Department of Information Technology
IT 350 - Web Technologies
Fall - 2021

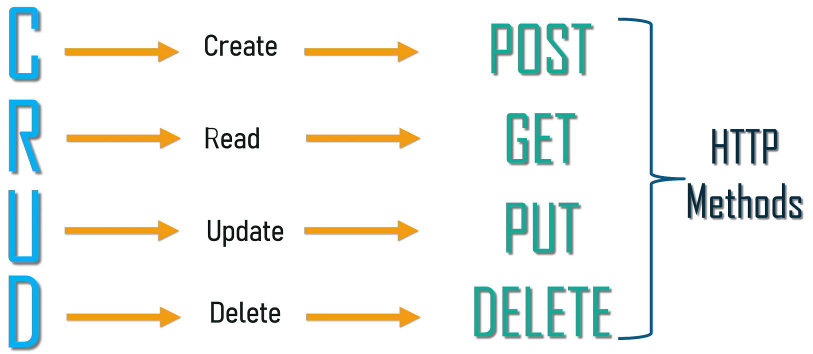
Routing and Controllers

Lecture Four

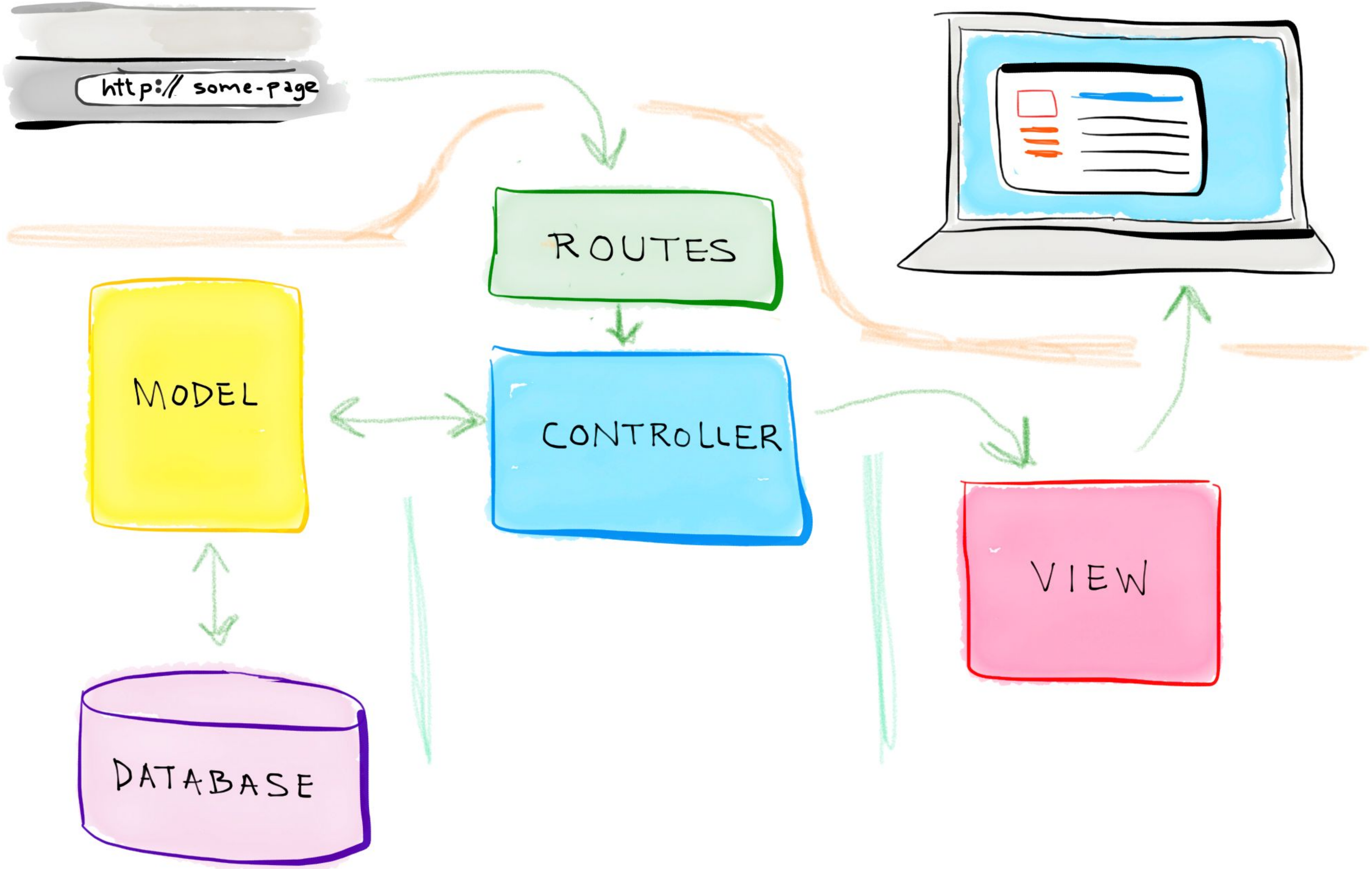
Rebin M. Ahmed
rebin.mohammed@tiu.edu.iq

Previous Lecture

- Pass Request Data to Views
- What Is MVC?
- HTTP Verbs
- REST
- Controllers



Contents


http:// some-page
ROUTES
MODEL
CONTROLLER
VIEW
DATABASE

Routing

In a Laravel application, you will define your web routes in routes/web.php and your API routes in routes/api.php.

