L
A
M
P
I
R
A
N

SignIn-user.tsx

```
"use server";

import { createClient } from
    "@/utils/supabase/server";

import { redirect } from
    "next/navigation";

export const signInAction =
    async (previousState: {
    success: boolean; message:
    string }, formData:
    FormData) => {

  const email =
    formData.get("email") as
    string;

  const password =
    formData.get("password")
    as string;

  const supabase = await
    createClient();

  if (!email || !password)
    return { success: false,
    message: "Email dan
    password tidak boleh
    kosong" };

  if (password.length < 6)
    return { success: false,
    message: "Password minimal
    6 karakter" };

  const { error } = await
    supabase.auth.signInWithPa
    ssword({

    email,

    password,

  });

  if (error) {

    return { success: false,
    message: error.message };

  }

  return
    redirect("/dashboard/datas
    et");

};
```

Page-singIn.tsx

```
"use client";

import { Alert,
    AlertDescription } from
    "@/components/ui/alert";

import { Button } from
    "@/components/ui/button";

import { Input } from
    "@/components/ui/input";

import { Label } from
    "@/components/ui/label";

import { cn } from
    "@/lib/utils";

import { AlertCircle, Loader2
    } from "lucide-react";

import Link from "next/link";

import { useActionState } from
    "react";

import { signInAction } from
    "../action/signin";

export function SignInForm({
    className, ...props }:
    React.ComponentProps<"div"
    >) {

  const [formState,
    formAction, isPending] =
    useActionState(signInActio
    n, { success: false,
    message: "" });

  return (

    <div className={cn("flex
    flex-col gap-6",
    className)} {...props}>

      <form
```

```
action={formAction}>

    <div className="flex
flex-col gap-6">

      <div className="flex
flex-col items-center gap-
2">

        <h1
className="text-xl font-
bold">Welcome to Skripsi
PAD.</h1>

        <div
className="text-center
text-sm">

          Tidak ada
akun?{" "}

          <Link
href="/register"
className="underline
underline-offset-4">

            Register

          </Link>

        </div>

      </div>

      {formState.message
&& (

        <Alert
variant="destructive">

          <AlertCircle
className="h-4 w-4" />

          <AlertDescriptio
n>{formState.message}</Ale
rtDescription>

        </Alert>

      )}

      <div className="flex
flex-col gap-6">

        <div
className="grid gap-3">

          <Label
htmlFor="email">Email</Lab
el>

          <Input
id="email" name="email"
type="email"
placeholder="Masukkan
email anda" required />

        </div>

        <div
className="grid gap-3">

          <Label
htmlFor="email">Password</
Label>

          <Input
id="email" name="password"
type="password"
placeholder="Masukkan
password" required />

        </div>

        <Button
disabled={isPending}
type="submit"
className="w-full">

          {isPending &&
<Loader2 className="mr-2
h-4 w-4 animate-spin" />}

          Login

        </Button>

      </div>

    </div>

  </form>

</div>

  );

}
```

SignUp-user.tsx

```
"use server";

import { createClient } from
    "@/utils/supabase/server";

import { redirect } from
    "next/navigation";

export const signUpAction =
    async (previousState: {
    success: boolean; message:
    string }, formData:
    FormData) => {

  const email =
    formData.get("email") as
    string;

  const password =
    formData.get("password")
    as string;

  const supabase = await
    createClient();

  console.log(email);

  if (!email || !password)
    return { success: false,
    message: "Email dan
    password tidak boleh
    kosong" };

  if (password.length < 6)
    return { success: false,
    message: "Password minimal
    6 karakter" };

  const { error } = await
    supabase.auth.signUp({

    email,

    password,

  });

  if (error) {

    return { success: false,
    message: error.message };

  }

  return
    redirect("/dashboard/datas
    et");

};
```

Page-SignUp.tsx

```
"use client";

import { Alert,
    AlertDescription } from
    "@/components/ui/alert";

import { Button } from
    "@/components/ui/button";

import { Input } from
    "@/components/ui/input";

import { Label } from
    "@/components/ui/label";

import { cn } from
    "@/lib/utils";

import { AlertCircle, Loader2
    } from "lucide-react";

import Link from "next/link";

import { useActionState } from
    "react";

import { signUpAction } from
    "../action/signup";

export function SignUpForm({
    className, ...props }:
    React.ComponentProps<"div"
    >) {

  const [formState,
    formAction, isPending] =
    useActionState(signUpActio
    n, { success: false,
    message: "" });

  return (

    <div className={cn("flex
    flex-col gap-6",
