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

Nama : Adinda Ivanka Maysanda Putri
Kelas : SIB 2B
NIM : 2341760058

JOBSHEET 03

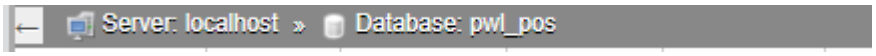
MIGRATION, SEEDER, DB FAÇADE, QUERY BUILDER, dan ELOQUENT ORM

Sebelumnya kita sudah membahas mengenai *Routing*, *Controller*, dan *View* 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.

Untuk itu, kita memerlukan teknik untuk merancang/membuat table basis data sebelum membuat aplikasi. Laravel memiliki fitur dalam pengelolaan basis data seperti, migration, seeder, model, dll.

Sebelum kita masuk materi, kita buat dulu project baru yang akan kita gunakan untuk membangun aplikasi sederhana dengan topik *Point of Sales (PoS)*, sesuai dengan `Studi Kasus PWL.pdf`. Jadi kita bikin project Laravel 10 dengan nama **PWL_POS.**

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

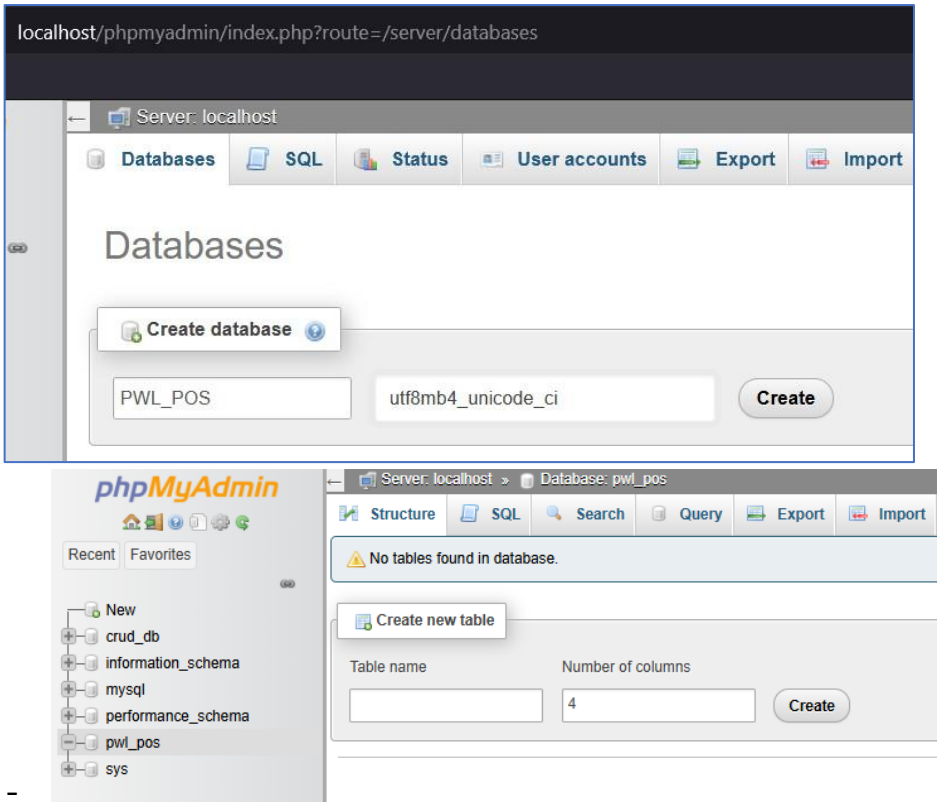


A. PENGATURAN DATABASE

Database atau basis data menjadi komponen penting dalam membangun sistem. Hal ini dikarenakan database menjadi tempat untuk menyimpan data-data transaksi yang ada pada sistem. Koneksi ke database perlu kita atur agar sesuai dengan database yang kita gunakan.

Praktikum 1 - pengaturan database:

1. Buka aplikasi phpMyAdmin, dan buat database baru dengan nama PWL_POS

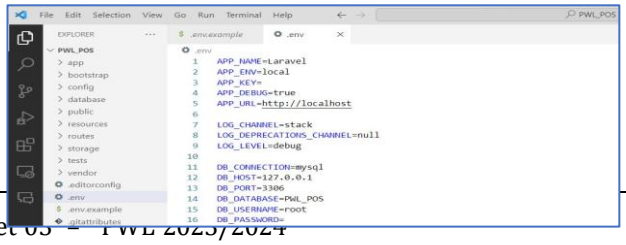


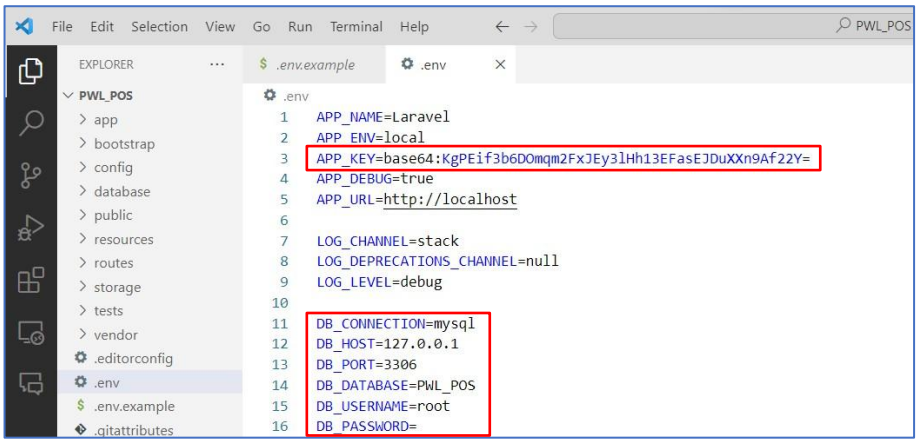
-

2. Buka aplikasi VSCode dan buka folder project PWL_POSyang sudah kita buat

-

3. Copy file .env.examplemenjadi .env
4. Buka file .env, dan pastikan konfigurasi **APP_KEY**bernilai. Jika belum bernilai silahkan kalian *generate* menggunakan php artisan.
5. Edit file .envdan sesuaikan dengan database yang telah dibuat





