

**LAPORAN PRAKTIKUM**  
**MATA KULIAH PEMROGRAMAN WEB LANJUT**  
**JOBSHEET IV : MODEL DAN ELOQUENT ORM**

Dosen Pengampu : Dimas Wahyu Wibowo, S.T., M.T.



**Disusun oleh :**

Nama : Vita Eka Saraswati

NIM : 2341760082

No. Absen : 29

**JURUSAN TEKNOLOGI INFORMASI**  
**PROGRAM STUDI D-IV SISTEM INFORMASI BISNIS**  
**POLITEKNIK NEGERI MALANG**

**2025**



Nama : Vita Eka Saraswati  
NIM / Kelas : 2341760082 / SIB 2A  
No. Absen : 29  
Mata Kuliah : Pemrograman Web Lanjut – Pertemuan 4

## JOBSHEET 04

### MODEL dan ELOQUENT ORM

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

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

Sesuai dengan **studi Kasus PWL.pdf**.

Jadi project Laravel 10 kita masih sama dengan menggunakan repositori **PWL\_POS**.

*Project PWL\_POS* akan kita gunakan sampai pertemuan 12 nanti, sebagai project yang akan kita pelajari

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

#### Kelebihan - Kelebihan Menggunakan ORM

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



2. perintah query memiliki kinerja yang lebih baik, daripada kita menulisnya secara manual.
  3. Kita menulis model data hanya di satu tempat, sehingga lebih mudah untuk update, maintain, dan reuse the code.
  4. Memungkinkan kita memanfaatkan OOP (object oriented programming) dengan baik
- Di Laravel sendiri telah disediakan Eloquent ORM untuk mempermudah kita dalam melakukan berbagai macam query ke database, dan membuat pekerjaan kita menjadi lebih mudah karena tidak perlu menuliskan query sql yang panjang untuk memproses data.

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

### 1. `$fillable`

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

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

### Praktikum 1 - `$fillable`:

---

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

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

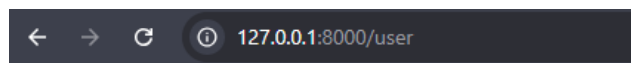
2. Buka file controller dengan nama `UserController.php` dan ubah *script* untuk menambahkan data baru seperti gambar di bawah ini



```
UserController.php M x UserModel.php M
PWL_POS > app > Http > Controllers > UserController.php > PHP > UserController > index()

3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\UserModel;
7 use Illuminate\Support\Facades\DB;
8 use Illuminate\Support\Facades\Hash;
9
10 class UserController extends Controller
11 {
12     //tambah data user dengan Eloquent Model
13     // $data = [
14     //     // 'username' => 'customer-1',
15     //     // 'nama' => 'Pelanggan Pertama',
16     //     // 'password' => Hash::make('12345'),
17     //     // 'level_id' => 4
18     // ];
19     // UserModel::insert($data); //tambahkan data ke tabel user
20
21     $data = [
22         'level_id' => 2,
23         'username' => 'manager_dua',
24         'nama' => 'Manager 2',
25         'password' => Hash::make('12345'),
26     ];
27
28     UserModel::create($data);
29
30     // coba akses model UserModel
31     $user = UserModel::all(); // ambil semua data dari tabel m_user
32     return view('user', data: ['data' => $user]);
33 }
34
35
36
37
38 }
```

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



## Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1
2	manager	Manager	2
3	staff	Staf/Kasir	3
4	customer-1	Customer 1	4
16	customer-2	Pelanggan Kedua	4
18	manager_dua	Manager 2	2

**Jawab :** Kode di atas digunakan untuk menambahkan atau insert data manager 2 pada database dengan eloquent ORM, sehingga saat server dijalankan dan view ditampilkan pada web browser tabel akan menampilkan data baru yaitu manager 2



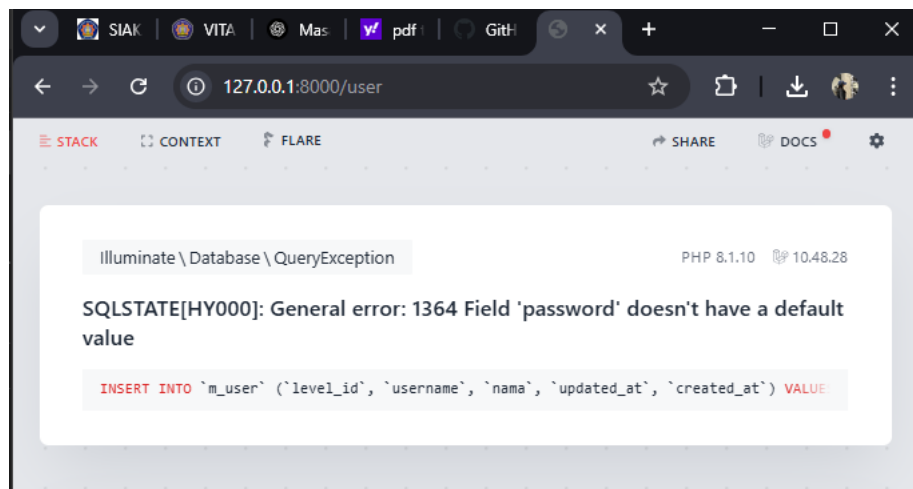
4. Ubah file model `UserModel.php` seperti pada gambar di bawah ini pada bagian `$fillable`

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

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

```
30  
31  
32 $data = [  
33     'level_id' => 2,  
34     'username' => 'manager_tiga',  
35     'nama' => 'Manager 3',  
36     'password' => Hash::make('12345'),  
37 ];  
38  
39 UserModel::create($data);
```

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



7.

**Jawab :** Akan terjadi error, karena saat data manager 3 akan ditambahkan hanya mengisi field `level_id`, `username`, dan `nama`. Error ini terjadi karena field `password` tidak memiliki nilai default sehingga saat kolom tidak diisi /null menyebabkan error.

