KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
POLITEKNIK NEGERI MALANG
JURUSAN TEKNOLOGI INFORMASI
Jl. Soekarno Hatta No. 9, Jatimulyo, Lowokwaru, Malang 65141
Telp. (0341) 404424 – 404425, Fax (0341) 404420
http://www.polinema.ac.id

Courses : Advanced Web Programming
Studina Program : D4 – Informatics Engineering / D4 – Business Information Systems
Semester : 4 (four) / 6 (six)
Meeting to- : 2 (two)

JOBSHEET 02

ROUTING, CONTROLLER, DAN VIEW

1. MVC on Laravel

MVC stands for Model View Controller. Laravel uses the MVC model, therefore there are three core parts of the framework that work together: the model, view, and controller. The controller is the main part where most of the work is done. The controller connects to the Model to get, create, or update data and displays the result on the View, which contains the actual HTML structure of the application.

a. Model
   In Laravel, the Model class contains all the methods and attributes needed to interact with the specified database schema.
b. View
   A view represents how information is displayed, used for all software user interface logic. A view represents the User Interface (Frontend) of a web page.
c. Controller
   The controller acts as an intermediary between the Model and the View, processing all input sent by the user from the View. The controller processes all the business logic, manipulates the data using the Model component, and interacts with the View to render the final output.

2. Routing

In Laravel there is a feature called *route*. This route is used as a connector between the user and the application. In other words, the URL we write in the browser will pass through the route. And on the route will be determined where to go next, can be to the Controller or to the View.

Routing itself is the process of sending data and information to users through a request made to an address that has been registered, then the address will process from our request earlier. After the process is complete, it will return an output or result of the process.

To create a route used **Facade Route** followed by a verb which is an **HTTP verb,** generally consisting of **get, post, put, delete, options, patch**. In addition, it takes a **path** that is a URL after the domain name of the application accessed by the user. And at the end there is a callback **which can be a** callback function **or** controller action **that executes logic when the path is accessed by the user.**

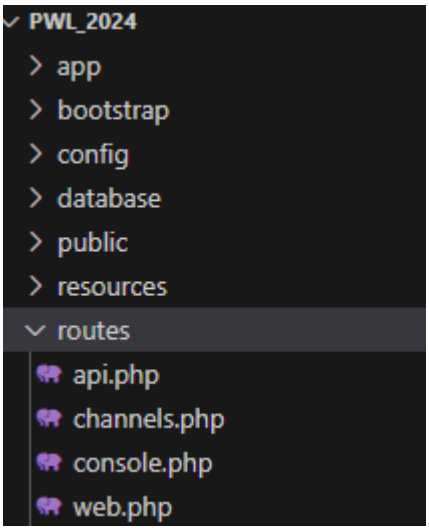
Here's a simple example of writing a route in Laravel 10.

```
use Illuminate\Support\Facades\Route;

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

The above route can be translated when the user accessing the URL in **/hello** will execute a callback function that displays the message 'Hello World'.

However, the use of callback functions is rarely used in making actual applications, because for complex logic makes the code difficult to maintain. As a solution introduced the concept of Controller. If the above route is converted to controller it becomes as follows:

```
use Illuminate\Support\Facades\Route;

Route::get('/hello', [WelcomeController::class,'hello']);
```



Inside the Laravel project, there is a **routes** folder. In general, laravel divides into four places, namely:

a. `routes/web.php` used for standard web
b. `routes/api.php` used for web services/APIs
c. `routes/console.php` used for command line

d. `routes/channel.php` used to broadcast channels over websockets

In general, applications are made simply with `routes / web.php` and `routes / api.php`. Even if the application doesn't need to provide APIs, it only uses `routes/web.php`.

In Laravel we can use all http verbs to be installed as the router method you want to use, it has been explained earlier that all http verbs can be served with the Router on Laravel. Endpoint / router url should follow the following best practices where a resource can be served with different functions on each http verb.

Resource	POST	GET	PUT	DELETE
/mahasiswa	Create a student record	Taking a student List	Update a lot of student data	Delete a lot of student data
/mahasiswa/{id}	Error	Show One Student Data	Update student data if there is data with id sent	Delete one student data

Please note that laravel can support a single route that has more than one http verb or has all http verbs. Here is the routing program code for the table above