Web routes are those that will be visited by your end users; API routes are those for your API, if you have one.

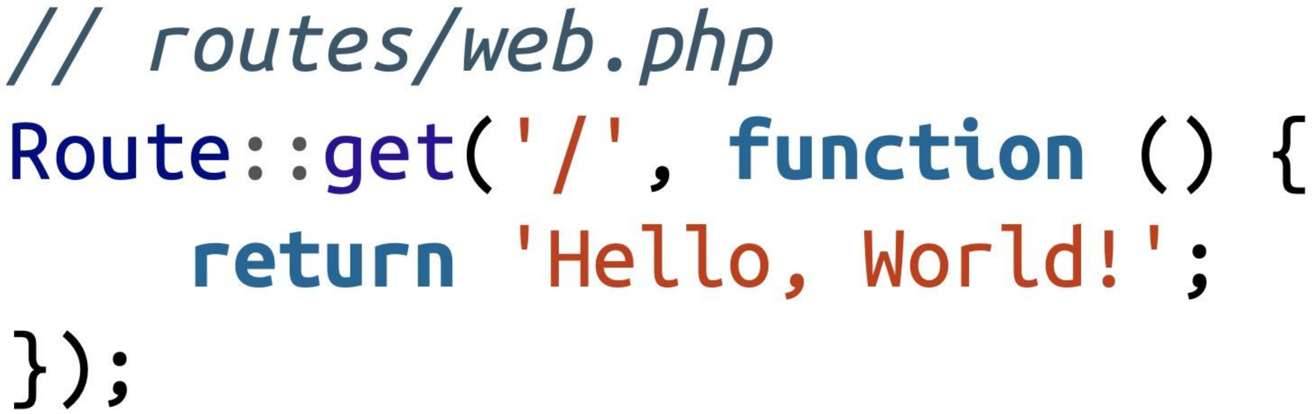
For now, we'll primarily focus on the routes in routes/web.php.

Routes File Location in Laravel Prior to 5.3

In projects running versions of Laravel prior to 5.3, there will be only one routes file, located at *app/Http/routes.php*.

Routing

The simplest way to define a route is to match a path (e.g., /) with a closure, as seen in this example

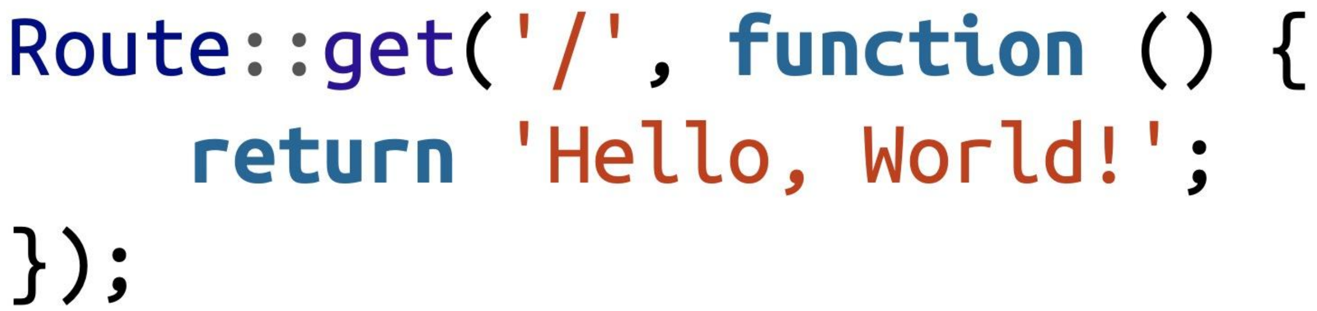


```php
// routes/web.php
Route::get('/', function () {
    return 'Hello, World!';
});
```

What’s a Closure?

A closure is a function that you can pass around as an object, assign to a variable, pass as a parameter to other functions and methods.

