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

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

KHUZAIMA FILLA JANUARTHA
SIB2A/17

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 bagainama 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 pelajarai

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

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\UserModel;
use Illuminate\Support\Facades\Hash;

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

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

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

Data bertambah

Data User

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

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

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

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

```php
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

Error, karena field password pada model sehingga value pada user controller tidak dapat terbaca.

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:

```php
// 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

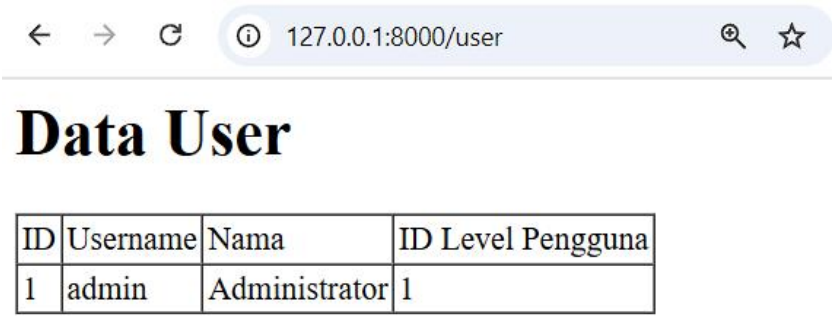
```php
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::find(1);
        return view('user', ['data' => $user]);
    }
}
```

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

Menampilkan tabel dengan data (id,username,nama,level_id) menggunakan function find() untuk mencari data by ID yang sudah ada.



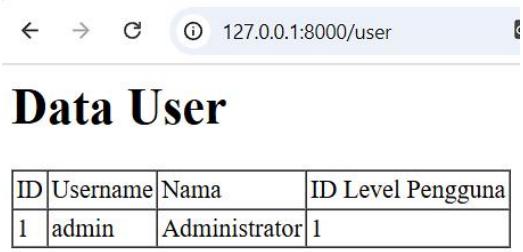
4. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
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

Mengambil dan menampilkan data user dengan menggunakan function where untuk

mengambil kolom dari tabel berdasarkan ID, function first() digunakan untk mengambil 1 baris pertama.



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

```php
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

Ouputannya sama, yang berbeda hanyalah function yang digunakan, jika tadi kita menggunakan first(), disini kita menggunakan firstwhere() untuk menampilkan 1 data pertama saja.

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:

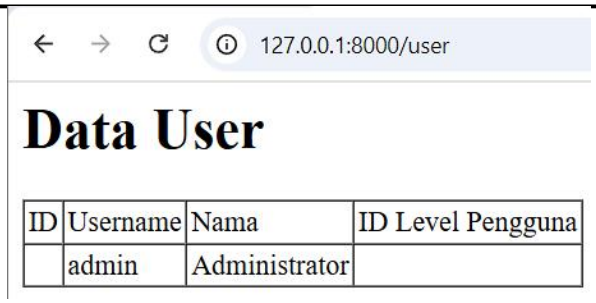
```php
$user = UserModel::findOr(1, function () {
    // ...
});

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

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

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

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

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

Kolom ID dan ID level pengguna tidak muncul, karena pada function findOr hanya memanggil kolom username dan nama pada id 1.



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

```php
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

Langsung mentrigger function abort karena pemanggilan yang dilaukan melebihi id yang sudah ada.

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

Praktikum 2.2 – *Not Found Exceptions*

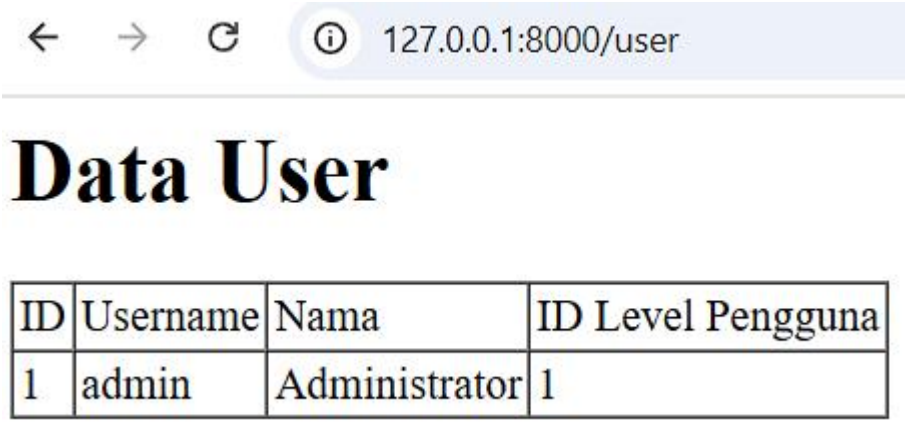
Terkadang Anda mungkin ingin memberikan pengecualian jika model tidak ditemukan. Hal ini sangat berguna dalam *rute* atau pengontrol. Metode `findOrFail` and `firstOrFail` akan mengambil hasil pertama dari kueri; namun, jika tidak ada hasil yang ditemukan, sebuah `Illuminate\Database\Eloquent\ModelNotFoundException`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

