LAMPIRAN

Lampiran 1. Kode Program

```
<?php

namespace App\Http\Controllers;

use App\Models\Kriteria;
use Illuminate\Support\Facades\DB;

class HasilAHPController extends Controller
{
    public function index()
    {
        $alternatif_id        = $_POST['alternatif_id'];
        $kriteria_id          = $_POST['kriteria_id'];
        $sub_kriteria_id      = $_POST['sub_kriteria_id'];

        $jumlah_data = count($alternatif_id);
        for($i = 0; $i < $jumlah_data;$i++) {
            $data['alternatif_id'] = $alternatif_id[$i];
            $data['kriteria_id'] = $kriteria_id[$i];
            $data['sub_kriteria_id'] = $sub_kriteria_id[$i];

            $jml = DB::table("nilai_alternatif")
            ->where('alternatif_id', '=', $alternatif_id[$i])
            ->where('kriteria_id', '=', $kriteria_id[$i])
            ->count();
            if ($jml==0) {
                DB::table('nilai_alternatif')->insert([
                    'alternatif_id' => $alternatif_id[$i],
                    'kriteria_id' => $kriteria_id[$i],
                    'sub_kriteria_id' => $sub_kriteria_id[$i],
                ]);
            } else {
                DB::table('nilai_alternatif')
                    ->where('alternatif_id', $alternatif_id[$i])
                    ->where('kriteria_id', $kriteria_id[$i])
                    ->update(['sub_kriteria_id' => $sub_kriteria_id[$i]]);
            }
        }
        $list = DB::select("SELECT nilai_alternatif.*,pv_sub_kriteria.nilai FROM
nilai_alternatif JOIN pv_sub_kriteria on
pv_sub_kriteria.sub_kriteria_id=nilai_alternatif.sub_kriteria_id");
        // cek jml data
        $jml_list =count($list); //24
        $jml_data = 0;
        //cek nilai pv kriteria
        foreach ($list as $key => $row) {
```

```
$urutan = $key + 1;
$alternatif_id = $row->alternatif_id; //1
//ambil nilai pv kriteria
$query = DB::table('pv_kriteria')
->where('pv_kriteria.kriteria_id', $row->kriteria_id)
->first();
if($query) {
   $nilai_pv = $query->nilai;
 } else {
   $nilai_pv = null;
 }


// $nilai_pv = $query->nilai;

$jumlah_nilai_alternatif = $nilai_pv * $row->nilai;
if ($urutan ==1) {
   $jml_data = $jml_data + $jumlah_nilai_alternatif;
   $alternatif_sebelumnya = $row->alternatif_id;
}
else if($alternatif_sebelumnya == $row->alternatif_id) {
   if ($jml_list != $urutan) {
      $jml_data = $jml_data + $jumlah_nilai_alternatif;
   }
   else if( $jml_list == $urutan){
      $jml_data = $jml_data + $jumlah_nilai_alternatif;
      $jml = DB::table("rangking_ahp")
      ->where('alternatif_id', '=', $alternatif_sebelumnya)
      ->count();
      if ($jml==0) {
         DB::table('rangking_ahp')->insert([
            'alternatif_id' => $alternatif_sebelumnya,
            'nilai_prioritas' => $jml_data,
         ]);
      }
      else {
         DB::table('rangking_ahp')
            ->where('alternatif_id', $alternatif_sebelumnya)
            ->update(['nilai_prioritas' => $jml_data]);
      }
   }
}
else if ($alternatif_sebelumnya != $row->alternatif_id) {
   //cen untuk input atw update data jumlah ke tebel
   $jml = DB::table("rangking_ahp")
   ->where('alternatif_id', '=', $alternatif_sebelumnya)
   ->count();
   if ($jml==0) {
```

```
                    DB::table('rangking_ahp')->insert([
                        'alternatif_id' => $alternatif_sebelumnya,
                        'nilai_prioritas' => $jml_data,
                    ]);
                } else {
                    DB::table('rangking_ahp')
                        ->where('alternatif_id', $alternatif_sebelumnya)
                        ->update(['nilai_prioritas' => $jml_data]);
                }
                //set alternatif baru
                $alternatif_sebelumnya = $row->alternatif_id;
                //set jml jadi 0 lagi dan tambah dengan jml baru
                $jml_data = 0 + $jumlah_nilai_alternatif;
            }
                }

        $criteriaCount = Kriteria::count();

