

LAPORAN PRAKTIKUM
JOBSHEET 04
MODEL DAN ELOQUENT ORM

Disusun untuk memenuhi nilai tugas
Mata Kuliah : Pemrograman Web Lanjut



Oleh :
Aqueena Regita Hapsari
2341760096
SIB 2B
03

PROGRAM STUDI D-IV SISTEM INFORMASI BISNIS
POLITEKNIK NEGERI MALANG
TAHUN AJARAN 2024/2025

Mata Kuliah : Pemrograman Web Lanjut (PWL)
Program : D4 – Teknik Informatika / D4 – Sistem
Studi : Informasi Bisnis
Semester : 4 (empat) / 6 (enam)
Pertemuan : 1 (satu)
ke-

JOBSHEET 04

MODEL dan ELOQUENT ORM

Sebelumnya kita sudah membahas mengenai *Migration*, *Seeder*, *DB Façade*, *Query Builder*, dan sedikit tentang *Eloquent ORM* yang ada di Laravel. Sebelum kita masuk pada pembuatan aplikasi berbasis website, alangkah baiknya kita perlu menyiapkan Basis data sebagai tempat menyimpan data-data pada aplikasi kita nanti. Selain itu, umumnya kita perlu menyiapkan juga data awal yang kita gunakan sebelum membuat aplikasi, seperti data user administrator, data pengaturan sistem, dll.

Dalam pertemuang kali ini kita akan memahami tentang bagaimana cara menampilkan data, mengubah data, dan menghapus data menggunakan teknik Eloquent.

Sesuai dengan **Studi Kasus PWL.pdf**.

Jadi project Laravel 10 kita masih sama dengan menggunakan repositori **PWL_POS**.

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

ORM (Object Relation Mapping) merupakan teknik yang merubah suatu table menjadi sebuah object yang nantinya mudah untuk digunakan. Object yang dibuat memiliki property yang sama dengan field — field yang ada pada table tersebut. ORM tersebut bertugas sebagai penghubung dan sekaligus mempermudah kita dalam membuat aplikasi yang menggunakan database relasional agar menjadikan tugas kita lebih efisien.

Kelebihan - Kelebihan Menggunakan ORM

1. Terdapat banyak fitur seperti transactions, connection pooling, migrations, seeds, streams, dan lain sebagainya.

2. perintah query memiliki kinerja yang lebih baik, daripada kita menulisnya secara manual.
3. Kita menulis model data hanya di satu tempat, sehingga lebih mudah untuk update, maintain, dan reuse the code.
4. Memungkinkan kita memanfaatkan OOP (object oriented programming) dengan baik

Di Laravel sendiri telah disediakan Eloquent ORM untuk mempermudah kita dalam melakukan berbagai macam query ke database, dan membuat pekerjaan kita menjadi lebih mudah karena tidak perlu menuliskan query sql yang panjang untuk memproses data.

A. PROPERTI `$fillable` DAN `$guarded`

1. `$fillable`

Variable `$fillable` berguna untuk mendaftarkan atribut (nama kolom) yang bisa kita isi ketika melakukan insert atau update ke database. Sebelumnya kita sudah memahami menambahkan record baru ke database. Untuk langkah menambahkan Variable `$fillable` bisa dengan menambahkan *script* seperti di bawah ini pada file model

```
protected $fillable = ['level_id', 'username'];
```

Praktikum 1 - \$fillable:

1. Buka file model dengan nama `UserModel.php` dan tambahkan `$fillable` seperti gambar di bawah ini

```
class UserModel extends Model
{
    use HasFactory;

    protected $table = 'm_user';
    protected $primaryKey = 'user_id';
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = ['level_id', 'username', 'nama', 'password'];
}
```

2. Buka file controller dengan nama `UserController.php` dan ubah *script* untuk menambahkan data baru seperti gambar di bawah ini

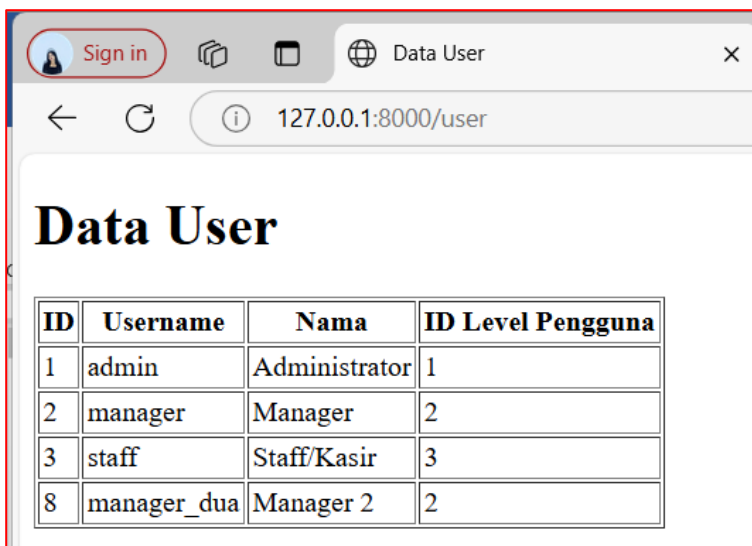
```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use App\Models\UserModel;
7  use Illuminate\Support\Facades\Hash;
8
9  class UserController extends Controller
10 {
11     public function index()
12     {
13         $data = [
14             'level_id' => 2,
15             'username' => 'manager_dua',
16             'nama' => 'Manager 2',
17             'password' => Hash::make('12345')
18         ];
19         UserModel::create($data);
20
21         $user = UserModel::all();
22         return view('user', ['data' => $user]);
23     }
24 }
25

```

3. Simpan kode program Langkah 1 dan 2, dan jalankan perintah web server. Kemudian jalankan link `localhostPWL_POS/public/user` pada *browser* dan amati apa yang terjadi

Hasil : Data baru dengan username `manager_dua` berhasil tersimpan di database.



ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1
2	manager	Manager	2
3	staff	Staff/Kasir	3
8	manager_dua	Manager 2	2

4. Ubah file model `UserModel.php` seperti pada gambar di bawah ini pada bagian `$fillable`

```
protected $fillable = ['level_id', 'username', 'nama'];
```

5. Ubah kembali file controller `UserController.php` seperti pada gambar di bawah hanya bagian array pada `$data`

```

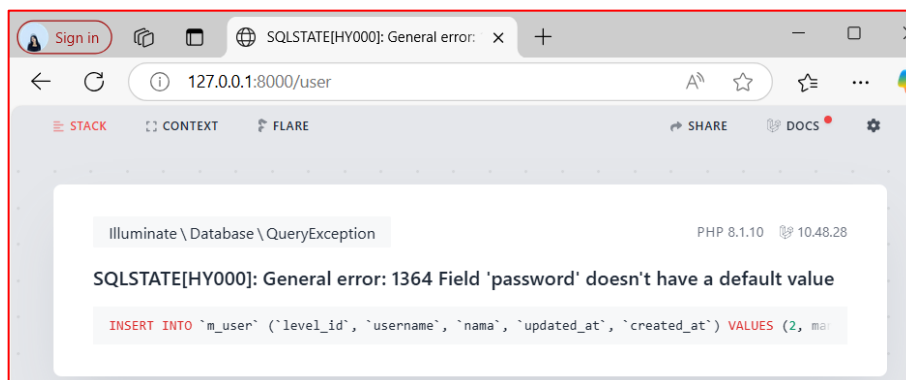
public function index()
{
    $data = [
        'level_id' => 2,
        'username' => 'manager_tiga',
        'nama' => 'Manager 3',
        'password' => Hash::make('12345')
    ];
    UserModel::create($data);

    $user = UserModel::all();
    return view('user', ['data' => $user]);
}

```

6. Simpan kode program Langkah 4 dan 5. Kemudian jalankan pada *browser* dan amati apa yang terjadi

Hasil : Error karena password tidak termasuk dalam \$fillable, sehingga mass assignment gagal. Jadi pada UserModel perlu ditambah field yang fillable password sehingga pada user controller bisa diinputkan passwordnya.



