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 (PWL)
Studina Program : D4 – Informatics Engineering / D4 – Business Information Systems
Semester : 4 (four) / 6 (six)
Meeting to- : 6 (six)

JOBSHEET 06

Template Form (AdminLTE), Server Validation, Client Validation, CRUD

Admin LTE is one of the templates that is often used by web developers as a backend template on projects that are often worked on. So this LTE admin is an Administrator dashboard created using bootstrap which is the most widely used css framework.

In this meeting, we will understand how to operate forms on AdminLTE, Client & server validation, and CRUD interactions.

In accordance with `Case Study PWL.pdf`. So our Laravel 10 project is still the same as using **the PWL_POS repository.**

We will use PWL_POS project until the 12th meeting later, as a project that we will study

Laravel provides several different approaches to validating application logins. The most common way is to use the *validate method* available on all incoming HTTP requests. However, we will also discuss other validation approaches.

The process of client side validation or *client side validation* translation is more done by the web browser as the client, where in the web browser there is already the use of libraries that are able to translate commands that are run on web pages using client side scripting.

Examples of client side scripting include:

- HTML
- XHTML
- CSS
- JavaScript
- XML
- jQuery

The validation process from the *server side* is carried out on the server side, after the data is sent by the browser and successfully received by the server, then in the server there is a module that is done to validate the data, before finally the process is continued.

The server-side validation process is done using programming languages such as PHP or SQL and others.

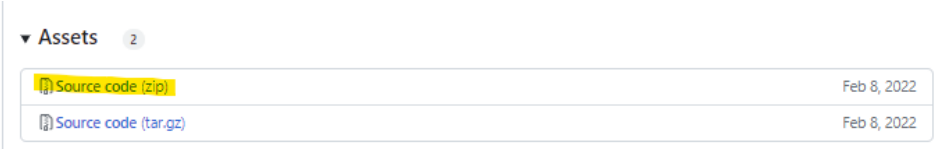
Examples of server side scripting include:

- ASP (Active Server Pages)
- PHP Hypertext Processor
- JSP (Java Server Pages)
- Dll

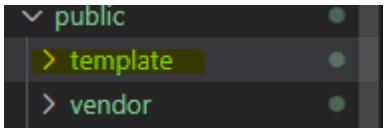
Client side validation is more done on the browser side and not for security purposes, but more for user convenience. While *server side validation* is carried out on the server side with security purposes by *filtering* all *incoming requests* before finally being processed further.

A. Template Form (AdminLTE)

1. Use the AdminLTE Bootstrap Admin Dashboard Template template with the first step of accessing https://adminlte.io/, then click download on the source code (zip)



2. Extract All and rename the *AdminLTE-3.2.0* folder to **template,** move the template folder to laragon/www/PWL_POS/public. Open the public folder on VSCode and a new folder named template will appear.



3. Copy the contents of the public/template/index2.html in welcome.blade.php then edit it so that it can be loaded, edit href ="{{ asset ('template/.....') }}" and src= " {{ asset ('template/.....')" as shown below

```
<!-- Google Font: Source Sans Pro -->
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,400i,700&display=fallback">
<!-- Font Awesome Icons -->
<link rel="stylesheet" href="{{ asset('template/plugins/fontawesome-free/css/all.min.css') }}">
<!-- overlayScrollbars -->
<link rel="stylesheet" href="{{ asset('template/plugins/overlayScrollbars/css/OverlayScrollbars.min.css') }}">
<!-- Theme style -->
<link rel="stylesheet"href="{{ asset('template/dist/css/adminlte.min.css') }}">
/head>
body class="hold-transition dark-mode sidebar-mini layout-fixed layout-navbar-fixed layout-footer-fixed">
div class="wrapper">

<!-- Preloader -->
<div class="preloader flex-column justify-content-center align-items-center">
  <img class="animation__wobble" src="{{asset('template/dist/img/AdminLTELogo.png') }}" alt="AdminLTELogo" height="60" width="60">
</div>
```

