

PERTEMUAN KE 1

Tujuan Pembelajaran

Tujuan Pembelajaran yang harus dicapai dalam modul ini adalah mahasiswa mampu:

- Menginstal XAMPP dan Composer di komputer masing-masing
- Membuat project Laravel baru
- Menjalankan Laravel melalui terminal
- Mengatur koneksi database MySQL lokal
- Memahami struktur database aplikasi Poliklinik
- Membuat migrasi di Laravel sesuai struktur database
- Menjalankan migrasi ke database MySQL
- Membuat model Eloquent menggunakan perintah `php artisan make:model`
- Menambahkan relasi antar model menggunakan fungsi `hasMany` dan `belongsTo`
- Menerapkan relasi antar tabel pada studi kasus aplikasi Poliklinik (contoh: Poli → Dokter, Dokter → Jadwal)

Modul 1 : Pengenalan Laravel & Lingkungan + Instalasi

1.0 Overview

Tahap awal pengembangan Laravel meliputi instalasi XAMPP, Composer, dan Visual Studio Code sebagai lingkungan kerja. Proyek Laravel dibuat dan dijalankan dengan `php artisan serve`, lalu database dikonfigurasi melalui file `.env` dan migrasi dijalankan. Laravel menggunakan arsitektur MVC (Model–View–Controller) untuk memisahkan logika aplikasi: Model mengelola data, View menampilkan antarmuka, dan Controller menghubungkan keduanya. Dengan ini, aplikasi siap dikembangkan secara terstruktur dan efisien.

1.1 Instalasi XAMPP

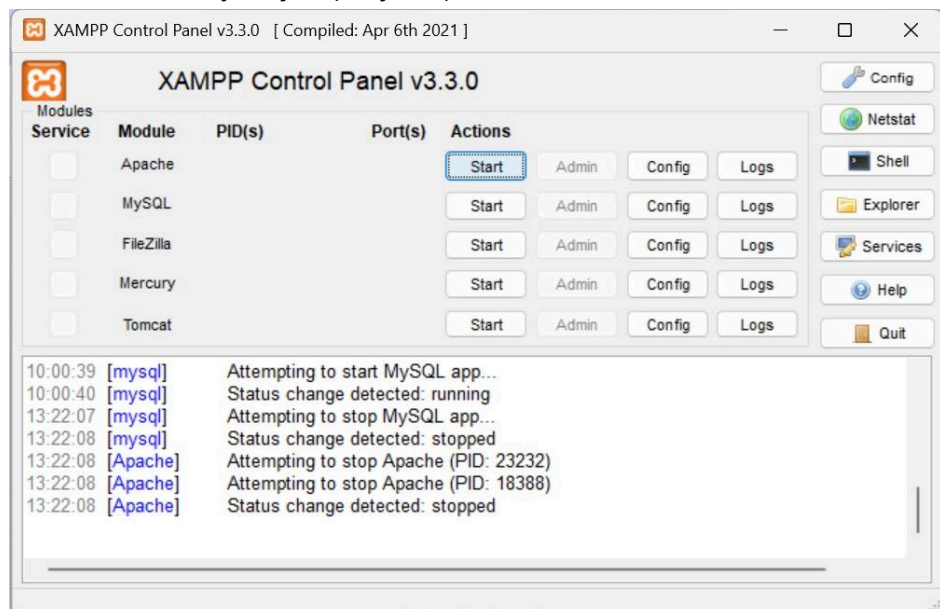
Apa itu XAMPP?

XAMPP adalah paket server gratis yang berisi:

- Apache (web server)
- MySQL/MariaDB (database)
- PHP (interpreter)
- phpMyAdmin (GUI untuk MySQL)

Langkah-Langkah Instalasi XAMPP

1. Download
 - Buka <https://www.apachefriends.org/index.html>
 - Pilih versi untuk sistem operasi Anda (Windows / macOS / Linux)
 - **Direkomendasikan versi PHP 8.2 atau lebih tinggi** untuk Laravel 10+
2. Instalasi (Windows/macOS)
 - Jalankan file installer
 - Ikuti wizard instalasi, centang: Apache, MySQL, PHP, phpMyAdmin
 - Install ke folder default (C:\xampp atau /Applications/XAMPP di macOS)
3. Menjalankan XAMPP
 - Buka **XAMPP Control Panel**
 - Klik **Start** pada **Apache** dan **MySQL**
 - Pastikan statusnya hijau (berjalan)



1.2 Instalasi Composer

Apa itu Composer?

