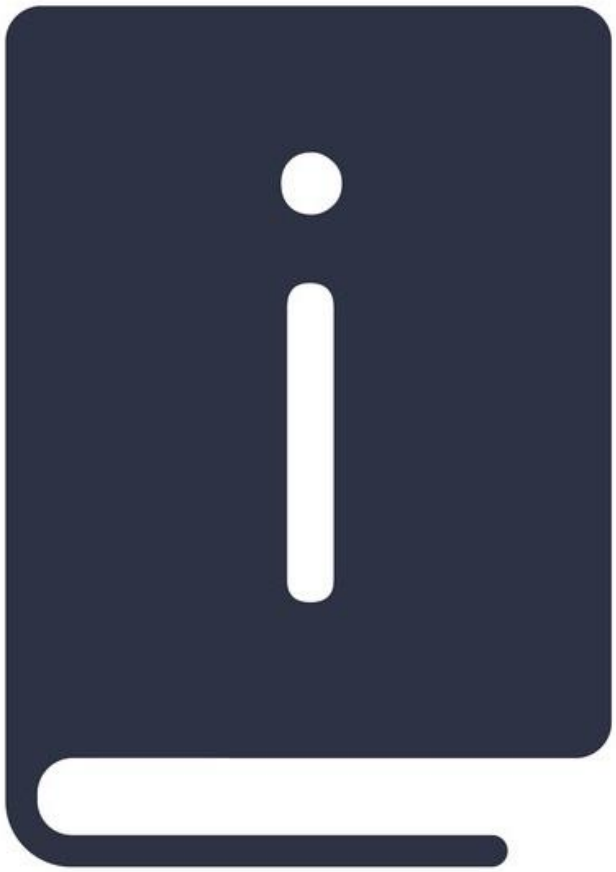


INSTRUCTION

Web App - Laravel

Dokumen Standarisasi Code

TABLE OF CONTENTS

PREPARATION

Dokumen ini merupakan sebuah standar dan panduan bagi backend maupun fullstack developer yang memiliki keinginan untuk mendeploy/hosting aplikasinya di server Politeknik Negeri Indramayu. Dokumen ini dibuat untuk mempermudah developer kami dalam memaintenance aplikasi yang anda buat ketika terjadi error maupun ketika menambahkan fitur-fitur pada aplikasi.

Pada dokumen ini terdapat beberapa sub-instruction yang harus anda terapkan saat mendevelop aplikasi. Beberapa hal yang harus anda ketahui sebelum mendevelop aplikasi, di antaranya :

- Bahasa yang digunakan adalah PHP
- Framework yang digunakan adalah Laravel
- Bagi backend boleh menggunakan Laravel ataupun Lumen
- Versi PHP >= 7.4
- Versi Laravel >= 8.x
- DBMS yang boleh digunakan : Mysql, Postgresql
- Versi DBMS >= 5.7

Untuk hal-hal yang tidak disebutkan di atas, maka developer diberi kebebasan untuk menentukan requirementnya sendiri.

GENREAL STANDARD

A. gitignore

Hal-hal berikut ini seharusnya dimasukkan ke dalam list **.gitignore** dan tidak boleh di push ke dalam repository:

- direktori **vendors**, **node_modules**
- file upload dari user
- file **.env**
- Informasi credential penting atau krusial

B. Variable Naming Conversion

Gunakan penamaan variable atau method yang singkat & jelas, serta tidak membingungkan.

Good: **$user**, **$storeData**, **$debetAccount**

Bad: **$aaaa**, **$name1**, **$thisistoloongvariableyoumaynotseeit**

C. CamelCase

Variable atau method menggunakan format **CamelCase**, yang setiap huruf awalnya Kapital

D. Error message

Error message / debug message hanya boleh ditampilkan pada mode **development** atau **staging**. Pada saat deployment atau tahap production, harap ubah **APP_ENV** menjadi **production** pada file **.env**

E. Sensitive Information

Tidak meletakkan endpoint atau informasi credential yang bersifat private secara hard code di dalam source code. Contoh:

```
protected $secretKey = 'ThisIsN0tSuppOs3dToBeHere';
protected $ProdUrl = 'https://someprivateip/api';
```

Manfaatkan file **.env** untuk menyimpan value sensitif tanpa terekspose di dalam source code.

