Introduction to MVC and Laravel

Old School PHP

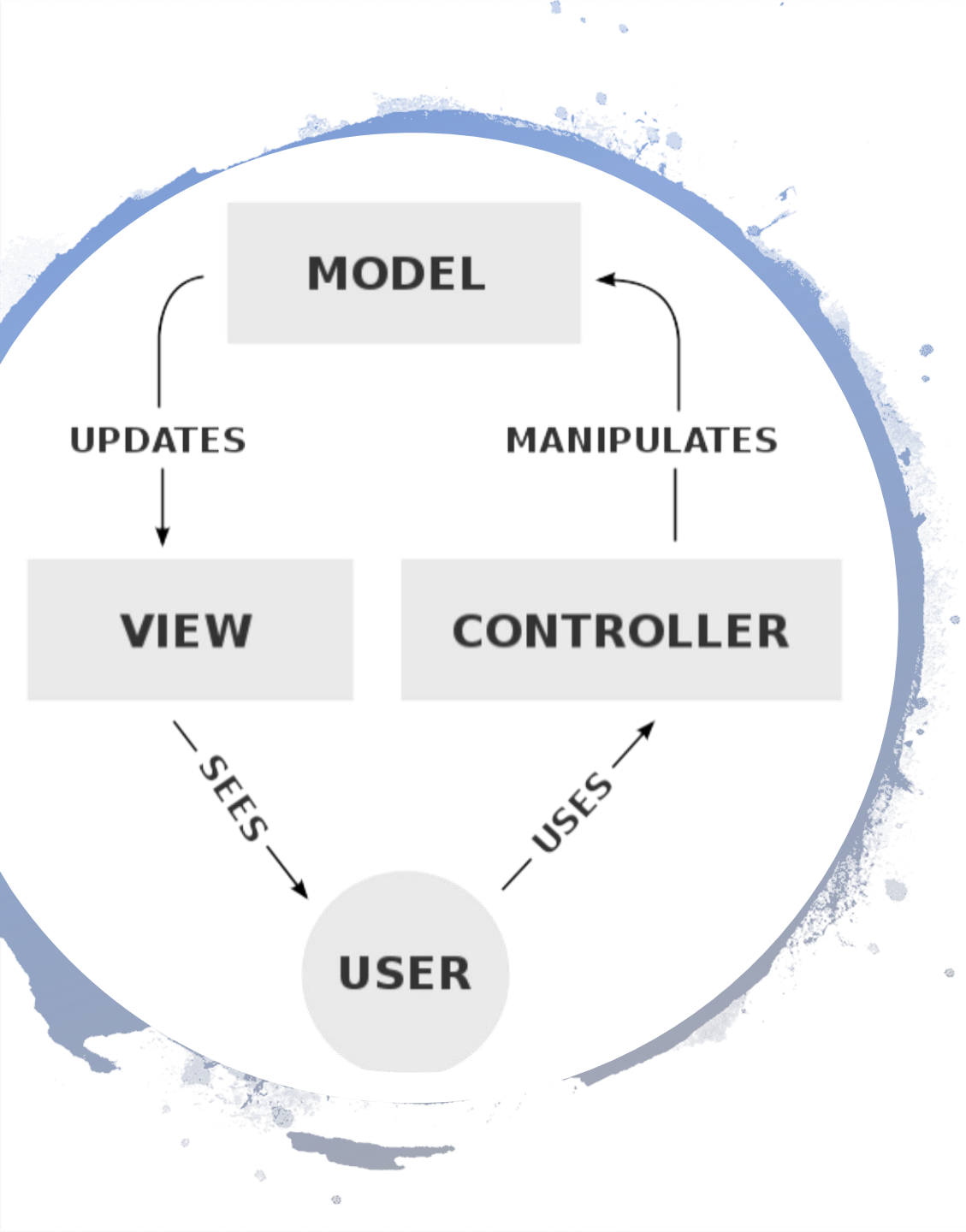
- We all started out using PHP scripts like this… (literally last lecture)
  - Output an HTML Header
  - Output start of the body
  - Output a navbar
  - Connect to a Database
    - If this fails output an error in HTML and abort
  - Pull records from database
  - Perform some kind of logic
  - Loop around all this outputting data as HTML as you go

The problem with old school

- Technically it works but you end up with messy scripts which mix up…
  - The business logic (i.e. what you want the application to do with your data)
  - The logic related to accessing the database
  - The front end presentation
  - Error handling
  - …and probably lots more
- This doesn’t scale well and results in lots of copied and pasted boilerplate code all over your application
- …this is why we now use frameworks with strict software design patterns

The MVC Pattern

- Model-View-Controller (MVC) is an architectural pattern
- First introduced in 1979 as Thing-Model-View-Editor
- **The goal is to separate concerns within an application**
- This pattern can be applied to any application by a disciplined programmer but now many frameworks enforce MVC, or at least make it easy to use (hard not to)
- MVC applies it self extremely well to web applications



The MVC Pattern in general

- MVC separates the UI of an application into three main aspects
  - The Model: A set of classes that describes the data you are working with and the rules for how the data can be manipulated.
  - The View: Defines how the application's UI will be displayed
  - The Controller: A set of classes which handles communication from the user, the overall application flow and application specific logic. This tells the view what to display and also updates the model.

