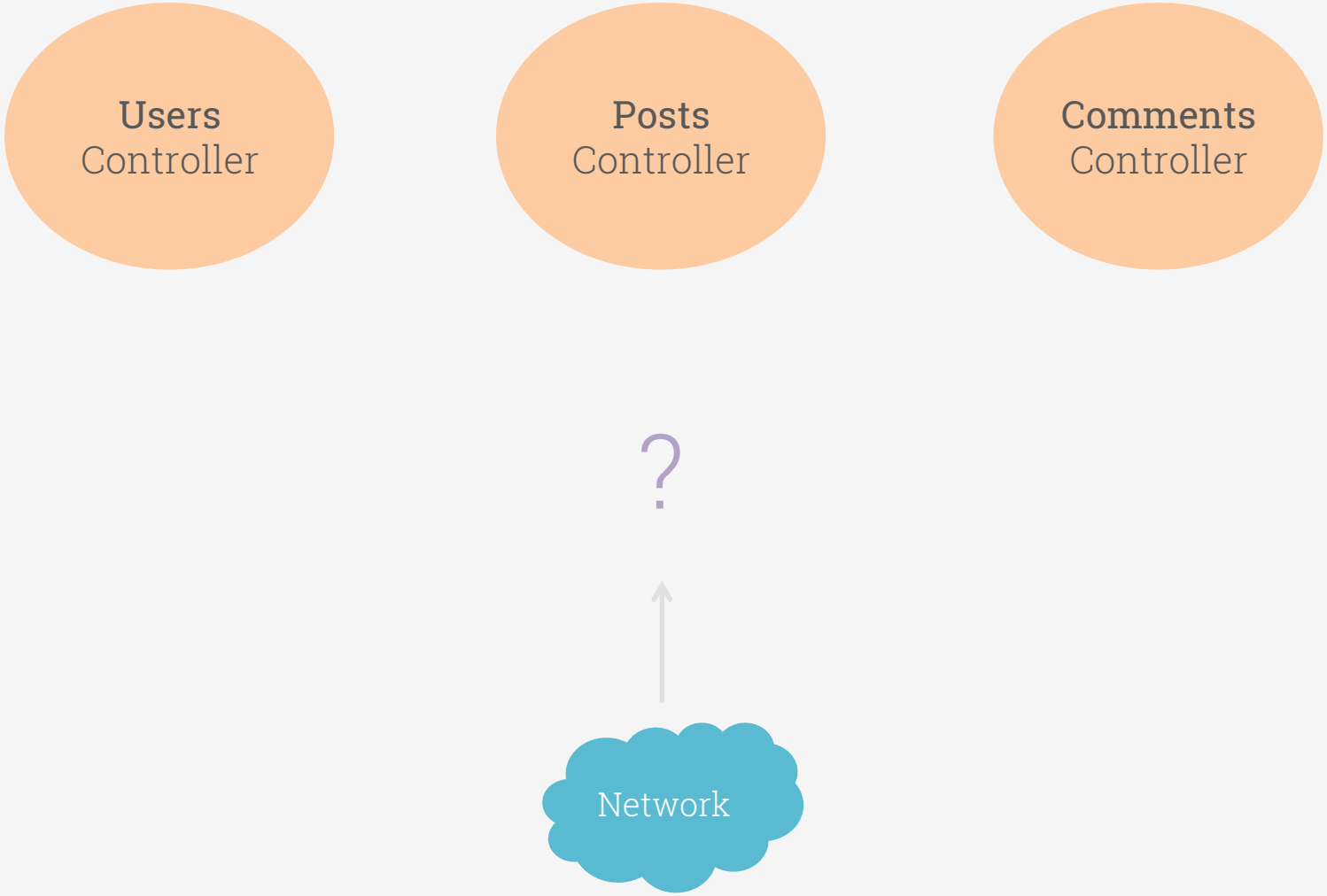
The **Model-View-Controller** pattern



There are many Controllers

How to know which we want?

