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)

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'];
}
```

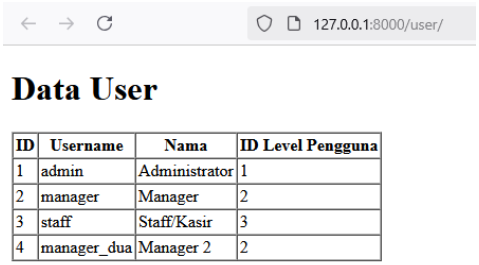
```php
2 references | 0 implementations
class UserModel extends Model
{
    use HasFactory;

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

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    0 references
    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]);
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $data = [
            'level_id' => 2,
            'username' => 'manager_dua',
            'nama' => 'Manager 2',
            'password' => Hash::make(value: '12345')
        ];

        UserModel::create(attributes: $data);

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



Controller mengambil data dari database menggunakan model, lalu mengirimnya ke view untuk ditampilkan.

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

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

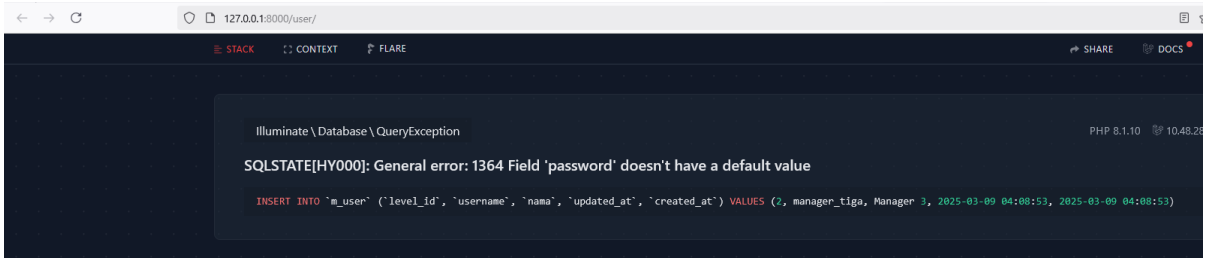
```
0 references
    protected $fillable = ['level_id', 'username', 'nama'];
}
```

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

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

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

```
1 reference | 0 overrides
public function index(): Factory|View
{
    $data = [
        'level_id' => 2,
        'username' => 'manager_tiga',
        'nama' => 'Manager 3',
        'password' => Hash::make(value: '12345')
    ];

    UserModel::create(attributes: $data);

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

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



Kode akan error karena kolom password tidak ada di $fillable pada model UserModel.

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]);
    }
}
```

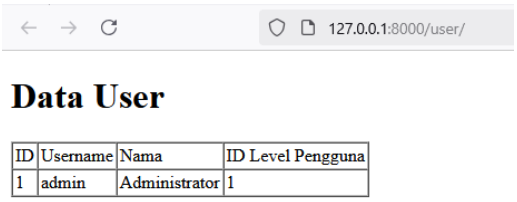
```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::find(id: 1);
        return view(view: 'user', data: ['data' => $user]);
    }
}
```

2. Buka 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>
```

```
<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



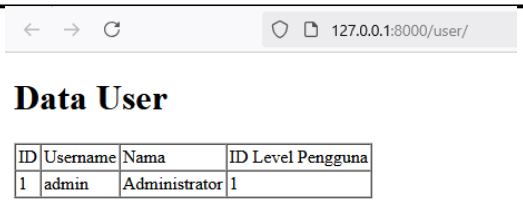
mengambil data pengguna dengan ID 1 dari database melalui UserController dan menampilkannya dalam tabel di halaman user.blade.php

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]);
    }
}
```

```
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::where(column: 'level_id', operator: 1)->first();
        return view(view: 'user', data: ['data' => $user]);
    }
}
```

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



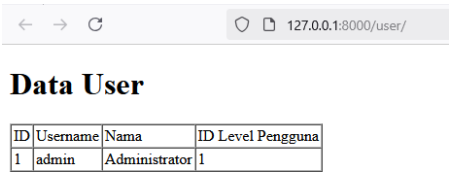
mengambil satu data pengguna pertama (first()) dengan level_id bernilai 1 dari database menggunakan UserModel

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]);
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::firstWhere(column: 'level_id', operator: 1);
        return view(view: 'user', data: ['data' => $user]);
    }
}
```

