# DOKUMENTASI QUIZ BEFORE UTS
# WEB PROGRAMMING


| | | |
|---|---|---|
| Nama | : | Fendy Wijaya |
| NIM | : | 2602092150 |
| Kelas | : | LJ01 |
| Mata Kuliah | : | COMP6821001 – Web Programming |
| Dosen | : | D6191 – Reinert Yosua Rumagit, S.Kom., M.T.I. |


## A. Route

Dalam project ini, ketika user masuk ke main path '/', user akan diarahkan secara langsung ke Home, tidak perlu ke template welcome bawaan Laravel. Tampilan yang sama juga akan muncul ketika URL yang dimasukkan oleh user ditambahkan path '/home'. Route ini akan memanggil fungsi index pada DetailController. Untuk mempermudah, route ini dinamai home.

Sesuai dengan kriteria yang ada pada navigation bar, kita memerlukan rute untuk kategori, penulis, tentang kami, dan materi populer. Untuk about us dan materi populer, saya akan menetapkan path URL yang akan digunakan sebagai '/about-us' dan '/popular'.

Untuk about us, sebuah controller dengan nama AboutUsController dibentuk untuk mengendalikan koneksi dengan view, lalu route dinamai dengan about. Sementara untuk popular, akan ada keterkaitan tertentu dengan materi yang disimpan dalam database, sehingga saya tetap menggunakan DetailController, sebagai jembatan penghubung view dan model yang terkait dengan materi setiap kategori mata kuliah yang tersimpan di database. Rute ini kemudian dinamai materi.popular.

Jika diperhatikan, menu kategori disusun oleh drop-down yang mengandung dua kategori mata kuliah yang didukung oleh kasus ini. Oleh karena NIM saya genap, maka kategori yang tersimpan untuk kasus saya ini adalah Interactive Multimedia dan Software Engineering. Sebagai tindak lanjut untuk mempermudah routing, saya menggunakan prefix agar kategori dan penulis dapat ditambah lagi path-nya dengan menu pilihan lain di dalamnya.

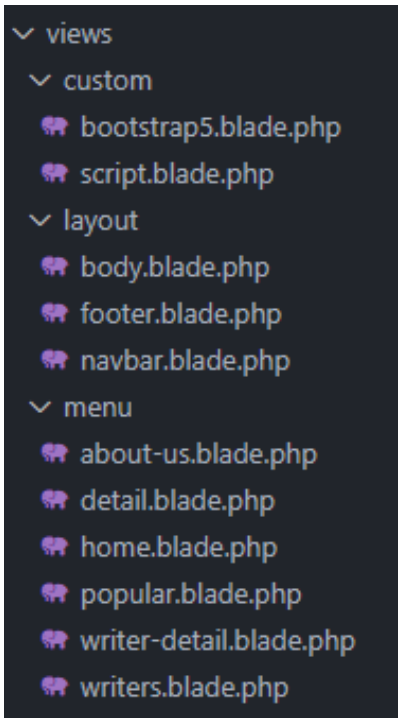
Prefix untuk kategori disusun atas dua grup, yaitu rute '/category/{categoryId}' yang berfungsi untuk menampilkan Category page. Rute ini terkait dengan CategoryController dan akan memunculkan apa saja materi yang disimpan oleh kategori yang dipilih oleh user. Rute lainnya di prefiks ini adalah '/category/{categoryId}/detail/{detailId}', yang berfungsi untuk menampilkan secara khusus isi konten dari materi yang dipilih dari satu kategori tertentu. Oleh karena keterkaitannya lebih ke materi dibanding kategorinya, maka rute ini diarahkan kepada DetailController.

Prefix untuk penulis juga disusun atas dua grup, yaitu rute '/writer' yang berfungsi untuk menampilkan Our Writers page, yang berisi nama-nama penulis yang tersimpan dalam database. Rute ini terkait dengan WriterController untuk menampilkan nama-nama penulis yang ada. Rute lainnya di prefiks ini adalah '/writer/{writerId}', yang berfungsi untuk menampilkan secara khusus konten-konten yang ditulis oleh penulis yang dipilih, sehingga terkait hubungannya dengan WriterController.

```php
Route::get('/', [DetailController::class, 'index'])->name('home');

Route::get('/home', [DetailController::class, 'index'])->name('home');

Route::get('/about-us', [AboutUsController::class, 'index'])->name('about');

Route::get('/popular', [DetailController::class, 'popular'])->name('materi.popular');

Route::prefix('/category')->group(function () {
    Route::get('/{categoryId}', [CategoryController::class, 'index'])->name('category');
    Route::get('/{categoryId}/detail/{detailId}', [DetailController::class, 'detail'])->name('materi.detail');
});

Route::prefix('/writer')->group(function () {
    Route::get('/', [WriterController::class, 'index'])->name('writer');
    Route::get('/{writerId}', [WriterController::class, 'detail'])->name('writer.detail');
});
```

*Route di project ini*

## B. View dan Blade




Untuk kasus project ini, view yang dibuat terpisah dalam tiga folder, yaitu custom, layout, dan menu. Folder custom adalah file blade untuk menyimpan tag style yang telah mengimplementasikan Bootstrap 5 dari public folder (bootstrap5.blade.php) serta file blade yang menyimpan tag script yang menyimpan kode JavaScript untuk memungkinkan drop-down pada navigation bar (script.blade.php). Kedua file blade ini kelak hanya perlu di-include saja agar bisa dipakai di semua file dengan mudah.