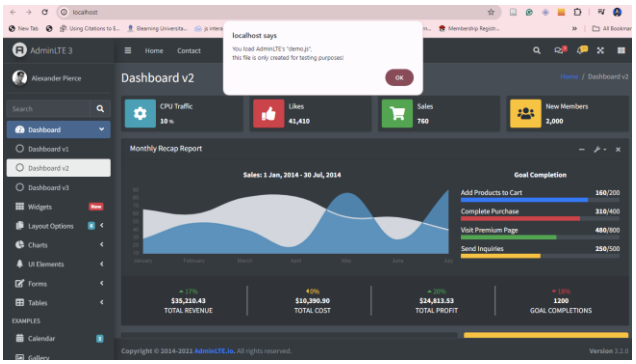
4. Scroll to the very bottom and adjust it with the following code

```
1713    <!-- REQUIRED SCRIPTS -->
1714    <!-- jQuery -->
1715    <script src="{{ asset ('template/plugins/jquery/jquery.min.js') }}"></script>
1716    <!-- Bootstrap -->
1717    <script src="{{ asset ('template/plugins/bootstrap/js/bootstrap.bundle.min.js') }}"></script>
1718    <!-- overlayScrollbars -->
1719    <script src="{{ asset ('template/plugins/overlayScrollbars/js/jquery.overlayScrollbars.min.js') }}"></script>
1720    <!-- AdminLTE App -->
1721    <script src="{{ asset ('template/dist/js/adminlte.js') }}"></script>
1722
1723    <!-- PAGE PLUGINS -->
1724    <!-- jQuery Mapael -->
1725    <script src="{{ asset ('template/plugins/jquery-mousewheel/jquery.mousewheel.js') }}"></script>
1726    <script src="{{ asset ('template/plugins/raphael/raphael.min.js') }}"></script>
1727    <script src="{{ asset ('template/plugins/jquery-mapael/jquery.mapael.min.js') }}"></script>
1728    <script src="{{ asset ('template/plugins/jquery-mapael/maps/usa_states.min.js') }}"></script>
1729    <!-- ChartJS -->
1730    <script src="{{ asset ('template/plugins/chart.js/Chart.min.js') }}"></script>
1731
1732    <!-- AdminLTE for demo purposes -->
1733    <script src="{{ asset ('template/dist/js/demo.js') }}"></script>
1734    <!-- AdminLTE dashboard demo (This is only for demo purposes) -->
1735    <script src="{{ asset ('template/dist/js/pages/dashboard2.js') }}"></script>
1736    </body>
1737    </html>
1738
```

Note: if the image in dist/img cannot be loaded, then adjust it with the asset

```
<img src="{{ asset('template/dist/img/user1-128x128.jpg')}}"
```

5. Run in browser and give screen shoot.

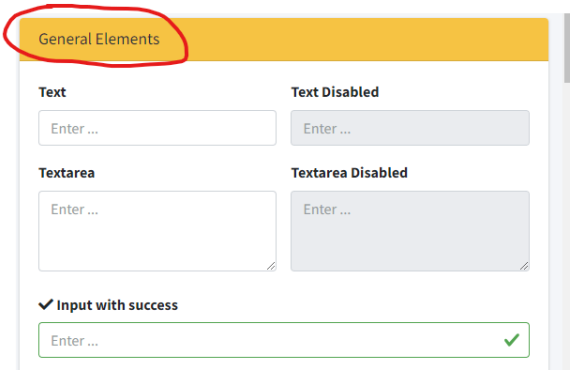


*Note: This template loads everything, delete it if not needed. Visuals on the LTE admin can make it easier to retrieve code.

6. In the menu section there is a Form option we first specialize in the Form provided by the LTE Admin

- General Elements
- Advanced Elements
- Editor
- Validation

How to use the template can be seen the name of the form example title 'General Elements', so that it can be used to access the folder containing the adminLTE template example: C:\laragon\www\PWL_2024\public\template\pages\forms



