= ⚠️ Sebelum menginstall, yang harus dipersiapkan :

1. **Composer** : *package-manager* (di level aplikasi) untuk bahasa pemrogaman PHP.
2. **NodeJS** : *package-manager* (di level aplikasi) untuk bahasa pemrogaman Javascript.

Apa itu *package-manager?* 🧐 tools yang dapat membantu pengelolaan package pada proyek agar lebih mudah.
Apa itu *package*? 📦 Aplikasi (template) yang dibuat oleh seseorang/tim dan di share secara global maupun private agar dapat digunakan oleh orang lain.

tipe-tipe penulisan pada codingan ✍️

1. **Snake Case** 🐍 : memisahkan kata dengan _ (underscore)
   ➡️ nama file migration : create_users_table
   ➡️ name route : name('index_user')
2. **Kebab Case** 🥙: memisahkan kata dengan - (garis)
   ➡️ css : margin-top
   ➡️ class bootstrap : d-flex
   ➡️ nama folder : student-dashboard
3. **All Caps** 🔤 : semuanya kapital
   ➡️ env : DB_DATABASE
4. **Camel Case** 🐪: menggabungkan antar kata
   ➡️ huruf besar per tiap kata
   ➡️ nama controller : TodoController
   ➡️ huruf besar per kata selain yg pertama
   ➡️ nama variable : dataStudent

Hal-hal penting dalam pembuatan project laravel 🙂

1. Perintah create project 🆕 :
   composer create-project laravel/laravel student-data
   Gunakan aturan penulisan tipe Kebab Case untuk penamaan project.

2. Mengaktifkan server laravel 🌍 :
   php artisan serve
   Kenapa perlu diaktifkan?
   Karna agar bisa diakses di browser
   Kustom port server :
   php artisan serve –port=2222
   Kustom ini biasanya digunakan ketika?
   Supaya dapat menjalankan lebih dari satu project laravel

3. Membuat database di phpmyadmin 🗒️
4. Mengubah **.env** 📟 untuk menyambungkannya dengan database yg ada di phpmyadmin
   Bagian apa yang diubah?
   DB_DATABASE

5. Mempersiapkan **model, migration, controller** 🔃
   `php artisan make:model Todo -mcr`
   - make:model ➡️ **model** : untuk mengelola database ➡️ berada di folder app\models
   - m ➡️ **migration** : untuk membuat table di database ➡️ berada di folder database\migrations
   - c ➡️ **controller** : untuk menghubungkan model dan view ➡️ berada di folder app\http\controllers
   - r ➡️ **resources** : method-method default pada controller ➡️ menengani proses CRUD, 7 method :
     - 📑 index : menampilkan halaman awal yang mengambil banyak data
     - 📑 show : menampilkan satu data spesifik
     - 📑 create : menampilkan form untuk tambah data
     - 📑 store : menambahkan data baru ke database
     - 📑 edit : menampilkan halaman form untuk edit
     - 📑 update : mengubah data di database
     - 📑 destroy : menghapus data di database

   Bagaimana jika ingin membuat file-file tersebut (migration, controller, model) secara terpisah?
   `php artisan make:migration create_students_table`
   `php artisan make:controller StudentController -r`
   `php artisan make:model Student`

6. Membuat **column** pada table melalui **migration** 📑
   `$table->tipedata('nama_column');`
   Gunakan tipe penulisan snake case untuk penamaan column
   Apa itu tipe data enum pada migration laravel?
   Tipe data yang value nya hanya boleh sesuai dengan yang telah diatur
   Maksimal pilihan : 3
   Bagaimana mendefinisikannya?
   `$table->enum("nama_column", ['pilihan1','pilihan2','pilihan3']);`

7. **Memindahkan table migration** ke server database 📤
   `php artisan migrate`
   Ketika ada penambahan file migration baru atau penambahan column baru pada migration yang telah dipindahkan ke server, perintah yang diberikan?
   `php artisan migrate:refresh`

