Lampiran 3 *Source Code*

- Front End

Halaman root front end

```
 <script setup lang="ts">
import NaiveUiProvider from
"~/components/NaiveUiProvider.vue"
</script>

<template>
  <NaiveUiProvider>
    <NDialogProvider>
      <AppProvider></AppProvider>
    </NDialogProvider>
  </NaiveUiProvider>
</template>

<style>
@import url("./assets/main.css");
</style>
```

Halaman beranda

```
<script setup lang="ts">
const router = useRouter()
const auth = useAuthStore()

const isLoggedIn = computed(() => auth.data.token !==
"")
const handleClick = () => {
  if (isLoggedIn.value) {
    router.push({ name: "home" })
  } else {
    router.push({ name: "login" })
  }
}
</script>

<template>
  <div class="absolute w-full h-full flex justify-
center animate__animated animate__fadeIn ease-out
wrapper">
```

```
<h1 class="text-4xl">
          <img class="mr-3" src="/assets/socio-map-
logo.svg" width="32" alt="" srcset="" />
          <span class="text-[#079cadff]">Socio</span>
<span class="text-[#e7ac25ff]">Map</span>
        </h1>
        <div class="text-6xl font-bold w-full mb-4
animate__fadeInLeft ease-out">
          Layanan Manajemen Data Socio Map
        </div>
        <div class="text-xl pr-4 mb-6">
          Selamat datang di Socio Map, platform yang
menyediakan layanan manajemen data kesejahteraan
sosial
          terintegrasi. Dengan Socio Map, Anda dapat
mengakses informasi lengkap mengenai profil, kondisi
          sosial, ekonomi, dan tingkat kesejahteraan
seluruh penduduk secara cepat dan mudah. Platform ini
          dirancang untuk membantu program
perlindungan sosial, pemberdayaan masyarakat, serta
mendukung upaya
          penghapusan kemiskinan ekstrem.
        </div>
        <div class="">
          <div class="animate__slideInRight duration-
300 ease-in">
            <NButton color="#079cadff" class="py-5 px-
6" @click="handleClick">
              <template #icon>
                <NIcon>
                  <div v-if="isLoggedIn" class="i-
carbon:map text-sm"></div>
                    <div v-else class="i-carbon:login
text-sm"></div>
                </NIcon>
              </template>
              <span class="text-lg">{{ isLoggedIn ?
"Go to Home" : "Login" }}</span>
            </NButton>
          </div>
        </div>
      </div>
```

UNIVERSITAS
DARMA PERSADA

```
if (err?.response?.status === 401 ||
err?.response?.status === 400) {
      window.$message.error("Email / Password tidak
sesuai")
    } else {
      console.error(e)
      window.$message.error("Server sedang tidak
tersedia")
    }
  } finally {
    isLoading = false
  }
}
</script>

<template>
  <NCard class="p-5">
    <template #header>
      <div class="text-2xl">
        <span class="text-[#045560ff]">Socio</span>
<span class="text-[#e7ac25ff]">Map</span>
      </div>
    </template>

    <NForm
      class="login-form my-5"
      ref="loginForm"
      :model="loginModel"
      :rules="loginRules"
      :disabled="isLoading"
      @keyup.enter="login"
    >
      <NFormItem path="username" label="Username">
        <NInput size="large" v-
model:value="loginModel.username" class="rounded-lg"
placeholder="Username" />
      </NFormItem>
```

Halaman *login*

```
<script setup lang="ts">
import type { HTTPError } from "ky"
import type { FormInst } from "naive-ui"
import ButtonGradient from
"~/components/common/button/ButtonGradient.vue"

const router = useRouter()
const route = useRoute()

const loginForm = $ref<FormInst | null>()
const loginModel = reactive({
  username: import.meta.env.VITE_USER_DEV ?? "",
  password: import.meta.env.VITE_PASSWORD_DEV ?? ""
})

if (import.meta.env.PROD) {
  loginModel.username = ""
  loginModel.password = ""
}

const loginRules = {
  username: {
    required: true,
    message: "Username is required",