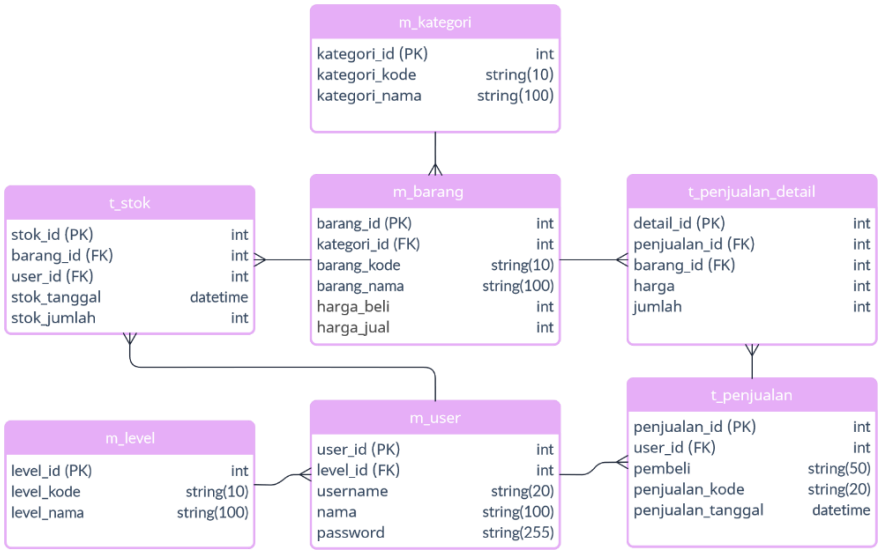
6. Laporkan hasil Praktikum-1 ini dan *commit* perubahan pada *git*.

B. MIGRATION

Migration pada Laravel merupakan sebuah fitur yang dapat membantu kita mengelola database secara efisien dengan menggunakan kode program. Migration membantu kita dalam membuat (*create*), mengubah (*edit*), dan menghapus (*delete*) struktur tabel dan kolom pada database yang sudah kita buat dengan cepat dan mudah. Dengan Migration, kita juga dapat melakukan perubahan pada struktur database tanpa harus menghapus data yang ada.

Salah satu keunggulan menggunakan migration adalah mempermudah proses instalasi aplikasi kita, Ketika aplikasi yang kita buat akan diimplementasikan di server/komputer lain.

Sesuai dengan topik pembelajaran kita untuk membangun sistem *Point of Sales (PoS)* sederhana, maka kita perlu membuat migration sesuai desain database yang sudah didefinisikan pada file `Studi Kasus PWL.pdf`



Dalam membuat file migration di Laravel, yang perlu kita perhatikan adalah struktur table yang ingin kita buat.

TIPS *MIGRATION*

Buatlah file migration untuk table yang tidak memiliki relasi (table yang tidak ada *foreign key*) dulu, dan dilanjutkan dengan membuat file migrasi yang memiliki relasi yang sedikit, dan dilanjut ke file migrasi dengan table yang memiliki relasi yang banyak.

Dari tips di atas, kita dapat melakukan cek untuk desain database yang sudah ada dengan mengetahui jumlah *foreign key* yang ada. Dan kita bisa menentukan table mana yang akan kita

buat migrasinya terlebih dahulu.

No Urut	Nama Tabel	Jumlah FK
1	m_level	0
2	m_kategori	0
3	m_user	1
4	m_barang	1
5	t_penjualan	1
6	t_stok	2
7	t_penjualan_detail	2

INFO

Secara default Laravel sudah ada table `users` untuk menyimpan data pengguna, tapi pada praktikum ini, kita gunakan table sesuai dari file `Studi Kasus PWL.pdf` yaitu `m_user`.

Pembuatan file migrasi bisa menggunakan 2 cara, yaitu

a. Menggunakan artisan untuk membuat *file migration*

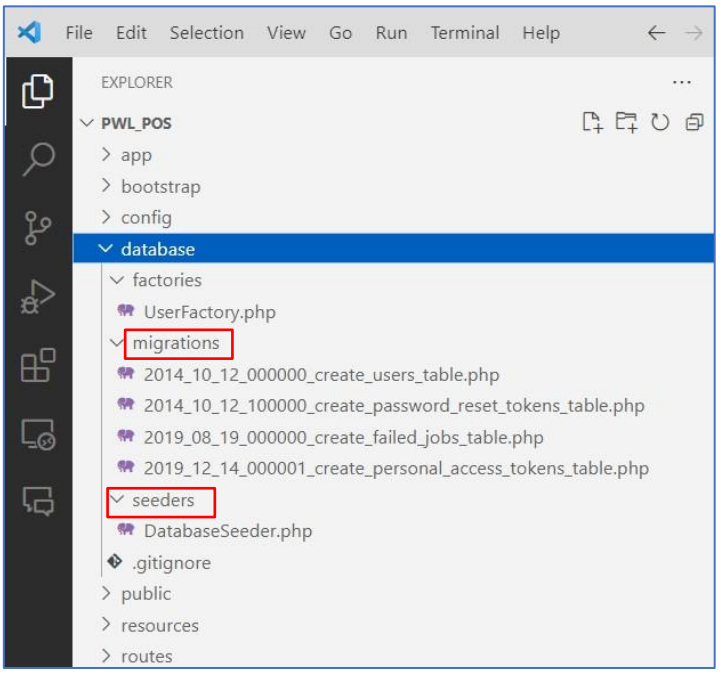
```
php artisan make:migration <nama-file-tabel> --create=<nama-tabel>
```

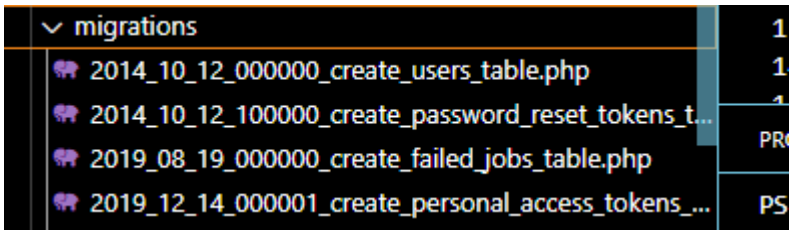
b. Menggunakan artisanuntuk membuat *file model* + *file migration*

```
php artisan make:model <nama-model> -m
```

Perintah **-m**di atas adalah *shorthand* untuk opsi membuat file migrasi berdasarkan model yang dibuat.