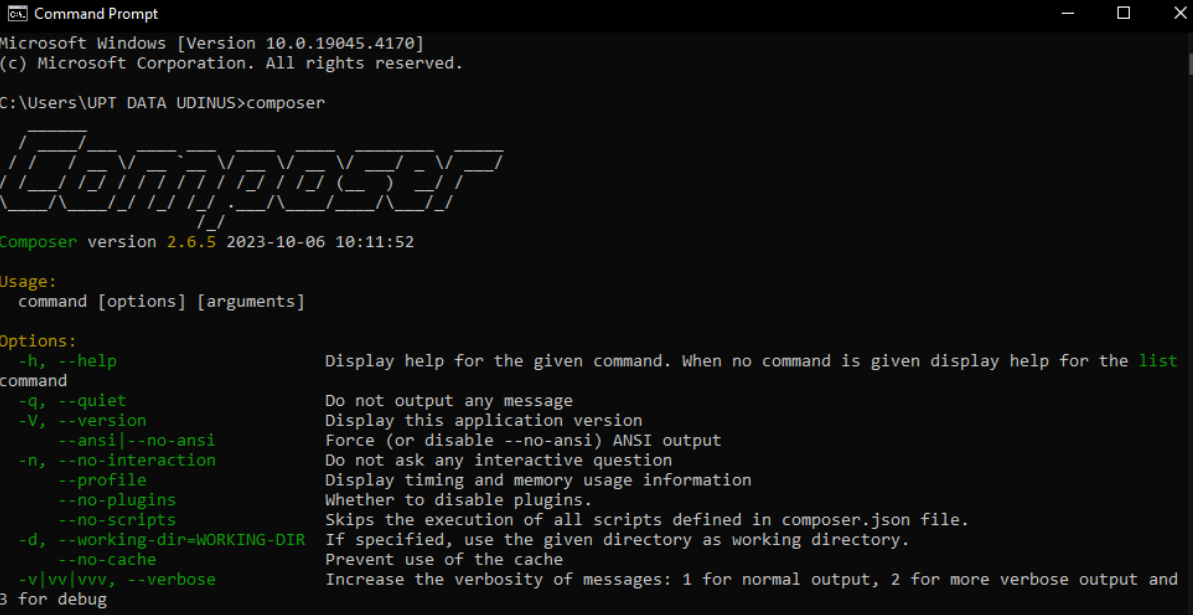
Composer adalah **package manager untuk PHP**. Laravel membutuhkan Composer untuk mengelola dependency (library).

Cara Install Composer

- **Windows:**
 1. Download installer dari <https://getcomposer.org/download/>

2. Jalankan installer
 3. Pilih file `php.exe` di `C:\xampp\php\php.exe`
 4. Selesai
- **macOS:**
 1. Buka Terminal
 2. Jalankan perintah berikut: `brew install composer`
 - **atau Manual :**
 1. `php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"`
 2. `php composer-setup.php`
 3. `sudo mv composer.phar /usr/local/bin/composer`

Setelah selesai install lakukan pengecekan di terminal untuk cek apakah composer sudah benar-benar terinstall di komputer kalian.



```
Command Prompt
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

C:\Users\UPT DATA UDINUS>composer

Composer version 2.6.5 2023-10-06 10:11:52

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display help for the given command. When no command is given display help for the list
command
  -q, --quiet               Do not output any message
  -V, --version              Display this application version
  --ansi|--no-ansi          Force (or disable) ANSI output
  -n, --no-interaction       Do not ask any interactive question
  --profile                 Display timing and memory usage information
  --no-plugins              Whether to disable plugins.
  --no-scripts              Skips the execution of all scripts defined in composer.json file.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  --no-cache                Prevent use of the cache
  -v|vv|vvv, --verbose      Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and
3 for debug
```