Folder layout adalah folder yang menjadi kerangka frontend dari website yang dibuat. File body.blade.php berperan untuk menyimpan kerangka tubuh yang akan menampung konten dari setiap halaman. File footer.blade.php menyimpan footer setiap halaman, dan file navbar.blade.php menyimpan navigation bar setiap halaman. Setiap file ini menerapkan mekanisme penulisan blade, baik untuk route, include, extends, yield, dan section.

Folder menu adalah folder yang terkait dengan seluruh opsi pada navigation bar. Hampir semua namanya telah merepresentasikan menu yang dikendalikan. Hanya saja, detail.blade.php mengendalikan page untuk setiap materi, dan write-detail.blade.php mengendalikan page untuk tulisan-tulisan yang pernah ditulis oleh penulis.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@yield('title')</title>
    @include('custom.bootstrap5')
</head>
<body style="background-color: #add8e6; margin: 0; display: flex; flex-direction: column; overflow-x: hidden; min-height: 100vh;">
    <div class="container-fluid p-0 flex-grow-1">
        @include('layout.navbar')
        @yield('konten')
    </div>
    @include('layout.footer')

    @include('custom.script')
</body>
</html>
```

*View untuk body.blade.php*

```
@extends('layout.body')
@section('title', $details->materi)

@section('konten')
<div class="row">
    <div class="col-md-10 offset-md-1">
        <h2 style="margin-top: 120px; margin-bottom: 20px;">{{ $details->materi }}</h2>

        <div style="display: flex; flex-direction: column; gap: 20px; max-width: 1400px; margin: 40px auto 40px auto;">
            <div style="flex: 1; background-color: #004aad; display: flex; align-items: center; justify-content: center;">
                <img src="{{ asset('images/HCI.png') }}" alt="HCI" style="width: 100%; height: 100%;">
            </div>
            <div style="display: flex; background-color: #fff; border-radius: 10px; overflow: hidden; box-shadow: 0 4px 8px rgba(0,0,0,0.1);">

                <div style="flex: 2; padding: 15px; display: flex; flex-direction: column;">
                    <p style="font-size: 12px; color: #666; margin: 5px 0;">{{ $details->tanggalPost }} | By {{ $details->writers->namaPenulis }}
                    <p style="font-size: 14px; color: #444; margin: 10px 0; flex: 1;">{{ $details->konten }}</p>
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

*View untuk detail.blade.php*

## C. Controller

Dalam kasus project ini, terdapat lima controller yang dipakai sebagai jembatan penghubung dari route ke setiap view-nya. Pada bagian route, setiap jalur yang telah dideklarasikan sudah diimplementasi secara langsung ke controller-nya masing-masing.

```php
class AboutUsController extends Controller
{
    public function index(){
        return view('menu.about-us');
    }
}
```

*AboutUsController*

AboutUsController diisi dengan fungsi index yang dimanfaatkan untuk me-return langsung view about-us.blade.php.

```php
class CategoryController extends Controller
{
    public function index($categoryId){
        $category = Category::find($categoryId);
        $details = $category->detail;
        return view('menu.home')->with('details', $details);
    }
}
```

*CategoryController*

CategoryController diisi dengan function index yang menerima parameter categoryId dari route untuk diarahkan ke model agar dapat dicari di database. Pemanggilan view memerlukan passing parameter details untuk menampilkan nama-nama dan atribut setiap materi yang ada di home page.

```php
class DetailController extends Controller
{
    public function index(){
        $details = Detail::all();
        return view('menu.home')->with('details', $details);
    }

    public function popular(){
        $details = Detail::orderBy('tanggalPost', 'desc')->paginate(3);
        return view('menu.popular')->with('details', $details);
    }

    public function detail($categoryId, $detailId){
        $detail = Detail::find($detailId);
        return view('menu.detail')->with('details', $detail);
    }
}
```

*DetailController*

DetailController merupakan controller yang cukup banyak diakses oleh beberapa route, sebab controller ini mengatur terkait semua materi yang ada di website. Function index dipakai untuk menampilkan isi materi dalam kategori di home page. Fungsi popular digunakan ketika mengelola popular page. Function detail dipakai ketika ingin menampilkan isi dari setiap materi dari halaman Category.

```php
class HomeController extends Controller
{
    public function index(){
        return view('menu.home');
    }
}
```

*HomeController*

HomeController diisi dengan fungsi index yang dimanfaatkan untuk me-return langsung view home.blade.php.

```php
class WriterController extends Controller
{
    public function index(){
        $writers = Writer::all();
        return view('menu.writers', compact('writers'));
    }

    public function detail($writerId){
        $writer = Writer::find($writerId);
        $details = $writer->detail;
        return view('menu.writer-detail')->with(['details' => $details, 'writer' => $writer]);
    }
}
```

*WriterController*

WriterController tersusun atas fungsi index yang melempar parameter ke view untuk melakukan looping nama-nama penulis yang terdata di database. Sedangkan fungsi detail digunakan untuk melempar data dari model untuk dipakai oleh view sebagai sarana memunculkan detail dari materi yang ditulis tersebut.

## D. Model

Terdapat tiga model yang akan digunakan dalam project ini untuk melakukan komunikasi dengan database, yaitu Category, Detail, dan Writer.

```php
class Category extends Model
{
    use HasFactory;

    protected $table = 'category';
    protected $fillable = ['namaKategori'];

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

*Model Category*

Sebagai penanda bahwa model ini akan melakukan komunikasi dengan tabel category, maka kelas model ini perlu dideklarasikan terlebih dahulu sebagai ‘category’. Dalam konteks relational database, satu buah kategori bisa memiliki banyak detail materi, sehingga model perlu dibuatkan function untuk berelasi dengan tabel detail secara one-to-many.