Routing

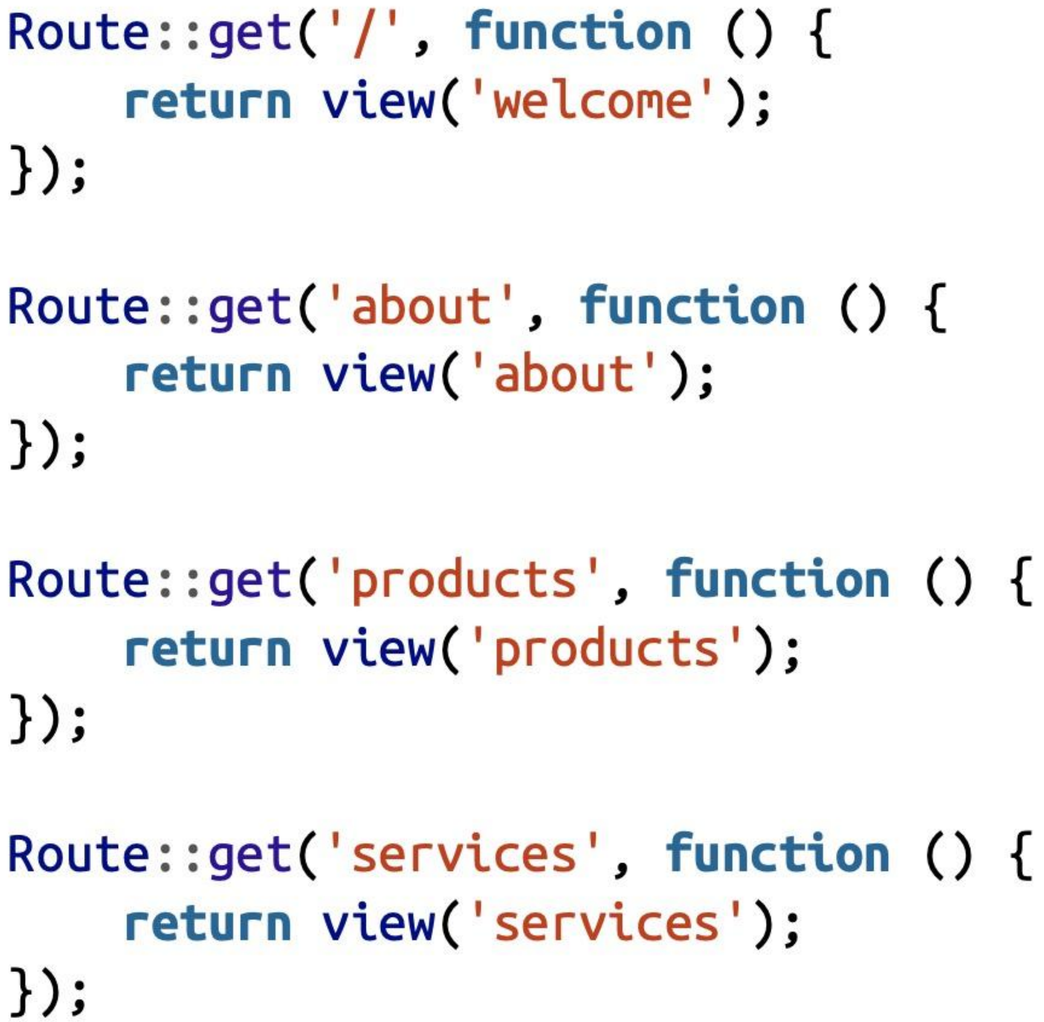


```
Route::get('/', function () {
    return 'Hello, World!';
});
```

You've now defined that if anyone visits / (the root of your domain), Laravel's router should run the closure defined there and return the result. Note that we return our content and don't echo or print it.

Routing

Many simple websites could be defined entirely within the web routes file. With a few simple GET routes combined with some templates.



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

Route::get('about', function () {
    return view('about');
});

Route::get('products', function () {
    return view('products');
});

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

Route Verbs

You might've noticed that we've been using Route::get() in our route definitions.

There are a few other options for methods to call on a route definition, as illustrated in this example.

```
Route::get('/', function () {
    return 'Hello, World!';
});

Route::post('/', function () {
    // Handle someone sending a POST request to this route
});

Route::put('/', function () {
    // Handle someone sending a PUT request to this route
});

Route::delete('/', function () {
    // Handle someone sending a DELETE request to this route
});

Route::any('/', function () {
    // Handle any verb request to this route
});

Route::match(['get', 'post'], '/', function () {
    // Handle GET or POST requests to this route
});
```

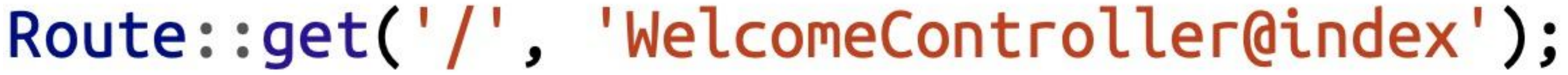
Route Handling

Passing a closure to the route definition is not the only way handle a route, Closures are quick and simple, but the larger your application gets, the clumsier it becomes to put all of your routing logic in one file.

Additionally, applications using route closures can’t take advantage of Laravel’s route caching.

The other common option is to pass a controller name and method as a string in place of the closure.

Route Handling



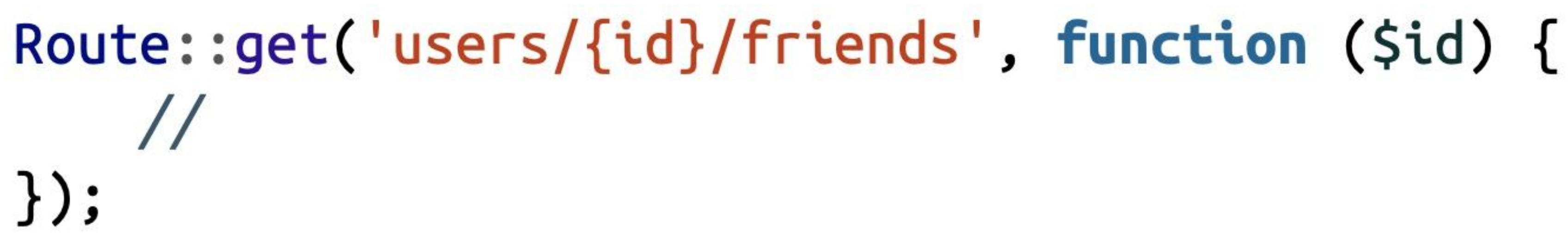
```
Route::get('/', 'WelcomeController@index');
```

This is telling Laravel to pass requests to that path to the **index()** method of the **App\Http\Controllers\WelcomeController** controller.

This method will be passed the same parameters and treated the same way as a closure you might’ve alternatively put in its place.