7. Open the file on the form> general search for keywords with 'general elements' and try modifying the welcome.blade.php

```
2   <html lang="en">
15  <body class="hold-transition sidebar-mini">
16  <div class="wrapper">
831   <div class="content-wrapper">
850     <section class="content">
1153              </div>
1154            </div>
1155            <!-- /.card-body -->
1156          </div>
1157          <!-- /.card -->
1158
1159          <!-- general form elements disabled -->
1160          <div class="card card-warning">
1161            <div class="card-header">
1162              <h3 class="card-title">General Elements</h3>
1163            </div>
1164            <!-- /.card-header -->
1165            <div class="card-body">
1166              <form>
1167                <div class="row">
1168                  <div class="col-sm-6">
1169                    <!-- text input -->
1170                    <div class="form-group">
1171                      <label>Text</label>
1172                      <input type="text" class="form-control" placeholder="Enter ...">
1173                    </div>
1174                  </div>
```

```
@extends('adminlte::p age')

@section('title', 'Dashboard')

@section('content_header')
<h1>Dashboard</h1>
@stop

@section('content')

<div class="card-body">
<form>
<div class="row">
<div class="col-sm-6">
<!-- text input -->
<div class="form-group">
<label>Level id</label><input type="text" class="form-control" placeholder="id">
<div>
</div>
<button type = "submit" class ="btn btn-info">Submit </button>
</div>
@stop

@section('css')
{{-- Add here extra stylesheets --}}
{{-- <link rel="stylesheet" href="/css/admin_custom.css"> --}}
@stop

@section('js')
<script> console.log("Hi, I'm using the Laravel-AdminLTE package!"); </script>
@stop
```

8. Run in the browser and give screen shoot.
9. Explore the form types in adminLTE and try to apply for case studies, create forms for m_user and m_level tables.

B. VALIDATION ON SERVER

1. Reopen the task file in jobsheet 05, click and read this documentation Validation - Laravel 10.x - The PHP Framework For Web Artisans
2. Make sure the route is defined in the web.php:

```
Route::get('/kategori/create', [KategoriController::class, 'create']);
Route::post('/kategori', [KategoriController::class, 'store']);
```

3. Edit on CategoryController with code:

app > Http > Controllers > KategoriController.php > PHP > App\Http\Controllers\KategoriController

```php
namespace App\Http\Controllers;

use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\View\View;

3 references | 0 implementations
class KategoriController extends Controller
{
    /**
     * Show the form to create a new post.
     */
    0 references | 0 overrides
    public function create(): View
    {
        return view('kategori.create');
    }
    /**
     * Store a new post.
     */
    0 references | 0 overrides
    public function store(Request $request): RedirectResponse
    {
    $validated = $request->validate([
        'kategori_kode' => 'required',
        'kategori_nama' => 'required',
    ]);

    // The post is valid...

    return redirect('/kategori');
    }
}
```

4. The following is an alternative example of writing validation and you can match the words to what you are doing,

Alternatively, validation rules may be specified as arrays of rules instead of a single | delimited string:

```php
$validatedData = $request->validate([
    'title' => ['required', 'unique:posts', 'max:255'],
    'body' => ['required'],
]);
```

In addition, you may use the `validateWithBag` method to validate a request and store any error messages within a named error bag:

```php
$validatedData = $request->validateWithBag('post', [
    'title' => ['required', 'unique:posts', 'max:255'],
    'body' => ['required'],
]);
```

5. Write the difference in using validate with validateWithBag!

6. Sometimes you may want to stop running validation rules on an attribute after the first validation failure. To do so, assign a **bail** rule to the attribute: try adjusting it to the field on the m_kategori. What happened?

```
$request->validate([
    'title' => 'bail|required|unique:posts|max:255',
    'body' => 'required',
]);
```

7. In view/create.blade.php add the following code so that when validation fails, we can display an error message in the view:

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

8. Screenshot the result.
9. Note: Customizing Error Messages
   Laravel's built-in validation rules each have an error message located in lang/en/validation.php your application file. If your application doesn't have langdirectory, you can instruct Laravel to create it using the lang:publish Artisan command. Inside the lang/en/validation.php file, you'll find translation entries for each validation rule. You are free to change or modify these messages based on the needs of your application.
10. In view/create.blade.php add and try running the following code:

```
<label for="kategori_kode">Kode kategori</label>

<input id="kategori_kode"
    type="text"
    name="kategori_kode"
    class="@error('kategori_kode') is-invalid @enderror">

@error('kategori_kode')
    <div class="alert alert-danger">{{ $message }}</div>
@enderror
```

C. FORM REQUEST VALIDATION

1. Make a form request by writing in the terminal

```
php artisan make:request StorePostRequest
```

The resulting form request class will be placed in the app/Http/Requests directory. If this directory does not exist, it will be created when you run the make:request command. Each form request generated by Laravel has two methods: authorize and rules.