8. Konfigurasi **model** 📝
   Memastikan terdapat property :
   `protected $fillable = [];`
   Fungsinya untuk? Menentukan column mana saja yang akan diberi nilai/value

Apabila nama model dan migrationnya tidak sesuai standar laravel, maka perlu menambahkan property : protected $table = 'nama_table';
Contoh penamaan model-controller-migration yang sesuai dengan standar laravel :
Todo : model ➡️ Tunggal dan menggunakan tipe penulisan camel case
todos : migration ➡️ Jamak dan menggunakan tipe penulisan snake case
TodoController : controller ➡️ Tunggal dan menggunakan tipe penulisan camel case

9. Konfigurasi **route** 🗺️
   - Import controller yang akan digunakan dibagian atas :
     use App\Http\Controllers\NamaController;
   - Format :
     Route::method('/path', [NamaController::class, 'methodController']);
   - Terdapat 4 method route :
     ➡️ get : mengambil data
     ➡️ post : menyimpan data atau menambahkan data
     ➡️ patch/put : merubah data
     ➡️ delete : menghapus data

10. Membuat **template layout** 🎨
    Mengapa membuat template layout? Untuk menghindari penulisan baris kode yang sama secara berulang
    File yang berfungsi untuk menampilkan sesuatu dibuat dengan ekstensi file .blade.php dan disimpan pada folder resources/views
    Biasanya berisikan tag pembuka html, css cdn ataupun custom, js.
    ➡️ tambahkan @yield('nama') di dalam tag body, berfungsi untuk menampung baris kode yang berbeda tiap blade nya
    ➡️ tambahkan @extends('nama_file') pada blade yang akan menggunakan file layout yang telah dibuat
    ➡️ tambahkan @section('namayield') @endsection pada blade yang extends, berfungsi untuk memanggil yield

11. Penyimpanan **assets** 🖼️
    Berupa css, js, image yang berasal dari luar/custom (selain default laravel).
    Pemanggilannya : "{{asset('folder/file')}}"
    Misal : terdapat sebuah gambar yang disimpan di dalam folder assets/image dan nama filenya "gambar" dengan esktensi jpg. Maka pemanggilannya?
    {{asset('assets/image/gambar.jpg')}}

12. Fitur CRUD 👩‍💻
    Apa itu CRUD? Create, Read, Update, Delete : fitur untuk memodifikasi database
    a. C (**Create**) ➕
       Ketika akan membuat fitur "C" pada Laravel, hal-hal yang perlu diperhatikan pada file .blade.php yang akan menampilkan formulir tambah datanya, ialah 🤔
       - Terdapat tag **form** yang memiliki **attribute** **action** dan **method**

- Untuk form yang akan mengupload file/image harus menambahkan attribute **enctype** dengan value **multipart/form-data** pada tag form nya
- Terdapat @csrf dibawah tag pembuka form
- Pada tag input terdapat attribute **name** yang valuenya sama dengan **column** pada table migration
- Terdapat tag button dengan attribute type yang memiliki value **submit**

Pada route yang menangani proses tambah data (route yang disimpan pada attribute **action** tag form), pastikan route method nya merupakan **post**
Adapun flow dasar pada method resources store nya berupa 😉

- Terdapat parameter **Request $request**
- Terdapat validasi

```
$request->validate([
        'value_attribute_name' => 'validasi1|validasi2',
]);
```

Berikut contoh pembuatan validasi :

```
$request->validate([
        'umur' => 'required|numeric',
]);
```

- Setelah validasi, berikan perintah untuk menambahkan data baru ke database melalui model dengan format :

```
NamaModel::create([
        'nama_column' => $request->value_attribute_name,
]);
```

- Menambahkan baris kode berikut yang berfungsi untuk **menentukan** kemana **halaman** akan diarahkan ketika berhasil tambah data dan **mengirim pesan pemberitahuan**

```
return redirect()->route('nama')->with('status', 'message');
```

Pilihan berikut dapat dilakukan dan memiliki fungsi yang sama :

a. return redirect()->route('nama')->with('status', 'message');
b. return redirect('/nama')->with('status', 'message');
c. return redirect()->back()->with('status', 'message');

Apa yang membedakan antara ketiganya?