1.3 Instal Visual Studio Code



Install Visual Studio Code <https://code.visualstudio.com/download>



1.4 Instalasi Laravel

1. **Buka terminal (CMD / Git Bash / Terminal)**
Setelah membuka visual studio code, silahkan buka folder yang akan digunakan untuk menyimpan project dengan cara open folder kemudian pilih folder yang diinginkan
2. Setelah itu buka terminal kemudian install laravel dengan cara
`composer create-project laravel/laravel poliklinik-app "11.*"`
3. Setelah selesai, folder `poliklinik-app` akan muncul

1.5 Menjalankan Laravel

1. **Jalankan Laravel dari Terminal**
Setelah selesai install, silahkan masuk ke directory projectnya dan menjalankan laravel dengan cara dibawah ini :
`cd poliklinik-app`
`php artisan serve`

Akses di browser : `http://localhost:8000`

⚙️ 1.6 Konfigurasi Koneksi Database

1. Buka file `.env` di dalam folder project:

```
DB_CONNECTION=sqlite  
  
# DB_HOST=127.0.0.1  
  
# DB_PORT=3306  
  
# DB_DATABASE=laravel  
  
# DB_USERNAME=root  
  
# DB_PASSWORD=
```

2. Rubah Menjadi

```
DB_CONNECTION=mysql  
  
DB_HOST=127.0.0.1  
  
DB_PORT=3306  
  
DB_DATABASE=poliklinik_db  
  
DB_USERNAME=root  
  
DB_PASSWORD= # kosongkan jika XAMPP default
```

3. Jalankan migrasi awal:

```
php artisan migrate
```

📁 Modul 2: Desain Database & Migrasi Laravel

📄 2.0 Overview

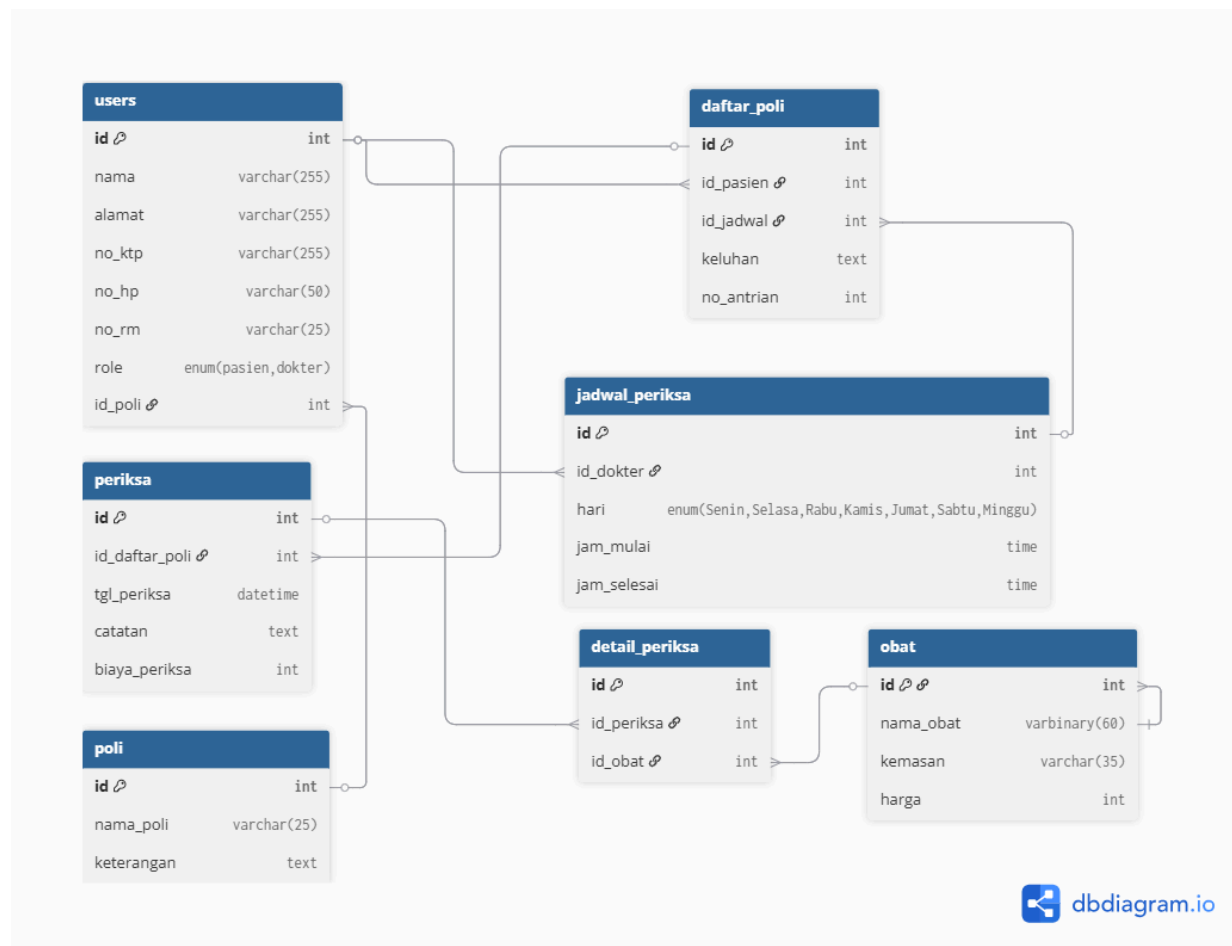
Pada modul ini, kita akan membangun "tulang punggung" dari aplikasi kita, yaitu struktur database. Kita akan menggunakan fitur **Laravel Migrations** untuk membuat semua tabel yang dibutuhkan secara otomatis, seperti tabel untuk data pengguna, poli, jadwal pemeriksaan, hingga data pemeriksaan dan obat-obatan.