2. Type the following code in Http/request/StorePostRequest

```php
/**
 * Get the validation rules that apply to the request.
 *
 * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
 */
0 references | 0 overrides
public function rules(): array
{
    return [
    'kategori_kode' => 'required',
    'kategori_nama' => 'required',
    ];
}
```

```php
0 references | 0 overrides
public function store(StorePostRequest $request): RedirectResponse
{
    // The incoming request is valid...

    // Retrieve the validated input data...
    $validated = $request->validated();

    // Retrieve a portion of the validated input data...
    $validated = $request->safe()->only(['kategori_kode', 'kategori_nama']);
    $validated = $request->safe()->except(['kategori_kode', 'kategori_nama']);

    // Store the post...

    return redirect ('/kategori');
}
```

Note: If validation fails, a redirect response will be generated to send the user, return to the previous location. The error will also be shown to the session so that it is available to display. If the request is an XHR request, the HTTP response with Status code 422 is returned to the user.

3. Apply validation also to m_user and m_level tables.

D. CRUD(create, read, update, delete)

1. Create a POSController complete with its resources, Create a Resource Controller and Route that functions for CRUD routes so there is no need to bother creating each route such as post, get, delete and update.

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

2. Here's the command: now add the code to route/web.php

```
Route::resource('m_user', POSController::class);
```

3. Set it to Models/m_user

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class m_user extends Model
{
    protected $table ="m_user";
    public $timestamps = false;
    protected $primaryKey = 'user_id';

   protected $fillable =[
    'user_id',
    'level_id',
    'username',
    'name',
    'password',
];

}
```

4. Set on Migration/m_user_table

```
<?php

use App\Models\m_level;
use Illuminate\Database\Migrations\Migration;
```

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
* Run the migrations.
*/
    public function up(): void
{
        Schema::create('useri', function (Blueprint $table) {
            $table-> id('user_id');
            $table -> unsignedBigInteger('level_id')->index;
            $table ->string ('username', 20)->unique();
            $table  ->string('name',  100);
            $table ->string ('password');
            $table ->timestamps();

            $table->foreign('level_id')-> references('level_id')->on
('m_level');
});
}

    /**
* Reverse the migrations.
*/
    public function down(): void
{
        Schema::dropIfExists('useri');
}
};
```

5. In the Controller file that we can automatically there will be the following 7 functions that we can use to create CRUD operations.
   - index()
   - create()
   - store()
   - show()
   - edit()
   - update()
   - destroy()

6. Then open app/Http/Controllers/POSController.php then type, the code as follows:

```php
<?php

namespace App\Http\Controllers;

use App\Models\m_user;
use Illuminate\Http\Request;

class POSController extends Controller
{
    /**
* Display a listing of the resource.
*/
    public function index()
{
        The eloquent function displays data using pagination
        $useri = m_user::all();  Retrieve all contents of a table
        return view('m_user.index', compact('useri'))->with('i');
}



    /**
* Show the form for creating a new resource.
*/
    public function create()
{
        return view('m_user.create');
}

    /**
* Store a newly created resource in storage.
*/
    public function store(Request $request)
{
        Perform data validation
     $request->validate([
    'user_id' = > 'max 20',
    'username' => 'required',
    'nama' => 'required',

]);
    Eloquent function to add data
    m_user::create($request->all());
```

```
    return redirect()->route('m_user.index')
 ->with('success', 'user added successfully');
}

    /**
* Display the specified resource.
*/
    public function show(string $id, m_user $useri)
{
        $useri = m_user::findOrFail($id);
        return view('m_user.show', compact('useri'));
}

    /**
* Show the form for editing the specified resource.
*/
    public function edit(string $id)
{
        $useri = m_user::find($id);
        return view('m_user.edit', compact('useri'));
}

    /**
* Update the specified resource in storage.
*/
    public function update(Request $request, string $id)
{
        $request->validate([

            'username' => 'required',
            'nama' => 'required',
            'password' => 'required',
]);
           eloquent function to update our input data
m_user::find($id)->update($request->all());
If the data is successfully updated, it will return to the main page
return redirect()->route('m_user.index')
->with('success', 'Data updated successfully');
}

    /**
* Remove the specified resource from storage.
*/
    public function destroy(string $id)
```

```
{
        $useri = m_user::findOrFail($id)->delete();
        return \redirect()->route('m_user.index')

 -> with('success', 'data deleted successfully');
}
}
```

