



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

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 pertemuan 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

```
Week4 > JS4 > app > Models > UserModel.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  6 references | 0 implementations
9  class UserModel extends Model
10 {
11     use HasFactory;
12
13     0 references
14     protected $table = 'm_user'; // Mendefinisikan nama tabel yang digunakan oleh model ini
15     0 references
16     protected $primaryKey = 'user_id'; // Mendefinisikan primary key dari tabel yang digunakan
17
18     0 references
19     protected $fillable = ['level_id', 'username', 'nama', 'password'];
20 }
```

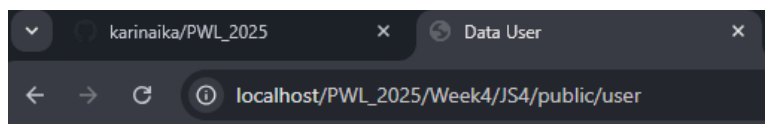
2. Buka file controller dengan nama `UserController.php` dan ubah *script* untuk menambahkan data baru seperti gambar di bawah ini



```
Week4 > JS4 > app > Http > Controllers > UserController.php > PHP Intelephense > UserController > index
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Models\UserModel;
6 use Illuminate\Http\Request;
7 use Illuminate\Support\Facades\Hash;
8
9 class UserController extends Controller
10 {
11     public function index(): Factory|View
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         // akses model UserModel
22         $user = UserModel::all(); // ambil semua data dari tabel m_user
23
24         return view('user', data: ['data' => $user]);
25     }
26 }
```

3. Simpan kode program Langkah 1 dan 2, dan jalankan perintah web server. Kemudian jalankan link [localhostPWL_POS/public/user](localhost/PWL_POS/public/user) pada *browser* dan amati apa yang terjadi

Jawab: Maka data akan bertambah



Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1
2	manager	Manager	2
3	staff	Staff/Kasir	3
5	customer-1	Pelanggan Pertama	4
6	manager_dua	Manager 2	2

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

```
0 references
protected $fillable = ['level_id', 'username', 'nama'];
// protected $fillable = ['level_id', 'username', 'nama', 'password'];
```

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



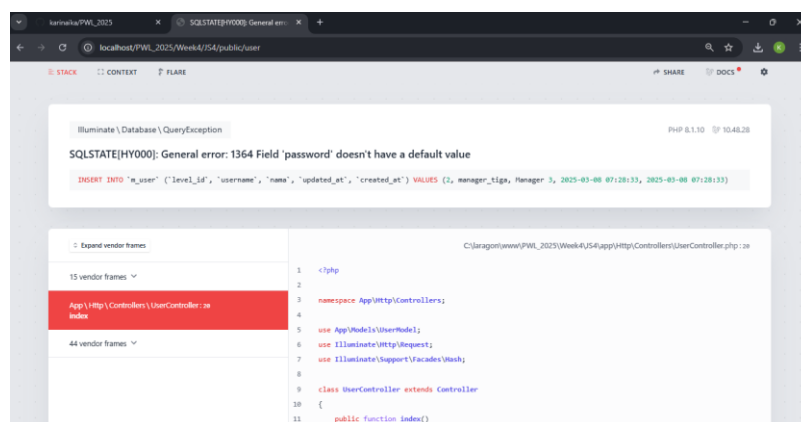
```
class UserController extends Controller
{
    2 references | 0 overrides
    public function index(): Factory|View
    {
        // tambah data user dengan Eloquent Model
        $data = [
            'level_id' => 2,
            'username' => 'manager_tiga', // 'username' => 'manager_dua',
            'nama' => 'Manager 3',
            'password' => Hash::make('12345')
        ];
        UserModel::create($data);

        // akses model UserModel
        $user = UserModel::all(); // ambil semua data dari tabel m_user

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

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

Jawab: terjadi error, karena kolom **password** tidak diizinkan dalam **\$fillable** di **UserModel.php** sehingga akan muncul **MassAssignment**.



- Solusinya bisa menambahkan kolom **password** atau dengan menggunakan **\$guarded** untuk mengizinkan semua kolom.