```php
class Detail extends Model
{
    use HasFactory;

    protected $table = 'detail';
    protected $fillable = ['category_id', 'writer_id', 'materi', 'tanggalPost', 'deskripsi', 'konten'];

    public function categories(){
        return $this->belongsTo(Category::class, 'category_id');
    }

    public function writers(){
        return $this->belongsTo(Writer::class, 'writer_id');
    }
}
```

*Model Detail*

Sebagai penanda bahwa model ini akan melakukan komunikasi dengan tabel detail, maka kelas model ini perlu dideklarasikan terlebih dahulu sebagai ‘detail’. Dalam konteks relational database, satu buah materi detail hanya bisa dimiliki oleh satu kategori, sehingga model perlu dibuatkan function untuk berelasi belongsTo dengan tabel detail secara one-to-one. Selain itu, satu buah materi detail juga hanya bisa dimiliki oleh satu penulis, sehingga model juga perlu berelasi belongsTo dengan tabel writer secara one-to-one.

```php
class Writer extends Model
{
    use HasFactory;

    protected $table = 'writer';
    protected $fillable = ['namaPenulis'];

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

*Model Writer*

Sebagai penanda bahwa model ini akan melakukan komunikasi dengan tabel writer, maka kelas model ini perlu dideklarasikan terlebih dahulu sebagai 'writer'. Dalam konteks relational database, satu orang penulis bisa menulis banyak materi, sehingga model perlu dibuatkan function untuk berelasi dengan tabel detail secara one-to-many.

## E. Database Migration

Untuk migrasi database, tabel yang dibuat harus sama dengan model yang ada. Oleh karena itu, perlu dibuat file migration bernama create_category_table, create_writer_table, dan create_detail_table.

```php
public function up(): void
{
    Schema::create('category', function (Blueprint $table) {
        $table->id();
        $table->string('namaKategori');
        $table->timestamps();
    });
}
```

*Create table category*

Tabel kategori hanya akan berisi atribut nama kategorinya saja, sedangkan id dan timestamp sudah otomatis terbuatkan oleh template Laravel. Pada tabel ini, id akan menjadi primary key secara otomatis dengan sifat auto-increment secara bawaan.

```php
public function up(): void
{
    Schema::create('writer', function (Blueprint $table) {
        $table->id();
        $table->string('namaPenulis');
        $table->timestamps();
    });
}
```

*Create table writer*

Tabel writer hanya akan berisi atribut nama penulisnya saja, sedangkan id dan timestamp sudah otomatis terbuatkan oleh template Laravel. Pada tabel ini, id juga akan menjadi primary key secara otomatis dengan sifat auto-increment secara bawaan.

```php
public function up(): void
{
    Schema::create('detail', function (Blueprint $table) {
        $table->id();
        $table->foreignId('category_id')->constrained('category')->onDelete('cascade');
        $table->foreignId('writer_id')->constrained('writer')->onDelete('cascade');
        $table->string('materi');
        $table->date('tanggalPost');
        $table->string('deskripsi');
        $table->string('konten');
        $table->timestamps();
    });
}
```

*Create table detail*

Tabel detail berisi atribut berupa judul materi, tanggal posting materi, deskripsi singkat (untuk ditampilkan di home), isi konten (untuk ditampilkan di halaman materi), serta category_id dan writer_id yang berperan sebagai foreign key dari tabel category dan writer. Sementara id dan timestamp sudah otomatis terbuatkan oleh template Laravel. Pada tabel ini, id akan menjadi primary key secara otomatis dengan sifat auto-increment secara bawaan.

## F. Seeder dan Faker

Secara default, Laravel sudah menyediakan tempat khusus untuk mengatur seeding (memasukkan nilai ke database), yaitu di DatabaseSeeder. Khusus untuk soal kasus ini, akan ada beberapa kolom pada tabel yang diisi secara manual, dan ada beberapa kolom yang diisi dengan menggunakan Faker.

```php
public function run(): void
{
    $faker = Faker::create('id_ID');

    // Menggabungkan data faker dengan data manual
    $detailData = array_merge([
        'category_id' => Category::inRandomOrder()->first()->id,
        'writer_id' => Writer::inRandomOrder()->first()->id,
        'tanggalPost' => $faker->dateTimeBetween('-1 week', '+1 week'),
        'deskripsi' => $faker->sentence,
        'konten' => $faker->paragraph,
    ], self::$manualData);

    Detail::create($detailData);
}
```

*DetailSeeder*

Untuk melakukan seeding pada tabel detail, kita memerlukan faker untuk membuat tanggal posting sesuai range tertentu, deskripsi, dan konten.

```php
public function run(): void
{
    $faker = Faker::create('id_ID');

    for($i=0; $i<3; $i++){
        Writer::create([
            'namaPenulis' => $faker->name
        ]);
    }
}
```

*WriterSeeder*

Untuk tabel writer, website akan memerlukan nama-nama penulis, dan untuk ini, saya kembali menggunakan faker. Faker di-setting untuk looping sebanyak 3 kali agar terbentuk 3 nama.

```php
public function run(): void
{
    // Buat kategori secara manual
    $categoryID1 = Category::create([
        'namaKategori' => 'Interactive Multimedia'
    ]);

    $categoryID2 = Category::create([
        'namaKategori' => 'Software Engineering'
    ]);

    // Panggil seeder untuk Writer
    $this->call([
        WriterSeeder::class,
    ]);

    // Tetapkan data manual dan panggil DetailSeeder
    DetailSeeder::$manualData = [
        'materi' => 'Human and Computer Interaction',
        'category_id' => $categoryID1->id,
        'writer_id' => Writer::inRandomOrder()->first()->id,
    ];
    $this->call(DetailSeeder::class);

    DetailSeeder::$manualData = [
        'materi' => 'User Experience',
        'category_id' => $categoryID1->id,
        'writer_id' => Writer::inRandomOrder()->first()->id,
    ];
    $this->call(DetailSeeder::class);
```

