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

Subject	:	ADVANCE WEB PROGRAMMING
Program Studi	:	D4 – Informatics Engineering / D4 – Business Information Systems
Semester	:	4 (four) / 6 (six)
Meeting to-	:	11 (eleven)

JOBSHEET 11
RESTFUL API 2

Before we enter the material, we first create a new project that we will use to build a simple application with the topic *of Point of Sales (PoS),* according to

Sebelum kita masuk materi, kita buat dulu project baru yang akan kita gunakan untuk membangun aplikasi sederhana dengan topik *Point of Sales (PoS)*, sesuai dengan `Studi Kasus PWL.pdf`.
Jadi kita bikin project Laravel 10 dengan nama **PWL_POS.**

Project **PWL_POS** akan kita gunakan sampai pertemuan 12 nanti, sebagai project yang akan kita pelajari

A. ELOQUENT ACCESSOR

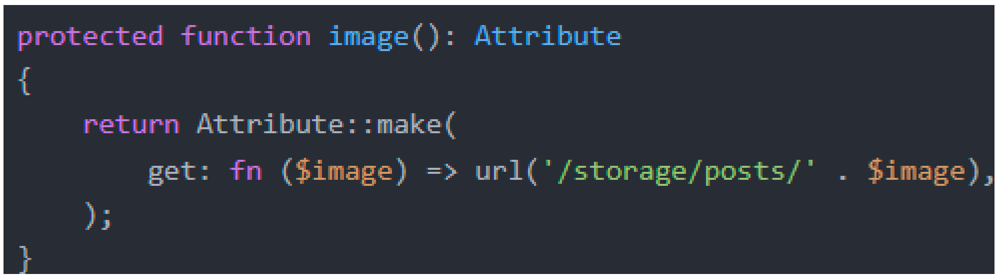
Laravel has features called mutator, accessor and casting, these features are used to manipulate data in database attributes very easily. For example, when inserting data with encryption into the database and doing descriptions when displaying from the database automatically.

The accessor can change the value when the attribute or eloquent field is accessed. To define the Accessor can create a method inside the model to specify the attributes to be accessed. The name of the method created must be the same as the name of the attribute to be formatted: example:

Attribute/field in table first_name then method firstName()

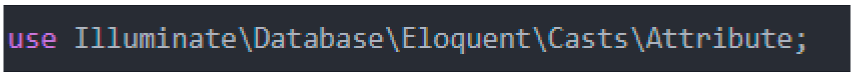
```
protected function firstName(): Attribute
{
   //...
}
```

If we create an existing image attribute/field in the m_user we will provide the full path value of the directory where the image file is stored.



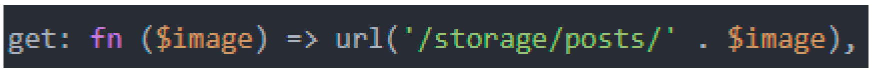
```
protected function image(): Attribute
{
    return Attribute::make(
        get: fn ($image) => url('/storage/posts/' . $image),
    );
}
```

That way you can import Eloquent Attributes with



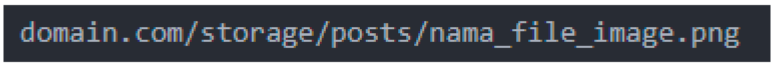
```
use Illuminate\Database\Eloquent\Casts\Attribute;
```

Then a new method with the name image() returns the path of the image file name it is located in



```
get: fn ($image) => url('/storage/posts/' . $image),
```

The final result calls the image attribute

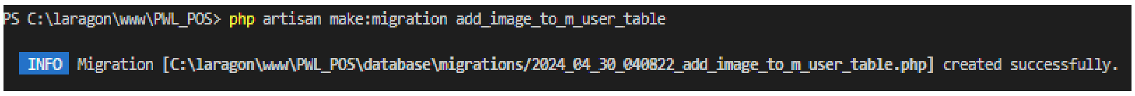


```
domain.com/storage/posts/nama_file_image.png
```