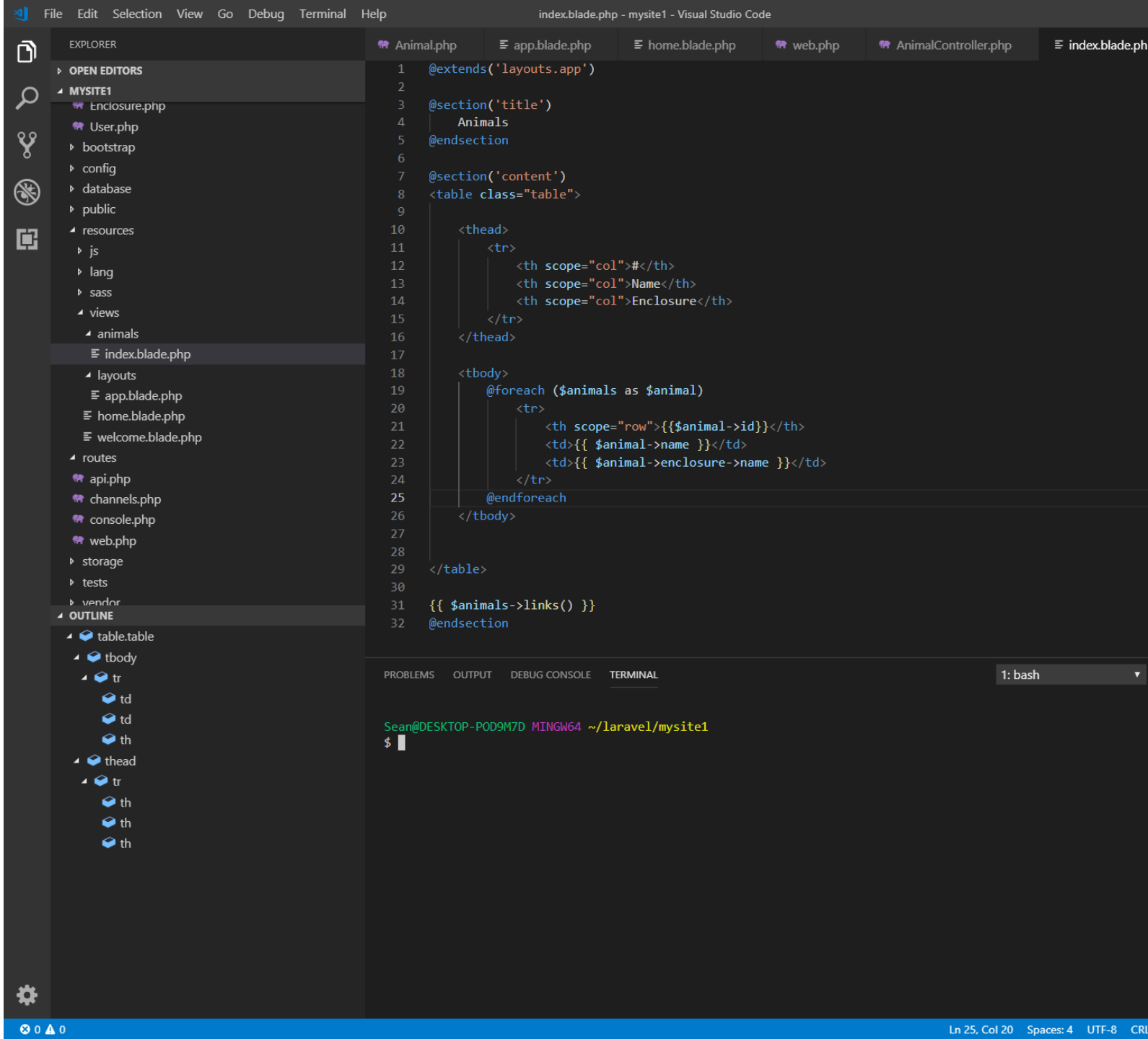
Laravel

- Frameworks are essentially lots of components and packages put together in a elegant and logical way
  - Someone else has made all the important decisions for you which means you can concentrate on making something!
- Laravel is a PHP framework designed for developers – Laravel wants you to be happy – it is elegant and a joy to work with
- Laravel provides an entire ecosystem of tools to help you with testing and deployment
- The community is welcoming and full of people who want to teach you

Warning: When you start working with frameworks you'll be creating projects with lots of files and folders you don't understand… …don't get overwhelmed!