*DatabaseSeeder*

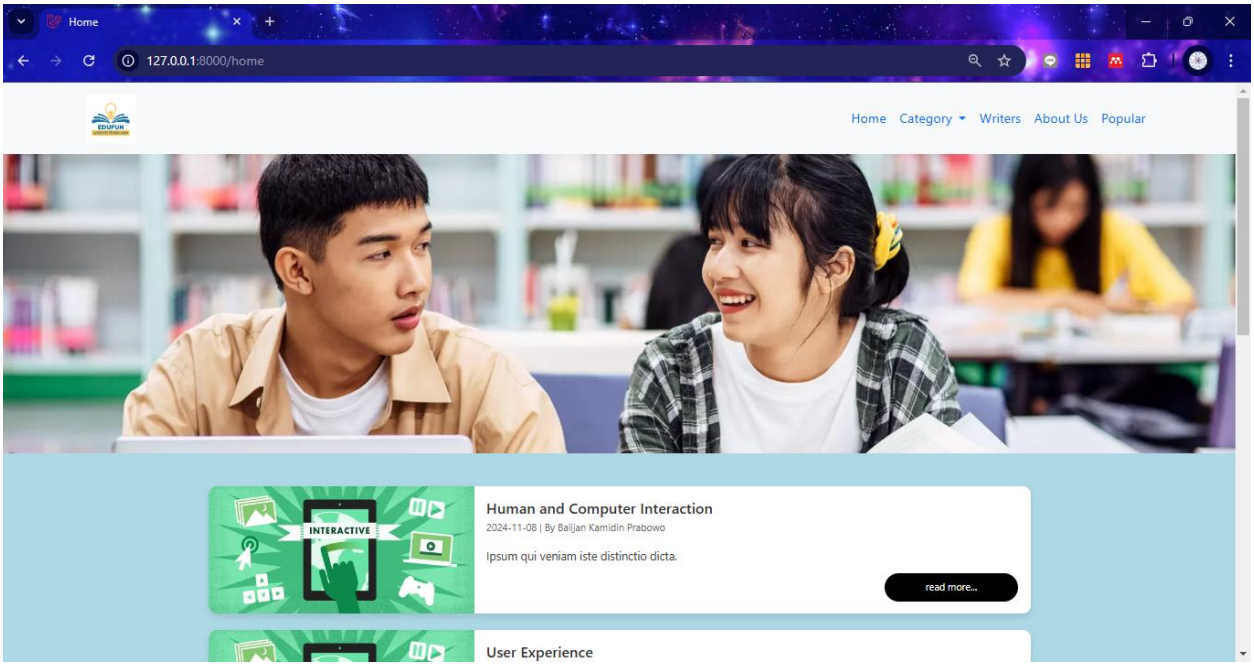
Dengan mendaftarkan seeder ke DatabaseSeeder, maka seluruh tabel akan terisi, baik writer, category, maupun detail. Tampak bahwa kategori diisi secara manual, karena hanya ada dua kategori. Judul materi untuk setiap kategori juga ditulis manual. Sisanya, kecuali id, dapat dibantu dengan menggunakan faker.

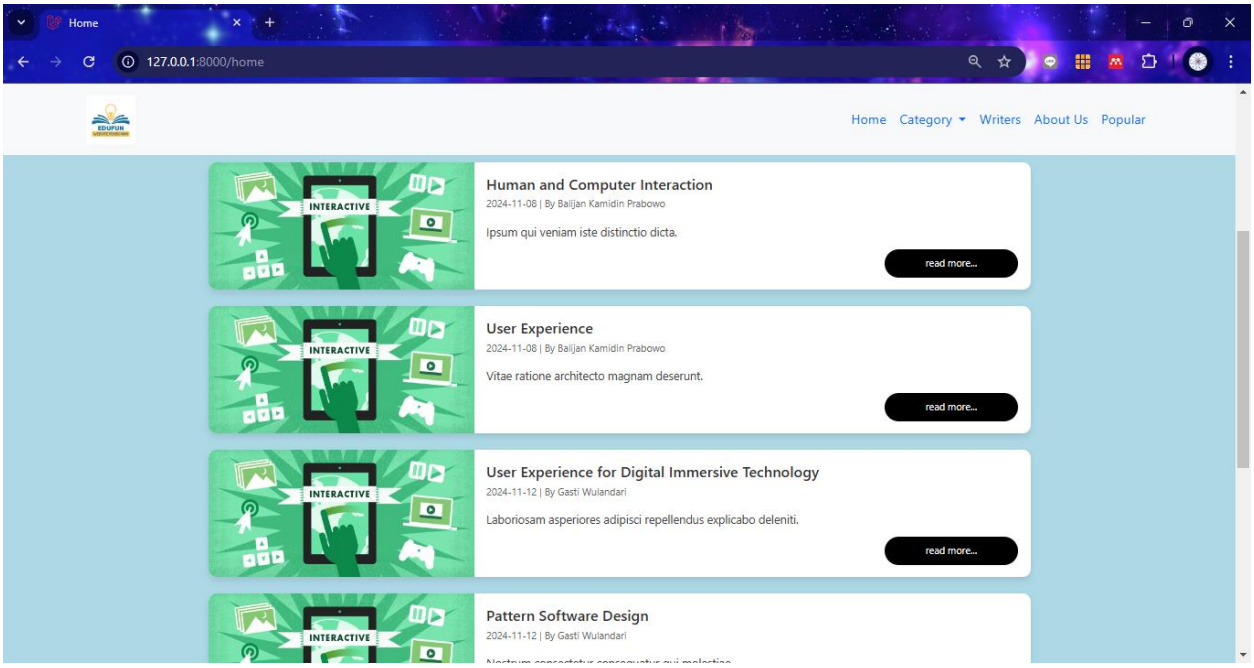
Notes:

Lampiran screenshoot di halaman berikutnya.