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



## 2. `$guarded`

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

## B. RETRIEVING SINGLE MODELS

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

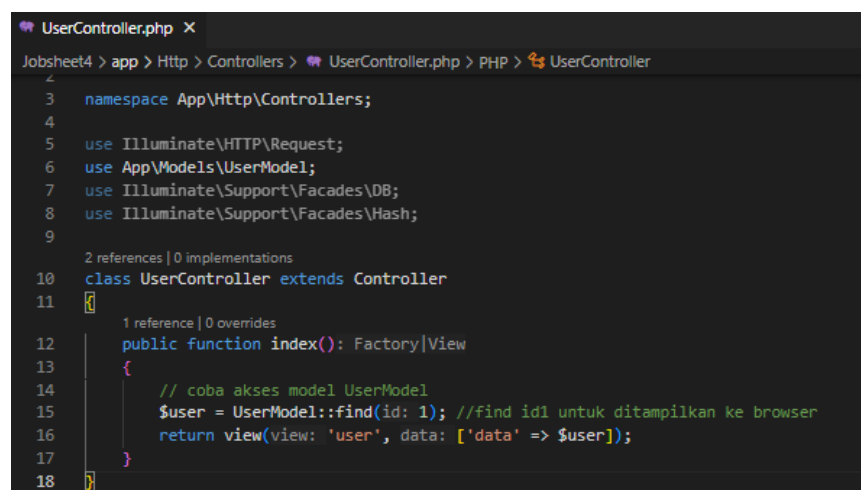
```
// Ambil model dengan kunci utamanya...
$user = UserModel::find(1);

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

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

### Praktikum 2.1 – Retrieving Single Models

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



```
UserController.php X
Jobsheet4 > app > Http > Controllers > UserController.php > PHP > UserController

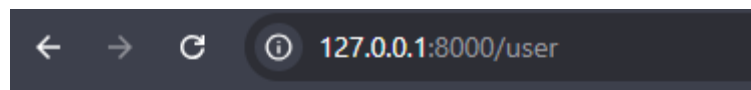
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\UserModel;
7 use Illuminate\Support\Facades\DB;
8 use Illuminate\Support\Facades\Hash;
9
10 2 references | 0 implementations
11 class UserController extends Controller
12 {
13     1 reference | 0 overrides
14     public function index(): Factory|View
15     {
16         // coba akses model UserModel
17         $user = UserModel::find(id: 1); //find id1 untuk ditampilkan ke browser
18         return view('user', data: ['data' => $user]);
19     }
20 }
```



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



## Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1

### Jawab :

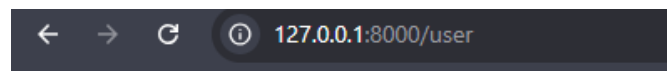
- Perubahan atau modifikasi pada `UserController` untuk `find(1)` berarti `find` data dengan id 1.
- Perubahan pada file *view* -> `user blade` untuk menampilkan data tanpa perulangan (menampilkan id 1 saja) yaitu data administrator.

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



```
UserController.php x user.blade.php
Jobsheet4 > app > Http > Controllers > UserController.php > PHP Intelephense > UserController > index
5 use Illuminate\Http\Request;
6 use App\Models\UserModel;
7 use Illuminate\Support\Facades\DB;
8 use Illuminate\Support\Facades\Hash;
9
10 2 references | 0 implementations
11 class UserController extends Controller
12 {
13     1 reference | 0 overrides
14     public function index(): Factory|View
15     {
16         // // find berdasarkan id
17         // $user = UserModel::find(1); //find id1 untuk ditampilkan ke browser
18         // return view('user', ['data' => $user]);
19
20         $user = UserModel::where(column: 'level_id', operator: 1) -> first(); //mencari data UserModel di mana kolom level_id = 1
21         return view(view: 'user', data: ['data' => $user]);
22     }
23 }
```

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



## Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1

### Jawab :

- Where untuk mencari data dalam tabel yang terkait dengan model `UserModel` di mana kolom `level_id` memiliki nilai `1`.
- `->first()` ; → Mengambil satu data pertama yang ditemukan (bukan dalam bentuk array/list, tetapi sebagai satu objek).

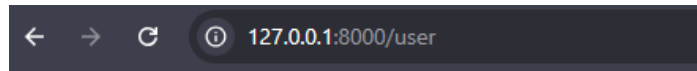
6. Ubah file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

```
UserController.php M x user.blade.php M
Mingg4 > Jobsheet4 > app > Http > Controllers > UserController.php > PHP Intelephense > UserController > index
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\UserModel;
7 use Illuminate\Support\Facades\DB;
8 use Illuminate\Support\Facades\Hash;
9
10 8 references | 0 implementations
11 class UserController extends Controller
12 {
13     3 references | 0 overrides
14     public function index(): Factory|View
15     {
16         // find berdasarkan id
17         // $user = UserModel::find(1); // find id1 untuk ditampilkan ke browser
18         // return view('user', ['data' => $user]);
19
20         // where
21         // $user = UserModel::where('level_id', 1) -> first(); // where id1 untuk ditampilkan ke browser
22         // return view('user', ['data' => $user]);
23
24         //firstwhere
25         $user = UserModel::firstwhere('level_id', 1) -> first(); // mencari data dimana kolom level_id = 1
26         return view(view: 'user', data: ['data' => $user]);
27     }
28 }
```





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



## Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1

**Jawab :** Hasil nya sama dengan menggunakan where, hanya saja penulisannya yang berbeda

- `firstWhere('level_id', 1)` → Ini adalah cara yang lebih ringkas untuk menulis `where('level_id', 1)->first()`.
- Fungsi ini akan mencari satu **data pertama** di tabel `UserModel` di mana kolom `level_id = 1`.
- Jika data ditemukan, maka `$user` akan berisi **satu objek user**. Jika tidak ada data yang sesuai, maka `$user` akan bernilai `null`.

Terkadang Anda mungkin ingin melakukan beberapa tindakan lain jika tidak ada hasil yang ditemukan. Metode `findOr` and `firstOr` akan mengembalikan satu contoh model atau, jika tidak ada hasil yang ditemukan maka akan menjalankan didalam fungsi. Nilai yang dikembalikan oleh fungsi akan dianggap sebagai hasil dari metode ini:

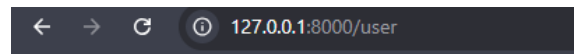
```
$user = UserModel::findOr(1, function () {  
    // ...  
});  
  
$user = UserModel::where('level_id', '>', 3)->firstOr(function () {  
    // ...  
});
```

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