Visual Studio Code

- Light weight text editor with built in terminal and great extensions
- Recommended extensions
  - Laravel 5 snippets
  - Laravel Blade Snippets
  - PhP intellisense (for this to work you need PhP installed locally)
- If you are on windows I recommend switching the terminal to bash (which you can get from a git install)



Setting up a local development environment

- Essentially two options
  1. Install everything manually on your PC (or use Valet on MacOS) then run a server locally – mess with lots of config files to get a bad approximation of an external server
  2. Install everything on a virtual machine which acts like a remote server
- You should pick option (2) and use Laravel Homestead
  - https://laravel.com/docs/5.8/homestead

Laravel Homestead

- Homestead is a pre-packaged virtual machine using the Vagrant system
- The Laravel developers have set up everything you need in on easy to download thing – you just need to do a little configuration
- A folder on your host machine will be mapped to the virtual machine which means you can edit Laravel code and test it in real time
- First download and install VirtualBox and Vagrant

...also you need git installed

(But if you don’t already have that installed why are you doing a computer science degree?)

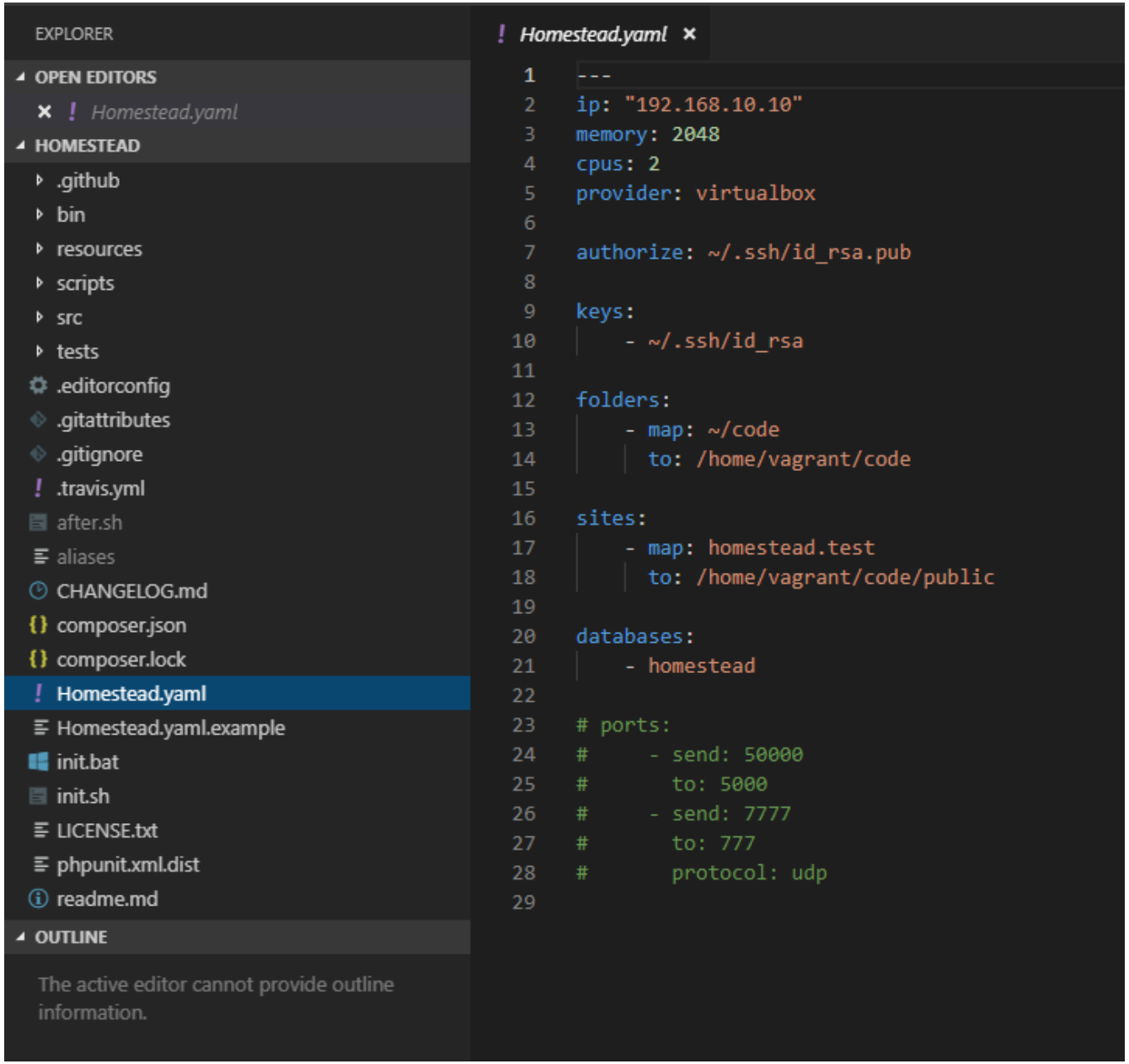
You may need to enable virtualization in your bios! (The setting can be hidden, especially on AMD, so google)

