



Laravel



Faculty of Applied Science
Information Technology
2023-24 Fall Semester

Web Technologies
03 - Passing Data, REST and Routing

Previous Lecture

- System Requirements
- Creating a New Laravel Project
- Laravel's Directory Structure
- Configurations
- Architecture Concepts
- Request Lifecycle

Content

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

Pass Request Data to Views

- How to pass data in request.
- How to send the data to the Views.
- How to show the data in Views with Blade.

Pass Request Data to Views

```
//Showing a view
Route::get('/form',function (){
    return view('form');
});

//Response to the form
Route::get('/response',function (){
    return request()->fullname;
});

//Getting a variable from the URL
Route::get('/news/{id}',function ($id){
    return $id;
});

//Getting a variable from URL manually
Route::get('/news',function ($id){
    return request()->all();
});
```

Passing Data To Views

you may pass an array of data to views to make that data available to the view:

```
return view('greetings', ['name' => 'Victoria']);
```

When passing information in this manner, the data should be an array with key / value pairs. After providing data to a view, you can then access each value within your view using the data's keys, such as

```
<?php echo $name; ?>
```

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:

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


A decorative graphic on the left side of the slide, consisting of two overlapping green arrow shapes pointing to the right. The top arrow is a lighter shade of green, and the bottom arrow is a darker shade, creating a layered effect.

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(uri: '/', [WelcomeController::class, '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:

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

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

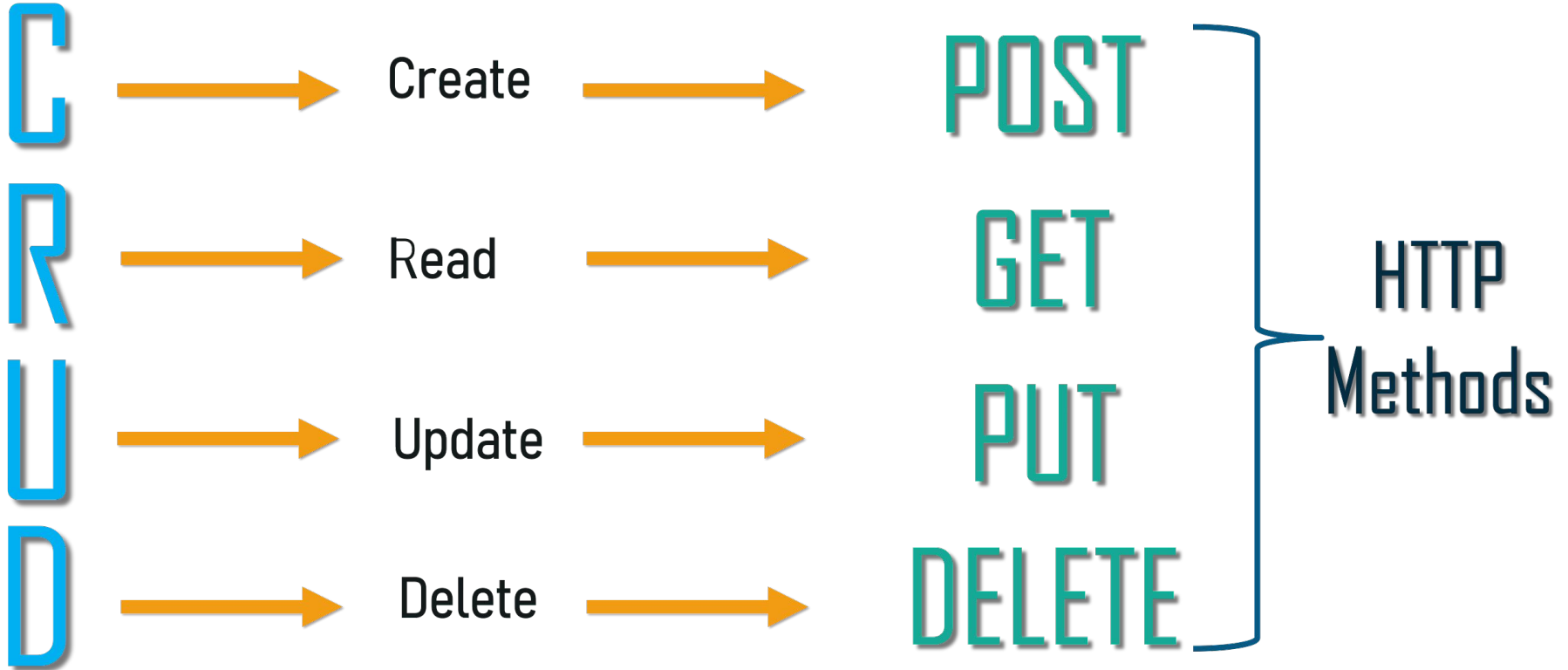
HTTP Verbs

The most common HTTP verbs are **GET** and **POST**, followed by **PUT** and **DELETE**. There are also **HEAD**, **OPTIONS**, and **PATCH**, and two others that are pretty much never used in normal web development, **TRACE** and **CONNECT**.

REST

REST is acronym for **RE**presentational **S**tate **T**ransfer. REST is a software architectural style that defines a set of constraints to be used for creating Web services, Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the internet.