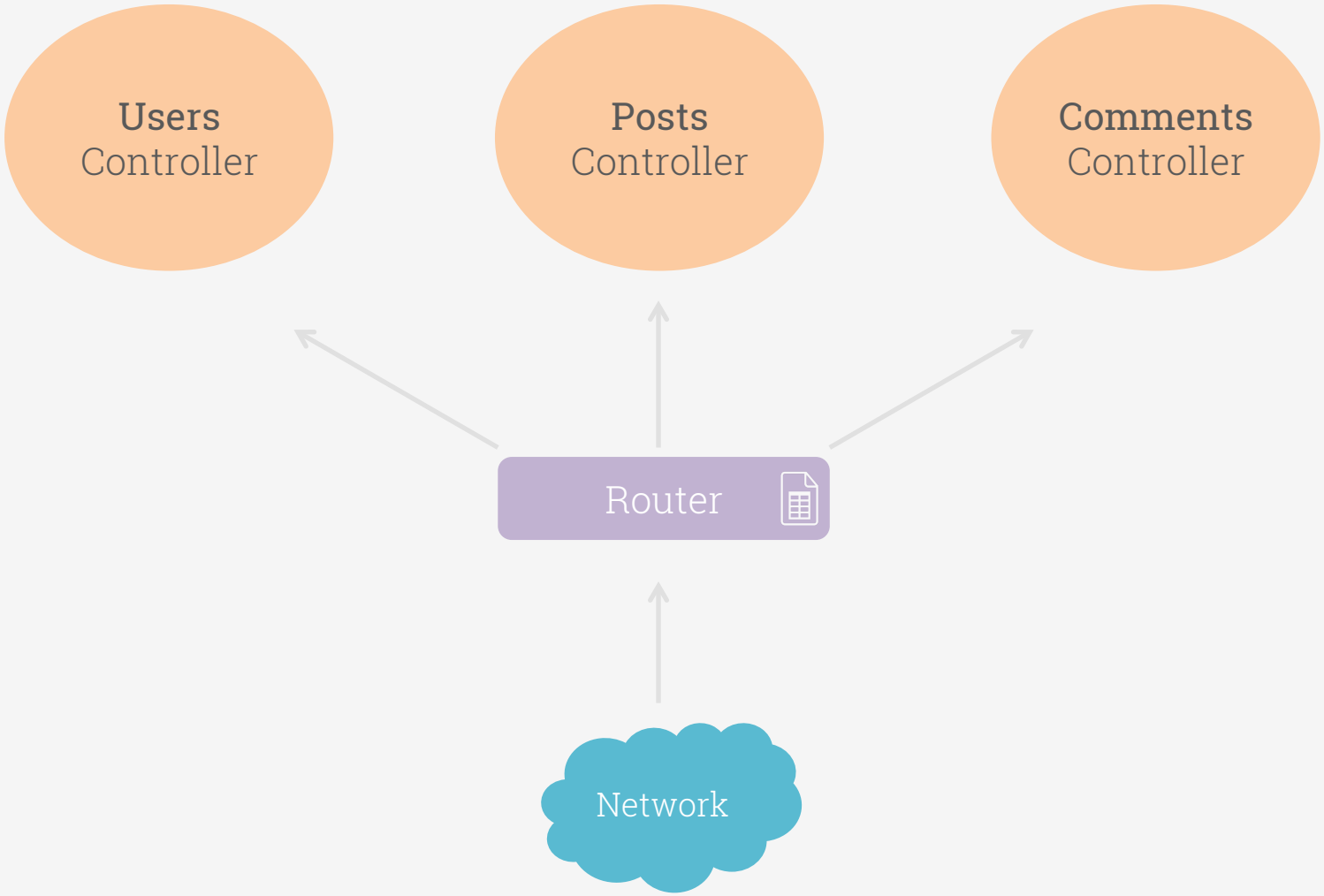
The **Router** component



Router Component

knows how to deal with that.