Installing and Configuring Homestead

In a terminal run ‘vagrant box add Laravel/homestead’ to install the homestead box

Then you will need to clone the Homestead config files ‘git clone https://github.com/laravel/homestead.git ~/Homestead’ cd to that directory then run the init script

Default settings



The homestead folder we created will hold all the config information about our virtual machine

The virtual machine specs at the top can be left as is.

Delete the ‘authorize’ and ‘keys’ properties – these will be sorted out by vagrant

The folders and sites properties are where we configure the websites.

Folders maps a folder on your host machine (where you will do the editing) to the virtual machine. *Make sure to create the folder on your host machine before launching the VM!!*

The sites maps domains to folders. .test is reserved for testing so use .test on all your sites. You would typically have one entry in for each website.

Homestead.yaml

```yaml
---
ip: "192.168.10.10"
memory: 2048
cpus: 2
provider: virtualbox

folders:
    - map: ~/Laravel
      to: /home/vagrant/Laravel

sites:
    - map: homestead1.test
      to: /home/vagrant/Laravel/homestead1/public
    - map: homestead2.test
      to: /home/vagrant/Laravel/homestead2/public

databases:
    - homestead

# ports:
#     - send: 50000
#       to: 5000
#     - send: 7777
#       to: 777
#       protocol: udp

```

Hosts File

- Even with all this configures correctly and running if we went to homestead1.test our browser would try to find an external website with that address
- We need to edit the Hosts File to override that
- On Mac and Linux: /etc/hosts
- On Windows: C:\Windows\System32\drivers\etc\hosts

Homestead.yaml hosts

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host nam
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#   127.0.0.1       localhost
#   ::1             localhost

192.168.10.10       homestead1.test
192.168.10.10       homestead2.test
```

To Run the VM

- cd to the Homestead folder
- To start the machine run ‘vagrant up’
- ‘vagrant ssh’ then automatically logs you into the virtual machine – this is where you’ll be doing lots of Laravel related commands
- To shut down the VM use ‘vagrant halt’ and to destroy it ‘vagrant destroy’

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Sean Walton@DESKTOP-2P7AUB7 MINGW64 ~/Homestead (master)
$ vagrant up
Bringing machine 'homestead-7' up with 'virtualbox' provider...
==> homestead-7: Importing base box 'laravel/homestead'...
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-50-generic x86_64)

Thanks for using
 _                                _                  _
| |                              | |                | |
| |__   __  _ __  __   __  __| |_ __  __ _  __| |
| '_ \ / _ \| '_ `_ \ / _ \/ __| __/ _ \/ _` |/ _` |
| | | | (_) | | | | | |  __/\__ \ ||  __/ (_| | (_| |
|_| |_|\___/|_| |_| |_|\___||___/\__\___|\__,_|\__,_|

* Homestead 8.4.0 released!
* Settler v7.2.1 released! Make sure you update

0 packages can be updated.
0 updates are security updates.


vagrant@homestead:~$
```