```
22 // firstWhere
23 // $user = UserModel::firstWhere('level_id', 1); //mencari data UserModel di mana kolom level_id = 1
24 // return view('user', ['data' => $user]);
25
26 $user = UserModel::findOr(id: 1, columns: ['username', 'nama'], callback: function(): never {
27     abort (code: 404);
28 });
29
30 return view(view: 'user', data: ['data' => $user]);
31
32 }
```

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



## Data User

ID	Username	Nama	ID Level Pengguna
	admin	Administrator	

### Jawab :

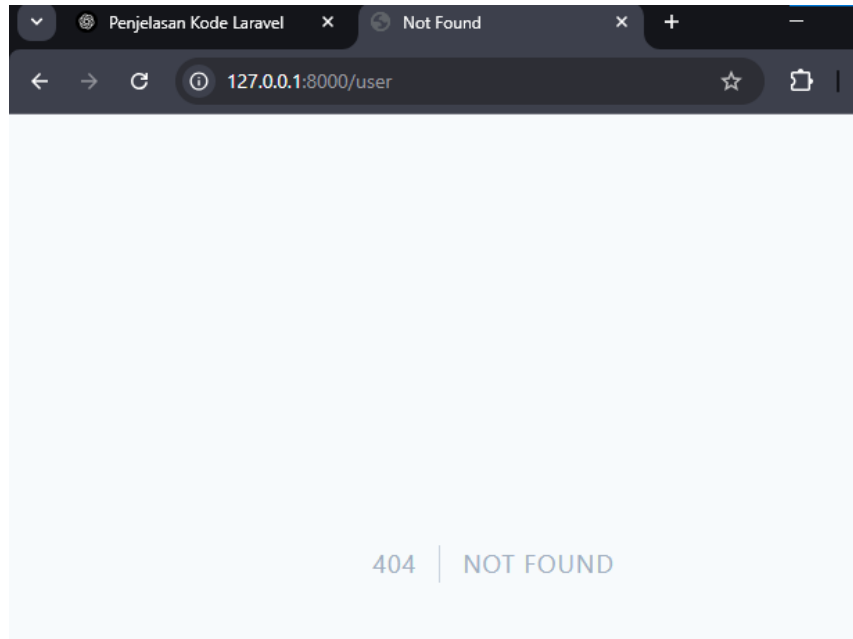
- Mencari data dalam tabel `UserModel` berdasarkan **ID = 1**.
- Hanya mengambil kolom `username` dan `nama` dari database tanpa `id` dan `level_id`
- Jika data ditemukan, hasilnya akan disimpan dalam `$user`. Jika data **tidak ditemukan**, fungsi `abort(404)` akan dijalankan, yang berarti akan menampilkan halaman **Error 404 (Not Found)**.

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

```
22 // firstWhere
23 // $user = UserModel::firstWhere('level_id', 1); //mencari data UserModel di mana kolom level_id
24 // return view('user', ['data' => $user]);
25
26 $user = UserModel::findOr(id: 20, columns: ['username', 'nama'], callback: function(): never {
27     abort (code: 404);
28 });
29
30 return view(view: 'user', data: ['data' => $user]);
31
32 }
```



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



**Jawab :** Output pada browser adalah 404 not found, karena id 20 tidak ada dalam database sehingga fungsi abort 404 dijalankan.

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

## **Praktikum 2.2** – *Not Found Exceptions*

---

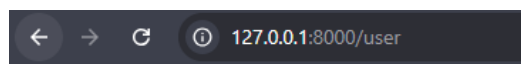
Terkadang Anda mungkin ingin memberikan pengecualian jika model tidak ditemukan. Hal ini sangat berguna dalam *route* atau pengontrol. Metode `findOrFail` and `firstOrFail` akan mengambil hasil pertama dari kueri; namun, jika tidak ada hasil yang ditemukan, sebuah `Illuminate\Database\Eloquent\ModelNotFoundException`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

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



```
Minggu4 > Jobsheet4 > app > Http > Controllers > UserController.php > PHP Intelephense >  
9  
8 references | 0 implementations  
10 class UserController extends Controller  
11 {  
12     3 references | 0 overrides  
13     public function index(): Factory|View  
14     {  
15         //PRAKTIKUM 2.1  
16         $user = UserModel::findOrFail(1);  
17         return view(view: 'user', data: ['data' => $user]);  
18     }  
19 }
```

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



## Data User

ID	Username	Nama	ID Level Pengguna
1	admin	Administrator	1

**Jawab :** `findOrFail(1)` merupakan method Eloquent yang mencari data user dengan ID 1. Karena ID 1 ada dalam database sehingga output nya memunculkan data administrator.

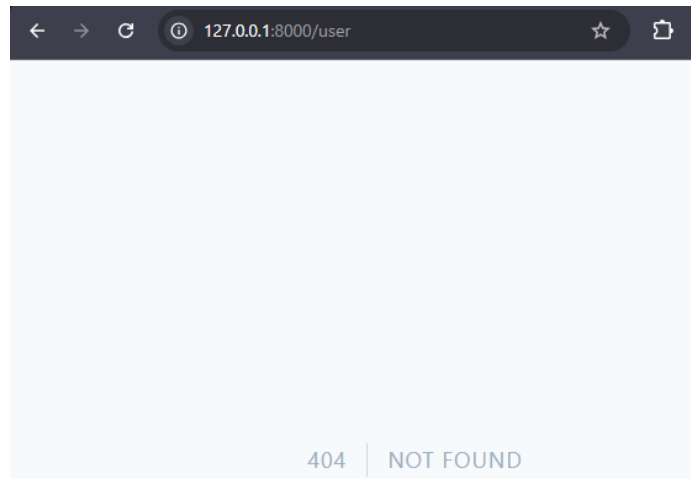
- Jika data ditemukan, maka data tersebut akan disimpan dalam variabel `$user`.
- Jika data tidak ditemukan, maka akan muncul error 404 Not Found.

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

```
10 class UserController extends Controller  
11 {  
12     3 references | 0 overrides  
13     public function index(): Factory|View  
14     {  
15         //PRAKTIKUM 2.1  
16  
17         // find or fail  
18         // $user = UserModel::findOrFail(1);  
19         // return view('user', ['data' => $user]);  
20  
21         $user = UserModel::where('username', 'manager9')->firstOrFail();  
22         return view(view: 'user', data: ['data' => $user]);  
23     }  
24 }
```



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



**Jawab :** Output dari kode tersebut adalah 404 Not Found, itu berarti data user dengan username 'manager9' tidak ditemukan di database.

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

### Praktikum 2.3 – Retrieving Aggregates

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

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

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