(a) Manggilnya lewat name yang ada di route
(b) Manggil lewat path yang ada di route
(c) **Ngembaliin** ke halaman tempat awal

- Pada file blade yang menjadi tujuan redirect diberi baris kode berikut yang berfungsi untuk menampilkan message pada with :

```
@if(session('status_with'))
        <tag> {{ session('status_with') }} </tag>
@endif
```

atau bisa juga seperti berikut :

```
@if(Session::get('status_with'))
        <tag> {{ Session::get('status_with') }} </tag>
@endif
```

- Selain itu, juga menyertakan baris kode berikut untuk menampilkan message ketika terdapat error :

```
@if ($errors->any())
    <ul>
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
@endif
```

Juga bisa dengan cara berikut apabila ingin menampilkan error validasi per satu column nya :

```
@error('title')
  <p>{{ $message }}</p>
@enderror
```

b. R (**Read**) 👀

Ketika akan membuat fitur “R” pada Laravel, maka pada method controller yang menanganinya, selain return view juga perlu menambahkan baris kode untuk mengambil data :

- Mengambil semua data

```
$data = NamaModel::all();
return view('file', compact('data'));
```

- Mengambil data dengan persyaratan tertentu

```
$data = NamaModel::where('nama_column', '=',$value)->get();
return view('file', compact('data'));
```

Untuk operator, tidak hanya (=) juga bisa menggunakan (>, >=, <, <=)

Apabila persyaratan lebih dari satu column, maka bisa dengan cara :

```
$data = NamaModel::where([
        ['nama_column', '=',$value],
        ['nama_column2', '=',$value2],
])->get();
return view('file', compact('data'));
```

- **Compact** berfungsi untuk mengirim data ke blade

Isi dari compact **sama dengan** nama **variable** nya.

Untuk penulisan compact yang lebih dari satu dapat dilakukan dengan :

```
compact('data1', 'data2');
```

- Mengambil data dengan pagination

```
$data = NamaModel::simplePaginate(5)->get();
return view('file', compact('data'));
```

Argument pada method **simplePaginate** merupakan **jumlah data** yang akan ditampilkan per-page.

Apa itu **pagination**?😵

Proses **pembagian konten/data** menjadi beberapa bagian terpisah.

Ketika mengambil data dengan **pagination**, maka pada file **blade** nya perlu ditambahkan baris kode berikut untuk **menampilkan component** pagination prev-next .

```
{{ $dataYangDiCompact->links() }}
```

Adapun lainnya yang juga dapat dimunculkan⬇️
Current Page: {{ $dataYangDiCompact->currentPage() }}
Jumlah Data: {{ $dataYangDiCompact->total() }}

Data perhalaman: {{ $dataYangDiCompact->perPage() }}

- Menampilkan data dengan urutan tertentu
  Fungsi yang dapat digunakan untuk mengurutkan data, dapat menggunakan :
  `NamaModel::orderBy('column', 'tipe')->get();`
  tipe : dapat diisi dengan 'ASC' atau 'DESC'
  ➡️ **ASC** untuk mengurutkan lurus (terkecil-terbesar atau a-z)
  ➡️ **DESC** untuk mengurutkan terbalik (terbesar-terkecil atau z-a)
- Ketika mengambil banyak data dengan (`->get()`) maka akan mengembalikan data dengan tipe **array multidimensi** sehingga ketika akan mengakses tiap-tiap datanya harus menggunakan perulangan, seperti **foreach** atau **for**.
  Untuk menggunakan **foreach** pada **blade** dapat dilakukan dengan cara :

```
@foreach($dataCompact as $item)
        //untuk akses data dapat dilakukan dengan $item['column']
        //atau $item->column
@endforeach
```

atau

```
@for($i = 0; $i < count($dataCompact); $i++)
        //untuk akses data dapat dilakukan dengan
        //$dataCompact[$i]['column']
@endfor
```