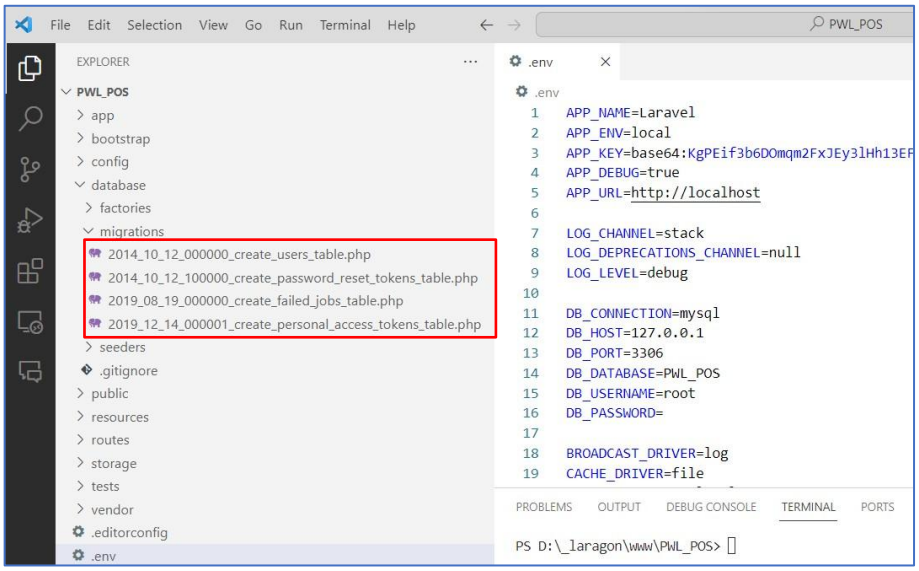
Pada Laravel, file-file *migration* ataupun *seeder* berada pada folder **PWL_POS/database**


File Edit Selection View Go Run Terminal Help
EXPLORER
PWL_POS
app
bootstrap
config
database
factories
UserFactory.php
migrations
2014_10_12_000000_create_users_table.php
2014_10_12_100000_create_password_reset_tokens_table.php
2019_08_19_000000_create_failed_jobs_table.php
2019_12_14_000001_create_personal_access_tokens_table.php
seeders
DatabaseSeeder.php
.gitignore
public
resources
routes


migrations
2014_10_12_000000_create_users_table.php
2014_10_12_100000_create_password_reset_tokens_t...
2019_08_19_000000_create_failed_jobs_table.php
2019_12_14_000001_create_personal_access_tokens_...

Praktikum 2.1 - Pembuatan file migrasi tanpa relasi

1. Buka *terminal* VSCode kalian, untuk yang di kotak merah adalah default dari laravel



2. Kita abaikan dulu yang di kotak merah (jangan di hapus)
3. Kita buat file migrasi untuk table m_level dengan perintah

```
php artisan make:migration create_m_level_table --create=m_level
```

4. Kita perhatikan bagian yang di kotak merah, bagian tersebut yang akan kita modifikasi sesuai desain database yang sudah ada

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_level', function (Blueprint $table) {
            $table->id('level_id');
            $table->string('level_kode', 10)->unique();
            $table->string('level_nama', 100);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('m_level');
    }
};
```

```
PS C:\laragon\www\Pemrograman-Web-Lanjut\Minggu 3 (PWL_POS)> php artisan migrate

  INFO  Preparing database.

  Creating migration table ..... 90ms DONE

  INFO  Running migrations.

  2014_10_12_000000_create_users_table ..... 103ms DONE
  2014_10_12_100000_create_password_reset_tokens_table ..... 52ms DONE
  2019_08_19_000000_create_failed_jobs_table ..... 81ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ..... 127ms DONE
  2025_02_25_070845_create_m_level_table ..... 98ms DONE
  2025_02_25_072305_create_m_kategori_table ..... 87ms DONE
  2025_02_25_072853_create_m_user_table ..... 181ms DONE
  2025_03_02_044610_create_t_penjualan_table ..... 172ms DONE
  2025_03_02_050145_create_m_barang_table ..... 167ms DONE
  2025_03_02_050439_create_t_stok_table ..... 292ms DONE
  2025_03_02_051031_create_t_penjualan_detail_table ..... 226ms DONE

PS C:\laragon\www\Pemrograman-Web-Lanjut\Minggu 3 (PWL_POS)>
```

INFO

Dalam fitur migration Laravel, terdapat berbagai macam function untuk membuat kolom di table database. Silahkan cek disini

https://laravel.com/docs/10.x/migrations#available-column-types

5. Simpan kode pada tahapan 4 tersebut, kemudian jalankan perintah ini pada terminal VSCode untuk melakukan migrasi

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\_laragon\www\PWL_POS> php artisan migrate

  INFO  Preparing database.

  Creating migration table ..... 12ms DONE

  INFO  Running migrations.

  2014_10_12_000000_create_users_table ..... 16ms DONE
  2014_10_12_100000_create_password_reset_tokens_table ..... 6ms DONE
  2019_08_19_000000_create_failed_jobs_table ..... 42ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ..... 15ms DONE
  2024_02_25_133526_create_m_level_table ..... 13ms DONE

PS D:\_laragon\www\PWL_POS>
```

6. Kemudian kita cek di phpMyAdmin apakah table sudah ter-generate atau belum

7. Ok, table sudah dibuat di database
8. Buat table *database* dengan *migration* untuk table **m_kategori** yang sama-sama tidak memiliki *foreign key*
9. Laporkan hasil Praktikum-2.1 ini dan *commit* perubahan pada *git*.

Praktikum 2.2 - Pembuatan file migrasi dengan relasi

1. Buka *terminal* VSCode kalian, dan buat file migrasi untuk table **m_user**

```
php artisan make:migration create_m_user_table --table=m_user
```