```
// protected $fillable = ['level_id', 'username', 'nama'];
// protected $fillable = ['level_id', 'username', 'nama', 'password'];
0 references
protected $guarded = [];
```

localhost/PWL_2025/Week4/JS4/public/user

Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1
2	manager	Manager	2
3	staff	Staff Kasir	3
5	customer-1	Pelanggan Pertama	4
6	manager_dua	Manager 2	2
7	manager_tiga	Manager 3	2



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

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

```
0 references | 0 implementations
class UserController extends Controller
{
    2 references | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::find(1);
        return view('user', data: ['data' => $user]);
    }
}
```



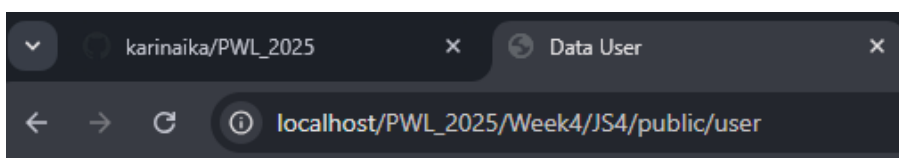
2. Buka file *view* dengan nama `user.blade.php` dan ubah *script* seperti gambar di bawah ini

```
8 <body>
9     <h1>Data User</h1>
10    <table border="1" cellpadding="2" cellspacing="0">
11        <tr>
12            <th>ID</th>
13            <th>Username</th>
14            <th>Nama</th>
15            <th>ID Level Pengguna</th>
16        </tr>
17        <tr>
18            <td>{{ $data->user_id }}</td>
19            <td>{{ $data->username }}</td>
20            <td>{{ $data->nama }}</td>
21            <td>{{ $data->level_id }}</td>
22        </tr>
23    </table>
24 </body>
25
26 </html>
```

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

Jawab: Kode tersebut akan mencari data user dengan `user_id = 1` dari tabel `m_user`.

Fungsi `UserModel::find(1)`; digunakan untuk mencari data dengan `user_id = 1`.



Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1

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

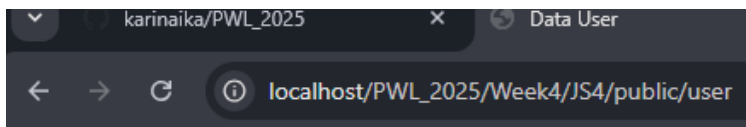
```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::where('level_id', 1)->first();
        return view('user', ['data' => $user]);
    }
}
```



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

Jawab: Kode tersebut akan mencari **satu user pertama** yang memiliki `level_id = 1` dalam tabel **m_user**. Fungsi `UserModel::where('level_id', 1)->first();`

Jika ada user dengan `level_id = 1`, data user pertama yang ditemukan akan ditampilkan di tabel.



Data User

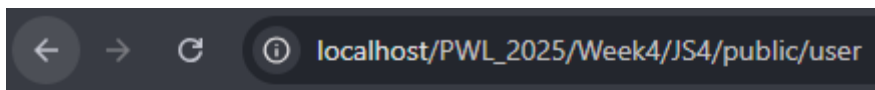
ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1

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::firstWhere('level_id', 1);
        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

Jawab: Kode ini menggunakan `firstWhere()` untuk mengambil satu baris data pertama dari tabel `m_user` dengan kondisi `level_id = 1`. Jika ada user dengan `level_id = 1`, data user pertama yang ditemukan akan dikirim ke `view('user')` dan ditampilkan di tabel.



Data User

ID	Username	Nama	ID Level 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:

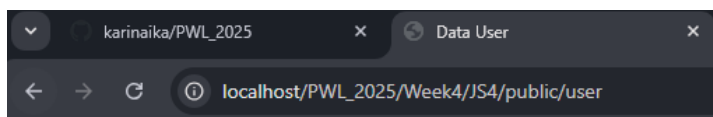
```
$user = UserModel::findOr(1, function () {  
    // ...  
});  
  