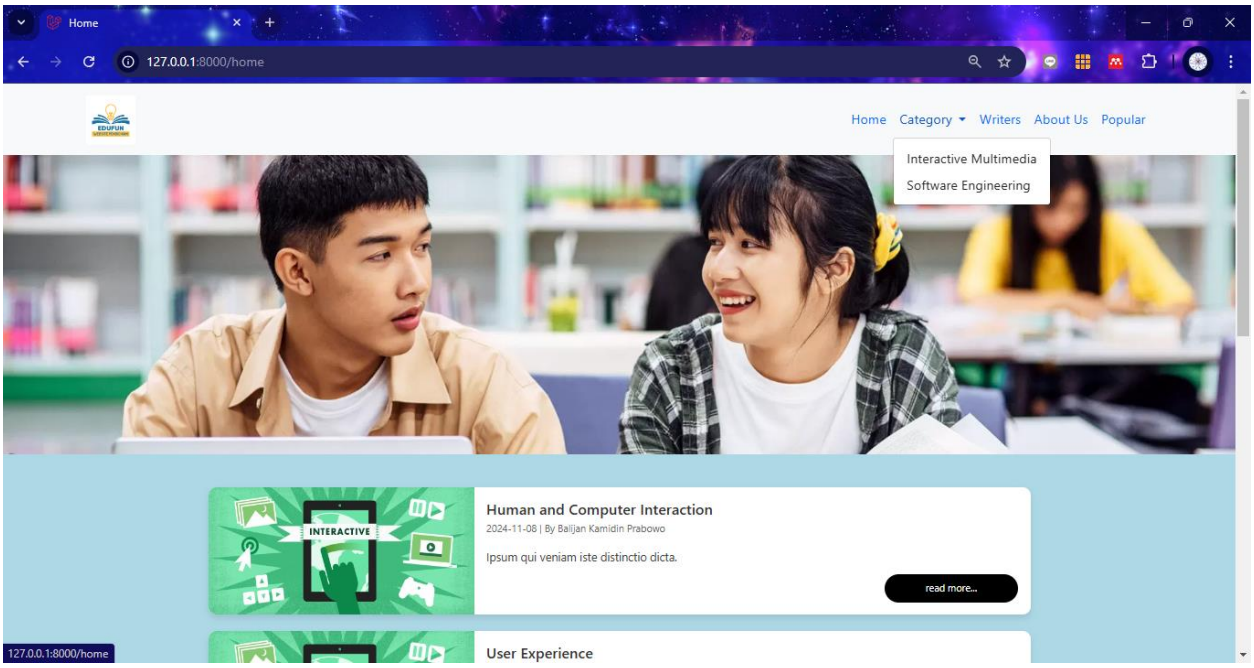
# LAMPIRAN

## Home Page:

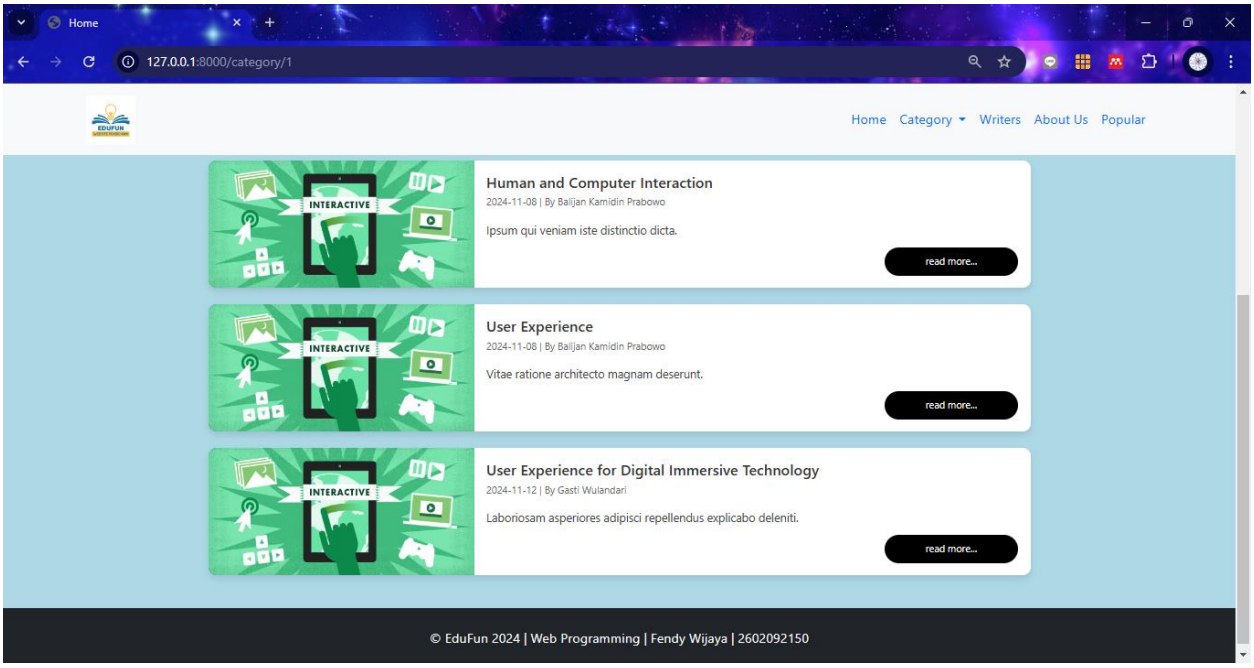


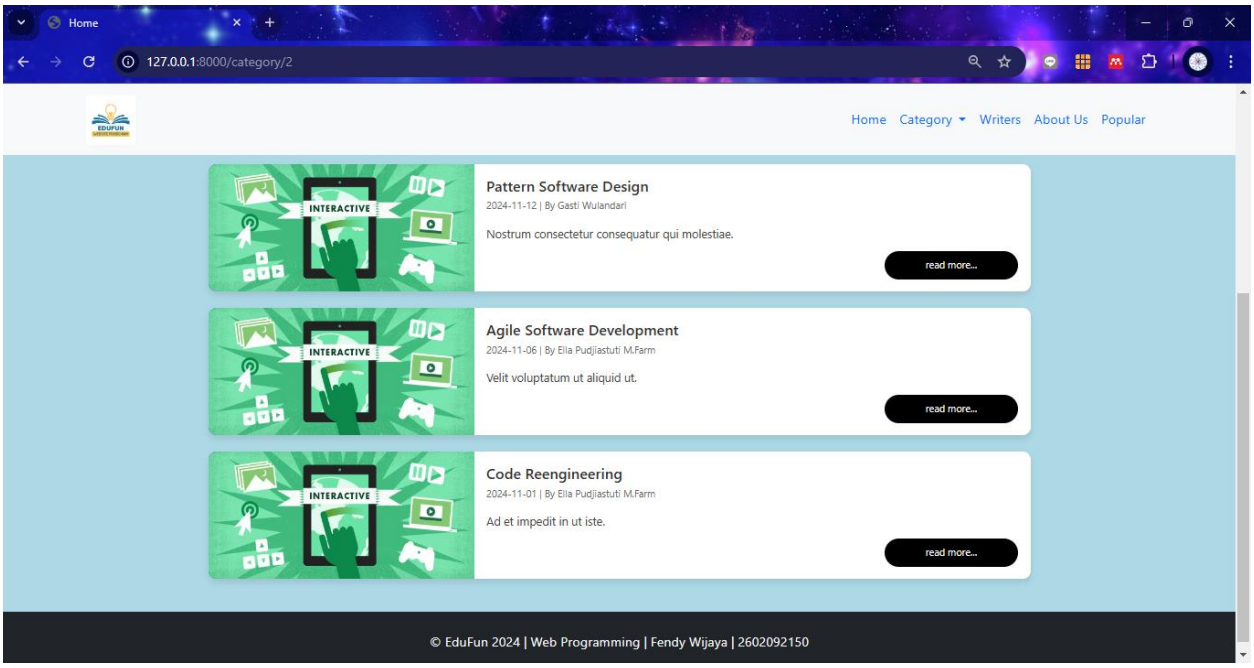
## Category Page: (drop-down bekerja)



