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) |
| NAMA | : | LYRA FAIQAH BILQIS |
| KELAS | : | 2A SIB |
| ABSEN | : | 19 |

# JOBSHEET 04
# MODEL dan ELOQUENT ORM

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

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

Sesuai dengan `Studi Kasus PWL.pdf`.
Jadi project Laravel 10 kita masih sama dengan menggunakan repositori **PWL_POS.**

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

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

**Kelebihan - Kelebihan Menggunakan ORM**

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

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

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

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

### 1. `$fillable`

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

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

**Praktikum 1** - $fillable:

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

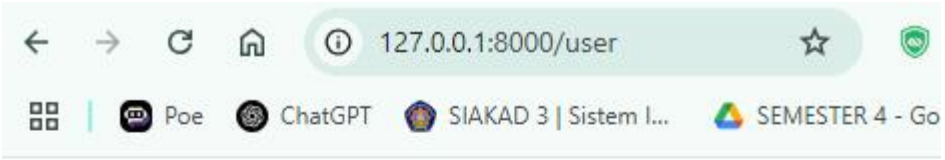
```
class UserModel extends Model
{
    use HasFactory;

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

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

```php
<?php

namespace App\Http\Controllers;

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

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

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

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

Berikut :




# Data User

| ID | Username | Nama | ID Level Pengguna |
|---|---|---|---|
| 1 | admin | Administrator | 1 |
| 2 | manager | Manager | 2 |
| 3 | staff | Staff/Kasir | 3 |
| 11 | manager_dua | Manager 2 | 2 |

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

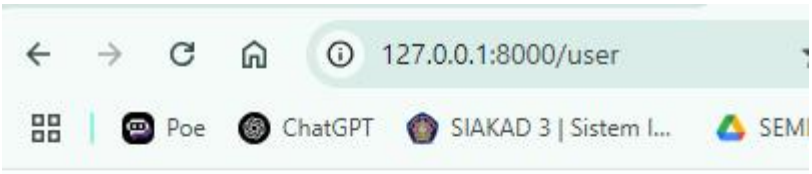
```php
protected $fillable = ['level_id', 'username', 'nama'];
```

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

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

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

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

Berikut :




# Data User

| ID | Username | Nama | ID Level Pengguna |
|---|---|---|---|
| 1 | admin | Administrator | 1 |
| 2 | manager | Manager | 2 |
| 3 | staff | Staff/Kasir | 3 |
| 11 | manager_dua | Manager 2 | 2 |
| 12 | manager_tiga | Manager 3 | 2 |

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

**2.** `$guarded`

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

## B. RETRIEVING SINGLE MODELS

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

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

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

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

**Praktikum 2.1** – Retrieving Single Models

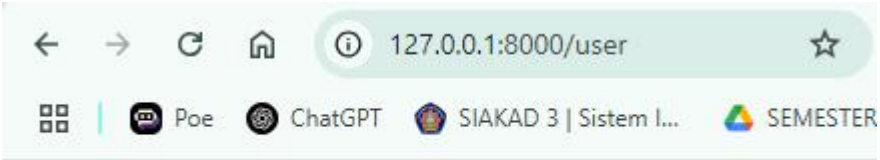
1. Buka file controller dengan nama `UserController.php` dan ubah *script* seperti gambar di bawah ini

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

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

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

3. Simpan kode program Langkah 1 dan 2. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Halaman /user berhasil menampilkan data pengguna dengan ID 1 dari tabel m_user. Controller menggunakan UserModel::find(1); untuk mengambil data pengguna berdasarkan ID, lalu mengirimkannya ke tampilan user.blade.php. Di dalam tampilan, data ditampilkan dalam bentuk tabel dengan kolom ID, Username, Nama, dan ID Level Pengguna. Jika data dengan ID 1 ada di database, maka informasi pengguna tersebut akan muncul di tabel seperti yang terlihat pada hasil tampilan.



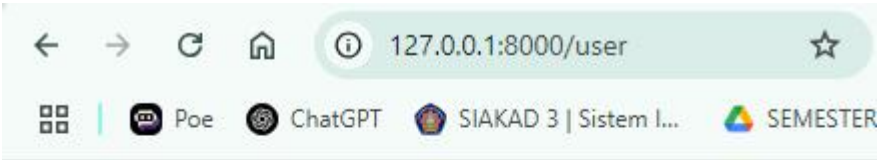

# Data User

| ID | Username | Nama | ID Level Pengguna |
|---|---|---|---|
| 1 | admin | Administrator | 1 |

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

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

5. Simpan kode program Langkah 4. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Kode ini digunakan untuk menampilkan data satu pengguna pertama yang memiliki level_id = 1 di dalam view user.



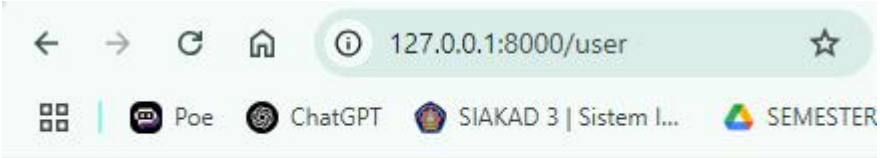

# Data User

| ID | Username | Nama | ID Level Pengguna |
|---|---|---|---|
| 1 | admin | Administrator | 1 |

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

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

7. Simpan kode program Langkah 6. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Kode tersebut digunakan untuk mengambil satu data pertama dari tabel m_user yang memiliki level_id bernilai 1, lalu mengirimkannya ke tampilan (view) bernama user.




# Data User

| ID | Username | Nama | ID Level Pengguna |
|---|---|---|---|
| 1 | admin | Administrator | 1 |

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

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

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

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

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

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

9. Simpan kode program Langkah 8. Kemudian pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Kode tersebut mencari data pengguna dengan `id = 1` dalam model `UserModel`. Jika data ditemukan, hanya kolom `username` dan `nama` yang akan diambil. Namun, jika data tidak ditemukan, fungsi callback akan dipanggil untuk menjalankan `abort(404)`, yang berarti menampilkan halaman **404 Not Found**. Fungsi ini berguna untuk memastikan bahwa jika data tidak tersedia, aplikasi akan langsung memberikan respons error yang sesuai, sehingga tidak menampilkan halaman kosong atau error yang tidak terkontrol.

