CHEATSHEET WEBPROG

By : WinterRide

TABLE OF CONTENTS

Kisi - Kisi UAS

- Materi Sebelum UTS (Semua)
- Localization
- Session
- Middleware
- Authentication

Cache

Kadang kalo ada config yang kalian ubah itu laravel ga langsung bisa apply perubahanya, hal ini dikarenakan adanya “caching” atau semacam fitur nyimpen konfigurasi supaya si laravel enggak request untuk ngecek config kita.

Berikut beberapa command yang bisa kalian apply kalo seandainya ada perubahan yang kalian lakuin trus kesannya si laravel malah tetep gak apply perubahannya :

1. **php artisan route:clear** => Clear Route Cache
2. **php artisan cache:clear** => Clear Application Cache

Bootstrap

1. (File Laravel 11 & Bootstrap 5 sudah ada pada PC lab di local disk E/Project)
2. Copy Folder Laravel sama bootstrap ke D:
3. Pindahin bootstrap ke Folder public di Laravel
4. Taruh
   // Pada Head
   <link rel="stylesheet" href="/bootstrap-5.0.2-dist/css/bootstrap.css">
   // Pada Body
   <script src="/bootstrap-5.0.2-dist/js/bootstrap.js"></script>

Laravel install : **composer create-project --prefer-dist laravel/laravel namaFile**

Stopping process in terminal : **CTRL + C**

Materi Sebelum UTS

Beberapa materi yang muncul di UTS berarti perlu dipelajarin kembali, berikut materi"nya :

- Search Logic
- Pagination

Migration Stuff

Langkah / practice terbaik untuk melakukan migrasi adalah dengan langsung menyambungkannya ke Model. Bisa paka command ini :

"php artisan make:model Post -mf"

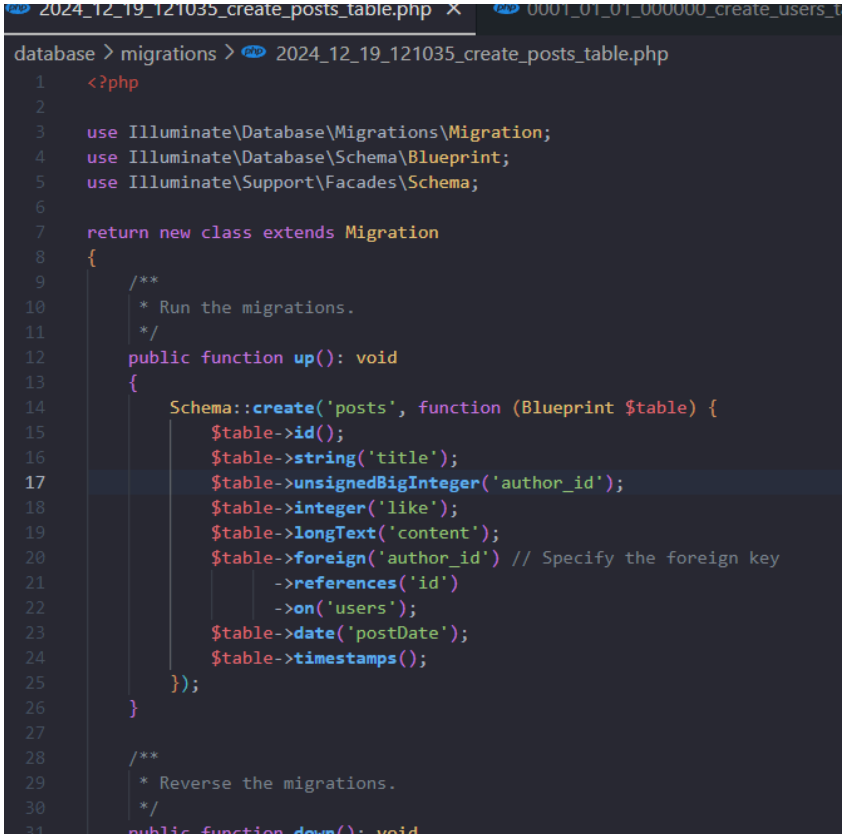
- **m** : Untuk secara otomatis membuat migration di folder database yang sudah tersambung ke model.
- **f** : Untuk secara otomatis membuat file factory di /database/factories untuk melakukan hal - hal yang berhubungan dengan faker().

"**php artisan migrate"** : Untuk melakukan migrasi table beserta kolomnya ke database.
"**php artisan migrate:rollback**" : Untuk mengembalikan migrasi table beserta kolomnya ke database. (Biasa dipakai kalau kita kelupaan perlu nambahin kolom atau salah nama kolom, maka kita bisa pakai ini untuk mengembalikan state database ke null dan melakukan migrasi secara fresh)
"**php artisan migrate:fresh"** : Untuk melakukan migrasi table beserta kolomnya secara paksa ke database.

- (Kalau mau lebih cepet bisa pakai ini, tapi ingat command ini ibarat seperti overwrite database secara paksa dengan konfigurasi migrasi yang baru, jadi kalau misalkan databasenya jadi kacau bisa pakai cara manual (rollback dulu baru migrate ulang))
- (Kalau masih rusak mendingan delete database di xampp dan buat database baru, kemudian bisa kita migrate ulang)

"**php artisan migrate:fresh - -seed"** : Untuk memasukkan data dummy dari file seeder yang sudah kita buat.

1. **Migrations**



- Setelah mengesekusi **“php artisan make:model Post -mf”** maka akan terbentuk file migration secara otomatis.
- Untuk menambahkan kolom baru bisa dengan menambahkan **$table->{dataType}({columnName})**.
- Ingat usahakan nama kolom atau konten data dari sebuah migration harus reflection dari tujuan migration. (Contoh : Migration untuk table “Post” harus berisi kolom - kolom yang berhubungan dengan Post, jangan campur sama kategori data lain (contoh : username)) (Bisa saja ada pengurangan nilai)
- Jika memang data table tersebut memiliki hubungan kepunyaan, maka kita bisa melakukan **FOREIGN KEY** :
    - Seperti yang kita ketahui untuk mengidentifikasi data atau membedakan satu data dengan data lain kita menggunakan sebuah “id”. Id ini lah yang dinamakan sebagai **PRIMARY KEY** yang bertugas sebagai kunci utama dari sebuah data.
    - Ada pula yang dinamakan **FOREIGN KEY**, foreign key sendiri sesuai dengan namanya “kunci asing / kunci lain” yang artinya kunci tersebut mengarah ke **PRIMARY KEY TABLE LAIN**.
    - Dengan mendefinisikan foreign key, kita bisa tetap keep track data “Post” ini milik username siapa.
    - Cara mendefinisikan foreign key kita harus buat dulu kolom yang menyimpan foreign key tersebut dengan **$table->unsignedBigInteger(‘column_name’)**.

- Setelah itu kita buatlah relasi di file migration yang sama dengan kode berikut :

  $table->foreign('author_id') // Foreign key yang ingin direlasikan
  ->references('id') // Nama kolom primary key dari table yang relasikan
  ->on('users'); // Nama table yang ingin relasikan (lihat file migration)

- Asumsi diatas adalah kita ingin memberikan statement bahwa sebuah “Post” merupakan postingan dari sebuah user. Pada website akan ditampilkan konten postingan beserta siapa yang melakukan postingan. Permasalahannya nama user berada di table “User” bukan table “Post”. Maka disini kita bisa mengakses nama user tersebut melalui id user. Id dari user akan kita gunakan untuk mencari collection dari database. Setelah mendapatkannya kita bisa mengakses $user->name untuk mendapatkan username dari user.

- Jenis - jenis relasi : *(Sebenernya ga cuman ini tapi rata - rata kita pake ini)*
  - **One to one**

    Relation “One to one” berarti hanya ada 1 collection merupakan kepunyaan dari 1 collection. Sebagai contoh sebuah postingan di media sosial pasti hanya di posting dan dimiliki oleh 1 user.

```php
class Post extends Model
{
    /** @use HasFactory<\Database\Factories\PostFactory> */
    protected $fillable = ['title1', 'like', 'content', 'author_id', 'postDate'];

    public function author() {
        return $this->belongsTo(User::class);
    }

    use HasFactory;
}
```

    1. Setelah kita migrasi foreign keynya, maka step selanjutnya adalah mendefinisikan relasinya di model.
    2. Pertama, buat function yang akan digunakan untuk memanggil hasil relasi
    3. Kedua, lakukan return $this->belongsTo({model_namel}::class). Kode ini berarti $this(Post) merupakan kepunyaan dari class model “Post”
    4. Selanjutnya bisa kita gunakan relasinya seperti dibawah :

```blade
<p>Author : {{ $post->author->name }}</p>
```

       Disini $post sudah didapatkan dari controller dan karena sudah mendefinisikan relasi dengan table User, maka kita bisa mengakses username melalui foreign key yang sebelumya kita sudah migrasi. Cara memanggilnya kita gunakan nama function

yang sebelumnya kita buat di model dan kita bisa mengakses seluruh attribute yang dimiliki dari class relasi tersebut.