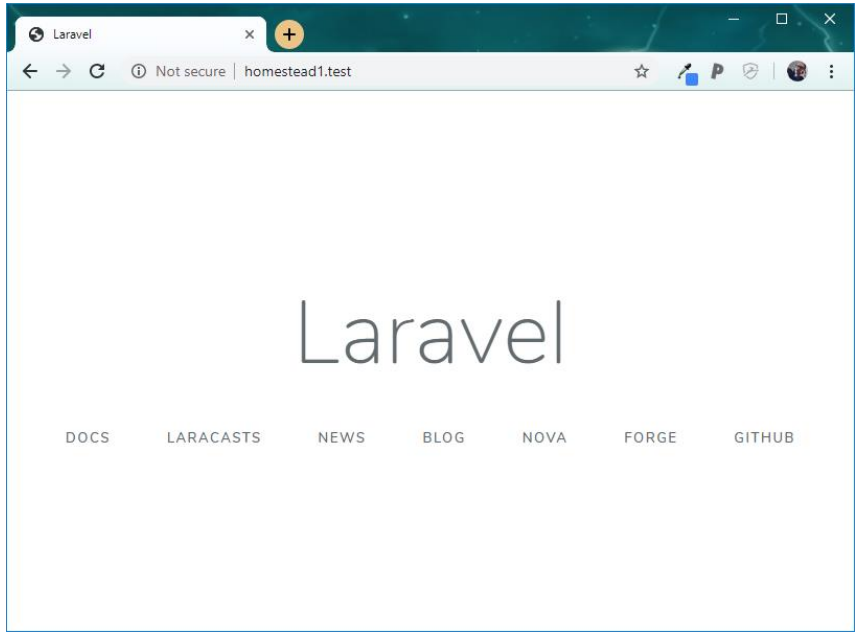
Composer

- Composer is a Package Manager for PHP
- Your homestead VM has the latest version installed which will be used to install Laravel and everything it needs

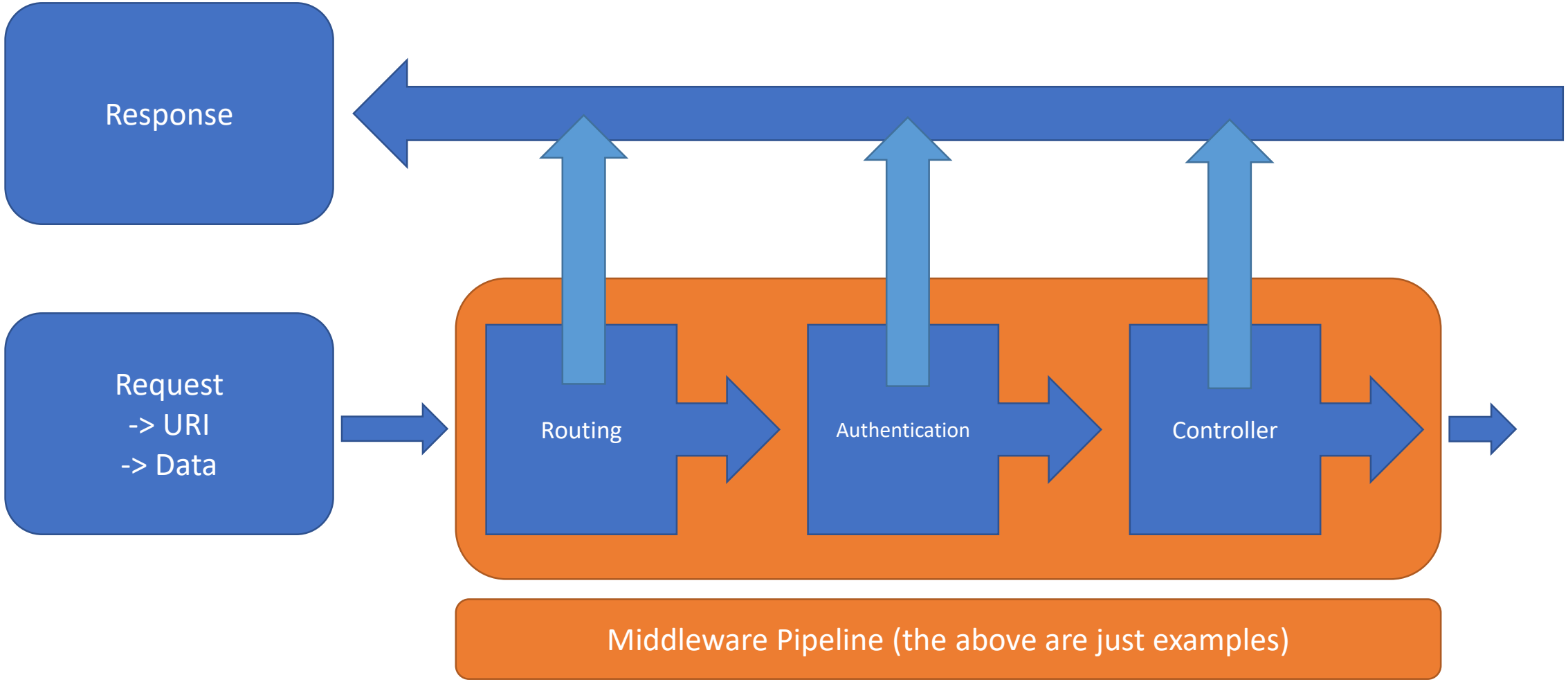
Creating a Laravel Project

- SSH into the VM using Vagrant
- cd to your source folder (in my case Laravel)
- Run ‘laravel new homestead1’ replacing homestead1 with one of the sites you configured in Homestead.yaml
- You can now navigate to that URL in a browser and see the default project page!

```
sebastian/global-state suggests installing ext-uopz (*)
phpunit/php-code-coverage suggests installing ext-xdebug (^2.6.0)
phpunit/phpunit suggests installing ext-xdebug (*)
phpunit/phpunit suggests installing phpunit/php-invoker (^2.0)
Generating optimized autoload files
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
> @php artisan key:generate --ansi
Application key set successfully.
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: beyondcode/laravel-dump-server
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
Application ready! Build something amazing.
vagrant@homestead:~/Laravel$
```