    className)} {...props}>

      <form
```

```
action={formAction}>

    <div className="flex
flex-col gap-6">

      <div className="flex
flex-col items-center gap-
2">

        <h1
className="text-xl font-
bold">Buat akun baru di
Skripsi PAD.</h1>

        <div
className="text-center
text-sm">

          Sudah punya
akun?{" "}

          <Link href="/"
className="underline
underline-offset-4">

            Login

          </Link>

        </div>

      </div>

      {formState.message
&& (

        <Alert
variant="destructive">

          <AlertCircle
className="h-4 w-4" />

          <AlertDescriptio
n>{formState.message}</Ale
rtDescription>

        </Alert>

      )}

      <div className="flex
flex-col gap-6">

        <div
className="grid gap-3">

          <Label
htmlFor="email">Email</Lab
el>

          <Input
id="email" name="email"
type="email"
placeholder="Masukkan
email anda" required />

        </div>

        <div
className="grid gap-3">

          <Label
htmlFor="email">Password</
Label>

          <Input
id="email" name="password"
type="password"
placeholder="masukkan
password" required />

        </div>

        <Button
disabled={isPending}
type="submit"
className="w-full">

          {isPending &&
<Loader2 className="mr-2
h-4 w-4 animate-spin" />}

          Buat akun

        </Button>

      </div>

    </div>

  </form>

  </div>

 );

}
```

Dataset.tsx

```
import { z } from "zod";
```

```
import prisma from
    "../db/prisma";

import { privateProcedure,
    publicProcedure, router }
    from "../trpc";

export const datasetRouter =
    router({

  get: publicProcedure

    .input(

      z.object({

        tahun: z.string(),

      })

    )

    .query(async ({ input })
    => {

      if (input.tahun) {

        return await
    prisma.dataset.findMany({

          orderBy: {

            createdAt: "desc",

          },

          where: {

            tahun:
    input.tahun,

          },

        });

      }

      return
    prisma.dataset.findMany({

        orderBy: {

          createdAt: "desc",

        },

      });

  }),

create: privateProcedure

  .input(

    z.object({

      bulan: z.string(),

      tahun: z.string(),

      pajak_hotel:
  z.string(),

      pajak_parkir:
  z.string(),

      pajak_hiburan:
  z.string(),

      pajak_reklame:
  z.string(),

    })

  )

  .mutation(async ({ input
  }) => {

    const { tahun, bulan,
  ...dataPajak } = input;

    const totalPajak =
  Object.values(dataPajak)

      .map(Number)

      .reduce((acc, curr) =>
  acc + curr, 0);

    return await
  prisma.dataset.create({

      data: {

        kategori_total_pajak
  : totalPajak < 20000000 ?
  "Rendah" : totalPajak <
  50000000 ? "Sedang" :
  "Tinggi",
```

```
        total_pajak:
  String(totalPajak),
        ...dataPajak,
        tahun,
        bulan,
      },
    });
  }),
update: privateProcedure
  .input(
    z.object({
      id: z.string(),
      bulan: z.string(),
      tahun: z.string(),
      pajak_hotel:
  z.string(),
      pajak_parkir:
  z.string(),
      pajak_hiburan:
  z.string(),
      pajak_reklame:
  z.string(),
    })
  )
  .mutation(async ({ input
  }) => {
    const dataset = await
  prisma.dataset.update({
      where: {
        id: input.id,
      },
        data: {
          ...input,
        },
      });
      return dataset;
    }),
  delete: privateProcedure
    .input(
      z.object({
        id: z.string(),
      })
    )
    .mutation(async ({ input
    }) => {
      const dataset = await
    prisma.dataset.delete({
        where: {
          id: input.id,
        },
      });
      return dataset;
    }),
});
```

Page-dataset.tsx

```
"use client";

import { NavBreadCrumb } from
    "@/components/nav-
    breadcrumb";

import { columns } from
    "@/features/dataset/table/
    columns";
```

```
import { DataTable } from
    "@/features/dataset/table/
    data-table";

import { useTRPC } from
    "@/utils/trpc/client";

import { useQuery } from
    "@tanstack/react-query";

import { useState } from
    "react";

export default function Page()
    {

  const trpc = useTRPC();

  const [tahun, setTahun] =
    useState("");

  const getDataset =
    useQuery(trpc.dataset.get.
    queryOptions({ tahun:
    tahun }));

  return (

    <main className="space-y-
    8">

      <NavBreadCrumb
    current="dashboard/dataset
    " />

      <section
    className="space-y-1 ">

        <h4 className="text-lg
    font-semibold leading-
    none">Dataset</h4>

        <p className="text-sm
    text-muted-
    foreground">View dataset
    pad</p>

      </section>

      <DataTable

        columns={columns}

        data={getDataset?.data
    as []}

        tahun={tahun}

        setTahun={setTahun}

        isLoading={getDataset.