```
Route::get('mahasiswa', function ($id) {
});
Route::post('mahasiswa', function ($id) {
});
Route::put('mahasiswa', function ($id) {
});
Route::delete('mahasiswa', function($id) {
});
Route::get('mahasiswa/{id}', function ($id) {
});
Route::put('mahasiswa/{id}', function ($id) {
});
Route::delete('mahasiswa/{id}', function ($id) {
});
```

To check and validate whether the route created is correct by using the following command

```
php artisan route:list
```

The output of the command if the routing you created is correct will be like this:

```
+--------+----------+----------------+------+---------+------------+
| Domain | Method   | URI            | Name | Action  | Middleware |
+--------+----------+----------------+------+---------+------------+
|        | GET|HEAD | /              |      | Closure | web        |
|        | GET|HEAD | api/user       |      | Closure | api        |
|        |          |                |      |         | auth:api   |
|        | GET|HEAD | mahasiswa      |      | Closure | web        |
|        | POST     | mahasiswa      |      | Closure | web        |
|        | PUT      | mahasiswa      |      | Closure | web        |
|        | DELETE   | mahasiswa      |      | Closure | web        |
|        | GET|HEAD | mahasiswa/{id} |      | Closure | web        |
|        | PUT      | mahasiswa/{id} |      | Closure | web        |
|        | DELETE   | mahasiswa/{id} |      | Closure | web        |
+--------+----------+----------------+------+---------+------------+
```

If you need a route that can have more than one http routing method can be created this way.

```
Route::match(['get', 'post'], '/specialUrl', function () {
});

Route::any('/specialMahasiswa', function ($id) {
});
```

```
+--------+-----------------------------------------+-----------------+------+---------+------------+
| Domain | Method                                  | URI             | Name | Action  | Middleware |
+--------+-----------------------------------------+-----------------+------+---------+------------+
|        | GET|HEAD                                | /               |      | Closure | web        |
|        | GET|HEAD                                | api/user        |      | Closure | api        |
|        |                                         |                 |      |         | auth:api   |
|        | GET|HEAD|POST|PUT|PATCH|DELETE|OPTIONS | specialMahasiswa |      | Closure | web        |
|        | GET|POST|HEAD                           | specialUrl      |      | Closure | web        |
+--------+-----------------------------------------+-----------------+------+---------+------------+
```

- **Basic Routing**

  Basically, Routing in Laravel requires information about http verbs, then input in the form of urls and what to do when receiving the url. To create a route you can use a callback function or use a controller.

Practicum Steps:

a. In this section, we will create two routes with the following conditions.

No	Http Verb	Url	Function
1	get	/hello	Display String Hello to the browser.
2	get	/world	Show String World to browser

We'll use the previous week's project, which is PWL_2024.

b. Open the `routes/web.php` file. Add a route for number 1 as below:

```
use Illuminate\Support\Facades\Route;

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

c. Open a browser, write the URL to call the route: localhost/PWL_2024/public/hello. Pay attention to the page that appears to see if it is appropriate and describe your observations.

d. To create a second route, add the /world route as below:

```
use Illuminate\Support\Facades\Route;

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

e. Open your browser, write the URL to call the route: localhost/PWL_2024/public/world. Pay attention to the page that appears to see if it is appropriate and describe your observations.

f. Next, try creating a '/' route that displays a 'Welcome' message.

g. Then create a '/about' route that will display your NIM and name.

- **Route Parameters**

Sometimes when creating a URL, we need to retrieve a parameter that is part of the URL segment in our route. For example, we need a username sent via a URL.

Practicum Steps:

To create routing with parameters can be done in the following way.

a. We will call route /user/*{name}* as well as send parameters in the form of user names $name as the code below.