    trigger: "blur"
  },
  password: {
    required: true,
    message: "Password is required",
    trigger: "blur"
  }
}

let isLoading = $ref(false)
const auth = useAuthStore()

async function login() {
  isLoading = true
```

Halaman Peta

```
const mapId = "map-home"
let map = $shallowRef<Mapl>()
provideMap(mapId, $$(map))

const { zoomToPoints3d } = useMapZoom($$(map))

const navState = useNavigationState()
const { filterMenuOptionSelected, filterMenuKeySelected
} = storeToRefs(navState)

const mUiState = useMaplUiState()

const { tooltip, pickedInfo, currentDataset,
currentRegion, defaultRegion } = mUiState
const { chartMode, householdVisible } =
storeToRefs(mUiState)

const nProgress = useNProgress()

let columnLayer: MapboxLayer<ColumnLayer<LayerData>>
let heatmapLayer: MapboxLayer<HeatmapLayer<LayerData>>
let boundaryLayer: MaplBaseLayer
const getCentroid = useGetCentroid()
const aggregateApi = new AggregateApi()
const scale = useDivergingColor().classes(6)

const { searchRegionVisible } = storeToRefs(mUiState)
const currentData = reactive({
  layerData: [] as LayerData[],
  object: {} as Record<string, LayerData>,
  stat: {} as {
    min: number
    max: number
    sum: number
  }
```

```
let loadingResetButton = $ref(false)
const searchRegionRef = ref<typeof
SearchByRegionCard>()

const pageIsReady = ref(false)

const changeDataByContent = async (content: Contents,
filter?: Filter, year?: number): Promise<void> => {
  nProgress.start()
  householdVisible.value = false

  currentDataset.filter = filter
  navState.currentMenu.filter = filter
  navState.currentMenu.year = year
  console.log(content, "CONTENT")

  if (content.id === 1 && filter && filter?.id !== 0)
{
    content.id = 2
    content.unit = "persen"
  }

  if (content.id === 2 && filter && filter?.id === 0)
{
    content.id = 1
    content.unit = "keluarga"
  }

  navState.currentMenu.content = content
  currentDataset.menu = content

  pickedInfo.id = undefined
  pickedInfo.regionCode = ""
  pickedInfo.regionName = ""

  tooltip.visible = false
  pickedInfo.visible = false
```

```
const resetLevel = async () => {
  loadingResetButton = true
  console.log(defaultRegion.levelId, "TEST")

  if (defaultRegion.regionCode !== "0") {
    await changeRegionLevel(defaultRegion.levelId,
defaultRegion.regionCode, navState.currentMenu.year)
  } else {
    await changeRegionLevel(defaultRegion.levelId)
  }

  if (navState.filterMenuOptions.length === 1) {
    filterMenuKeySelected.value =
navState.filterMenuOptions[0].name
  }

 loadingResetButton = false
}

const changeRegionData = async (content: Contents,
filter?: Filter, year?: number) => {
  let levelId = currentRegion.levelId
  let regionCode = currentRegion.regionCode

  let aggregates: ApiResponse<RegionInfo>
  let centroids: FeatureCollection<Point>

  if (regionCode === "0" && levelId === 4 &&
chartMode.value === MapChartMode.COLUMN) {
    chartMode.value = MapChartMode.HEATMAP
  }

  if (chartMode.value === MapChartMode.HEATMAP) {
    levelId = 4
    regionCode = defaultRegion.regionCode
```

```
  try {
    const results = await Promise.all([
      aggregateApi.getContentAggregates({
        content_id: content.id,
        region_level: levelId,
        region_id: regionCode,
        ...(filter?.id !== 0 && { filter: filter?.id
}),
        ...(year && { year })
      }),
      getCentroid(levelId, regionCode)
    ])

    aggregates = results[0]
    centroids = results[1]
  } catch (e) {
    nProgress.done()
    return
  }
currentDataset.filter = filter

  const aggregateObject = objectify(aggregates.result,
(d) => d.region_name.toString().toUpperCase())

  currentData.stat = aggregates.stat

  currentData.layerData = centroids.features.map((d)
=> {
    const p = d.properties!