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



mengambil satu pengguna pertama dengan level_id = 1 menggunakan firstWhere()

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:

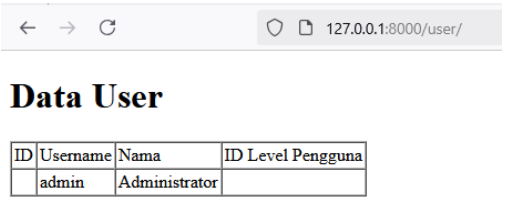
```php
$user = UserModel::findOr(1, function () {
    // ...
});

$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]);
    }
}
```

```php
public function index(): Factory|View
{
    $user = UserModel::findOr(id: 1, columns: ['username', 'nama'], callback: function (): never {
        abort(code: 404);
    });
    return view(view: 'user', data: ['data' => $user]);
}
```

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



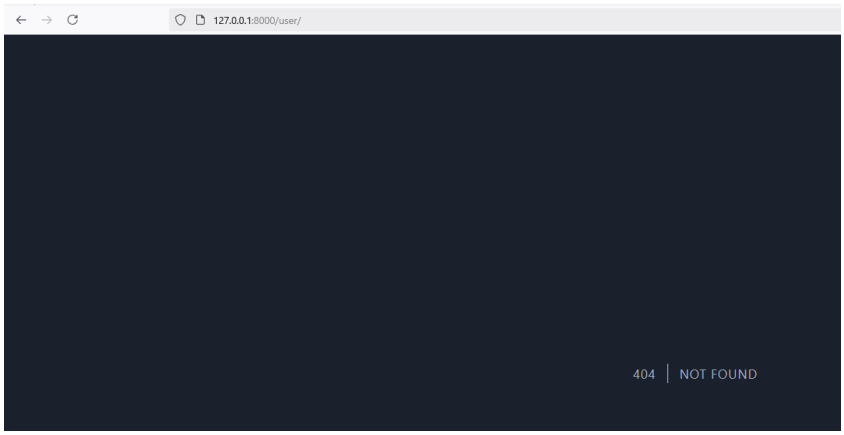
mencari pengguna dengan id = 1, mengambil hanya kolom username dan nama, lalu menampilkannya di view user.blade.php; jika tidak ditemukan, akan menampilkan error 404.

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]);
    }
}
```

```
1 reference | 0 overrides
public function index(): Factory|View
{
    $user = UserModel::findOr(id: 20, columns:
        abort(code: 404);
    });
```

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



jika tidak ditemukan, akan menampilkan error 404.

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

Praktikum 2.2 – *Not Found Exceptions*

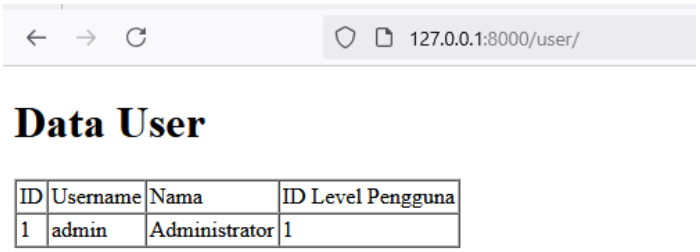
Terkadang Anda mungkin ingin memberikan pengecualian jika model tidak ditemukan. Hal ini sangat berguna dalam *rute* atau pengontrol. Metode `findOrFail` and `firstOrFail` akan mengambil hasil pertama dari kueri; namun, jika tidak ada hasil yang ditemukan, sebuah `Illuminate\Database\Eloquent\ModelNotFoundException`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

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

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



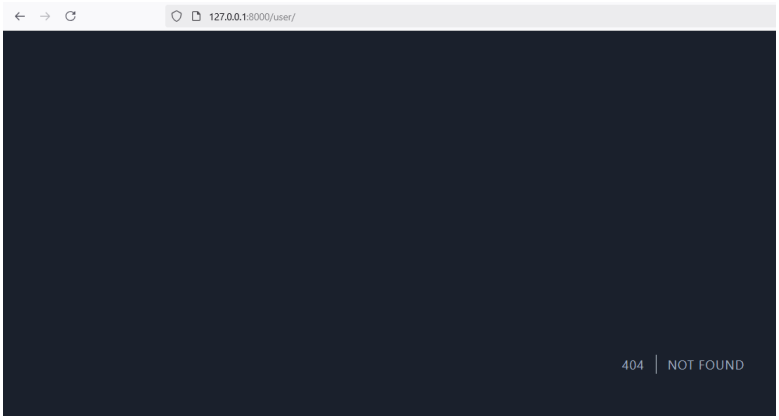
Kode ini mencari pengguna dengan id = 1, jika tidak ditemukan akan menampilkan error 404, lalu mengirim data ke view user.blade.php.

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]);
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::where(column: 'username', operator: 'manager9')->firstOrFail();
        return view(view: 'user', data: ['data' => $user]);
    }
}
```