```php
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

Menampilkan data user dengan id 1, sesuai dengan namanya, menemukan data sesuai dengan id nya, jikalau tidak akan otomatis menampilkan abort 404 tanpa harus mengetik function abortnya.

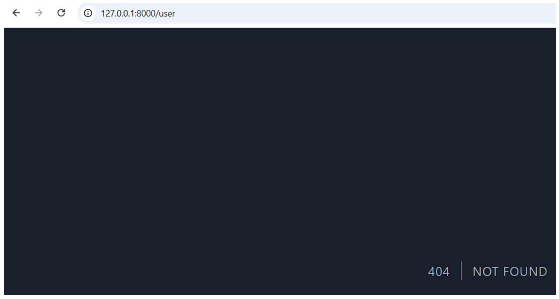


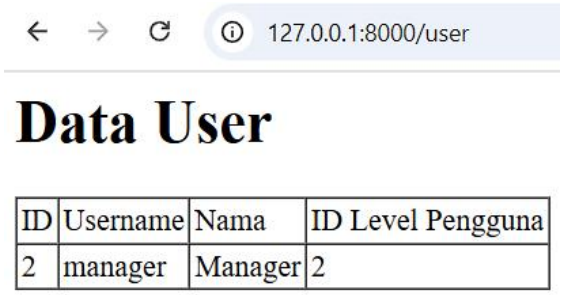
3. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```php
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

Ini digunakan untuk menampilkan data user yang lebih spesifik yaitu mencari data user berdasarkan kolom username dengan value manager9, karena pada database saya tidak ada manager9, maka akan otomatis mentrigger error 404 karena tidak ditemukan data yang dicari, Tetapi jika datanya ada, maka akan ditampilkan sebagaimana mestinya.





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

Praktikum 2.3 – *Retreiving Aggregrates*

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

   Ini adalah hasil eksekusi dari function dd/dump n die

```
3 // app\Http\Controllers\UserController.php:32
```

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

Mnghapus function DD, dan mengubah tampilan views menjadi demikian;

4.

```
<body>
    <h1>Data User</h1>
    <table border="1" cellpadding="2" cellspacing="0">
        <tr>
            <td>Jumlah Pengguna</td>
        </tr>
        <tr>
            <td>{{ $data }}</td>
        </tr>
    </table>
</body>
```


127.0.0.1:8000/user
Data User
Jumlah Pengguna
3

Data User

Jumlah Pengguna
2

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

Praktikum 2.4 – *Retreiving 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 datadase tersebut sesuai dengan nilai yang dimasukkan.

Metode `firstOrNew`, 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 `firstOrNew` belum disimpan ke database. Anda perlu memanggil metode `save()` secara manual untuk menyimpannya:

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

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

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

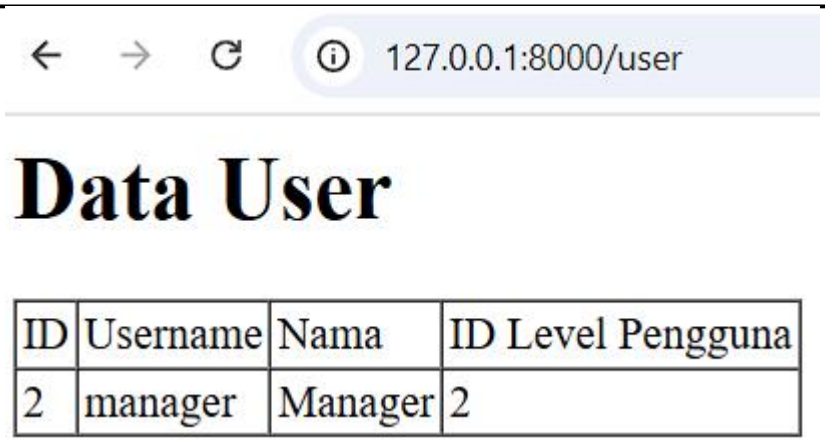
```php
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrCreate(
            [
                'username' => 'manager',
                'nama' => 'Manager',
            ],
        );

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

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