F. Secure & Verified Code

Source code tidak boleh mengandung **Backdoor** atau shell script yang berbahaya.

Jika menggunakan script dari referensi luar, pastikan script tersebut aman & verified.

G. CSRF Token

Semua form harus menggunakan **CSRF Protection**

H. Dead Code

Tidak boleh ada Dead Code saat merge ke branch **master**. **Dead Code** adalah source code (method, attributes, class) yang sudah tidak digunakan akan tetapi masih tersimpan di dalam codebase dan biasanya hanya dinonaktifkan menggunakan **comment**

overview:

```php
<?php
class Foo {

    public function bar($arg1, $arg2)
    {
        // some code
    }

    //public function deadcode($arg1, $arg2)
    //{
    // some DEAD CODE
    //
    //}
}
```

LARAVEL FRAMEWORK ENGINEERING GUIDELINE

A. Framework Version

- Untuk project yang bersifat longterm disarankan menggunakan framework versi LTS (Long Term Support) atau disesuaikan dengan kebutuhan.
- Jika tidak ada kebutuhan URGENT, dilarang mengupgrade versi framework pada project yang sedang berjalan.

B. DB Transaction

Gunakan fitur **Database Transaction** untuk operasi persistence yang menggunakan lebih dari 1 tabel database. Karena jika tidak menggunakan **Database Transaction**, maka akan ada ambiguitas atau ketidak-jelasan pada database.

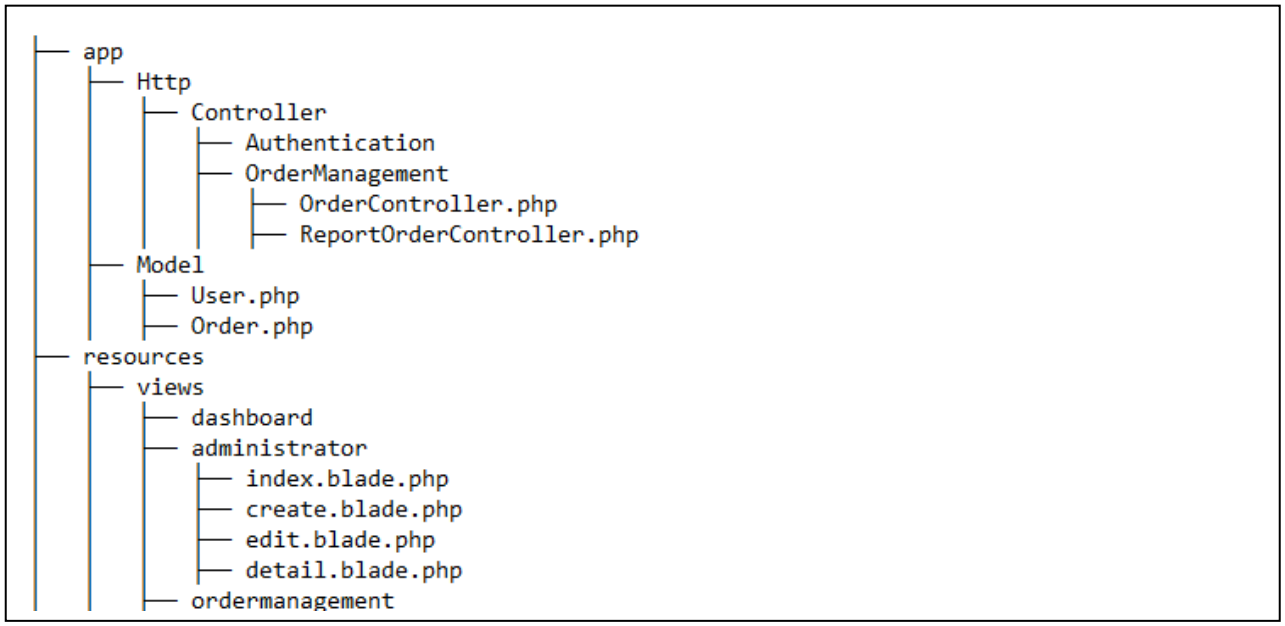
overview:

```php
try {
    DB::beginTransaction();

    // first persistence operation

    // second persistence operation

    // operation success, then commit transaction
    DB::commit();
    return true;
} catch(\Exception $e) {
    // if error happened, rollback transaction
    DB::rollback();
    return false;
}
```