The **Router** component

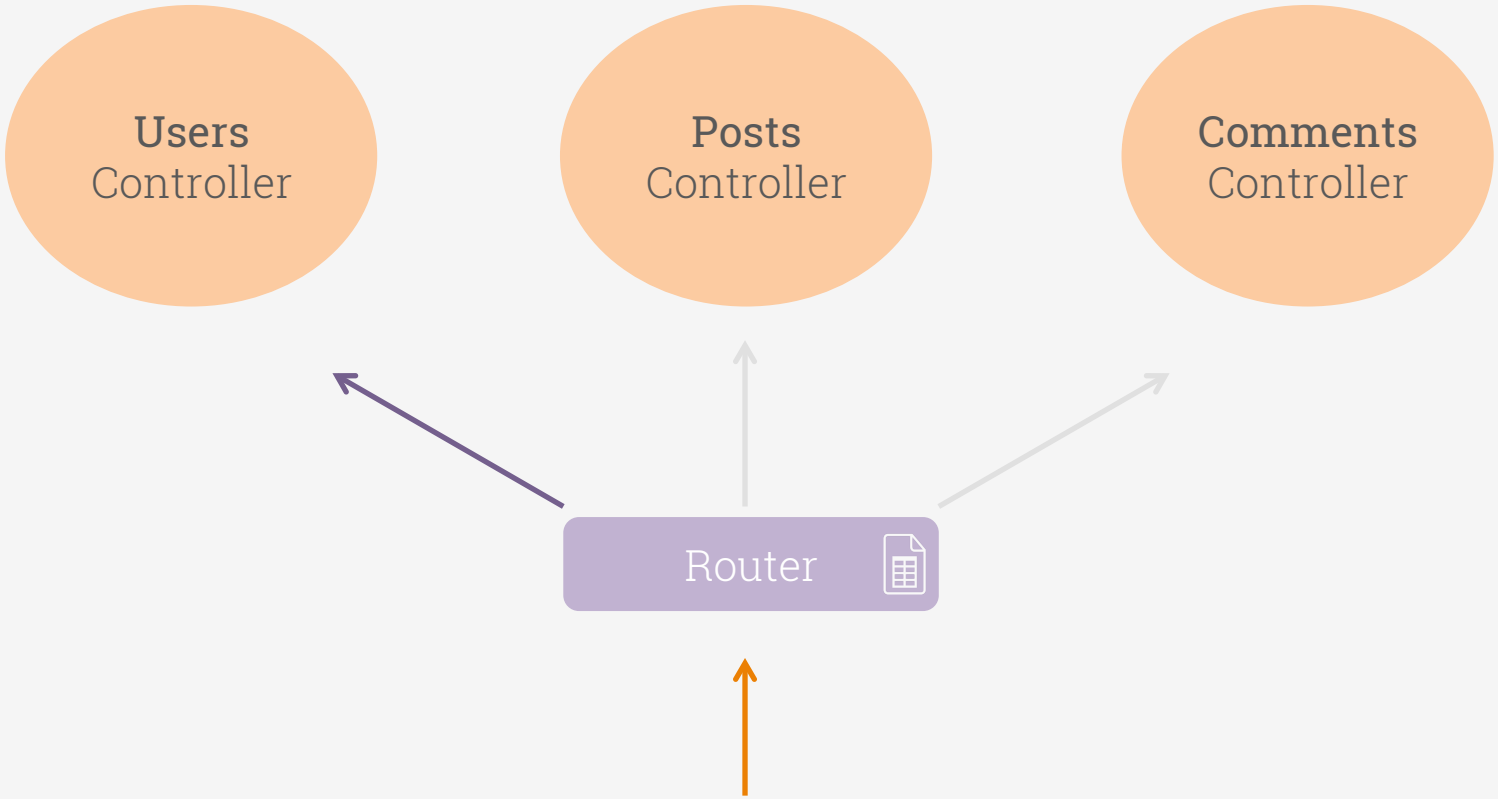


The **Router** component

http://larwitter.app**/users**/...