    isLoading}

      />

    </main>

  );

}
```

Mining.tsx

```
import { TRPCError } from
    "@trpc/server";

import { format } from "date-
    fns";

import { id } from "date-
    fns/locale";

import { z } from "zod";

import { handlePrismaError }
    from
    "../db/handleErrorPrisma";

import prisma from
    "../db/prisma";

import { publicProcedure,
    router } from "../trpc";

export const miningRouter =
    router({

  getMiningResult:
    publicProcedure.query(asyn
    c () => {

    const miningResult = await
    prisma.miningResult.findMa
    ny({

      orderBy: {

        taxType: "asc",

      },
```

```
  });

  return miningResult;

}),

createMining:
  publicProcedure

  .input(

    z.object({

      tahun: z.string(),

      pajak_hotel:
  z.string(),

      pajak_parkir:
  z.string(),

      pajak_hiburan:
  z.string(),

      pajak_reklame:
  z.string(),

    })

  )

  .mutation(async ({ input
  }) => {

    try {

      const dataset = await
  prisma.dataset.findMany({

        where: {

          tahun:
  input.tahun,

        },

      });

      if (!dataset.length ||
  dataset.length === 0) {

        throw new
  TRPCError({

          code:
"BAD_REQUEST",

          message: "Tida
data ditemukan",

        });

      }

      const thresholds = {

        pajak_hotel:
parseFloat(input.pajak_hot
el),

        pajak_parkir:
parseFloat(input.pajak_par
kir),

        pajak_hiburan:
parseFloat(input.pajak_hib
uran),

        pajak_reklame:
parseFloat(input.pajak_rek
lame),

      };

      const
calculateForTaxType =
(taxType: keyof typeof
thresholds, threshold:
number) => {

        const categoryLE = {
Low: 0, Medium: 0, High:
0, Total: 0 };

        const categoryGT = {
Low: 0, Medium: 0, High:
0, Total: 0 };

        const totalCount = {
Low: 0, Medium: 0, High:
0, Total: 0 };

        dataset.forEach((dat
a) => {

          const raw =
data[taxType];

          if (!raw) return;
```

```
        const pajakValue =
parseFloat(raw.replace(/,/
g, ""));

        if
(isNaN(pajakValue))
return;

        const category =
data.kategori_total_pajak;

        if (category ===
"Rendah")
totalCount.Low++;

        if (category ===
"Sedang")
totalCount.Medium++;

        if (category ===
"Tinggi")
totalCount.High++;

        totalCount.Total++
;

        if (pajakValue <=
threshold) {

          if (category ===
"Rendah")
categoryLE.Low++;

          if (category ===
"Sedang")
categoryLE.Medium++;

          if (category ===
"Tinggi")
categoryLE.High++;

          categoryLE.Total
++;

        } else {

          if (category ===
"Rendah")
categoryGT.Low++;

          if (category ===
"Sedang")
categoryGT.Medium++;

          if (category ===
"Tinggi")
categoryGT.High++;

          categoryGT.Total
++;

        }

      });

      const
calculateEntropy =
(counts: typeof
categoryLE): number => {

        if (counts.Total
=== 0) return 0;

        return parseFloat(

          (["Low",
"Medium", "High"] as
const)

            .reduce((entro
py, key) => {

              const p =
counts[key] /
counts.Total;

              return p > 0
? entropy - p *
Math.log2(p) : entropy;

            }, 0)

            .toFixed(3)

        );

      };

      const entropyAwal =
calculateEntropy(totalCoun
t);

      const entropyLE =
calculateEntropy(categoryL
E);

      const entropyGT =
calculateEntropy(categoryG
T);
```

```
      const totalData =
totalCount.Total;

      const totalData1 =
categoryLE.Total;

      const totalData2 =
categoryGT.Total;

      const
weightedEntropy =
(totalData1 / totalData) *
entropyLE + (totalData2 /
totalData) * entropyGT;

      const
informationGain =
entropyAwal -
weightedEntropy;

      return {

        taxType,

        threshold,

        categoryLE,

        categoryGT,

        entropyLE,

        entropyGT,

        informationGain:
parseFloat(informationGain
.toFixed(3)),

      };

    };

    const results =
Object.entries(thresholds)
.map(([key, value]) =>

      calculateForTaxType(
key as keyof typeof
thresholds, value)

    );

    await Promise.all(
      results.map((result)
=>

        prisma.miningResul
t.create({

          data: {

            tahun:
`${input.tahun}
${format(new Date(),
"EEEE, dd MMM yyyy HH:mm",
{ locale: id })}`,

            taxType:
result.taxType,

            threshold:
result.threshold,

            categoryLE: {

              Low:
result.categoryLE.Low,

              Medium:
result.categoryLE.Medium,

              High:
result.categoryLE.High,

              Total:
result.categoryLE.Total,

            },

            categoryGT: {

              Low:
result.categoryGT.Low,

              Medium:
result.categoryGT.Medium,

              High:
result.categoryGT.High,

              Total:
result.categoryGT.Total,

            },

            entropyLE:
result.entropyLE,
```

```
                entropyGT:
    result.entropyGT,

                informationGai
    n: result.informationGain,

              },

            })

          )

        );

        return {

          results,

        };

      } catch (error) {

        handlePrismaError(erro
    r);

      }

    }),

});
```

Page-mining.tsz

```
"use client";

import { NavBreadCrumb } from
    "@/components/nav-
    breadcrumb";

import { Tabs, TabsContent,
    TabsList, TabsTrigger }
    from
    "@/components/ui/tabs";

import { PajakHiburan } from
    "@/features/mining/compone
    nts/pajak-hiburan";

import { PajakHotel } from
    "@/features/mining/compone
    nts/pajak-hotel";