```
Route::get('/user/{name}', function ($name) {
    return 'My name is '.$name;
});
```

b. Run the code by writing the URL to call the route: **localhost/PWL_2024/public/user/yourname**. Pay attention to the page that appears and describe your observations.
c. Next, try writing the URL: **localhost/PWL_2024/public/user/**. Pay attention to the page that appears and describe your observations

d. A route, can also accept more than 1 parameter as the following code. The route accepts $postId parameters and also $comment.

```
Route::get('/posts/{post}/comments/{comment}', function
($postId, $commentId) {
    return 'Pos ke-'.$postId." Komentar ke-: ".$commentId;
});
```

e. Run the code by writing the URL to call the route: **localhost/PWL_2024/public/posts/1/comments/5**. Pay attention to the page that appears and describe your observations.
f. Then create a /articles/{id} route that will display the output "Article Page with ID {id}", replace id according to the input of url.

- **Optional Parameters**

We can specify the value of the route parameter, but make the value of the route parameter optional. Make sure to provide the appropriate variable on route as the default value. Optional parameters are marked '?'.

Practicum Steps:

To create routing with optional parameters can be done in the following way.

a. We will call route */user* as well as send a parameter in the form of a username $name where the parameter is optional.

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

b. Run the code by typing the URL: **localhost/PWL_2024/public/user/**. Pay attention to the page that appears and describe your observations.
c. Next write the URL: **localhost/PWL_2024/public/user/yourname**. Pay attention to the page that appears and describe your observations

d. Change the code in the /user route to as below.

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

e. Run the code by typing the URL: **localhost/PWL_2024/public/user/**. Pay attention to the page that appears and describe your observations.

- **Route Name**

Route names are usually used to make it easier for us to call routes when building applications. We simply call the name of the route.

```
Route::get('/user/profile', function () {
    //
})->name('profile');

Route::get(
    '/user/profile',
    [UserProfileController::class, 'show']
)->name('profile');

// Generating URLs...
$url = route('profile');

// Generating Redirects...
return redirect()->route('profile');
```

- **Route Group dan Route Prefixes**

Several routes that have the same attributes as the same middleware can be grouped into one group to make it easier to write routes, in addition to being used for middleware, there is still the use of route groups for routes that are under one subdomain. Examples of using route groups are as follows:

```
Route::middleware(['first', 'second'])->group(function () {
    Route::get('/', function () {
        // Uses first & second middleware...
    });

Route::get('/user/profile', function () {
        // Uses first & second middleware...
```

```
    });
});

Route::domain('{account}.example.com')->group(function () {
    Route::get('user/{id}', function ($account, $id) {
        //
    });
});

Route::middleware('auth')->group(function () {
     Route::get('/user', [UserController::class, 'index']);
     Route::get('/post', [PostController::class, 'index']);
     Route::get('/event', [EventController::class, 'index']);

});
```

Route Prefixes

Route grouping can also be done for routes that have the same prefix. For route creation with prefix can be seen the code as below

```
Route::prefix('admin')->group(function () {
     Route::get('/user', [UserController::class, 'index']);
     Route::get('/post', [PostController::class, 'index']);
     Route::get('/event', [EventController::class, 'index']);

});
```

- **Redirect Routes**

  To redirect Laravel can be done using Route::redirect how to use it can be seen in the program code below.

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

  This redirect will often be used in CRUD cases or other cases that require a redirect.

- **View Routes**

  Laravel also provides a special route that makes it easy to create a route without using controllers or callback functions. These routes directly accept input in the form of urls and return views. Here's how to create view routes.

```
Route::view('/welcome', 'welcome');
Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

In the view routes above /welcome will display the welcome view and in the second route /welcome will display the welcome view with additional data in the form of name variables.

Save the changes you've made to Git.

3. Controller

Controllers are used to organize application logic into more structured structures. Unrelated application action logic can be collected in a single Controller class. Or a Controller can contain only one action. Controllers in Laravel are stored in the `app/Http/Controllers folder`.