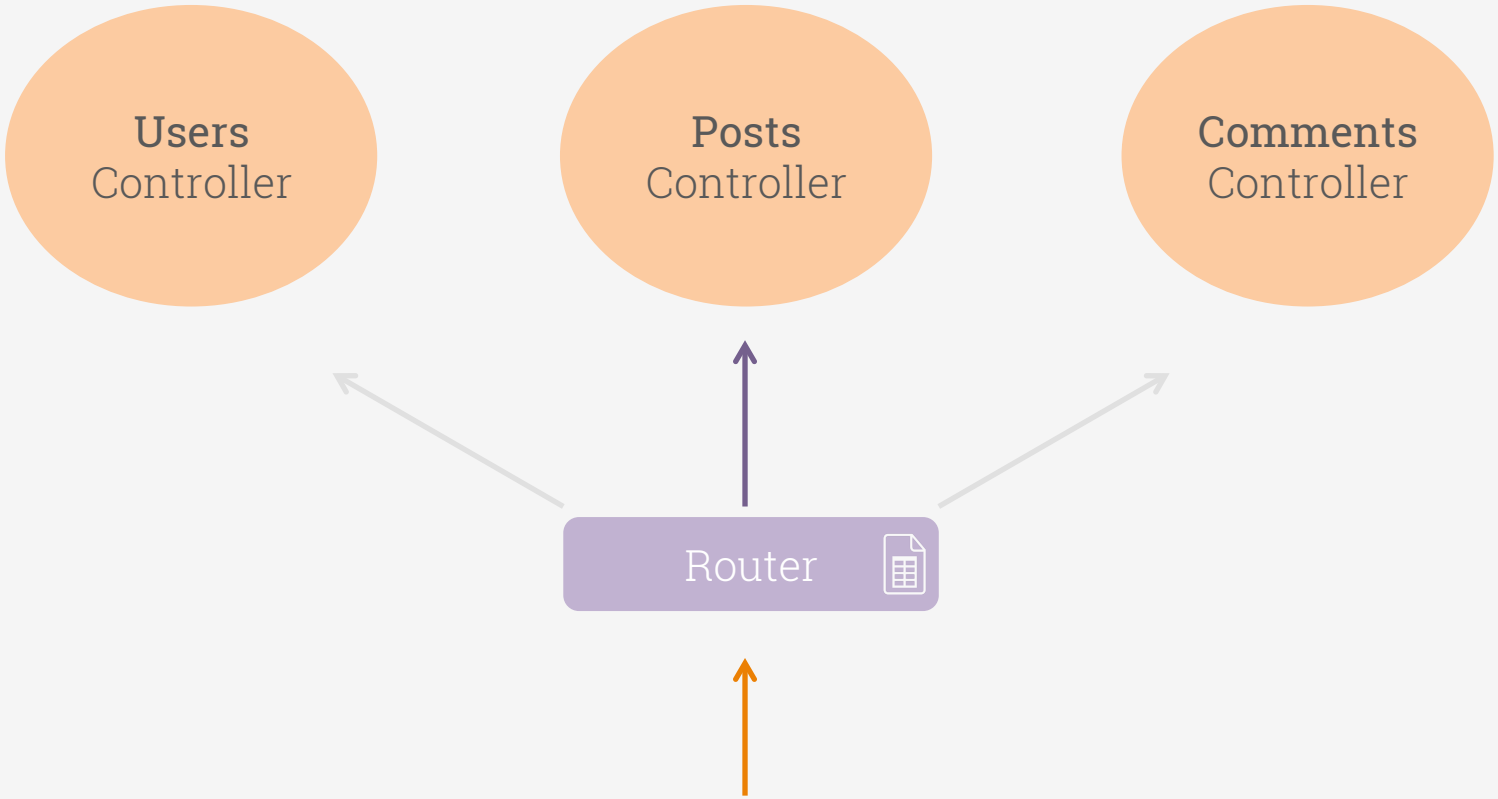
UsersController

The **Router** component



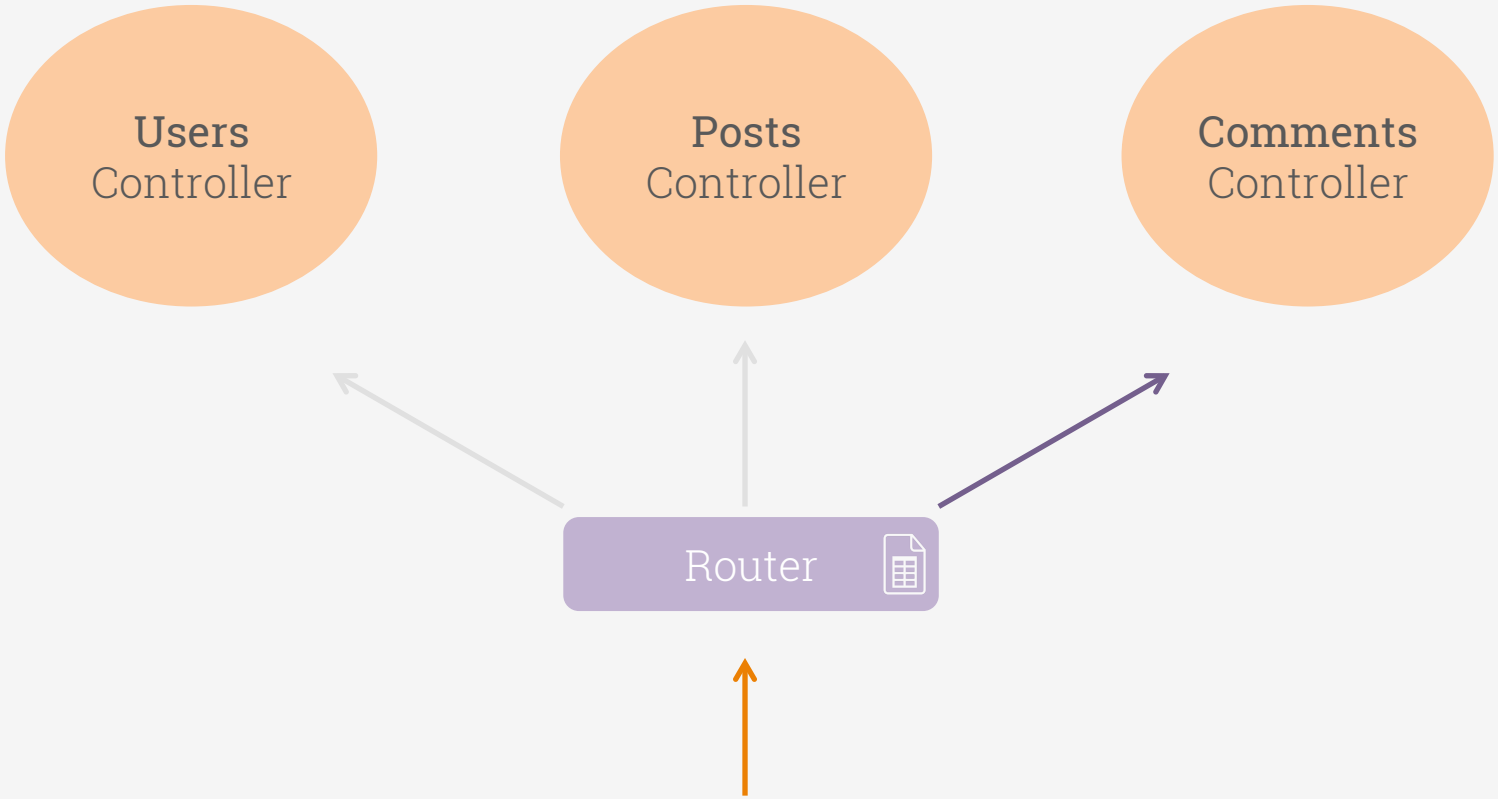
Request **/users** will be routed to **UsersController**