        return view('dashboard.hasil_ahp.index', compact(['criteriaCount']))-
>with([
            'alternatif_id' => $alternatif_id,
            'kriteria_id' => $kriteria_id,
            'sub_kriteria_id' => $sub_kriteria_id,
        ]);
    }

    public function hasil()
    {
        $checkHasEmptyData = $this->check_nilai_bobot_has_empty_data();
        $hasilPerankingan = [];
        if (!$checkHasEmptyData) {
            $hasilPerankingan = $this->sorting_vektor_v();
        }

        return view('dashboard.hasil_ahp.hasil', compact(['hasilPerankingan',
'checkHasEmptyData']));
    }

    protected function persentase_bobot()
    {
        $allBobot = DB::table('kriteria')->get();
        $totalBobot = $allBobot->sum('bobot');
        $result = [];

        foreach ($allBobot as $item) {
            // Kali 10 agar total menjadi bobot = 10
            $resultBobot = $item->bobot / $totalBobot * 10;
```

```
          $result[] = ['id' => $item->id, 'name' => $item->name, 'type' => $item-
>type, 'bobot' => $item->bobot, 'persentase_bobot' => $resultBobot];
        }

        return $result;
    }

    protected function matrix_ternormalisasi($arrayNilaiBobotByAlternatifId =
[])
    {
        $result = [];

        foreach ($arrayNilaiBobotByAlternatifId as $item) {
            $selectedNilaiBobot = DB::table('nilai_bobot')
                ->where('alternatif_id', '=', $item->alternatif_id)
                ->orderBy('kriteria_id')
                ->get();

            $result[] = $this->min_max($selectedNilaiBobot);
        }

        return $result;
    }

    protected function min_max($arraySelectedNilaiBobot = [])
    {
        $nilaiBobotGroupByKriteriaId = DB::table('nilai_bobot')
            ->join('kriteria', 'nilai_bobot.kriteria_id', '=', 'kriteria.id')
            ->select('kriteria_id', 'kriteria.name', 'kriteria.type',
DB::raw('MAX(nilai) AS max_nilai_kriteria'), DB::raw('MIN(nilai) AS
min_nilai_kriteria'))
            ->orderBy('kriteria_id')
            ->groupBy('kriteria_id')
            ->get();

        $result = [];

        for ($i = 0; $i < count($nilaiBobotGroupByKriteriaId); $i++) {
            // nilai / MAX VALUE
            if ($nilaiBobotGroupByKriteriaId[$i]->type == 'benefit') {
                $valueR = $arraySelectedNilaiBobot[$i]->nilai /
$nilaiBobotGroupByKriteriaId[$i]->max_nilai_kriteria;
                $result[] = ['krit_name' => $nilaiBobotGroupByKriteriaId[$i]->name,
'value_r' => $valueR];
            }

            // MIN VALUE / nilai
            if ($nilaiBobotGroupByKriteriaId[$i]->type == 'cost') {
                $valueR = $nilaiBobotGroupByKriteriaId[$i]->min_nilai_kriteria /
$arraySelectedNilaiBobot[$i]->nilai;
                $result[] = ['krit_name' => $nilaiBobotGroupByKriteriaId[$i]->name,
'value_r' => $valueR];
            }
        }

        return $result;
    }

    protected function calculate_ranking()
    {
        $nilaiBobotGroupByAlternatifId = DB::table('nilai_bobot')
            ->join('alternatif', 'nilai_bobot.alternatif_id', '=', 'alternatif.id')
            ->select('nilai_bobot.alternatif_id', 'alternatif.code',
'alternatif.kode_database')
            ->orderBy('nilai_bobot.alternatif_id')
            ->groupBy('nilai_bobot.alternatif_id')
            ->get();
        $persentaseBobot = $this->persentase_bobot();
        $matrixTernormalisasi = $this-
>matrix_ternormalisasi($nilaiBobotGroupByAlternatifId);
        $result = [];

        for ($i = 0; $i < count($nilaiBobotGroupByAlternatifId); $i++) {
            // 0 Karena Penjumlahan
            $vektorV = 0;

            for ($j = 0; $j < count($persentaseBobot); $j++) {
                $vektorV += $persentaseBobot[$j]['persentase_bobot'] *
$matrixTernormalisasi[$i][$j]['value_r'];
            }

            $result[] = ['alternatif_code' => $nilaiBobotGroupByAlternatifId[$i]-
>code, 'kode_database' => $nilaiBobotGroupByAlternatifId[$i]-