7. Laporkan hasil Praktikum-1 ini dan *commit* perubahan pada *git*.

Jawaban :

- \$fillable menentukan atribut mana yang bisa diisi menggunakan metode mass assignment seperti create().
- Jika suatu atribut tidak termasuk dalam \$fillable, maka Laravel akan mengabaikannya dan bisa menyebabkan error saat mencoba menyimpan data yang tidak diinginkan.
- Menggunakan \$fillable dengan benar sangat penting untuk menghindari error serta meningkatkan keamanan aplikasi dengan mencegah perubahan data yang tidak diinginkan.

2. \$guarded

Kebalikan dari \$fillable adalah \$guarded. Semua kolom yang kita tambahkan ke \$guarded

akan diabaikan oleh Eloquent ketika kita melakukan insert/update. Secara default `$guarded` isinya `array("*")`, yang berarti semua atribut tidak bisa diset melalui **mass assignment**. **Mass Assignment** adalah fitur canggih yang menyederhanakan proses pengaturan beberapa atribut model sekaligus, menghemat waktu dan tenaga. Pada praktikum ini, kita akan mengeksplorasi konsep penugasan massal di Laravel dan bagaimana hal itu dapat dimanfaatkan secara efektif untuk meningkatkan alur kerja pengembangan Anda.

B. RETRIEVING SINGLE MODELS

Selain mengambil semua rekaman yang cocok dengan kueri tertentu, Anda juga dapat mengambil rekaman tunggal menggunakan metode `find`, `first`, atau `firstWhere`. Daripada mengembalikan kumpulan model, metode ini mengembalikan satu contoh model dan dilakukan pada controller:

```
// Ambil model dengan kunci utamanya...
$user = UserModel::find(1);

// Ambil model pertama yang cocok dengan batasan kueri...
$user = UserModel::where('level_id', 1)->first();

// Alternatif untuk mengambil model pertama yang cocok dengan batasan kueri...
$user = UserModel::firstWhere('level_id', 1);
```

Praktikum 2.1 – Retrieving Single Models

1. Buka file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

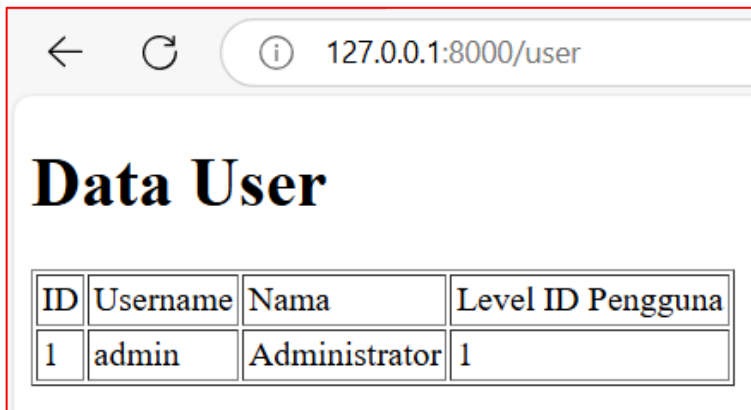
```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::find(1);
        return view('user', ['data' => $user]);
    }
}
```

2. Buka file `view` dengan nama `user.blade.php` dan ubah *script* seperti gambar di bawah ini

```
<body>
  <h1>Data User</h1>
  <table border="1" cellpadding="2" cellspacing="0">
    <tr>
      <td>ID</td>
      <td>Username</td>
      <td>Nama</td>
      <td>ID Level Pengguna</td>
    </tr>
    <tr>
      <td>{{ $data->user_id }}</td>
      <td>{{ $data->username }}</td>
      <td>{{ $data->nama }}</td>
      <td>{{ $data->level_id }}</td>
    </tr>
  </table>
</body>
```

3. Simpan kode program Langkah 1 dan 2. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Metode `find(1)` mengembalikan satu objek model berdasarkan primary key. Jika tidak ada data, akan terjadi error.



ID	Username	Nama	Level ID Pengguna
1	admin	Administrator	1

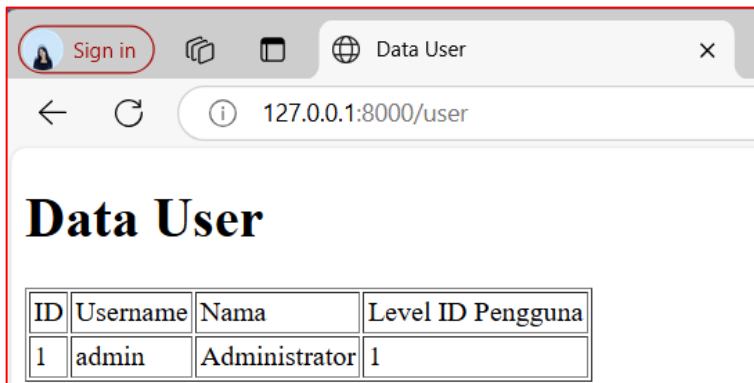
4. Ubah file `controller` dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::where('level_id', 1)->first();
        return view('user', ['data' => $user]);
    }
}
```

5. Simpan kode program Langkah 4. Kemudian jalankan pada *browser* dan amati apa yang

terjadi dan beri penjelasan dalam laporan

Hasil : Metode `where(...)->first()` digunakan untuk mengambil data pertama yang sesuai dengan kondisi yang diberikan.



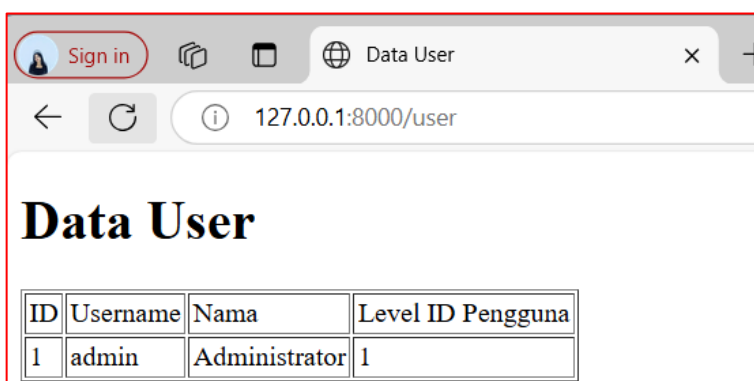
ID	Username	Nama	Level ID Pengguna
1	admin	Administrator	1

- Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstWhere('level_id', 1);
        return view('user', ['data' => $user]);
    }
}
```

- Simpan kode program Langkh 6. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Metode `firstOrFail()` akan melemparkan exception jika data tidak ditemukan, sehingga lebih baik digunakan saat data wajib ada.



ID	Username	Nama	Level ID Pengguna
1	admin	Administrator	1

Terkadang Anda mungkin ingin melakukan beberapa tindakan lain jika tidak ada hasil yang ditemukan. Metode `findOr` and `firstOr` akan mengembalikan satu contoh model atau, jika

tidak ada hasil yang ditemukan maka akan menjalankan didalam fungsi. Nilai yang dikembalikan oleh fungsi akan dianggap sebagai hasil dari metode ini:

```
$user = UserModel::findOr(1, function () {
    // ...
});

$user = UserModel::where('level_id', '>', 3)->firstOr(function () {
    // ...
});
```

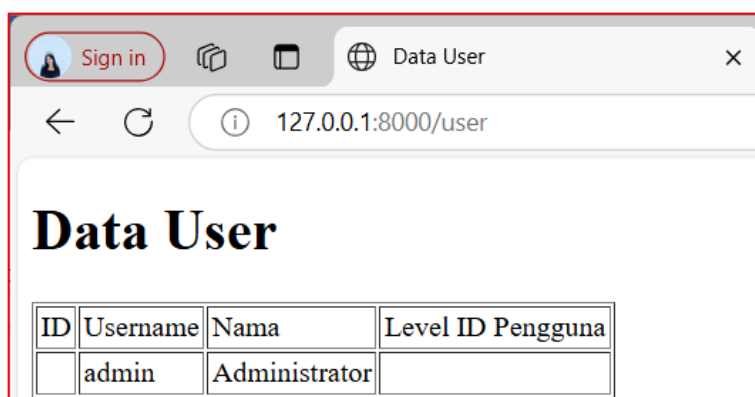
- Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::findOr(1, ['username', 'nama'], function () {
            abort(404);
        });

        return view('user', ['data' => $user]);
    }
}
```

- Simpan kode program Langkah 8. Kemudian pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Metode `findOr()` ini memberikan solusi alternatif ketika data tidak ditemukan dengan `abort 404`.



- Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```

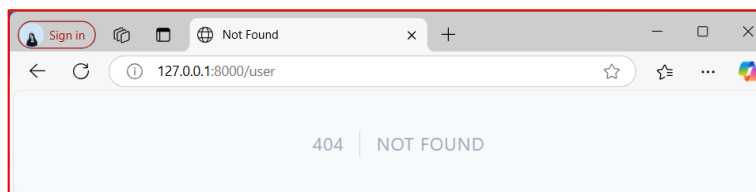
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::findOrFail(20, ['username', 'nama'], function () {
            abort(404);
        });

        return view('user', ['data' => $user]);
    }
}

```

11. Simpan kode program Langkah 10. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Inilah hasilnya apabila diberikan data ke 20, yang dimana pada database tidak ada. Sehingga hasil yang dikeluarkan metode `findOrFail` adalah 404 Not found.