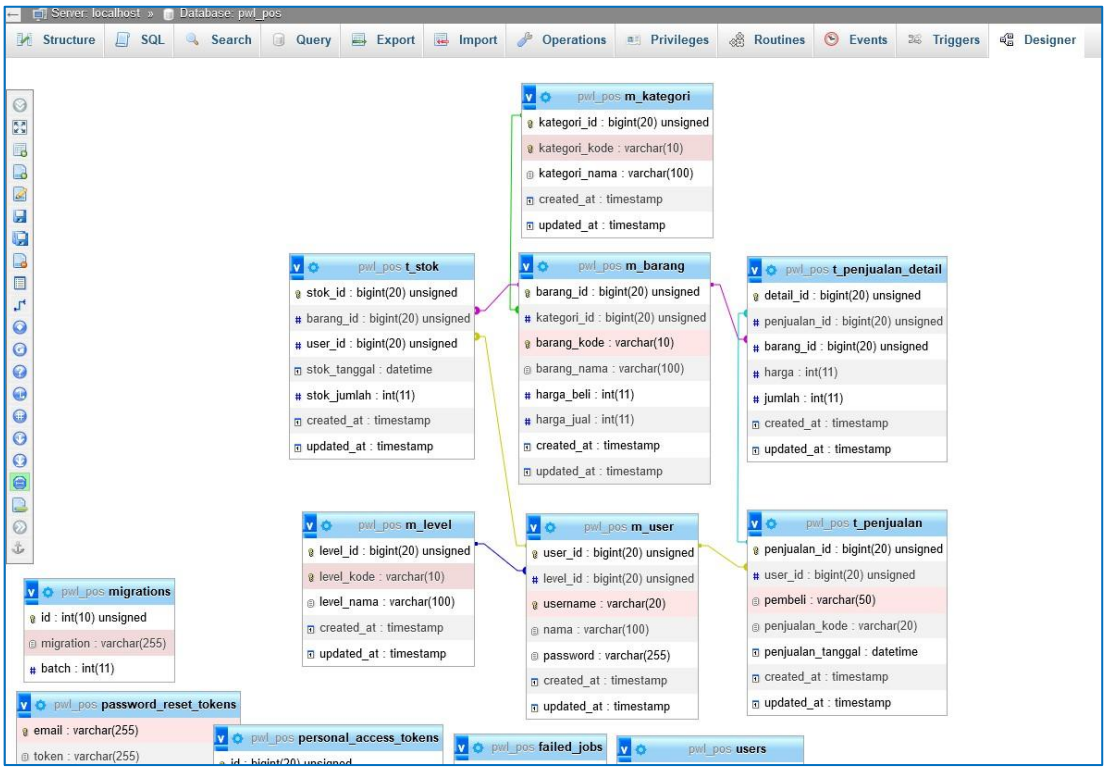
2. Buka file migrasi untuk table **m_user**, dan modifikasi seperti berikut

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_user', function (Blueprint $table) {
            $table->id('user_id');
            $table->unsignedBigInteger('level_id')->index(); // indexing untuk ForeignKey
            $table->string('username', 20)->unique(); // unique untuk memastikan tidak ada username yang sama
            $table->string('nama', 100);
            $table->string('password');
            $table->timestamps();

            // Mendefinisikan Foreign Key pada kolom level_id mengacu pada kolom level_id di tabel m_level
            $table->foreign('level_id')->references('level_id')->on('m_level');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('m_user');
    }
};
```

3. Simpan kode program Langkah 2, dan jalankan perintah **php artisan migrate**. Amati apa yang terjadi pada database.
4. Buat table *database* dengan *migration* untuk table-tabel yang memiliki *foreign key*

m_barang
t_penjualan
t_stok
t_penjualan_detail

5. Jika semua file migrasi sudah di buat dan dijalankan maka bisa kita lihat tampilan *designer*pada **phpMyAdmin**seperti berikut



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

C. SEEDER

Seeder merupakan sebuah fitur yang memungkinkan kita untuk mengisi database kita dengan data awal atau data *dummy* yang telah ditentukan. Seeder memungkinkan kita untuk membuat data awal yang sama untuk setiap penggunaan dalam pembangunan aplikasi. Umumnya, data yang sering dibuat *seeder* adalah data penggunna karena data tersebut akan digunakan saat aplikasi pertama kali di jalankan dan membutuhkan aksi *login*.

1. Perintah umum dalam **membuat *file seeder*** adalah seperti berikut

```
php artisan make:seeder <nama-class-seeder>
```

   Perintah tersebut akan men-generate file seeder pada folder **PWL_POS/database/seeders**

2. Dan perintah untuk **menjalankan *file seeder*** seperti berikut

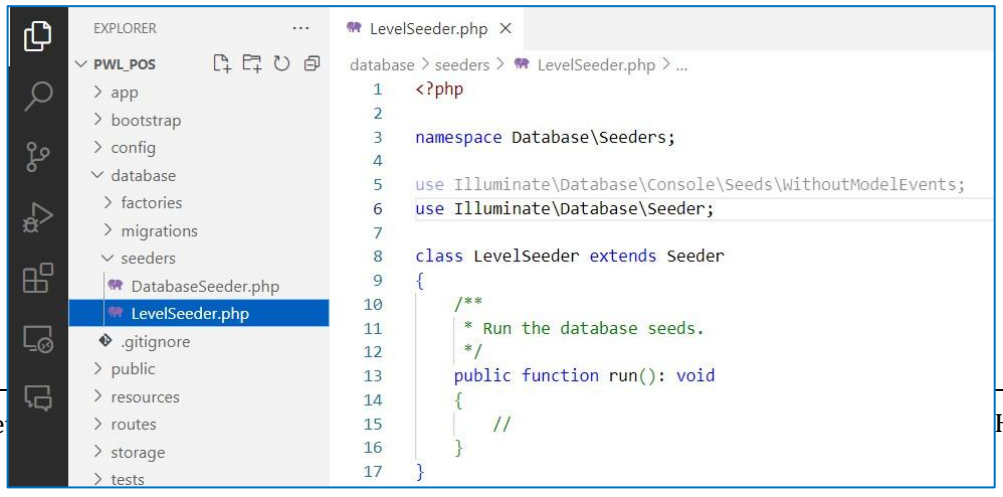
```
php artisan db:seed --class=<nama-class-seeder>
```

Dalam proses pengembangan suatu aplikasi, seringkali kita membutuhkan data awal tiruan atau *dummy* data untuk memudahkan pengujian dan pengembangan aplikasi kita. Sehingga fitur *seeder* bisa kita pakai dalam membuat sebuah aplikasi web.

Praktikum 3 – Membuat file *seeder*

1. Kita akan membuat file seeder untuk table m_level dengan mengetikkan perintah

```
php artisan make:seeder LevelSeeder
```

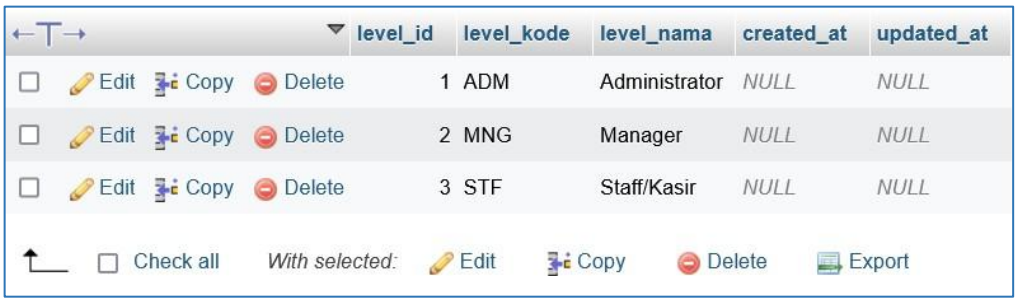


2. Selanjutnya, untuk memasukkan data awal, kita modifikasi file tersebut di dalam function run()

3. Selanjutnya, kita jalankan file *seeder* untuk table m_levelpada terminal