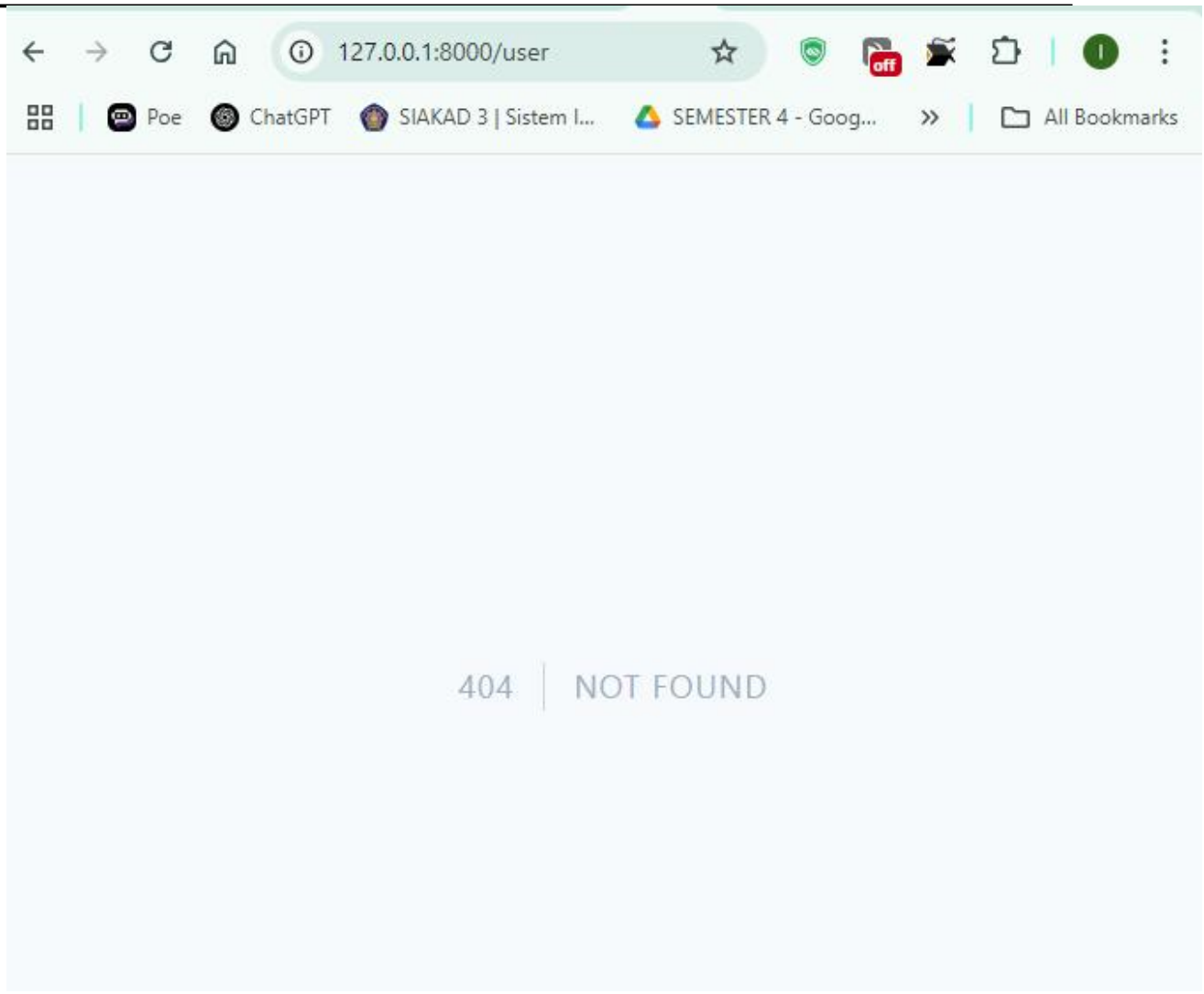

# Data User

| ID | Username | Nama | ID Level Pengguna |
|---|---|---|---|
| | admin | Administrator | |

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

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

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

11. Simpan kode program Langkah 10. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Kode tersebut mencari data pengguna dengan ID 20 di dalam model `UserModel` menggunakan metode `findOr()`. Jika data ditemukan, hanya kolom `username` dan `nama` yang akan diambil. Namun, jika data dengan ID tersebut tidak ditemukan, fungsi `abort(404)` akan dijalankan, yang akan menampilkan halaman error 404 (Not Found). Setelah itu, data yang ditemukan atau hasil dari pencarian tersebut akan dikirim ke tampilan `user` dalam bentuk variabel `data`.




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

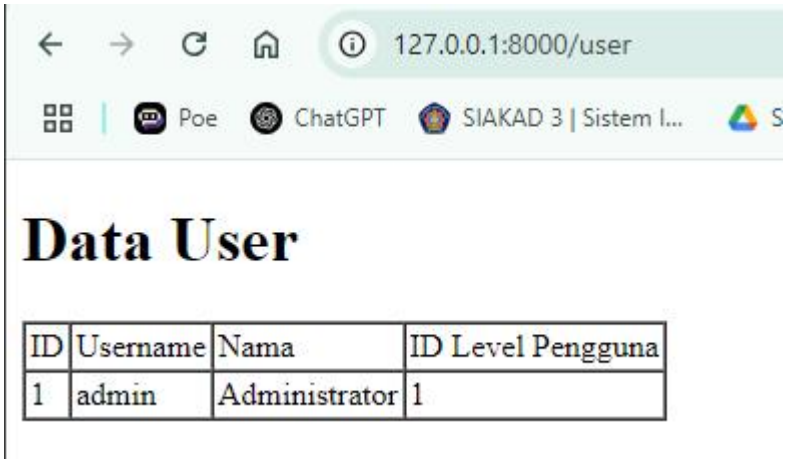
## **Praktikum 2.2** – *Not Found Exceptions*

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

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

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

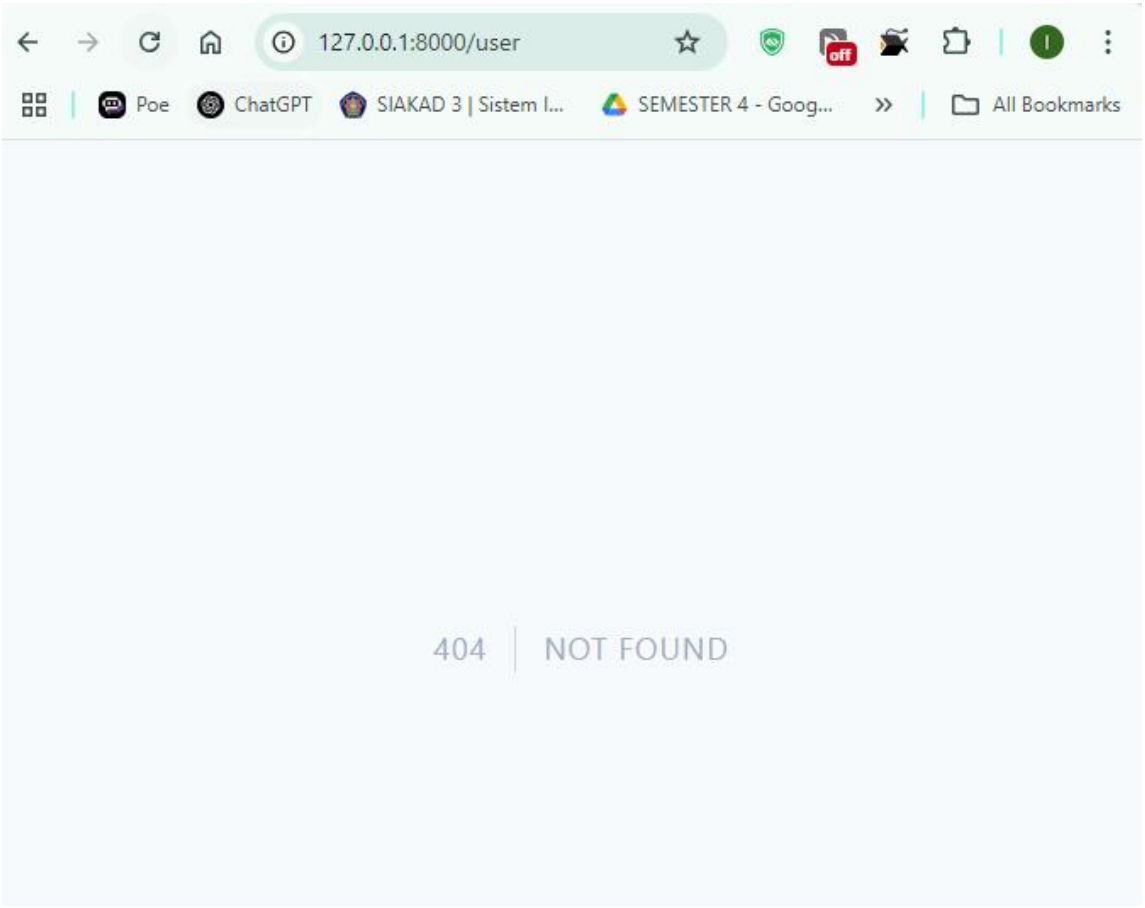
2. Simpan kode program Langkah 1. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Mengambil data pengguna berdasarkan id tertentu dari database menggunakan metode findOrFail(1). Fungsi findOrFail(1) akan mencoba mencari pengguna dengan id bernilai 1 dalam model UserModel. Jika data ditemukan, maka data tersebut akan disimpan dalam variabel $user. Namun, jika tidak ditemukan, Laravel akan otomatis menampilkan halaman error 404 (Not Found). Setelah itu, data yang diperoleh akan dikirim ke tampilan user dengan variabel data untuk ditampilkan di view.