```
10 class UserController extends Controller  
11  
12     3 references | 0 overrides  
13     public function index(): Factory|View  
14     {  
15         //PRAKTIKUM 2.3  
16         $user = UserModel::where('level_id', 2)->count();  
17         dd(vars: $user);  
18         return view('user', data: ['data' => $user]);  
19     }
```

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



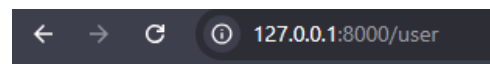
terjadi dan beri penjelasan dalam laporan

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

```
12 // references | 0 overrides  
13 public function index(): Factory|View  
14 {  
15     //PRAKTIKUM 2.3  
16     // Menghitung jumlah pengguna dengan level_id = 2  
17     $jumlahPengguna = UserModel::where('level_id', 2)->count();  
18     // Mengirim data ke view  
19     return view(view: 'user', data: ['jumlahPengguna' => $jumlahPengguna]);  
20 }  
21
```

- View user.blade.php



## Data User

Jumlah Pengguna
2

```
Minggu4 > Jobsheet4 > resources > views > user.blade.php  
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4     <title>Data User</title>  
5 </head>  
6 <body>  
7     <h1>Data User</h1>  
8     <table border="1" cellpadding="2" cellspacing="0">  
9         <tr>  
10             <th>Jumlah Pengguna</th>  
11         </tr>  
12         <tr>  
13             <td>{{ $jumlahPengguna }}</td>  
14         </tr>  
15     </table>  
16 </body>  
17 </html>
```

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



## Praktikum 2.4 – Retrieving or Creating Models

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

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

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

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

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

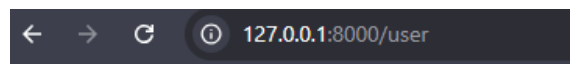
```
UserController.php M x user.blade.php
Minggu4 > Jobsheet4 > app > Http > Controllers > UserController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use App\Models\UserModel;
7  use Illuminate\Support\Facades\DB;
8  use Illuminate\Support\Facades\Hash;
9
10 8 references | 0 implementations
11 class UserController extends Controller
12 {
13     3 references | 0 overrides
14     public function index(): Factory|View
15     {
16         //PRAKTIKUM 2.4
17         $user = UserModel::firstOrCreate(
18             [
19                 'username' => 'manager',
20                 'nama' => 'Manager',
21             ],
22         );
23         return view('user', data: ['data' => $user]);
24     }
25 }
```



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

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

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



## Data User

ID	Username	Nama	ID Level Pengguna
2	manager	Manager	2

**Jawab :** `firstOrCreate()` merupakan method Eloquent yang mencari data user berdasarkan kondisi yang diberikan.

- **Kondisi:** `['username' => 'manager', 'nama' => 'Manager']`. Artinya, method ini akan mencari user dengan username 'manager' dan nama 'Manager'.
- **Jika data ditemukan:** method akan mengembalikan data user tersebut.
- **Jika data tidak ditemukan:** method akan membuat data user baru dengan atribut username 'manager' dan nama 'Manager', lalu mengembalikan data user yang baru dibuat.
- Hasil dari `firstOrCreate()` (data yang ditemukan atau yang baru dibuat) akan disimpan dalam variabel `$user`.





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

				user_id	level_id	username	nama
<input type="checkbox"/>	Edit	Copy	Delete	1	1	admin	Administrator
<input type="checkbox"/>	Edit	Copy	Delete	2	2	manager	Manager
<input type="checkbox"/>	Edit	Copy	Delete	3	3	staff	Staf/Kasir
<input type="checkbox"/>	Edit	Copy	Delete	4	4	customer-1	Customer 1
<input type="checkbox"/>	Edit	Copy	Delete	16	4	customer-2	Pelanggan Kedua
<input type="checkbox"/>	Edit	Copy	Delete	18	2	manager_dua	Manager 2



**Jawab :** Terjadi error karena kolom password tidak memiliki nilai default maka data tidak dapat tersimpan di database. Karena. Dalam database tidak ada username

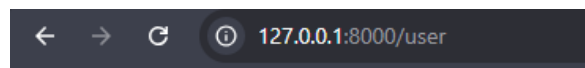


manager22 sehingga saat `firstOrCreate` mencoba membuat data baru dan data baru tersebut tidak memiliki nilai `password` maka terjadilah error.

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

```
14 //PRAKTIKUM 2.4
15 $user = UserModel::firstOrCreate(
16     [
17         'username' => 'manager',
18         'nama' => 'Manager',
19     ]
20 );
21 return view(view: 'user', data: ['data' => $user]);
22 }
23 }
```

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



## Data User

ID	Username	Nama	ID Level Pengguna
2	manager	Manager	2

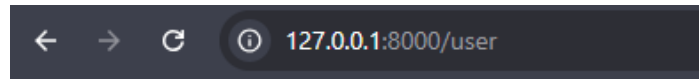
**Jawab :** Metode `firstOrCreate` mencoba mencari data di database berdasarkan kondisi yang diberikan. Jika data ditemukan, maka data tersebut dikembalikan sebagai objek **tanpa menyimpannya ke database**. Namun, jika tidak ditemukan, Laravel akan membuat objek baru dengan nilai tersebut, tetapi **tidak langsung menyimpan ke database**.

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