```php
<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

Menampilkan data dengan username dan nama data ‘manager’, apabila tidak ada, maka akan ditambahkan datanya seperti pada value yang diberikan

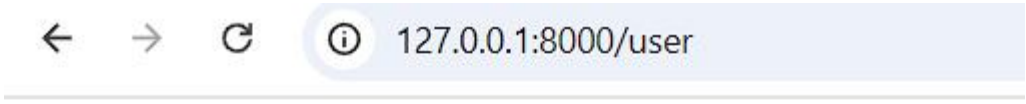


4. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```php
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

Dikarenakan data yang dicari sesuai dengan value itu tidak ada, maka akan langsung mengupdate dengan menambahkan data sesuai dengan value yang ada.



Data User

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

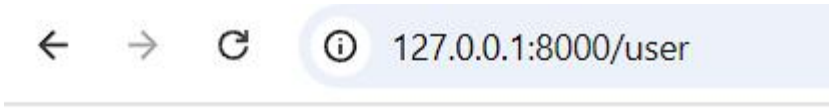
	user_id	level_id	username	nama	password	created_at	updated_at
Edit Copy Delete	1	1	admin	Administrator	$2y$10$flfbRRNkHQhYgcswpl26Ku8ZG/iYVwRF.M1eeG2fXpY...	NULL	NULL
Edit Copy Delete	2	2	manager	Manager	$2y$10$GHGLmo4CVD.2YBWdr25jRevhtZRebBGME/SPemmC6qZ...	NULL	NULL
Edit Copy Delete	3	3	staff	Staff/Kasir	$2y$10$S.ZjrePP4Ehq2pqi2Fy/vueRvmjstrIN6S2OdLX0Wqg...	NULL	NULL
Edit Copy Delete	6	5	customer-1	Pelanggan Pertama	$2y$10$pW9B2d/58XjU0DZ9U0GEkeVwggMTBy/e.xJM84Vv7tp...	NULL	2025-03-02 10:56:04
Edit Copy Delete	7	2	manager_dua	Manager 2	$2y$10$gLmY2uQNcLH82H.Yy171S.a/eb0frXldDEwZIOQpg3B...	2025-03-06 06:56:55	2025-03-06 06:56:55
Edit Copy Delete	8	2	manager_tiga	Manager 3	$2y$10$R3OTw.m34rfa8mTFUqCtBuPfdOz9ClyZbl2091gvEzw...	2025-03-06 06:58:06	2025-03-06 06:58:06
Edit Copy Delete	9	2	manager22	Manager Dua Dua	$2y$10$bvAbVSuoPjAiYy4qlYZyp.A5yJtUQlNX7cOerTUclAs...	2025-03-08 05:15:40	2025-03-08 05:15:40

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

```php
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrNew(
            [
                '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

Memanggil data sesuai value, jika data ditemukan, maka objek model yang sesuai akan dikembalikan. Jika data tidak ditemukan, maka metode ini akan membuat instance baru dari model tersebut, tetapi tidak langsung menyimpannya ke database.



Data User

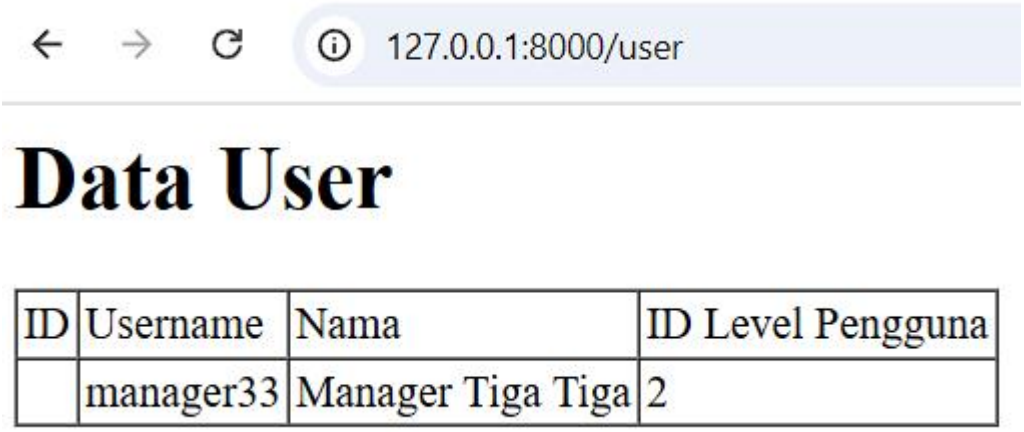
ID	Username	Nama	ID Level Pengguna
2	manager	Manager	2

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

```php
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrNew(
            [
                '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

Seperti pada langkah sebelumnya, karena data belum ada, maka yang ditampilkan adalah data yang barusaja dinput sesuai pada value tanpa ada ID nya. Tetapi data ini tidak tersimpan di database.