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

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

4. Simpan kode program Langkah 3. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Kode tersebut adalah bagian dari controller dalam framework Laravel yang digunakan untuk mengambil data pengguna dari database berdasarkan username tertentu. Metode `where('username', 'manager9')` digunakan untuk mencari entri dalam tabel yang memiliki kolom `username` dengan nilai `'manager9'`. Kemudian, metode `firstOrFail()` akan mengambil entri pertama yang ditemukan, atau jika tidak ada data yang sesuai, Laravel akan secara otomatis menampilkan error 404 (Not Found). Setelah mendapatkan data pengguna, controller akan meneruskannya ke tampilan `user.blade.php` dengan menyertakan variabel `data`, yang berisi informasi pengguna yang ditemukan.



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

## **Praktikum 2.3** – *Retreiving Aggregrates*

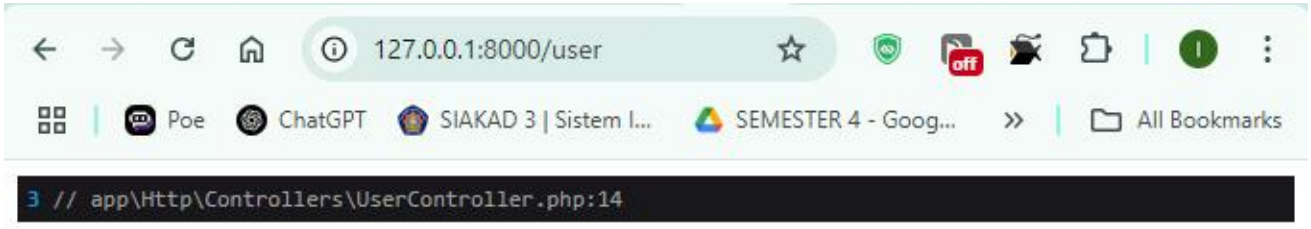
Saat berinteraksi dengan model Eloquent, Anda juga dapat menggunakan metode agregat `count`, `sum`, `max`, dan lainnya yang disediakan oleh pembuat kueri Laravel. Seperti yang Anda duga, metode ini mengembalikan nilai skalar dan contoh model Eloquent:

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

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

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

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

2. Simpan kode program Langkah 1. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Kode di atas mengambil jumlah pengguna yang memiliki `level_id` bernilai `2` dalam tabel database yang terkait dengan model `UserModel`. Fungsi `where('level_id', 2)->count()` menghitung jumlah entri yang memenuhi kondisi tersebut. Kemudian, fungsi `dd($user);` digunakan untuk menampilkan hasil perhitungan dan menghentikan eksekusi kode, sehingga halaman akan menampilkan jumlah pengguna dengan `level_id` 2 tanpa melanjutkan ke `return view(...)`. Akibatnya, tampilan yang seharusnya dikirim ke `view('user', ['data' => $user])` tidak akan pernah ditampilkan karena proses sudah dihentikan lebih awal oleh `dd()`.



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. Sudah

A) User.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Data User</title>
</head>
<body>
    <h1>Data User</h1>
    <table border="1" cellpadding="2" cellspacing="0">
        <tr>
            <td>Jumlah Pengguna</td>
        </tr>
        <tr>
            <td>{{ $jumlah_pengguna }}</td>
        </tr>
    </table>
</body>
</html>
```

B) UserController.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()
    {
        $userCount = UserModel::where('level_id', 2)->count();
        return view('user', ['jumlah_pengguna' => $userCount]);
    }
}
```