import { PajakParkir } from
    "@/features/mining/compone
    nts/pajak-parkir";

import { PajakReklame } from
    "@/features/mining/compone
    nts/pajak-reklame";

import { CreateMiningForm }
    from
    "@/features/mining/form/cr
    eate";

import { useTRPC } from
    "@/utils/trpc/client";

import { useMutation } from
    "@tanstack/react-query";

import { toast } from
    "sonner";

export default function Page()
    {

  const trpc = useTRPC();

  const mining = useMutation(

    trpc.mining.createMining.m
    utationOptions({

      onSuccess: async () => {

        console.log("success")
    ;

      },

      onError: (error) => {

        toast.error(error.mess
    age);

      },

    })

  );

  const getResult = (taxType:
    string) =>
    mining.data?.results.find(
    (item) => item.taxType ===
    taxType);

  console.log(getResult("pajak
    _hotel"));
```

```
return (
  <main className="space-y-
  8">
    <NavBreadCrumb
  current="dashboard/mining"
  />
    <section
  className="space-y-1 ">
      <h4 className="text-lg
  font-semibold leading-
  none">Data Mining</h4>
      <p className="text-sm
  text-muted-
  foreground">View data
  mining</p>
    </section>

    <CreateMiningForm
  mining={mining} />

    <Tabs
  orientation="horizontal"
  activationMode="automatic"
  defaultValue="pajak-
  hotel">
      <TabsList>
        <TabsTrigger
  value="pajak-hotel">Pajak
  Hotel</TabsTrigger>
        <TabsTrigger
  value="pajak-parkir">Pajak
  Parkir</TabsTrigger>
        <TabsTrigger
  value="pajak-
  hiburan">Pajak
  Hiburan</TabsTrigger>
        <TabsTrigger
  value="pajak-
  reklame">Pajak
  Reklame</TabsTrigger>
      </TabsList>
      <TabsContent
  value="pajak-hotel">
        <PajakHotel
  data={getResult("pajak_hot
  el")} />
      </TabsContent>
      <TabsContent
  value="pajak-parkir">
        <PajakParkir
  data={getResult("pajak_par
  kir")} />
      </TabsContent>
      <TabsContent
  value="pajak-hiburan">
        <PajakHiburan
  data={getResult("pajak_hib
  uran")} />
      </TabsContent>
      <TabsContent
  value="pajak-reklame">
        <PajakReklame
  data={getResult("pajak_rek
  lame")} />
      </TabsContent>
    </Tabs>
  </main>
  );
}
```

Decision.tsx

```
/* eslint-disable */

// @ts-ignore

import { TRPCError } from
    "@trpc/server";
```

```
import { z } from "zod";

import { handlePrismaError }
    from
    "../db/handleErrorPrisma";

import prisma from
    "../db/prisma";

import { privateProcedure,
    router } from "../trpc";



// Definisi tipe untuk dataset

interface Dataset {

  pajak_reklame: number;

  pajak_parkir: number;

  pajak_hotel: number;

  pajak_hiburan: number;

  kategori_total_pajak:
    string;

}

// Definisi struktur node
    pohon keputusan

interface TreeNode {

  attribute?: string;

  gain?: number;

  count?: Record<string,
    number>;

  result?: string;

  branches?: { value: string;
    next: TreeNode }[];

}

// ========================
    Algoritma Decision Tree
    ========================

// Fungsi untuk menghitung
    entropy

function
    calculateEntropy(data:
    any[], className: string):
    number {

  const counts: Record<string,
    number> = {};

  let total = data.length;

  // Hitung jumlah setiap
    kelas

  data.forEach((item) => {

    const classValue =
    item[className];

    counts[classValue] =
    (counts[classValue] || 0)
    + 1;

  });

  // Hitung entropy

  return
    Object.values(counts).redu
    ce((entropy, count) => {

    const probability = count
    / total;

    return entropy -
    probability *
    Math.log2(probability);

  }, 0);

}

// Fungsi untuk menghitung
    information gain

function calculateGain(data:
    any[], attribute: string,
    className: string): number
    {

  const totalEntropy =
    calculateEntropy(data,
    className);
```

```
  const attributeValues:
    Record<string, any[]> =
    {};

  // Kelompokkan data
    berdasarkan nilai atribut

  data.forEach((item) => {

    const value =
    item[attribute];

    if
    (!attributeValues[value])
    attributeValues[value] =
    [];

    attributeValues[value].pus
    h(item);

  });

  // Hitung entropy untuk
    setiap subset

  let gainSum = 0;

  Object.values(attributeValue
    s).forEach((subset) => {

    const proportion =
    subset.length /
    data.length;

    gainSum += proportion *
    calculateEntropy(subset,
    className);

  });

  // Hitung gain

  return totalEntropy -
    gainSum;

}

// Fungsi untuk mendapatkan
    atribut dengan gain
    tertinggi

function
    getBestAttribute(data:
    any[], attributes:
    string[], className:
    string): { attribute:
    string; gain: number } {

  let bestAttribute = "";

  let bestGain = -1;

  attributes.forEach((attribut
    e) => {

    const gain =
    calculateGain(data,
    attribute, className);

    if (gain > bestGain) {

      bestGain = gain;

      bestAttribute =
    attribute;

    }

  });

  return { attribute:
    bestAttribute, gain:
    bestGain };

}

// Fungsi untuk membuat pohon
    keputusan secara manual
    dengan struktur tetap

function buildDecisionTree(

  data: any[],

  attributes: string[],

  className: string,

  depth = 0,

  maxDepth = 4,

  fixedOrder: string[] = []

): TreeNode {

  const classCounts:
    Record<string, number> =
    {};
```

```
data.forEach((item) => {

  const classValue =
  item[className];

  classCounts[classValue] =
  (classCounts[classValue]
  || 0) + 1;

});

// Syarat berhenti 1: semua
  data satu kelas

if
  (Object.keys(classCounts).