```
12 public function index(): Factory|View
13 {
14     //PRAKTIKUM 2.4
15     $user = UserModel::firstOrCreate(
16         [
17             'username' => 'manager33',
18             'nama' => 'Manager Tiga Tiga',
19             'password' => Hash::make('12345'),
20             'level_id' => 2
21         ]
22     );
23
24     return view(view: 'user', data: ['data' => $user]);
25 }
26 }
```



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

		user_id	level_id	username	nama	password
<input type="checkbox"/>	Edit Copy Delete	1	1	admin	Administrator	\$2y\$12\$/Hh.yEKKI
<input type="checkbox"/>	Edit Copy Delete	2	2	manager	Manager	\$2y\$12\$/V.DotlrqFf
<input type="checkbox"/>	Edit Copy Delete	3	3	staff	Staf/Kasir	\$2y\$12\$/7uXHG2s
<input type="checkbox"/>	Edit Copy Delete	4	4	customer-1	Customer 1	\$2y\$12\$/2UmDOiki
<input type="checkbox"/>	Edit Copy Delete	16	4	customer-2	Pelanggan Kedua	\$2y\$12\$/6a.ks.Qro
<input type="checkbox"/>	Edit Copy Delete	18	2	manager_dua	Manager 2	\$2y\$12\$/XjoID6ithn

**Jawab:** Metode **firstOrCreate** mencoba mencari data di database berdasarkan kondisi yang diberikan. Karena data **manager33** tidak ditemukan di database, Laravel akan membuat objek baru dengan nilai tersebut, tetapi **tidak langsung menyimpan ke database**. Tampilan yang diberikan sesuai dengan kode program hanya menampilkan username, nama, dan level id.

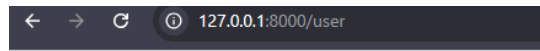
10. Ubah file controller dengan nama **UserController.php** dan ubah *script* seperti gambar di bawah ini

```
14 //PRAKTIKUM 2.4
15 $user = UserModel::firstOrCreate(
16     [
17         'username' => 'manager33',
18         'nama' => 'Manager Tiga Tiga',
19         'level_id' => 2
20     ]
21 );
22 $user->password = Hash::make('12345');
23 $user->save();
24
25 return view('user', data: ['data' => $user]);
26 }
27 }
```

Lakukan sedikit modifikasi agar data tersimpan dalam database, letakkan **password** sebelum statement **\$user -> save()** atau bisa menambahkan password pada **\$fillable UserModel.php**



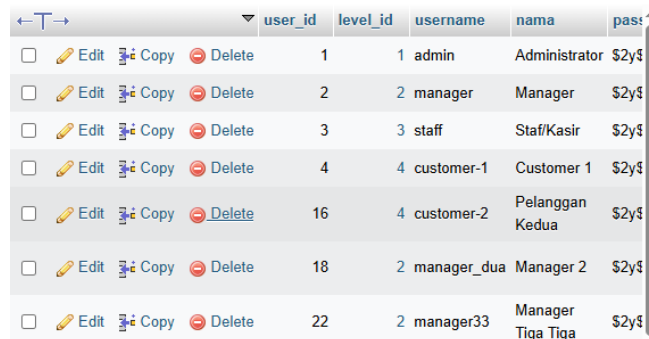
11. Simpan kode program Langkah 9. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel *m\_user* serta beri penjelasan dalam laporan



### Data User

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

**Jawab :** Jika terjadi error, bisa dilakukan modifikasi agar data tersimpan dalam database. Cara nya letakkan password sebelum statement \$user -> save( ) atau bisa menambahkan password pada \$fillable UserModel.php



	user_id	level_id	username	nama	password
<input type="checkbox"/> Edit Copy Delete	1	1	admin	Administrator	\$2y\$
<input type="checkbox"/> Edit Copy Delete	2	2	manager	Manager	\$2y\$
<input type="checkbox"/> Edit Copy Delete	3	3	staff	Staf/Kasir	\$2y\$
<input type="checkbox"/> Edit Copy Delete	4	4	customer-1	Customer 1	\$2y\$
<input type="checkbox"/> Edit Copy Delete	16	4	customer-2	Pelanggan Kedua	\$2y\$
<input type="checkbox"/> Edit Copy Delete	18	2	manager_dua	Manager 2	\$2y\$
<input type="checkbox"/> Edit Copy Delete	22	2	manager33	Manager Tiga Tiga	\$2y\$

Pada dasarnya :

- `firstOrCreate` → Harus dipanggil `save()` secara manual.
- `firstOrCreate` → Langsung menyimpan 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:

```
$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();

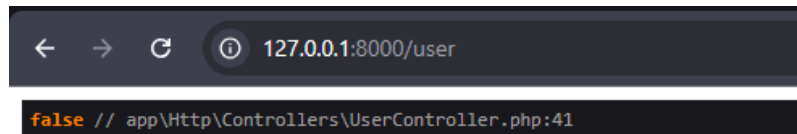
$user->isDirty(); // false
$user->isClean(); // true
```

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

```
13 //PRAKTIKUM 2.5
14 $user = UserModel::create([
15     'username' => 'manager55',
16     'nama' => 'Manager55',
17     'password' => Hash::make('12345'),
18     'level_id' => 2,
19 ]);
20
21 $user->username = 'manager56';
22
23 $user->isDirty(); // true
24 $user->isDirty('username'); // true
25
26 $user->isDirty('nama'); // false
27 $user->isDirty(['nama', 'username']); // true
28
29 $user->isClean(); // false
30 $user->isClean('username'); // false
31
32 $user->isClean('nama'); // true
33 $user->isClean(['nama', 'username']); // false
34
35 $user->save();
36
37 $user->isDirty(); // false
38 $user->isClean(); // true
39
40 dd(vars: $user->isDirty());
41
42
```



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



**Jawab :** `isDirty()` → Mengecek apakah ada perubahan yang belum disimpan. Setelah `save()`, perubahan disimpan, dan model menjadi bersih (`isDirty() == false`). Alur nya sebagai berikut :

- **Membuat user baru** dengan `username = 'manager55'`.
- **Ubah username** → dari `manager55` ke `manager56`, tapi belum disimpan.
- **Cek apakah ada perubahan?**
  - `isDirty()` → `true` (karena username berubah).
  - `isClean()` → `false` (karena ada perubahan).
- **Simpan ke database** → `save()`.
- **Cek kembali**
  - `isDirty()` → `false` (karena sudah disimpan).
  - `isClean()` → `true` (sekarang tidak ada perubahan).
- **Tampilkan hasil** → `false` di browser karena tidak ada perubahan setelah `save()`.

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

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

        $user->username = 'manager12';

        $user->save();

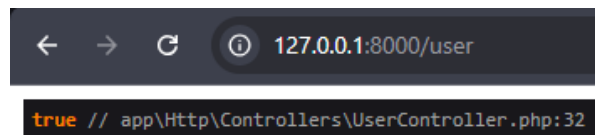
        $user->wasChanged(); // true
        $user->wasChanged('username'); // true
        $user->wasChanged(['username', 'level_id']); // true
        $user->wasChanged('nama'); // false
        $user->wasChanged(['nama', 'username']); // true
    }
}
```