Route Parameters

If the route you’re defining has parameters—segments in the URL structure that are variable—it’s simple to define them in your route and pass them to your closure

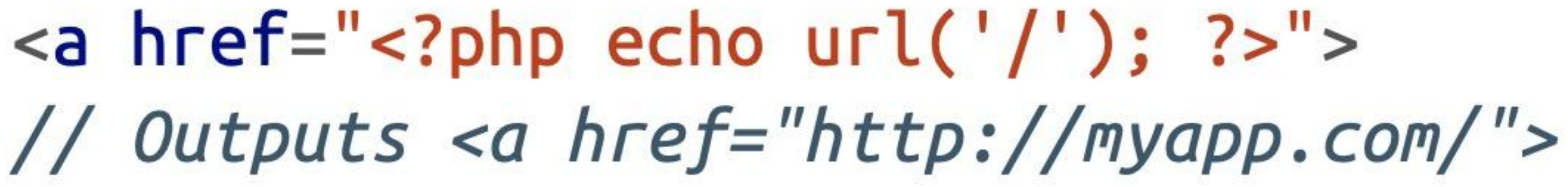


```php
Route::get('users/{id}/friends', function ($id) {
    //
});
```

You can also make your route parameters optional by including a question mark (?) after the parameter name

Route Names

The simplest way to refer to these routes elsewhere in your application is just by their path. There’s a url() global helper to simplify that linking in your views, if you need it:

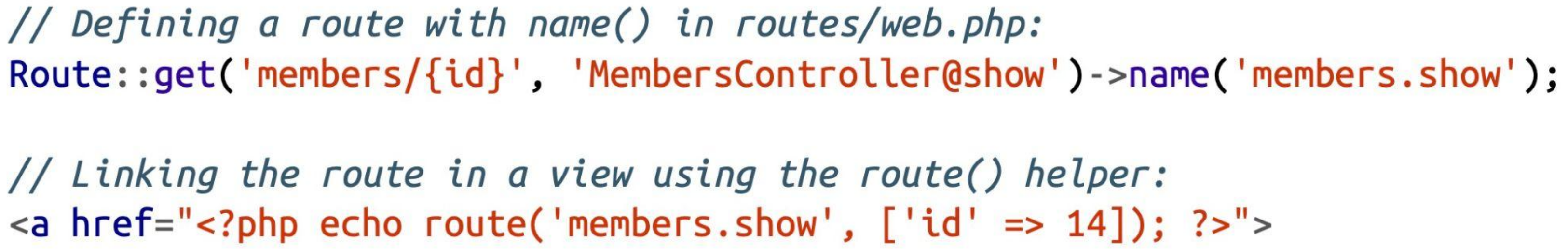


```
<a href="<?php echo url('/'); ?>">
// Outputs <a href="http://myapp.com/">
```

Route Names

However, Laravel also allows you to name each route, which enables you to refer to it without explicitly referencing the URL.

This is helpful because it means you can give simple nicknames to complex routes, and also because linking them by name means you don’t have to rewrite your frontend links if the paths change.



```
// Defining a route with name() in routes/web.php:
Route::get('members/{id}', 'MembersController@show')->name('members.show');

// Linking the route in a view using the route() helper:
<a href="<?php echo route('members.show', ['id' => 14]); ?>">
```

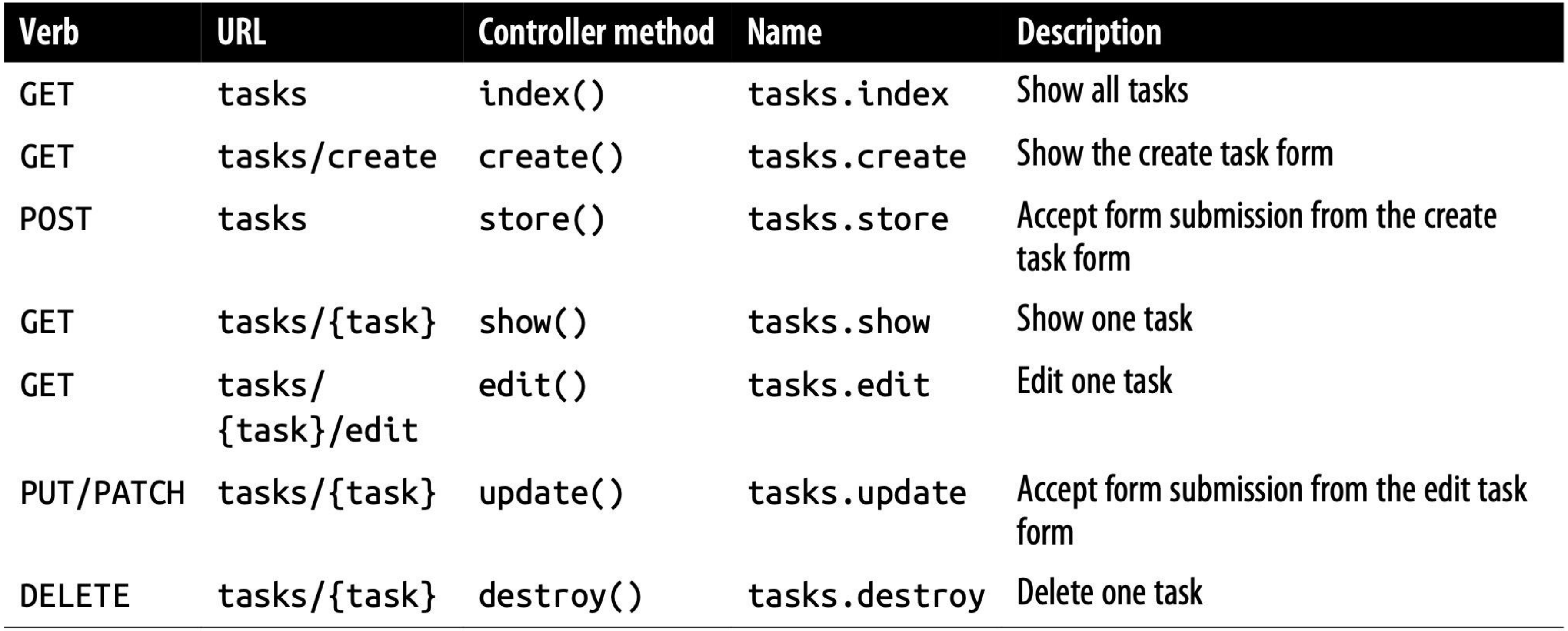
Break Time!