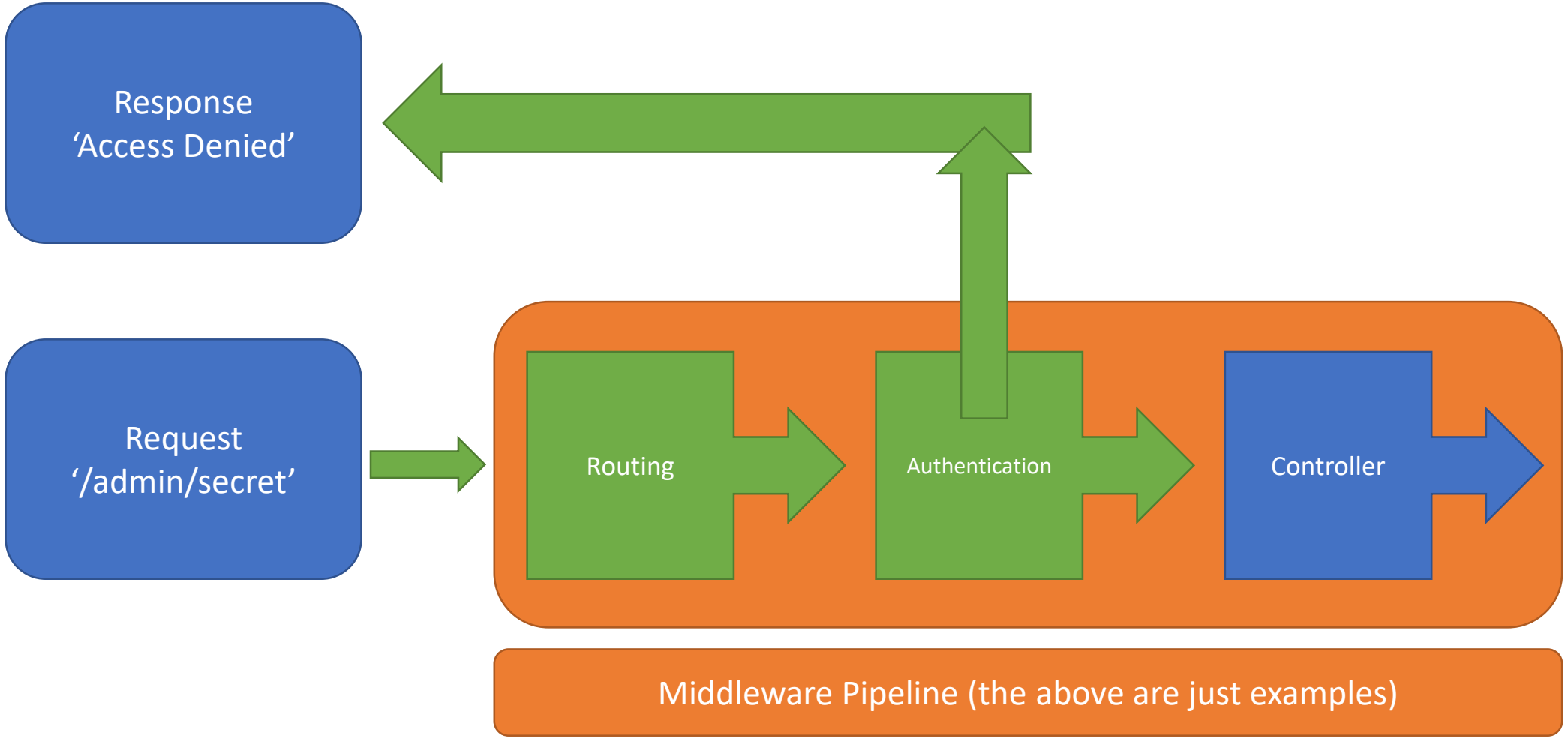


Web Framework Basic Structure



A request enters the middleware pipeline (through index.php) and is passed along until one of the middleware packages creates a response.

If a request gets all the way through the pipeline with no request generated an error is sent to the user.