  length === 1) {

  return {

    result:
  Object.keys(classCounts)[0
  ],

    count: classCounts,

  };

}

// Syarat berhenti 2:
  mencapai kedalaman
  maksimum atau tidak ada
  atribut

if (depth >= maxDepth ||
  attributes.length === 0) {

  return
  getMajorityClass(data,
  className);

}

// Pilih atribut sesuai
  fixedOrder atau atribut
  terbaik

let attribute = "";

let gain = 0;

if (fixedOrder.length >
  depth) {

  attribute =
  fixedOrder[depth]; //
  Paksa pakai urutan fixed

  gain = calculateGain(data,
  attribute, className);

} else {

  const bestAttribute =
  getBestAttribute(data,
  attributes, className);

  attribute =
  bestAttribute.attribute;

  gain = bestAttribute.gain;

}

// HAPUS PENGEcekan GAIN
  KECIL DI SINI

// (Bagian yang
  mengembalikan majority
  class dihapus)

// Kelompokkan data
  berdasarkan nilai atribut

const attributeValues:
  Record<string, any[]> =
  {};

data.forEach((item) => {

  const value =
  item[attribute];

  if
  (!attributeValues[value])
  attributeValues[value] =
  [];

  attributeValues[value].pus
  h(item);

});

// Bangun node dengan
  atribut terpilih

const node: TreeNode = {

  attribute,
```

```
    gain,
    count: classCounts,
    branches: [],
  };
  // Rekursif untuk setiap
    cabang
  Object.entries(attributeValu
    es).forEach(([value,
    subset]) => {
    const subtree =
    buildDecisionTree(
      subset,
      attributes, // Tetap
    kirim semua atribut
      className,
      depth + 1,
      maxDepth,
      fixedOrder
    );
    node.branches!.push({
      value,
      next: subtree,
    });
  });
  return node;
}
// Fungsi bantu untuk
    mendapatkan kelas
    mayoritas
function
    getMajorityClass(data:
    any[], className: string):
    TreeNode {
  const counts: Record<string,
    number> = {};
  data.forEach((item) => {
    counts[item[className]] =
    (counts[item[className]]
    || 0) + 1;
  });
  const majority =
    Object.entries(counts).red
    uce((a, b) => (a[1] > b[1]
    ? a : b))[0];
  return { result: majority,
    count: counts };
}
// Fungsi untuk memprediksi
    kelas dari input
function predict(tree:
    TreeNode, input: any):
    string {
  // Jika node adalah leaf
    node, kembalikan hasilnya
  if (tree.result) {
    return tree.result;
  }
  // Jika node adalah decision
    node, cari cabang yang
    sesuai
  if (tree.attribute &&
    tree.branches) {
    const inputValue =
    input[tree.attribute];
    // Cari cabang yang sesuai
    dengan nilai input
    for (const branch of
    tree.branches) {
```

```
      if (branch.value ===
    inputValue) {

        return
    predict(branch.next,
    input);

      }

    }

    // Jika tidak ada cabang
    yang cocok, ambil cabang
    pertama (atau implementasi
    lain)

    if (tree.branches.length >
    0) {

      return
    predict(tree.branches[0].n
    ext, input);

    }

  }

  // Fallback jika tidak ada
    hasil yang ditemukan

  return "TIDAK_DIKETAHUI";

}

export const decisionRouter =
    router({

  createDecision:
    privateProcedure

    .input(

      z.object({

        tahun:
    z.string().trim().nonempty
    ({ message: "Tahun wajib
    diisi" }),

        pajak_hotel:
    z.string().trim().nonempty
    ({ message: "Pajak hotel
    wajib diisi" }),

        pajak_parkir:
z.string().trim().nonempty
({ message: "Pajak parkir
wajib diisi" }),

    pajak_hiburan:
z.string().trim().nonempty
({ message: "Pajak hiburan
wajib diisi" }),

    pajak_reklame:
z.string().trim().nonempty
({ message: "Pajak reklame
wajib diisi" }),

  })

)

.mutation(async ({ input
}) => {

  try {

    // Ambil data dari
database

    const dataset = await
prisma.dataset.findMany({

      where: {

        tahun:
input.tahun,

      },

    });

    if (!dataset.length) {

      throw new
TRPCError({

        code:
"BAD_REQUEST",

        message: "Tidak
ada data ditemukan",

      });

    }

// Parse nilai input
```

```
    const userInputValues
= {

      pajak_reklame:
parseFloat(input.pajak_rek
lame),

      pajak_parkir:
parseFloat(input.pajak_par
kir),

      pajak_hotel:
parseFloat(input.pajak_hot
el),

      pajak_hiburan:
parseFloat(input.pajak_hib
uran),

    };

    // Ekstrak data
numerik dari dataset

    const cleanedData:
Dataset[] =
dataset.map((row) => ({

      pajak_reklame:
parseFloat(row.pajak_rekla
me),

      pajak_parkir:
parseFloat(row.pajak_parki
r),

      pajak_hotel:
parseFloat(row.pajak_hotel
),

      pajak_hiburan:
parseFloat(row.pajak_hibur
an),

      kategori_total_pajak
:
row.kategori_total_pajak,

    }));

    // Bin data
berdasarkan nilai input
sebagai threshold