- **One to many**

Relation “One to many” berarti 1 collection memiliki banyak collection. Sebagai contoh 1 user bisa memiliki banyak postingan.

```
11  class User extends Authenticatable
12  {
13
14      public function posts() : hasMany {
15          return $this->hasMany(Post::class, 'author_id');
16      }
```

1. Sama seperti diatas, setelah migrasi kita buat function yang akan digunakan untuk memanggil hasil relasi. Untuk relasi yang kita mau itu “Satu user bisa membuat banyak post” jadinya kita pakai relasi “One to Many”.
2. Disini untuk relasi one to many kita harus import dulu data type “hasMany”

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

3. Kemudian lakukan return $this->hasMany({model_namel}::class, {column_name}). Disini ada tambahan kita harus specify column_name, karena :
    - Kita tahu bahwa penanda sebuah postingan milik user tersebut adalah mencocokan user **id** dengan **author_id** yang disimpan pada migration “Post”.
    - Masalahnya secara otomatis dia cocokinnya pakai **migrationTableName_id** alias pakainya **users_id**.
    - Disini kita harus kasih tahu bahwa yang perlu dicocokin antara id dari user dengan foreign key / id yang disimpen di “Post” itu menggunakan nama kolom “**author_id**”.

4. Berbeda dengan relasi one to one dimana value yang direturn oleh relasi pasti satu, dikarenakan hubungan relasi 1:1 (1 Post dimiliki oleh 1 User). Relasi “One To Many” akan mengembalikan value array karena kita ingin mengembalikan hasil relasi dari “Satu user memiliki banyak post”.

    Berikut cara pakai relasinya untuk mengembalikan semua post yang dimiliki oleh user tersebut :

    - **Cara tanpa relasi**

```php
public function viewYourPost(){
    $userId = 1;
    $post = Post::where('author_id', $userId)->get();
    return view('yourPost', ['dataPost' => $post]);
}
```

resources > views > yourPost.blade.php

```blade
@extends('layouts.app')

@section('content')
<h1>Your Posts</h1>
<div style="display: flex; flex-direction: column; width: full; height: full">
    @foreach($dataPost as $item)
    <div class="card" style="width: full;">
        <div class="card-body">
            <h5 class="card-title">{{ $item->title }}</h5>
            <p class="card-text">{{ $item->content }}/p>
            <br>
            <a href="{{ route('post.view', ['id' => $item->id])}}" class="btn btn-primary">View Details</a>
        </div>
    </div>
    @endforeach
</div>


@endsection
```

Kalau tanpa relasi kita harus melakukan query dengan mencari postingan dimana author_id nya sama dengan user id yang kita inginkan. (Ini cara terpaksa sih, kek kalau lupa cara bikin relasi ya kita pake cara manual kayak begini)

Kalau dah ketemu dia bakal return semua data postingan yang berhubungan sama user id tersebut dan kita bisa akses datanya di page menggunakan variable $dataPost.

- **Cara dengan relasi**

```php
public function viewYourPost(){
    // $userId = 1;
    // $post = Post::where('author_id', $userId)->get();
    // return view('yourPost', ['dataPost' => $post]);

    $user = User::findOrFail(1);
    return view('yourPost', ['user' => $user]);
}
```

Bedanya kalau kita pake relasi, kita gak perlu melakukan query kembali karena sebelumnya kita sudah mendefinisikan relasi di file model “User”. Relasi yang kita definisikan waktu itu adalah “Satu user memiliki banyak post”.

Maka dari itu kita cukup melakukan return collection dari User ke front page dan nantinya bisa langsung kita pakai relasinya dengan function yang sebelumnya sudah kita buat.

resources > views > yourPost.blade.php

```
@extends('layouts.app')

@section('content')
<h1>Your Posts</h1>
<div style="display: flex; flex-direction: column; width: full; height: full">
    @foreach($user->posts as $item)
    <div class="card" style="width: full;">
        <div class="card-body">
            <h5 class="card-title">{{ $item->title }}</h5>
            <p class="card-text">{{ $item->content }}</p>
            <br>
            <a href="{{ route('post.view', ['id' => $item->id])}}" class="btn btn-primary">View Details</a>
        </div>
    </div>
    @endforeach
</div>

@endsection
```

Bisa dilihat yang sebelumnya kita pakai $dataPost yang didapatkan melalui query di controller untuk mengembalikan semua data postingan terkait user tersebut.

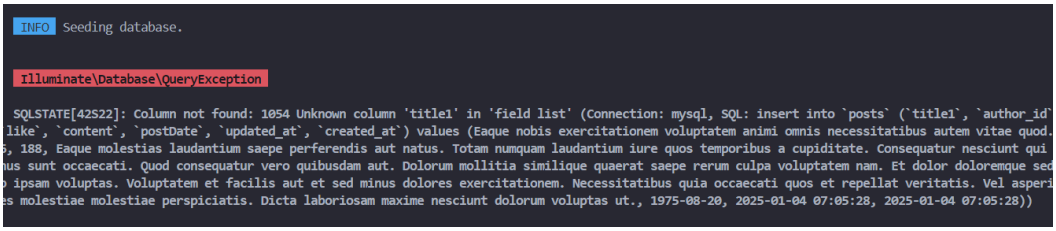
Kalau pakai relasi kita cukup pakai $user yang kita send dari controller dan untuk mengembalikan semua postingan milik user tersebut bisa dengan **$user->posts**.

(Kita pakai function posts bisa dilihat saat mendefinisikan relasi “One to Many”. Jadi ya kita panggil sesuai nama function yang kita buat di model tersebut.)

2. Faker

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Post>
 */
class PostFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition(): array
    {
        return [
            'title' => $this->faker->sentence(mt_rand(5, 10)),
            'author_id' => mt_rand(1, 10),
            'like' => mt_rand(100, 1000),
            'content' => $this->faker->paragraph(mt_rand(5, 10)),
            'postDate' => $this->faker->date()
        ];
    }
}
```

- Untuk memberikan data dummy bisa modifikasi di bagian function definition()
- Ingat penamaan kolom harus sesuai dengan nama kolom yang sudah di migrasi ke database (bisa lihat file migration), jika tidak maka akan muncul error seperti ini :



(Ini asumsi kita melakukan seed column “title1” yang pada database itu gaada kolom yang namanya “title1”)

DatabaseSeeder.php | PostFactory.php | home.blade.php

database > seeders > DatabaseSeeder.php

```php
<?php

namespace Database\Seeders;

use App\Models\User;
use App\Models\Post;
// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        User::factory(10)->create();
        Post::factory(10)->create();

        User::factory()->create([
            'name' => 'Test User',
            'email' => 'test@example.com',
        ]);

        Post::factory()->create([
            'title' => 'FoodBlogger',
            'author_id' => 1,
        ]);
    }
}
```

- Setelah mengisi kolom - kolom yang ingin di faker(), kita bisa eksekusi dengan 2 cara :
    - **Post::factory()->create([])**

      Sebelumnya kita sudah mengisi kolom ‘title’ dan ‘author_id’ dengan faker() data, tapi misalkan kita ingin mengubah bagian tersebut menjadi data yang kita mau bisa menggunakan command ini.

    - **Post::factory(10)->create()**

      Kolom - kolom yang sudah kita kode di file factory dapat diesekusi mengg unakan command ini. Jumlah datanya juga bisa kita sesuaikan. (Pastikan jumlah kolom serta nama kolom tidak kurang atau berbeda dari database) (Cek bagian factory kalau ada error yang berhubungan sama “column “title” doesnt have a default value”)

- Langkah terakhir bisa kita masukan data faker() ke database dengan command “**php artisan migrate:fresh - -seed**”

Form Processing, Validation, & CRUD

Untuk form kalau lupa kita bisa langsung copy-paste layout form bawaan docs bootstrap