- Untuk mengambil data dari **database** melalui **model** diantaranya dapat dilakukan dengan :
  a. `get()`
  b. `first()`
  c. `firstOrFail()`

  Keduanya memiliki fungsi yang sama yaitu mengambil data dan disimpan dibagian akhir proses pengambilan data, lalu apa yang membedakan antara ketiganya? 🤷
  a. Mengambil **banyak data (lebih dari satu baris)** dan disimpan dalam **array multidimensi** sehingga untuk mengakses memerlukan perulangan
  b. Mengambil hanya **satu baris data pertama** berbentuk **array assosiatif** (pemanggilan tidak memerlukan perulangan), dan

apabila data tidak ada (**null**) maka akan mengembalikan **error** laravel

c. Mengambil hanya **satu baris data pertama** berbentuk **array assosiatif** (pemanggilan tidak memerlukan perulangan), dan apabila data tidak ada (**null**) maka akan mengembalikan **"null"**

c. U (**update**) 🆙

Ketika akan membuat fitur "U" pada Laravel, maka memerlukan setidaknya **2** method resources controller, yaitu **resources edit** dan **update**
Resources edit berfungsi untuk menampilkan form edit dengan mengirim dan menampilkan data sebelumnya (data yang akan diupdate), sehingga pada **resources edit** tersebut, selain perlu menyertakan return view, juga perlu menambahkan baris kode untuk mengambil satu data spesifik yang akan diedit tersebut. Maka baris kode yang dapat dibuat ialah ⌨️

```
$data = NamaModel::where('nama_column, '=', $parameter)->firstOrFail();
Return view('file', compact('data'));
```

Lalu, ketika akan menampilkan data yang diambil tersebut didalam **input form** nya. Maka, pada **input** tersebut perlu menambahkan **attribute value**
Contohnya :

```
value="{{ $data->nama }}"
```

Dan untuk **resources update** yang berperan sebagai proses mengubah data ke databasenya. Maka, pada resources tersebut perlu terdapat beberapa hal berikut 🔢

- Memiliki minimal **2 parameter**, yaitu **Request $request** dan **$path_dinamis**
- Terdapat **validasi**
- Untuk mengirim perubahan data ke **database** nya dapat dilakukan dengan format perintah :

  ```
  NamaModel::where('nama_column', '=',
  $parameter_path_dinamis)->update([
          'nama_column' => $request->value_name_input,
  ]);
  ```

  Mengapa perlu terlebih dahulu memanggil **where** ?🤫
  Untuk filter/cari data dulu, data spesifik mana yang akan diupdate
  Lalu, ketika tidak menggunakan **where** apa diperbolehkan?🙅 **tidak**.
- Setelah proses ubah data ke **database**, maka yang berikutnya harus dilakukan ialah menentukan halaman mana yang akan diakses setelah berhasil mengubah data. Perintah yang dapat diberikan ialah 🚦

  ```
  return redirect()->route('nama')->with('status', 'message');
  ```
- Dan untuk menampilkan message berhasil ubah datanya, pada blade tujuan perlu ditambahkan baris kode berikut :

  ```
  @if(session('status_with'))
          {{ session('status_with') }}
  ```

```
@endif
```

atau bisa dengan⚔️

```
@if(Session::get('status_with'))
        {{ Session::get('status_with') }}
@endif
```

Coba perhatikan **session()** dan **Session::get()** menurutmu mengapa session pertama tidak menggunakan huruf kapital, dan session kedua menggunakan huruf kapital?🔬 karena nama-nama yang disimpan sebelum tanda (::) merupakan nama sebuah class (**camel case**)

- Selain itu, pada **route** yang menangani proses nya jangan lupa untuk menggunakan **method route** **patch** atau **put**
- Pada file blade edit nya, perlu ditambahkan `@method('patch')` atau `@method('put')` sesuaikan dengan **method route** pada **attribute acction** nya

d. D (**Delete**)🛢️

Ketika akan membuat fitur "D" pada Laravel, maka **route** yang menangani prosesnya menggunakan **method route** **delete**