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

```
14 //PRAKTIKUM 2.5
15 $user = UserModel::create([
16     'username' => 'manager11',
17     'nama' => 'Manager11',
18     'password' => Hash::make('12345'),
19     'level_id' => 2,
20 ]);
21
22 $user->username = 'manager12';
23
24 $user->save();
25
26 $user->wasChanged(); // true
27 $user->wasChanged('username'); // true
28
29 $user->wasChanged(['username', 'level_id']); // true
30 $user->wasChanged('nama'); // false
31
32 dd(vars: $user->wasChanged(['nama', 'username'])); // true
33
34 }
```

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



**Jawab :** `wasChanged()` digunakan untuk **mengecek apakah suatu atribut dalam model berubah setelah disimpan ke database (`save()`)**

- Berbeda dengan `isDirty()`, yang mengecek perubahan **sebelum** disimpan, `wasChanged()` hanya bisa digunakan **setelah `save()` dipanggil**.
- Berikut alurnya :
  - **Buat user baru** → `manager11` disimpan ke database.
  - **Ubah `username`** → dari `'manager11'` ke `'manager12'`, tapi `nama` tetap.
  - **Simpan ke database** → `save()`.
  - **Cek perubahan dengan `wasChanged()`**
    - `wasChanged('username')` → `true` (karena `username` berubah).
    - `wasChanged('nama')` → `false` (karena `nama` tidak berubah).
    - `wasChanged(['nama', 'username'])` → `true` (karena **ada 1 perubahan, yaitu `username`**).
  - **Hasil `true` ditampilkan di browser** karena ada perubahan di `username`

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 scripnya menjadi seperti di bawah ini

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Data User</title>
5 </head>
6 <body>
7     <h1>Data User</h1>
8     <a href="/user/tambah">+ Tambah User</a>
9     <table border="1" cellpadding="2" cellspacing="0">
10         <tr>
11             <td>ID</td>
12             <td>Username</td>
13             <td>Nama</td>
14             <td>ID Level Pengguna</td>
15             <td>Aksi</td>
16         </tr>
17         @foreach ($data as $d)
18             <tr>
19                 <td>{{ $d->user_id }}</td>
20                 <td>{{ $d->username }}</td>
21                 <td>{{ $d->nama }}</td>
22                 <td>{{ $d->level_id }}</td>
23                 <td><a href="/user/ubah/{{ $d->user_id }}">Ubah</a> | <a href="/user/hapus/{{ $d->user_id }}">Hapus</a></td>
24             </tr>
25         @endforeach
26     </table>
27 </body>
28 </html>
```

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

```
10 class UserController extends Controller
11 {
12     3 references | 0 overrides
13     public function index(): Factory|View
14     {
15         //PRAKTIKUM 2.6
16         $user = UserModel::all();
17         return view(view: 'user', data: ['data' => $user]);
18     }
19 }
```

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





## Data User

[+ Tambah User](#)

ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	<a href="#">Ubah</a>   <a href="#">Hapus</a>
2	manager	Manager	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
3	staff	Staf/Kasir	3	<a href="#">Ubah</a>   <a href="#">Hapus</a>
4	customer-1	Customer 1	4	<a href="#">Ubah</a>   <a href="#">Hapus</a>
16	customer-2	Pelanggan Kedua	4	<a href="#">Ubah</a>   <a href="#">Hapus</a>
18	manager_dua	Manager 2	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
22	manager33	Manager Tiga Tiga	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
23	manager56	Manager55	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
24	manager12	Manager11	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>

**Jawab :** Halaman browser menampilkan isi data pada tabel user secara keseluruhan karena menggunakan `UserModel::all()` dan mengirimkannya ke view `user.blade.php`

- `UserModel::all()` → Mengambil semua data dari tabel sebagai **koleksi** (**Collection**).
- `return view('user', ['data' => $user]);` → Mengirimkan data ke view dengan variabel `data`.

- 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

```
Minggu4 > Jobsheet4 > resources > views > user_tambah.blade.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Form Tambah Data User</title>
7 </head>
8 <body>
9     <h1>Form Tambah Data User</h1>
10    <form method="post" action="/user/tambah_simpan">
11        {{ csrf_field() }}
12        <label>Username</label>
13        <input type="text" name="username" placeholder="Masukan Username">
14        <br>
15        <label>Nama</label>
16        <input type="text" name="nama" placeholder="Masukan Nama">
17        <br>
18        <label>Password</label>
19        <input type="password" name="password" placeholder="Masukan Password">
20        <br>
21        <label>Level ID</label>
22        <input type="number" name="level_id" placeholder="Masukan ID Level">
23        <br><br>
24        <input type="submit" class="btn btn-success" value="Simpan">
25    </form>
26 </body>
27 </html>
```



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

```
26 | Route::get('/user/tambah', [UserController::class, 'tambah']); // route tambahan untuk user tambah data
```

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

```
10 class UserController extends Controller
11 {
12     3 references | 0 overrides
13     public function index(): Factory|View
14     {
15         //PRAKTIKUM 2.6
16         $user = UserModel::all();
17         return view(view: 'user', data: ['data' => $user]);
18     }
19     1 reference | 0 overrides
20     public function tambah(): Factory|View
21     {
22         return view(view: 'user.tambah');
23     }
24 }
```

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

← → ↻ ⓘ 127.0.0.1:8000/user/tambah

## Form Tambah Data User

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

**Jawab :** Halaman browser akan memunculkan form tambah user. Route untuk mengatur alamat route ke `/user/tambah`. UserController dilakukan penambahan function tambah dan untuk mereturn view ke browser. View akan memuat tampilan dari halaman `user_tambah.blade.php`



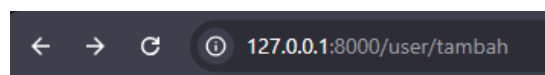
8. Tambahkan *script* pada *routes* dengan nama file **web.php**. Tambahkan seperti gambar di bawah ini