2.1 Struktur Tabel Aplikasi Poliklinik

1. **users** – data pengguna (dokter & pasien)
2. **poli** – data poli
3. **jadwal_periksa** – jadwal dokter per hari
4. **daftar_poli** – pendaftaran pasien ke poli
5. **periksa** – hasil pemeriksaan
6. **obat** – daftar obat
7. **detail_periksa** – relasi pemeriksaan ↔ obat

Gambar Diagram Database (Link DB Diagram :

<https://dbdiagram.io/d/BK-Web-Dev-Udinus-682df5bbb9f7446da38b98dd>):



2.2 Membuat Migrasi di Laravel

Migrasi digunakan untuk mengelola struktur database menggunakan kode PHP (seperti kontrol versi database).

2.2.1: Buat Semua File Migrasi

Di terminal, jalankan:

[TERMINAL]

```
php artisan make:migration create_poli_table
php artisan make:migration create_jadwal_periksa_table
php artisan make:migration create_daftar_poli_table
php artisan make:migration create_periksa_table
php artisan make:migration create_obat_table
php artisan make:migration create_detail_periksa_table
```


2.2.2: Isi File Migrasi

Setelah file-file migrasi dibuat, isi file-file migrasi dengan:

1. **Migrasi users** (nama file: `xxxx_create_users_table.php`):

```
1 public function up(): void
2 {
3     Schema::create('users', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->string('alamat')->nullable();
7         $table->foreignId('id_poli')->nullable()->constrained('poli')->cascadeOnDelete();
8         $table->string('no_ktp')->nullable();
9         $table->string('no_hp')->nullable();
10        $table->string('no_rm', 25)->nullable();
11        $table->enum('role', ['pasien', 'dokter', 'admin']);
12        $table->string('email')->unique();
13        $table->string('password');
14        $table->rememberToken();
15        $table->timestamps();
16    });
```

2. **Migrasi poli** (nama file: `xxxx_create_poli_table.php`):



```

1 public function up(): void
2     {
3         Schema::create('poli', function (Blueprint $table) {
4             $table->id();
5             $table->string('nama_poli', 25);
6             $table->text('keterangan')->nullable();
7             $table->timestamps();
8         });
9     }

```

3. Migrasi **jadwal_periksa** (nama file: xxxx_create_jadwal_periksa_table.php):




```

1 public function up(): void
2     {
3         Schema::create('jadwal_periksa', function (Blueprint $table) {
4             $table->id();
5             $table->foreignId('id_dokter')->constrained('users')->cascadeOnDelete();
6             $table->enum('hari', ['Senin', 'Selasa', 'Rabu', 'Kamis', 'Jumat', 'Sabtu', 'Minggu']);
7             $table->time('jam_mulai');
8             $table->time('jam_selesai');
9             $table->timestamps();
10        });
11    }