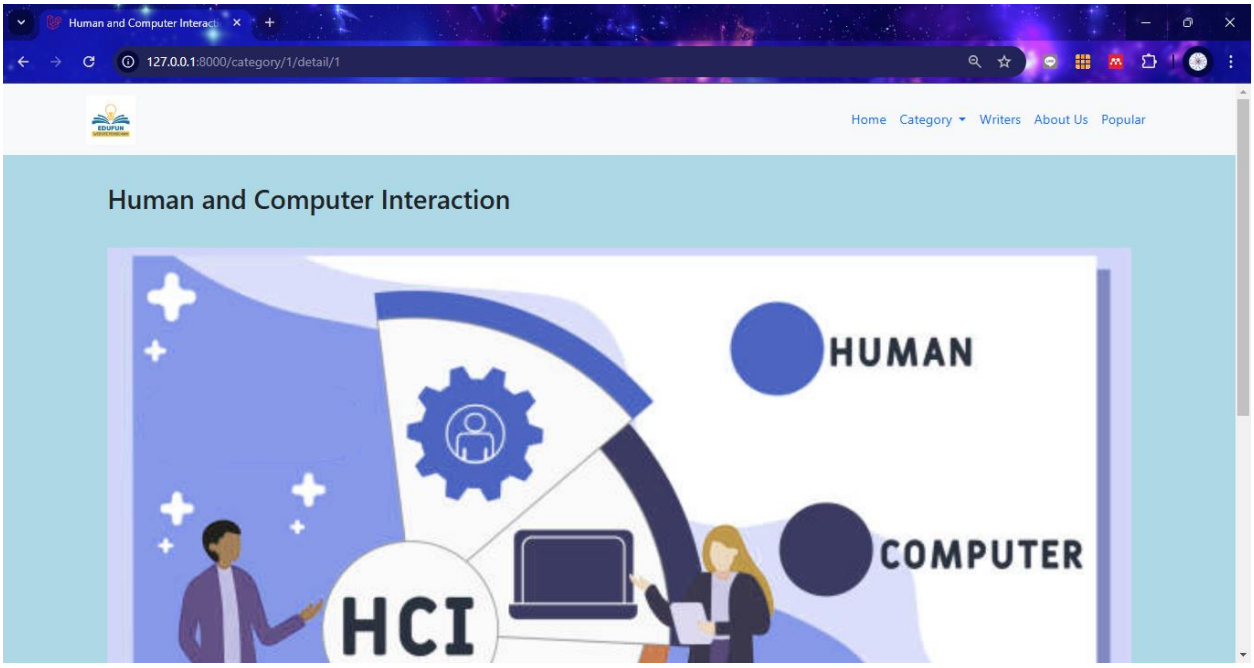

## Category Page (Interactive Multimedia):



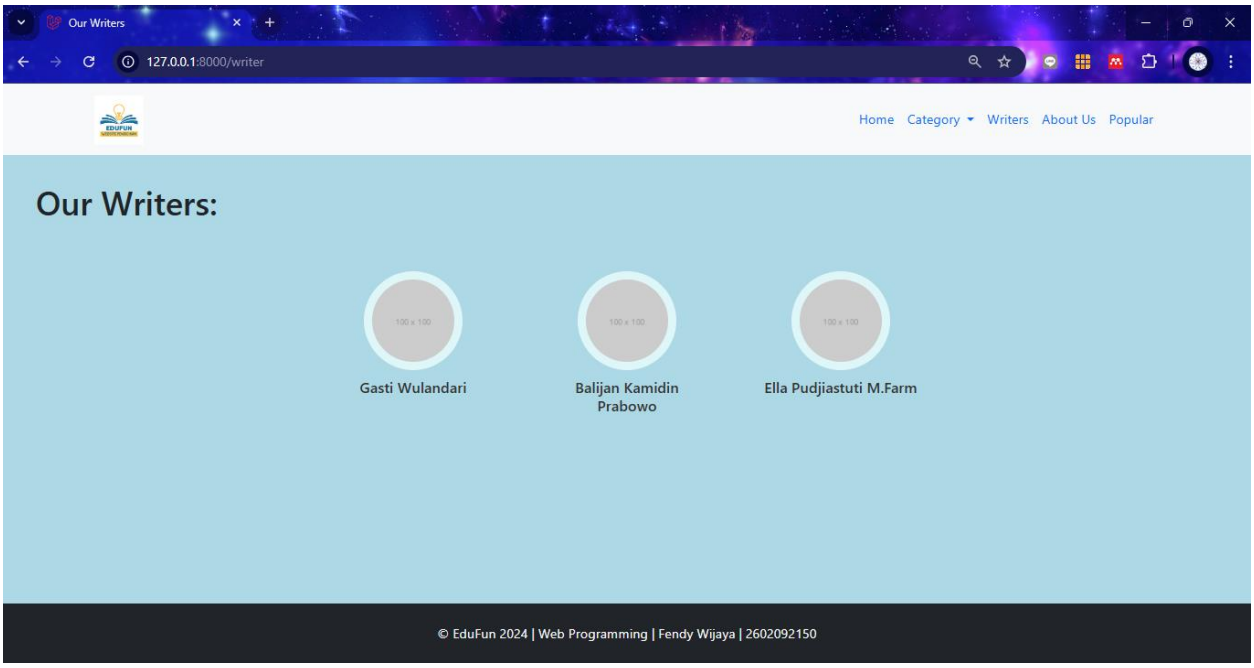
## Category Page (Software Engineering):