```
28 | Route::post('/user/tambah_simpan', [UserController::class, 'tambah_simpan']); // route tambahan untuk user simpan data
```

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

```
23 | public function tambah_simpan (Request $request): Redirector|RedirectResponse
24 | {
25 |     UserModel::create([
26 |         'username' => $request->username,
27 |         'nama' => $request->nama,
28 |         'password' => Hash::make($request->password),
29 |         'level_id' => $request->level_id
30 |     ]);
31 |     return redirect(to: '/user');
32 | }
```

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



## Form Tambah Data User

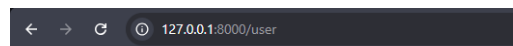
vitae a

Username

Nama

Password

Level ID



## Data User

ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	<a href="#">Ubah</a> <a href="#">Hapus</a>
2	manager	Manager	2	<a href="#">Ubah</a> <a href="#">Hapus</a>
3	staff	Staf/Kasir	3	<a href="#">Ubah</a> <a href="#">Hapus</a>
4	customer-1	Customer 1	4	<a href="#">Ubah</a> <a href="#">Hapus</a>
16	customer-2	Pelanggan Kedua	4	<a href="#">Ubah</a> <a href="#">Hapus</a>
18	manager_dua	Manager 2	2	<a href="#">Ubah</a> <a href="#">Hapus</a>
22	manager33	Manager Tiga Tiga	2	<a href="#">Ubah</a> <a href="#">Hapus</a>
23	manager56	Manager55	2	<a href="#">Ubah</a> <a href="#">Hapus</a>
24	manager12	Manager11	2	<a href="#">Ubah</a> <a href="#">Hapus</a>
25	vitaeka	vita	2	<a href="#">Ubah</a> <a href="#">Hapus</a>



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

```
Minggu4 > Jobsheet4 > resources > views > user_ubah.blade.php
1 <!DOCTYPE html>
2 <html lang="id">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Form Ubah Data User</title>
7 </head>
8 <body>
9 <h1>Form Ubah Data User</h1>
10 <a href="/user">Kembali</a>
11 <br><br>
12
13 <form method="post" action="/user/ubah_simpan/{{ $data->user_id }}">
14 <{{ csrf_field() }}>
15 <{{ method_field('PUT') }}>
16
17 <label>Username</label>
18 <input type="text" name="username" placeholder="Masukan Username" value="{{ $data->username }}">
19 <br>
20
21 <label>Nama</label>
22 <input type="text" name="nama" placeholder="Masukan Nama" value="{{ $data->nama }}">
23 <br>
24
25 <label>Password</label>
26 <input type="password" name="password" placeholder="Masukan Password" value="{{ $data->password }}">
27 <br>
28
29 <label>Level ID</label>
30 <input type="number" name="level_id" placeholder="Masukan ID Level" value="{{ $data->level_id }}">
31 <br><br>
32
33 <input type="submit" class="btn btn-success" value="Ubah">
34 </form>
35 </body>
36 </html>
```

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

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

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

```
1 reference | 0 overrides
36 public function ubah(Request $request, $id): Factory|View
37 {
38     $user = UserModel::find($id);
39     return view(view: 'user_ubah', data: ['data' => $user]);
40 }
41 }
```

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



**Form Ubah Data User**

[Kembali](#)

Username

Nama

Password

Level ID

404 | NOT FOUND

**Jawab :** Halaman not found 404 karena belum ditambahkan route setelah user mengklik button simpan dan belum ada method yang menangani perubahan data

15. Tambahkan *script* pada *routes* dengan nama file [web.php](#). Tambahkan seperti gambar di bawah ini

```
34 | 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 diletakkan di bawah method `ubah` seperti gambar di bawah ini

```
42 | public function ubah_simpan(Request $request, $id): Redirector|RedirectResponse  
43 | {  
44 |     $user = UserModel::find($id);  
45 |     $user->username = $request->username;  
46 |     $user->nama = $request->nama;  
47 |     $user->password = Hash::make($request->password);  
48 |     $user->level_id = $request->level_id;  
49 |     $user->save();  
50 |  
51 |     return redirect(to: '/user');  
52 | }
```

17. Simpan kode program Langkah 15 dan 16. Kemudian jalankan link <localhost:8000/user/ubah/1> atau [localhost/PWL\\_POS/public/user/ubah/1](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

**Form Ubah Data User**

[Kembali](#)

Username

Nama

Password

Level ID

**Data User**

ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	<a href="#">Ubah</a>   <a href="#">Hapus</a>
2	manager	Manager	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
3	staff	Sta/Kasir	3	<a href="#">Ubah</a>   <a href="#">Hapus</a>
4	customer-1	Customer 1	4	<a href="#">Ubah</a>   <a href="#">Hapus</a>
16	customer-2	Pelanggan Kedua	4	<a href="#">Ubah</a>   <a href="#">Hapus</a>
18	manager_dua	Manager 2	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
22	manager33	Manager Tiga Tiga	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
23	manager56	Manager55	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
24	manager12	Manager11	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
25	vitaeka11	vita	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>



**Jawab :** Setelah melakukan penambahan method yang menangani simpan ubah data dan route redirect, data yang telah di ubah akan otomatis tersimpan di database dan ditampilkan ke user.

Pada langkah di atas saya mengubah username yang semula “vitaeka” menjadi “vitaekal1”

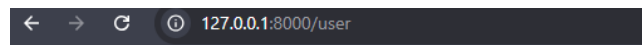
18. Berikut untuk langkah *delete* . Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

```
35 | 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

```
35 | 1 reference | 0 overrides
54 | public function hapus($id): Redirector|RedirectResponse{
55 |     $user = UserModel::find($id);
56 |     $user->delete();
57 |
58 |     return redirect(to: '/user');
59 | }
60 | }
```

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

ID	Username	Nama	ID Level Pengguna	Aksi
1	admin	Administrator	1	<a href="#">Ubah</a>   <a href="#">Hapus</a>
2	manager	Manager	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
3	staff	Staf/Kasir	3	<a href="#">Ubah</a>   <a href="#">Hapus</a>
4	customer-1	Customer 1	4	<a href="#">Ubah</a>   <a href="#">Hapus</a>
16	customer-2	Pelanggan Kedua	4	<a href="#">Ubah</a>   <a href="#">Hapus</a>
18	manager_dua	Manager 2	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
22	manager33	Manager Tiga Tiga	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
23	manager56	Manager55	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>
24	manager12	Manager11	2	<a href="#">Ubah</a>   <a href="#">Hapus</a>

**Jawab :** Disini saya melakukan hapus/delete data dengan id 25, dan username “vitaekal1”. Saat user mengklik button hapus data pada database juga akan terhapus.

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



---

## Praktikum 2.7 – Relationships

---

### One to One

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