Route Naming Conventions

You can name your route anything you'd like, but the common convention is to use the plural of the resource name, then a period, then the action. So, here are the routes most common for a resource named photo:

photos.index
photos.create
photos.store
photos.show
photos.edit
photos.update
photos.destroy

Route Naming Conventions



Verb	URL	Controller method	Name	Description
GET	tasks	index()	tasks.index	Show all tasks
GET	tasks/create	create()	tasks.create	Show the create task form
POST	tasks	store()	tasks.store	Accept form submission from the create task form
GET	tasks/{task}	show()	tasks.show	Show one task
GET	tasks/{task}/edit	edit()	tasks.edit	Edit one task
PUT/PATCH	tasks/{task}	update()	tasks.update	Accept form submission from the edit task form
DELETE	tasks/{task}	destroy()	tasks.destroy	Delete one task

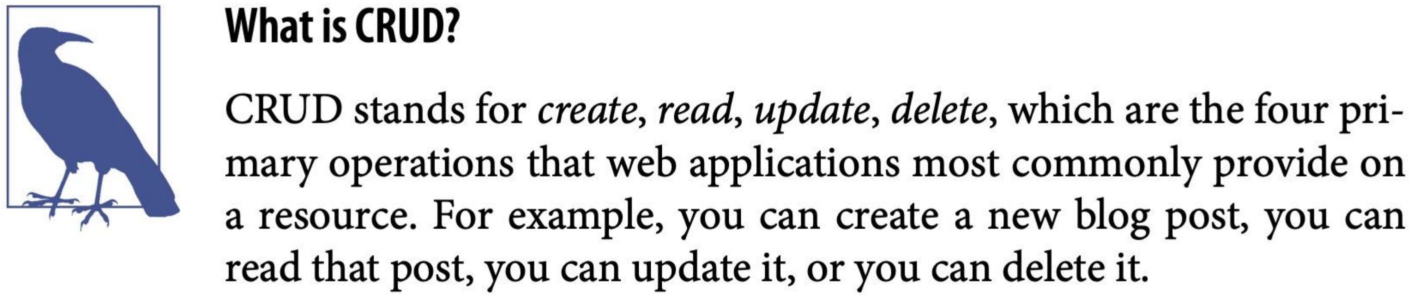
HTTP Verbs

Verb	Description
GET	Request a resource (or a list of resources).
HEAD	Ask for a headers-only version of the GET response.
POST	Create a resource.
PUT	Overwrite a resource.
PATCH	Modify a resource.
DELETE	Delete a resource.
OPTIONS	*Ask the server which verbs are allowed at this URL.*