```
<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp">
    <div id="emailHelp" class="form-text">We'll never share your email with anyone else.</div>
  </div>
  <div class="mb-3">
    <label for="exampleInputPassword1" class="form-label">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1">
  </div>
  <div class="mb-3 form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

1. **Form Validation**

```
<div class="mb-3">
    <label for="title" class="form-label">{{ __('editPost.title') }}</label>
    <input type="text" class="form-control" id="title" name="title" value="{{ $post->title }}">
    @error('title')
        <span style="color: red;">{{ $message }}</span>
    @enderror
</div>
```

- Yang perlu diperhatikan adalah attribute “name” pada tag <input>, pemberian nama kepada attribute “name” adalah sebagai acuan bagaimana data tersebut akan direferensikan di controller.

```
public function store(Request $request){
    $data = $request->validate([
        'title' => 'required|min:8|max:30',
        'date' => 'required|date|after_or_equal:today'
        'content' => 'required|max:255|min:10'
    ]);
```

- Untuk mengambil data yang diinput oleh user adalah dengan menambahkan Request $request pada parameter function yang digunakan untuk memasukkan data ke database.
- Bisa dilihat pada bagian validasi nama field yang dipanggil oleh controller sesuai dengan penamaan attribute “name” pada .blade.php. (Perhatikan bagian ini jika ketika sudah menekan tombol “submit” data tidak tervalidasi atau data tidak masuk ke database).
- Untuk menampilkan error / hasil validasi kita bisa menambahkan :
  @error(‘nama field yang divalidasi’)
  {{ $message }}
  @enderror

- Jika validasi masih salah, maka secara otomatis system tidak akan mengizinkan return view tereksekusi. (Jadi misalkan disuruh ketika sudah berhasil input data maka akan diarahkan ke page lain, maka lakukan return view() seperti biasa).

2. **Data Insertion Form**

resources > views > insertPost.blade.php

```
@extends('layouts.app')

@section('content')
<h1>Insert Post</h1>
<form action="{{ route('post.store')}}" method="POST">
    @csrf
    @method('POST')
    <div class="mb-3">
        <label for="title" class="form-label">Title</label>
        <input type="text" class="form-control" id="title" name="title" value="{{ old('title')}}">
        @error('title')
            <span style="color: red;">{{ $message }}</span>
        @enderror
    </div>
    <div class="mb-3">
        <label for="date" class="form-label">Date</label>
        <input type="date" class="form-control" id="date" name="date" value="{{ old('date')}}">
        @error('date')
            <span style="color: red;">{{ $message }}</span>
        @enderror
    </div>
    <div class="mb-3">
        <label for="content" class="form-label">Content</label>
        <input type="text" class="form-control" id="content" name="content" value="{{ old('content')}}">
        @error('content')
            <span style="color: red;">{{ $message }}</span>
        @enderror
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>

@endsection
```

```
Route::post('/insertPost', [PostController::class, 'store'])->name('post.store');
```

- Gunakan method “**POST**” karena dengan asumsi kita ingin memasukkan data ke database
- @csrf dan @method(‘POST’) tetap diperlukan karena merupakan bagian dari form processing laravel demi menjaga integritas data.
- Pada route jangan lupa gunakan **Route::post**
- Contoh : <label **for=”1”**> adalah untuk memberikan label kepada input dengan **id=”1”**
- Yang perlu diperhatikan adalah attribute “name” pada tag <input>, pemberian nama kepada attribute “name” adalah sebagai acuan bagaimana data tersebut akan direferensikan di controller.

```php
public function store(Request $request){
    $data = $request->validate([
        'title' => 'required|min:8|max:30',
        'date' => 'required|date|after_or_equal:today',
        'content' => 'required|max:255|min:10'
    ]);

    Post::create([
        'title' => $data['title'],
        'content' => $data['content'],
        'like' => 0,
        'author_id' => 1,
        'postDate' => $data['date']
    ]);

    return view('insertPost');
}
```

- Untuk mengambil data yang diinput oleh user adalah dengan menambahkan Request $request pada parameter function yang digunakan untuk memasukkan data ke database.
- Bisa dilihat pada bagian validasi nama field yang dipanggil oleh controller sesuai dengan penamaan attribute “name” pada .blade.php. (Perhatikan bagian ini jika ketika sudah menekan tombol “submit” data tidak tervalidasi atau data tidak masuk ke database).

Models > Post.php

```php
<?php

namespace App\Models;

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

class Post extends Model
{
    /** @use HasFactory<\Database\Factories\PostFactory> */
    protected $fillable = ['title', 'like', 'content', 'author_id', 'postDate'];

    public function author() {
        return $this->belongsTo(User::class);
    }
    use HasFactory;
}
```

- Pada model Post **WAJIB** membuat $fillable property yang akan digunakan untuk memberikan nilai kepada attribute dari sebuah class / model. (Perlu diingat bahwa fillable property yang diisi adalah sesuai dengan table column name yang sudah di migrasi ke database, jika penamaan tidak sesuai maka akan muncul error dibawah). (Asumsi salah menulis ‘title’ menjadi ‘title1’ sehingga nama kolom gagal dibaca oleh database)

```php
protected $fillable = ['title1', 'like', 'content', 'author_id', 'postDate'];
```

- Untuk memasukkan data kita perlu memanggil model Post::create dan memasukkan data secara manual SESUAI DENGAN $fillable PROPERTY. (Perhatikan bagian ini jika data yang dicoba masukan ada yang rusak / kosong, hal itu bisa saja karena fillable property yang dipanggil di Post::create berbeda penamaannya dengan yang ada di model Post).

3. Data Update / Edit Form

resources > views > editPost.blade.php

```
<h1>{{ __('editPost.header') }}</h1>
<form action="{{ route('post.update', $post->id)}}" method="POST">
    @csrf
    @method('PUT')
    <div class="mb-3">
        <label for="title" class="form-label">{{ __('editPost.title') }}</label>
        <input type="text" class="form-control" id="title" name="title" value="{{ $post->title }}">
        @error('title')
            <span style="color: red;">{{ $message }}</span>
        @enderror
    </div>
    <div class="mb-3">
        <label for="date" class="form-label">{{ __('editPost.date') }}</label>
        <input type="date" class="form-control" id="date" name="date" value="{{ $post->postDate }}">
        @error('date')
            <span style="color: red;">{{ $message }}</span>
        @enderror
    </div>
    <div class="mb-3">
        <label for="content" class="form-label">{{ __('editPost.content') }}</label>
        <input type="text" class="form-control" id="content" name="content" value="{{ $post->content }}">
        @error('content')
            <span style="color: red;">{{ $message }}</span>
        @enderror
    </div>
    <button type="submit" class="btn btn-primary">{{ __('editPost.button') }}</button>
</form>
<a href="{{ route('post.viewEditId', $post->id) }}">View form in indonesian</a>
<a href="{{ route('post.viewEdit', $post->id) }}">View form in english</a>
@endsection
```

```
Route::put('/editPost/{id}', [PostController::class, 'update'])->name('post.update');
```

- Tetap gunakan method “**POST**” pada tag form karena dengan asumsi kita ingin mengirim data baru ke database.
- Meletakkan @csrf dan @method(‘PUT’) pada sebelum input dan gunakan **‘PUT’ pada @method** karena sesuai dengan kamus.
- Pada route jangan lupa gunakan **Route::put**

- Untuk memberikan konten dari data yang ingin diupdate bisa menambahkan attribute **value="{{ $variable->attribute }}"** pada tag input. (Opsional tapi bisa aja disuruh)
- Sekali lagi perhatikan attribute "name" pada tag <input>, pemberian nama kepada attribute "name" adalah sebagai acuan bagaimana data tersebut akan direferensikan di controller.

```php
public function update(Request $request, $id){
    $data = $request->validate([
        'title' => 'required|min:8|max:30',
        'date' => 'required|date|after_or_equal:today',
        'content' => 'required|max:255|min:10'
    ]);

    $post = Post::findOrFail($id);
    $post->update($data);

    return redirect()->route('post.viewEdit', $post->id);
}
```

- Sama seperti konsep memasukkan / insert data, kita bisa melakukan validasi kembali data baru yang ingin di update ke database.
- Setelah validasi kita ingin memasukkan data yang sudah tervalidasi ke database dengan cara memanggil collection atau model Post sesuai dengan post id yang ingin di update. Data tersebut disimpan ke dalam 1 variable ($post).
- Setelah memanggil data, kita bisa mengeksekusi function update() dengan cara melakukan $post->update("data yang sudah divalidasi").