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



Kode ini mencari pengguna dengan username = 'manager9', jika tidak ditemukan akan menampilkan error 404, lalu mengirim data ke view user.blade.php

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:

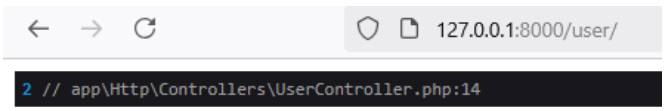
```
$count = UserModel::where('active', 1)->count();

$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]);
    }
}
```

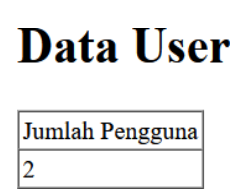
```
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::where(column: 'level_id', operator: 2)->count();
        dd(vars: $user); // Akan menampilkan jumlah pengguna dengan level_id = 2 dan menghentikan eksekusi

        return view(view: 'user', data: ['data' => $user]); // Tidak akan berjalan setelah dd()
    }
}
```

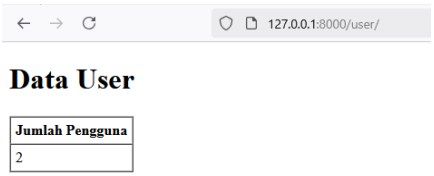
2. Simpan kode program Langkah 1. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan



Kode ini menghitung jumlah pengguna dengan level_id = 2, lalu menampilkan hasilnya dengan dd($user), sehingga return view() tidak akan dieksekusi.

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





Usercontroller

```php
<?php

namespace App\Http\Controllers;

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

2 references | 0 implementations | You, 41 seconds ago | 1 author (You)
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $userCount = UserModel::where(column: 'level_id', operator: 2)->count();
        return view(view: 'user', data: ['data' => $userCount]);
    }
}
```

User.blade

```php
<!DOCTYPE html>
<html>

<head>
    <title>Data User</title>
</head>

<body>
    <h1>Data User</h1>
    <table border="1" cellpadding="5" cellspacing="0">
        <tr>
            <th>Jumlah Pengguna</th>
        </tr>
        <tr>
            <td>{{ $data }}</td>
        </tr>
    </table>
</body>

</html>      You, 18 minutes ago • minggu 4
```

4. 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]);
    }
}
```

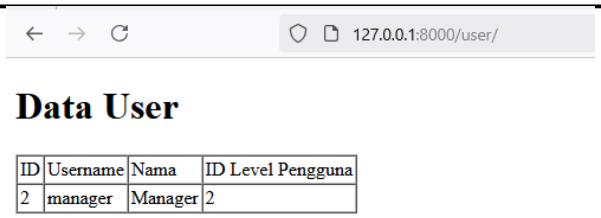
```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::firstOrCreate(
            attributes: ['username' => 'manager'], // Kondisi pencarian
            values: ['nama' => 'Manager'] // Data yang akan dibuat jika tidak ditemukan
        );