    let regionName = ""
    if (levelId === 1 || levelId === 2) {
      regionName = p.name
    } else if (levelId === 3 || levelId === 4) {
      if
(p.NAMOBJ.toUpperCase().includes("KELURAHAN")) {
        regionName =
p.NAMOBJ.toUpperCase().replace("KELURAHAN", "").trim()
```

```
   const value = aggregateObject[regionName]?.value
    const color = scale(minMaxScale(value,
currentData.stat.min, currentData.stat.max))
    const id = d.id?.toString() ?? ""
    return {
      id,
      position: d.geometry.coordinates as [number,
number],
      value,
      region_code: id,
      region_name: regionName,
      color: color.hex(),
      colorRgb: color.rgb()
    }
  })

  currentData.object =
objectify(currentData.layerData, (d) =>
d?.region_code)

  tooltip.suffix = (content.unit?.toLowerCase() ===
"persen" ? "%" : content.unit) ?? ""
  tooltip.label = content.name ?? ""

  if (chartMode.value === MapChartMode.COLUMN) {
    changeColumnData()
    return
  }

  if (levelId !== 4) {
    await changeRegionLevel(levelId, regionCode,
year)
  }

  changeHeatmapData()
}

const mapColor = useMapColor()
const highlightColor =
chroma(mapColor.state.hover.color).rgb()
```

```
  if (val === 0) {
    elevationScale = minMaxScale(val, 0, max)
  } else {
    elevationScale =
REGION_LEVEL_RADIUSCOL[currentRegion.levelId] * 10
  }

  // @ts-expect-error
  const props: ColumnLayerProps<LayerData> = {
    data: currentData.layerData,
    elevationScale,
    radius:
REGION_LEVEL_RADIUSCOL[currentRegion.levelId],
    opacity: 0.8,
    visible: true,

    getFillColor: (d: any) => {
      if (pickedInfo.id === d.id) {
        return [...highlightColor, 180]
      }

      return d.colorRgb ?? [0, 0, 0]
    },

    getElevation: (d: any) => {
      return d.value != null ? minMaxScale(d.value, 0,
max) : 0
    }
  }

  columnLayer.setProps(props)
  zoomToPoints3d(currentData.layerData.map((d) =>
d.position))
  chartMode.value = MapChartMode.COLUMN
}

const changeBoundaryData = () => {
  zoomToPoints3d(currentData.layerData.map((d) =>
d.position))
```

```
const { layer: fl } = useRegionExtrudeLayer(map!,
toRaw(currentData))
  if (boundaryLayer) {
    map?.removeLayer(boundaryLayer.layerId)
  }

  boundaryLayer = fl

  chartMode.value = MapChartMode.REGION
}

const changeHeatmapData = () => {
  const { max } = currentData.stat

  // @ts-expect-error
  const props: HeatmapLayerProps<LayerData> = {
    data: currentData.layerData,
    radiusPixels: 75,
    aggregation: "SUM",
    visible: true,
    getPosition: (d: any) => d.position,
    getWeight: (d: any) => {
      return d.value != null ? minMaxScale(d.value, 0,
max) :0
    }
  }

  if (heatmapLayer) {
    map?.removeLayer(heatmapLayer.id)
  }

  heatmapLayer = useHeatmapLayer(map!)
  // @ts-expect-error
  heatmapLayer.setProps(props)
  zoomToPoints3d(currentData.layerData.map((d) =>
d.position))
  chartMode.value = MapChartMode.HEATMAP
}
```

```
const changeChartMode = (value: number) => {
  chartMode.value = value

  // @ts-expect-error
  columnLayer.setProps({
    visible: value === MapChartMode.COLUMN
  })

  if (heatmapLayer) {
    // @ts-expect-error
    heatmapLayer.setProps({
      visible: value === MapChartMode.HEATMAP
    })
  }

  switch (value) {
    case MapChartMode.COLUMN:
      resetLevel()
      break
    case MapChartMode.HEATMAP:
      changeRegionLevel(4, defaultRegion.regionCode,
navState.currentMenu.year)
      break
  }
}

const changeRegionLevel = async (regionLevel: number,
regionCode?: string, year?: number) => {
  currentRegion.levelId = chartMode.value ===
MapChartMode.HEATMAP ? 4 : regionLevel
  currentRegion.regionCode = regionCode ?? "0"
  navState.currentMenu.regionCode = regionCode ?? "0"
  navState.currentMenu.year = year
  console.debug(chartMode.value, regionLevel,
regionCode, year, "changeRegionLevel")
```

```
const onStartChangeRegion = async (menu: Contents,
filter?: Filter, regionCode?: string, year?: number)
=> {
  await until(pageIsReady).toBe(true)
  console.debug(menu, filter, regionCode, year,
"onStartChangeRegion")

  if (regionCode != null && regionCode !== "0") {
    currentDataset.menu = menu
    currentDataset.filter = filter
    const code = currentRegion.levelId === 0 ? "0" :
regionCode
    await changeRegionLevel(currentRegion.levelId,
code, year)
  } else {
    await changeDataByContent(menu, filter, year)
  }
}

onMounted(async () => {
  map = useMap3d()

  await onMapLoad(map)
  useMaplState(useMapId(map))()