```

4. Migrasi **daftar_poli** (nama file: xxxx_create_daftar_poli_table.php):



```

1 public function up(): void
2     {
3         Schema::create('daftar_poli', function (Blueprint $table) {
4             $table->id();
5             $table->foreignId('id_pasien')->constrained('users')->cascadeOnDelete();
6             $table->foreignId('id_jadwal')->constrained('jadwal_periksa')->cascadeOnDelete();
7             $table->text('keluhan');
8             $table->integer('no_antrian');
9             $table->timestamps();
10        });
11    }

```


5. Migrasi **periksa** (nama file: xxxx_create_periksa_table.php):

```
1 public function up(): void
2 {
3     Schema::create('periksa', function (Blueprint $table) {
4         $table->id();
5         $table->foreignId('id_daftar_poli')->constrained('daftar_poli')->cascadeOnDelete();
6         $table->dateTime('tgl_periksa');
7         $table->text('catatan')->nullable();
8         $table->integer('biaya_periksa');
9         $table->timestamps();
10    });
11 }
```

6. Migrasi **obat** (nama file: xxxx_create_obat_table.php):

```
1 public function up(): void
2 {
3     Schema::create('obat', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama_obat');
6         $table->string('kemasan', 35)->nullable();
7         $table->integer('harga');
8         $table->timestamps();
9     });
10 }
```

7. Migrasi **detail_periksa** (nama file:
xxxx_create_detail_periksa_table.php):

```

1 public function up(): void
2 {
3     Schema::create('detail_periksa', function (Blueprint $table) {
4         $table->id();
5         $table->foreignId('id_periksa')->constrained('periksa')->cascadeOnDelete();
6         $table->foreignId('id_obat')->constrained('obat')->cascadeOnDelete();
7         $table->timestamps();
8     });
9 }

```

2.2.3: Jalankan Migrasi

Setelah semua file selesai, jalankan:

[TERMINAL]

```
php artisan migrate
```

Jika berhasil, maka semua tabel akan muncul di database **poliklinik_db**. Untuk melakukan pengecekan apakah database sudah sesuai, bisa di cek di phpmyadmin dan membuka database bernama **poliklinik_db**.