$user = UserModel::where('level_id', '>', 3)->firstOr(function () {  
    // ...  
});
```

- 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

Jawab: Terlihat bahwa kolom ID dan ID Level Pengguna **tidak muncul**. Metode `findOr` akan mengembalikan satu contoh model atau, jika tidak ada hasil yang ditemukan maka akan menjalankan didalam fungsi.



Data User

ID	Username	Nama	ID Level Pengguna
	admin	Administrator	

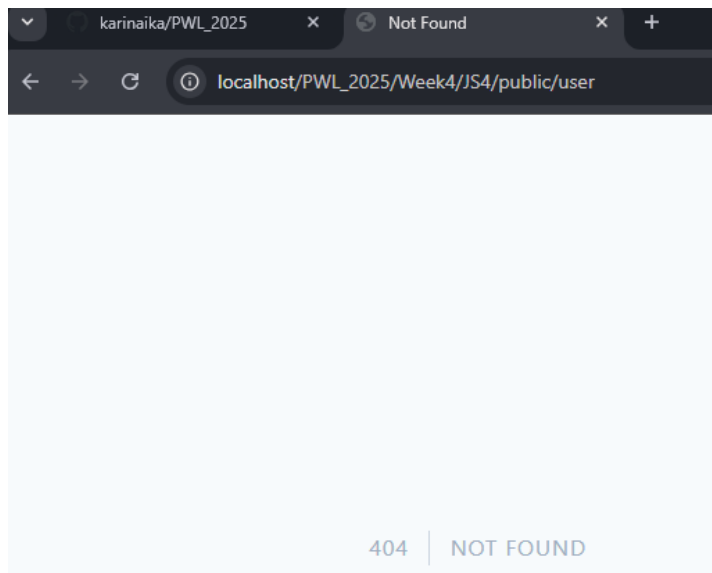
- Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini



```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::findOr(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

Jawab: id = 20 tidak ada dalam database, akan muncul **error 404** dengan metode yang benar.



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:

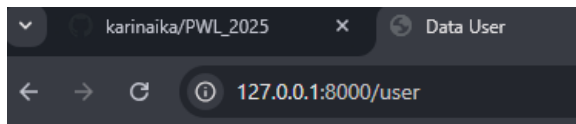
1. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini



```
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

Jawab: Jika id = 1 ada didatabase, data akan dikembalikan. Jika tidak ada maka laravel akan memunculkan ModelNotFoundException dan menampilkan halaman error 404.



Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1

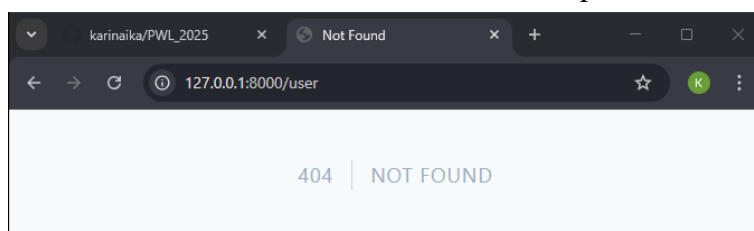
3. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

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

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

Jawab:

- `firstOrFail()` mengambil satu baris data sesuai kondisi.
- Jika data ada, maka akan ditampilkan.
- Jika data tidak ada, Laravel otomatis menampilkan error 404.





5. 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:

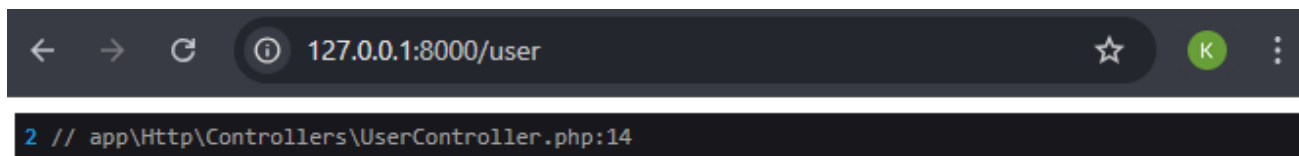
```
$count = UserModel::where('active', 1)->count();  
$max = UserModel::where('active', 1)->max('price');
```

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

Jawab: Kode akan menampilkan jumlah user dengan `level_id = 2`. Fungsi `dd($user);` (dump and die) akan menghentikan eksekusi program, sehingga Laravel tidak akan melanjutkan proses hingga `return view()`.



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 datanya

Data User

Jumlah Pengguna
2



Jawab:

```
$user = UserModel::where('level_id', 2)->count();  
//dd($user);  
return view(view: 'user', data: ['data' => $user]);
```