```
php artisan db:seed --class=LevelSeeder
```

4. Ketika *seeder* berhasil dijalankan maka akan tampil data pada table m_level



5. Sekarang kita buat file *seeder* untuk table m_useryang me-*refer* ke table m_level

```
php artisan make:seeder UserSeeder
```

6. Modifikasi file class**UserSeeder**seperti berikut

```php
class UserSeeder extends Seeder
{
    public function run(): void
    {
        $data = [
            [
                'user_id' => 1,
                'level_id' => 1,
                'username' => 'admin',
                'nama' => 'Administrator',
                'password' => Hash::make('12345'), // class untuk mengenkripsi/hash password
            ],
            [
                'user_id' => 2,
                'level_id' => 2,
                'username' => 'manager',
                'nama' => 'Manager',
                'password' => Hash::make('12345'),
            ],
            [
                'user_id' => 3,
                'level_id' => 3,
                'username' => 'staff',
                'nama' => 'Staff/Kasir',
                'password' => Hash::make('12345'),
            ],
        ];
        DB::table('m_user')->insert($data);
    }
}
```

7. Jalankan perintah untuk mengeksekusi class UserSeeder

```
php artisan db:seed --class=UserSeeder
```

8. Perhatikan hasil seeder pada table m_user

	user_id	level_id	username	nama	password
Edit Copy Delete	1	1	admin	Administrator	$2y$12$Tevu4dDO1CUAQpeM6H.Vp.LySwhY.4oAKU7FzwS6fXV...
Edit Copy Delete	2	2	manager	Manager	$2y$12$Ajfns20/FdPTeUgghz31muEhlFaruLxkh5wvZ9NGRpu...
Edit Copy Delete	3	3	staff	Staff/Kasir	$2y$12$Gi23TqGcIW5pYeR0VL4o5OxPwb3Osk99VMy/BHnbJ9W...

Check all *With selected:* Edit Copy Delete Export

9. Ok, data seeder berhasil di masukkan ke database.
10. Sekarang coba kalian masukkan data *seeder* untuk table yang lain, dengan ketentuan seperti berikut

No	Nama Tabel	Jumlah Data	Keterangan
1	m_kategori	5	5 kategori barang
2	m_barang	10	10 barang yang berbeda
3	t_stok	10	Stok untuk 10 barang
4	t_penjualan	10	10 transaksi penjualan
5	t_penjualan_detail	30	3 barang untuk setiap transaksi

			penjualan

11. Jika sudah, laporkan hasil Praktikum-3 ini dan *commit* perubahan pada *git*

D. DB FACADE

DB Façade merupakan fitur dari Laravel yang digunakan untuk melakukan *query* secara langsung dengan mengetikkan perinta SQL secara utuh (*raw query*). Disebut *raw query* (query mentah) karena penulisan query pada DB Façade langsung ditulis sebagaimana yang biasa dituliskan pada database, seperti "select * from m_user" atau "insert into m_user…" atau "update m_user set … Where …"

Raw query adalah cara paling dasar dan tradisional yang ada di Laravel. Raw query terasa familiar karena biasa kita pakai ketika melakukan query langsung ke database.

INFO

Dokumentasi penggunaan DB Façade bisa dicek di laman ini

https://laravel.com/docs/10.x/database#running-queries

Terdapat banyak method yang bisa digunakan pada DB Façade ini. Akan tetapi yang kita pelajari cukup 4 (empat) method yang umum dipakai, yaitu

a. DB::select()

   Method ini digunakan untuk mengambil data dari database. Method ini **mengembalikan *(return)*** data hasil *query*. Contoh

```
DB::select('select * from m_user'); //Query semua data pada tabel m_user
```

```
DB::select('select * from m_user where level_id = ?', [1]); //Query tabel m_user dengan level_id = 1
```

```
DB::select('select * from m_user where level_id = ? and username = ?', [1, 'admin']);
```

b. DB::insert()

   Method ini digunakan untuk memasukkan data pada table database. Method ini **tidak memiliki nilai pengembalian *(no return)***. Contoh

```
DB::insert('insert into m_level(level_kode, level_nama) values(?,?)', ['CUS', 'Pelanggan']);
```

c. DB::update()

   Method ini digunakan saat menjalankan *raw query* untuk meng-update data pada database. Method ini **memiliki nilai pengembalian *(return)*** berupa jumlah baris data yang ter-*update*. Contoh

```
DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
```

d. DB::delete()

Method ini digunakan saat menjalankan *raw query* untuk menghapus data dari table. Method ini **memiliki nilai pengembalian *(return)*** berupa jumlah baris data yang telah dihapus. Contoh

```
DB::delete('delete from m_level where level_kode = ?', ['CUS']);
```

Ok, sekarang mari kita coba praktikkan menggunakan DB Façade pada project kita

Praktikum 4 – Implementasi DB Facade

1. Kita buat controller dahulu untuk mengelola data pada table m_level

```
php artisan make:controller LevelController
```

2. Kita modifikasi dulu untuk *routing*-nya, ada di PWL_POS/routes/web.php

LevelController.php | web.php

routes > web.php > ...

```
1  <?php
2
3  use App\Http\Controllers\LevelController;
4  use Illuminate\Support\Facades\Route;
5
6
7  Route::get('/', function () {
8      return view('welcome');
9  });
10
11 Route::get('/level', [LevelController::class, 'index']);
```

3. Selanjutnya, kita modifikasi file LevelControlleruntuk menambahkan 1 data ke table m_level

LevelController.php | web.php

app > Http > Controllers > LevelController.php > ...

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);

        return 'Insert data baru berhasil';
    }
}
```

4. Kita coba jalankan di browser dengan url localhost/PWL_POS/public/level dan amati apa yang terjadi pada table m_level di database, *screenshot* perubahan yang ada pada table m_level

	level_id	level_kode	level_nama	created_at	updated_at
Edit Copy Delete	1	ADM	Administrator	NULL	NULL
Edit Copy Delete	2	MNG	Manager	NULL	NULL
Edit Copy Delete	3	STF	Staff/Kasir	NULL	NULL
Edit Copy Delete	4	CUS	Pelanggan	2024-02-26 08:20:00	NULL

5. Selanjutnya, kita modifikasi lagi file LevelControlleruntuk meng-*update* data di table m_levelseperti berikut

LevelController.php | web.php

app > Http > Controllers > LevelController.php > ...