>kode_database, 'vektor_v' => $vektorV];
        }

        return $result;
    }

    protected function sorting_vektor_v()
    {
        $result = $this->calculate_ranking();
        $selectedSorting = array_column($result, 'vektor_v');
        array_multisort($selectedSorting, SORT_DESC, $result);

        return $result;
```

```
   }

   // Get From NilaiBobotController with some Edit
   protected function check_nilai_bobot_has_empty_data()
   {
      $condition = false;
      $EMPTY_VALUE = 'Empty';
      $allKriteria = Kriteria::all();
      $nilaiBobotGroupByAlternatifId = DB::table('nilai_bobot')
         ->join('alternatif', 'nilai_bobot.alternatif_id', '=', 'alternatif.id')
         ->select('alternatif_id', 'code', 'kode_database',)
         ->orderBy('alternatif.code')
         ->groupBy('alternatif.id')
         ->get();

      // Array For Compare to Data Nilai Bobot
      $arrayKriteriaIdFromKriteria = [];
      foreach($allKriteria as $item) {
         $arrayKriteriaIdFromKriteria[] = $item->id;
      }

      foreach ($nilaiBobotGroupByAlternatifId as $item) {
         $dataKriteria = [];

         $selectedNilaiBobot = DB::table('nilai_bobot')
            ->where('alternatif_id', '=', $item->alternatif_id)
            ->orderBy('kriteria_id')
            ->get();

         // Array for Compare to Data Kriteria ID
         $arrayKriteriaIdFromSelectedNilaiBobot = [];
         foreach ($selectedNilaiBobot as $itemKriteria) {
            $arrayKriteriaIdFromSelectedNilaiBobot[] = $itemKriteria-
>kriteria_id;
            $dataKriteria[] = ['kriteria_id' => $itemKriteria->kriteria_id, 'nilai' =>
$itemKriteria->nilai];
         }

         // Comparing Data Kriteria ID & Nilai Bobot
         $emptyKriteriaId = array_diff($arrayKriteriaIdFromKriteria,
$arrayKriteriaIdFromSelectedNilaiBobot);
         foreach($emptyKriteriaId as $kriteriaId) {
            $dataKriteria[] = ['kriteria_id' => $kriteriaId, 'nilai' =>
$EMPTY_VALUE];
            $condition = true;
            break;
         }
      }
```

```
        return $condition;
    }
}
```

```
<?php

namespace App\Http\Controllers;

use App\Models\Kriteria;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class HasilSAWController extends Controller
{
    public function index()
    {
        $checkHasEmptyData = $this->check_nilai_bobot_has_empty_data();
        $nilaiBobotGroupByAlternatifId = [];
        $persentaseBobot = [];
        $matrixTernormalisasi = [];
        if (!$checkHasEmptyData) {
            $nilaiBobotGroupByAlternatifId = DB::table('nilai_bobot')
                ->join('alternatif', 'nilai_bobot.alternatif_id', '=', 'alternatif.id')
                ->select('nilai_bobot.alternatif_id', 'alternatif.code',
'alternatif.kode_database')
                ->orderBy('nilai_bobot.alternatif_id')
                ->groupBy('nilai_bobot.alternatif_id')
                ->get();
            $persentaseBobot = $this->persentase_bobot();
            $matrixTernormalisasi = $this-
>matrix_ternormalisasi($nilaiBobotGroupByAlternatifId);
        }

        return view('dashboard.hasil_saw.index',
compact(['nilaiBobotGroupByAlternatifId', 'persentaseBobot',
'matrixTernormalisasi', 'checkHasEmptyData']));
    }

    public function hasil()
    {
        $checkHasEmptyData = $this->check_nilai_bobot_has_empty_data();
        $hasilPerankingan = [];
        if (!$checkHasEmptyData) {
            $hasilPerankingan = $this->sorting_vektor_v();
        }
        return view('dashboard.hasil_saw.hasil', compact(['hasilPerankingan',
'checkHasEmptyData']));
    }

    protected function persentase_bobot()
    {
        $allBobot = DB::table('kriteria')->get();
        $totalBobot = $allBobot->sum('bobot');
        $result = [];

        foreach ($allBobot as $item) {
            // Kali 10 agar total menjadi bobot = 10
            $resultBobot = $item->bobot / $totalBobot * 10;