Controllers

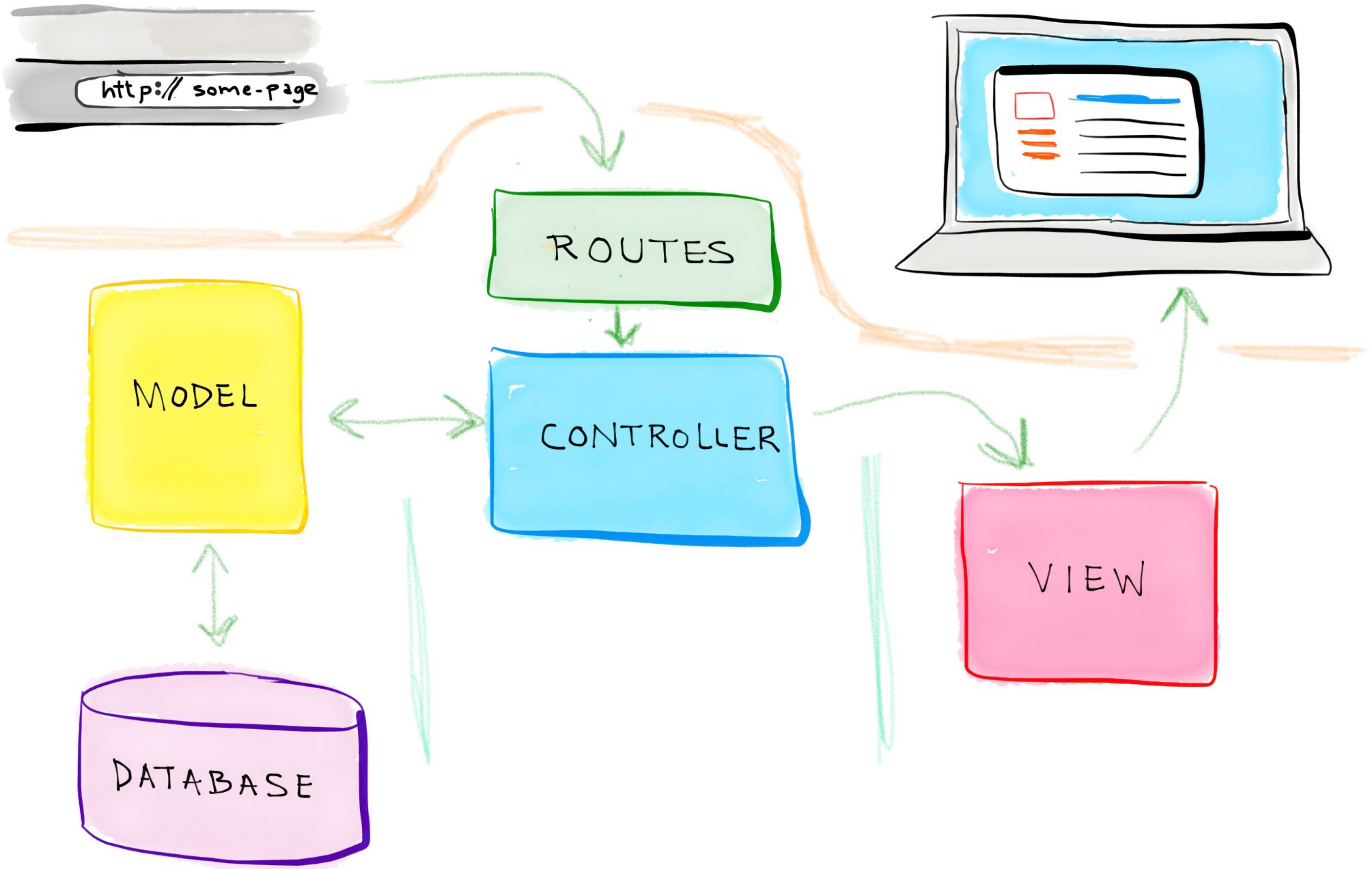
Controllers are essentially classes that organize the logic of one or more routes together in one place.

Controllers tend to group similar routes together, especially if your application is structured in a traditionally **CRUD**-like format; in this case, a controller might handle all the actions that can be performed on a particular resource.



What is CRUD?

CRUD stands for *create*, *read*, *update*, *delete*, which are the four primary operations that web applications most commonly provide on a resource. For example, you can create a new blog post, you can read that post, you can update it, or you can delete it.


http:// some-page
ROUTES
MODEL
CONTROLLER
VIEW
DATABASE

Controllers

let’s create a controller. One easy way to do this is with an Artisan command, so from the **command line** run the following:

```
php artisan make:controller PostController
```

This will create a new file named PostController.php in app/Http/Controllers.

And this is how you can link it to a route:

```
Route::get('/posts', 'PostController@index');
```

Resource Controllers

Laravel resource routing assigns the typical "CRUD" routes to a controller with a single line of code. For example, you may wish to create a controller that handles all HTTP requests for "photos" stored by your application. Using the make:controller Artisan command, we can quickly create such a controller:

```
php artisan make:controller PostController --resource
```

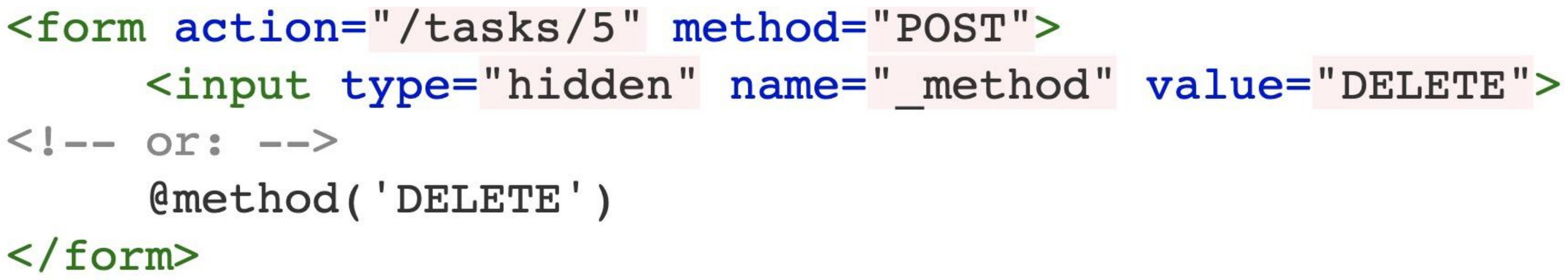
This command will generate a controller at app/Http/Controllers/PostController.php. The controller will contain a method for each of the available resource operations.

Resource Controllers

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

HTTP Method Spoofing in HTML Forms

Since HTML forms can't make PUT, PATCH, or DELETE requests, you will need to add a hidden _method field to spoof these HTTP verbs. The @method Blade directive can create this field for you:



```
<form action="/tasks/5" method="POST">
    <input type="hidden" name="_method" value="DELETE">
<!-- or: -->
    @method('DELETE')
</form>
```

CSRF Protection

If you’ve tried to submit a form in a Laravel application already, including the one in, you’ve likely run into the dreaded TokenMismatchException.

