Instruction for Lab4

The lab is based on the outcome of Lab3. The purpose of this Lab assignment is to extend the Laravel project from Lab3 with CRUD functionality.

The assignment can be completed using DDEV, Laravel Herd, Wamp.Net and other web servers. Please refer to Lab3 instructions to do the necessary development setup, create database migrations, models and data seeding.

1. Updates to the database seeder

Since at least one user is necessary to add job listings, update the database seeder to create a record in users table. It can be done, for example, using the following code:

```
User::create([
    'name' => 'Admin User',
    'email' => 'admin@example.com',
    'password' => bcrypt('adminpassword'),
]);
```

Run the database seeder to create a user record.

2. Controller for job listings

Using artisan, create a resource controller:

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

It creates a new file ListingController.php in *Laravel installation folder*\app\Http\Controllers.

Define routes to controller methods by adding the following code to the web.php file:

```
Route::resource('listing', ListingController::class);
```

Test that all 7 resource routes are registered by running a command:

```
php artisan route:list
```

3. View layout

3.1. Layout component

Now let's create a layout component to be used in all views of the application. Using artisan, create a layout component:

```
php artisan make:component layout --view
```

This command will create an anonymous component (without a class) in *Laravel installation folder*\resources\views\components\layout.blade.php. Edit this file to add layout code, for example, like the following (feel free to use another layout structure if you prefer it):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ $title }}</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
    <main class="container">
        {{ $slot }}
    </main>
</body>
</html>
```

3.2. Navigation bar

Create a navigation bar component. It must contain links to the index of all listings and a link to the listing creation form (see the following sections). Use `action` or `route` functions to define these links. The corresponding methods of the `ListingController` are already available at this point.

Add the navigation bar component to the layout.

4. Index of job listings

Now let's create a list of job listings.

4.1. Index method

First, add a code to the index method of the `ListingController` to select all listings and pass them to the `listings.index` view:

```
public function index()
{
    $listings = Listing::all();
    return view('listings.index', compact('listings'));
}
```

4.2. Listing card component

Let's create a `<x-job-card>` Blade component to display each job listing in a card format. Run the Artisan command:

```
php artisan make:component JobCard
```

This creates:

- app/View/Components/JobCard.php
- resources/views/components/job-card.blade.php

Update the JobCard component class like the following:

```
class JobCard extends Component
{
    public $listing;

    public function __construct($listing)
    {
        $this->listing = $listing;
    }

    public function render(): View|Closure|string
    {
        return view('components.job-card');
    }
}
```

Create the blade template for the card by updating the component view file like the following (feel free to use another HTML structure if you prefer it):

```
<div class="card mb-3 shadow-sm">
    <div class="card-body">
        <h5 class="card-title">{{ $listing->title }}</h5>
        <h6 class="card-subtitle mb-2 text-muted">{{ $listing->company_name }}</h6>
        <p class="card-text">{{ $listing->description }}</p>

        <ul class="list-unstyled mb-3">
            <li><strong>Salary:</strong> {{ $listing->salary ? '$' .
number_format($listing->salary, 2) : 'N/A' }}</li>
            <li><strong>Category:</strong> {{ $listing->jobCategory->name }}</li>
            <li><strong>Type:</strong> {{ $listing->employmentType->name }}</li>
            <li><strong>Posted by:</strong> {{ $listing->user->name }}</li>
            <li><strong>Date:</strong> {{ $listing->created_at?->format('Y-m-d')
}}</li>
        </ul>

        <a href="#" class="btn btn-primary">View Details</a>
    </div>
</div>
```

Check that the names of the relationships in `Listing` model correspond to `jobCategory` and `employmentType`. Otherwise, update the view code with the relationship names that you used.

Replace `href="#"` with a link to the route that displays information about a job listing (use action or route functions for this).

4.3. Index view

Create a folder `listings` in resources/views folder. Create a view file `index.blade.php` in the `listings` folder.

Put the following code in the index.blade.php file (feel free to use another HTML structure if you prefer it):

```
<x-layout>
    <x-slot name="title">
        Job Listings
    </x-slot>

    <h1 class="mb-4">Job Listings</h1>
    @if ($listings->count())
        <div class="row">
            @foreach ($listings as $listing)
                <div class="col-md-6 col-lg-4">
                    <x-job-card :listing="$listing" />
                </div>
            @endforeach
        </div>
    @else
        <div class="alert alert-info">No job listings available.</div>
    @endif
</x-layout>
```

This view uses layout component (x-layout tags) created in section 3. It defines the title slot of the layout (x-slot tags). Then there comes the heading “Job Listings”. After that the code checks if there are any listings to show and if yes, displays information about each listing as a job card component. If there are no listings, the text “No job listings available” is displayed.

Test the index functionality by visiting /listing URL. Since we don’t have any listings yet, the text “No job listings available” will be displayed.

5. Creating a new job listing

Let’s add job listing creation functionality to our application.

5.1. Create method

The method `create` of `ListingController` is used to show the form to add a new job listing. We need to add the code to load the form view and pass all job categories and employment types to the view to make it possible to populate the corresponding dropdowns.