  (Perhatikan $fillable pada model jika data yang dicoba masukan ada yang rusak / kosong, hal itu bisa saja karena fillable property yang divalidasi berbeda penamaannya dengan yang ada di model Post).

4. **Data deletion form**

```blade
<form action="{{ route('post.delete', ['id' => $post->id]) }}" method="POST">
    @csrf
    @method('DELETE')
    <a href="">
        <button>
            Delete Post
        </button>
    </a>
</form>
```

```php
Route::delete('/post/{id}', [PostController::class, 'delete'])->name('post.delete');
```

- Tetap gunakan method "**POST**" pada tag form karena dengan asumsi kita ingin mengirim suatu instruksi baru (penghapusan data) ke database.

- Meletakkan @csrf dan @method('PUT') pada sebelum input dan gunakan **'DELETE' pada @method** karena sesuai dengan kamus.
- Pada route jangan lupa gunakan **Route::delete**
- Untuk data deletion, kita hanya perlu mendapatkan data yang diiinginkan dari database dan langsung bisa kita delete() datanya. Dalam kebanyakan kasus kita akan mencari data menggunakan **primary key / id** dari data ($post->id).

```php
public function delete($id){
    $post = Post::findOrFail($id);
    $post->delete();

    $postData = Post::all();
    return view('home', ['dataPost' => $postData]);
}
```

- Pada controller kita memanggil collection atau model Post sesuai dengan post id yang ingin di update. Data tersebut disimpan ke dalam 1 variable ($post).
- Kemudian, kita bisa mengeksekusi function **delete()** dan data akan langsung dihapus dari database.

Session & Middleware

SESSION sesuai namanya sebuah sesi yang menandakan periode waktu dari user didalam website. Biasanya untuk mengidentifikasi pengguna, Menyimpan preferensi, Mempertahankan status login.

- Biasanya di trigger saat user melakukan login ke website. Session yang dibuat akan menyimpan informasi yang esensial terhadap user dan nantinya akan digunakan untuk kayak misal kalo user rolenya “guest” berarti fitur - fitur yang bisa diakses tuh apa dan kalau user rolenya ”admin” ya bisa akses fiturnya apa aja.

- Cuman mungkin nanti ulangan bakal skenarionya user yang login bisa masukin postingan dan postingan tersebut harus punya koneksi dengan user. Kayak contoh diatas itu kita pakai dummy id kalau usernya masukin sebuah postingan maka kita set author_id = 1.

- Disini kita bakal ngebuat website bisa tahu siapa yang sedang login dan user id berapa yang masukin postingan tanpa kita kasih tahu.

Konsep dari session sendiri untuk menyimpan informasi / preferensi pengguna layaknya global variable yang bisa diakses di file manapun.

Untuk contoh kasus kita lanjutin kayak yang diatas :

```php
public function store(Request $request){
    $data = $request->validate([
        'title' => 'required|min:8|max:30',
        'date' => 'required|date|after_or_equal:today',
        'content' => 'required|max:255|min:10'
    ]);

    Post::create([
        'title' => $data['title'],
        'content' => $data['content'],
        'like' => 0,
        'author_id' => 1,
        'postDate' => $data['date']
    ]);

    return view('insertPost');
}
```

Disini kode untuk insertion datanya itu gak sinkron dengan user yang sedang login, bisa diliat author_id yang di assign selalu 1 untuk semua post yang dimasukin. Kali ini kita bakal assign **author_id** based on **id dari user asli** yang sedang **login** di website.

Berikut langkah untuk memasukkan postingan sesuai user yang sedang login :

- **Siapin frontend untuk login page**

resources > views > auth > login.blade.php

```blade
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="{{ route('login.check')}}" method="POST">
        @csrf
        <div class="mb-3">
            <label for="email" class="form-label">Email address</label>
            <input type="email" class="form-control" id="email" name="email">
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Password</label>
            <input type="password" class="form-control" id="password" name="password">
        </div>
        <button type="submit" class="btn btn-primary">Submit</button>
        <a href="{{ route('register')}}">Dont have an account? Register here</a>
    </form>
</body>
</html>
```

Disini method kita pakai “POST” supaya pas usernya login nanti gaada informasi yang bocor di bagian url tab. (Gambar lebih jelas bisa lihat dibawah)

```
127.0.0.1:8000/login/request?_token=JMpOZMgZyWMD6A3mRhwzuVjeUaQYhJf9RKW4VPnJ&_method=GET&email=user%40gmail.com&password=12345678
```

- **Buat controller untuk handle login sama register**

  Inget ya bisa pakai command seperti ini, jangan nyusahin diri sendiri :v

```
php artisan make:controller SessionController
```

```php
public function viewLogin(){
    return view('auth.login');
}

public function login(Request $request){
    // $data = $request->all();
    $email = $request->input('email'); // Same as $request->get('email');
    $password = $request->input('password');

    $user = User::where('email', $email)->first();
    if ($user && Hash::check($password, $user->password)){
        Session::put('user', $user);

        $postData = Post::all();
        return view('home', ['dataPost' => $postData]);
    }
    return back();
}
```

Penjelasan :

- Disini logikanya kita mau ambil email sama password yang diisi user di login page. Untuk cara ambilnya bisa pakai $request->input('namaField').

  (Perhatiin untuk $request->input() yang diambil itu berdasarkan NAME dari field inputnya)

  ```
  name="email">
  ```

  ```
  name="password">
  ```

- Setelah kita ambil input dari user, kali ini bakal ngecek dulu emailnya ada di database atau enggak pakai **User::where('nama kolom email', $email yang di input)->get();**
- Selanjutnya kita cek passwordnya, bisa dilihat if-elsenya itu ($user && **Hash::check($password, $user->password))**
  - **If ($user)** karena kita pastiin kalau email yang diinput oleh user pas login itu memang ada emailnya. Kalau emailnya ada maka hasil query akan return collection user ke $user. Makanya kita cek apakah $user == true || null.
  - **If (Hash::check($password, $user->password))**, karena password yang kita pengen bandingin itu secara default migrationnya nge-hash atau enkripsi passwordnya. Misalkan "123" dia enkripsi jadi "yey123^#", intinya biar orang gak gampang hacknya. Karena passwordnya masih berupa Hash, maka kita ceknya juga pakai Hash.

    $password = password yang diinput user
    $user->password = password yang tersimpan di database

- Kalau misalkan email dan password yang diinput sesuai sama database, maka kita bakal simpen informasi user ke sebuah **Session**. Supaya sewaktu - waktu kita mau akses **role dari user** atau **id dari user** tinggal panggil aja **Sessionnya** dari file manapun.

  (PENTING)
  Caranya bisa dengan :

  ```
  use Illuminate\Support\Facades\Session;
  ```

  ```
  Session::put('user', $user);
  ```

  'user' => mirip kayak kita mendefinisikan sebuah variable yang namanya 'user'
  '$user' => sebelumnya kita udah ambil user dari database dan kita simpen di $user. Nah $user ini lah yang akan menjadi value dari global variable / session yang kita bikin dengan nama variablenya itu 'user'.

- Skenario lain kalau email dan passwordnya masih salah kita return back() aja atau kalau mau validasi bisa lihat cara di bagian “Form Validation”.

- Setelah buat controllernya kita assign route untuk trigger function loginnya

```
Route::get('/login', [SessionController::class, 'viewLogin'])->name('login.view');
Route::post('/login/request', [SessionController::class, 'login'])->name('login.check');
```

(Untuk trigger function cek login ‘/login/request’ kita tetep pakai Route::post WALAUPUN bukan masukin data ke database, hal ini dikarenakan di login form kita pakainya method=”POST” supaya datanya gak bocor di url tab, makanya disini kita juga sinkron pake Route::post sesuai action method yang dipake di form page)

- Kalau usernya udah login otomatis Session yang berisi informasi $user bakal dibuat. Sekarang kita tes coba masukin sebuah postingan yang bisa tau siapa yang posting

```
public function store(Request $request){
    $data = $request->validate([
        'title' => 'required|min:8|max:30',
        'date' => 'required|date|after_or_equal:today',
        'content' => 'required|max:255|min:10'
    ]);

    $user = Session::get('user'); // Tambahan

    Post::create([
        'title' => $data['title'],
        'content' => $data['content'],
        'like' => 0,
        // 'author_id' => 1,
        'author_id' => $user->id, // Ubah ini
        'postDate' => $data['date']
    ]);

    return view('insertPost');
}
```

Yang sebelumnya kita set ‘author_id’ selalu 1, sekarang kita bakal set berdasarkan user id yang sedang login. Untuk tahu user id yang sedang login bisa kita buka pake **Session**.