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';
    }
}
```

6. Kita coba jalankan di browser dengan url localhost/PWL_POS/public/level lagi dan amati apa yang terjadi pada table m_level di database, *screenshot* perubahan yang ada pada table m_level
7. Kita coba modifikasi lagi file LevelControlleruntuk melakukan proses hapus data

LevelController.php | web.php

app > Http > Controllers > LevelController.php > LevelController > index

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        // $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

        $row = DB::delete('delete from m_level where level_kode = ?', ['CUS']);
        return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';
    }
}
```

8. Method terakhir yang kita coba adalah untuk menampilkan data yang ada di table m_level. Kita modifikasi file LevelControllerseperti berikut

```php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        // $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

        // $row = DB::delete('delete from m_level where level_kode = ?', ['CUS']);
        // return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';

        $data = DB::select('select * from m_level');
        return view('level', ['data' => $data]);
    }
}
```

9. Coba kita perhatikan kode yang diberi tanda kotak merah, berhubung kode tersebut memanggil view(‘level’), maka kita buat file view pada VSCode di PWL_POS/resources/view/level.blade.php

LevelController.php | level.blade.php | web.php

resources > views > level.blade.php > ...

```blade
<!DOCTYPE html>
<html>
    <head>
        <title>Data Level Pengguna</title>
    </head>
    <body>
        <h1>Data Level Pengguna</h1>
        <table border="1" cellpadding="2" cellspacing="0">
            <tr>
                <th>ID</th>
                <th>Kode Level</th>
                <th>Nama Level</th>
            </tr>
            @foreach ($data as $d)
            <tr>
                <td>{{ $d->level_id }}</td>
                <td>{{ $d->level_kode }}</td>
                <td>{{ $d->level_nama }}</td>
            </tr>
            @endforeach
        </table>
    </body>
</html>
```

10. Silahkan dicoba pada browser dan amati apa yang terjadi
11. Laporkan hasil Praktikum-4 ini dan *commit* perubahan pada *git*.

E. QUERY BUILDER

Query builder adalah fitur yang disediakan Laravel untuk melakukan proses CRUD *(create, retrieve/read, update, delete)* pada database. Berbeda dengan *raw query* pada DB Facede yang mengharuskan kita menulis perintah SQL, pada *query builder* perintah SQL ini diakses menggunakan method. Jadi, kita tidak menulis perintah SQL secara langsung, melainkan cukup memanggil method-method yang ada di *query builder*.

Query builder membuat kode kita menjadi rapi dan lebih mudah dibaca. Selain itu *query builder* tidak terikat ke satu jenis database, jadi query builder bisa digunakan untuk mengakses berbagai jenis database seperti MySQL, MariaDB, PostgreSQL, SQL Server, dll. Jika suatu saat ingin beralih dari database MySQL ke PostgreSQL, tidak akan banyak kendala. Namun kelemahan dari *query builder* adalah kita harus mengetahui method-method apa saja yang ada di *query builder*.

INFO

Dokumentasi penggunaan Query Builder pada Laravel bisa dicek di laman ini

https://laravel.com/docs/10.x/queries

Ciri khas *query builder* Laravel adalah kita tentukan dahulu target table yang akan kita akses untuk operasi CRUD.

```
DB::table('<nama-tabel>'); // query builder untuk melakukan operasi CRUD pada tabel yang dituju
```

Perintah pertama yang dilakukan pada query builder adalah menentukan nama table yang akan dilakukan operasi CRUD. Kemudian baru disusul method yang ingin digunakan sesuai dengan peruntukannya. Contoh

a. Perintah untuk *insert* data dengan method insert()

```
DB::table('m_kategori')->insert(['kategori_kode' => 'SMP', 'kategori_nama' => 'Smartphone']);
```

Query yang dihasilkan dari kode di atas adalah

```
insert into m_kategori(kategori_kode, kategori_nama) values('SMP', 'Smartphone');
```

b. Perintah untuk *update* data dengan method where()dan update()

```
DB::table('m_kategori')->where('kategori_id', 1)->update(['kategori_nama' => 'Makanan Ringan']);
```

Query yang dihasilkan dari kode di atas adalah

```
update m_kategori set kategori_nama = ‘Makanan Ringan’ where kategori_id = 1;
```

c. Perintah untuk *delete* data dengan method where()dan delete()

```
DB::table('m_kategori')->where('kategori_id', 9) ->delete();
```

Query yang dihasilkan dari kode di atas adalah

```
delete from m_kategori where kategori_id = 9;
```

d. Perintah untuk ambil data

Method Query Builder	**Query yang dihasilkan**
DB::table('m_kategori')->get();	select * from m_kategori
DB::table('m_kategori') ->where('kategori_id', 1)->get();	select * from m_kategori where kategori_id = 1;
DB::table('m_kategori') ->select('kategori_kode') ->where('kategori_id', 1)->get();	select kategori_kode from m_kategori where kategori_id = 1;

Praktikum 5 – Implementasi *Query Builder*

1. Kita buat controller dahuku untuk mengelola data pada table m_kategori

```
php artisan make:controller KategoriController
```

2. Kita modifikasi dulu untuk routing-nya, ada di PWL_POS/routes/web.php

LevelController.php | KategoriController.php | level.blade.php | web.php

routes > web.php > ...

```
1  <?php
2
3  use App\Http\Controllers\KategoriController;
4  use App\Http\Controllers\LevelController;
5  use Illuminate\Support\Facades\Route;
6
7
8  Route::get('/', function () {
9      return view('welcome');
10 });
11
12 Route::get('/level', [LevelController::class, 'index']);
13 Route::get('/kategori', [KategoriController::class, 'index']);
```

3. Selanjutnya, kita modifikasi file KategoriController untuk menambahkan 1 data ke

table m_kategori

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class KategoriController extends Controller
{
    public function index()
    {
        $data = [
            'kategori_kode' => 'SNK',
            'kategori_nama' => 'Snack/Makanan Ringan',
            'created_at' => now()
        ];
        DB::table('m_kategori')->insert($data);
        return 'Insert data baru berhasil';
    }
}
```

4. Kita coba jalankan di browser dengan url localhost/PWL_POS/public/kategori dan amati apa yang terjadi pada table m_kategori di database, *screenshot* perubahan yang ada pada table m_kategori
5. Selanjutnya, kita modifikasi lagi file KategoriController untuk meng-*update* data di table m_kategori seperti berikut

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class KategoriController extends Controller
{
    public function index()
    {
        /* $data = [
            'kategori_kode' => 'SNK',
            'kategori_nama' => 'Snack/Makanan Ringan',
            'created_at' => now()
        ];
        DB::table('m_kategori')->insert($data);
        return 'Insert data baru berhasil'; */

        $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->update(['kategori_nama' => 'Camilan']);
        return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';
    }
}
```