  columnLayer = useColumnLayer(map)

  map.on("click", (e: any) => {
    const outside =
      // @ts-expect-error
      map!.__deck.pickObject({
        x: e.point.x,
        y: e.point.y
      }) == null

    if (outside) {
      pickedInfo.visible = false
    }
  })

  pageIsReady.value = true
```

```
onBeforeUnmount(() => {
  map?.remove()
  navState.$reset()
})
</script>

<template>
  <DefaultLayout
    @change-menu="
      (content, filterMenu, year) => {
        changeDataByContent(content, filterMenu, year)
      }
    "
    @change-menu-region="
      (menu, filterMenu, code, year) => {
        onStartChangeRegion(menu, filterMenu, code,
year)
      }
    "
    @reset-level="resetLevel"
  >
    <div class="absolute z-50 w-1/4" style="top: 6rem;
left: 2rem">
      <Transition name="slide-right" mode="out-in">
        <template v-if="searchRegionVisible">
          <SearchByRegionCard
            :allowed-region="defaultRegion.regionCode"
            :default-selected-
region="navState.currentMenu.regionCode"
            ref="searchRegionRef"
            @close="searchRegionVisible = false"
          ></SearchByRegionCard>
        </template>
      </Transition>
    </div>
    <div id="side-button__left" class="absolute z-10"
style="left: 0; bottom: 1rem">
      <div
        class="side-button side-button__expand mb-3"
        style="border-radius: 0 var(--border-radius-
xs) var(--border-radius-xs) 0"
      >
```

```
 <Transition name="slide-left" mode="out-in">
      <div v-if="pickedInfo.visible"
class="absolute z-100" style="bottom: 7rem;
right: 1rem">
        <DetailCard
          :region-id="pickedInfo.id ||
currentRegion.regionCode"
          :content="navState.currentMenu.content"
          :is-micro="currentRegion.levelId > 4"
          :filter="filterMenuOptionSelected"
          :current-value="pickedInfo.value"
          @change-region-level="changeRegionLevel"
          @close="pickedInfo.visible = false"
        ></DetailCard>
      </div>

      <div
        v-else-if="!pickedInfo.visible && !['',
'0'].includes(pickedInfo.regionCode)"
        class="absolute z-10 bg-glass-white px-2
py-2"
        style="top: calc(8rem + 150px); right: 0;
border-radius: 1rem 0 0 1rem"
      >
        <NButton type="primary" secondary circle
@click="pickedInfo.visible = true">
          <template #icon>
            <div class="i-ph:info-duotone"></div>
          </template>
        </NButton>
      </div>
    </Transition>

    <div class="map--wrapper">
      <div class="absolute z-10 animate__animated
animate__fadeInRight w-1/6" style="top: 1rem;
right: 1rem">
        <Legend :min="currentData.stat.min"
:max="currentData.stat.max" />
      </div>
```

```
<span v-if="defaultRegion.regionCode === '0'">Kembali
ke level provinsi</span>
              <span v-else>
                Kembali ke <span class="font-
semibold">{{ defaultRegion.regionName }}</span>
              </span>
            </NButton>
          </div>
        </template>
      </Transition>

      <div class="menu__bottom-right mr-7
animate__animated animate__fadeInRight" style="bottom:
0.75rem">
        <BasemapManagerMgl v-if="map" :map="map" />
      </div>
    </div>
  </DefaultLayout>
</template>

<style>
@import "maplibre-gl/dist/maplibre-gl.css";
</style>

<route>
{
meta: {
requiresAuth: true
}
}
</route>
```

Back End

Endpoint Peta Batas Administrasi

```
def process_urban_village_name(urban_village_name):
    if not urban_village_name:
        return None
    urban_village_name_upper =
urban_village_name.upper()
    return
urban_village_name_upper.replace("KELURAHAN",
"").strip()

def load_geojson(file_path):
    if not os.path.isfile(file_path):
        raise HTTPException(status_code=404,
detail="File not found")

    with open(file_path, "r") as f:
        return json.load(f)

def get_feature_centroid(feature):
    coordinates = feature["geometry"]["coordinates"]
    if feature["geometry"]["type"] == "MultiPolygon":
        coordinates = coordinates[0][0]
    elif feature["geometry"]["type"] == "Polygon":
        coordinates = coordinates[0]

    xs, ys = zip(*[(coord[0], coord[1]) for coord in
coordinates])
    centroid_x = sum(xs) / len(xs)
    centroid_y = sum(ys) / len(ys)
    return {"type": "Point", "coordinates":
[centroid_x, centroid_y]}

def get_feature_vertices(feature):
    coordinates = feature["geometry"]["coordinates"]
    if feature["geometry"]["type"] == "MultiPolygon":