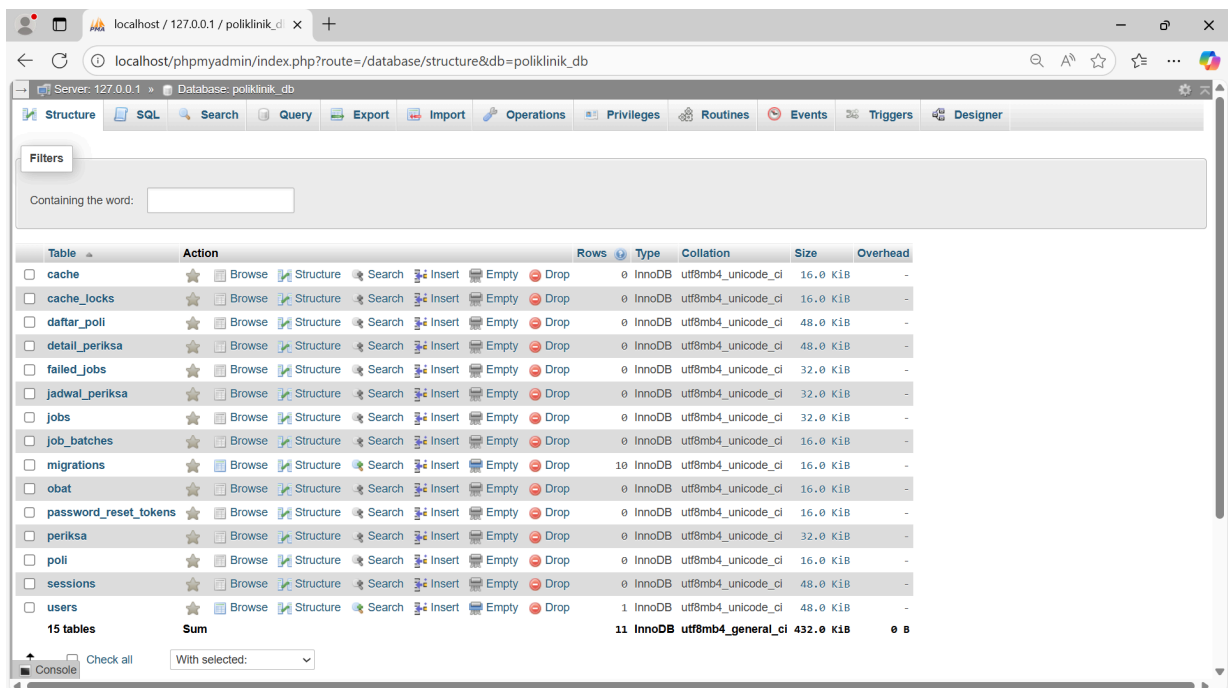


Table	Action	Rows	Type	Collation	Size	Overhead
cache		0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
cache_locks		0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
daftar_poli		0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
detail_periksa		0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
failed_jobs		0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
jadwal_periksa		0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
jobs		0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
job_batches		0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
migrations		10	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
obat		0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
password_reset_tokens		0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
periksa		0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
poli		0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
sessions		0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
users		1	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
15 tables	Sum	11	InnoDB	utf8mb4_general_ci	432.0 KiB	0 B

Modul 3 : Membuat Model dan Relasi Eloquent

3.1 Overview

Pada modul ini, kita akan membangun fondasi logika data aplikasi dengan memanfaatkan Laravel Eloquent Model, di mana setiap tabel pada database direpresentasikan dalam bentuk model agar lebih mudah diakses dan dikelola. Selain itu, kita akan mempelajari bagaimana mendefinisikan relasi antar tabel menggunakan fungsi `hasMany` dan `belongsTo`, misalnya satu Poli dapat memiliki banyak Dokter, sementara setiap Dokter hanya dimiliki oleh satu Poli. Studi kasus yang digunakan adalah aplikasi Poliklinik, sehingga peserta dapat langsung memahami hubungan nyata antar data seperti Poli, Dokter, Jadwal Periksa, hingga Detail Pemeriksaan. Tidak hanya itu, modul ini juga memperkenalkan penggunaan properti `$fillable` sebagai langkah penting untuk mengamankan proses mass assignment, sehingga hanya kolom tertentu yang bisa diisi secara otomatis dari input pengguna.

3.2 Model dan Relasi

3.2.1 Apa itu Model ?

Model adalah bagian dari MVC (Model, View, Controller) di laravel yang berfungsi sebagai penghubung antara aplikasi dan database. Model mempresentasikan tabel dalam database, sehingga setiap objek model adalah representasi dari satu baris data di tabel tersebut. Dalam Laravel, model digunakan untuk:

- Berinteraksi dengan tabel database (baca, tambah, edit, dan hapus data)
- Membangun relasi antar tabel menggunakan fitur **Eloquent ORM (Object Relational Mapping)**
- Menyederhanakan logika manipulasi data

3.2.2 Relasi di Laravel Eloquent

Laravel menggunakan **Eloquent ORM (Object Relational Mapping)** untuk menangani relasi antar tabel database secara objektif dan efisien. Beberapa jenis relasi yang sering digunakan:

- `hasMany()`

Relasi satu ke banyak, artinya satu data di model A bisa memiliki banyak data di model B. contoh:

- Satu `Poli` memiliki banyak `Dokter`
- Satu `Dokter` memiliki banyak `JadwalPeriksa`

- `belongsTo()`

Relasi kebalikan dari `hasMany`, yaitu data di model B dimiliki oleh satu data di model A. Contoh:

- Setiap **Dokter** hanya milik satu **Poli**
- Setiap **JadwalPeriksa** hanya milik satu **Dokter**

3.2.3 Apa itu **\$fillable**?

Properti di model Laravel yang digunakan untuk menentukan kolom apa saja yang boleh diisi sekaligus secara otomatis (massal) atau mass assign (diisi secara langsung menggunakan array atau request).

3.3 Langkah-Langkah Membuat Model dan Relasi Eloquent

3.3.1 Membuat Model dengan Artisan

Gunakan perintah berikut untuk membuat model yang diperlukan dalam aplikasi Poliklinik:

[TERMINAL]

```
php artisan make:model Poli
php artisan make:model JadwalPeriksa
php artisan make:model DaftarPoli
php artisan make:model Periksa
php artisan make:model Obat
php artisan make:model DetailPeriksa
```

Perintah ini akan menghasilkan file model di dalam direktori:

app/models

3.3.2 Menambahkan Relasi Antar Model dan Fillable

Setelah model dibuat, langkah berikutnya adalah menentukan relasi antar model dan menambahkan atribut **\$table** dan **\$fillable** agar data bisa di-mass assign melalui controller atau form.