Pada **resources controller** **destroy** yang menangani proses hapus data nya perlu ditambahkan perintah berikut agar data terhapus dari database :

```
NamaModel::where('column', '=', $parameter)->delete();
```

Tak lupa tambahkan perintah untuk mengembalikan ke halaman yang akan diakses setelah berhasil menghapus data :

```
return redirect()->back()->with('status', 'message');
```

Lalu perintah untuk menampilkan message pada **blade** nya :

```
@if(Session::get('status'))
        {{ Session::get('status') }}
@endif
```

13. Upload image
    - Pastikan pada tag form terdapat attribute `enctype="multipart/form-data"`
    - Input `type="file"`
    - Validasinya dapat berisi -> `'required|image|mimes:jpeg,png,jpg,gif,svg'`
    - Agar gambar yang diupload **bisa diakses/ditampilkan**, maka file yang diupload tersebut harus dipindahkan terlebih dahulu ke **public atau storage**. Untuk kali ini, kita akan memindahkannya ke public

```
$image = $request->file('name_input');
$imgName = rand() . '.' . $image->extension();
$destinationPath = public_path('/folder/');
$image->move($destinationPath, $imgName);
$uploaded = $imgName;
```

- Kemudian, untuk data yang **dimasukkan ke database ialah nama file** yang dipindahkan ke folder public tersebut

```
NamaModel::create([
        …
        'nama_column' => $uploaded
]);
```

- Untuk **mengakses/menampilkan** gambarnya dapat dilakukan seperti berikut

```
<img src="{{asset('folder/'.$item->foto)}}" alt="">
```

- Menghapus file upload yang telah tersimpan pada **folder public** dapat dilakukan dengan : (pada contoh berikut proses hapus file pada public dilakukan pada proses **destroy** baris data)

```
//mengambil data yang akan dihapus
$data = NamaModel::where('column', '=',$param)->firstOrFail();
// mencari posisi file pada folder public
$image = public_path('assets/folder/'.$data['imageColumn']);
//proses hapus file pada public
unlink($image);
//hapus baris data
$data->delete();
//mengembalikan halaman dan memberi pemberitahuan
return redirect()->back()->with('status', 'pesan');
```

- Menampilkan **gambar pada satu tab** page penuh (zoom/lihat detail gambar atau file) dapat dilakukan dengan cara berikut :

```
<a href="../folder/{{ $data->imageColumn }}" target="blank">
        <img src="{{asset('folder/'.$item->foto)}}" alt="">
</a>
```

Attribute target="blank" berfungsi untuk membuka halaman (file yg dimaksud) pada tab baru di browsernya.

14. Authentication : proses penyimpanan data login
    - Fitur **registrasi** dapat dibuat sama seperti fitur **Create** pada CRUD. yang membedakan hanya untuk bagian **create password** nya, menggunakan enkripsi data :

      ```
      Hash::make($request->password),
      ```

      Atau dapat dengan cara :

      ```
      bcrypt($request->password);
      ```

      Untuk penggunaan **hash**, harus terlebih dahulu **import** dibagian atasnya dengan

      ```
      use Illuminate\Support\Facades\Hash;
      ```

    - Untuk fitur **login** pun, aturan **form blade** nya **sama** dengan aturan form untuk **create** pada umumnya
    - Untuk method controller nya dapat diatur mulai dari **validasi**

      ```
      $request->validate([ … ]);
      ```

- Untuk proses penyimpanan data login ke fitur auth nya dapat dilakukan dengan
```
$user = $request->only('username', 'password');
if (Auth::attempt($user)) {
        return redirect()->route('name-route');
}else {
        return redirect()->back()->with('error', 'Gagal login, silahkan cek dan coba lagi!');
}
```
Untuk dapat menggunakan class **Auth::** harus terlebih dahulu **import** pada bagian atas
```
use Illuminate\Support\Facades\Auth;
```
- Untuk fitur **logout**, dapat menggunakan baris kode berikut :
```
public function logout()
{
        Auth::logout();
        return redirect('/');
}
```