    const binnedData =
cleanedData.map((row) =>
({

      pajak_reklame:
row.pajak_reklame <=
userInputValues.pajak_rekl
ame ? "<= Threshold" : ">
Threshold",

      pajak_parkir:
row.pajak_parkir <=
userInputValues.pajak_park
ir ? "<= Threshold" : ">
Threshold",

      pajak_hotel:
row.pajak_hotel <=
userInputValues.pajak_hote
l ? "<= Threshold" : ">
Threshold",

      pajak_hiburan:
row.pajak_hiburan <=
userInputValues.pajak_hibu
ran ? "<= Threshold" : ">
Threshold",

      kategori_total_pajak
:
row.kategori_total_pajak,

    }));

    const features =
["pajak_reklame",
"pajak_parkir",
"pajak_hotel",
"pajak_hiburan"];

    const className =
"kategori_total_pajak";

    // Hitung gain untuk
setiap atribut secara
manual

    const manualGains:
Record<string, number> =
{};

    features.forEach((feat
ure) => {

      manualGains[feature]
=
calculateGain(binnedData,
```

```
feature, className);

    });

    console.log("Manual
Calculated Gains:",
manualGains);

    // Analisis distribusi
data untuk setiap atribut

    const
distributionAnalysis:
Record<string,
Record<string,
Record<string, number>>> =
{};

    features.forEach((feat
ure) => {

      distributionAnalysis
[feature] = {};

      // Kelompokkan
berdasarkan nilai atribut

      binnedData.forEach((
item) => {

        const attrValue =
item[feature as keyof
typeof item];

        const classValue =
item.kategori_total_pajak;

        if
(!distributionAnalysis[fea
ture][attrValue]) {

          distributionAnal
ysis[feature][attrValue] =
{};

        }

        distributionAnalys
is[feature][attrValue][cla
ssValue] =

          (distributionAna
lysis[feature][attrValue][
classValue] || 0) + 1;

      });

    });

    console.log("Data
Distribution Analysis:",
JSON.stringify(distributio
nAnalysis, null, 2));

    // Tentukan urutan
tetap untuk atribut
(berdasarkan gain dari
tinggi ke rendah)

    const sortedFeatures =
[...features].sort((a, b)
=> manualGains[b] -
manualGains[a]);

    console.log("Sorted
features by gain:",
sortedFeatures);

    // Buat decision tree
secara manual dengan
urutan tetap

    const tree =
buildDecisionTree(binnedDa
ta, features, className,
0, 4, sortedFeatures);

    // Buat input untuk
prediksi

    const
userInputForPrediction = {

      pajak_reklame: "<=
Threshold", // Selalu
gunakan <= Threshold
karena dibandingkan dengan
nilai diri sendiri

      pajak_parkir: "<=
Threshold",

      pajak_hotel: "<=
Threshold",

      pajak_hiburan: "<=
Threshold",

    };
```

```
    // Lakukan prediksi
menggunakan pohon
keputusan manual

    const prediction =
predict(tree,
userInputForPrediction);

    // Log struktur pohon
dengan gain values

    console.log("Tree
Structure with Gains:");

    const logTreeStructure
= (node: TreeNode, indent
= 0) => {

      const prefix = "
".repeat(indent);

      if (node.attribute)
{

        console.log(`${pre
fix}Attribute:
${node.attribute}, Gain:
${node.gain?.toFixed(4)}`)
;

        if (node.branches)
{

          node.branches.fo
rEach((branch) => {

            console.log(`$
{prefix}  Branch:
${branch.value}`);

            logTreeStructu
re(branch.next, indent +
4);

          });

        }

      } else if
(node.result) {

        console.log(`${pre
fix}Result:
${node.result}`);

      }

    };

    logTreeStructure(tree)
 // Format nilai threshold
untuk tampilan

    const formatCurrency =
(value: number): string =>
{

      return new
Intl.NumberFormat("id-
ID").format(value);

    };

    // Temukan jalur
prediksi

    const
findPredictionPath =
(tree: TreeNode, input:
any, path: string[] = []):
string[] => {

      if (tree.result) {

        return path;

      }

      if (tree.attribute
&& tree.branches) {

        const attribute =
tree.attribute;

        const inputValue =
input[attribute];

        for (const branch
of tree.branches) {

          if (branch.value
=== inputValue) {

            const nodeName
=

              path.length
=== 0

                ? "root"
```

```
          :
`${path[path.length -
1]}_${branch.value === "<=
Threshold" ? "left" :
"right"}`;

            return
findPredictionPath(branch.
next, input, [...path,
nodeName]);

          }

        }

      }

      return path;

    };

    const predictionPath =
findPredictionPath(tree,
userInputForPrediction);

    const highlightedNodes
= new Set(predictionPath);

    // Buat diagram
Mermaid

    const treeToMermaid =
(tree: TreeNode, nodeId =
"root", depth = 0,
maxDepth = 3): string[] =>
{

      const lines:
string[] = [];

      const isHighlighted
=
highlightedNodes.has(nodeI
d);

      const highlightClass
= isHighlighted ?