```
Week4 > JS4 > resources > views > user.blade.php > html > body  
1 <!DOCTYPE html>  
2 <html>  
3  
4 <head>  
5 <meta charset="UTF-8">  
6 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
7 <title>Data User</title>  
8 <style>  
9     table {  
10         border-collapse: collapse;  
11         width: 200px;  
12     }  
13  
14     th, td {  
15         border: 1px solid black;  
16         padding: 8px;  
17         text-align: center;  
18     }  
19 </style>  
20 </head>  
21  
22 <body>  
23 <h1>Data User</h1>  
24 <table>  
25 <tr>  
26 <th>Jumlah Pengguna</th>  
27 </tr>  
28 <tr>  
29 <td>{{ $data }}</td>  
30 </tr>  
31 </table>  
32 </body>
```

← → ↻ ⓘ 127.0.0.1:8000/user

Data User

Jumlah Pengguna
2

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]);
    }
}
```

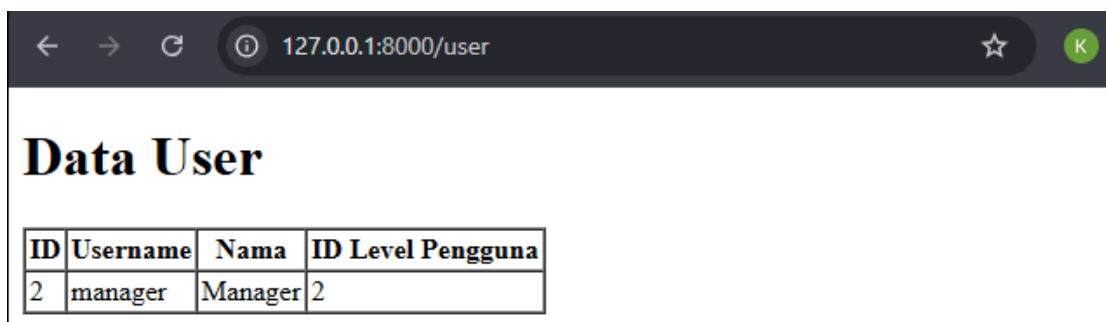


2. Ubah kembali file *view* dengan nama **user.blade.php** dan ubah *script* seperti gambar di bawah ini

```
<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>
```

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

Jawab: Fungsi `firstOrCreate()` digunakan untuk mencari data berdasarkan kondisi yang diberikan. Jika data ditemukan, maka akan dikembalikan. Jika tidak ditemukan, Laravel akan otomatis membuat data baru dengan nilai yang diberikan.



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::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

Jawab: Mengecek apakah ada data dengan username = 'manager22' di tabel `m_user`. Jika **ada**, maka data yang sudah ada akan dikembalikan (tidak adac perubahan pada database). Jika **tidak ada**, Laravel akan membuat entri baru dengan data yang diberikan.

Data User

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

- Sebelum

		user_id	level_id	username	nama	password	created_at	updated_at
<input type="checkbox"/>	Edit Copy Delete	1	1	admin	Administrator	\$2y\$12\$Tbat8iftyc/lcvz0ihgAFerHII5K47QNS6imzEcluzX...	NULL	NULL
<input type="checkbox"/>	Edit Copy Delete	2	2	manager	Manager	\$2y\$12\$gQBSTAVxloo/EuHvLNafseOjcBHMxk9b6BplrwpuDaW...	NULL	NULL
<input type="checkbox"/>	Edit Copy Delete	3	3	staff	Staff/Kasir	\$2y\$12\$2Fwun3kRudRY5PR.mdYvFOp6lqvR2zbx0.OGV5F.P2...	NULL	NULL
<input type="checkbox"/>	Edit Copy Delete	5	4	customer-1	Pelanggan Pertama	\$2y\$12\$vmD3coXUTDf.bzr9lggFwe9ZdvZETetqzkeYGeMvZrC...	NULL	2025-03-04 05:...
<input type="checkbox"/>	Edit Copy Delete	6	2	manager_dua	Manager 2	\$2y\$12\$1TW5LH6TTgza6S.wBMm1DuOYKJGMWLeuKEgjtLjrZ2...	2025-03-08 07:20:17	2025-03-08 07:...