C. File Grouping

Untuk aplikasi yang kompleks, kumpulkan class **Model**, **Controller**, atau **Views** dalam direktori tersendiri sesuai dengan konteks aplikasinya.

simple case:



D. Library

Untuk efisiensi dan efektivitas masa development, gunakan *library* yang sudah umum digunakan dengan kriteria sebagai berikut:

- Library aktif di maintain oleh kontributor / owner
- Sesuai dengan kebutuhan engineering project
- Sesuai dengan development stack yang sedang digunakan
- Penggunaan library harus benar-benar dapat mempermudah proses development

E. Eloquent : Eager Loading

Gunakan **eager loading** untuk optimasi penggunaan relationship di eloquent. Baca implementasi Eager Loading.

F. DB Migration

- Selalu (WAJIB) gunakan **migration** untuk pembuatan atau modifikasi skema database saat development baik itu menambah kolom, edit kolom, hapus kolom, atau modifikasi index.
- Metode ini lebih baik daripada merubah satu file migration lalu menjalankan perintah **php artisan migrate:rollback** lalu **php artisan migrate** lagi. Baca implementasi Migration

G. Try Catch , Report & Throw

- Gunakan blok **try - catch** untuk handling exception terutama di operasi yang berkaitan dengan I/O seperti database, HTTP request, file, service layer.
- Try & Catch berfungsi untuk menangkap **error exception** yang terjadi & memungkinkan aplikasi melakukan aksi tertentu terkait error tersebut.

DON'T

Jangan biarkan technical error muncul / terbaca oleh client app/frontend:

```php
/**
 * This is description of this class
 *
 * @param Request $request
 * @return \Illuminate\Http\JsonResponse
 */
public function register(Request $request)
{
    try {
        $service = $this->applicationService->registerUser($user);
        return response()->json($service);
    } catch(Exception $e) {
        return response()->json(['error' => $e->getMessage()]);
    }
}
```

- **DO**

  - Gunakan Laravel error log untuk menulis detail exception di file **laravel.log**.

  - Dan tampilkan pesan error yang human friendly ke client app / frontend.

```php
/**
 * This is description of this class
 *
 * @param Request $request
 * @return \Illuminate\Http\JsonResponse
 */
public function register(Request $request)
{
    try {
        $service = $this->applicationService->registerUser($user);
        return response()->json($service);
    } catch(Exception $e) {
        return response()->json(['Error Human Language']);
    }
}
```

Baca selengkapnya di **Laravel Logging Documentation**

H. DATA

Semua **data** WAJIB di simpan di **database**, tidak diperbolehkan menyimpan data pada **JSON file**.

CODE STANDARD RULES

A. Migration

Dalam setiap pembuatan, update, drop, maupun modify table, selalu gunakan **Migration** laravel agar setiap perubahan database dapat dimaintain dengan mudah. Namun ada beberapa hal yang perlu diperhatikan dalam pembuatan *table*, diantaranya :

- Tidak menggunakan ID **auto increment** di *table* yang krusial, atau dapat dilihat user, gunakan **UUID.**
- Jika *table* memiliki relasi, wajib disertakan di dalam migration.
- Selalu gunakan **softdelete** untuk menghindari kehilangan data penting.