(PENTING)
Caranya bisa pakai seperti ini :

```
$user = Session::get('user'); // Tambahan
```

Sebelumnya kita simpen informasi user ke dalam Session yang namanya ‘user’. Maka dari itu kalau mau akses Session tersebut tinggal Session::get(‘user’).

Kalau udah kita fetch data $user dari Session, sekarang kita bakal pake $user->id nya untuk dimasukin ke kolom ‘author_id’ saat membuat postingan baru. Dan selamat sekarang sistem kita udah bisa detect siapa yang membuat postingan.

(PENTING)
Oke masa user udah login gak bisa **LOGOUT**, nah untuk fitur logout bisa caranya seperti ini :

```
public function logout(){
    Session::forget('user');
    return redirect()->route('login.view');
}
```

```
<a href="{{ route('logout')}}">Logout</a>
```

```
Route::get('/logout', [SessionController::class, 'logout'])->name('logout');
```

Sebelumnya kita simpen informasi user ke dalam Session yang namanya ‘user’. Maka dari itu kalau usernya logout anggepan kita bakal hapus data user dari Session, nah caranya itu Session::forget(‘user’).

MIDDLEWARE adalah kelas khusus yang berfungsi untuk memfilter permintaan HTTP dari web browser ke aplikasi kita. Atau dalam arti yang lebih sederhana, middleware adalah kode program yang dapat mengganggu setiap permintaan suatu halaman web.

Long short story sesuai namanya “middle”, dia semacam penengah / observer untuk menginterupsi suatu request jika ada kriteria yang belum terpenuhi. Contoh familiar kalau kalian ke suatu website dan ketika kalian navigasi suatu page dan tiba” diredirect ke Login Page, itulah efek yang diberikan dari middleware.

Untuk lebih jelas, disini kita coba restrict si user kalau belum login maka ga boleh ke page “Insert Post”, karena ya asumsi ya kita harus tau dong siapa yang posting postnya, masa anonymous. Disini middleware nya bakal kita kombinasi sama Session, karena kalau user sudah login otomatis itu Session pasti ada isinya. Jadi kalau gak ada isinya ya balikin aja ke “Login Page”, anggep suruh dia login.

<u>Berikut langkah - langkahnya :</u>

- Pakai command ini untuk membuat middleware

```
php artisan make:middleware AuthMiddleware
```

- Kalau udah berhasil dibuat middlewarenya, maka tampilan default akan seperti ini

app > Http > Middleware > AuthMiddleware.php

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class AuthMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        return $next($request);
    }
}

```

- Next kita isi middlewarenya, konsepnya kita cek **Session[‘user’]** kalau **kosong (null)** berarti kita asumsi si user belum melakukan login. Karena sebelumnya kita buat kalau user sudah input email dan password dan dengan catatan datanya sesuai dengan database, maka Session yang berisi informasi user akan dibuat.

app > Http > Middleware > AuthMiddleware.php

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;
use Illuminate\Support\Facades\Session;

class AuthMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        $user = Session::get('user');
        if ($user == null){
            return redirect()->route('login.view');
        }
        return $next($request);
    }
}
```

Disini kita bikin seandainya Session[‘user’] ternyata null / kosong (belum login), maka kita redirect aja dia ke Login Page. Anggepan suruh login dulu supaya bisa pakai fitur ini gitu.

- Disini middleware yang kita buat adalah “Custom Middleware” dimana system tidak mendeteksi kalau ada middleware ini, maka bisa kita daftarin dulu middlewarenya di file /bootstrap/app.php.

bootstrap > app.php

```php
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;
use App\Http\Middleware\AuthMiddleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        $middleware->web(append:[
            App\Http\Middleware\SetLocale::class,
        ]);
        $middleware->alias([
            'auth' => AuthMiddleware::class
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();
```

Di bagian ->withMiddleware, kita masukin kode :

$middleware->alias([

'auth' => AuthMiddleware::class

]);

‘auth’ => nama middleware yang bakal kita pake di route
‘AuthMiddleware::class’ => file middleware yang kita pake (Jangan lupa import dulu)

- Habis itu middleware yang kita bikin udah bisa kita pakai. Disini kita coba kalau user mau ke bagian page “Insert Post” maka user harus login dulu.

```
Route::get('/insertPost', [PostController::class, 'viewInsert'])->name('post.insert')->middleware('auth');
```

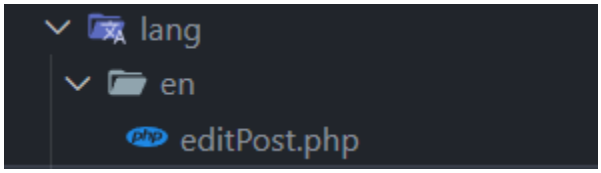
Di route yang kita inginkan kita tambahin lah **->middleware(‘nama alias yang digunakan’)** untuk pakai middleware nya. Sebelumnya kita buat alias **‘auth’** untuk pakai file **AuthMiddleware** kita, maka bisa kita extend routenya dengan **->middleware(‘auth’)**

Penjelasan :
Setelah kita lakuin ini, bisa langsung dicoba seandainya kalian akses lah /insertPost **sebelum login**, maka harusnya akan **diredirect ke “Login Page”**. Sebaliknya kalau **sudah login**, maka dia bakal beralih ke **$next->($request)** dimana akan **menampilkan page “Insert Post”** seperti biasa.

Localization

Localization adalah sebuah feature dimana memungkinkan kita untuk mengembalikan suatu tulisan dengan berbagai bahasa. Yak intinya mirip kalo kita ke website itu ada “Switch language” trus nanti tulisan yang awalnya inggris berubah jadi indonesia semua contohnya begitu.

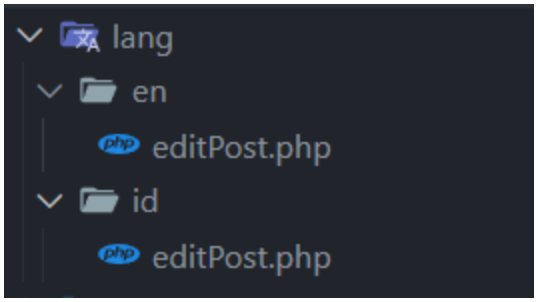
Langkah awal kita command “**php artisan lang:publish**” di terminal. Nanti akan terbentuk folder baru bernama “lang” di project kita.



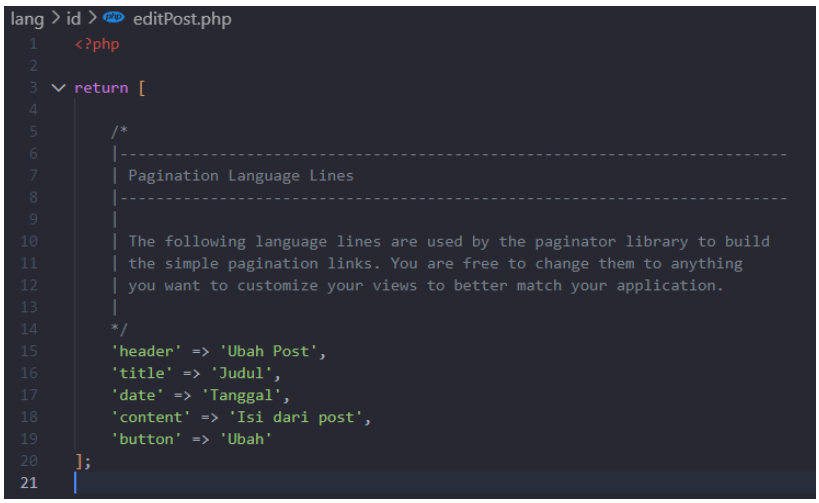
Secara default bakal ada folder “en” yang secara default dia bakal trigger folder ini ketika kita panggil keyword di front page kita.