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

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

```blade
<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>
```

```blade
<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

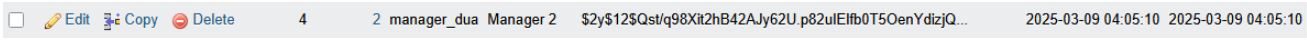


Kode ini mencari pengguna dengan username = 'manager', jika tidak ditemukan, akan membuatnya dengan nama = 'Manager',

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



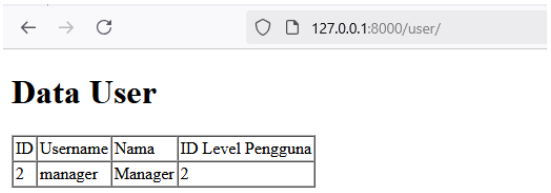
mencari pengguna dengan username = 'manager22', jika tidak ditemukan, akan membuatnya dengan nama, password yang telah di-hash, dan level_id = 2

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]);
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::firstOrNew(
            attributes: ['username' => 'manager'], // Kondisi pencarian
            values: ['nama' => 'Manager'] // Data default jika tidak ditemukan
        );

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

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



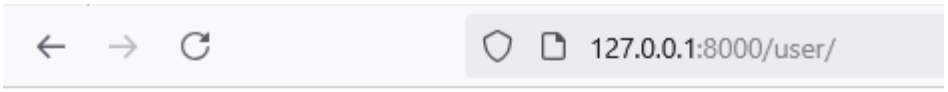
Kode ini mencari pengguna dengan username = 'manager', jika tidak ditemukan, akan membuat instance baru tanpa menyimpannya ke database,

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]);
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::firstOrNew(
            attributes: ['username' => 'manager33'], // Kondisi pencarian
            values: [
                'nama' => 'Manager Tiga Tiga',
                'password' => Hash::make(value: '12345'),
                'level_id' => 2
            ] // Data default jika tidak ditemukan
        );

        return view(view: 'user', data: ['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



Data User

ID	Username	Nama	ID Level Pengguna
	manager33	Manager Tiga Tiga	2

Kode ini mencari pengguna dengan username = 'manager33', jika tidak ditemukan, akan membuat instance baru tanpa menyimpannya ke 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]);
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::firstOrNew(attributes: ['username' => 'manager33']);

        // Mengisi atribut yang diperlukan
        $user->nama = 'Manager Tiga Tiga';
        $user->password = Hash::make(value: '12345');
        $user->level_id = 2;

        // Simpan ke database
        $user->save();

        return view(view: 'user', data: ['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 User

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

Kode ini mencari pengguna dengan username = 'manager33', jika tidak ditemukan, akan membuat instance baru dan menyimpannya ke database,

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:

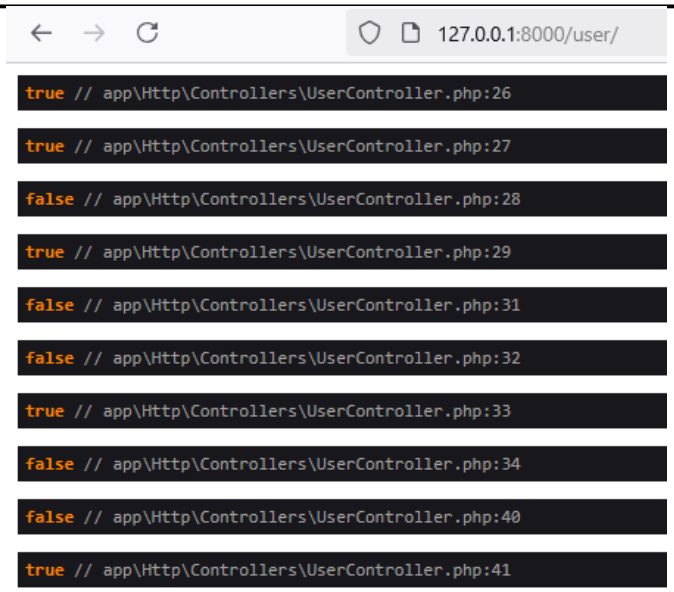
```php
$user = UserModel::create([
    'username' => 'manager44',
    'nama' => 'Manager44',
    'password' => Hash::make('12345'),
    'level_id' => 2,
]);