The **Router** component

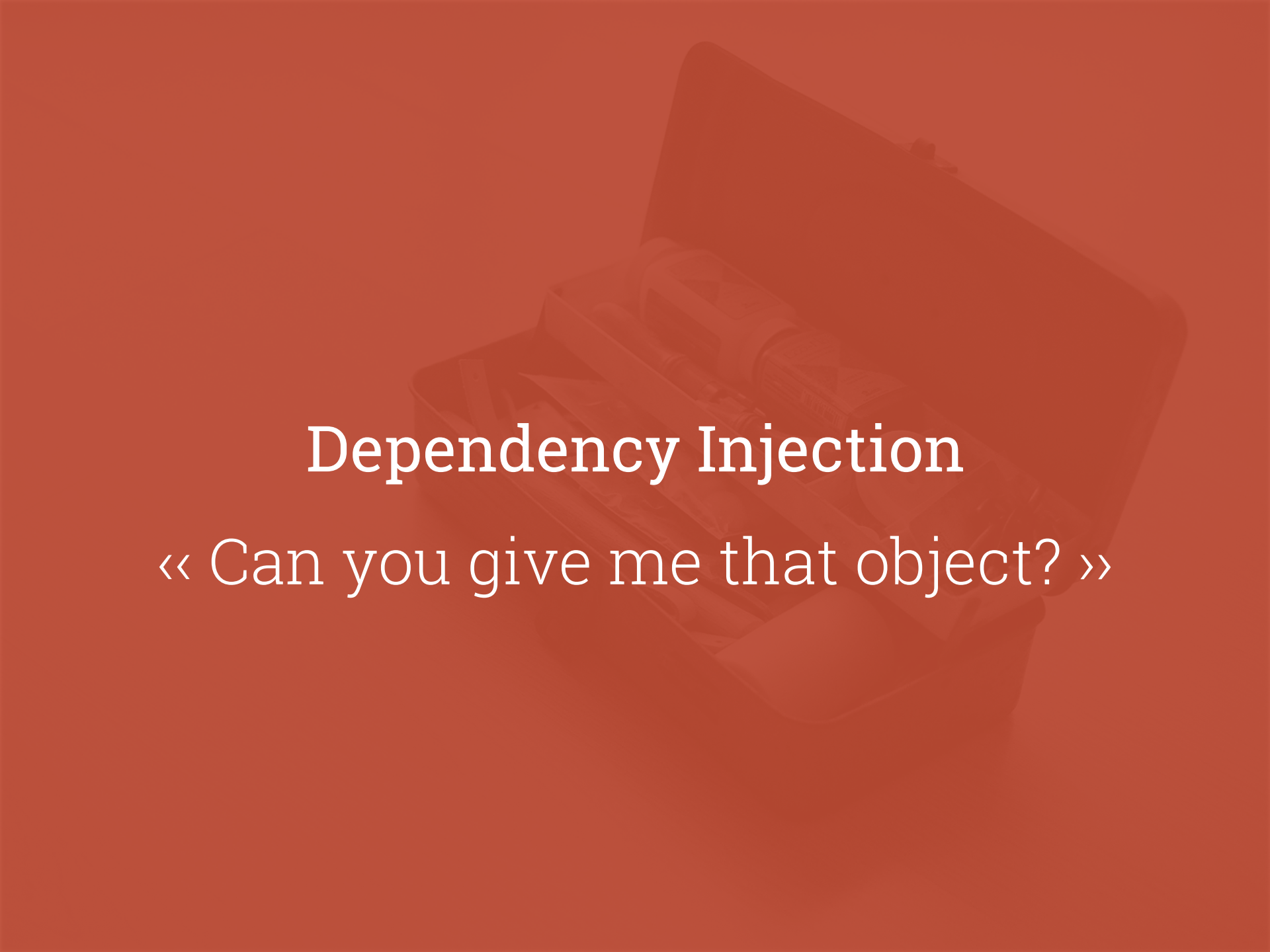


Request **/posts/create** will be routed to **PostsController**

The **Router** component



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

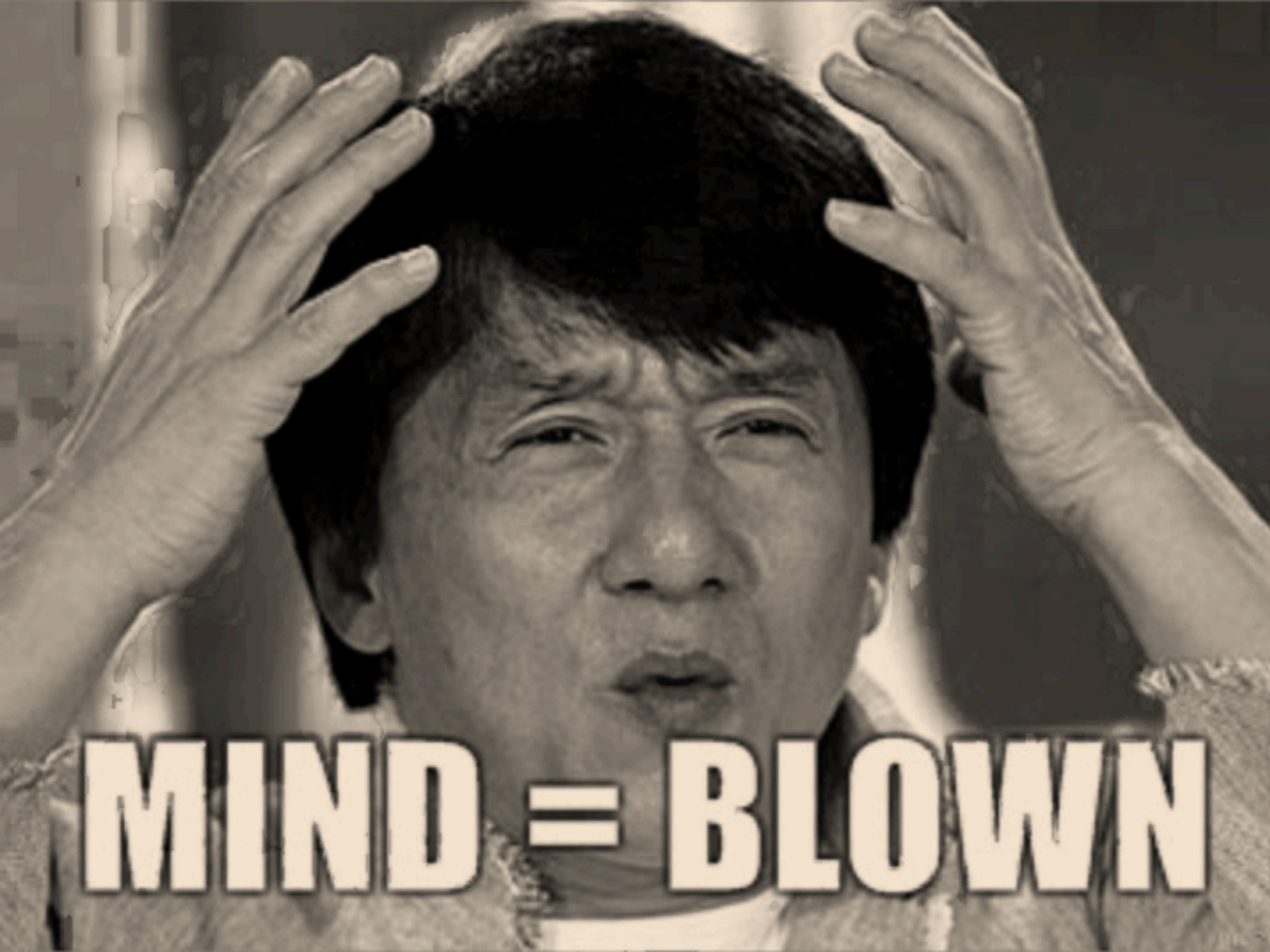


Dependency Injection

‹‹ Can you give me that object? ››

Then we have

Inversion of Control (IoC)


MIND = BLOWN


Let me present to you...


Laravel
The PHP framework for web artisans

What is Laravel?

First release 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*

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*

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*

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*

Project folder structure

Knowing your workspace

The **Folder Structure**

app	Controllers, Application classes, Routes
bootstrap	Bootstrap files like classes auto-loading
config	Application configuration files, i.e. database
database	Database migrations and seeds
public	Public items, like *robots.txt* and *index.php*
resources	Views, template's assets and language files
storage	Storage folder for sessions, files, etc.
tests	Application tests, like Unit Tests
.env.example	Specifies environment, for production or dev
artisan	Swiss army knife for multiple purposes
composer.json	Lists project configuration, like dependencies
gulpfile.js	Task-manager (as Grunt)
package.json	Dependencies for *gulpfile.js*
phpunit.xml	Configuration for Testing environment
server.php	Local server for development only

The swiss army knife

Artisan CLI - Developer’s best-friend

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/master/artisan

Controllers

So many kinds

Basic Controller

You define everything

The Basic Controller

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

app/Http/Controllers/TweetsController.php

```php
<?php namespace Larwitter\Http\Controllers;

use Larwitter\Http\Controllers\Controller;

class TweetsController extends Controller
{
    /*
     * showHello is an action of this controller
     */

    public function showHello()
    {
        return "Hello, welcome to Tweets Controller";
    }
}
```

The Basic Controller

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

app/Http/routes.php

```php
<?php


/*
 * Now, we can access
 * GET http://larwitter.app/tweets
 *
 * Access to /tweets will be routed to
 * showHello action of TweetsController controller
 */

$router->get('/tweets', 'TweetsController@showHello');
```

RESTful Resource Controller

Controller represents resources