Overview :

- Table Role

```
public function up()
{
    Schema::create('roles', function (Blueprint $table) {
        $table->uuid('id')->primary();
        $table->string('name');

        $table->softDeletes();
        $table->timestamps();
    });
}
```

- Table User

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->uuid('id')->primary();
        $table->uuid('role_id');
        $table->string('email')->nullable();
        $table->string('username');
        $table->string('password');
        $table->date('registered_at');
        $table->string('tmt')->nullable();

        $table->softDeletes();
        $table->timestamps();

        $table->foreign('role_id')->references('id')->on('roles');
    });
}
```

B. Seeding

Gunakan database seeding untuk membuat base data minimum agar aplikasi dapat berjalan dengan lancar dan tanpa error. Pembuatan seeding diperbolehkan menggunakan **Model Factories**, maupun **Query Builder**.

Overview :

- Membuat seed dengan model factory, baca dokumentasi
- Query Builder – *Example : User Seeder*

```php
public function run()
{
    $groupAdmin = Role::where('name', "Admin")->firstOrFail()->id;
    $groupUser = Role::where('name', "User")->firstOrFail()->id;

    $datas = [
        [
            "email" => "admin@example.id",
            "username" => "vgadmin",
            "password" => "4dm1n2022",
            "user_group_id" => $groupAdmin
        ],
        [
            "email" => "creative@example.id",
            "username" => "vgcreative",
            "password" => "cr34t1v3w1sud4pr4smul2021",
            "user_group_id" => $groupUser
        ],
        [
            "email" => "committee@wisuda2021.prasetiyamulya.ac.id",
            "username" => "vgcommittee",
            "password" => "c0mm1tt33w1sud4pr4smul2021",
            "user_group_id" => $groupUser
        ]
    ];

    foreach ($datas as $data){
        $user = new User();
        $user->email = $data["email"];
        $user->username = $data["username"];
        $user->password = Hash::make($data["password"]);
        $user->user_group_id = $data["user_group_id"];
        $user->save();
    }
}
```

C. Route

Routing secara garis besar memiliki arti *mapping* URL aplikasi untuk mengarahkan ke *view, closure,* ataupun *controller*. *Routing* bertanggung jawab untuk menangani permintaan HTTP. Kita bisa menggunakan tipe request GET, POST, PUT, DELETE sesuai kebutuhan.

Dalam dokumen ini ada beberapa aturan yang harus diperhatikan dalam membuat *routing* pada aplikasi, diantaranya:

- Gunakan *grouping* pada route untuk menghindari pengulangan penulisan *path url*
- Setiap *route* harus memiliki *route name*
- Jangan terlalu banyak *clossure* atau *anonymous function*
- Gunakan *resource route* (optional)
- Selalu dahulukan Route::get() sebelum Route::resource()
- Selalu tempatkan *middleware* pada *route*, jangan tempatkan *middleware* pada *controller*
- Boleh menggunakan *route model binding.*

Overview :

```
Route::group(['as' => 'admin.', 'middleware' => 'role:admin'], function () {
    Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard');

    Route::group(['prefix' => 'meta', 'as' => 'meta.'], function () {
        Route::get('/', [MasterDataController::class, 'index'])->name('index');
        Route::get('/datatables', [MasterDataController::class, 'datatables'])->name('datatable');
        Route::get('/show/{id?}', [MasterDataController::class, 'show'])->name('show');
        Route::post('/store', [MasterDataController::class, 'store'])->name('store');
        Route::post('/destroy/{id?}', [MasterDataController::class, 'destroy'])->name('destroy');
    });
});
```

D. Controller

Aturan dalam controller :

- Setiap POST request harus selalu memiliki validation request
- Gunakan blok Try catch seperti yang dijelaskan di atas
- Gunakan DB::Transaction dalam setiap operasi insert, update, dan delete pada database
- Upload file harus selalu berada pada logic terakhir atau line terakhir di setiap fungsinya, taruh tepat di atas DB::commit() agar ketika ada operasi database yang gagal, file belum terupload ke aplikasi
- Untuk datatable, gunakan library yajra datatable (github)

Overview :