6. Kita coba jalankan di browser dengan url localhost/PWL_POS/public/kategori lagi dan amati apa yang terjadi pada table m_kategori di database, *screenshot* perubahan yang ada pada table m_kategori
7. Kita coba modifikasi lagi file KategoriControlleruntuk melakukan proses hapus data

```php
    public function index()
    {
        /* $data = [
            'kategori_kode' => 'SNK',
            'kategori_nama' => 'Snack/Makanan Ringan',
            'created_at' => now()
        ];
        DB::table('m_kategori')->insert($data);
        return 'Insert data baru berhasil'; */

        // $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->update(['kategori_nama' => 'Camilan']);
        // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

        $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->delete();
        return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';
    }
```

8. Method terakhir yang kita coba adalah untuk menampilkan data yang ada di table m_kategori. Kita modifikasi file KategoriControllerseperti berikut

```php
    public function index()
    {
        /* $data = [
            'kategori_kode' => 'SNK',
            'kategori_nama' => 'Snack/Makanan Ringan',
            'created_at' => now()
        ];
        DB::table('m_kategori')->insert($data);
        return 'Insert data baru berhasil'; */

        // $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->update(['kategori_nama' => 'Camilan']);
        // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

        // $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->delete();
        // return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';

        $data = DB::table('m_kategori')->get();
        return view('kategori', ['data' => $data]);
    }
```

9. Coba kita perhatikan kode yang diberi tanda kotak merah, berhubung kode tersebut memanggil view(‘kategori’), maka kita buat file view pada VSCode di PWL_POS/resources/view/kategori.blade.php

resources > views > kategori.blade.php > html > body > table > tr > td

```blade
<!DOCTYPE html>
<html>
    <head>
        <title>Data Kategori Barang</title>
    </head>
    <body>
        <h1>Data Kategori Barang</h1>
        <table border="1" cellpadding="2" cellspacing="0">
            <tr>
                <th>ID</th>
                <th>Kode Kategori</th>
                <th>Nama Kategori</th>
            </tr>
            @foreach ($data as $d)
            <tr>
                <td>{{ $d->kategori_id }}</td>
                <td>{{ $d->kategori_kode }}</td>
                <td>{{ $d->kategori_nama }}</td>
            </tr>
            @endforeach
        </table>
    </body>
</html>
```

10. Silahkan dicoba pada browser dan amati apa yang terjadi.
11. Laporkan hasil Praktikum-5 ini dan *commit* perubahan pada *git*

F. ELOQUENT ORM

Eloquent ORM adalah fitur bawaan dari laravel. Eloquent ORM adalah cara pengaksesan database dimana setiap baris tabel dianggap sebagai sebuah object. Kata ORM sendiri merupakan singkatan dari ***Object-relational mapping***, yakni suatu teknik programming untuk mengkonversi data ke dalam bentuk object.

INFO

Eloquent ORM memerlukan Model untuk proses konversi data pada tabel menjadi object. Object inilah yang nantinya akan kita akses dari dalam controller. Oleh karena itu **membuat Model pada Laravel berarti menggunakan Eloquent ORM**. Silahkan cek disini

https://laravel.com/docs/10.x/eloquent

Perintah untuk membuat model adalah sebagai berikut

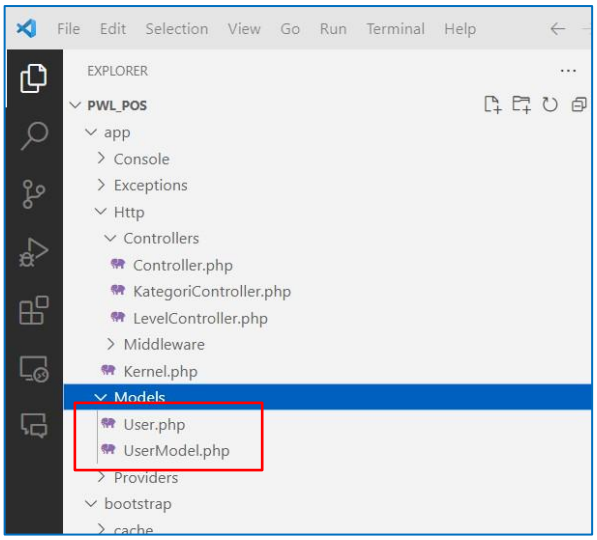
```
php artisan make:model <nama-model-CamelCase>
```

Untuk bisa melakukan operasi ***CRUD*** (*create, read/retrieve, update, delete*), kita harus membuat sebuah model sesuai dengan target tabel yang ingin digunakan. Jadi, **dalam 1 model, merepresentasikan 1 tabel database.**

Praktikum 6 – Implementasi Eloquent ORM

1. Kita buat file model untuk tabel m_userdengan mengetikkan perintah

```
php artisan make:model UserModel
```



2. Setelah berhasil generate model, terdapat 2 file pada folder model yaitu file User.php bawaan dari laravel dan file UserModel.php yang telah kita buat. Kali ini kita akan menggunakan file UserModel.php
3. Kita buka file UserModel.phpdan modifikasi seperti berikut

app > Models > UserModel.php > UserModel

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class UserModel extends Model
{
    use HasFactory;

    protected $table = 'm_user';        // Mendefinisikan nama tabel yang digunakan oleh model ini
    protected $primaryKey = 'user_id';  // Mendefinisikan primary key dari tabel yang digunakan
}

```

4. Kita modifikasi route web.phpuntuk mencoba routing ke controller UserController

routes > web.php > ...

```php
<?php

use App\Http\Controllers\KategoriController;
use App\Http\Controllers\LevelController;
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;


Route::get('/', function () {
    return view('welcome');
});

Route::get('/level', [LevelController::class, 'index']);
Route::get('/kategori', [KategoriController::class, 'index']);
Route::get('/user', [UserController::class, 'index']);
```

5. Sekarang, kita buat file controller UserControllerdan memodifikasinya seperti berikut

app > Http > Controllers > UserController.php > ...