Secara default kita bakal dikasih banyak file tapi ya sebenernya pake yang perlu” aja, kalo kurang tinggal buat file baru dengan format seperti ini. Intinya file ini sebagai penyedia keyword atau kata kunci untuk menampilkan set of text / string. Untuk format filenya dia bakal return dalam bentuk array yang isinya “KEYWORD” => “VALUE”. Jadi contoh ketika kita pakai keyword “header” maka yang muncul di page adalah “Edit Post”.

Nah seperti yang kita tahu localization itu memungkinkan kita untuk menampilkan page dengan berbagai bahasa tapi ga mungkin kita harus bikin page terpisah khusus tampilin bahasa tersebut. Makanya disini kita pakai keyword. Oleh karena itu, biasanya kita bakal sediain dengan nama file, dan nama keyword yang sama ke folder yang berbeda.



Disini anggeplah kita mau sediain user bisa switch bahasa dari inggris ke indonesia dan sebaliknya di editPostPage. Untuk penamaan disini bebas ga harus sama dengan .blade.php nya. Tapi disini pastiin aja nama file yang dipake di folder yang satu dengan folder yang lain itu sama.



Sekilas kayak file yang sama seperti yang diatas tapi kali ini filenya ada di folder “id”. Bisa diliat, untuk format semua hampir sama persis dengan file yang ada di folder “en” sebelumnya, tapi yang beda adalah value yang di return dari setiap keyword. Sebelumnya di folder “en” textnya dalam bahasa inggris tapi kali ini dalam bahasa indonesia.

Kalau sudah siapin keywordnya, sekarang bisa kita implementasi di website kita.

```
<label for="title" class="form-label">{{ __('editPost.title') }}</label>
```

Untuk pakai keywordnya cukup statement {{ __(‘namaFile.keyword’)}}. Secara default dia bakal pakai folder en karena language dari default html adalah “en”.

Sekarang kita coba bikin triggernya supaya pas user pencet itu bisa berubah dari bahasa inggris ke bahasa indonesia. Untuk cara triggernya bisa pakai 2 cara, ya sesuaikan aja dengan kebutuhan :

- **Local (Sementara)** #CaraYangDiajarinDosen
  Cara ini memungkinkan kita untuk mengubah semua keyword yang default awalnya itu pakai inggris alias pakai folder “en” menjadi bahasa indonesia atau bahasa yang lain. Tapi cara yang ini cuman sementara, yaitu pas kita reload page atau pindah page ya nanti balik lagi dia jadi bahasa inggris. (Ya harusnya soalnya ga bakal suruh sampe bikin

semua page kena trigger bahasa inggris / indonesia, tapi jaga - jaga coba yang global juga ya #SiapaTauKepake)

- Pertama kita buat semacam tombol untuk trigger localizationnya, ya buat tombol seperti biasa yang trigger suatu function di controller kita.

```
<a href="{{ route('post.viewEditId', $post->id) }}">View form in indonesian</a>
<a href="{{ route('post.viewEdit', $post->id) }}">View form in english</a>
```

Hiraukan bypass $post->id nya karena konsepnya di editPostPage butuh id untuk akses data detail dari sebuah post.

```
Route::get('/editPost/{id}/en', [PostController::class, 'viewEdit'])->name('post.viewEdit');
Route::get('/editPost/{id}/id', [PostController::class, 'viewEditId'])->name('post.viewEditId');
```

Buat route seperti biasa untuk trigger function di controller, untuk url pembeda antara indonesia dan inggris kasih /en atau /id aja.

```
public function viewEdit($id){
    App::setLocale('en');
    $post = Post::findOrFail($id);
    return view('editPost', ['post' => $post]);
}

public function viewEditId($id){
    App::setLocale('id');
    $post = Post::findOrFail($id);
    return view('editPost', ['post' => $post]);
}
```

Berikut isi function di controllernya, ya gampang sih tinggal App::setLocale('namaFolder') habis itu bisa kita return view seperti biasa kalau mau tampilin page. (Ingat untuk trigger bahasa itu menggunakan nama folder yang sebelumnya udah kita bikin, bukan alias)

Kalau udah selamat bahasanya berhasil terganti, tapi ya kalo coba direload bakal balik ke inggris lagi, soo..liat cara global atau permanentnya yak kalo mau format bahasa yang dipilih oleh user di apply ke semua page.

- **Global (Permanent)**

Intinya sama, paling yang bedain kalau kita pindah page bahasanya tetep pake yang kita switch sebelumnya.

Jadi kita tau sebelumnya konsep App:setLocale itu gak bertahan secara permanent melainkan ya habis kita trigger ya dia cuman view page tersebut dengan format bahasa yang kita mau. Kalau pindah ke page lain ya berubah lagi dia. Well kalo gitu suruh aja laravelnya eksekusi terus - menerus si App:setLocale di semua page.

(Cara ini gak cocok kalo digabungin sama cara local(sementara), jadi lebih baik pilih salah satu)

- **Disini kita enaknya buat “LocaleController”**
  Inget ya bisa pakai command seperti ini, jangan nyusahin diri sendiri :v

```
php artisan make:controller LocaleController
```

app > Http > Controllers > LocaleController.php

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\App;
use Illuminate\Support\Facades\Session;


class LocaleController extends Controller
{
    public function switchLanguage($locale){
        if (in_array($locale, ['en', 'id'])){
            App::setlocale($locale);
            Session::put('locale', $locale);
        }
        return back();
    }
}

```

  Basically buat function triggernya untuk setLocale seperti biasa, tapi disini ada validasi if-else untuk pastiin parameter yang di bypass di url ya memang tersedia di localization kita. Makanya dicek dulu sebelum di App::setLocale karena bisa aja orang tembak url “/lang/{eu}” dimana “eu” belum kita bikin yang bisa bikin satu sistem kita rusak.

  Disini paling ada tambahan yaitu masukin format bahasa dari user ke **Session**. Hal ini harus dilakukan karena kita mau ngerecord format bahasa yang diinginkan user untuk di apply ke semua page. Jadinya konsepnya kita bisa pakai Session sebagai global variable yang bisa diakses di file manapun. Untuk global variablenya kita set namanya dengan ‘locale’ dan kita isi dengan format bahasa yang diinginkan oleh user.

  Return back() kek semacam reload page dengan data sebelumnya anggepan gaada perubahan karena kita cuman merubah format bahasa aja.

- **Setelah buat function triggernya, sekarang kita buatlah semacam “pendengar” atau “observer” dari perubahan menggunakan Middleware**

Inget ya bisa pakai command seperti ini, jangan nyusahin diri sendiri + kalau pake command ini nanti langsung di auto generate seperti yang dibawah jadi gak usah dihafal.

```
php artisan make:middleware SetLocale
```

app > Http > Middleware > SetLocale.php

```php

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class SetLocale
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        return $next($request);
    }
}

```

Next kita isi middlewarenya

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class SetLocale
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        $locale = Session::get('locale') ?? 'en';
        Session::put('locale', $locale);
        App::setLocale($locale);
        return $next($request);
    }
}
```

Penjelasan :

- $locale = Session::get(‘locale’) ?? ‘en’
  Sebelumnya kita setuju kalo konsepnya itu untuk merecord atau menyimpan format bahasa yang dipilih dari user bakal kita simpen di Session. Well time to take that format dari sessionnya. Kita get(‘locale’) ya karena kita tadi simpen format bahasanya ke dalam variable session yang namanya ‘locale’’. Habis itu ada buntutnya tuh “?? ‘en’”, itu untuk kalau si user belum memilih format bahasa apapun berarti kan asumsi dia pakai “en” yak sesuai defaultnya DAN dia gak trigger penyimpanan format bahasa ke Session.

  Cuman masalahnya itu config default “en” didapetin dari html, dan kali ini yang trigger perubahan bahasa itu sekarang kita maunya pakai Session.

Makanya semisal Sessionnya kosong, kita kasih lah default value “en”. Jadi misal user baru masuk ke website pertama kali dan kalau si user gak melakukan perubahan bahasa apapun alias gaada format bahasa apapun yang disimpen ke Session, tetap bisa memberikan default format bahasa yaitu “en”. (Ya kalau gamau defaultnya “en” bisa ubah ke bahasa yang kita mau)

- Session::put('locale', $locale)
  Habis kita get format bahasa yang baru dari Session kita mau update format bahasa dari global variable ‘locale’ kita dengan format bahasa yang terbaru.