# Data User

| Jumlah Pengguna |
| --- |
| 2 |

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

## **Praktikum 2.4** – *Retreiving or Creating Models*

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

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

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

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

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

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

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

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

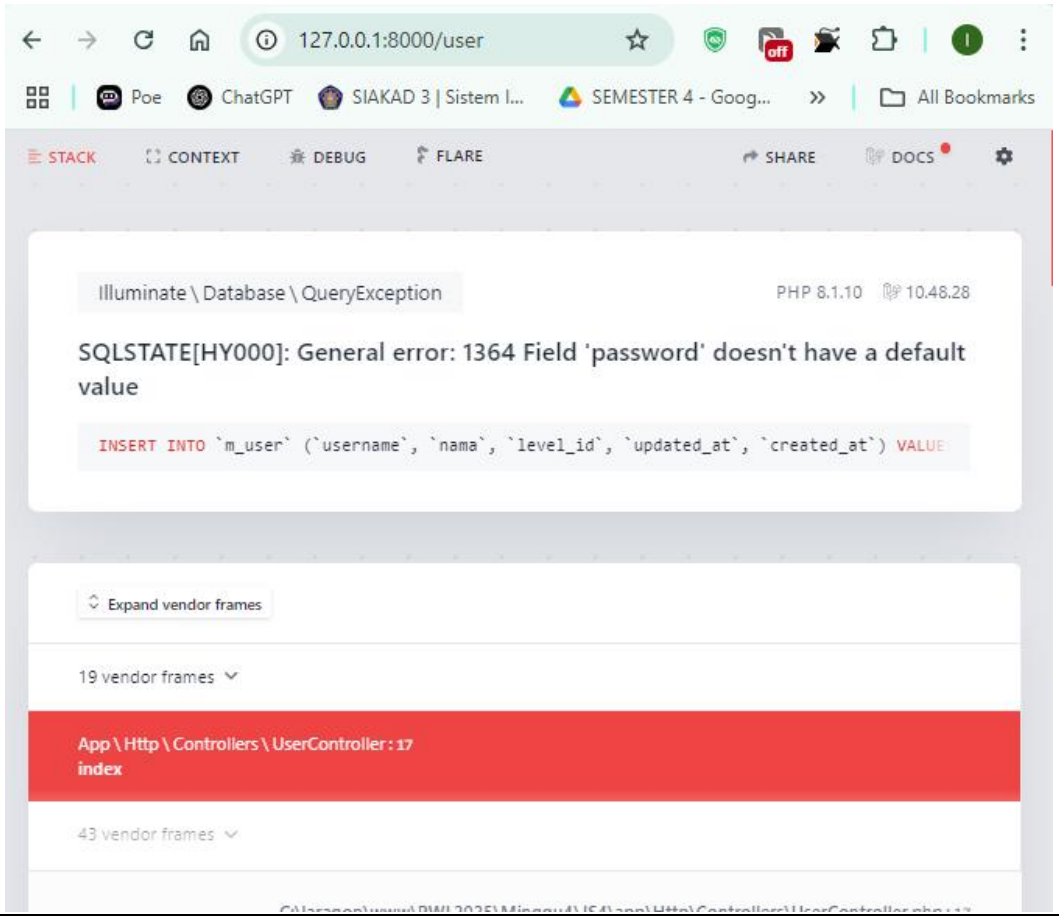
```php
<body>
    <h1>Data User</h1>
    <table border="1" cellpadding="2" cellspacing="0">
        <tr>
            <td>ID</td>
            <td>Username</td>
            <td>Nama</td>
            <td>ID Level Pengguna</td>
        </tr>
        <tr>
            <td>{{ $data->user_id }}</td>
       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

3. Simpan kode program Langkah 1 dan 2. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Setelah kode dijalankan, Laravel akan mencari atau membuat pengguna dengan username "manager". Data tersebut kemudian ditampilkan di halaman dalam bentuk tabel. Jika `level_id` tidak ada, kolomnya bisa kosong atau menimbulkan error.


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

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

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

5. Simpan kode program Langkah 4. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel `m_user` serta beri penjelasan dalam laporan. Error SQLSTATE[HY000]: General error: 1364 Field 'password' doesn't have a default value terjadi karena field password di tabel m_user diset sebagai NOT NULL tetapi tidak diberi nilai saat melakukan insert.



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

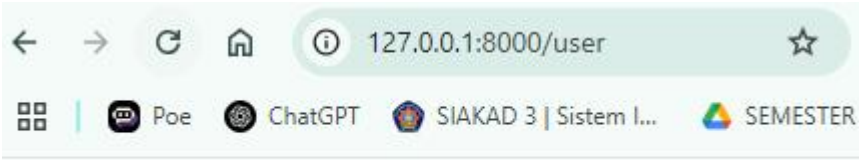
```php
class UserController extends Controller
{
    public function index()
    {
        $user = UserModel::firstOrNew(
            [
                'username' => 'manager',
                'nama' => 'Manager',
            ],
        );

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

7. Simpan kode program Langkah 6. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Kode di atas adalah controller Laravel UserController` yang berisi metode `index()` untuk mencari pengguna dengan username "manager" menggunakan `firstOrNew()`. Jika data tidak ditemukan, Laravel akan membuat instance baru tanpa menyimpannya ke database. Namun, terdapat kesalahan sintaksis pada koma setelah array, yang harus dihapus agar kode berjalan. Selain itu, jika atribut seperti `user_id` atau `level_id` diperlukan di tampilan, datanya mungkin tidak tersedia karena `firstOrNew()` hanya mengisi atribut yang diberikan. Akhirnya, metode ini mengembalikan tampilan `user.blade.php` dengan data pengguna.


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

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

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

9. Simpan kode program Langkah 8. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel *m_user* serta beri penjelasan dalam laporan. Kode di atas adalah metode `index()` dalam `UserController` yang mencari atau membuat instance pengguna baru dengan username "manager33" menggunakan `firstOrNew()`. Jika data tidak ditemukan, Laravel akan membuat objek baru tanpa menyimpannya ke database. Password dienkripsi dengan `Hash::make()`, dan `level_id` diatur ke 2. Namun, karena `firstOrNew()` tidak menyimpan data ke database secara otomatis, jika pengguna belum ada, perubahan ini hanya terjadi di memori dan tidak tersimpan permanen. Akhirnya, data dikirim ke tampilan `user.blade.php` untuk ditampilkan.