By default, all routes in Laravel except “read-only” routes (those using GET, HEAD, or OPTIONS) are protected against **cross-site request forgery (CSRF)** attacks by requiring a token, in the form of an input named _token, to be passed along with each request.

What is CSRF?

What is CSRF?

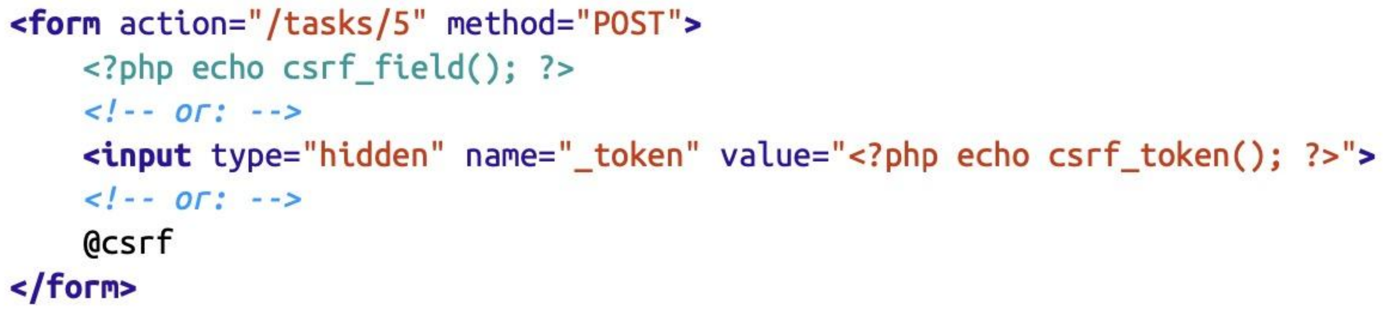
A cross-site request forgery is when one website pretends to be another. The goal is for someone to hijack your users' access to your website, by submitting forms from their website to your web- site via the logged-in user's browser.

The best way around CSRF attacks is to protect all inbound routes —POST, DELETE, etc.—with a token, which Laravel does out of the box.

CSRF Protection

ou have two options for getting around this CSRF error.

The first, and preferred, method is to add the _token input to each of your submissions. In HTML forms, that’s simple;

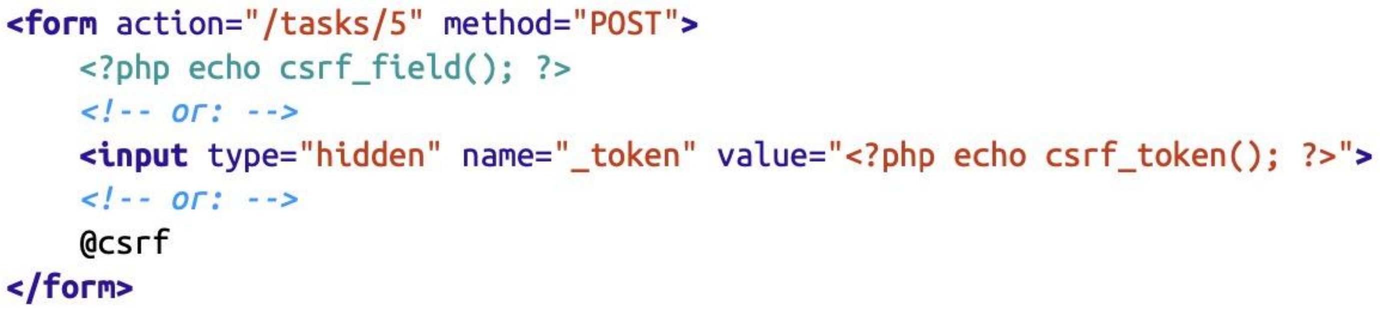


```
<form action="/tasks/5" method="POST">
    <?php echo csrf_field(); ?>
    <!-- or: -->
    <input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
    <!-- or: -->
    @csrf
</form>
```

CSRF Protection

ou have two options for getting around this CSRF error.

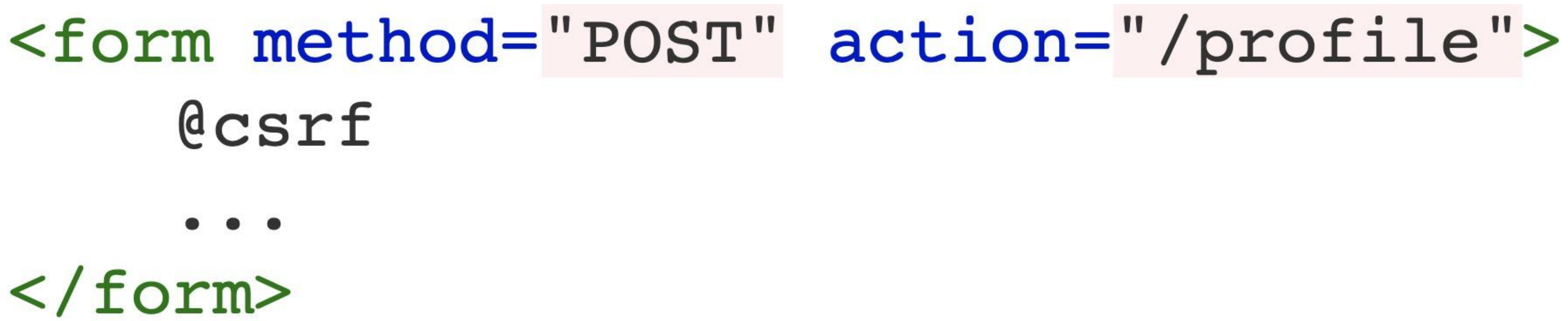
The first, and preferred, method is to add the _token input to each of your submissions. In HTML forms, that’s simple;



```
<form action="/tasks/5" method="POST">
    <?php echo csrf_field(); ?>
    <!-- or: -->
    <input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
    <!-- or: -->
    @csrf
</form>
```