- Sesudah

		user_id	level_id	username	nama	password	created_at	updated_at
<input type="checkbox"/>	Edit Copy Delete	1	1	admin	Administrator	\$2y\$12\$Tbat8iftyc/lcvz0ihgAFerHII5K47QNS6imzEcluzX...	NULL	NULL
<input type="checkbox"/>	Edit Copy Delete	2	2	manager	Manager	\$2y\$12\$gQBSTAVxloo/EuHvLNafseOjcBHMxk9b6BplrwpuDaW...	NULL	NULL
<input type="checkbox"/>	Edit Copy Delete	3	3	staff	Staff/Kasir	\$2y\$12\$2Fwun3kRudRY5PR.mdYvFOp6lqvR2zbx0.OGV5F.P2...	NULL	NULL
<input type="checkbox"/>	Edit Copy Delete	5	4	customer-1	Pelanggan Pertama	\$2y\$12\$vmD3coXUTDf.bzr9lggFwe9ZdvZETetqzkeYGeMvZrC...	NULL	2025-03-04 05:...
<input type="checkbox"/>	Edit Copy Delete	6	2	manager_dua	Manager 2	\$2y\$12\$1TW5LH6TTgza6S.wBMm1DuOYKJGMWLeuKEgjtLjrZ2...	2025-03-08 07:20:17	2025-03-08 07:...
<input type="checkbox"/>	Edit Copy Delete	8	2	manager22	Manager Dua Dua	\$2y\$12\$F8VWxNqzU5ys44Ms2OfsuBmx4DetlS7KgMct5SwQmH...	2025-03-08 10:03:45	2025-03-08 10:...

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

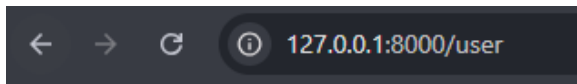
```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrCreate(
            [
                'username' => 'manager',
                'nama' => 'Manager',
            ],
        );

        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

Jawab: Mengecek apakah ada data dengan username = 'manager' dan nama = 'Manager' ditemukan, maka data akan ditampilkan, jika tidak ada data baru akan dibuat.



Data User

ID	Username	Nama	ID Level Pengguna
2	manager	Manager	2

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]);
    }
}
```

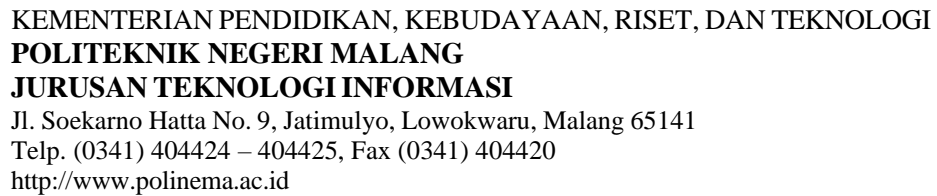
9. Simpan kode program Langkah 8. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel `m_user` serta beri penjelasan dalam laporan

Jawab:

- Mencari data dengan username = 'manager33' dan nama = 'Manager Tiga Tiga'.
- Jika ditemukan, data lama digunakan.
- Jika tidak ditemukan, data baru
- Tetapi data ini belum masuk ke database, hanya objek sementara di dalam kode.

Data User

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"/>		Edit		Copy		Delete	1	1	admin	Administrator	\$2y\$12\$Tbat8iftyc/cIcvz0ihgAFeRHii5K470NS6imzEcluzX...	NULL	NULL							
<input type="checkbox"/>		Edit		Copy		Delete	2	2	manager	Manager	\$2y\$12\$gQBSTAVxloo/EuHvLnafseOjcBHMxk9b6PlrwpuDaW...	NULL	NULL							
<input type="checkbox"/>		Edit		Copy		Delete	3	3	staff	Staff/Kasir	\$2y\$12\$2Fwun3kRudRY5PR.mdyVfOp6lqr2zbx0c.0GV5FP2...	NULL	NULL							
<input type="checkbox"/>		Edit		Copy		Delete	5	4	customer-1	Pelanggan Pertama	\$2y\$12\$vmdd3coXUTdFi.bzrgIggFwe9ZdvZetZetqkeYGeMvZrC...	NULL	2025-03-04 05:03:44							
<input type="checkbox"/>		Edit		Copy		Delete	6	2	manager_dua	Manager 2	\$2y\$12\$1TW5LH6TTgza6s.wBmM1DuOYKJGMW/LmEkgUllrZJ...	2025-03-08 07:20:17	2025-03-08 07:20:17							
<input type="checkbox"/>		Edit		Copy		Delete	8	2	manager22	Manager Dua Dua	\$2y\$12\$F8VWxqzU5ys44Ms2OfsuBmx4DetlS7KgmCt5SwQmH...	2025-03-08 10:03:45	2025-03-08 10:03:45							

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

Jawab: Fungsi \$user->save(); digunakan untuk menyimpan data ke database.