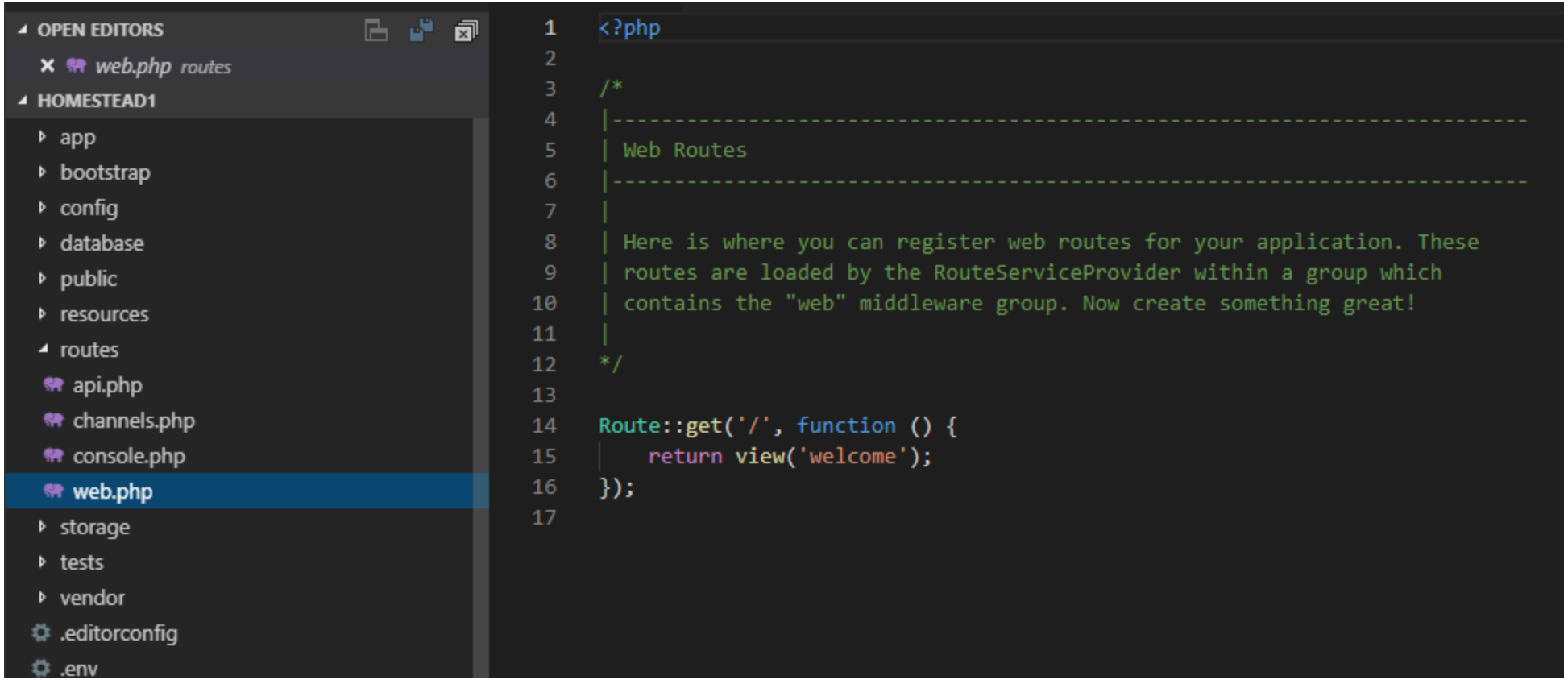


For example, if you request a URI you are not authorized for your request will get as far as authentication then a 'Access Denied' response will be created and sent back to you.

…or it might pass the request on to a 'login' controller which gives a 'please login' type response.

Laravel Routing

- The key method for a browser to communicate with your application is through routes
- Each path we want to exist in our web application can be thought of as a route. We have to tell Laravel what to do for each route.
- These are defined in the routes folder of a Laravel project



The PHP class which has all the route related methods

The HTTP verb for this route

The function to be called when a user navigates to this route

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

The route or path

```php
Route::get('/home', function () {
    return "This is the /home page";
});

Route::get('/blog', function () {
    return "This is the /blog page";
});
```