            $result[] = ['id' => $item->id, 'name' => $item->name, 'type' => $item-
>type, 'bobot' => $item->bobot, 'persentase_bobot' => $resultBobot];
        }

        return $result;
    }

    protected function matrix_ternormalisasi($arrayNilaiBobotByAlternatifId =
[])
    {
        $result = [];

        foreach ($arrayNilaiBobotByAlternatifId as $item) {
            $selectedNilaiBobot = DB::table('nilai_bobot')
                ->where('alternatif_id', '=', $item->alternatif_id)
                ->orderBy('kriteria_id')
                ->get();

            $result[] = $this->min_max($selectedNilaiBobot);
        }

        return $result;
    }

    protected function min_max($arraySelectedNilaiBobot = [])
    {
        $nilaiBobotGroupByKriteriaId = DB::table('nilai_bobot')
            ->join('kriteria', 'nilai_bobot.kriteria_id', '=', 'kriteria.id')
            ->select('kriteria_id', 'kriteria.name', 'kriteria.type',
DB::raw('MAX(nilai) AS max_nilai_kriteria'), DB::raw('MIN(nilai) AS
min_nilai_kriteria'))
            ->orderBy('kriteria_id')
            ->groupBy('kriteria_id')
            ->get();
        $result = [];

        for ($i = 0; $i < count($nilaiBobotGroupByKriteriaId); $i++) {
            // nilai / MAX VALUE
            if ($nilaiBobotGroupByKriteriaId[$i]->type == 'benefit') {
                $valueR = $arraySelectedNilaiBobot[$i]->nilai /
$nilaiBobotGroupByKriteriaId[$i]->max_nilai_kriteria;
                $result[] = ['krit_name' => $nilaiBobotGroupByKriteriaId[$i]->name,
'value_r' => $valueR];
            }

            // MIN VALUE / nilai
            if ($nilaiBobotGroupByKriteriaId[$i]->type == 'cost') {
                $valueR = $nilaiBobotGroupByKriteriaId[$i]->min_nilai_kriteria /
$arraySelectedNilaiBobot[$i]->nilai;
                $result[] = ['krit_name' => $nilaiBobotGroupByKriteriaId[$i]->name,
'value_r' => $valueR];
            }
        }

        return $result;
    }

    protected function calculate_ranking()
    {
        $nilaiBobotGroupByAlternatifId = DB::table('nilai_bobot')
            ->join('alternatif', 'nilai_bobot.alternatif_id', '=', 'alternatif.id')
            ->select('nilai_bobot.alternatif_id', 'alternatif.code',
'alternatif.kode_database')
            ->orderBy('nilai_bobot.alternatif_id')
            ->groupBy('nilai_bobot.alternatif_id')
            ->get();
        $persentaseBobot = $this->persentase_bobot();
        $matrixTernormalisasi = $this-
>matrix_ternormalisasi($nilaiBobotGroupByAlternatifId);
        $result = [];

        for ($i = 0; $i < count($nilaiBobotGroupByAlternatifId); $i++) {
            // 0 Karena Penjumlahan
            $vektorV = 0;

            for ($j = 0; $j < count($persentaseBobot); $j++) {
                $vektorV += $persentaseBobot[$j]['persentase_bobot'] *
$matrixTernormalisasi[$i][$j]['value_r'];
            }
            $result[] = ['alternatif_code' => $nilaiBobotGroupByAlternatifId[$i]-
>code, 'kode_database' => $nilaiBobotGroupByAlternatifId[$i]-
>kode_database, 'vektor_v' => $vektorV];
        }

        return $result;
    }

    protected function sorting_vektor_v()
    {
        $result = $this->calculate_ranking();
        $selectedSorting = array_column($result, 'vektor_v');
        array_multisort($selectedSorting, SORT_DESC, $result);

        return $result;
    }

    // Get From NilaiBobotController with some Edit
    protected function check_nilai_bobot_has_empty_data()
    {
        $condition = false;
        $EMPTY_VALUE = 'Empty';
        $allKriteria = Kriteria::all();
        $nilaiBobotGroupByAlternatifId = DB::table('nilai_bobot')
            ->join('alternatif', 'nilai_bobot.alternatif_id', '=', 'alternatif.id')
            ->select('alternatif_id', 'code', 'kode_database',)
            ->orderBy('alternatif.code')
            ->groupBy('alternatif.id')
            ->get();

        // Array For Compare to Data Nilai Bobot
        $arrayKriteriaIdFromKriteria = [];