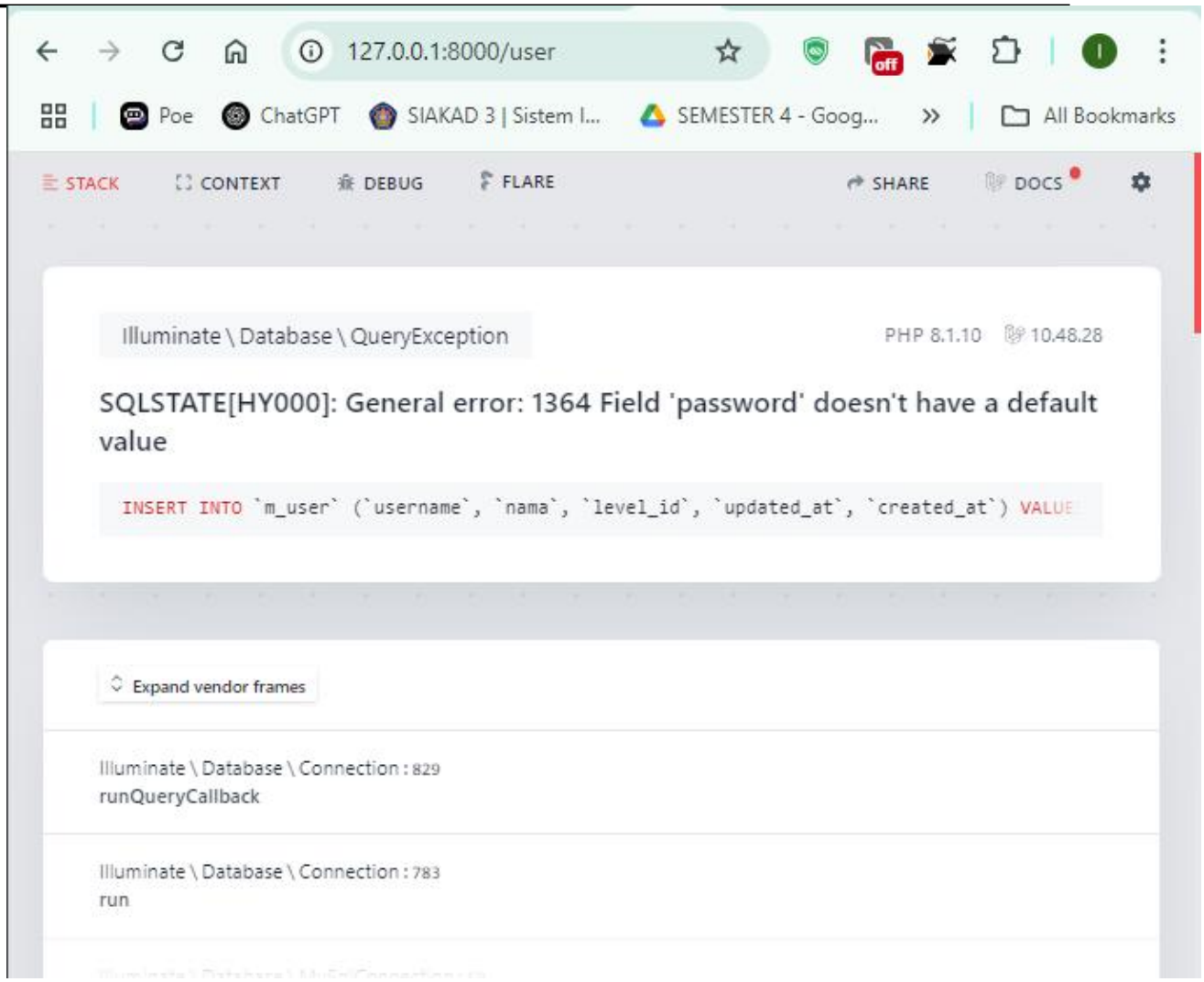

# Data User

| ID | Username | Nama | ID Level Pengguna |
|---|---|---|---|
| | manager33 | Manager Tiga Tiga | 2 |

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

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

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

11. Simpan kode program Langkah 9. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan cek juga pada *phpMyAdmin* pada tabel `m_user` serta beri penjelasan dalam laporan. Error "SQLSTATE[HY000]: General error: 1364 Field 'password' doesn't have a default value" terjadi karena metode firstOrNew() hanya mengisi objek model di memori tetapi tidak menyimpan data secara otomatis ke database. Saat Anda memanggil $user->save();, Laravel mencoba menyimpan objek tersebut, tetapi hanya kolom pencarian yang disertakan dalam firstOrNew(). Karena firstOrNew() hanya mencari berdasarkan username, jika pengguna sudah ada, atribut lainnya seperti password dan level_id tidak diperbarui. Tetapi jika pengguna belum ada, objek dibuat di memori tanpa menyertakan nilai default yang dibutuhkan oleh database.




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

## **Praktikum 2.5** – *Attribute Changes*

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

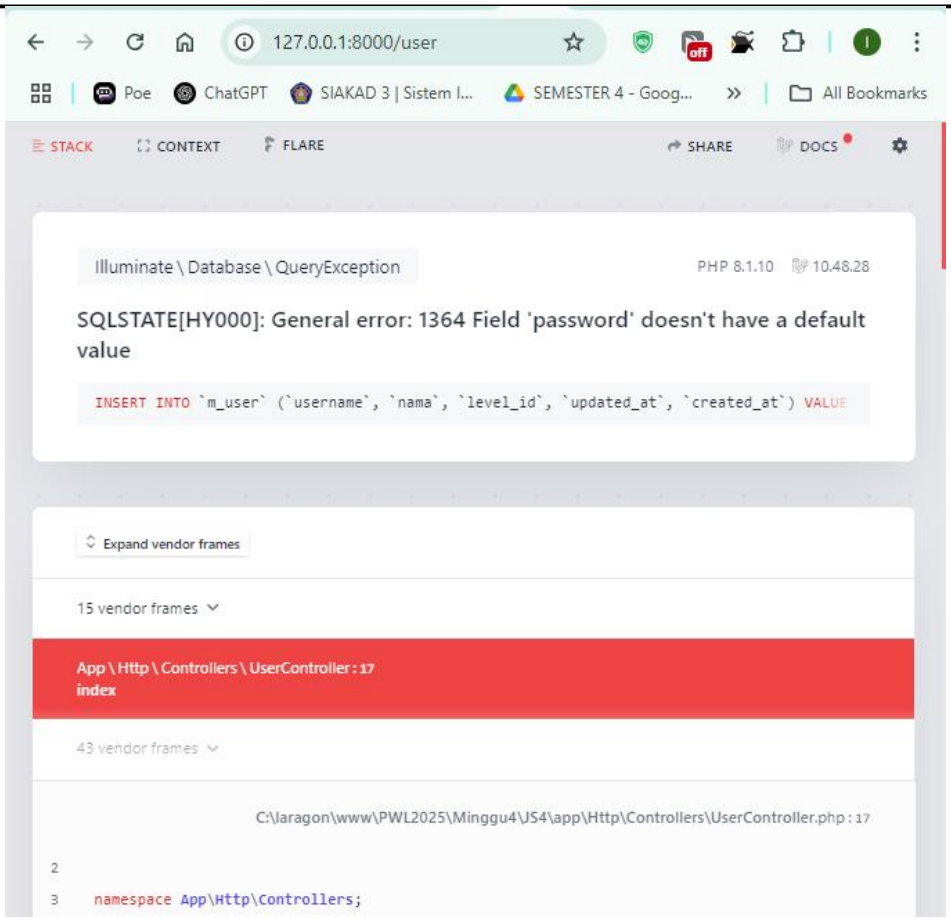
Metode `isDirty` menentukan apakah ada atribut model yang telah diubah sejak model diambil. Anda dapat meneruskan nama atribut tertentu atau serangkaian atribut ke metode `isDirty` untuk menentukan apakah ada atribut yang "kotor". Metode ini `isClean` akan menentukan apakah suatu atribut tetap tidak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

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

$user->username = 'manager45';

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

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

$user->save();

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

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

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

        $user->username = 'manager56';

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

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

        $user->save();

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

2. Simpan kode program Langkah 1. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Error "Field 'password' doesn't have a default value" terjadi karena kolom `password` di database disetel sebagai `NOT NULL`, tetapi tidak diberi nilai saat menggunakan `UserModel::create()`. Pastikan `password` selalu diisi dengan `Hash::make('12345')` dan tambahkan `password` ke dalam `$fillable` di model agar bisa digunakan dalam mass assignment.