12. Laporkan hasil Praktikum-2.1 ini dan *commit* perubahan pada *git*.

Praktikum 2.2 – Not Found Exceptions

Terkadang Anda mungkin ingin memberikan pengecualian jika model tidak ditemukan. Hal ini sangat berguna dalam *route* atau pengontrol. Metode `findOrFail` and `firstOrFail` akan mengambil hasil pertama dari kueri; namun, jika tidak ada hasil yang ditemukan, sebuah `Illuminate\Database\Eloquent\ModelNotFoundException` akan dilempar. Berikut ikuti langkah-langkah di bawah ini:

1. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```

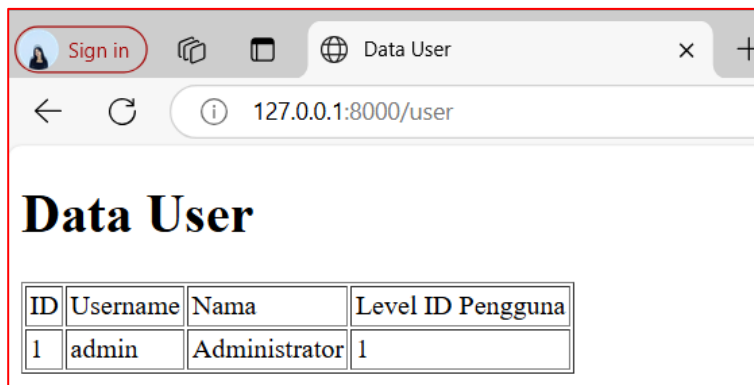
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::findOrFail(1);
        return view('user', ['data' => $user]);
    }
}

```

2. Simpan kode program Langkah 1. Kemudian jalankan pada *browser* dan amati apa yang

terjadi dan beri penjelasan dalam laporan

Hasil : `findOrFail(1)` mencoba mencari pengguna dengan ID 1. Jika user dengan ID 1 tidak ditemukan, maka Laravel akan melemparkan `ModelNotFoundException` dan menghasilkan halaman 404 Not Found.

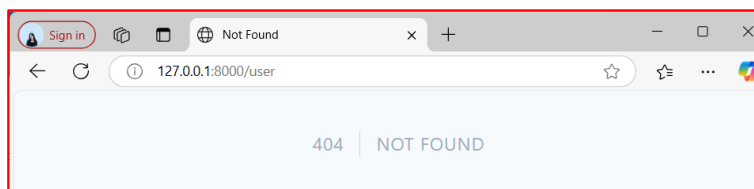


- Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::where('username', 'manager9')->firstOrFail();
        return view('user', ['data' => $user]);
    }
}
```

- Simpan kode program Langkah 3. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Laravel melempar `ModelNotFoundException` dan outputnya halaman 404 Not Found karena tidak ada username yang bernilai `manager9`.



- Laporkan hasil Praktikum-2.2 ini dan *commit* perubahan pada *git*.

Praktikum 2.3 – Retrieving Aggregates

Saat berinteraksi dengan model Eloquent, Anda juga dapat menggunakan metode agregat `count`, `sum`, `max`, dan lainnya yang disediakan oleh pembuat kueri Laravel. Seperti yang Anda duga, metode ini mengembalikan nilai skalar dan contoh model Eloquent:

```
$count = UserModel::where('active', 1)->count();

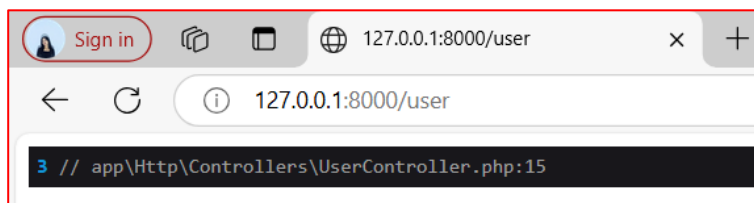
$max = UserModel::where('active', 1)->max('price');
```

1. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::where('level_id', 2)->count();
        dd($user);
        return view('user', ['data' => $user]);
    }
}
```

2. Simpan kode program Langkah 1. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

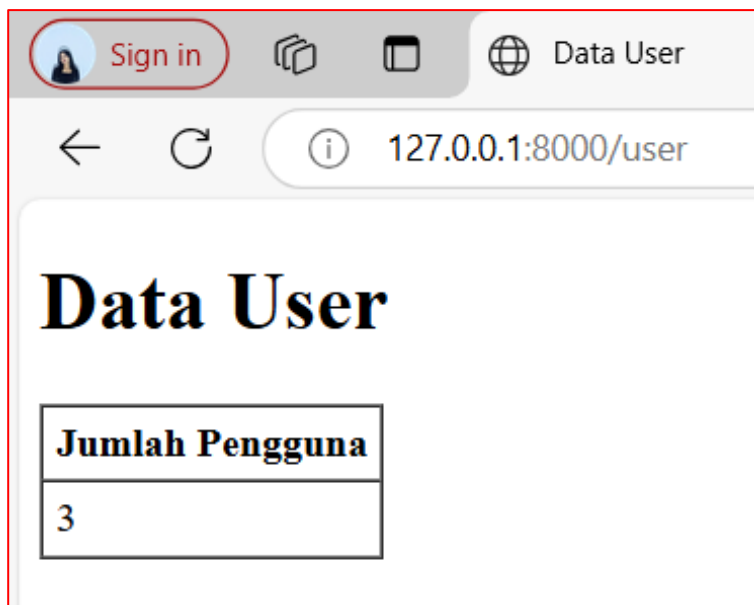
Hasil : Di Laravel, `dd()` (dump and die) akan langsung menghentikan eksekusi script setelah menampilkan nilai dari `$user`. Akibatnya, kode setelahnya (`return view(...)`) tidak akan pernah dijalankan.



Solusinya adalah dengan : `where('level_id', 2)->count()`; menghitung jumlah pengguna dengan `level_id = 2`. `$jumlahPengguna` dikirim ke view `user.blade.php`.

```
class UserController extends Controller
{
    Codeium: Refactor | Explain | Generate Function Comment | X
    public function index()
    {
        $jumlahPengguna = UserModel::where('level_id', 2)->count();
        return view('user', ['jumlahPengguna' => $jumlahPengguna]);
    }
}
```

```
resources > views > user.blade.php > ...
1 <!DOCTYPE html>
2 <html>
3 <head>
4 | <title>Data User</title>
5 </head>
6 <body>
7 | <h1>Data User</h1>
8 | <table border="1" cellpadding="5" cellspacing="0">
9 | | <tr>
10 | | | <td><b>Jumlah Pengguna</b></td>
11 | | </tr>
12 | | <tr>
13 | | | <td>{{ $jumlahPengguna }}</td>
14 | | </tr>
15 | </table>
16 </body>
17 </html>
```



3. Buat agar jumlah *script* pada langkah 1 bisa tampil pada halaman *browser*, sebagai contoh bisa lihat gambar di bawah ini dan ubah *script* pada file *view* supaya bisa muncul

Data User

Jumlah Pengguna
2

data

4. Laporkan hasil Praktikum-2.3 ini dan *commit* perubahan pada *git*.

Praktikum 2.4 – Retrieving or Creating Models

Metode `firstOrCreate` merupakan metode untuk melakukan *retrieving data* (mengambil data) berdasarkan nilai yang ingin dicari, jika data tidak ditemukan maka method ini akan melakukan insert ke table database tersebut sesuai dengan nilai yang dimasukkan.

Metode `firstOrCreate`, seperti `firstOrCreate`, akan mencoba menemukan/mengambil *record/data* dalam database yang cocok dengan atribut yang diberikan. Namun, jika data tidak ditemukan, data akan disiapkan untuk di-insert-kan ke database dan model baru akan dikembalikan. Perhatikan bahwa model yang dikembalikan `firstOrCreate` belum disimpan ke database. Anda perlu memanggil metode `save()` secara manual untuk menyimpannya:

```
$user = UserModel::firstOrCreate(
    [
        'username' => 'manager',
        'nama' => 'Manager',
    ],
);

$user = UserModel::firstOrCreate(
    [
        'username' => 'manager',
        'nama' => 'Manager',
    ],
);
```

1. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrCreate(
            [
                'username' => 'manager',
                'nama' => 'Manager',
            ],
        );

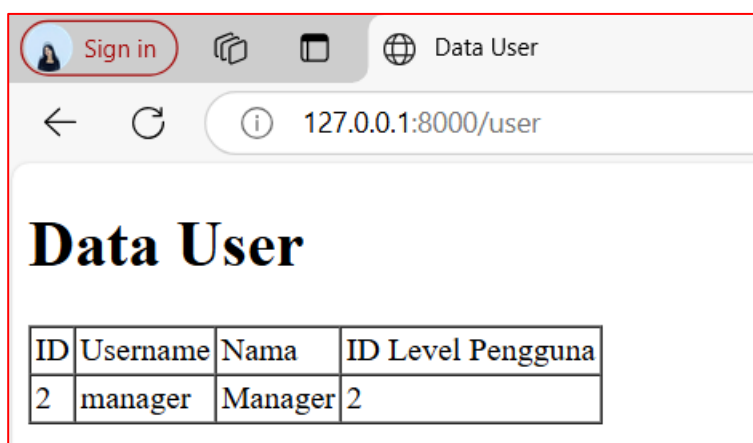
        return view('user', ['data' => $user]);
    }
}
```

- Ubah kembali file *view* dengan nama `user.blade.php` dan ubah *script* seperti gambar di bawah ini

```
<body>
<h1>Data User</h1>
<table border="1" cellpadding="2" cellspacing="0">
  <tr>
    <td>ID</td>
    <td>Username</td>
    <td>Nama</td>
    <td>ID Level Pengguna</td>
  </tr>
  <tr>
    <td>{{ $data->user_id }}</td>
    <td>{{ $data->username }}</td>
    <td>{{ $data->nama }}</td>
    <td>{{ $data->level_id }}</td>
  </tr>
</table>
</body>
```

- Simpan kode program Langkah 1 dan 2. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : kode ini mencoba mencari data berdasarkan username. Jika username manager belum ada, maka data akan otomatis ditambahkan ke database.



- Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini


```

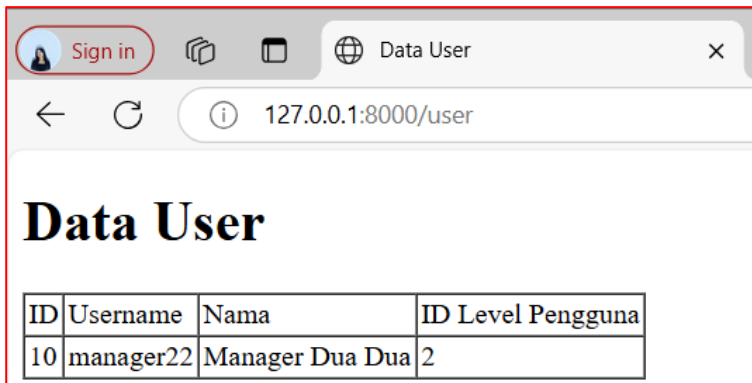
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrCreate(
            [
                'username' => 'manager22',
                'nama' => 'Manager Dua Dua',
                'password' => Hash::make('12345'),
                'level_id' => 2
            ],
        );

        return view('user', ['data' => $user]);
    }
}

```

5. Simpan kode program Langkah 4. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel `m_user` serta beri penjelasan dalam laporan

Hasil : Sama seperti sebelumnya, tetapi kali ini password dienkripsi sebelum disimpan. Jadi jika username sudah ada, data tetap dan tidak diperbarui. Jika username belum ada, data baru akan dibuat dengan password yang sudah di-hash. Password di database akan berbentuk string terenkripsi.



ID	Username	Nama	ID Level Pengguna
10	manager22	Manager Dua Dua	2

6. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```

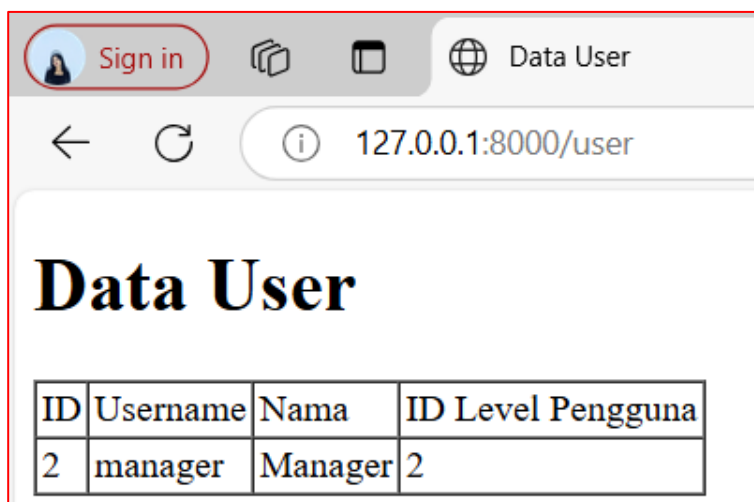
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrCreate(
            [
                'username' => 'manager',
                'nama' => 'Manager',
            ],
        );

        return view('user', ['data' => $user]);
    }
}

```

7. Simpan kode program Langkah 6. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Kode ini mencoba mencari data berdasarkan username, tetapi jika tidak ditemukan, data hanya akan dibuat sebagai objek tanpa disimpan ke database. Data hanya akan muncul di browser tetapi tidak akan masuk ke database karena belum dipanggil `$user->save();`.



8. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```

class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrCreate(
            [
                'username' => 'manager33',
                'nama' => 'Manager Tiga Tiga',
                'password' => Hash::make('12345'),
                'level_id' => 2
            ],
        );

        return view('user', ['data' => $user]);
    }
}

```

9. Simpan kode program Langkah 8. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel *m_user* serta beri penjelasan dalam laporan

Hasil : Data tentunya tidak akan masuk ke database karena hanya dibuat sebagai objek.

ID	Username	Nama	ID Level Pengguna
	manager33	Manager Tiga Tiga	2

	user_id	level_id	username	nama	password	created_at	updated_at
<input type="checkbox"/>	1	1	admin	Administrator	\$2y\$12\$hiAHE121M/oRe8npl5CBlec2ZbUeQOtzpl/k68tclOv...	NULL	NULL
<input type="checkbox"/>	2	2	manager	Manager	\$2y\$12\$8w/kP7UqjllqKHHwNlfe6Ei5U/heRCa0DWv8hqG2z...	NULL	NULL
<input type="checkbox"/>	3	3	staff	Staff/Kasir	\$2y\$12\$vLHKBa0t59XO0rq2PEX6D.kGX033YBi0n1M0ygKezPt...	NULL	NULL
<input type="checkbox"/>	8	2	manager_dua	Manager 2	\$2y\$12\$vBHHVqew0HbxUwk34PIOwu2fwJKR3.YKvm0Lxzy0PO...	2025-03-08 09:44:09	2025-03-08 09:44:09
<input type="checkbox"/>	9	2	manager_tiga	Manager 3	\$2y\$12\$wwxhESW8XYmWnpDh884IKO0ytAEyGo1hEyCJNJ34Get...	2025-03-08 13:09:58	2025-03-08 13:09:58
<input type="checkbox"/>	10	2	manager22	Manager Dua Dua	\$2y\$12\$it5GTx7stELnCAKGazJ50GuEutRYFu5hWV8/OVBPHr/A...	2025-03-08 17:15:18	2025-03-08 17:15:18

10. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```

class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrCreate(
            [
                'username' => 'manager33',
                'nama' => 'Manager Tiga Tiga',
                'password' => Hash::make('12345'),
                'level_id' => 2
            ],
        );
        $user->save();

        return view('user', ['data' => $user]);
    }
}

```

11. Simpan kode program Langkah 9. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel *m_user* serta beri penjelasan dalam laporan

Hasil : Dengan `$user->save();`, data benar-benar disimpan ke database.

	user_id	level_id	username	nama	password	created_at	updated_at
<input type="checkbox"/> Edit Copy Delete	1	1	admin	Administrator	\$2y\$12\$hiAHEt21M/oRe8npl5CBlec2ZbUeQOtzpI/k68tclOv...	NULL	NULL
<input type="checkbox"/> Edit Copy Delete	2	2	manager	Manager	\$2y\$12\$8w/kP7UqijllqKHtHwNife6EI5U/heRCa0DWv8hqG2z...	NULL	NULL
<input type="checkbox"/> Edit Copy Delete	3	3	staff	Staff/Kasir	\$2y\$12\$vLHKBa0t59XO0rq2PEX6D.kGX033YBi0n1M0ygKzPt...	NULL	NULL
<input type="checkbox"/> Edit Copy Delete	8	2	manager_dua	Manager 2	\$2y\$12\$vBHHVqew0HbxUwk34PIOwu2fwJKR3.YKvm0LxzpY0PO...	2025-03-08 09:44:09	2025-03-08 09:44:09
<input type="checkbox"/> Edit Copy Delete	9	2	manager_tiga	Manager 3	\$2y\$12\$wwxhESW8XYmWnpDh884IKO0ytAEyGo1hEyCJNJ34Get...	2025-03-08 13:09:58	2025-03-08 13:09:58
<input type="checkbox"/> Edit Copy Delete	10	2	manager22	Manager Dua Dua	\$2y\$12\$5GTX7stELnCAKGAzJ50GuEutRYFu5hVW8/OVBPhr/A...	2025-03-08 17:15:18	2025-03-08 17:15:18
<input type="checkbox"/> Edit Copy Delete	11	2	manager33	Manager Tiga Tiga	\$2y\$12\$kvf5fUyGan0c1whvb/XSSezXaFXcPm2x8Ykwh03cluy...	2025-03-08 17:24:42	2025-03-08 17:24:42

12. Laporkan hasil Praktikum-2.4 ini dan *commit* perubahan pada *git*.

Praktikum 2.5 – Attribute Changes

Eloquent menyediakan metode `isDirty`, `isClean`, dan `wasChanged` untuk memeriksa keadaan internal model Anda dan menentukan bagaimana atributnya berubah sejak model pertama kali diambil.

Metode `isDirty` menentukan apakah ada atribut model yang telah diubah sejak model diambil. Anda dapat meneruskan nama atribut tertentu atau serangkaian atribut ke metode `isDirty` untuk menentukan apakah ada atribut yang "kotor". Metode ini `isClean` akan menentukan apakah suatu atribut tetap tidak berubah sejak model diambil. Metode ini juga menerima argumen atribut opsional:

```
$user = UserModel::create([
    'username' => 'manager44',
    'nama' => 'Manager44',
    'password' => Hash::make('12345'),
    'level_id' => 2,
]);

$user->username = 'manager45';

$user->isDirty(); // true
$user->isDirty('username'); // true
$user->isDirty('nama'); // false
$user->isDirty(['nama', 'username']); // true

$user->isClean(); // false
$user->isClean('username'); // false
$user->isClean('nama'); // true
$user->isClean(['nama', 'username']); // false

$user->save();

$user->isDirty(); // false
$user->isClean(); // true
```

1. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```

class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::create([
            'username' => 'manager55',
            'nama' => 'Manager55',
            'password' => Hash::make('12345'),
            'level_id' => 2,
        ]);

        $user->username = 'manager56';

        $user->isDirty(); // true
        $user->isDirty('username'); // true
        $user->isDirty('nama'); // false
        $user->isDirty(['nama', 'username']); // true

        $user->isClean(); // false
        $user->isClean('username'); // false
        $user->isClean('nama'); // true
        $user->isClean(['nama', 'username']); // false

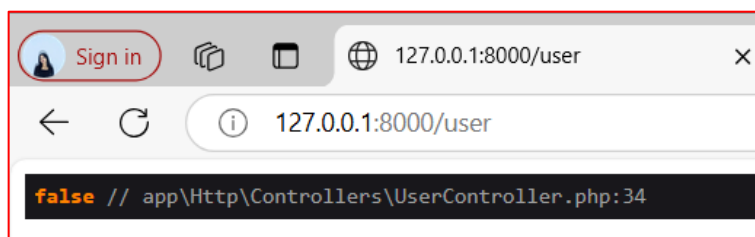
        $user->save();

        $user->isDirty(); // false
        $user->isClean(); // true
        dd($user->isDirty());
    }
}

```

2. Simpan kode program Langkah 1. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Pada awalnya, `$user->isDirty()`; bernilai `true` karena `username` diubah. setelah pemanggilan `$user->save()`;, perubahan disimpan ke database. Setelah penyimpanan, model tidak memiliki perubahan yang belum disimpan, sehingga `$user->isDirty()`; menjadi `false`. Karena `dd($user->isDirty())`; dipanggil setelah penyimpanan, hasil yang ditampilkan adalah `false`.



Metode ini `wasChanged` menentukan apakah ada atribut yang diubah saat model terakhir disimpan dalam siklus permintaan saat ini. Jika diperlukan, Anda dapat memberikan nama atribut untuk melihat apakah atribut tertentu telah diubah:

```

class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::create([
            'username' => 'manager11',
            'nama' => 'Manager11',
            'password' => Hash::make('12345'),
            'level_id' => 2,
        ]);

        $user->username = 'manager12';

        $user->save();

        $user->wasChanged(); // true
        $user->wasChanged('username'); // true
        $user->wasChanged(['username', 'level_id']); // true
        $user->wasChanged('nama'); // false
        $user->wasChanged(['nama', 'username']); // true
    }
}

```

- Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```

class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::create([
            'username' => 'manager11',
            'nama' => 'Manager11',
            'password' => Hash::make('12345'),
            'level_id' => 2,
        ]);

        $user->username = 'manager12';

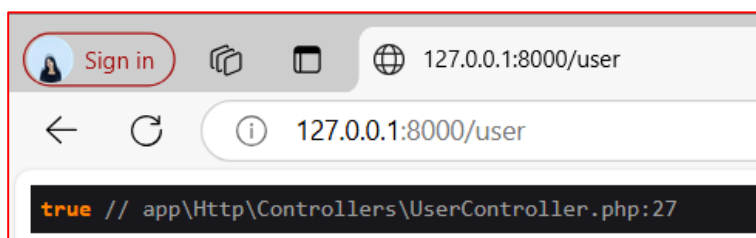
        $user->save();

        $user->wasChanged(); // true
        $user->wasChanged('username'); // true
        $user->wasChanged(['username', 'level_id']); // true
        $user->wasChanged('nama'); // false
        dd($user->wasChanged(['nama', 'username'])); // true
    }
}

```

- Simpan kode program Langkah 3. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Mengecek apakah suatu atribut berubah setelah model disimpan ke database. Jika atribut mengalami perubahan, metode ini mengembalikan true setelah `save()`. Pada kode ini, username berubah, sehingga `wasChanged('username')` akan true.



5. Laporkan hasil Praktikum-2.5 ini dan *commit* perubahan pada *git*.

Praktikum 2.6 – Create, Read, Update, Delete (CRUD)

Seperti yang telah kita ketahui, CRUD merupakan singkatan dari *Create*, *Read*, *Update* dan *Delete*. CRUD merupakan istilah untuk proses pengolahan data pada database, seperti input data ke database, menampilkan data dari database, mengedit data pada database dan menghapus data dari database. Ikuti langkah-langkah di bawah ini untuk melakukan CRUD dengan Eloquent

1. Buka file *view* pada *user.blade.php* dan buat scriptnya menjadi seperti di bawah ini