$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());
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): never
    {
        // Membuat user baru
        $user = UserModel::create(attributes: [
            'username' => 'manager55',
            'nama' => 'Manager55',
            'password' => Hash::make(value: '12345'),
            'level_id' => 2,
        ]);

        // Mengubah username
        $user->username = 'manager56';

        // Cek perubahan sebelum disimpan
        dump(vars: $user->isDirty()); // true (karena ada perubahan)
        dump(vars: $user->isDirty(attributes: 'username')); // true (username berubah)
        dump(vars: $user->isDirty(attributes: 'nama')); // false (nama tidak berubah)
        dump(vars: $user->isDirty(attributes: ['nama', 'username'])); // true (karena username berubah)

        dump(vars: $user->isClean()); // false (karena masih ada perubahan)
        dump(vars: $user->isClean(attributes: 'username')); // false
        dump(vars: $user->isClean(attributes: 'nama')); // true
        dump(vars: $user->isClean(attributes: ['nama', 'username'])); // false

        // Simpan perubahan ke database
        $user->save();

        // Cek setelah penyimpanan
        dump(vars: $user->isDirty()); // false (karena sudah disimpan)
        dump(vars: $user->isClean()); // true (karena tidak ada perubahan)

        // Hentikan eksekusi dan tampilkan hasil
        dd();
    }
}
```

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



Kode ini akan menampilkan hasil pengecekan perubahan data dalam format dump

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:

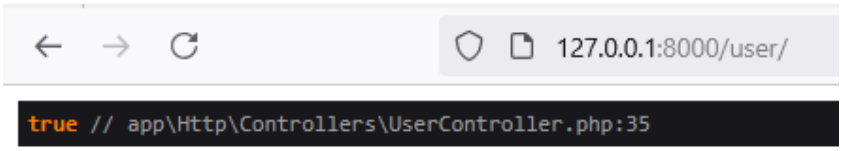
```php
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::create([
            'username' => 'manager11',
            'nama' => 'Manager11',
            'password' => Hash::make('12345'),
            'level_id' => 2,
        ]);

        $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
    }
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): never
    {
        // Membuat user baru
        $user = UserModel::create(attributes: [
            'username' => 'manager11',
            'nama' => 'Manager11',
            'password' => Hash::make(value: '12345'),
            'level_id' => 2,
        ]);

        // Mengubah username
        $user->username = 'manager12';

        // Simpan perubahan
        $user->save();

        // Cek apakah ada perubahan setelah `save()`
        $user->wasChanged(); // true (karena ada perubahan)
        $user->wasChanged(attributes: 'username'); // true (username berub
        $user->wasChanged(attributes: ['username', 'level_id']); // true (
        $user->wasChanged(attributes: 'nama'); // false (nama tidak beruba

        // Hentikan eksekusi dan tampilkan hasil
        dd(vars: $user->wasChanged(attributes: ['nama', 'username'])); //
    }
}
```

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



Kode ini membuat user baru dengan username "manager11", lalu mengubahnya menjadi "manager12", menyimpannya, dan mengecek perubahan menggunakan wasChanged()

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>
```

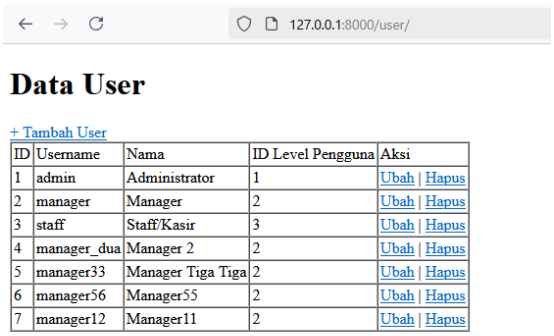
```
<body>
    <h1>Data User</h1>
    <a href="/user/tambah">+ Tambah User</a>
    <table border="1" cellpadding="2" cellspacing="0">
        <tr>
            <td>ID</td>
            <td>Username</td>
            <td>Nama</td>
            <td>ID Level Pengguna</td>
            <td>Aksi</td>
        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

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

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::all(); // Mengambil semua data user dari database
        return view(view: 'user', data: ['data' => $user]); // Mengirim data ke tampilan (view)
    }
}
```