```
public function create()
{
    $categories = JobCategory::all();
    $types = EmploymentType::all();
    return view('listings.create', compact('categories', 'types'));
}
```

5.2. Job listing creation form

In the next step, we will create a form view which will be used to fill in the information about a new listing. This view is the one that is returned by the method `create` (see the previous sub-section).

Create a view file `create.blade.php` in the `resources/views/listings` folder.

Put the following code in the create.blade.php file (feel free to use another HTML structure if you prefer it):

```
<x-layout>
    <x-slot name="title">
        Create New Job Listing
    </x-slot>

    <h1 class="mb-4">Create New Job Listing</h1>
    <form method="POST" action="{{ route('listing.store') }}">
        @csrf

        <div class="mb-3">
            <label class="form-label">Title</label>
            <input type="text" name="title" class="form-control">
        </div>

        <div class="mb-3">
            <label class="form-label">Description</label>
            <textarea name="description" class="form-control" rows="5"></textarea>
        </div>

        <div class="mb-3">
            <label class="form-label">Company Name</label>
            <input type="text" name="company_name" class="form-control">
        </div>

        <div class="mb-3">
            <label class="form-label">Salary</label>
            <input type="number" step="0.01" name="salary" class="form-control">
        </div>
```

```
        <div class="mb-3">
            <label class="form-label">Job Category</label>
            <select name="job_category_id" class="form-select">
                <option value="">Select Category</option>
                @foreach($categories as $category)
                    <option value="{{ $category->id }}">{{ $category->category }}</option>
                @endforeach
            </select>
        </div>

        <div class="mb-3">
            <label class="form-label">Employment Type</label>
            <select name="employment_type_id" class="form-select">
                <option value="">Select Type</option>
                @foreach($types as $type)
                    <option value="{{ $type->id }}">{{ $type->type }}</option>
                @endforeach
            </select>
        </div>

        <button type="submit" class="btn btn-primary">Create Listing</button>
    </form>
</x-layout>
```

This code uses a layout we created previously. It defines the title slot as “Create New Job Listing”. Then it creates a heading and a new listing form. The form code is generated as a regular HTML code. There are 2 select elements populated with a list of job categories and employment types. The action of the form will be generated as a URL which corresponds to the route “listing.store”. This is the name of the route which is mapped to the method `store` of `ListingController`. You can see this mapping by running Artisan command:

```
php artisan route:list
```

5.2. Method to store a new job listing in the database

Now let’s add a code to the method `store` of `ListingController` (user_id is hard-coded as 1 in this method implementation, because we currently don't have user authentication):

```
public function store(Request $request)
{
    Listing::create([
        'title' => $request->title,
        'description' => $request->description,
        'company_name' => $request->company_name,
        'salary' => $request->salary,
        'user_id' => 1,
        'job_category_id' => $request->job_category_id,
        'employment_type_id' => $request->employment_type_id,
    ]);

    return redirect()->route('listing.index')->with('success', 'Listing created
successfully!');
}
```

Now you can test the implemented functionality by visiting /listing/create URL. You must see a form. Enter data in the form and click “Create Listing”. The new listing must be added to the database and you must be redirected to the index view where a newly added listing will be displayed. Try adding several listings and see that list of them in the index view.

5.3. Input validation

Your task is to add input validation to the job listing creation functionality:

- Add validation rules to check request in the controller method `store`.
- Add validation error displaying in the new job listing form view.
- Add “old” values of input fields using a helper function `old` to repopulate a form input with the previously entered value after a validation error.

6. Showing information about a job listing

Your task is to add the functionality to view detailed information about a single job listing:

- Implement ListingController method `show`
- Add Blade view to display listing details
- Add a link to the show page from the job card view

7. Editing existing job listing

Your task is to add the functionality to edit information of the existing job listing:

- Implement `ListingController` method `edit` used to show the form for editing the job listing
- Implement `ListingController` method `update` used to update the job listing information in the database
- Add a form view for editing job listing information as a Blade view (`edit.blade.php`)
- Add a button on job listing detailed view created in section 6.
- Don’t forget to validate input data (see section 5.3.).