Practicum 1 – Eloquent Accessor Implementation

1. Before starting make sure the REST API, first make sure the Postman application is installed.

2. We will modify the Table m_user by adding column: image, open terminal and type
   `php artisan make:migration add_image_to_m_user_table`



```
PS C:\laragon\www\PWL_POS> php artisan make:migration add_image_to_m_user_table

 INFO  Migration [C:\laragon\www\PWL_POS\database\migrations/2024_04_30_040822_add_image_to_m_user_table.php] created successfully.
```

3. <?php

```
<?php

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::table('m_user', function (Blueprint $table) {
            $table->string('image');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::table('m_user', function (Blueprint $table) {
            $table->dropColumn('image');
        });
    }
};
```

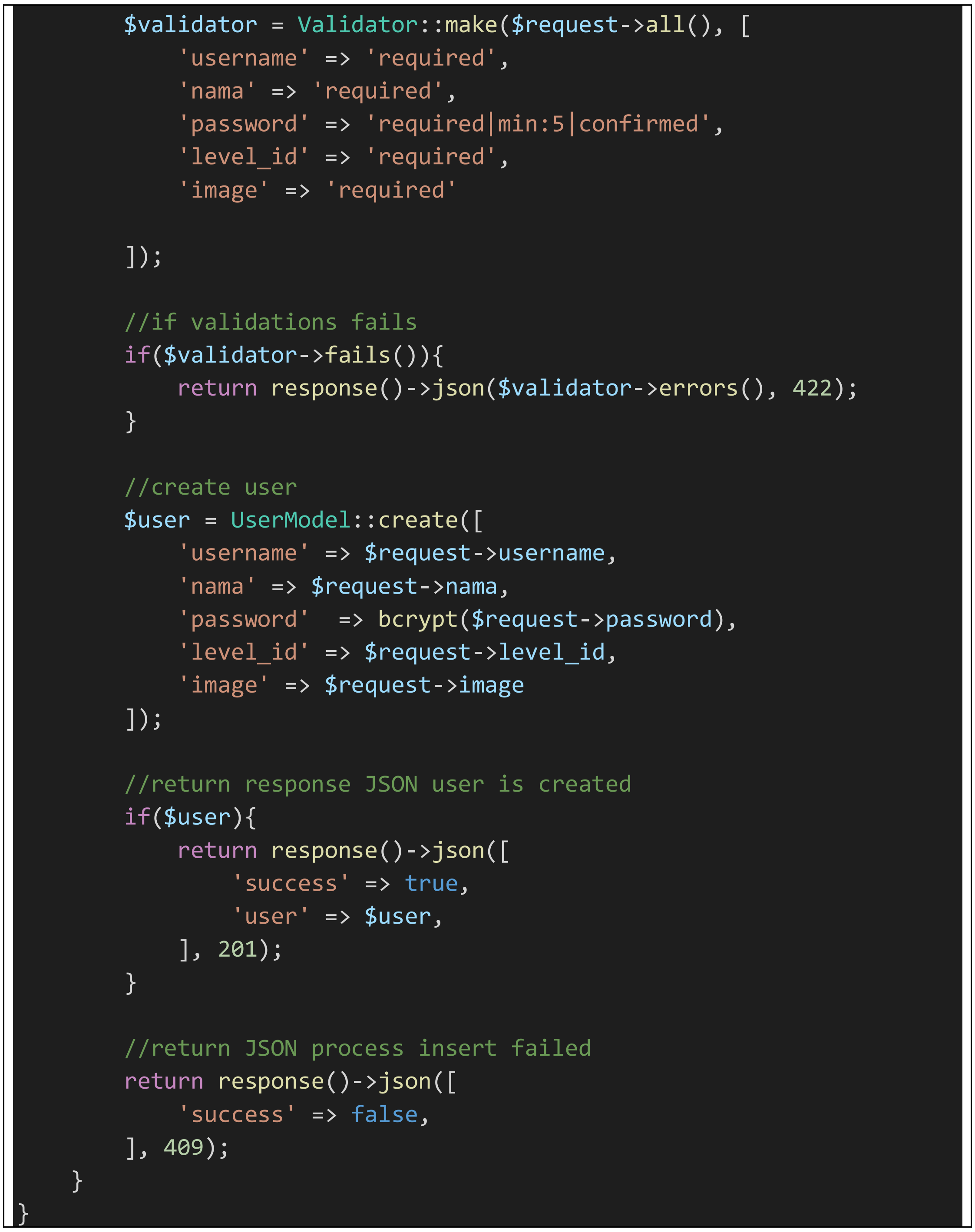
4. Run the migration update by:
   `php artisan migrate`