7. Create a folder in Resources/Views/**m_user** with some blades and the following code fields:

1. template.blade.php

```
<!DOCTYPE html>
<html>
<head>
<titill>CRUD Laravel</titill>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.mi
n.css">
</head>
<body>
<div class="container">
@yield('content')
</div>
</body>
</html>
```

2. index.blade.php

```
@extends('m_user/template')
@section('content')
<div class="row mt-5 mb-5">
<div class="col-lg-12 margin-tb">
<div class="float-left">
<h2>CRUD user</h2>
</div>
<div class="float-right">
<a class="btn btn-success" href="{{ route('m_user.create') }}"> Input
User</a>
</div>
</div>
```

```
</div>
@if ($message = Session::get('success'))
<div class="alert alert-success">
<p>{{ $message }}</p>
</div>
@endif
<table class="table table-bordered">
<tr>
<th width="20px" class="text-center">User id</th>
<th width="150px" class="text-center">Level id</th>
<th width="200px"class="text-center">username</th>
<th width="200px"class="text-center">nama</th>
<th width="150px"class="text-center">password</th>
</tr>
@foreach ($useri as $m_user)
<tr>

<td>{{ $m_user->user_id }}</td>
<td>{{ $m_user->level_id }}</td>
<td>{{ $m_user->username }}</td>
<td>{{ $m_user->nama }}</td>
<td>{{ $m_user->password }}</td>


<td class="text-center">
<form action="{{ route('m_user.destroy',$m_user->user_id) }}"
method="POST" >
<a class="btn btn-info btn-sm" href="{{ route('m_user.show',$m_user-
>user_id) }}">Show</a>
<a class="btn btn-primary btn-sm" href="{{
route('m_user.edit',$m_user->user_id) }}">Edit</a>
@csrf
@method('DELETE')
<button type="submit" class="btn btn-danger btn-sm" onclick="return
confirm('Are you sure you want to delete this data?')">Delete</button>
</form>
</td>
</tr>
@endforeach
</borð>
@endsection
```

3. create.blade.php

```
@extends('m_user/template')
@section('content')
<div class="row mt-5 mb-5">
<div class="col-lg-12 margin-tb">
<div class="float-left">
<h2>Create User</h2 List>
</div>
<div class="float-right">
<a class="btn btn-secondary" href="{{ route('m_user.index') }}">
Kembali </a>
</div>
</div>
</div>
@if ($errors->any()).
<div class="alert alert-danger">
<strong>Ops</strong> Input gagal<br><br>
<ul>
@foreach ($errors->all() as $error)
<li>{{ $error }}</li>
@endforeach
</ul>
</div>
@endif
<form action="{{ route('m_user.store') }}" method="POST">
@csrf



<div class="col-xs-12 col-sm-12 col-md-12">
<div class="form-group">
<strong>Username:</strong>
<input type="text" name="username" class="form-control"
placeholder="Masukkan username"></input>
</div>
</div>

<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
    <strong>nama:</strong>
    <input type="text" name="name" class="form-control"
placeholder="Enter name"></input>
    </div>
    </div>
```

```
<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
    <strong>Password:</strong>
    <input type="password" name="password" class="form-control"
placeholder="Masukkan password"></input>
    </div>
    </div>
<div class="col-xs-12 col-sm-12 col-md-12 text-center">
<button type="submit" class="btn btn-primary">Submit</button>
</div>
</div>
</form>
@endsection
```

4. show.blade.php

```
@extends('m_user/template')
@section('content')
<div class="row mt-5 mb-5">
<div class="col-lg-12 margin-tb">
<div class="float-left">
<h2> Show User</h2>
</div>
<div class="float-right">
<a class="btn btn-secondary" href="{{ route('m_user.index') }}">
Kembali </a>
</div>
</div>
</div>
<div class="row">
<div class="col-xs-12 col-sm-12 col-md-12">
<div class="form-group">
<strong>User_id:</strong>
{{ $useri->user_id }}
</div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12">
<div class="form-group">
<strong>Level_id: </strong>
{{ $useri->level_id }}
</div>
</div>
```

```
<div class="col-xs-12 col-sm-12 col-md-12">
<div class="form-group">
<strong>Username:</strong>
{{ $useri->username }}
</div>
</div>


<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
    <strong>Nama:</strong>
    {{ $useri->} }
    </div>
    </div>

<div class="col-xs-12 col-sm-12 col-md-12">
<div class="form-group">
<strong>Password:</strong>
{{ $useri->password }}
</div>
</div>

</div>
@endsection
```