REST



REST

There's more to it, but usually “**RESTful**” as it'll be used in this course will mean “patterned after these URL-based structures so we can make predictable calls like **GET /tasks/14/edit** for the edit page.”

This is relevant (even when not building APIs) because Laravel's routing structures are based around a REST-like structure.

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 web.php
```

```
Route::get(uri: '/', [WelcomeController::class, 'index'])->name(name: 'welcome');
```

```
Route::get(uri: '/member/{id}', [MemberController::class, 'show'])->name(name: 'member.show');
```

```
// Linking the route in a view using the route() helper:
```

```
<a href="<?php echo route('members.show', ['id' => 14]); ?>">
```

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

Break Time!



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

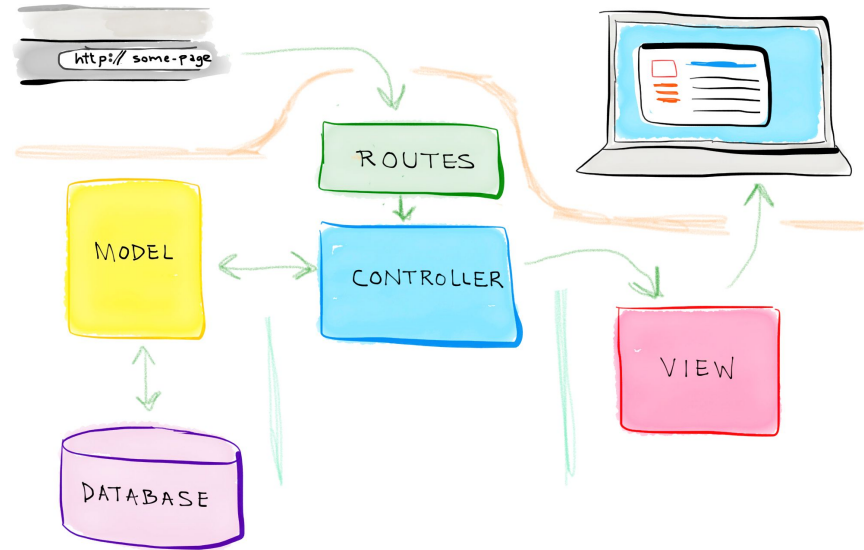
What Is MVC?

In MVC, we have three primary concepts:

- Model : Represents an individual database table (or a record from that table)—think “Company” or “Student.”
- View : Represents the template that outputs your data to the end user—think “the login page template with this given set of HTML and CSS and JavaScript.”

What Is MVC?

- Controller: Like a traffic cop, takes HTTP requests from the browser, gets the right data out of the database and other storage mechanisms, validates user input, and eventually sends a response back to the user.



What Is MVC?

In [Figure 3-1](#), you can see that the end user will

- First interact with the controller by sending an HTTP request using their browser.
- The controller, in response to that request, may write data to and/or pull data from the model (database).
- The controller will then likely send data to a view, and then the view will be returned to the end user to display in their browser.



**For more details on REST
Read Chapter 13**

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.

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::class,'index']);
```



Artisan and Artisan Generators

Laravel comes bundled with a command-line tool called Artisan. Artisan can be used to run migrations, create users and other database records manually, and perform many other manual, one-time tasks.

Under the `make` namespace, Artisan provides tools for generating skeleton files for a variety of system files. That's what allows us to run `php artisan make:controller`.

To learn more about this and other Artisan features, see [Chapter 8](#).

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	<code>/posts</code>	index	posts.index
GET	<code>/posts/create</code>	create	posts.create
POST	<code>/posts</code>	store	posts.store
GET	<code>/posts/{post}</code>	show	posts.show
GET	<code>/posts/{post}/edit</code>	edit	posts.edit
PUT/PATCH	<code>/posts/{post}</code>	update	posts.update
DELETE	<code>/posts/{post}</code>	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

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

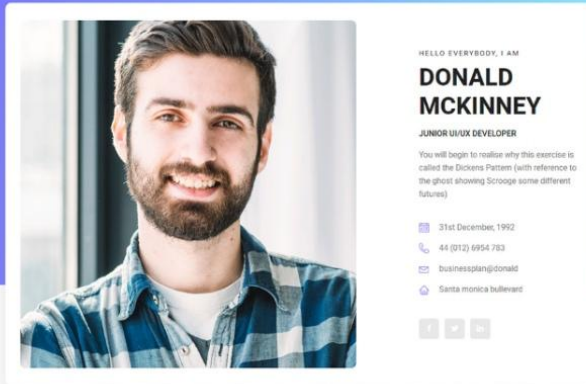
If you ever find yourself in a situation where you're wondering what routes your current application has available, there's a tool for that: from the command line, run **php artisan route:list** and you'll get a listing of all of the available routes

```
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 03 of Laravel: Up & Running, for more information on MVC, HTTP Verbs and REST.
- Practice different route and passing data through routes.
- Create a form and Response page to show a simple profile, for the submitted data through the form
- Prepare your questions for the practical session in the lab.

Next Week:

- Routing and Controllers.

References / Further Readings

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