- **Creating a Controller**

Practicum Steps:

a. To create a controller in Laravel a command is provided to generate its basic structure. We can use the artisan command followed by the definition of the name of the controller to be created.

```
php artisan make:controller WelcomeController
```

b. Open the file in `app/Http/Controllers/WelcomeController.php`. The structure on the controller can be described as follows:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class WelcomeController extends Controller
{
    //
}
```

c. To define an action, please add a function with public access. So that the controller above becomes as follows:

```
<?php
```

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class WelcomeController extends Controller
{
     public function hello() {
               return 'Hello World';
     }
}
```

d. Once a controller has defined an action, we can add that controller to the route. Change the /hello route to something like this:

```
Route::get('/hello', [WelcomeController::class,'hello']);
```

e. Open a browser, write the URL to call the route: localhost/PWL_2024/public/hello. Pay attention to the page that appears and describe your observations.

f. Modification of results in practicum point 2 (Routing) with the concept of controller. Move the execution logic into the controller with the name PageController.

Resource	POST	GET	PUT	DELETE
/		Show 'Welcome' Message `PageController : index`		
/about		Show Name and NIM `PageController : about`		
/articles/{id}		Display dynamic page 'Article Page with Id {id}' id replaced as per input from url `PageController : articles`		

g. Modify the previous implementation with the concept of Single Action Controller. So for the final result obtained there will be `HomeController`, `AboutController` and `ArticleController`. Modifications are also the routes used.

- **Resource Controller**

Especially for controllers that are connected to the Eloquent model and can be performed CRUD operations on the Eloquent model, we can create a controller of type

Resource Controller. By creating a resource controller, the controller is equipped with methods that support the CRUD process, and there is a route resource that holds routes for the controller.

Practicum Steps:

a. To create it is done by running the following command in the terminal.

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

This command generates a controller named PhotoController that contains the standard method methods for the CRUD process.

b. After the controller is successfully degenerated, a route must be created so that it can connect to the frontend. Add the following program code to the web.php file.

```
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class);
```

c. Run the list route check (php artisan route:list) the following route will be generated.

Domain	Method	URI	Name	Action	Middleware
	GET|HEAD	/		Closure	web
	GET|HEAD	api/user		Closure	api auth:api
	GET|HEAD	photos	photos.index	App\Http\Controllers\PhotoController@index	web
	POST	photos	photos.store	App\Http\Controllers\PhotoController@store	web
	GET|HEAD	photos/create	photos.create	App\Http\Controllers\PhotoController@create	web
	GET|HEAD	photos/{photo}	photos.show	App\Http\Controllers\PhotoController@show	web
	PUT|PATCH	photos/{photo}	photos.update	App\Http\Controllers\PhotoController@update	web
	DELETE	photos/{photo}	photos.destroy	App\Http\Controllers\PhotoController@destroy	web
	GET|HEAD	photos/{photo}/edit	photos.edit	App\Http\Controllers\PhotoController@edit	web
	GET|HEAD|POST|PUT|PATCH|DELETE|OPTIONS	specialMahasiswa		Closure	web
	GET|POST|HEAD	specialUrl		Closure	web

d. In the route list all routes related to crud photos have been generated by laravel. If not all routes in the resource controller are needed, it can be reduced by updating the routes in the web.php to resemble the following.

```
Route::resource('photos', PhotoController::class)->only([
    'index', 'show'
]);

Route::resource('photos', PhotoController::class)->except([
    'create', 'store', 'update', 'destroy'
]);
```

Save the changes you've made to Git.

4. View

In the Laravel framework, View refers to the part of the web application that is responsible for displaying the user interface to the end user. A view is basically a template file used to generate HTML that will be displayed to the user.

Blade is Laravel's built-in templating engine. Useful for making it easier to write display code. It also provides additional features for manipulating data in views thrown from the controller. Blade also allows the use of plain PHP in View code. Because Laravel uses Blade's *built-in* templating engine, every View *file* ends with a .blade.php. Example: index.blade.php, home.blade.php, product.blade.php.