Data User

ID	Username	Nama	ID Level Pengguna
11	manager33	Manager Tiga Tiga	2

		user_id	level_id	username	nama	password	created_at	updated_at
<input type="checkbox"/>	 Edit  Copy  Delete	1	1	admin	Administrator	\$2y\$12\$Tbat8iftyc/lcvz0ihgAFeRHlI5K47QNS6imzEcluzX...	NULL	NULL
<input type="checkbox"/>	 Edit  Copy  Delete	2	2	manager	Manager	\$2y\$12\$gQBSTAVxloor/EuHvLnafseOjcBHMxk9b6BplrwpuDaW...	NULL	NULL
<input type="checkbox"/>	 Edit  Copy  Delete	3	3	staff	Staff/Kasir	\$2y\$12\$2Fwun3kRudRY5PR.mdYvOP6lqr2Bzx0.OGV5FP2...	NULL	NULL
<input type="checkbox"/>	 Edit  Copy  Delete	5	4	customer-1	Pelanggan Pertama	\$2y\$12\$vm3coXUTdF.bzr9lgrFwe9d2vZETetqzkeYGeMvZrC...	NULL	2025-03-04 05:03:44
<input type="checkbox"/>	 Edit  Copy  Delete	6	2	manager_dua	Manager 2	\$2y\$12\$1TW5LH6TtGza6S.wBmM1DuOYKJGmWLeuKEgllJrZ...	2025-03-08 07:20:17	2025-03-08 07:20:17
<input type="checkbox"/>	 Edit  Copy  Delete	8	2	manager22	Manager Dua Dua	\$2y\$12\$F/8VVXnqzU5y44Ms2OfsuBmx4DetlS7KgmCt5wQmH...	2025-03-08 10:03:45	2025-03-08 10:03:45
<input type="checkbox"/>	 Edit  Copy  Delete	11	2	manager33	Manager Tiga Tiga	\$2y\$12\$1zlbzi2OUGEJD94OzRnunYelc.a3T7toASXEG9g77...	2025-03-08 10:37:57	2025-03-08 10:37:57

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

Jawab:

- **isDirty()** digunakan untuk memeriksa apakah ada perubahan pada atribut model sebelum disimpan ke database.
- Setelah `$user->save();` dipanggil, perubahan yang sebelumnya ada sudah disimpan, sehingga `isDirty()` akan mengembalikan false, menandakan bahwa tidak ada lagi perubahan yang belum disimpan.

← → ↻ ⓘ 127.0.0.1:8000/user ☆ K ⋮

false // app\Http\Controllers\UserController.php:36



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();

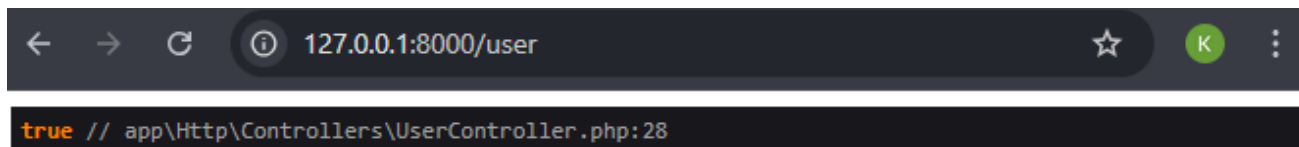
        $user->wasChanged(); // true
        $user->wasChanged('username'); // true
        $user->wasChanged(['username', 'level_id']); // true
        $user->wasChanged('nama'); // false
        dd($user->wasChanged(['nama', 'username'])); // true
    }
}
```



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

Jawab:

- `wasChanged()` digunakan untuk mengecek apakah ada perubahan pada model setelah data disimpan ke database.
- Jika tidak ada perubahan, maka `wasChanged()` akan mengembalikan `false`.
- Namun, jika ada perubahan pada salah satu atribut yang diperiksa, maka hasilnya akan `true`.
- Pada output yang ditampilkan, hasilnya adalah `true`, yang berarti terdapat perubahan data yang berhasil disimpan ke dalam database.



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



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>

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 scripnya 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

Jawab: Kode ini menampilkan daftar pengguna dalam tabel menggunakan data dari database. Setiap user memiliki tombol Ubah dan Hapus untuk mengedit atau menghapus data.



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
5	customer-1	Pelanggan Pertama	4	Ubah Hapus
6	manager_dua	Manager 2	2	Ubah Hapus
8	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

- 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 di bawah ini