5. <?php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Tymon\JWTAuth\Contracts\JWTSubject;
use Illuminate\Database\Eloquent\Casts\Attribute;
use Illuminate\Foundation\Auth\User as Authenticatable;

class UserModel extends Authenticatable implements JWTSubject
{

    public function getJWTIdentifier(){
        return $this->getKey();
    }
    public function getJWTCustomClaims(){
        return [];
    }

    protected $table = 'm_user';
    protected $primaryKey = 'user_id';

    protected $fillable = [
        'username',
        'nama',
        'password',
        'level_id',
        'image'//tambahan
    ];

    public function level()
    {
        return $this->belongsTo(LevelModel::class, 'level_id',
'level_id');
    }
    protected function image(): Attribute
    {
        return Attribute::make(
            get: fn ($image) => url('/storage/posts/' . $image),
        );
    }

}
```

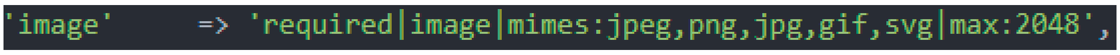
6. <?php

```
<?php

namespace App\Http\Controllers\Api;

use App\Models\UserModel;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

class RegisterController extends Controller
{
    public function __invoke(Request $request)
    {
        //set validation
```



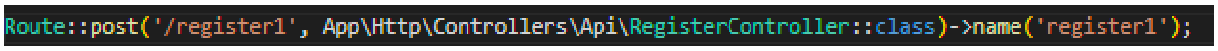
```
        $validator = Validator::make($request->all(), [
            'username' => 'required',
            'nama' => 'required',
            'password' => 'required|min:5|confirmed',
            'level_id' => 'required',
            'image' => 'required'

        ]);

        //if validations fails
        if($validator->fails()){
            return response()->json($validator->errors(), 422);
        }

        //create user
        $user = UserModel::create([
            'username' => $request->username,
            'nama' => $request->nama,
            'password'  => bcrypt($request->password),
            'level_id' => $request->level_id,
            'image' => $request->image
        ]);

        //return response JSON user is created
        if($user){
            return response()->json([
                'success' => true,
                'user' => $user,
            ], 201);
        }

        //return JSON process insert failed
        return response()->json([
            'success' => false,
        ], 409);
    }
}
```

7. You can add details for the image spec to the validator