15. Seeder : mengirim data ke database tanpa melalui input form
    - Penggunaan seeder biasanya digunakan untuk membuat akun admin
    - Penyimpanannya di **database/seeders**
    - Pada file **DatabaseSeeder.php** bagian **run()** dapat ditambahkan perintah create model seperti biasanya, contoh :
```
NamaModel::create([
        'column1' => 'value1',
        'password' => bcrypt('password_kamu'),
]);
```

    - Untuk mengirim datanya ke **database**, maka dapat menjalankan perintah :
```
php artisan db:seed
```

16. Middleware : pengaturan hak ijin akses
    - **Membuat** file **middleware** : `php artisan make:middleware NamaMiddleware`
    - **Definisikan** file middleware pada **kernel.php** bagian **protected $routeMiddleware**
```
'NamaMiddleware' => \App\Http\Middleware\NamaMiddleware::class,
```
    - Pada file middlewarenya, dapat diberi **logic**
```
if(Auth::check()) {
        return $next($request);
}
return redirect()->route('nama-route')->with('error', 'pesan');
```

Baris kode tersebut akan mengecek apakah auth sudah menyimpan data login, maka diperbolehkan mengakses halaman tersebut. Jika tidak ada, maka diarahkan ke halaman tertentu dengan pesan error nya.
Selain baris kode tersebut, juga dapat ditambah baris kode lain sesuai dengan kebutuhan.

- **Penggunaan** middleware pada **routenya** dapat dilakukan dengan

```
Route::middleware('NamaMiddleware')->group(function () {
        ...
});
```

17. Multiple Auth User : pengaturan hak izin akses untuk beberapa tipe akun
    - Menyediakan column pada migration yang akan menampung tipe-tipe akun, contoh :
      `$table->enum('role', ['admin', 'user']);`
      Memperbarui pada server database : `php artisan migrate:refresh`
    - Membuat file middleware yang akan mengaturnya dengan perintah ⬇️
      `php artisan make:middleware CekRole`
    - Import file middleware pada **kernel.php**
    - Menambahkan baris kode pada **file middleware** yang akan mengatur hak akses tiap role nya

```
public function handle(Request $request, Closure $next, ...$roles)
    {
        if (in_array($request->user()->role, $roles)) {
            return $next($request);
        }
        return redirect()->back();
    }
```

    - Menambahkan middleware serta role yang diperbolehkannya, pada **web.php**
      `::middleware(['cekRole:roleyangdiperbolehkanakses'])`

18. Export **PDF**
    - Install **package** ➡️ `composer require barryvdh/laravel-dompdf`
    - Import di **config/app.php**

```
'providers' => [
        Barryvdh\DomPDF\ServiceProvider::class,
];
'aliases' => [
        'PDF' => Barryvdh\DomPDF\Facade::class,
]
```

    - **Publish package** agar dapat digunakan dengan ➡️ `php artisan vendor:publish` ➡️ masukkan **angka** yang mengarah ke **Provider: Barryvdh\DomPDF\ServiceProvider** ➡️ akan menghasilkan file **config/dompdf.php**
    - Untuk menggunakannya dapat **import konstanta** ➡️ `use PDF;`

Membuat **button** dengan **tag a** yang akan trigger method pada controller yang memproses download data ⬇️

```
<a href="{{ route('export.pdf') }}">Export to PDF</a>
```

Jangan lupa untuk **mendefinisikan route** nya ⬇️

```
Route::get('/export/pdf', [NamaController::class, 'createPDF'])->name('export.pdf');
```