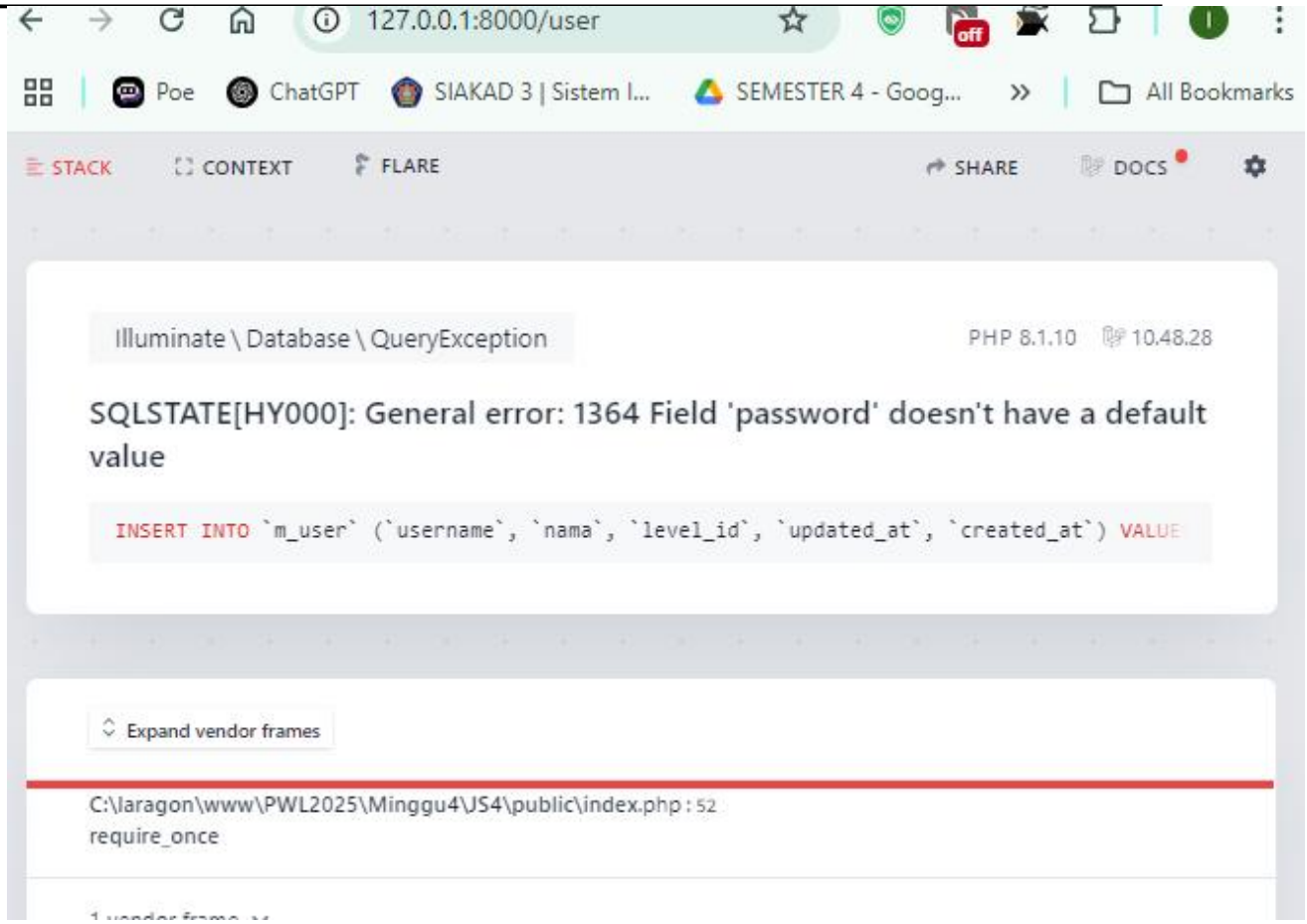

Metode ini `wasChanged` menentukan apakah ada atribut yang diubah saat model terakhir disimpan dalam siklus permintaan saat ini. Jika diperlukan, Anda dapat memberikan nama atribut untuk melihat apakah atribut tertentu telah diubah:

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

        $user->username = 'manager12';

        $user->save();

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

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

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

        $user->username = 'manager12';

        $user->save();

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

4. Simpan kode program Langkah 3. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Error ini terjadi karena kolom password di database disetel sebagai NOT NULL, tetapi tidak memiliki nilai default. Saat menggunakan UserModel::create(), pastikan password diisi dengan Hash::make('12345') dan tambahkan ke dalam $fillable di model agar bisa digunakan dalam mass assignment.




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

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

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

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

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

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

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

3. Simpan kode program Langkah 1 dan 2. Kemudian jalankan pada *browser* dan amati apa yang terjadi dan beri penjelasan dalam laporan. Ketika `UserController.php` dijalankan, metode `index()` mengambil semua data dari tabel `users` menggunakan `UserModel::all()` dan mengirimkannya ke tampilan `user.blade.php` dalam variabel `data`. Namun, jika terjadi error, kemungkinan penyebabnya adalah model tidak ditemukan, tabel tidak sesuai, atau kolom yang diakses salah. Pastikan `UserModel` sudah terhubung dengan tabel yang benar dan kolom yang dipanggil di Blade sesuai dengan struktur database.