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



Kode ini menampilkan daftar pengguna dalam tabel dengan opsi tambah, ubah, dan hapus, serta menggunakan perulangan @foreach untuk menampilkan data dari variabel $data

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>
```

```
<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

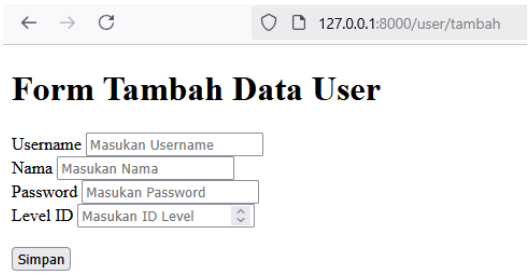
```
Route::get('/user/tambah', [UserController::class, 'tambah']);
Route::get(uri: '/user/tambah', action: [UserController::class, 'index']);
```

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

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

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

```php
class UserController extends Controller
{
    2 references | 0 overrides
    public function index(): Factory|View
    {
        $user = UserModel::all(); // Mengambil semua data user da
        return view(view: 'user', data: ['data' => $user]); // Me
    }

    0 references | 0 overrides
    public function tambah(): Factory|View
    {
        return view(view: 'user_tambah'); // Menampilkan halaman
    }
}
```

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



form tambah data user yang mengirimkan data ke /user/tambah_simpan menggunakan metode POST.

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

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

```php
Route::post(uri: '/user/tambah_simpan', action: [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');
}
```

```php
public function tambah_simpan(Request $request): Redirector|Red
{
    UserModel::create(attributes: [
        'username' => $request->username,
        'nama' => $request->nama,
        'password' => Hash::make(value: $request->password), //
        'level_id' => $request->level_id
    ]);

    return redirect(to: '/user'); // Redirect ke halaman daftar
}
```

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

8	wahyu1	wahyu1	1	Ubah | Hapus

Fungsi tambah_simpan() menerima data dari form, menyimpannya ke dalam database menggunakan Eloquent create(), mengenkripsi password dengan Hash::make(), lalu mengarahkan pengguna kembali ke halaman daftar user dengan redirect('/user')

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>
```

```
<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(method: '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->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" 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

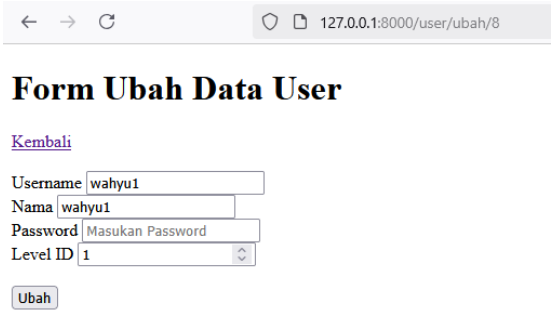
```
Route::get('/user/ubah/{id}', [UserController::class, 'ubah']);
Route::get(uri: '/user/ubah/{id}', action: [UserController::class, 'ubah']);
```

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]);
}
```

```php
public function ubah($id): Factory|View
{
    $user = UserModel::find(id: $id);
    return view(view: 'user_ubah', data: ['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



halaman form "Ubah Data User" berhasil ditampilkan dengan data pengguna yang sesuai

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

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

```php
public function ubah_simpan($id, Request $request): Redirecto
{
    // Cari user berdasarkan ID
    $user = UserModel::findOrFail(id: $id);

    // Update data user
    $user->username = $request->username;
    $user->nama = $request->nama;
    $user->password = Hash::make(value: $request->password);
    $user->level_id = $request->level_id;

    // Simpan perubahan
    $user->save();

    // Redirect ke halaman user
    return redirect(to: '/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

8	wahyu2	wahyu2	1	Ubah | Hapus

Data user di database akan diperbarui sesuai input yang dimasukkan

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

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

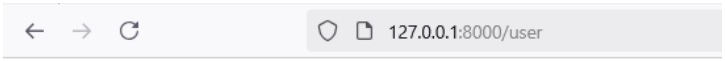
```php
public function hapus($id)
{
    $user = UserModel::find($id);
    $user->delete();