- Creating a View

Practicum Steps:

a. In the app/resources/views directory, create a hello.blade.php file.

```
<!-- View pada resources/views/hello.blade.php -->
<html>
     <body>
     <h1>Hello, {{ $name }}</h1>
     </body>
</html>
```

b. The View can be run via Routing, where the *route* will call the View according to the *file name* without the 'blade.php'. (Note: Replace Andi with your name)

```
Route::get('/greeting', function () {
     return view('hello', ['name' => 'Andi']);
});
```

c. Run the code by opening the url localhost/PWL_2024/public/greeting. Pay attention to the page that appears and describe your observations.

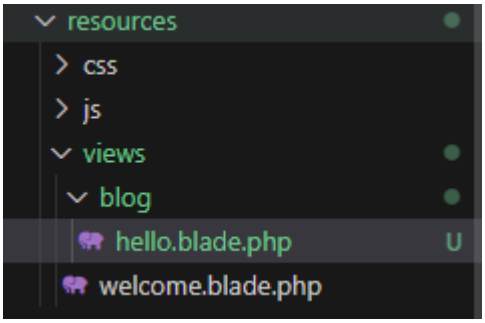
- View in directory

If in the `resources/views` directory there is another directory to store view *files*, for example hello.blade.php is in the blog directory, then we can use "dot" notation to reference the directory,

Practicum Steps:

a. Create a blog directory within the views directory.

b. Move the hello.blade.php file into the blog directory.



c. Next, make changes to the route.

```
Route::get('/greeting', function () {
    return view('blog.hello', ['name' => 'Andi']);
});
```

d. Run the code by opening the url localhost/PWL_2024/public/greeting. Pay attention to the page that appears and describe your observations.

- **Displaying a View from a Controller**

Views can be invoked via the Controller. So the Routing will call the Controller first, and the Controller will *return* the view in question.

Practicum Steps:

a. Open WelcomeController.php and add a new function, greeting.

```
class WelcomeController extends Controller
{
    public function hello(){
        return('Hello World');
    }

    public function greeting(){
        return view('blog.hello', ['name' => 'Andi']);
    }
}
```

b. Change the /greeting route and navigate to the WelcomeController in the greeting function.

```
Route::get('/greeting', [WelcomeController::class,
'greeting']);
```

c. Run the code by opening the url localhost/PWL_2024/public/greeting. Pay attention to the page that appears and describe your observations.

- **Forward data to view**

In the previous example, we can pass array data to the view to make that data available to the view:

```
return view('blog.hello', ['name' => 'Andi']);
```

When passing information in this way, the data must be an array with key/value pairs. After passing data to the view, we can then access each value in the view using a data key like: `<?php echo $name; ?>` or `{{$name}}`. As an alternative to passing a full data array to the view helper function, we can use the **with** method to add individual pieces of data to the view. The **with** method returns an instance of the view object so that we can continue the set of methods before returning the view notation to reference directories.

Practicum Steps:

a. Open WelcomeController.php and add change greeting function.

```
class WelcomeController extends Controller
{
    public function hello(){
        return('Hello World');
    }

    public function greeting(){
        return view('blog.hello')
            ->with('name','Andi')
            ->with('occupation','Astronaut');
    }
}
```

b. Change the hello.blade.php to display two parameters.

```
<html>
     <body>
     <h1>Hello, {{ $name }}</h1>
     <h1>You are {{ $occupation }}</h1>
     </body>
</html>
```

c. Run the code by opening the url localhost/PWL_2024/public/greeting. Pay attention to the page that appears and describe your observations.

Save the changes you've made to Git.

PRACTICUM QUESTIONS

1. Run the Practicum steps on the jobsheet above. Sync changes to PWL_2024 project to Github.
2. Create a new project named POS. This project is a Point of Sales application that is used to help sales.
3. Create several routes, controllers, and views according to the following conditions.

1	Home Page Display the start page of the website
2	Halaman Products Display product list (route prefix) /category/food-beverage /category/beauty-health /category/home-care /category/baby-kid
3	User Page Display user profiles (route param) /user/{id}/name/{name}
4	Sales Page Display the POS transaction page

4. The route performs functions on different controllers on each page.
5. The function on the Controller will call the view according to the page to be displayed.
6. Save any changes made to the POS project in Git, sync the changes to Github.

*Thank you, and good luck ****