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

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

        {{ csrf_field() }}

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

    </form>
</body>
```

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

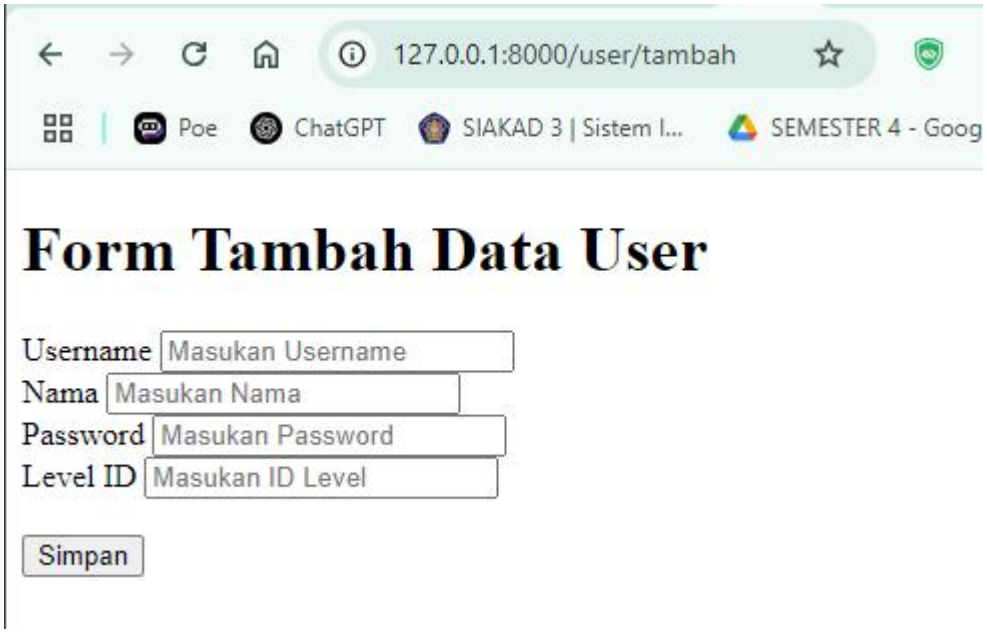
```
Route::get('/user/tambah', [UserController::class, 'tambah']);
```

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

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

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

7. Simpan kode program Langkah 4 s/d 6. Kemudian jalankan pada *browser* dan klik link "**+ Tambah User**" amati apa yang terjadi dan beri penjelasan dalam laporan.. Kode yang telah ditambahkan memungkinkan pengguna mengakses halaman tambah data user melalui URL `/user/tambah`. Route di `web.php` menghubungkan URL tersebut dengan metode `tambah()` di `UserController.php`, yang menampilkan tampilan `user_tambah.blade.php`. Namun, karena belum ada metode untuk menyimpan data ke database, saat pengguna mengisi formulir dan mengklik tombol "Simpan", permintaan POST ke `/user/tambah_simpan` akan menghasilkan error 404 atau Method Not

Allowed.



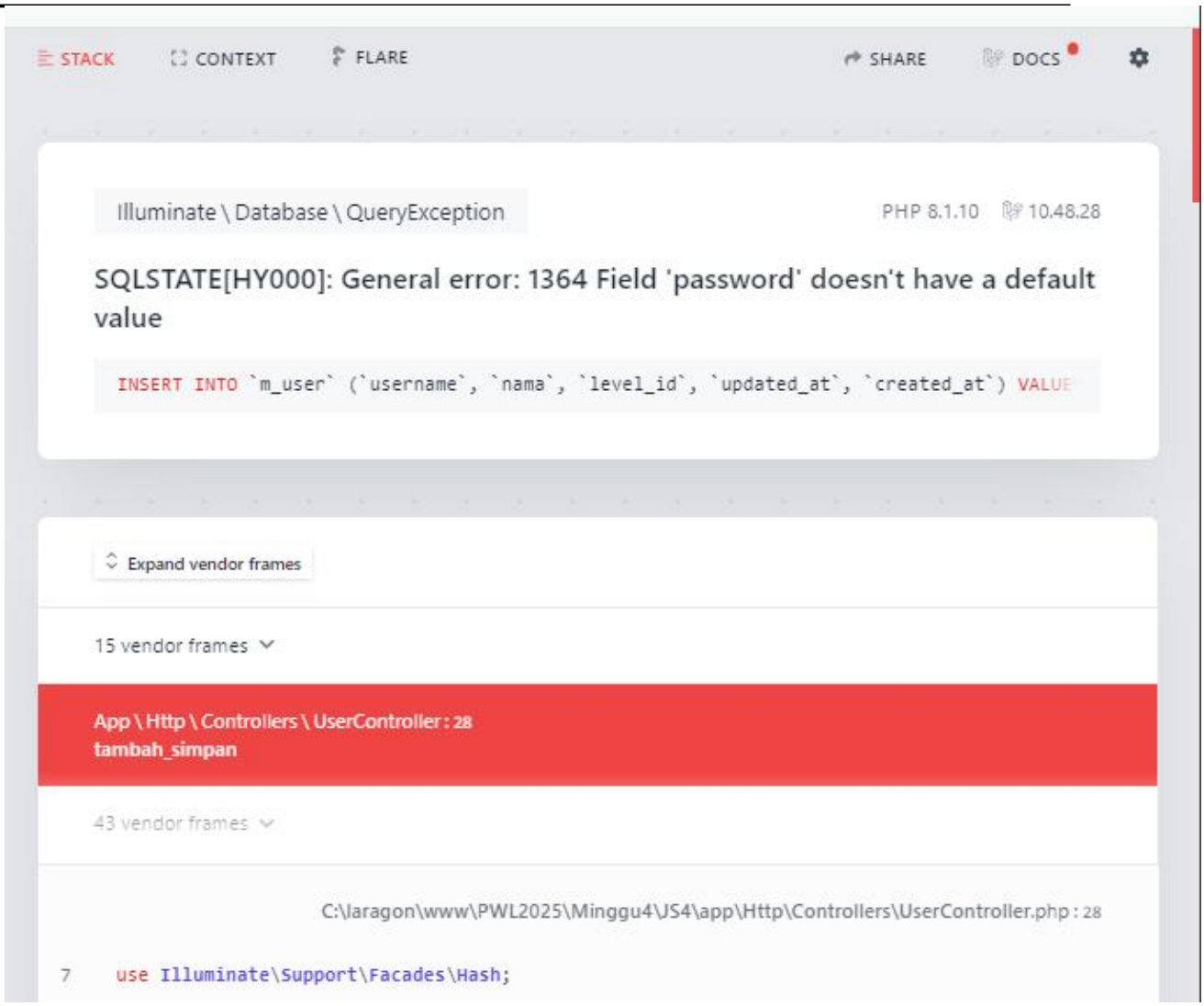

8. Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

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

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

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

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

10. Simpan kode program Langkah 8 dan 9. Kemudian jalankan link `localhost:8000/user/tambah` atau `localhost/PWL_POS/public/user/tambah` pada *browser* dan input formnya dan simpan, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan. Error "SQLSTATE[HY000]: General error: 1364 Field 'password' doesn't have a default value" terjadi karena kolom `password` di database tidak memiliki nilai default, sehingga Laravel gagal menyimpan data. Penyebabnya bisa karena kesalahan pada `Hash::make('$request->password')`, yang seharusnya tanpa tanda kutip (`Hash::make($request->password)`). Selain itu, pastikan model `UserModel` memiliki `protected $fillable` untuk `password`, form mengirimkan input password dengan benar, dan kolom `password` di database tidak diset sebagai `NOT NULL` tanpa nilai yang diberikan.




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

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

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

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

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

    </form>
</body>
```

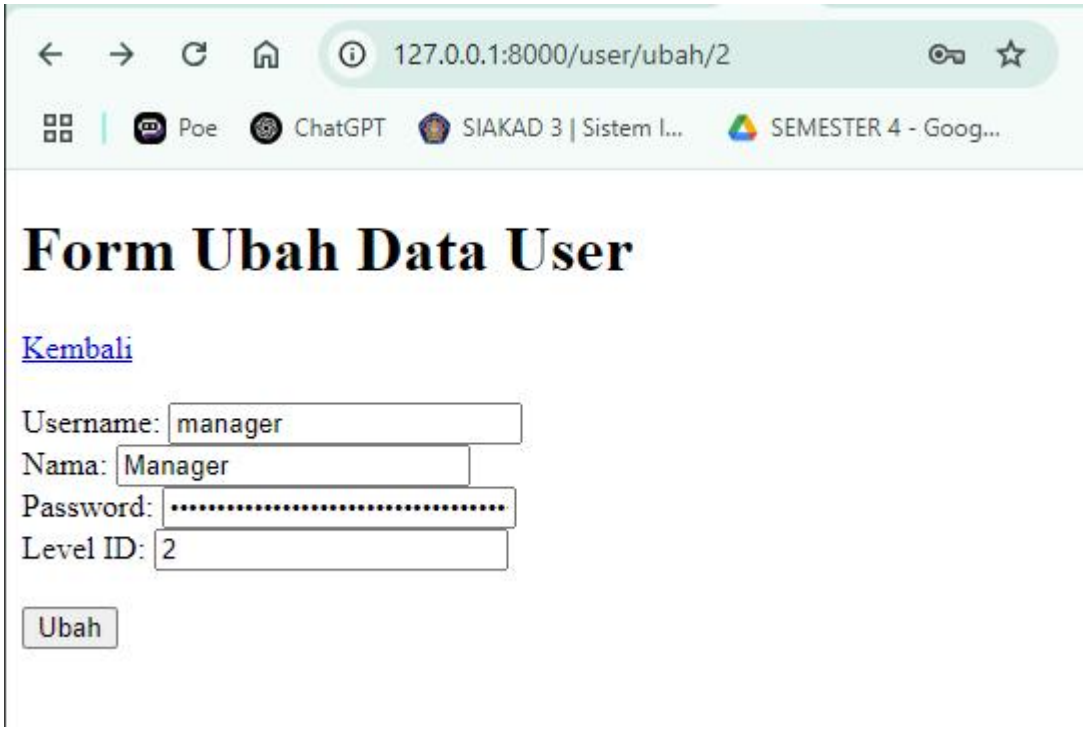
12. Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

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

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

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

14. Simpan kode program Langkah 11 sd 13. Kemudian jalankan pada *browser* dan klik link "**Ubah**" amati apa yang terjadi dan beri penjelasan dalam laporan. .Data pengguna dengan ID 2 telah diambil dari database dan ditampilkan dalam form. Username, nama, password dalam bentuk bintang (karena input type-nya password), serta Level ID telah terisi sesuai dengan data pengguna.