":::highlighted" : "";

      if (tree.result) {

        // Node daun

        const countsStr =
Object.entries(tree.count
|| {})

          .map(([key,
value]) => `${key}:
${value}`)

          .join("<br/>");

        lines.push(`${node
Id}["Hasil:
${tree.result}<br/>${count
sStr}"]${highlightClass}`)
;

      } else if
(tree.attribute) {

        // Node keputusan

        const attrLabel =
tree.attribute;

        const threshold =
formatCurrency(userInputVa
lues[tree.attribute as
keyof typeof
userInputValues]);

        // Tampilkan gain

        const manualGain =

          manualGains[tree
.attribute] !== undefined
?
manualGains[tree.attribute
].toFixed(3) : "N/A";

        lines.push(`${node
Id}["${attrLabel} ≤
${threshold}<br/>Gain:
${manualGain}"]${highlight
Class}`);

        // Proses cabang
jika belum mencapai
kedalaman maksimum

        if (tree.branches
&& depth < maxDepth) {

          const leftBranch
= tree.branches.find((b)
=> b.value === "<=
Threshold");
```

```
          const
rightBranch =
tree.branches.find((b) =>
b.value === ">
Threshold");

          if (leftBranch)
{

            const leftId =
`${nodeId}_left`;

            lines.push(...
treeToMermaid(leftBranch.n
ext, leftId, depth + 1,
maxDepth));

            lines.push(`${
nodeId} -->|≤
${threshold}| ${leftId}`);

          }

          if (rightBranch)
{

            const rightId
= `${nodeId}_right`;

            lines.push(...
treeToMermaid(rightBranch.
next, rightId, depth + 1,
maxDepth));

            lines.push(`${
nodeId} -->|>
${threshold}|
${rightId}`);

          }

        }

      }

      return lines;

    };

    const mermaidLines = [

      "graph TD;",

      "classDef
highlighted
fill:#e6f7ff,stroke:#1890f
f,stroke-width:2px;",

      ...treeToMermaid(tre
e),

    ];

    const mermaidDiagram =
mermaidLines.join("\n");

    // Simpan hasil ke
database

    await
prisma.decisionResult.crea
te({

      data: {

        tahun:
input.tahun,

        pajak_hotel:
input.pajak_hotel,

        pajak_parkir:
input.pajak_parkir,

        pajak_hiburan:
input.pajak_hiburan,

        pajak_reklame:
input.pajak_reklame,

        mermaidDiagram,

      },

    });

    // Kembalikan hasil
dengan debugging info

    return {

      prediction,

      decisionTree: tree,

      mermaidDiagram,

      inputData: {

        ...input,
```

```
        // Tambahkan nilai yang sudah diformat
        pajak_hotel_formatted: formatCurrency(userInputValues.pajak_hotel),
        pajak_parkir_formatted: formatCurrency(userInputValues.pajak_parkir),
        pajak_hiburan_formatted: formatCurrency(userInputValues.pajak_hiburan),
        pajak_reklame_formatted: formatCurrency(userInputValues.pajak_reklame),
      },
      // Info tambahan untuk debugging
      gainAnalysis: {
        manualGains,
        dataDistribution: distributionAnalysis,
      },
    };
  } catch (error) {
    handlePrismaError(error);
  }
}),
});
```

Page-decision.tsx

```
"use client";

import { NavBreadCrumb } from "@/components/nav-breadcrumb";

import { Accordion, AccordionContent, AccordionItem, AccordionTrigger } from "@/components/ui/accordion";

import { Card, CardContent } from "@/components/ui/card";

import { CreateDecisionForm } from "@/features/decision/forms/create";

import { useTRPC } from "@/utils/trpc/client";

import { useMutation } from "@tanstack/react-query";

import { Loader2 } from "lucide-react";

import { useEffect, useRef } from "react";

import { toast } from "sonner";



export default function Page() {

  const trpc = useTRPC();

  const chartRef = useRef<HTMLDivElement>(null);

  const decision = useMutation(

    trpc.decision.createDecision.mutationOptions({

      onSuccess: () => {

        console.log("Decision tree created
```

```
  successfully");

    },

    onError: (error) => {

      toast.error(error.mess
  age);

    },

  })

);

const mermaidDiagram =
  decision.data?.mermaidDiag
  ram;



// Effect to render mermaid
  diagram when data changes

useEffect(() => {

  // Early return if
  conditions aren't met

  if (!mermaidDiagram ||
  !chartRef.current) {

    return;

  }

  const renderMermaid =
  async () => {

    try {

      const currentRef =
  chartRef.current;

      if (!currentRef) {

        return; // Double-
  check ref is still valid

      }



      // Dynamically import
  mermaid

    const mermaid = (await
import("mermaid")).default
;

    // Initialize mermaid

    mermaid.initialize({

      startOnLoad: false,

      theme: "default",

      securityLevel:
"sandbox",

      flowchart: {

        useMaxWidth:
false,

        htmlLabels: true,

      },

    });

    // Clear previous
content (safely)

    currentRef.innerHTML =
"";

    // Generate a unique
ID

    const id = `mermaid-
${Date.now()}`;



    // Render the diagram

    const { svg } = await
mermaid.render(id,
mermaidDiagram);