- Membuat **method** tambahan pada **controller** yang akan menangani proses **export pdf** ⬇️

```
public function createPDF() {
    // ambil data yg akan ditampilkan pada pdf, bisa juga dengan where atau eloquent lainnya dan jangan gunakan pagination
    $data = NamaModel::all()->toArray();
    // kirim data yg diambil kepada view yg akan ditampilkan, kirim dengan inisial
    view()->share('inisial',$data);
    // panggil view blade yg akan dicetak pdf serta data yg akan digunakan
    $pdf = PDF::loadView('pdf_view', $data);
    // download PDF file dengan nama tertentu
    return $pdf->download('pdf_file.pdf');
}
```

- Untuk file **blade** yang dipanggil pada **loadView** tersebut. pembuatan tampilan yg akan dicetak pdf hanya dapat menggunakan html css (tidak bootstrap atau framework css lainnya). pemanggilan **$data** dapat dilakukan seperti file blade lainnya, baik itu **foreach** ataupun **{{ ... }}**
- Pastikan untuk **memanggil** column pada **blade loadView** dengan `$data['column']`
- Untuk **pemanggilan gambar, tidak perlu** menggunakan **asset()** (akan memperlambat proses load) cukup **panggil langsung folder dan file** nya
- Untuk penggunakan **css**, gunakan Internal CSS

19. Export **excel** atau csv
    - **Perhatikan** beberapa persyaratan berikut sebelum menginstall package ⬇️
        1. PHP: ^7.2\|^8.0
        2. PHP extension php_zip enabled
        3. PHP extension php_xml enabled
        4. PHP extension php_gd2 enabled
        5. PHP extension php_iconv enabled
        6. PHP extension php_simplexml enabled
        7. PHP extension php_xmlreader enabled
        8. PHP extension php_zlib enabled
    - **Install package** ➡️ `composer require maatwebsite/excel`
      Apabila error, maka dapat mencoba dengan perintah berikut ⬇️
      `composer require psr/simple-cache:^1.0 maatwebsite/excel`

Lalu jalankan perintah ⬇️
composer require maatwebsite/excel

- Import pada **config/app.php** ⬇️
'providers' => [
....
Maatwebsite\Excel\ExcelServiceProvider::class,
];
'aliases' => [
....
'Excel' => Maatwebsite\Excel\Facades\Excel::class,
];
- **Publish package** agar dapat digunakan dengan ➡️ php artisan vendor:publish --provider="Maatwebsite\Excel\ExcelServiceProvider" --tag=config ➡️ akan menghasilkan file **config/excel.php**
- Membuat **file** yg akan **menangani proses export excel** ➡️ php artisan make:export NamaMigrationExport --model=NamaModel
- Pada **file**, minimal buat ketiga **method** berikut ⬇️
    1. **collection()** : menentukan atau mengambil data yang akan diexport pada excel. ini dapat select all, hasil filter where atau query eloquent lainnya
    2. **map($item)** : menentukan data yang dikirim pada column-column excel, pada method ini diperbolehkan menulis logika php apapun yang relefan
    3. **headings()** : menentukan nama-nama column header pada excel nantinya
- Jangan lupa untuk import class-class yang diperlukan ⬇️
    a. use Maatwebsite\Excel\Concerns\FromCollection; ➡️ default
    b. use Maatwebsite\Excel\Concerns\WithHeadings; ➡️ menggunakan header
    c. use Maatwebsite\Excel\Concerns\WithMapping; ➡️ menggunakan function map

Contoh :
```
public function collection()
{
        return NamaModel::select('column1','column2')->get();
}

Public function map($item) : array
{
        return [
                $item->column_1,
                $item->column_2,
        ];
}
```

```
public function headings(): array
{
        return [
                'Column 1',
                'Column 2'
        ];
}
```

- Membuat **button** dengan **tag a** yang akan **trigger method pada controller** yang memproses download data ⬇️

```
<a href="{{ route('export.excel') }}">Export to Excel</a>
```

Jangan lupa untuk mendefinisikan **route** nya ⬇️

```
Route::get('/export/excel', [NamaController::class, 'createExcel'])->name('export.excel');
```