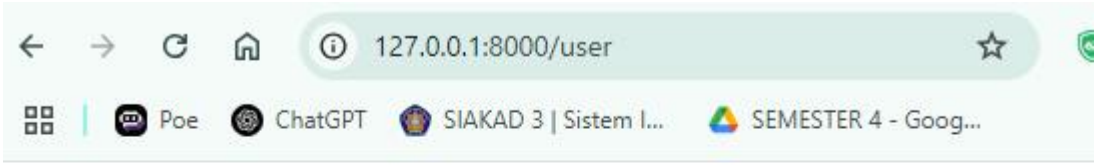

15. Tambahkan *script* pada *routes* dengan nama file `web.php`. Tambahkan seperti gambar di bawah ini

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

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

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

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

    $user->save();

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

17. Simpan kode program Langkah 15 dan 16. Kemudian jalankan link `localhost:8000/user/ubah/1` atau `localhost/PWL_POS/public/user/ubah/1` pada *browser* dan ubah input formnya dan klik tombol ubah, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan. Habis muncul dalam tampilan ubah, ketika klik ubah akan langsung terubah di data user dan langsung ditampilkan seperti dibawah ini.




# 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 |
| 11 | manager_dua | Manager 2 | 2 | Ubah \| Hapus |
| 12 | manager_tiga | Manager 3 | 2 | Ubah \| Hapus |

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

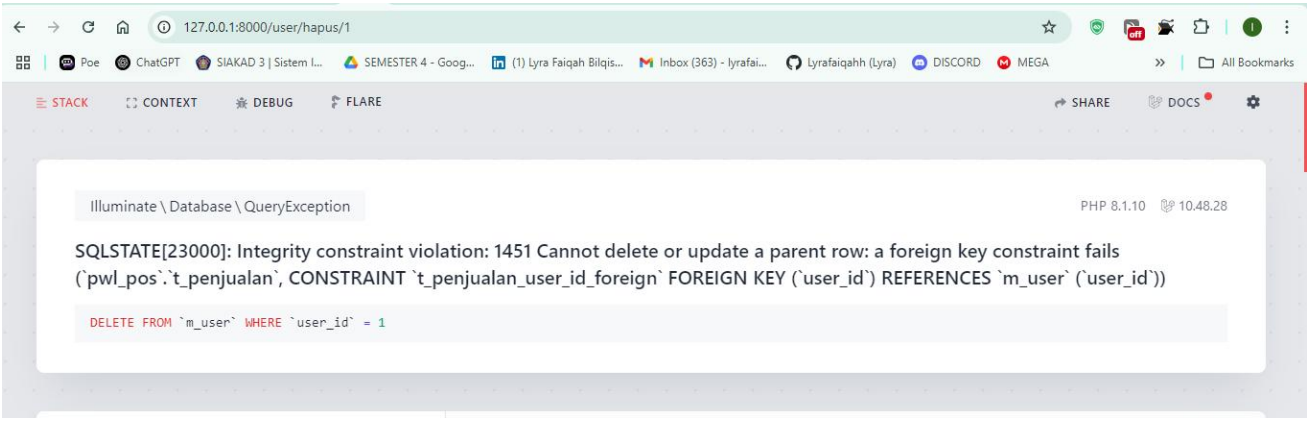
```php
Route::get('/user/hapus/{id}', [UserController::class, 'hapus']);
```

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

method ubah_simpan seperti gambar di bawah ini

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

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

20. Simpan kode program Langkah 18 dan 19. Kemudian jalankan pada *browser* dan klik tombol hapus, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan. Kesalahan SQLSTATE[23000]: Integrity constraint violation: 1451 terjadi karena data yang ingin dihapus dari tabel `UserModel` masih memiliki keterkaitan dengan data lain di tabel `t_penjualan` melalui foreign key constraint. Artinya, ada referensi ke entri pengguna di tabel `t_penjualan`, sehingga database mencegah penghapusan untuk menjaga integritas data. Untuk mengatasi masalah ini, Anda bisa menghapus terlebih dahulu data terkait di `t_penjualan`, mengubah foreign key constraint menjadi `ON DELETE CASCADE`




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

## **Praktikum 2.7** – *Relationships*

### *One to One*

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

```php
<?php

namespace App\Models;

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

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

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

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

```php
<?php

namespace App\Models;

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

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

## One to Many

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

```php
<?php

namespace App\Models;

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

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

## One to Many (Inverse) / Belongs To

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

```php
<?php

namespace App\Models;

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

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

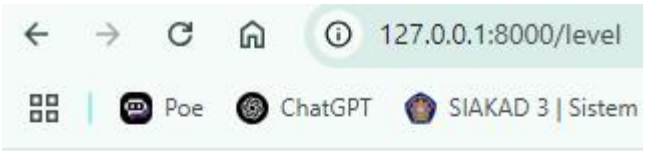
1. Buka file model pada `UserModel.php` dan tambahkan scritpnya menjadi seperti di bawah ini

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

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

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

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

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

3. Simpan kode program Langkah 2. Kemudian jalankan link pada *browser*, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan.. Karena dalam kode UserController terdapat dd($user), maka yang muncul di browser adalah dump data collection dari user beserta relasi levelnya.Outputnya berupa struktur data dalam bentuk array atau JSON yang menampilkan data user dan level yang terkait. Jika data sudah benar, maka akan terlihat setiap user memiliki atribut seperti user_id, username, nama, password, dan juga bagian level yang menampilkan data dari tabel level. Ini menandakan bahwa relasi belongsTo() antara UserModel dan LevelModel sudah berfungsi dengan baik, serta query yang menggunakan with('level')->get() berhasil mengambil data dengan relasi yang sesuai.




# Data Level Pengguna

| ID | Kode Level | Nama Level |
|---|---|---|
| 1 | ADM | Administrator |
| 2 | MNG | Manager |
| 3 | STF | Staff/Kasir |

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

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

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

```html
<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>
```

6. Simpan kode program Langkah 4 dan 5. Kemudian jalankan link pada *browser*, kemudian amati apa yang terjadi dan beri penjelasan dalam laporan. Bagian yang berjalan dengan baik adalah metode index() di UserController yang menggunakan with('level')->get();, yang memastikan setiap data pengguna juga mengambil informasi terkait dari tabel level melalui relasi belongsTo di model. Selain itu, tampilan user.blade.php berhasil menampilkan data level, karena properti seperti $d->level->kode_level telah diakses dengan benar dan tidak ada nilai null yang menyebabkan error.


# Data Level Pengguna

| ID | Kode Level | Nama Level |
|---|---|---|
| 1 | ADM | Administrator |
| 2 | MNG | Manager |
| 3 | STF | Staff/Kasir |

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

Sudah

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