CSRF Protection

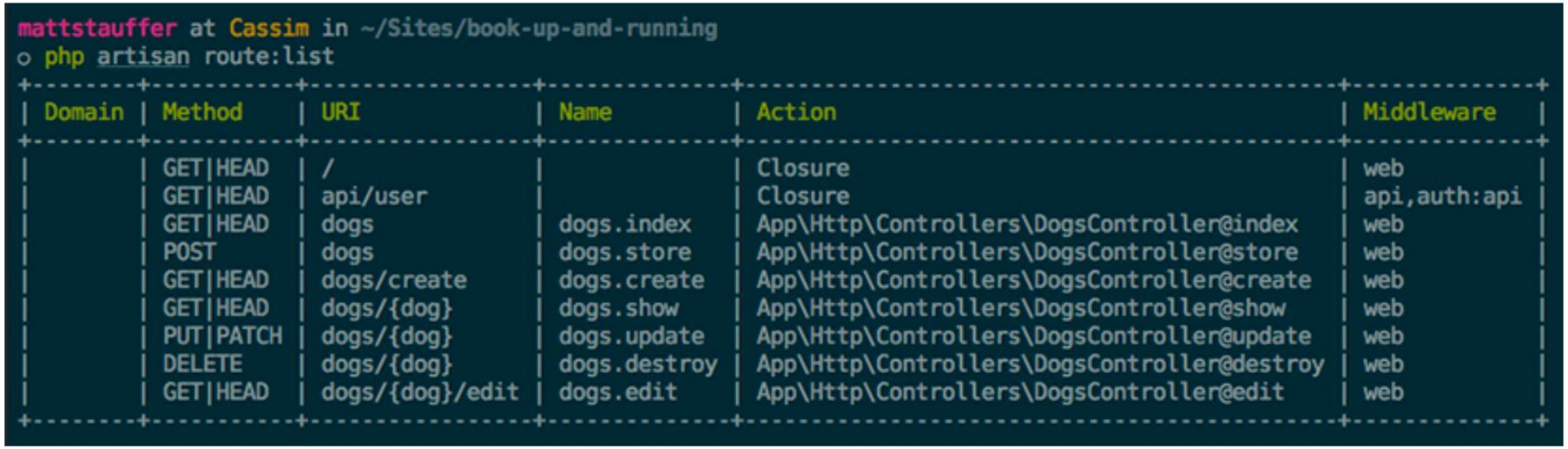
Preferred, method is to add the _token input to each of your submissions. In HTML forms, that’s simple;



```
<form method="POST" action="/profile">
    @csrf
    ...
</form>
```

Route:list

If you ever find yourself in a situation where you’re wondering what routes your current application has available, there’s a tool for that: from the command line, run php artisan route:list and you’ll get a listing of all of the available routes



```
mattstauffer at Cassim in ~/Sites/book-up-and-running
o php artisan route:list
+--------+-----------+------------------+-----------------+-------------------------------------------------+----------------+
| Domain | Method    | URI              | Name            | Action                                          | Middleware     |
+--------+-----------+------------------+-----------------+-------------------------------------------------+----------------+
|        | GET|HEAD  | /                |                 | Closure                                         | web            |
|        | GET|HEAD  | api/user         |                 | Closure                                         | api,auth:api   |
|        | GET|HEAD  | dogs             | dogs.index      | App\Http\Controllers\DogsController@index       | web            |
|        | POST      | dogs             | dogs.store      | App\Http\Controllers\DogsController@store       | web            |
|        | GET|HEAD  | dogs/create      | dogs.create     | App\Http\Controllers\DogsController@create      | web            |
|        | GET|HEAD  | dogs/{dog}       | dogs.show       | App\Http\Controllers\DogsController@show        | web            |
|        | PUT|PATCH | dogs/{dog}       | dogs.update     | App\Http\Controllers\DogsController@update      | web            |
|        | DELETE    | dogs/{dog}       | dogs.destroy    | App\Http\Controllers\DogsController@destroy     | web            |
|        | GET|HEAD  | dogs/{dog}/edit  | dogs.edit       | App\Http\Controllers\DogsController@edit        | web            |
+--------+-----------+------------------+-----------------+-------------------------------------------------+----------------+
```

For more details on topics of this lecture:
Read Chapter 03

Activities and Next Week Topics

This Week:

- Read Chapter the rest of 03 of Laravel: Up & Running, for more information Routing and Controllers.
- Practice different route and controller types and passing data through routes and requests.

Next Week:

- Databases and Eloquent: Configuration and Connection, Migrations.

References / Further Readings

- Laravel.com : Laravel's official Documentation.
- Matt Stauffer, 2019. Laravel: Up & Running: A Framework for Building Modern PHP Apps. O'Reilly Media.
- Dayle Rees, 2016. Laravel: Code Smart.