The **RESTful Resource Controller**

Verb	Path	Action	Route Name
GET	/resource	index	resource.index
GET	/resource/create	create	resource.create
POST	/resource	store	resource.store
GET	/resource/{resource}	show	resource.show
GET	/resource/{resource}/edit	edit	resource.edit
PUT/PATCH	/resource/{resource}	update	resource.update
DELETE	/resource/{resource}	destroy	resource.destroy

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

The RESTful Resource Controller

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

app/Http/Controllers/TweetsController.php

```php
<?php namespace Larwitter\Http\Controllers;

use Larwitter\Http\Controllers\Controller;

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

The Basic Controller

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

app/Http/routes.php

```php
<?php


/*
 * Now, we can access:
 * GET http://larwitter.app/tweets
 * GET http://larwitter.app/tweets/create
 * ...
 *
 * Access to /tweets will be routed to
 * index action of TweetsController controller
 */

$router->resource('tweets', 'TweetsController');
```

Models

How to create?

The Model

Tweet Model extend Eloquent ORM Model

app/Tweet.php

```php
<?php namespace Larwitter;

use Illuminate\Database\Eloquent\Model;

class Tweet extends Model
{
    // Table in database (default is plural: tweets)
    protected $table = 'cesium_tweets';

    // Fields that can be filled by request (security)
    protected $fillable = ['text'];

    // Fields that are not included in JSON response
    protected $hidden = [];

    ...
}
```

The **Model**

Tweet extended **Eloquent ORM Model**, so you can use few defined methods

app/Http/Controllers/TweetsController.php

```php
<?php namespace Larwitter\Http\Controllers;

use Larwitter\Http\Controllers\Controller;

class TweetsController extends Controller
{
    public function index()
    {
        // Grab all tweets from database
        return Tweet::all();
    }
}

...
```

Take a look to other methods: http://cheats.jesse-obrien.ca/#eloquent

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 templates **should use the .blade.php extension**.

Description from: http://laravel.com/docs/master/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 namespace Larwitter\Http\Controllers;

use Larwitter\Http\Controllers\Controller;

class TweetsController extends Controller
{
    public function index()
    {
        $tweets = Tweet::all();
        return view('views.index',['tweets' => $tweets]);
    }
}

...
```