        coordinates = coordinates[0][0]
    elif feature["geometry"]["type"] == "Polygon":
        coordinates = coordinates[0]
    return coordinates

def filter_features_by_name(features, name):
```

UNIVERSITAS
DARMA PERSADA

```
@router.get("/centroid")
@cache(namespace="centroid", expire=60 * 60 * 24)
def get_centroids(
    db: Session = Depends(deps.get_db),
    region_level: int = 1,
    parent_id: str = "0",
    period: int = None,
):
    geojson_file_map = {
        1: "bataskotajakarta.geojson",  # provinsi
        2: "bataskotajakarta.geojson",  # kabupaten
        3: "bataskecamatanjakarta.geojson",  #
kecamatan
        4: "batasdesajakarta.geojson",  # kelurahan
    }

    if region_level not in geojson_file_map:
        raise HTTPException(status_code=400,
detail="Invalid region level")

    geojson_path = os.path.join(geojsons_dir,
geojson_file_map[region_level])
    geojson_data = load_geojson(geojson_path)

    total_family_data: schemas.Aggregate =
crud.get_total_family_by_region_level(
        db, region_level, parent_id
    )

    logger.info("Total Family Data: %s",
total_family_data)

    if not geojson_data["features"]:
        raise HTTPException(status_code=404,
detail="GeoJSON features not found")

    features_to_process = geojson_data["features"]

    if parent_id and parent_id != "0":
        if region_level == 3:
            sub_districts =
```

```
            features_to_process = [
                feature
                for feature in
geojson_data["features"]
                if

str(feature["properties"].get("NAMOBJ")).upper()
                in sub_district_names
            ]
elif region_level == 4:
            urban_villages =
crud.get_urban_villages(db,
sub_district_id=str(parent_id))
            urban_village_names = [
                urban_village.name for urban_village
in urban_villages
            ]
            features_to_process = [
                feature
                for feature in
geojson_data["features"]
                if
process_urban_village_name(feature["properties"].get("
WADMKD"))
                in urban_village_names
            ]

    if region_level == 1:
        features_to_process = features_to_process[:1]

    features = []
    for feature in features_to_process:
        feature_name = ""

        if region_level == 1 or region_level == 2:
            feature_name =
feature["properties"].get("name")
        elif region_level == 3:
            feature name =
```

```
centroid = get_feature_centroid(feature)
        vertices = get_feature_vertices(feature)

        if region_level == 4:
            region = crud.get_urban_village_by_name(db,
feature_name)
        elif region_level == 3:
            region = crud.get_sub_district_by_name(db,
feature_name)
        elif region_level == 2 or region_level == 1:
            region = crud.get_district_by_name(db,
feature_name)
        else:
            region = None

        if not region:
            continue

        # Find the corresponding total_family value
        total_family_value = next(
            (
                item.value
                for item in total_family_data.result
                if item.region_name.lower() in
feature_name.lower()
            ),
            0,
        )

        updated_properties = feature["properties"].copy()
        if parent_id and parent_id != "0":
            updated_properties["parent_id"] = parent_id
        else:
            updated_properties["parent_id"] = "0"

        feature_data = {
            "type": "Feature",
            "geometry": centroid,
            "id": str(region.id),
            "properties": updated_properties,
        }
        features.append(feature_data)

    return JSONResponse(content=response)
```

```
@router.get("/",
response_model=Page[schemas.ClusteringResultWithUrban
DataPaginate])
def get_clustering_results(
    cluster: int = None,
    region_code_id: int = None,
    kelurahan_desa: str = None,
    kecamatan: str = None,
    kota_kabupaten: str = None,
    year: int = None,  # Added year filter
    skip: int = Query(default=0, ge=0),
    limit: int = Query(default=None),  # Allow limit
to be None
    db: Session = Depends(deps.get_db),
    # current_user: schemas.User =
Depends(deps.get_current_active_user)
):
    filters = {}
    if cluster is not None:
        filters['cluster'] = cluster
    if region_code_id is not None:
        filters['region_code_id'] = region_code_id
    if kelurahan_desa is not None:
        filters['kelurahan_desa'] = kelurahan_desa
    if kecamatan is not None:
        filters['kecamatan'] = kecamatan
    if kota_kabupaten is not None:
        filters['kota_kabupaten'] = kota_kabupaten
    if year is not None:  # Apply year filter
        filters['tahun'] = year

    total_count, clustering_results =
crud.get_clustering_results_with_count(db, skip=skip,
limit=limit, **filters)
    results = [
        schemas.ClusteringResultWithUrbanVillageName(
            id=result.ClusteringResult.id,

urban_village_name=result.urban_village_name,

sub district name=result.sub district name,
```

```
    jumlah_kepala_keluarga=result.ClusteringResult.juml
    ah_kepala_keluarga,

    jumlah_penduduk=result.ClusteringResult.jumlah_pend
    uduk,

    luas_wilayah=result.ClusteringResult.luas_wilayah,

    kepadatan=result.ClusteringResult.kepadatan,
                tahun=result.ClusteringResult.tahun,

    persentase_keluarga_kurang_gizi=result.ClusteringRe
    sult.persentase_keluarga_kurang_gizi,

    persentase_keluarga_hunian_layak=result.ClusteringR
    esult.persentase_keluarga_hunian_layak,

    persentase_keluarga_dengan_anggota_keluarga_stuntin
    g=result.ClusteringResult.persentase_keluarga_denga
    n_anggota_keluarga_stunting,

    persentase_keluarga_berpendapatan_rendah=result.Clu
    steringResult.persentase_keluarga_berpendapatan_ren
    dah,

    persentase_keluarga_berpendidikan_rendah=result.Clu
    steringResult.persentase_keluarga_berpendidikan_ren
    dah,

    region_code_id=result.ClusteringResult.region_code_
    id,
                geojson=None
            ) for result in clustering_results
        ]

        return paginate(results)

    @router.get("/by-urban-villages",
    response_model=schemas.ClusteringResultWithDsitrict
    Data2)
    def get_clustering_results_urban_villages(
        cluster: int = None,
        region_code_id: int = None,
        kelurahan_desa: str = None
```

UNIVERSITAS
DARMA PERSADA

```
    db: Session = Depends(deps.get_db),
        # current_user: schemas.User =
    Depends(deps.get_current_active_user)
    ):
    filters = {}
    if cluster is not None:
        filters['cluster'] = cluster
    if region_code_id is not None:
        filters['region_code_id'] = region_code_id
    if kelurahan_desa is not None:
        filters['kelurahan_desa'] = kelurahan_desa
    if kecamatan is not None:
        filters['kecamatan'] = kecamatan
    if kota_kabupaten is not None:
        filters['kota_kabupaten'] = kota_kabupaten
    if year is not None:  # Apply year filter
        filters['tahun'] = year

    total_count, clustering_results =
crud.get_clustering_results_with_count(db, skip=skip,
limit=limit, **filters)
    results = [
        schemas.ClusteringResultWithUrbanVillageName(
            id=result.ClusteringResult.id,

urban_village_name=result.urban_village_name,

sub_district_name=result.sub_district_name,
            district_name=result.district_name,
            cluster=result.ClusteringResult.cluster,

jumlah_kepala_keluarga=result.ClusteringResult.jumlah_
kepala_keluarga,

jumlah_penduduk=result.ClusteringResult.jumlah_pendudu
k,

luas_wilayah=result.ClusteringResult.luas_wilayah,

kepadatan=result.ClusteringResult.kepadatan,
            tahun=result.ClusteringResult.tahun,

persentase_keluarga_lansia=result.ClusteringResult.per
sentase keluarga lansia,
```

```
region_code_id=result.ClusteringResult.region_code_id,
            geojson=None
        ) for result in clustering_results
    ]

    geo_json_district_path =
os.path.join(geo_json_dir, 'batasdesajakarta.geojson')
    with open(geo_json_district_path, 'r') as f:
        geojson_data = json.load(f)

    results_with_geojson = []
    for feature in geojson_data['features']:
        wadmpr = feature['properties'].get('WADMPR')
        if(wadmpr != "Jawa Barat" and wadmpr != "Jawa
Tengah" and wadmpr != "Jawa Timur" and wadmpr !=
"Banten"):
            urban_village_name =
feature['properties'].get('NAMOBJ')
            if
urban_village_name.startswith("Kelurahan "):
                urban_village_name =
urban_village_name.replace("Kelurahan ", "").upper()

            for result in results:
                if result.urban_village_name.upper()
== urban_village_name:
                    result_with_geojson =
result.dict()
                    result_with_geojson['geojson'] =
feature

results_with_geojson.append(result_with_geojson)

    return {
        "total_count": total_count,