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

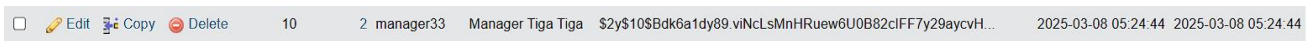
```php
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrNew(
            [
                'username' => 'manager33',
                'nama' => 'Manager Tiga Tiga',
                'password' => Hash::make('12345'),
                'level_id'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

Data sudah di save dan ditambahkan ke database, serta pada tampilannya juga sudah memiliki ID, karena sudah masuk ke database real.

Data User

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



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:

```php
$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

```php
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

   Muncul false yang berarti tidak ada perubahan data yang belum disimpan ke database.

   Setelah data disimpan, maka $user sudah dalam keadaan bersih (tidak ada perubahan yang belum disimpan). Itulah sebabnya saat isDirty() dipanggil setelah save(), hasilnya **false.**

```
false // app\Http\Controllers\UserController.php:75
```

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:

```php
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
    }
}
```

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

```php
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

wasChanged(['nama', 'username']) mengembalikan true karena setidaknya satu atribut dalam array mengalami perubahan setelah pemanggilan $user->save();

```
true // app\Http\Controllers\UserController.php:67
```

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 scritpnya menjadi seperti di bawah ini

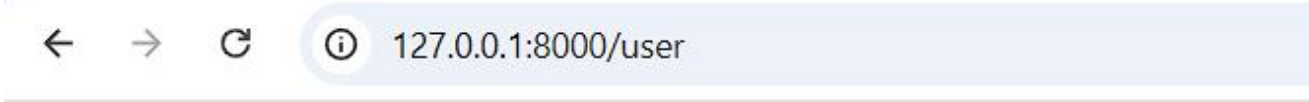
```
<body>
    <h1>Data User</h1>
    <a href="/user/tambah">+ Tambah User</a>
    <table border="1" cellpadding="2" cellspacing="0">
        <tr>
            <td>ID</td>
            <td>Username</td>
            <td>Nama</td>
            <td>ID Level Pengguna</td>
            <td>Aksi</td>
        </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

Menampilkan seluruh data pada tabel user dengan tambahan kolom aksi yang berisi tombol ubah dan hapus untuk melakukan CRUD pada data kita di databse melalui web.



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
6	customer-1	Pelanggan Pertama	5	Ubah | Hapus
7	manager_dua	Manager 2	2	Ubah | Hapus
8	manager_tiga	Manager 3	2	Ubah | Hapus
9	manager22	Manager Dua Dua	2	Ubah | Hapus
10	manager33	Manager Tiga Tiga	2	Ubah | Hapus
11	manager56	Manager55	2	Ubah | Hapus
12	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 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>
```

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

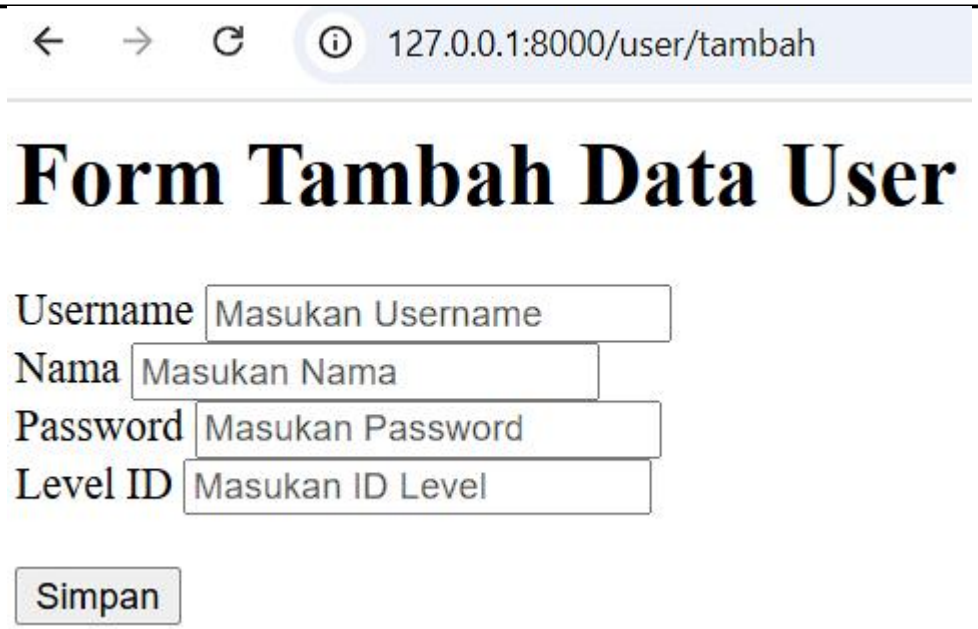
```
Route::get('/user/tambah', [UserController::class, 'tambah']);
```