```
'image'     => 'required|image|mimes:jpeg,png,jpg,gif,svg|max:2048',
```

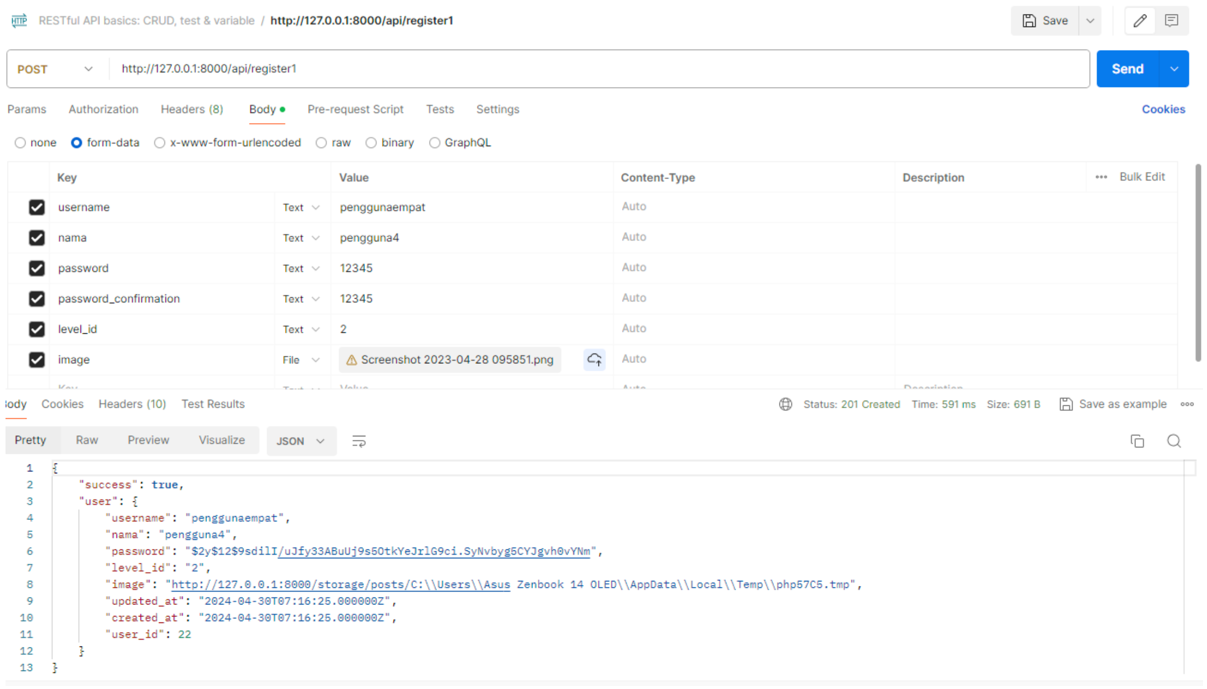
8. Change or add register1 to routes/api.php



```
Route::post('/register1', App\Http\Controllers\Api\RegisterController::class)->name('register1');
```

8. Save and access it in the Postman application, set it in the Body fill in the manual Key

and the Value in the Key image add the File value and upload the `http://127.0.0.1:8000/api/register1` image with the POST method and click send


RESTful API basics: CRUD, test & variable / http://127.0.0.1:8000/api/register1
Save
POST
http://127.0.0.1:8000/api/register1
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL

	Key		Value	Content-Type	Description
✓	username	Text	penggunaempat	Auto	
✓	nama	Text	pengguna4	Auto	
✓	password	Text	12345	Auto	
✓	password_confirmation	Text	12345	Auto	
✓	level_id	Text	2	Auto	
✓	image	File	Screenshot 2023-04-28 095851.png	Auto	

Body Cookies Headers (10) Test Results
Status: 201 Created Time: 591 ms Size: 691 B
Save as example
Pretty Raw Preview Visualize JSON

```
{
    "success": true,
    "user": {
        "username": "penggunaempat",
        "nama": "pengguna4",
        "password": "$2y$12$9sdilI/uJfy33ABuUj9s5OtkYeJrlG9ci.SyNvbyg5CYJgvh0vYNm",
        "level_id": "2",
        "image": "http://127.0.0.1:8000/storage/posts/C:\\Users\\Asus Zenbook 14 OLED\\AppData\\Local\\Temp\\php57C5.tmp",
        "updated_at": "2024-04-30T07:16:25.000000Z",
        "created_at": "2024-04-30T07:16:25.000000Z",
        "user_id": 22
    }
}
```

9. In the Controllers/Api/RegisterController section create user replace with

```
'image'         => $image->hashName(),
```

10. Test and screenshot the result what is the difference from the previous one

ASSIGNMENT

Implement an API to upload files/images to other tables, namely m_barang tables, and use them in transactions. Test with the GET method to call the data that has been inputted.

*That's it, and happy learning ****