        "results": results_with_geojson
    }

@router.get("/by-districts",
response_model=schemas.ClusteringResultWithDsitrictDat
a2)
def get_clustering_results_by_district(
    cluster: int = None,
```

```
filters = {}
    if cluster is not None:
        filters['cluster'] = cluster
    if region_code_id is not None:
        filters['region_code_id'] = region_code_id
    if kelurahan_desa is not None:
        filters['kelurahan_desa'] = kelurahan_desa
    if kecamatan is not None:
        filters['kecamatan'] = kecamatan
    if kota_kabupaten is not None:
        filters['kota_kabupaten'] = kota_kabupaten
    try:
        total_count, clustering_results =
crud.get_clustering_results_with_count(db, skip=skip,
limit=limit, **filters)
        results_by_district = defaultdict(lambda:
{'district': None, 'results': []})

        for result in clustering_results:
            clustering_result =
schemas.ClusteringResultWithUrbanVillageName(
                id=result.ClusteringResult.id,

urban_village_name=result.urban_village_name,

sub_district_name=result.sub_district_name,
                district_name=result.district_name,
                cluster=result.ClusteringResult.cluster,

jumlah_kepala_keluarga=result.ClusteringResult.jumlah_ke
pala_keluarga,

jumlah_penduduk=result.ClusteringResult.jumlah_penduduk,

luas_wilayah=result.ClusteringResult.luas_wilayah,

kepadatan=result.ClusteringResult.kepadatan,
                tahun=result.ClusteringResult.tahun,

persentase_keluarga_lansia=result.ClusteringResult.perse
ntase_keluarga_lansia,

persentase_keluarga_kurang_gizi=result.ClusteringResult.
```

```
region_code_id=result.ClusteringResult.region_code_id,
                geojson=None
            )
            district_name = result.district_name

results_by_district[district_name]['district'] =
district_name

results_by_district[district_name]['results'].append(c
lustering_result)
        geo_json_district_path =
os.path.join(geo_json_dir, 'bataskotajakarta.geojson')
        # Load GeoJSON file
        with open(geo_json_district_path, 'r') as f:
            geojson_data = json.load(f)

        for feature in geojson_data['features']:
            district_name =
feature['properties'].get('name').upper()
            if district_name in results_by_district:
                for result in
results_by_district[district_name]['results']:
                    result.geojson = feature

        # Convert defaultdict to list of dictionaries
        results_list =
list(results_by_district.values())
        return {
            "total_count": total_count,
            "results": results_list
        }
    except Exception as e:
        logger.error(f"Terjadi kesalahan: {str(e)}")
        raise HTTPException(status_code=500,
detail=str(e))

@router.get("/by-sub-districts",
response_model=schemas.ClusteringResultWithDsitrictDat
a2)
def get_clustering_results_by_sub_district(
    cluster: int = None,
    region_code_id: int = None,
```

```
 # current_user: schemas.User =
Depends(deps.get_current_active_user)
):
    filters = {}
    if cluster is not None:
        filters['cluster'] = cluster
    if region_code_id is not None:
        filters['region_code_id'] = region_code_id
    if kelurahan_desa is not None:
        filters['kelurahan_desa'] = kelurahan_desa
    if kecamatan is not None:
        filters['kecamatan'] = kecamatan
    if kota_kabupaten is not None:
        filters['kota_kabupaten'] = kota_kabupaten

    total_count, clustering_results =
crud.get_clustering_results_with_count(db, skip=skip,
limit=limit, **filters)
    results_by_sub_district = defaultdict(lambda:
{'sub_district': None, 'results': []})

    for result in clustering_results:
        clustering_result =
schemas.ClusteringResultWithUrbanVillageName(
            id=result.ClusteringResult.id,

urban_village_name=result.urban_village_name,
            sub_district_name=result.sub_district_name,
            district_name=result.district_name,
            cluster=result.ClusteringResult.cluster,

jumlah_kepala_keluarga=result.ClusteringResult.jumlah_ke
pala_keluarga,

jumlah_penduduk=result.ClusteringResult.jumlah_penduduk,

luas_wilayah=result.ClusteringResult.luas_wilayah,
            kepadatan=result.ClusteringResult.kepadatan,
            tahun=result.ClusteringResult.tahun,

persentase_keluarga_lansia=result.ClusteringResult.perse
ntase_keluarga_lansia,

persentase_keluarga_kurang_gizi=result.ClusteringResult.