- Untuk menggunakannya dapat **import konstanta** ⬇️
    1. untuk memanggil file proses export ➡️ `use App\Exports\NamaMigrationExport;`
    2. untuk menggunakan class excel ➡️ `use Excel;`
- Membuat **method tambahan** pada **controller** ⬇️

```
public function createExcel()
{
        $file_name = 'nama_file_yang_diinginkan'.'.xlsx';
        return Excel::download(new NamaMigrationExport, $file_name);
}
```

Apabila ekstensi yang ingin di download merupakan **file csv** maka pada **$file_name** dapat diganti dengan **'.csv'**

20. Fitur **search** : pencarian dengan form method get
    - Menyediakan input search pada blade terkait, dan gunakan method **GET** sesuai dengan method route yang digunakan untuk mengakses halaman ini

```
<form action="" method="GET">
        @csrf
        <input type="text" name="search" placeholder="Cari melalui nama…" >
        <button type="submit">Cari</button>
        <a href="{{route('name')}}">Refresh</a>
</form>
```

    - Pada method controller yang menampilkan halaman (data) tempat adanya input search tersebut tambahkan parameter ➡️ `index(Request $request)`
    - Ubah proses get data pada model eloquent dengan where()

```
$data = NamaModel::where('column', 'LIKE', '%' . $request->nameInput . '%')->get();
```

➡️ **get()** dapat diganti dengan **paginate()** atau **simplePaginate()** ataupun eloquent tamabahan lainnya sesuai kebutuhan.

21. Wa Link : menyambungkan href dengan whatsapp web
    - Menambahkan tag **a href** pada file **blade**
    - Memastikan value dari data yang berperan sebagai nomor telp (misal column : no_telp) berisi nomor telepon yang menggunakan format berdasarkan negaranya namun tanpa tanda (+), contoh :
      081234567890 ❎
      6281234567890 ✅
    - Untuk dapat mengubah format 08 menjadi 628 tersebut, dapat menggunakan cara berikut :
      `substr_replace($data->no_telp, "62", 0, 1)`
    - Pada attribute href dapat diisi dengan `href="https://wa.me/6281234567890"`
    - Jika ingin mengirim pesan beserta isi text defaultnya, maka attribute href dapat diisi dengan `href="https://wa.me/6281234567890?text=[message-url-encoded]"`
      Contoh message url encoded : **text=Hello%20Fema**
    - Untuk **referensi URL encoding** dapat dilihat pada laman berikut :
      https://www.w3schools.com/tags/ref_urlencode.ASP

22. Relations Table : menyambungkan beberapa table dengan foreign dan primary key
    - Membuat fungsi relations pada model. Untuk model yang berperan sebagai primary key (penghubung), simpan fungsi :

```
public function nama()
{
        return $this->hasMany(NamaModelYangDihubungkan::class);
}
```

      ➡️ nama fungsi dapat disesuaikan dengan tipe relasinya. Untuk hasOne (1 to 1) dapat menggunakan nama migration tanpa “s” dan untuk tipe hasMany (1 to many) dapat menggunakan nama migration dengan “s”
      ➡️ pastikan pada column migration yang dihubungkan, nama foreign key nya merupakan nama table penghubung (yg menyimpan primary key) tanpa “s” dan ditambahkan _id

      Pada model yang dihubungkan (foreign key), tambahkan fungsi berikut :

```
public function nama()
{
        return $this->belongsTo(NamaModelPenghubung::class);
}
```

      ➡️ nama fungsi disamakan dengan nama fungsi pada model penghubungnya

    - Pada eloquent model controller nya, fungsi relasi tersebut dapat dipanggil dengan cara :
      `NamaModel::with('nama_fungsi_dimodel')->get();`

➡️ setelah **with()** dapat juga dipanggil eloquent lain seperti where, orderBy, dan lain-lain

- Untuk pemanggilan column pada table yang direlasikan dapat dilakukan dengan cara berikut : `$item['nama_fungsi']['column']`