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

Menampilkan form untuk menambahkan data pada tabel user yang berisi form username,nama,password,levelID, dan juga tombol 'Simpan' untuk menyimpan data.



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

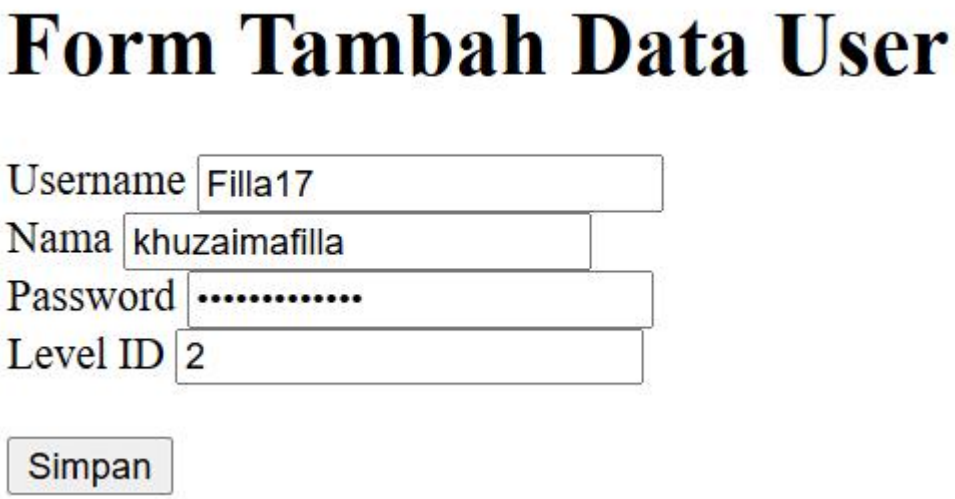
```
Route::post('/user/tambah_simpan', [UserController::class, 'tambah_simpan']);
```