persentase_keluarga_kurang_gizi,
```

```
region_code_id=result.ClusteringResult.region_code_id,
            geojson=None
        )
        sub_district_name = result.sub_district_name

results_by_sub_district[sub_district_name]['sub_district
t'] = sub_district_name

results_by_sub_district[sub_district_name]['results'].a
ppend(clustering_result)

    geo_json_district_path = os.path.join(geo_json_dir,
'bataskecamatanjakarta.geojson')

    with open(geo_json_district_path, 'r') as f:
        geojson_data = json.load(f)

    for feature in geojson_data['features']:
        sub_district_name =
feature['properties'].get('NAMOBJ').upper()
        if sub_district_name in
results_by_sub_district:
            for result in
results_by_sub_district[sub_district_name]['results']:
                result.geojson = feature

    # Convert defaultdict to list of dictionaries
    results_list =
list(results_by_sub_district.values())

    return {
        "total_count": total_count,
        "results": results_list
    }

@router.post("/uploadfile/")
async def upload_file(file: UploadFile = File(...), db:
Session = Depends(deps.get_db)):
    try:
        # Membaca file CSV
        logger.info("Membaca file CSV")
        df = pd.read_csv(file.file)
```

```
        for index, row in df.iterrows():
            existing_sub_district =
crud.get_sub_district_by_name(db, row['kecamatan'])
            existing_region_code =
crud.get_region_code_by_sub_district_name(db,
existing_sub_district.id)
            existing_urban_village =
crud.get_urban_village_by_name(db,
row['kelurahan_desa'])
            # print(existing_urban_village.name,
existing_sub_district.name, existing_region_code)
            print(row['kepadatan'])
            result = {
                "urban_village_id":
existing_urban_village.id,
                "jumlah_kepala_keluarga":
row['jumlah_kepala_keluarga'],
                "jumlah_penduduk":
row['jumlah_penduduk'],
                "luas_wilayah": row['luas_wilayah'],
                "kepadatan": row['kepadatan'],
                "tahun": row['tahun'],
                "persentase_keluarga_kurang_gizi":
row['persentase_keluarga_kurang_gizi'],
                "persentase_keluarga_hunian_layak":
row['persentase_keluarga_hunian_layak'],

"persentase_keluarga_dengan_anggota_keluarga_stunting"
: round(random.uniform(5.0, 25.0), 2),

"persentase_keluarga_berpendapatan_rendah":
round(random.uniform(10.0, 40.0), 2),

"persentase_keluarga_berpendidikan_rendah":
round(random.uniform(10.0, 30.0), 2),
                "region_code_id":
existing_region_code.id,
            }
            results.append(result)
```

```
# Now assign the dataframe to the model
        kmeans_model.df = pd.DataFrame(results)
        kmeans_model.preprocess_data()
        clustered_df = kmeans_model.fit()
        clustered_df = clustered_df.fillna(0)
        clusters =
clustered_df.to_dict(orient='records')

        logger.info("Hasil pengelompokan diambil
sebagai dictionary records")

# Simpan data ke dalam database
        for cluster in clusters:
            db_cluster =
models.ClusteringResult(**cluster)
            db.add(db_cluster)
        db.commit()

        logger.info("Hasil pengelompokan berhasil
disimpan ke database")

        return JSONResponse(content={"message": "File
processed successfully", "clusters":
jsonable_encoder(cluster)})
    except Exception as e:
        logger.error(f"Terjadi kesalahan: {str(e)}")
        raise HTTPException(status_code=500,
detail=str(e))
```