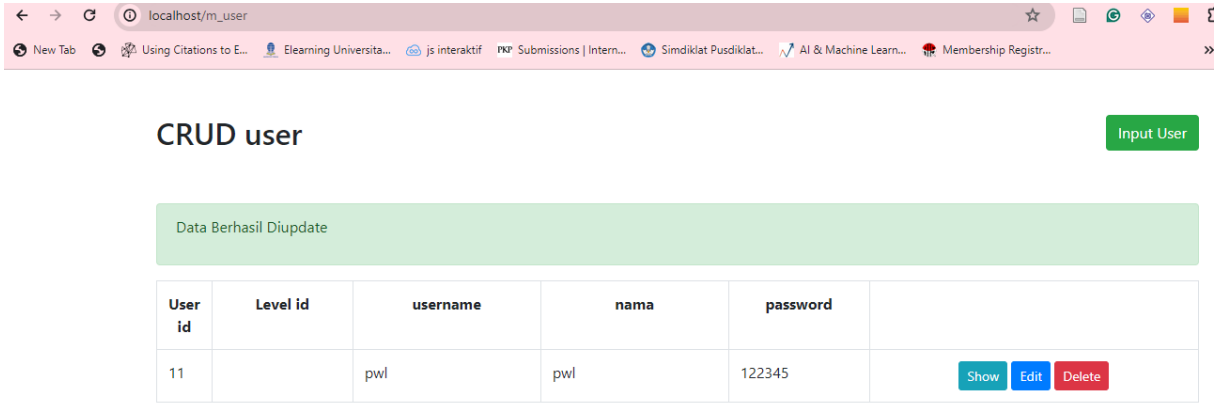
5. edit.blade.php

```
@extends('m_user/template')
@section('content')
<div class="row mt-5 mb-5">
<div class="col-lg-12 margin-tb">
<div class="float-left">
<h2>Edit User</h2>
</div>
<div class="float-right">
<a class="btn btn-secondary" href="{{ route('m_user.index') }}">
Kembali </a>
</div>
</div>
</div>
@if ($errors->any()).
<div class="alert alert-danger">
<strong>Ops!</strong> Error <br><br>
<ul>
```

```
@foreach ($errors->all() as $error)
<li>{{ $error }}</li>
@endforeach
</ul>
</div>
@endif
<form action="{{ route('m_user.update', $useri->user_id) }}"
method="POST" >
@csrf
@method('PUT')
<div class="row">
    <div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
    <strong>User_id:</strong>
    <input type="text" name="userid" value="{{ $useri->user_id }}"
class="form-control" placeholder="Masukkan user id">
    </div>
    </div>

<div class="col-xs-12 col-sm-12 col-md-12">
<div class="form-group">
<strong>Level_id: </strong>
<input type="text" name="levelid" value="{{ $useri->level_id }}"
class="form-control" placeholder="Masukkan level">
</div>
</div>

<div class="col-xs-12 col-sm-12 col-md-12">
<div class="form-group">
<strong>Username:</strong>
<input type="text" value= "{{ $useri->username }}" class="form-
control" name="username" placeholder="Enter Username number",">
</div>
</div>

<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
    <strong>nama:</strong>
    <input type="text" value= "{{ $useri->name }}"name="name"
class="form-control" placeholder="Enter name"></input>
    </div>
    </div>
```

```
<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
    <strong>Password:</strong>
    <input type="password" value= "{{ $useri->password
}}"name="password" class="form-control" placeholder="Masukkan
password"></input>
    </div>
    </div>


<div class="col-xs-12 col-sm-12 col-md-12 text-center">
<button type="submit" class="btn btn-primary">Update</button>
</div>
</div>
</form>
@endsection
```

Please access localhost/m_user



Assignment:

1. Try displaying the level_id on the web page where this field is a foreign key
2. Modify with your favorite theme / template.
3. What are the functions of `$request->validate`, $error and alerts on the CRUD page.

*** Thank you, and good luck ***