```
<body>
  <h1>Data User</h1>
  <a href="/user/tambah">+ Tambah User</a>
  <table border="1" cellpadding="2" cellspacing="0">
    <tr>
      <td>ID</td>
      <td>Username</td>
      <td>Nama</td>
      <td>ID Level Pengguna</td>
      <td>Aksi</td>
    </tr>
    @foreach ($data as $d)
      <tr>
        <td>{{ $d->user_id }}</td>
        <td>{{ $d->username }}</td>
        <td>{{ $d->nama }}</td>
        <td>{{ $d->level_id }}</td>
        <td><a href="/user/ubah/{{ $d->user_id }}">Ubah</a> | <a href="/user/hapus/{{ $d->user_id }}">Hapus</a></td>
      </tr>
    @endforeach
  </table>
</body>
```

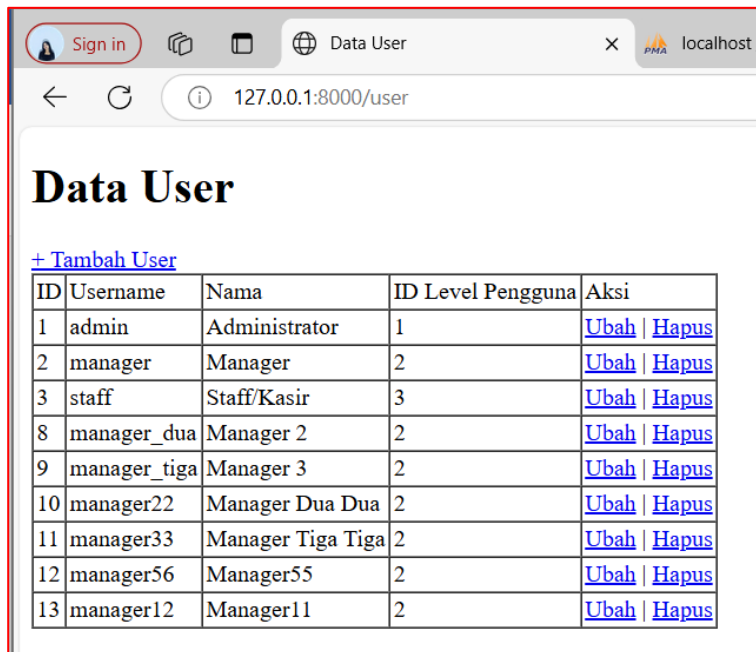
2. Buka file controller pada *UserController.php* dan buat scriptnya untuk *read* menjadi seperti di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::all();
        return view('user', ['data' => $user]);
    }
}
```

3. Simpan kode program Langkah 1 dan 2. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Menggunakan perulangan `@foreach($data as $d)` untuk menampilkan semua user. Terdapat tombol Ubah dan Hapus untuk mengedit atau menghapus

user. Method `index()` di dalam `UserController.php` bertugas untuk mengambil semua data user dari database menggunakan `UserModel::all()` dan mengirimkannya ke view `user.blade.php`.



Data User

[+ Tambah User](#)

ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	Ubah Hapus
2	manager	Manager	2	Ubah Hapus
3	staff	Staff/Kasir	3	Ubah Hapus
8	manager_dua	Manager 2	2	Ubah Hapus
9	manager_tiga	Manager 3	2	Ubah Hapus
10	manager22	Manager Dua Dua	2	Ubah Hapus
11	manager33	Manager Tiga Tiga	2	Ubah Hapus
12	manager56	Manager55	2	Ubah Hapus
13	manager12	Manager11	2	Ubah Hapus

4. Langkah berikutnya membuat *create* atau tambah data user dengan cara bikin file baru pada view dengan nama `user_tambah.blade.php` dan buat scriptnya menjadi seperti

```
<body>
  <h1>Form Tambah Data User</h1>
  <form method="post" action="/user/tambah_simpan">

    {{ csrf_field() }}

    <label>Username</label>
    <input type="text" name="username" placeholder="Masukan Username">
    <br>
    <label>Nama</label>
    <input type="text" name="nama" placeholder="Masukan Nama">
    <br>
    <label>Password</label>
    <input type="password" name="password" placeholder="Masukan Password">
    <br>
    <label>Level ID</label>
    <input type="number" name="level_id" placeholder="Masukan ID Level">
    <br><br>
    <input type="submit" class="btn btn-success" value="Simpan">

  </form>
</body>
```

5. Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

```
Route::get('/user/tambah', [UserController::class, 'tambah']);
```

6. Tambahkan *script* pada controller dengan nama file `UserController.php`.

Tambahkan *script* dalam class dan buat method baru dengan nama tambah dan diletakan di bawah method index seperti gambar di bawah ini

```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::all();
        return view('user', ['data' => $user]);
    }

    public function tambah()
    {
        return view('user_tambah');
    }
}
```

7. Simpan kode program Langkah 4 s/d 6. Kemudian jalankan pada *browser* dan klik link “+ **Tambah User**” amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Membuat halaman form untuk menambahkan user baru. Form ini akan mengirimkan data ke method tambah_simpan di UserController.php. kemudian disini ada tambah route Ketika user mengklik tombol + Tambah User, mereka akan diarahkan ke halaman form tambah user. Namun disini belum disimpan karena pada route tidak ada untuk menangani form submission yaitu pake POST

8. Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

```
Route::post('/user/tambah_simpan', [UserController::class, 'tambah_simpan']);
```

9. Tambahkan *script* pada controller dengan nama file `UserController.php`. Tambahkan *script* dalam class dan buat method baru dengan nama tambah_simpan dan diletakan di bawah method tambah seperti gambar di bawah ini

```

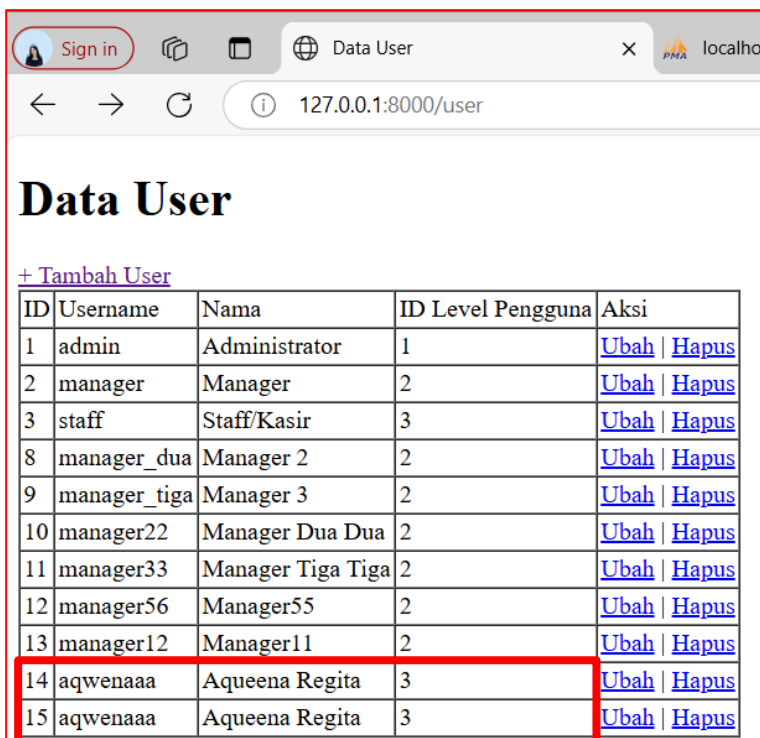
public function tambah_simpan(Request $request)
{
    UserModel::create([
        'username' => $request->username,
        'nama' => $request->nama,
        'password' => Hash::make('$request->password'),
        'level_id' => $request->level_id
    ]);

    return redirect('/user');
}

```

10. Simpan kode program Langkah 8 dan 9. Kemudian jalankan link <localhost:8000/user/tambah> atau localhost/PWL_POS/public/user/tambah pada browser dan input formnya dan simpan, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Route POST ini akan memproses form tambah user ketika tombol Simpan ditekan. Data yang diinputkan user akan disimpan ke database menggunakan Eloquent ORM. Password akan dienkripsi menggunakan Hash::make(). Setelah berhasil disimpan, pengguna akan diarahkan kembali ke halaman daftar user.



ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	Ubah Hapus
2	manager	Manager	2	Ubah Hapus
3	staff	Staff/Kasir	3	Ubah Hapus
8	manager_dua	Manager 2	2	Ubah Hapus
9	manager_tiga	Manager 3	2	Ubah Hapus
10	manager22	Manager Dua Dua	2	Ubah Hapus
11	manager33	Manager Tiga Tiga	2	Ubah Hapus
12	manager56	Manager55	2	Ubah Hapus
13	manager12	Manager11	2	Ubah Hapus
14	aqwenaaa	Aqueena Regita	3	Ubah Hapus
15	aqwenaaa	Aqueena Regita	3	Ubah Hapus

11. Langkah berikutnya membuat *update* atau ubah data user dengan cara bikin file baru pada *view* dengan nama [user_ubah.blade.php](#) dan buat scriptnya menjadi seperti di bawah ini

```

<body>
<h1>Form Ubah Data User</h1>
<a href="/user">Kembali</a>
<br><br>

<form method="post" action="/user/ubah_simpan/{{ $data->user_id }}">

    {{ csrf_field() }}
    {{ method_field('PUT') }}

    <label>Username</label>
    <input type="text" name="username" placeholder="Masukan Username" value="{{ $data->username }}">
    <br>
    <label>Nama</label>
    <input type="text" name="nama" placeholder="Masukan Nama" value="{{ $data->username }}">
    <br>
    <label>Password</label>
    <input type="password" name="password" placeholder="Masukan Password" value="{{ $data->password }}">
    <br>
    <label>Level ID</label>
    <input type="number" name="level_id" placeholder="Masukan ID Level" value="{{ $data->level_id }}">
    <br><br>
    <input type="submit" class="btn btn-success" value="Ubah">

</form>
</body>

```

12. Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

```
Route::get('/user/ubah/{id}', [UserController::class, 'ubah']);
```

13. Tambahkan *script* pada *controller* dengan nama file `UserController.php`. Tambahkan *script* dalam class dan buat method baru dengan nama `ubah` dan diletakkan di bawah method `tambah_simpan` seperti gambar di bawah ini

```

public function ubah($id)
{
    $user = UserModel::find($id);
    return view('user_ubah', ['data' => $user]);
}

```

14. Simpan kode program Langkah 11 sd 13. Kemudian jalankan pada *browser* dan klik link “Ubah” amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Menggunakan method POST dengan PUT sebagai metode HTTP untuk mengupdate data yang sudah ada. Namun hingga langkah 14 ini belum bisa untuk disimpan dalam database karena belum ada form acceptance untuk ubah data.

Sign in Ubah Data User

127.0.0.1:8000/user/ubah/14

Form Ubah Data User

Username

Nama

Password

Level ID

15. Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

```
Route::put('/user/ubah_simpan/{id}', [UserController::class, 'ubah_simpan']);
```

16. Tambahkan *script* pada *controller* dengan nama file `UserController.php`. Tambahkan *script* dalam class dan buat method baru dengan nama `ubah_simpan` dan diletakan di bawah method `ubah` seperti gambar di bawah ini

```
public function ubah_simpan($id, Request $request)
{
    $user = UserModel::find($id);

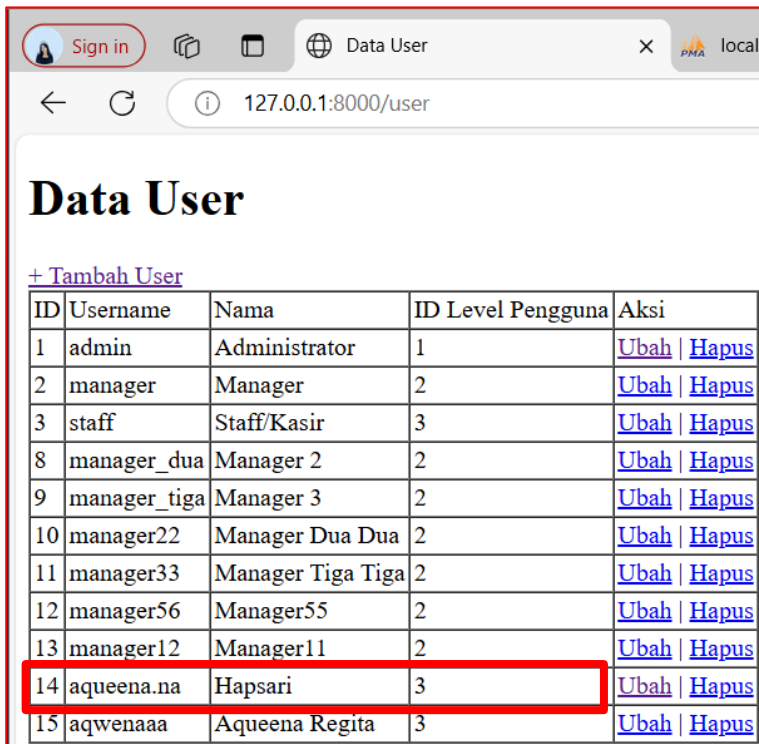
    $user->username = $request->username;
    $user->nama = $request->nama;
    $user->password = Hash::make('$request->password');
    $user->level_id = $request->level_id;

    $user->save();

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

17. Simpan kode program Langkah 15 dan 16. Kemudian jalankan link `localhost:8000/user/ubah/1` atau `localhost/PWL_POS/public/user/ubah/1` pada *browser* dan ubah input formnya dan klik tombol `ubah`, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil :



ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	Ubah Hapus
2	manager	Manager	2	Ubah Hapus
3	staff	Staff/Kasir	3	Ubah Hapus
8	manager_dua	Manager 2	2	Ubah Hapus
9	manager_tiga	Manager 3	2	Ubah Hapus
10	manager22	Manager Dua Dua	2	Ubah Hapus
11	manager33	Manager Tiga Tiga	2	Ubah Hapus
12	manager56	Manager55	2	Ubah Hapus
13	manager12	Manager11	2	Ubah Hapus
14	aqueena.na	Hapsari	3	Ubah Hapus
15	aqwenaaa	Aqueena Regita	3	Ubah Hapus

18. Berikut untuk langkah *delete* . Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

```
Route::get('/user/hapus/{id}', [UserController::class, 'hapus']);
```

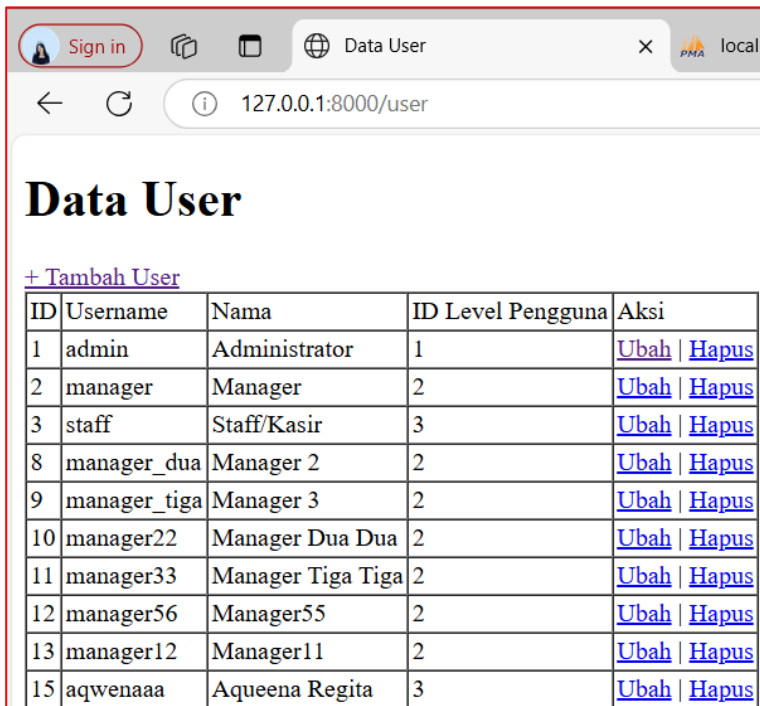
19. Tambahkan *script* pada controller dengan nama file `UserController.php`. Tambahkan *script* dalam class dan buat method baru dengan nama hapus dan diletakan di bawah method `ubah_simpan` seperti gambar di bawah ini

```
public function hapus($id)
{
    $user = UserModel::find($id);
    $user->delete();

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

20. Simpan kode program Langkah 18 dan 19. Kemudian jalankan pada *browser* dan klik tombol hapus, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Menambahkan rute GET untuk menangani permintaan penghapusan user berdasarkan ID. Memanggil method `hapus()` pada `UserController`. Data user `aqueena.na` berhasil dihapus dan halaman otomatis kembali ke daftar user.



ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	Ubah Hapus
2	manager	Manager	2	Ubah Hapus
3	staff	Staff/Kasir	3	Ubah Hapus
8	manager_dua	Manager 2	2	Ubah Hapus
9	manager_tiga	Manager 3	2	Ubah Hapus
10	manager22	Manager Dua Dua	2	Ubah Hapus
11	manager33	Manager Tiga Tiga	2	Ubah Hapus
12	manager56	Manager55	2	Ubah Hapus
13	manager12	Manager11	2	Ubah Hapus
15	aqwenaaa	Aqueena Regita	3	Ubah Hapus

21. Laporkan hasil Praktikum-2.6 ini dan *commit* perubahan pada *git*.

Praktikum 2.7 – Relationships

One to One

Hubungan satu-ke-satu adalah tipe hubungan database yang sangat mendasar. Misalnya, suatu `Usermodel` mungkin dikaitkan dengan satu model `Levelmodel`. Untuk mendefinisikan hubungan ini, kita akan menempatkan `Levelmodel` metode pada model `Usermodel`. Metode tersebut `Levelmodel` harus memanggil `hasOne` metode tersebut dan mengembalikan hasilnya. Metode ini `hasOne` tersedia untuk model Anda melalui kelas dasar model `Illuminate\Database\Eloquent\Model`:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasOne;

You, 1 second ago | 1 author (You)
class UserModel extends Model
{
    public function level(): HasOne
    {
        return $this->hasOne(LevelModel::class);
    }
}
```

Mendefinisikan Kebalikan dari Hubungan *One-to-one*

Jadi, kita dapat mengakses model `Levelmodel` dari model `Usermodel` kita. Selanjutnya, mari kita tentukan hubungan pada model `Levelmodel` yang memungkinkan kita mengakses user. Kita dapat mendefinisikan kebalikan dari suatu `hasOne` hubungan menggunakan `belongsTo` metode:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class LevelModel extends Model
{
    public function user(): BelongsTo
    {
        return $this->belongsTo(UserModel::class);
    }
}
```

One to Many

Hubungan satu-ke-banyak digunakan untuk mendefinisikan hubungan di mana satu model adalah induk dari satu atau lebih model turunan. Misalnya, 1 kategori mungkin memiliki jumlah barang yang tidak terbatas. Seperti semua hubungan Eloquent lainnya, hubungan satu-ke-banyak ditentukan dengan mendefinisikan metode pada model Eloquent Anda:

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class KategoriModel extends Model
{
    public function barang(): HasMany
    {
        return $this->hasMany(BarangModel::class, 'barang_id', 'barang_id');
    }
}

```

One to Many (Inverse) / Belongs To

Sekarang kita dapat mengakses semua barang, mari kita tentukan hubungan agar barang dapat mengakses kategori induknya. Untuk menentukan invers suatu `hasMany` hubungan, tentukan metode hubungan pada model anak yang memanggil `belongsTo` tersebut:

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class BarangModel extends Model
{
    public function kategori(): BelongsTo
    {
        return $this->belongsTo(KategoriModel::class, 'kategori_id', 'kategori_id');
    }
}

```

1. Buka file model pada `UserModel.php` dan tambahkan scripnya menjadi seperti di bawah ini

```

class UserModel extends Model
{
    use HasFactory;

    protected $table = 'm_user';
    protected $primaryKey = 'user_id';
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = ['level_id', 'username', 'nama', 'password'];

    public function level(): BelongsTo
    {
        return $this->belongsTo(LevelModel::class, 'level_id', 'level_id');
    }
}

```

2. Buka file controller pada `UserController.php` dan ubah method *script* menjadi seperti di bawah ini

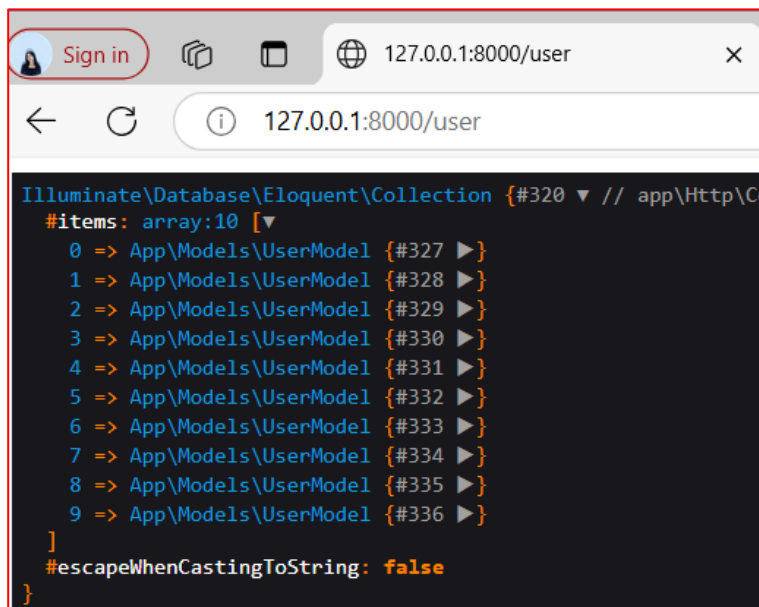
```

public function index()
{
    $user = UserModel::with('level')->get();
    dd($user);
}

```

3. Simpan kode program Langkah 2. Kemudian jalankan link pada *browser*, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : Model UserModel memiliki relasi belongsTo dengan LevelModel, yang menghubungkan level_id di UserModel dengan level_id di LevelModel. Dengan ini, kita dapat mengambil data level dari setiap user melalui UserModel. UserModel::with('level')->get(); digunakan untuk mengambil semua data user beserta data level yang terkait. dd(\$user); digunakan untuk debugging, yaitu menampilkan hasil query dalam bentuk array di browser. Relasi berhasil diterapkan karena data user yang diambil kini juga memiliki informasi level



4. Buka file controller pada `UserController.php` dan ubah method `script` menjadi seperti di bawah ini

```

public function index()
{
    $user = UserModel::with('level')->get();
    return view('user', ['data' => $user]);
}

```

5. Buka file view pada `user.blade.php` dan ubah `script` menjadi seperti di bawah ini

```

<body>
<h1>Data User</h1>
<a href="/user/tambah">+ Tambah User</a>
<table border="1" cellpadding="2" cellspacing="0">
  <tr>
    <td>ID</td>
    <td>Username</td>
    <td>Nama</td>
    <td>ID Level Pengguna</td>
    <td>Kode Level</td>
    <td>Nama Level</td>
    <td>Aksi</td>
  </tr>
  @foreach ($data as $d)
    <tr>
      <td>{{ $d->user_id }}</td>
      <td>{{ $d->username }}</td>
      <td>{{ $d->nama }}</td>
      <td>{{ $d->level_id }}</td>
      <td>{{ $d->level->level_kode }}</td>
      <td>{{ $d->level->level_nama }}</td>
      <td><a href="/user/ubah/{{ $d->user_id }}">Ubah</a> | <a href="/user/hapus/{{ $d->user_id }}">Hapus</a></td>
    </tr>
  @endforeach
</table>
</body>

```

6. Simpan kode program Langkah 4 dan 5. Kemudian jalankan link pada browser, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan

Hasil : @foreach(\$data as \$user) digunakan untuk menampilkan semua data user. {{ \$user->level->level_name }} mengambil nama level dari relasi level. Data user kini ditampilkan dengan informasi levelnya di halaman web. Relasi BelongsTo

antara UserModel dan LevelModel bekerja dengan baik.

Data User

[+ Tambah User](#)

ID	Username	Nama	ID Level Pengguna	Kode Level	Nama Level	Aksi
1	admin	Administrator	1	ADM	Administrator	Ubah Hapus
2	manager	Manager	2	MNG	Manager	Ubah Hapus
3	staff	Staff/Kasir	3	STF	Staff/Kasir	Ubah Hapus
8	manager_dua	Manager 2	2	MNG	Manager	Ubah Hapus
9	manager_tiga	Manager 3	2	MNG	Manager	Ubah Hapus
10	manager22	Manager Dua Dua	2	MNG	Manager	Ubah Hapus
11	manager33	Manager Tiga Tiga	2	MNG	Manager	Ubah Hapus
12	manager56	Manager55	2	MNG	Manager	Ubah Hapus
13	manager12	Manager11	2	MNG	Manager	Ubah Hapus
15	aqwenaaa	Aqueena Regita	3	STF	Staff/Kasir	Ubah Hapus

7. Laporkan hasil Praktikum-2.7 ini dan *commit* perubahan pada *git*.