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

```php
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

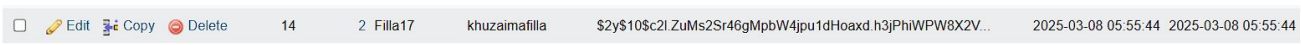
Data bertambah sesuai dengan yang saya input, sudah bisa ditampilkan di page Data user dan masuk ke database table m_user.



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
6	customer-1	Pelanggan Pertama	5	Ubah | Hapus
7	manager_dua	Manager 2	2	Ubah | Hapus
8	manager_tiga	Manager 3	2	Ubah | Hapus
9	manager22	Manager Dua Dua	2	Ubah | Hapus
10	manager33	Manager Tiga Tiga	2	Ubah | Hapus
11	manager56	Manager55	2	Ubah | Hapus
12	manager12	Manager11	2	Ubah | Hapus
13	khuzaima17	khuzaima	2	Ubah | Hapus
14	Filla17	khuzaimafilla	2	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

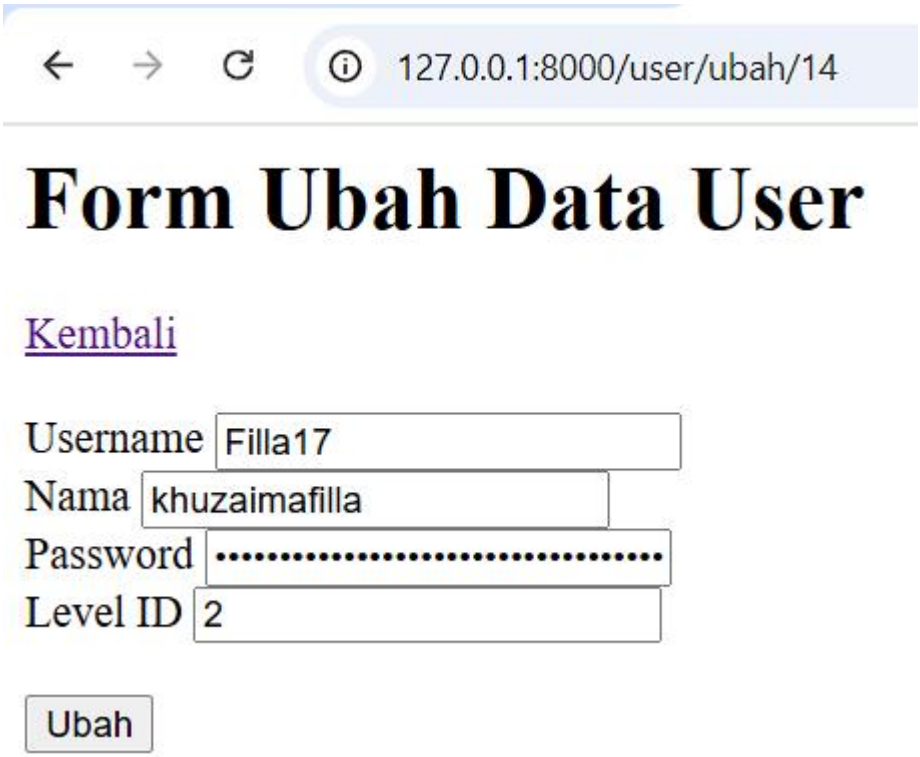
```
Route::get('/user/ubah/{id}', [UserController::class, 'ubah']);
```

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

Menampilkan page Form ubah data sesuai dengan baris data yang dipilih, disini saya memilih baris yang paling baru, yaitu data yang baru saja saya tambahkan di step yang sebelumnya. Tetapi di sini untuk tombol 'Ubah' belum bisa berfungsi, karena kita belum mengkonfigurasi fungsi dari tombol ubah itu sendiri melalui Route.



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

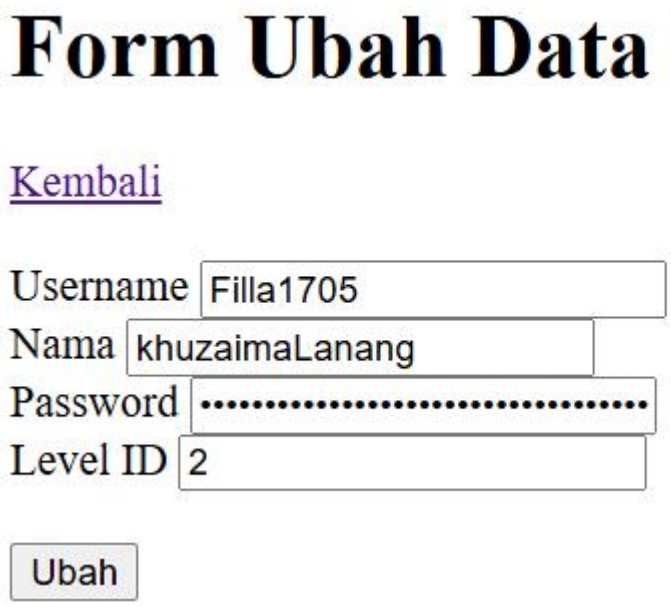
```
Route::put('/user/ubah_simpan/{id}', [UserController::class, 'ubah_simpan']);
```

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

```php
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

Saya berhasil mengubah Username dan Nama dari data terbaru saya, sudah bisa tampil di page Data user dan masuk ke database tabel m_user sesuai dengan perubahan yang terakhir.



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
6	customer-1	Pelanggan Pertama	5	Ubah | Hapus
7	manager_dua	Manager 2	2	Ubah | Hapus
8	manager_tiga	Manager 3	2	Ubah | Hapus
9	manager22	Manager Dua Dua	2	Ubah | Hapus
10	manager33	Manager Tiga Tiga	2	Ubah | Hapus
11	manager56	Manager55	2	Ubah | Hapus
12	manager12	Manager11	2	Ubah | Hapus
13	khuzaima17	khuzaima	2	Ubah | Hapus
14	Filla1705	khuzaimaLanang	2	Ubah | Hapus

Edit Copy Delete 14 2 Filla1705 khuzaimaLanang $2y$10$ak.nCnN/EJgC9DiE8NULluHcrJSj3F8Mv8vRe7BpV6q... 2025-03-08 05:55:44 2025-03-08 06:05:53