    // Verify ref is still
valid before updating

    if (chartRef.current)
{

      chartRef.current.inn
```

```
  erHTML = svg;

      }

    } catch (error) {

      console.error("Failed
  to render mermaid
  diagram:", error);

    }

  };

  renderMermaid();

}, [mermaidDiagram]);

return (

  <main className="space-y-
  8">

    <NavBreadCrumb
  current="dashboard/decisio
  n" />

    <section
  className="space-y-1 ">

      <h4 className="text-lg
  font-semibold leading-
  none">Decision Tree</h4>

      <p className="text-sm
  text-muted-
  foreground">View decision
  tree visualization</p>

    </section>

    <CreateDecisionForm
  decision={decision} />

    {mermaidDiagram ? (

      <Card>

        <CardContent
  className="pt-6">

          <div
  ref={chartRef}
  className="mermaid-
  container w-full mx-auto
overflow-auto"></div>

        <Accordion
type="single" collapsible>

          <AccordionItem
value="item-1">

            <AccordionTrig
ger>Show
Rule</AccordionTrigger>

            <AccordionCont
ent>

              <pre
className="mt-2 p-4 bg-
gray-100 rounded overflow-
auto text-sm">

                {mermaidDi
agram || "Tidak ada aturan
yang ditentukan"}

              </pre>

            </AccordionCon
tent>

          </AccordionItem>

        </Accordion>

      </CardContent>

    </Card>

  ) : (

    <Card className="flex
items-center justify-
center min-h-[200px] text-
center">

      <CardContent>

        {decision.isPendin
g ? (

          <div
className="flex items-
center justify-center gap-
2">

            Membuat
```

```
    decision tree <Loader2
    className="animate-spin"
    />

              </div>

            ) : (

              "Klik tombol
    Buat decision tree untuk
    menampilkan visualisasi"

            )}

          </CardContent>

        </Card>

      )}

    </main>

  );

}
```

Target.tsx

```
import { z } from "zod";

import { handlePrismaError }
    from
    "../db/handleErrorPrisma";

import prisma from
    "../db/prisma";

import { privateProcedure,
    publicProcedure, router }
    from "../trpc";



export const targetRouter =
    router({

  get:
    publicProcedure.query(asyn
    c () => {

    return await
    prisma.target.findMany({

      orderBy: {
        createdAt: "desc",

      },

    });

}),

create: publicProcedure

  .input(

    z.object({

      name: z.string(),

      pajak_hotel:
  z.string(),

      pajak_parkir:
  z.string(),

      pajak_hiburan:
  z.string(),

      pajak_reklame:
  z.string(),

    })

  )

  .mutation(async ({ input
  }) => {

    try {

      const target = await
  prisma.target.create({

        data: {

          ...input,

        },

      });

      return target;

    } catch (error) {

      handlePrismaError(erro
  r);
```

```
    }

  }),


update: privateProcedure

  .input(

    z.object({

      id: z.string(),

      name: z.string(),

      pajak_hotel:
  z.string(),

      pajak_parkir:
  z.string(),

      pajak_hiburan:
  z.string(),

      pajak_reklame:
  z.string(),

    })

  )

  .mutation(async ({ input
  }) => {

    try {

      const target = await
  prisma.target.update({

        where: {

          id: input.id,

        },

        data: {

          ...input,

        },

      });

      return target;

    } catch (error) {

      handlePrismaError(erro
  r);

    }

  }),

delete: publicProcedure

  .input(

    z.object({

      id: z.string(),

    })

  )

  .mutation(async ({ input
  }) => {

    try {

      const target = await
  prisma.target.delete({

        where: {

          id: input.id,

        },

      });

      return target;

    } catch (error) {

      handlePrismaError(erro
  r);

    }

  }),

});
```

Page-target.tsx

```
"use client";
```

```
import { NavBreadCrumb } from
    "@/components/nav-
    breadcrumb";

import { columns } from
    "@/features/target/table/c
    olumns";

import { DataTable } from
    "@/features/target/table/d
    ata-table";

import { useTRPC } from
    "@/utils/trpc/client";

import { useQuery } from
    "@tanstack/react-query";



export default function Page()
    {

  const trpc = useTRPC();

  const getTarget =
    useQuery(trpc.target.get.q
    ueryOptions());



  return (

    <main className="space-y-
    8">

      <NavBreadCrumb
    current="dashboard/target"
    />

      <section
    className="space-y-1 ">

        <h4 className="text-lg
    font-semibold leading-
    none">Target</h4>

        <p className="text-sm
    text-muted-
    foreground">View target
    pad</p>

      </section>

                    <DataTable
                        columns={columns}
                        data={getTarget?.data   as
                        []}
                        isLoading={getTarget.isLoa
                        ding} />

                        </main>

                    );

                }
```