More about passing data to views: http://laravel.com/docs/master/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>
@stop

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

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

@stop
```

Take a look to other Blade Templating features: http://laravel.com/docs/master/templates#blade-templating

Database migrations

Change database with no effort

The Database Migration

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/master/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/master/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 Seed

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

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

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

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

The Database Seed

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

database/seeds/TweetsTableSeeder.php

```php
<?php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class TweetsTableSeeder extends Seeder
{
    public function run()
    {
        DB::table('tweets')->delete();
        Tweet::create(['text' => 'My first tweet!']);
    }
}
```

The Database Seed

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

database/seeds/DatabaseSeeder.php

```php
<?php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        Model::unguard();
        $this->call('TweetsTableSeeder');
    }
}
```

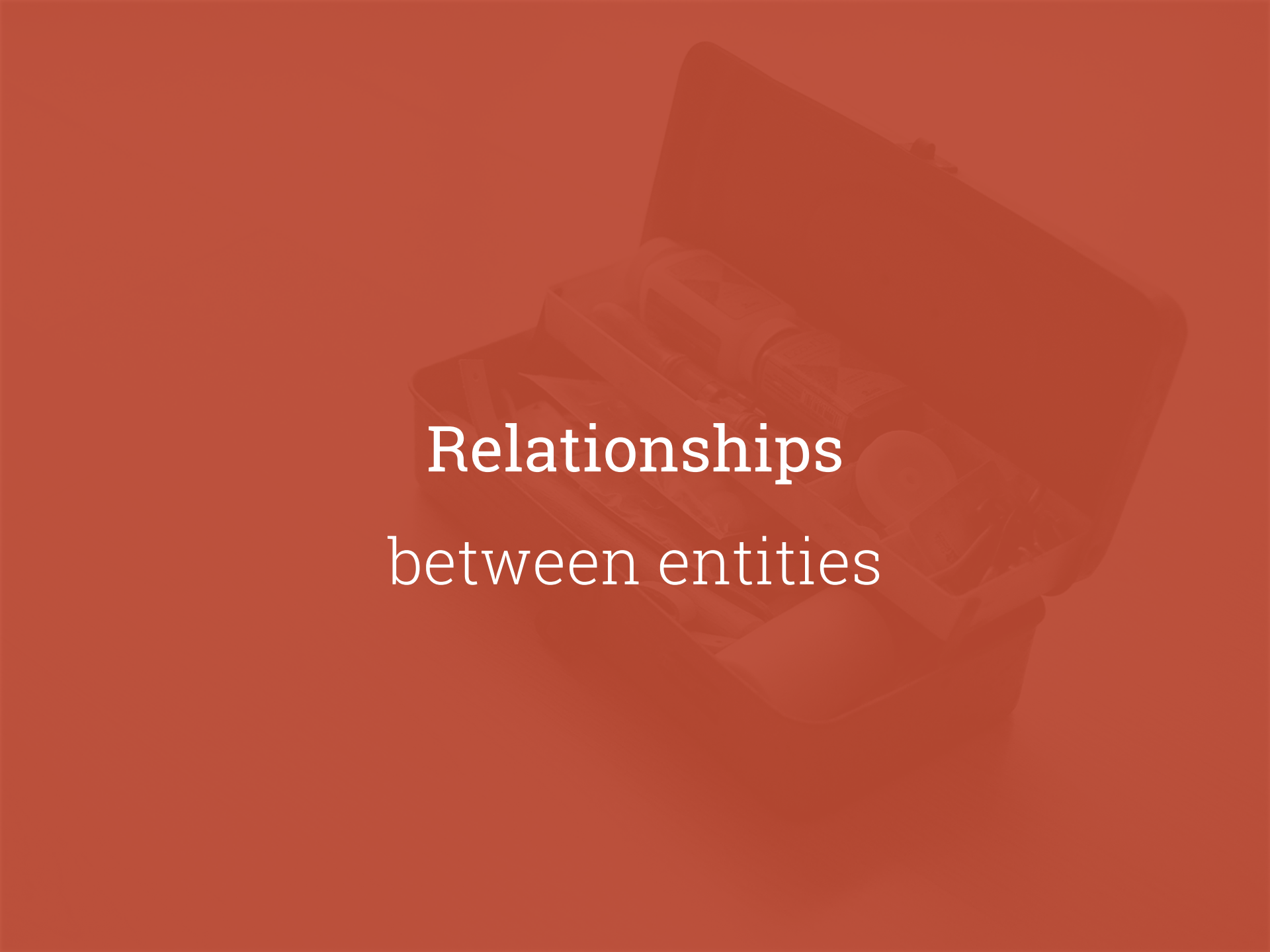
The Database Seed

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



Relationships

between entities

The One-One Relationship

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

Example: Tweet model might have one User.

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

The One-One Relationship

This is how we specify that **one Tweet belongs to one User**

app/Tweet.php

```php
<?php

use Illuminate\Database\Eloquent\Model;

class Tweet extends Model
{
    public function user()
    {
        // From one Tweet, we can get its owner
        return $this->belongsTo('User');
    }
}
```

The One-Many Relationship

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

One User has many Tweets.

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

The One-Many Relationship

This is how we specify that **one User has many Tweets**

app/User.php