```php
public function store(Request $request)
{
    try {
        DB::beginTransaction();
        $data = User::updateOrCreate(
            ['id' => $request->id],
            [
                'role_id'           => Roles::where('name', 'user')->first()->id,
                'email'             => $request->email,
                'username'          => $request->username,
                'password'          => $request->password ?? Hash::make('anggota312'),
                'registered_at'     => $request->registered_at,
                'status'            => $request->status,
                'member_id'         => $request->member_id,
                'name'              => $request->name,
                'phone_number'      => $request->phone_number,
                'address'           => $request->address,
                'tmt'               => $request->tmt,
            ]
        );
        if ($request->file("file_memento")) {
            $file_video = $request->file("file_memento");
            $nameVideo = time() . "_" . $file_video->getClientOriginalName();
            $pathVideo = 'memento/' . $nameVideo;
            $aws=  AwsClient::upload($request->file("file_memento"), $pathVideo);
        }
        DB::commit();
        $this->isSuccess = true;
    } catch (Exception $e) {
        DB::rollBack();

        $this->exception = App::environment('local') ? $e->getMessage() : "Terjadi kesalahan!";
    }

    return response()->json([
        "status"    => $this->isSuccess ?? false,
        "code"      => $this->isSuccess ? 200 : 600,
        "message"   => $this->isSuccess ? "Success!" : ($this->exception ?? "Unknown error(?)"),
        "data"      => $this->isSuccess ? $data : [],
    ], 201);
}
```

```php
public function datatables()
{
    $user = User::with('wallet')
                ->where('role_id', Roles::where('name', user)->first()->id);

    return DataTables::of($user)->make();
}
```

E. Model

Aturan dalam Model :

- Model tidak diperbolehkan menjadi tempat operasi query utama, query dan operasi database harus selalu ditempatkan pada controller
- Selalu gunakan softdelete, dan cascade delete library (github)
- Gunakan trait UUID, tidak diperbolehkan menggenarate uuid di controller
- Gunakan camelCase pada penamaan fungsi relasi model

Overview :

```php
class User extends Authenticatable
{
    use HasFactory, Notifiable, SoftDeletes, CascadeSoftDeletes, Uuid;

    protected $hidden = [
        'password',
    ];


    protected $casts = [
        'id' => 'string'
    ];

    protected $guarded = [];

    public $incrementing = false;

    // In Laravel 6.0+ make sure to also set $keyType
    protected $keyType = 'string';


    protected $cascadeDeletes = ['wallet'];


    public function roles() {
        return $this->belongsTo(Roles::class, 'role_id');
    }
    public function wallet(){
        return $this->hasOne(Wallet::class, 'user_id');
    }


}
```

F. View

Aturan dalam view :

- Agar lebih dinamis, pemanggilan file asset harus menggunakan helper asset, tidak boleh menggunakan direct path
- Setiap action form, ajax, anchor, harus menggunakan route name, tidak boleh menggunakan direct url
- Minimal layouting harus terdiri dari 4 bagian ; Main, Sidebar, Footer, dan Content
- Indentasi HTML harus diperhatikan, pastikan indentasi HTML sudah rapih dan sesuai
- Penggunaan library CSS atau JS harus ditempatkan sesuai kebutuhan page, jika library digunakan di semua page, maka tempatkan pada main layout file, tetapi jika digunakan untuk page tertentu, maka tempatkan library tersebut hanya pada page itu saja.