```
<?php

namespace App\Models;

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

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

### Mendefinisikan Kebalikan dari Hubungan *One-to-one*

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

```
<?php

namespace App\Models;

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

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





## One to Many

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

```
<?php

namespace App\Models;

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

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

## One to Many (Inverse) / Belongs To

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

```
<?php

namespace App\Models;

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

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

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





```
8 use App\Models\LevelModel;
9
10 11 references | 0 implementations
11 class UserModel extends Model
12 {
13     use HasFactory;
14
15     0 references
16     protected $table = 'm_user'; // Mendefinisikan nama tabel yang digunakan oleh model ini
17     0 references
18     protected $primaryKey = 'user_id'; // Mendefinisikan primary key dari tabel yang digunakan
19
20     0 references
21     protected $fillable = ['username', 'nama', 'password', 'level_id'];
22
23     0 references | 0 overrides
24     public function level(): BelongsTo
25     {
26         return $this->belongsTo(LevelModel::class, 'level_id', 'level_id');
27     }
28 }
```

- **Buat LevelModel.php**

```
user.blade.php | UserController.php M | user_ubah.blade.php | UserModel.php M | LevelModel.php U X |
Minggu4 > Jobsheet4 > app > Models > LevelModel.php > ...
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 2 references | 0 implementations
9 class LevelModel extends Model
10 {
11     use HasFactory;
12
13     0 references
14     protected $table = 'm_level'; // Mendefinisikan nama tabel yang digunakan oleh model ini
15     0 references
16     protected $primaryKey = 'level_id'; // Mendefinisikan primary key dari tabel yang digunakan
17
18     0 references
19     protected $fillable = ['level_kode', 'level_nama']; // Mendefinisikan kolom yang bisa diisi oleh user
20 }
```

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

```
3 references | 0 overrides
12 public function index(): never
13 {
14     //praktikum 2.7
15     $user = UserModel::with('level')->get();
16     dd(vars: $user);
17 }
18 }
```



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

```
← → ↺ 127.0.0.1:8000/user  
Illuminate\Database\Eloquent\Collection {#319 ▼ // app\Http\Controllers\UserController.php:16  
  #items: array:9 [▶]  
  #escapeWhenCastingToString: false  
}
```

**Jawab :** Dari tampilan ini, data sebenarnya sudah diambil dari database, tetapi langsung **mencetak koleksi (Collection) tanpa diubah ke format yang bisa dibaca di Blade**. Jika dilihat jumlahnya ada 9 rows.

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

```
14 //praktikum 2.7  
15 $user = UserModel::with('level')->get();  
16 return view('user', data: ['data' => $user]);  
17  
18 }
```

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

```
Minggu4 > Jobsheet4 > resources > views > user.blade.php  
1 <h1>Data User</h1>  
2 <a href="/user/tambah">+ Tambah User</a>  
3 <table border="1" cellpadding="2" cellspacing="0">  
4   <tr>  
5     <td>ID</td>  
6     <td>Username</td>  
7     <td>Nama</td>  
8     <td>ID Level Pengguna</td>  
9     <td>Kode Level</td>  
10    <td>Nama Level</td>  
11    <td>Aksi</td>  
12  </tr>  
13  @foreach ($data as $d)  
14    <tr>  
15      <td>{{ $d->user_id }}</td>  
16      <td>{{ $d->username }}</td>  
17      <td>{{ $d->nama }}</td>  
18      <td>{{ $d->level_id }}</td>  
19      <td>{{ $d->level->level_kode }}</td>  
20      <td>{{ $d->level->level_nama }}</td>  
21      <td>  
22        <a href="/user/ubah/{{ $d->user_id }}">Ubah</a> |  
23        <a href="/user/hapus/{{ $d->user_id }}">Hapus</a>  
24      </td>  
25    </tr>  
26  @endforeach  
27 </table>
```



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	<a href="#">Ubah</a>   <a href="#">Hapus</a>
2	manager	Manager	2	MNG	Manager	<a href="#">Ubah</a>   <a href="#">Hapus</a>
3	staff	Staf/Kasir	3	STF	Staf/Kasir	<a href="#">Ubah</a>   <a href="#">Hapus</a>
4	customer-1	Customer 1	4	CUS	Pelanggan	<a href="#">Ubah</a>   <a href="#">Hapus</a>
16	customer-2	Pelanggan Kedua	4	CUS	Pelanggan	<a href="#">Ubah</a>   <a href="#">Hapus</a>
18	manager_dua	Manager 2	2	MNG	Manager	<a href="#">Ubah</a>   <a href="#">Hapus</a>
22	manager33	Manager Tiga Tiga	2	MNG	Manager	<a href="#">Ubah</a>   <a href="#">Hapus</a>
23	manager56	Manager55	2	MNG	Manager	<a href="#">Ubah</a>   <a href="#">Hapus</a>
24	manager12	Manager11	2	MNG	Manager	<a href="#">Ubah</a>   <a href="#">Hapus</a>

**Jawab :** relasi **belongsTo** digunakan untuk menunjukkan bahwa suatu model memiliki hubungan **invers (kebalikan)** dari relasi **one-to-one**. **UserModel** memiliki atribut **level\_id** yang merupakan **foreign key (FK)** yang menghubungkan setiap user dengan **LevelModel**

- **Relasi UserModel dengan LevelModel**
  - **UserModel** memiliki **level\_id** sebagai **foreign key (FK)** yang menghubungkan setiap user ke **LevelModel**.
  - Menggunakan **belongsTo()**, setiap user memiliki **satu level terkait** (misal: Admin, Manager, dll.).
- **Tampilan View (user.blade.php)**
  - Menampilkan **informasi user + nama level** dalam satu tabel.
  - ORM otomatis menghubungkan data tanpa perlu query manual.

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