```php
<?php

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    public function tweets()
    {
        // From one User, we can get his tweets
        return $this->hasMany('Tweet');
    }
}
```

The **Many-Many Relationship**

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

Many Users has many Jobs.

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

The Many-Many Relationship

This is how we specify that **many Users has many Jobs**

app/User.php

```php
<?php

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    public function jobs()
    {
        // From one User, we can get his jobs
        return $this->hasMany('Job');
    }
}
```

The Many-Many Relationship

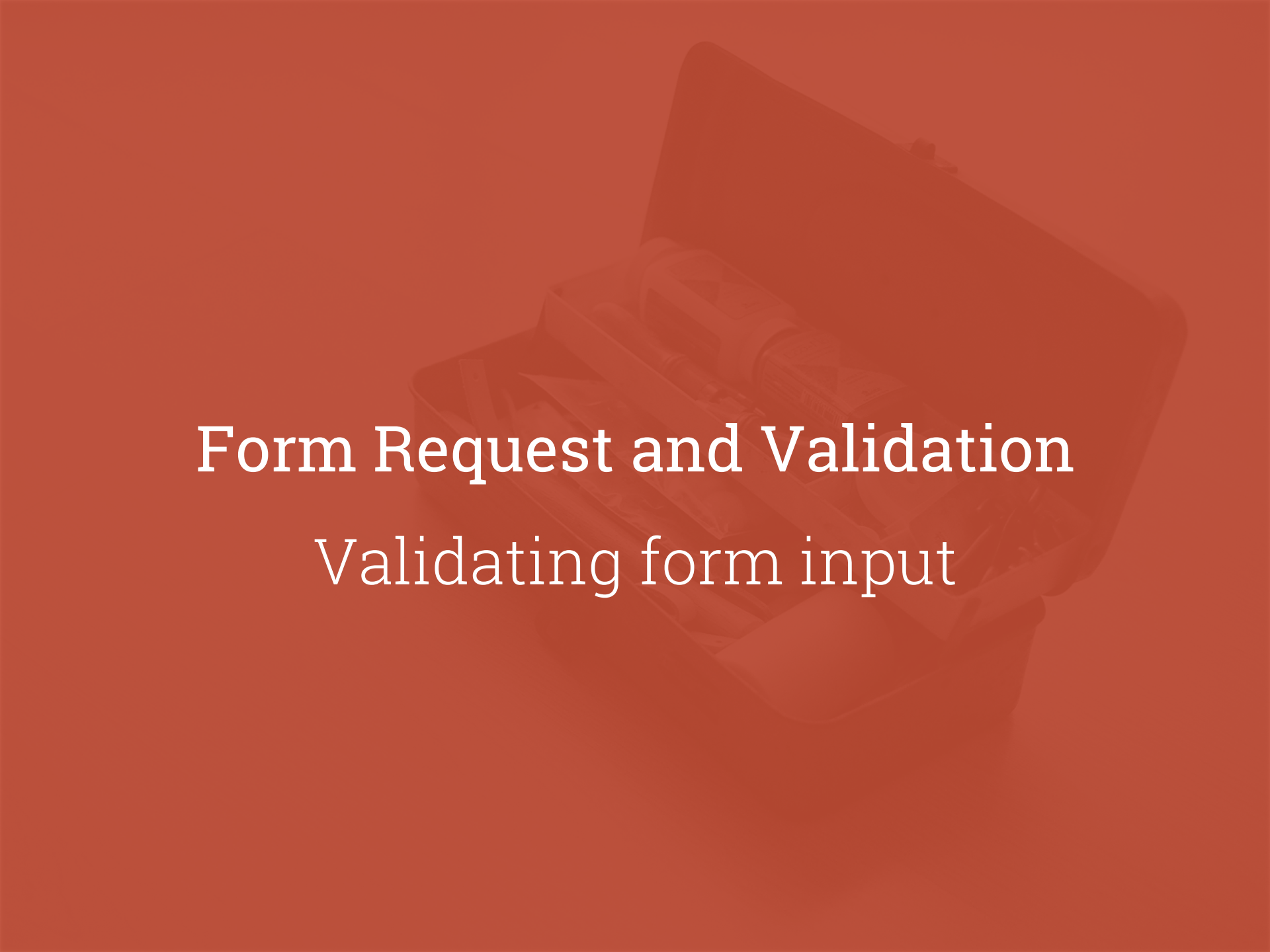
This is how we specify that **many Jobs belong to many Users**

app/Job.php

```php
<?php

use Illuminate\Database\Eloquent\Model;

class Job extends Model
{
    public function users()
    {
        // From one Job, we can get its Users
        return $this->belongsToMany('User');
    }
}
```



Form Request and Validation

Validating form input

The One-One Relationship

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

app/Requests/CreateTweetRequest.php

```php
<?php namespace App\Http\Requests;

use App\Http\Requests\Request;

class CreateTweetRequest extends Request
{
    public function rules()
    {
        return [
            'text' => 'required|max:190'
        ];
    }
    public function authorize()
    {
        return true;
    }
}
```

The One-One Relationship

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

app/Controllers/TweetsController.php

```
...

public function store(CreateTweetRequest $request)
{
    // Validation succeeded at this point

    // Note that we use Method injection here
}

...
```

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


That's all Folks!