```php
<?php

namespace App\Http\Controllers;

use App\Models\UserModel;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function index()
    {
        // coba akses model UserModel
        $user = UserModel::all(); // ambil semua data dari tabel m_user
        return view('user', ['data' => $user]);
    }
}
```

6. Kemudian kita buat view user.blade.php

resources > views > 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>
                <th>ID</th>
                <th>Username</th>
                <th>Nama</th>
                <th>ID Level Pengguna</th>
            </tr>
            @foreach ($data as $d)
            <tr>
                <td>{{ $d->user_id }}</td>
                <td>{{ $d->username }}</td>
                <td>{{ $d->nama }}</td>
                <td>{{ $d->level_id }}</td>
            </tr>
            @endforeach
        </table>
    </body>
</html>
```

7. Jalankan di browser, catat dan laporkan apa yang terjadi
8. Setelah itu, kita modifikasi lagi file UserController

app > Http > Controllers > UserController.php > ...

```
<?php

namespace App\Http\Controllers;

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

class UserController extends Controller
{
    public function index()
    {
        // tambah data user dengan Eloquent Model
        $data = [
            'username' => 'customer-1',
            'nama' => 'Pelanggan',
            'password' => Hash::make('12345'),
            'level_id' => 4
        ];
        UserModel::insert($data); // tambahkan data ke tabel m_user

        // coba akses model UserModel
        $user = UserModel::all(); // ambil semua data dari tabel m_user
        return view('user', ['data' => $user]);
    }
}
```

9. Jalankan di browser, amati dan laporkan apa yang terjadi
10. Kita modifikasi lagi file UserControllermenjadi seperti berikut

```php
class UserController extends Controller
{
    public function index()
    {
        // tambah data user dengan Eloquent Model
        $data = [
            'nama' => 'Pelanggan Pertama',
        ];
        UserModel::where('username', 'customer-1')->update($data); // update data user

        // coba akses model UserModel
        $user = UserModel::all(); // ambil semua data dari tabel m_user
        return view('user', ['data' => $user]);
    }
}
```

11. Jalankan di browser, amati dan laporkan apa yang terjadi
12. Jika sudah, laporkan hasil Praktikum-6 ini dan *commit* perubahan pada *git*

G. Penutup

Jawablah pertanyaan berikut sesuai pemahaman materi di atas

1. Pada **Praktikum 1 - Tahap 5**, apakah fungsi dari APP_KEYpada *file setting* .envLaravel?
2. Pada **Praktikum 1**, bagaimana kita men-*generate* nilai untuk APP_KEY?
3. Pada **Praktikum 2.1 - Tahap 1**, secara *default* Laravel memiliki berapa file migrasi? dan untuk apa saja file migrasi tersebut?
4. Secara *default*, file migrasi terdapat kode $table->timestamps();, apa tujuan/*output* dari fungsi tersebut?
5. Pada File Migrasi, terdapat fungsi $table->id();Tipe data apa yang dihasilkan dari fungsi tersebut?
6. Apa bedanya hasil migrasi pada table m_level, antara menggunakan $table->id();
   dengan menggunakan $table->id('level_id');?
7. Pada migration, Fungsi ->unique()digunakan untuk apa?
8. Pada **Praktikum 2.2 - Tahap 2**, kenapa kolom level_id pada tabel m_user menggunakan $tabel->unsignedBigInteger('level_id'), sedangkan kolom level_id pada tabel m_level menggunakan $tabel->id('level_id') ?
9. Pada **Praktikum 3 - Tahap 6**, apa tujuan dari Class Hash? dan apa maksud dari kode program Hash::make('1234');?
10. Pada **Praktikum 4 - Tahap 3/5/7**, pada *query builder* terdapat tanda tanya (?), apa kegunaan dari tanda tanya (?) tersebut?
11. Pada **Praktikum 6 - Tahap 3**, apa tujuan penulisan kode protected $table = ‘m_user’; dan protected $primaryKey = ‘user_id’;?
12. Menurut kalian, lebih mudah menggunakan mana dalam melakukan operasi CRUD ke database (*DB Façade / Query Builder / Eloquent ORM*) ? jelaskan

Jawaban ;

1. APP_KEY adalah sebuah konfigurasi pada file ‘.env’ di laravel yang digunakan untuk menyimpan kunci enkripsi aplikasi. Kunci ini digunakan untuk menjaga keamanan data yang dienkripsi, seperti session cookies dan data lainnya.
2. Untuk generate nilai ‘APP_KEY’, kita dapat menggunakan perintah artisan didalam terminal dengan menggunakan ‘php artisan key:generate’.
3. Secara default, laravel memiliki tiga file migrasi: ‘create_users_table.php’ ’create_password_resets_table.php’, dan ‘cretae_failed_jobs_table’. File ini digunakan untuk membuat table pengguna (users), reset password, dan table log untuk failed jobs.
4. Kode $table->timestaps();, secara otomatis menambahkan dua kolom, yaitu

'created_at' dan 'updated_at' yang digunakan untuk melacak waktu pembuatan dan pembaruan suatu record dalam tabel.

5. Fungsi '$table->ide();' menghasilkan tipe data 'unsigned big integer' yang secara otomatis bertambah nilainya (auto-incrementing). Digunakan untuk menentukan kolom primary key dalam sebuah tabel.
6. Perbedaan antara menggunakan '$table->id();' dan '$table->id('level_id');' pada hasil migrasi pada tabel 'm_level' menggunakan '->id('level_id')' memberi nama kolom primary key secara eksplisit, yaitu 'level_id'. Sedangkan '->id();' secara implisit.
7. Fungsi '->unique()' pada migration digunakan untuk menentukan nilai pada kolom tersebut harus unik, tidak boleh ada duplikat nilai antara baris-baris dalam tabel.
8. Kolom 'level_id' pada tabel 'm_user' menggunakan '$table->unsignedBigInteger('level_id')' diasumsikan sebagai foreign key yang merujuk ke primary key di tabel lain. Sedangkan pada tabel 'm_level', '$table->id('level_id')' digunakan untuk mendefinisikan primary key dengan nama kolom 'level_id'.
9. Class 'Hash' digunakan untuk melakukan enkripsi data, seperti enkripsi password pengguna. Kode program 'Hash::make('12345');' menghasilkan hasil enkripsi dari string '12345' yang biasanya digunakan saat membuat dan menyimpan password baru pengguna ke dalam database.
10. Tanda tanya (?) pada query builder digunakan untuk menandai parameter yang akan di-bind secara dinamis ke dalam query, memungkinkan penggunaan parameter query dan melindungi terhadap serangan SQL Injection.
11. Penulisan kode berikut adalah bagian dari model Eloquent dalam framework laravel. Tujuan dari kode ini adalah untuk menghubungkan model tersebut dengan tabel tertentu dalam basis data dan menetapkan primary key dari tabel tersebut.
12. Menurut saya pribadi lebih mudah jika menggunakan Query Builder dikarenakan memiliki sintaks yang digunakan untuk mempermudah penggunaan query dan juga sintaks tersebut mudah dipahami dibanding dengan Eloquent Orm yang sintaks nya sulit untuk dipahami dan juga pada DB Facade yang penulisan sintaksnya masih manual.