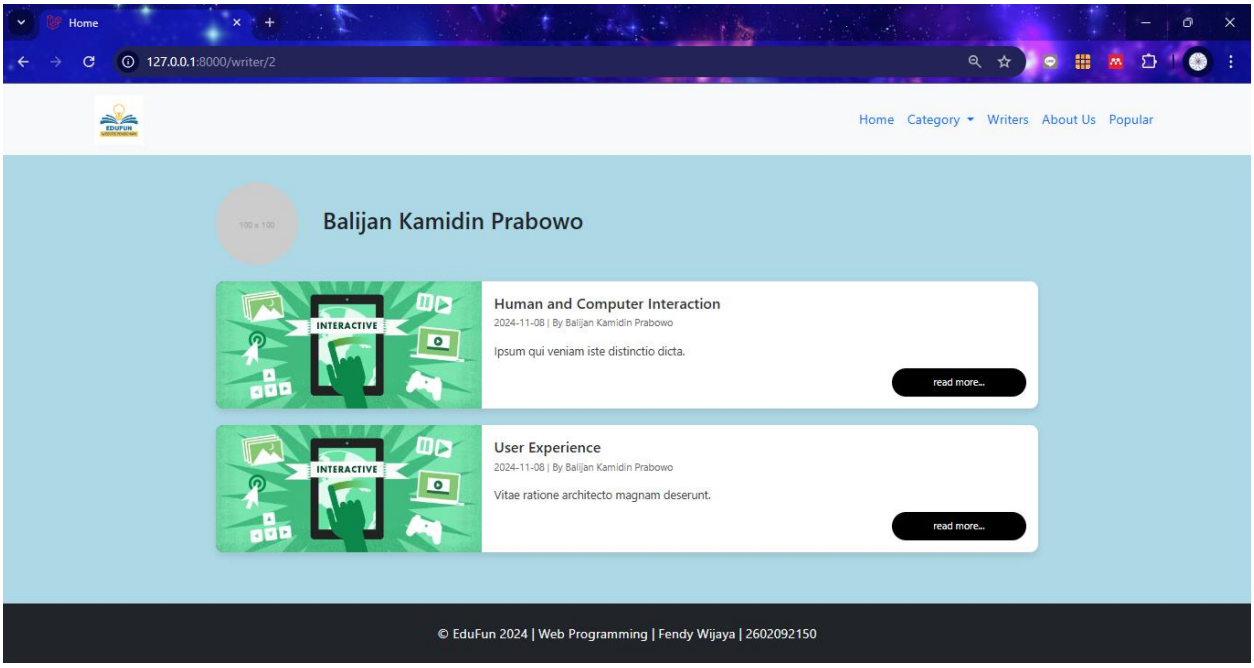

## Detail Materi:



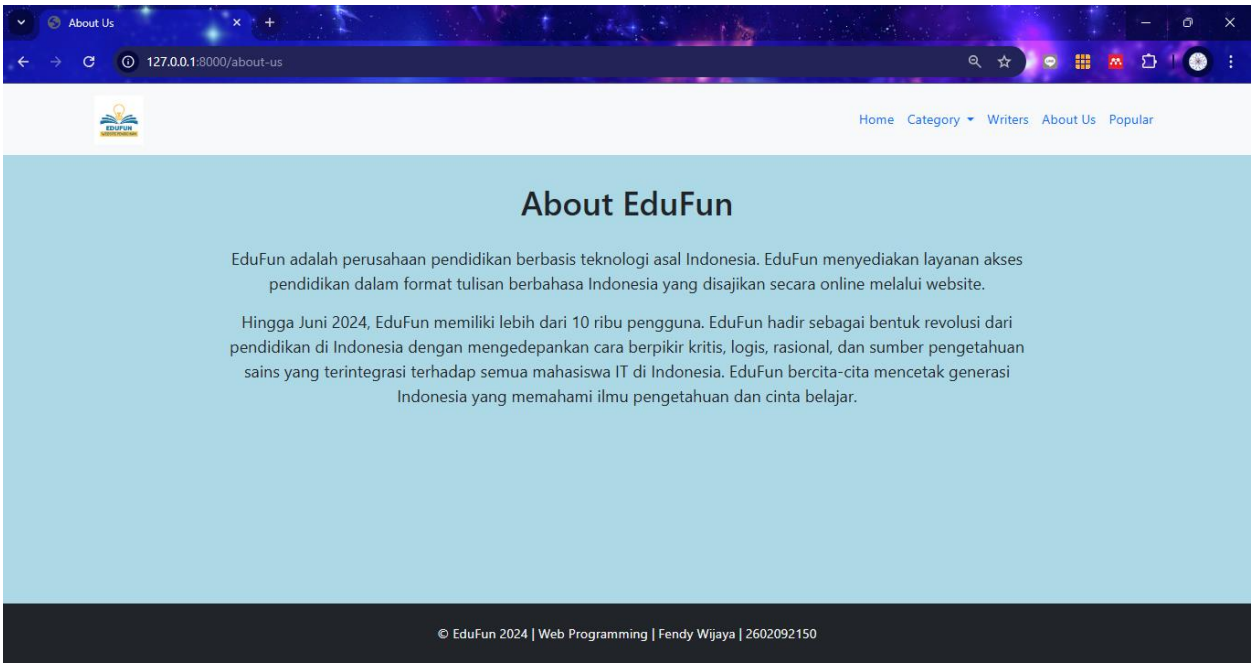
## Writers Page:



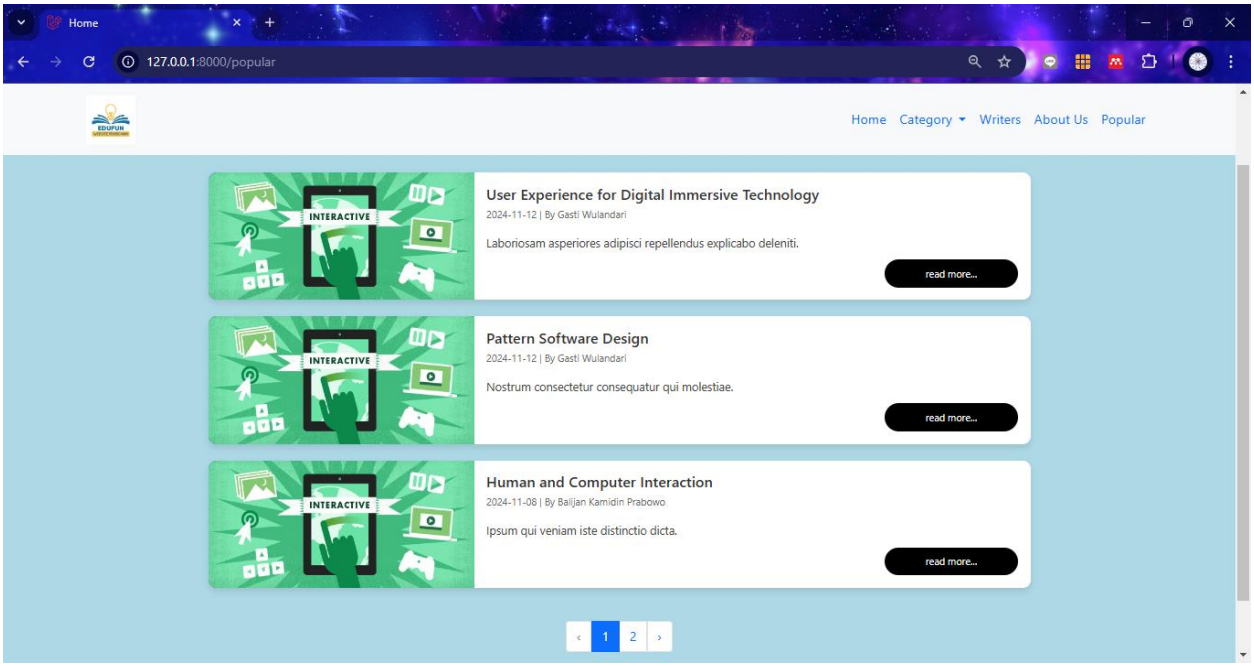

## Writers Detail Page:



## About Us Page:




## Popular Page:




Keterangan: Popularitas didasarkan pada konten yang paling baru posting.