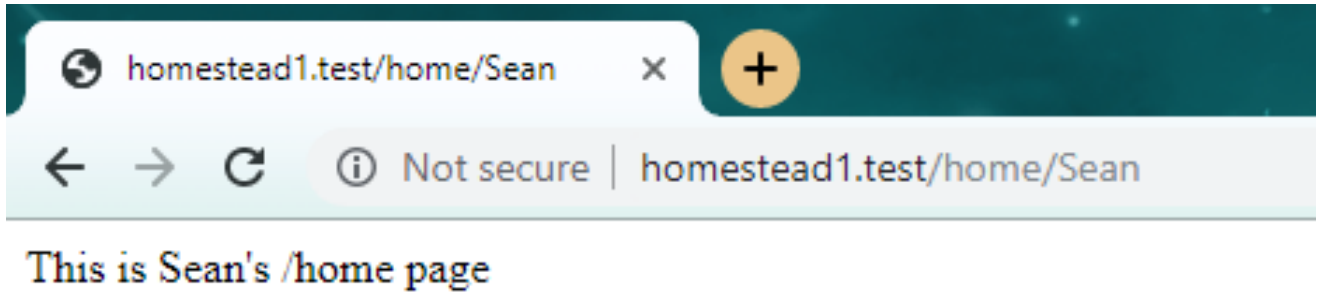
Redirect Routes

If you are defining a route that redirects to another URI, you may use the `Route::redirect` method. This method provides a convenient shortcut so that you do not have to define a full route or controller for performing a simple redirect:

```
Route::redirect('/here', '/there');
```

We can create routes which accept parameters as well…

```
Route::get('/home/{name}', function ($name) {
    return "This is $name's /home page";
});
```



These parameters are required, to make them optional we need to do more work…

Optional Parameters

Occasionally you may need to specify a route parameter, but make the presence of that route parameter optional. You may do so by placing a `?` mark after the parameter name. Make sure to give the route's corresponding variable a default value:

```
Route::get('user/{name?}', function ($name = null) {
    return $name;
});

Route::get('user/{name?}', function ($name = 'John') {
    return $name;
});
```

We can pass data (e.g., from the URL segment) to views:

```php
Route::get('/enclosures/{animal}', function ($animal) {
    return view('enclosure', ['animal' => $animal]);
});
```

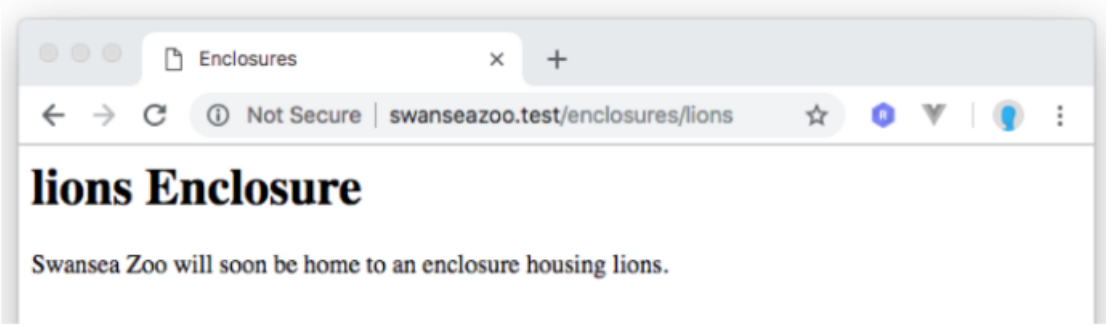
We build an associative array and passed it to the view.

enclosure.blade.php

```html
<html>
<head>
  <title>Enclosures</title>
</head>
<body>
  <h1>{{$animal}} Enclosure</h1>
  <p>Swansea Zoo will soon be home to an enclosure housing {{$animal}}.</p>
</body>
</html>
```

Blade syntax for echoing out a variable within HTML.

the keys of the associative array become the variables in the view.



Artisan Command

- Laravel comes with a useful command line tool artisan

- This tool can do many things to help you build your web site. Run the following command to see them all:

```
artisan list
```

- One of the useful things it can do it list all the routes. Run this from within your website folder within the VM:

```
artisan route:list
```

```
vagrant@homestead:~/Laravel/swanseazoo$ artisan route:list
+--------+----------+--------------------+------+---------+--------------+
| Domain | Method   | URI                | Name | Action  | Middleware   |
+--------+----------+--------------------+------+---------+--------------+
|        | GET|HEAD | /                  |      | Closure | web          |
|        | GET|HEAD | animals            |      | Closure | web          |
|        | GET|HEAD | animals/{name}     |      | Closure | web          |
|        | GET|HEAD | api/user           |      | Closure | api,auth:api |
|        | GET|HEAD | enclosures/{animal} |      | Closure | web          |
|        | GET|HEAD | food               |      | Closure | web          |
+--------+----------+--------------------+------+---------+--------------+
```

Tasks

1. Get a development environment up and running (i.e. install all the things!)
2. Create a two different routes which return two different strings.
3. Make one of those routes redirect to the other.
4. Create a route which accepts a string parameter and outputs that string in the browser.
5. Use artisan to display all your routes