    return redirect('/user');
}
public function hapus($id): Redirector|RedirectResponse
{
    $user = UserModel::find(id: $id);
    $user->delete();

    return redirect(to: '/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

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
4	manager_dua	Manager 2	2	Ubah | Hapus
5	manager33	Manager Tiga Tiga	2	Ubah | Hapus
6	manager56	Manager55	2	Ubah | Hapus
7	manager12	Manager11	2	Ubah | Hapus
8	wahyu2	wahyu2	1	Ubah | Hapus



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
4	manager_dua	Manager 2	2	Ubah | Hapus
5	manager33	Manager Tiga Tiga	2	Ubah | Hapus
6	manager56	Manager55	2	Ubah | Hapus
7	manager12	Manager11	2	Ubah | Hapus

Ketika tombol "Hapus" diklik, sistem akan mencari data pengguna berdasarkan ID dan menghapusnya dari database

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');
    }
}
```

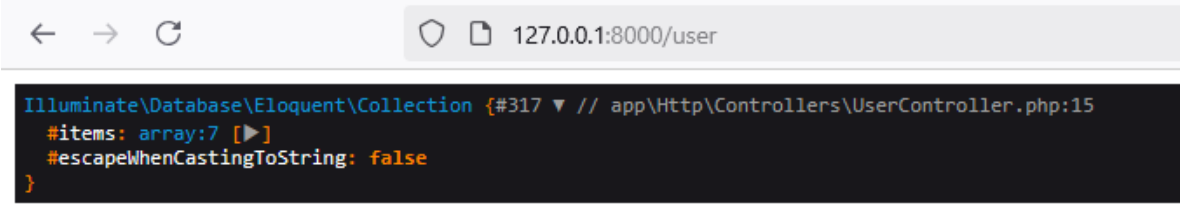
```php
class UserModel extends Model
{
    use HasFactory;

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

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


    /**
     * Relasi ke tabel LevelModel
     */
    0 references | 0 overrides
    public function level(): BelongsTo
    {
        return $this->belongsTo(related: LevelModel::class, foreignKey: 'level_id', ownerKey: '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);
}
```

```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function index(): never
    {
        $user = UserModel::with(relations: 'level')->get();
        dd(vars: $user);
    }
```

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



Ketika menjalankan /user, program menampilkan dump data (dd($user);), yang menunjukkan bahwa data berhasil diambil dari database

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]);
}
```

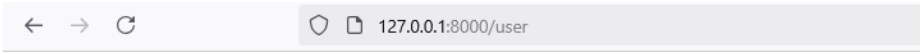
```php
public function index(): Factory|View
{
    // Mengambil data user beserta relasi level
    $user = UserModel::with(relations: 'level')->get();

    // Mengembalikan data ke view 'user.blade.php'
    return view(view: 'user', data: ['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>
            </tr>
        @endforeach
    </table>
</body>
```

```php
<body>
    <h1>Data User</h1>
    <a href="/user/tambah">+ Tambah User</a>

    <table border="1" cellpadding="2" cellspacing="0">
        <tr>
            <th>ID</th>
            <th>Username</th>
            <th>Nama</th>
            <th>ID Level Pengguna</th>
            <th>Kode Level</th>
            <th>Nama Level</th>
            <th>Aksi</th>
        </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
```

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



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
4	manager_dua	Manager 2	2	MNG	Manager	Ubah | Hapus
5	manager33	Manager Tiga Tiga	2	MNG	Manager	Ubah | Hapus
6	manager56	Manager55	2	MNG	Manager	Ubah | Hapus
7	manager12	Manager11	2	MNG	Manager	Ubah | Hapus

Setelah menjalankan link pada browser, halaman menampilkan daftar user dalam bentuk tabel dengan informasi ID, username, nama, level pengguna, serta opsi Ubah dan Hapus, menandakan bahwa data berhasil diambil dan ditampilkan dengan benar.

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