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

```php
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

Saya berhasil menghapus data terbaru saya tadi dengan menggunakan tombol 'Hapus', sudah bisa terhapus dari tampilan Data User dan sudah terhapus pula di databse tabel m_user

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
6	customer-1	Pelanggan Pertama	5	Ubah | Hapus
7	manager_dua	Manager 2	2	Ubah | Hapus
8	manager_tiga	Manager 3	2	Ubah | Hapus
9	manager22	Manager Dua Dua	2	Ubah | Hapus
10	manager33	Manager Tiga Tiga	2	Ubah | Hapus
11	manager56	Manager55	2	Ubah | Hapus
12	manager12	Manager11	2	Ubah | Hapus
13	khuzaima17	khuzaima	2	Ubah | Hapus

	user_id	level_id	username	nama	password	created_at	updated_at
Edit Copy Delete	1	1	admin	Administrator	$2y$10$flfbRRNkHQhYgcswpl26Ku8ZG/iYVwRF.M1eeG2fXpY...	NULL	NULL
Edit Copy Delete	2	2	manager	Manager	$2y$10$GHGLmo4CVD.2YBWdr25jRevhtZRebBGME/SPemmC6qZ...	NULL	NULL
Edit Copy Delete	3	3	staff	Staff/Kasir	$2y$10$S.ZjrePP4Ehq2pqi2Fy/vueRvmjstrIN6S2OdLX0Wqg...	NULL	NULL
Edit Copy Delete	6	5	customer-1	Pelanggan Pertama	$2y$10$pW9B2d/58XjU0DZ9U0GEkeVwggMTBy/e.xJM84Vv7tp...	NULL	2025-03-02 10:56:04
Edit Copy Delete	7	2	manager_dua	Manager 2	$2y$10$gLmY2uQNcLH82H.Yy171S.a/eb0frXldDEwZIOQpg3B...	2025-03-06 06:56:55	2025-03-06 06:56:55
Edit Copy Delete	8	2	manager_tiga	Manager 3	$2y$10$R3OTw.m34rfa8mTFUqCtBuPfdOz9ClyZbl2091gvEzw...	2025-03-06 06:58:06	2025-03-06 06:58:06
Edit Copy Delete	9	2	manager22	Manager Dua Dua	$2y$10$bvAbVSuoPjAiYy4qlYZyp.A5yJtUQINX7cOerTUclAs...	2025-03-08 05:15:40	2025-03-08 05:15:40
Edit Copy Delete	10	2	manager33	Manager Tiga Tiga	$2y$10$Bdk6a1dy89.viNcLsMnHRuew6U0B82cIFF7y29aycvH...	2025-03-08 05:24:44	2025-03-08 05:24:44
Edit Copy Delete	11	2	manager56	Manager55	$2y$10$kiC3f0StoslJU0C.tJwzlOu.flSiOT/WmGca2O.w11Q...	2025-03-08 05:32:13	2025-03-08 05:32:13
Edit Copy Delete	12	2	manager12	Manager11	$2y$10$WOu8pkyxBTzzt19xTwYl5OUd9PYtDrhUrVqc4xdcZgG...	2025-03-08 05:37:19	2025-03-08 05:37:19
Edit Copy Delete	13	2	khuzaima17	khuzaima	$2y$10$SRHetKmabLItj5mIH3LUZetlNRZyYAl9tfJDQ/HMrrk...	2025-03-08 05:54:59	2025-03-08 05:54:59

Check all *With selected:* Edit Copy Delete Export

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
<?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
<?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
<?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
<?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 scritpnya menjadi seperti di bawah ini

```php
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

```php
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

Ini adalah hasil dari query database yang mengembalikan 11 data pengguna dalam bentuk koleksi Eloquent yang menunjukkan bahwa koleksi ini memiliki 11 item, yang berarti ada 11 objek UserModel yang diambil dari database.