```
<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>
```

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

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

- 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



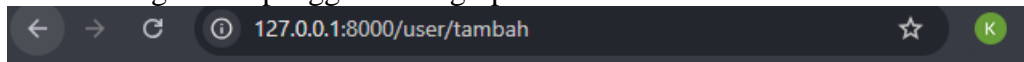
```
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::all();
        return view('user', ['data' => $user]);
    }

    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

Jawab:

- Route `Route::get('/user/tambah', [UserController::class, 'tambah']);` akan memanggil method `tambah()` di `UserController`.
- Method `tambah()` akan me-render view `user_tambah.blade.php`, yang menampilkan form tambah data user.
- Jika berhasil, halaman form "**Form Tambah Data User**" akan muncul di browser, memungkinkan pengguna menginput data baru.



Form Tambah Data User

Username	<input type="text" value="Masukkan Username"/>
Nama	<input type="text" value="Masukkan Nama"/>
Password	<input type="text" value="Masukkan Password"/>
Level ID	<input type="text" value="Masukkan ID Level"/>
<input type="button" value="Simpan"/>	

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
    ]);

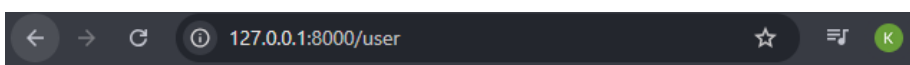
    return redirect('/user');
}
```

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

Jawab: Form dikirim ke /user/tambah_simpan dengan metode POST.

Form Tambah Data User

Username
Nama
Password
Level ID



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
5	customer-1	Pelanggan Pertama	4	Ubah Hapus
6	manager_dua	Manager 2	2	Ubah Hapus
8	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	admin2	Karina	1	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 diletakan 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
Jawab: hanya menampilkan form ubah

Form Ubah Data User

[Kembali](#)

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

14	admin2	Karina Ika	1	Ubah Hapus
----	--------	------------	---	--

Jawab:



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

Jawab:

- Sebelum

+ 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
5	customer-1	Pelanggan Pertama	4	Ubah Hapus
6	manager_dua	Manager 2	2	Ubah Hapus
8	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	admin2	Karina Ika	1	Ubah Hapus

- Sesudah

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
5	customer-1	Pelanggan Pertama	4	Ubah Hapus
6	manager_dua	Manager 2	2	Ubah Hapus
8	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

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

Jawab:

- Menambahkan LevelModel.php

```
Week4 > JS4 > app > Models > LevelModel.php > PHP > LevelModel
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6  use Illuminate\Database\Eloquent\Relations\HasMany;
7
8
9
10 1 reference | 0 implementations
11 class LevelModel extends Model
12 {
13     0 references
14     protected $table = 'm_level';
15     0 references
16     protected $primaryKey = 'level_id';
17     0 references
18     protected $fillable = ['level_kode', 'level_name']; //Foreign key
19
20     0 references | 0 overrides
21     public function users(): HasMany
22     {
23         return $this->hasMany(UserModel::class, 'level_id', 'level_id');
24     }
25 }
```

- Output: berhasil mengembalikan sebuah Collection yang berisi 8 item.

```
← → ↺ ⓘ 127.0.0.1:8000/user ☆ 🎵 K ⋮
Illuminate\Database\Eloquent\Collection {#320 ▼ // app\Http\Controllers\UserController.php:20
  #items: array:9 [▶]
  #escapeWhenCastingToString: false
}
```

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]);
}
```

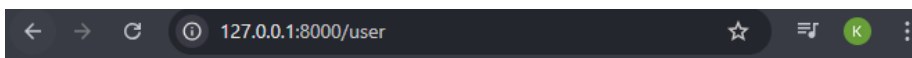
5. Buka file view pada `user.blade.php` dan ubah *script* menjadi seperti di bawah ini



```
<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

Jawab: Akan menampilkan semua data user dengan menambahkan Kode Level dan nama Level dari foreign key tabel level.



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
5	customer-1	Pelanggan Pertama	4	CUS	Customer	Ubah Hapus
6	manager_dua	Manager 2	2	MNG	Manager	Ubah Hapus
8	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

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

*** Sekian, dan selamat belajar ***