- **Model Poli**

app/Models/Poli.php



```
1  class Poli extends Model
2  {
3      protected $table = 'poli';
4
5      protected $fillable = [
6          'nama_poli',
7          'keterangan'
8      ];
9
10     public function dokters(){
11         return $this->hasMany(User::class, 'id_poli');
12     }
13 }
```

Model Poli berfungsi untuk mengelola data pada tabel poli. Pada model Poli terdapat fungsi relasi `dokters()` yang berfungsi untuk mendefinisikan relasi one to many (`hasMany`) artinya satu poli bisa memiliki banyak user role dokter dengan foreign key `id_poli` pada model `User`

- **Model Periksa**

`app/Models/Periksa.php`

```

1  class Periksa extends Model
2  {
3      protected $table = 'periksa';
4
5      protected $fillable = [
6          'id_daftar_poli',
7          'tgl_periksa',
8          'catatan',
9          'biaya_periksa'
10     ];
11
12     public function daftarPoli(){
13         return $this->belongsTo(DaftarPoli::class, 'id_daftar_poli');
14     }
15
16     public function detailPeriksas(){
17         return $this->hasMany(DetailPeriksa::class, 'id_periksa');
18     }
19 }

```

Model periksa berfungsi untuk mengelola data pada tabel periksa. Pada model periksa terdapat fungsi relasi `daftarPoli()` berfungsi untuk mendefinisikan relasi many to one (`belongsTo`) artinya Setiap periksa pasti terkait dengan satu `id_daftar_poli` yang dihubungkan pada model `DaftarPoli`. Pada model Periksa juga terdapat fungsi relasi `detailPeriksas()` berfungsi untuk mendefinisikan relasi one to many (`hasMany`) artinya setiap Satu pemeriksaan memiliki detail pemeriksaan dengan foreign key `id_periksa` pada Model `DetailPeriksa`

- **Model Obat**

`app/Models/Obat.php`

```
1  class Obat extends Model
2  {
3      protected $table = 'obat';
4
5      protected $fillable = [
6          'nama_obat',
7          'kemasan',
8          'harga'
9      ];
10
11     public function detailPeriksas(){
12         return $this->hasMany(DetailPeriksa::class, 'id_obat');
13     }
14 }
```

Model obat berfungsi untuk mengelola data pada tabel obat. Pada model obat terdapat fungsi relasi `detailPeriksas()` berbeda pada model periksa fungsi relasi kali ini mendefinisikan one to many (`hasMany`) pada setiap obat bisa muncul di banyak detail pemeriksaan dengan foreign key `id_obat` pada model `DetailPeriksa`

- **Model Jadwal Periksa**

`app/Models/JadwalPeriksa`

```

1  class JadwalPeriksa extends Model
2  {
3      protected $table = 'jadwal_periksa';
4
5      protected $fillable = [
6          'id_dokter',
7          'hari',
8          'jam_mulai',
9          'jam_selesai'
10     ];
11
12     public function dokter(){
13         return $this->belongsTo(User::class, 'id_dokter');
14     }
15
16     public function daftarPolis(){
17         return $this->hasMany(DaftarPoli::class, 'id_jadwal');
18     }
19
20 }

```

Model JadwalPeriksa berfungsi untuk mengelola data pada tabel `jadwal_periksa`. Pada model obat terdapat fungsi relasi `dokter()` berfungsi untuk mendefinisikan relasi many to one (`belongsTo`) artinya setiap jadwal pemeriksaan dimiliki oleh satu dokter dengan foreign key `id_dokter` pada Model `User`. Pada model pemeriksaan juga terdapat table relasi `daftarPolis()` berfungsi untuk mendefinisikan relasi one to many (`hasMany`) artinya Satu jadwal pemeriksaan bisa digunakan oleh banyak pendaftaran poli dengan foreign key `id_jadwal` pada Model `DaftarPoli`.