- App:setLocale($locale)
  Habis itu kita trigger lah App:setLocale menggunakan format bahasa yang paling baru seperti biasa.

- Udah buat trigger functionnya sama observernya (middleware), sekarang kita buat si observer ini dijalankan terus menerus selama website kita aktif. File yang perlu kita ubah ada di folder bootstrap/app.php. (Tampilannya kayak dibawah)

bootstrap > app.php

```php
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        //
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();

```

Habis itu daftarin lah seperti dibawah ini

bootstrap > app.php

```php
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        $middleware->web(append:[App\Http\Middleware\SetLocale::class]);
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();

```

Kalo udah tinggal kita buat triggernya sama routenya

```php
Route::get('/lang/{locale}', [LocaleController::class, 'switchLanguage'])->name('switch');
```

```
<h1>{{ __('home.header')}}</h1>

<a href="{{ route('switch', 'id')}}">Switch website language to Indonesian</a>
<a href="{{ route('switch', 'en')}}">Switch website language to English</a>
```

Disini terlihat perbedaan cara sementara dan cara global dimana kita gak perlu define url khusus seperti :

- /home/en
- /home/id

Button action yang didefinisikan juga terdapat value yang kita bypass, yaitu format bahasa yang diinginkan dari user. Sebelumnya kita buat function “switchLanguage” di controller yang menerima parameter $locale. Nah $locale ini lah yang mau kita isi saat trigger route functionnya.

Jadi ketika user pilih button “Switch website language to Indonesian”, maka dia akan trigger function switchLanguage dengan $locale = ‘id’.

Authentication

Authentication yak intinya berhubungan sama login dan register akun. Kesannya kita udah sempet buat diatas pake Session & Middleware, tapi yang kali ini pakai cara cepetnya / cara otomatisnya.

Untuk inti materinya tetep sama, user bakal bisa login dan register, DAN ada middleware pula. Untuk Authentication materi ini sediain 2 cara :

- **Login menggunakan EMAIL & PASSWORD**
  Well login seperti biasa pakai email dan password kayak yang diatas - atas TERNYATA ada provider yang sediain fungsi tersebut. Ya walaupun agak gak guna belajar yang diatas, disini kita diharapkan untuk paham cara kerjanya, jadi gak cuman taunya pake fitur tapi kalo ditanya cara kerjanya malah bingung. Nama package yang menyediakan fungsionalitas ini dinamakan dengan "**Laravel UI**"

- **Login menggunakan SOCIAL MEDIA**
  Bedanya disini kali yak, kita gak cuman bisa login email dan password melainkan bisa login pakai social media seperti Facebook, Twitter, LinkedIn, Google, GitHub, GitLab, and Bitbucket. Nah nama provider yang menyediakan fungsionalitas ini dinamakan dengan "**Laravel Socialite**". (Lihat dokumentasi paling bawah kalo mau coba, karena gak dijelasin disini)

(Jujur kita gak tau nanti ulangan dimintanya suruh pake **Laravel UI** / **Laravel Socialite** ATAU bahkan Authentication yang dimaksud adalah implementasi **Session & Middleware**, jadi lebih baik dipelajarin dua"nya)

(Dua"nya => **Laravel UI** / **Session & Middleware,** yang paling penting dua ini sih karena yang Laravel Socialite itu kita perlu buka web social medianya kan ya dan binus ada block beberapa website yang ga boleh diakses selama ulangan, cuman tetep kalo ada waktu dipelajarin juga Laravel Socialite)

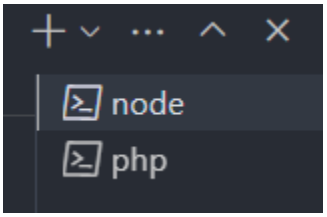
Berikut langkah - langkahnya :
(Lebih enak buat project baru karena bakal ada beberapa file yang di generate secara otomatis biar gak bingung aja)

- **Login menggunakan EMAIL & PASSWORD**
  Ini paling gampang sih basically cuman perlu install - install abis itu dah bisa langsung dipake.
  - Jalanin composer require laravel/ui --dev di terminal
    ```
    composer require laravel/ui --dev
    ```
  - Jalanin php artisan ui bootstrap --auth
    ```
    php artisan ui bootstrap --auth
    ```

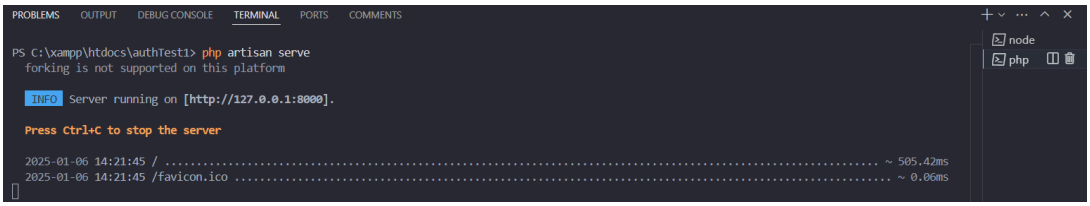
- Jalanin npm install

```
npm i
```

- Kalau udah ya udah sih gitu doang, palingan yang biasanya kita jalanin laravel cuman pake “php artisan serve”, kali ini kita jalanin barengan sama “npm run dev”.



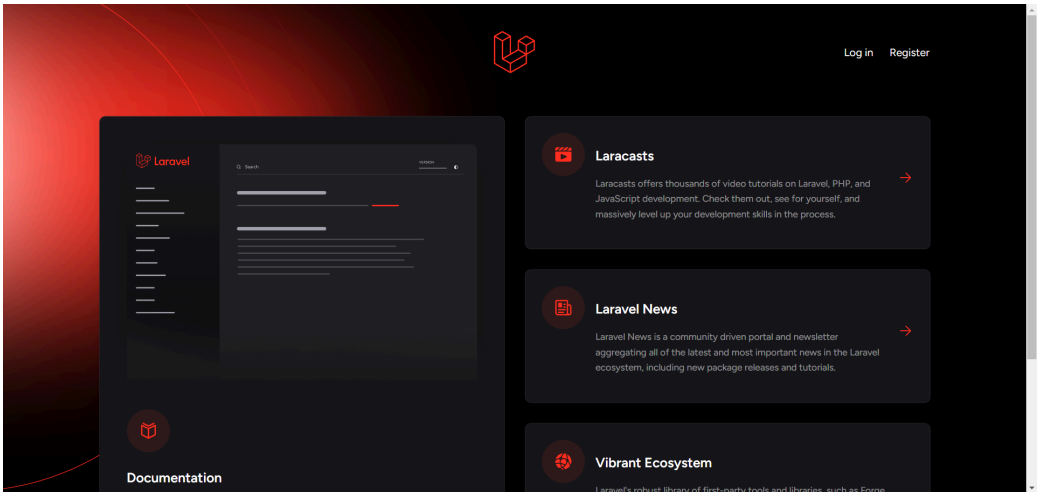
Kalian bisa klik logo “+” untuk nambah terminal lain. Jadi di terminal “**node**” kalian jalanin “**npm run dev**” dan di terminal “**php**” kalian jalanin “**php artisan serve**”



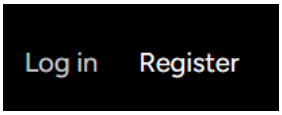
Nah kalau kalian tanya jadi pake url yang dari “**npm run dev**” atau “**php artisan serve**” untuk jalanin websitenya. **TETAP pakai yang dari “php artisan serve” ya manteman.**

(INGAT kalau pake **Laravel UI** kalian jalanin **npm run dev** + **php artisan serve**, otherwise nanti webnya gak jalan)

- Kalau udah di klik URL dari “php artisan serve”, maka tampilannya bakal kayak gini



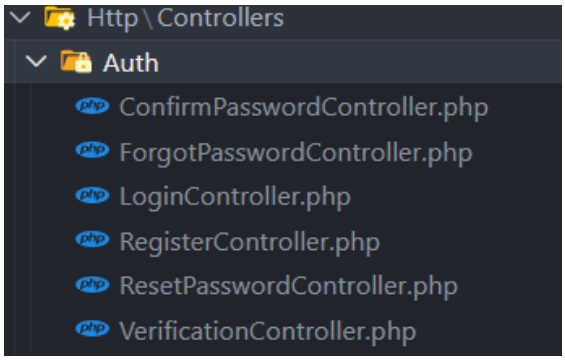
Sekilas mirip tapi paling ada tambahan di bagian pojok kanan atas