        foreach($allKriteria as $item) {
            $arrayKriteriaIdFromKriteria[] = $item->id;
        }

        foreach ($nilaiBobotGroupByAlternatifId as $item) {
            $dataKriteria = [];

            $selectedNilaiBobot = DB::table('nilai_bobot')
                ->where('alternatif_id', '=', $item->alternatif_id)
                ->orderBy('kriteria_id')
                ->get();

            // Array for Compare to Data Kriteria ID
            $arrayKriteriaIdFromSelectedNilaiBobot = [];
            foreach ($selectedNilaiBobot as $itemKriteria) {
                $arrayKriteriaIdFromSelectedNilaiBobot[] = $itemKriteria-
>kriteria_id;
                $dataKriteria[] = ['kriteria_id' => $itemKriteria->kriteria_id, 'nilai' =>
$itemKriteria->nilai];
            }

            // Comparing Data Kriteria ID & Nilai Bobot
            $emptyKriteriaId = array_diff($arrayKriteriaIdFromKriteria,
$arrayKriteriaIdFromSelectedNilaiBobot);
            foreach($emptyKriteriaId as $kriteriaId) {
                $dataKriteria[] = ['kriteria_id' => $kriteriaId, 'nilai' =>
$EMPTY_VALUE];
                $condition = true;
                break;
            }
        }

        return $condition;
    }
}
```

```
<?php
namespace App\Http\Controllers;
use App\Models\Alternatif;
use App\Models\Kriteria;
use App\TOPSIS;

class HasilTOPSISController extends Controller
{
    function index()
    {
        $data['rel_alternatif'] = get_rel_alternatif();
        $kriteria = Kriteria::all();
        $atribut = array();
        $bobot = array();
        foreach ($kriteria as $row) {
            $atribut[$row->id] = $row->atribut;
            $bobot[$row->id] = $row->bobot;
            $data['kriterias'][$row->id] = $row;
        }
        $topsis = new TOPSIS($data['rel_alternatif'], $bobot, $atribut);
        $data['categories'] = [];
        $data['series'][0] = [
            'name' => 'Total',
            'data' => [],
        ];
        foreach ($topsis->pref as $key => $val) {
            $alternatif = Alternatif::find($key);
            $alternatif->total = $val;
            $alternatif->rank = $topsis->rank[$key];
            $alternatif->save();
            $data['categories'][$key] = $alternatif->nama_alternatif;
            $data['series'][0]['data'][$key] = $val * 1;
        }
        $data['categories'] = array_values($data['categories']);
        $data['series'][0]['data'] = array_values($data['series'][0]['data']);
        $data['topsis'] = $topsis;
        $data['alternatifs'] = get_alternatif();
        $data['title'] = 'Perhitungan';
        return view('dashboard.hasil_topsis.index', $data);
    }
    function hasil()
    {
        $data['rel_alternatif'] = get_rel_alternatif();
        $kriteria = Kriteria::all();
        $atribut = array();
        $bobot = array();
        foreach ($kriteria as $row) {
            $atribut[$row->id] = $row->atribut;
            $bobot[$row->id] = $row->bobot;
            $data['kriterias'][$row->id] = $row;
        }
        $topsis = new TOPSIS($data['rel_alternatif'], $bobot, $atribut);
        $data['categories'] = [];
        $data['series'][0] = [
            'name' => 'Total',
            'data' => [],
        ];
        foreach ($topsis->pref as $key => $val) {
            $alternatif = Alternatif::find($key);
            $alternatif->total = $val;
            $alternatif->rank = $topsis->rank[$key];
            $alternatif->save();
            $data['categories'][$key] = $alternatif->nama_alternatif;
            $data['series'][0]['data'][$key] = $val * 1;
        }
        $data['categories'] = array_values($data['categories']);
        $data['series'][0]['data'] = array_values($data['series'][0]['data']);
        $data['topsis'] = $topsis;
        $data['alternatifs'] = get_alternatif();
        $data['title'] = 'Perhitungan';
        return view('dashboard.hasil_topsis.hasil', $data);
    }
}
```