- **Model Detail Periksa**

`app/Models/DetailPeriksa.php`


```
1 class DetailPeriksa extends Model
2 {
3     protected $table = 'detail_periksa';
4
5     protected $fillable = [
6         'id_periksa',
7         'id_obat'
8     ];
9
10    public function periksa(){
11        return $this->belongsTo(Periksa::class, 'id_periksa');
12    }
13
14    public function obat(){
15        return $this->belongsTo(Obat::class, 'id_obat');
16    }
17 }
```

Model DetailPeriksa berfungsi untuk mengelola data pada tabel `detail_periksa`. pada Model ini terdapat fungsi relasi `periksa()` berfungsi untuk mendefinisikan relasi many to one (`belongsTo`) artinya setiap detail pemeriksaan dimiliki oleh satu data pemeriksaan dengan foreign_key `id_periksa` pada model `Periksa`. Pada model `periksa` juga terdapat fungsi relasi `obat()` artinya setiap detail pemeriksaan terkait dengan satu obat tertentu dengan foreign key `id_obat` pada model `Obat`

- **Model Daftar Poli**

app/Models/DaftarPoli.php

```
1  class DaftarPoli extends Model
2  {
3      protected $table = 'daftar_poli';
4
5      protected $fillable = [
6          'id_pasien',
7          'id_jadwal',
8          'keluhan',
9          'no_antrian'
10     ];
11
12     public function pasien(){
13         return $this->belongsTo(User::class, 'id_pasien');
14     }
15
16     public function jadwalPeriksa(){
17         return $this->belongsTo(JadwalPeriksa::class, 'id_jadwal');
18     }
19
20     public function periksas(){
21         return $this->hasMany(Periksa::class, 'id_daftar_poli');
22     }
23 }
24 }
```

Model DaftarPoli berfungsi untuk mengelola data pada tabel `daftar_poli`. pada model ini terdapat model relasi `pasien()` berfungsi untuk mendefinisikan relasi many to one (`belongsTo`) artinya setiap pendaftaran poli dimiliki oleh satu pasien dengan foreign key `id_pasien` pada model `User`. pada model ini juga terdapat model relasi `jadwalPeriksa()` berfungsi untuk mendefinisikan relasi many to one (`belongsTo`) artinya setiap pendaftaran poli terkait dengan satu jadwal periksa dengan foreign key `id_jadwal` pada tabel `JadwalPeriksa`. selain dua tabel relasi diatas model ini juga terdapat fungsi relasi `periksas()` berfungsi untuk mendefinisikan relasi one to many (`hasMany`) artinya satu pendaftaran poli bisa memiliki banyak data pemeriksaan dengan foreign key `id_daftar_poli` pada model `Periksa`

- **Model User**

app/Models/User.php

```
1 class User extends Authenticatable
2 {
3     protected $fillable = [
4         'nama',
5         'alamat',
6         'no_ktp',
7         'no_hp',
8         'no_rm',
9         'role',
10        'id_poli',
11        'email',
12        'password'
13    ];
14    protected $hidden = [
15        'password',
16        'remember_token',
17    ];
18    protected function casts(): array
19    {
20        return [
21            'email_verified_at' => 'datetime',
22            'password' => 'hashed',
23        ];
24    }
25    public function poli(){
26        return $this->belongsTo(Poli::class, 'id_poli');
27    }
28    public function jadwalPeriksas(){
29        return $this->hasMany(JadwalPeriksa::class, 'id_dokter');
30    }
31 }
```

Model User adalah model bawaan Laravel untuk autentikasi. pada model ini terdapat properti `$hidden` berfungsi untuk menyembunyikan kolom tertentu ketika data user dikembalikan dalam bentuk JSON/array. pada model ini juga terdapat fungsi `cast()` untuk mengubah tipe data atribut model secara otomatis ketika diambil dari database atau disimpan. pada Model ini terdapat dua fungsi relasi `poli()` dan `jadwalPeriksas()`. pada `poli()` memiliki fungsi untuk mendefinisikan relasi many to one (`belongsTo`) artinya banyak user dengan role dokter bisa terhubung ke satu poli dengan foreign key `id_poli` pada model `Poli`. sedangkan pada `jadwalPeriksas()` memiliki fungsi untuk mendefinisikan relasi one to many

(hasMany) artinya satu dokter bisa memiliki banyak jadwal pemeriksaan dengan foreign key `id_dokter` pada tabel `JadwalPeriksa`