```
Illuminate\Database\Eloquent\Collection {#321 ▼ // app\Http\Controllers\UserController.php:70
  #items: array:11 [▼
    0 => App\Models\UserModel {#328 ▶}
    1 => App\Models\UserModel {#329 ▶}
    2 => App\Models\UserModel {#330 ▶}
    3 => App\Models\UserModel {#331 ▶}
    4 => App\Models\UserModel {#332 ▶}
    5 => App\Models\UserModel {#333 ▶}
    6 => App\Models\UserModel {#334 ▶}
    7 => App\Models\UserModel {#335 ▶}
    8 => App\Models\UserModel {#336 ▶}
    9 => App\Models\UserModel {#337 ▶}
    10 => App\Models\UserModel {#338 ▶}
  ]
  #escapeWhenCastingToString: false
}
```

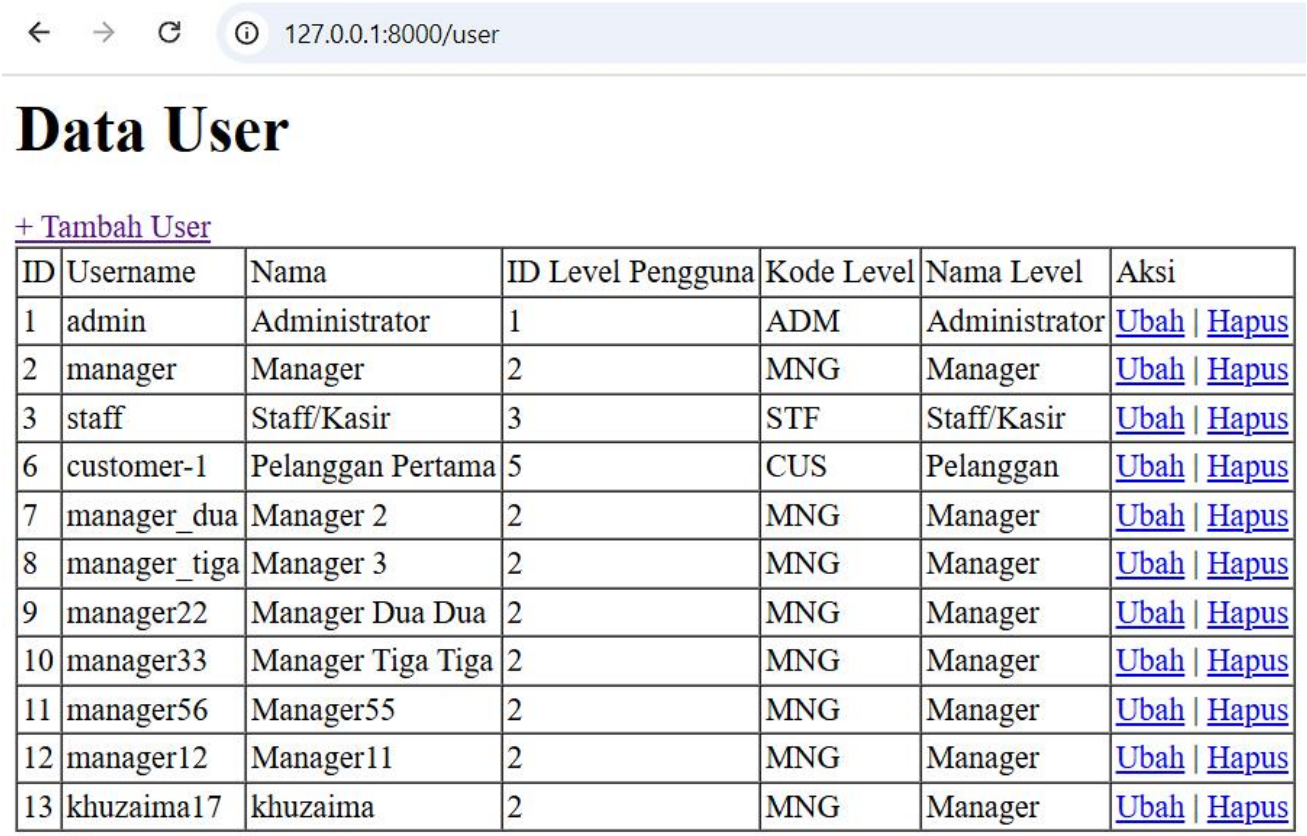
4. Buka file controller pada `UserController.php` dan ubah method *script* menjadi seperti di bawah ini

```php
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

```php
<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>
```

6. Simpan kode program Langkah 4 dan 5. Kemudian jalankan link pada *browser*, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan
   Tabel pada Data User menambahkan 2 Kolom yaitu ‘Kode Aksi’ dan ‘Nama Level



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
6	customer-1	Pelanggan Pertama	5	CUS	Pelanggan	Ubah | Hapus
7	manager_dua	Manager 2	2	MNG	Manager	Ubah | Hapus
8	manager_tiga	Manager 3	2	MNG	Manager	Ubah | Hapus
9	manager22	Manager Dua Dua	2	MNG	Manager	Ubah | Hapus
10	manager33	Manager Tiga Tiga	2	MNG	Manager	Ubah | Hapus
11	manager56	Manager55	2	MNG	Manager	Ubah | Hapus
12	manager12	Manager11	2	MNG	Manager	Ubah | Hapus
13	khuzaima17	khuzaima	2	MNG	Manager	Ubah | Hapus

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

**** Sekian, dan selamat belajar ****