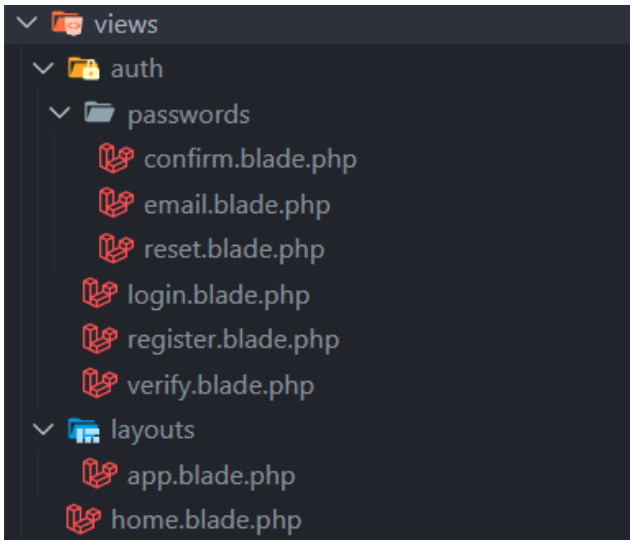


Well yeah disini langsung disediain fungsi untuk registrasi dan login user lengkap sama session, route, .blade.php, dan middlewarenya.

Habis installation kita langsung coba cara pakainya gimana :

Setelah instalasi, ada beberapa file yang otomatis di generate





routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;

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

Auth::routes();

Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');

```

Lengkap ya wkwk, ada Login Page, Register Page, dan Home Page yang bakal bisa kita akses kalau sudah login. Fungsinya 100% sama kayak yang kita pelajarin, basically ketika user sudah melakukan login maka dia bakal membuat suatu Session, dan di page home yang cuman boleh diakses oleh user yang sudah login juga sudah dikasih middleware. Untuk penggunaan middlewarenya sekilas gak keliatan, tapi dia taruh di bagian HomeController. (Untuk nyoba bisa tembak url /home sebelum login, harusnya dia arahin ke Login Page)

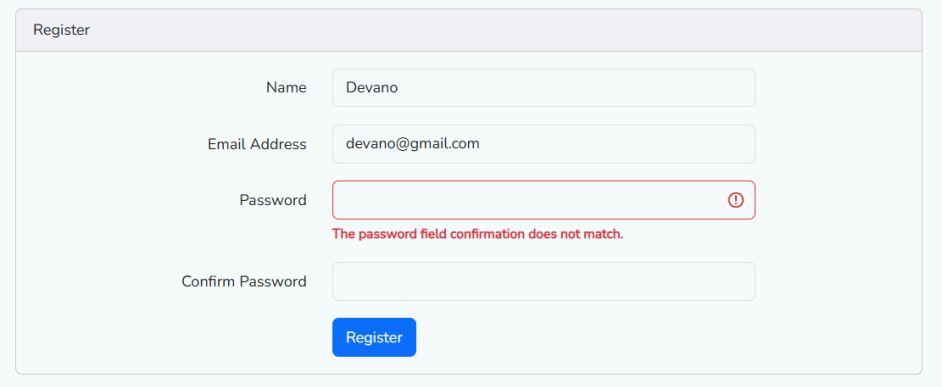
C:\xampp\htdocs\authTest1\.env meController.php

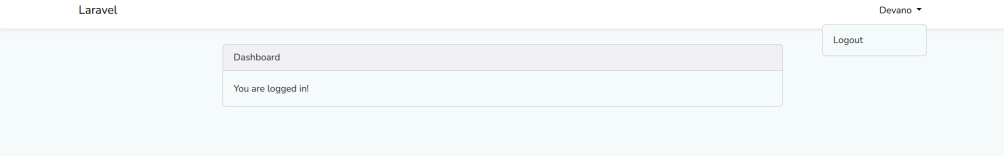
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
    }

    /**
     * Show the application dashboard.
     *
     * @return \Illuminate\Contracts\Support\Renderable
     */
    public function index()
    {
        return view('home');
    }
}
```

Untuk penggunaan middleware sama aja mau di taro di controller atau di route bakal memberikan efek yang sama. Jadi misalkan kita ilangin middleware di controller dan pake middlewarenya di route bakal sama aja (lihat dibawah).

```php
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home')->middleware('auth');
```

Disini kalo kita coba register pun juga udah ada validasinya





Paling yang agak beda adalah gimana cara kita panggil Sessionnya. Lets say kita butuh user role atau user id dari si user untuk masukin sebuah data ke database.

```php
use Illuminate\Support\Facades\Auth;
```

```php
$userId = Auth::id();
```

```php
$user = Auth::user();
```

Kalau mau ambil idnya tinggal Auth::id(). Kalo yang biasanya kita lakuin kan sebelumnya Session::put('user', $user). Misalkan mau ambil keseluruhan data dari user yang udah login tinggal Auth::user(), nanti dia otomatis fetch semua data usernya.

```php
$user = Auth::user();
$userId = $user->id;
```

Habis itu kita bisa pakai kolomnya seperti biasa sesuai dengan apa yang kalian migrate ke database.

resources > views > yourJobs.blade.php

C:\xampp\htdocs\webProgProject\resources

```blade
@if (Auth::user() == null)
    <div class="absolute flex flex-col justify-center items-center w-full h-full text-black gap-5">
        <div class="absolute w-full h-full bg-white opacity-50 pointer-events-none"></div> <!-- Add pointer-events-none -->
        <p class="text-4xl font-bold relative z-10">Login to see this page</p> <!-- Add relative and z-10 to ensure it's above the overlay -->
        <a href="{{ route('login') }}" class="text-white bg-black text-xl rounded-lg p-3 relative z-10">Login</a> <!-- Add relative and z-10 -->
    </div>
@endif
@auth
    <h1 class="text-3xl font-bold mb-5 mt-10">Your Accepted Jobs</h1>
    <div class="flex space-y-4 w-full gap-10">
        @forelse ($acceptedJobs as $job)
        <x-your-job-card :job="$job" />
        @empty
            <p>No jobs found that you've applied for.</p>
        @endforelse
    </div>

    <h1 class="text-3xl font-bold mb-5 mt-10">Your Pending Jobs</h1>
    <div class="space-y-4 w-full gap-10">
        @forelse ($pendingJobs as $job)
        <x-your-job-card :job="$job" />
        @empty
            <p>No jobs found that you've applied for.</p>
        @endforelse
    </div>

    <h1 class="text-3xl font-bold mb-5 mt-10">Job History</h1>
    <div class="space-y-4 w-full gap-10">
        @forelse ($jobHistory as $job)
        <x-your-job-card :job="$job" />
        @empty
            <p>No jobs found that you've applied for.</p>
        @endforelse
    </div>
@endauth
```

Gampang yak, ada beberapa Blade Directives yang bisa kita pake selain @if, @else….

@if(Auth::user() == null) : Kalau mungkin sebelumnya kita literally fetch $user dari controller baru kita cek null atau enggak, ya disini udah bisa langsung pake begini.

@auth….@enduauth : Bagian ini bakal di show kalau usernya udah login

(Cuman kalau kalian gak biasa pake begini atau udah lebih nyaman sama cara **MANUAL**, ya pake cara fetch user di controller baru ngecek ya gapapa, setidaknya kalian ada nunjukkin cara pemakaian yang diminta di materi “Authentication”)

- **Login menggunakan SOCIAL MEDIA**
  Oke bagian ini agak ribet karena di materi PPT nya pun dikit banget dah. Kek instruksinya juga kurang jelas tapi well the point tetep sama bisa login sama register user tapi lewat social media.

  (Disini gw gak jelasin karena high chance gak keluar di ulangan, tapi kalau mau coba bisa cek link ini)
  https://docs.google.com/document/d/1s935ticlRlKxsR9_BKcQwAwS3F9-iDU0/edit?usp=sharing&ouid=103125633616089657445&rtpof=true&sd=true

  Dokumentasinya bukan gw yg bikin, so credit to whoever creates it

  “Anyways thats it, goodluck UASnya”